Not Everything is a User Story

by Darren Petersen // March 21, 2018

At Lullabot, we've been using Agile techniques for years to help run our software projects. We've learned what pieces of Scrum and Kanban seem to fit our teams and what parts are safe to ignore. We've learned not just how different Agile methodologies prescribe we should work, but we've uncovered what's actually helpful to us. In other words, we have opinions.

One of those opinions really crystallized among our project managers over the last 12 months, and we want to share it with you. It has to do with the way we think and communicate about the work that our teams do.

User stories

Most of us in the software industry have encountered the idea of user stories. These minimalist requirement statements have been around for two decades, and are one of the fundamental ideas in Agile software development. As such, there's been a lot of thinking and discussion around them.

Rather than being some definitive statement on user stories, this article is a reflection of the experience of working with this kind of artifact within the specific context of our clients, developers, and project managers here at Lullabot.

A user story is supposed to be a short statement about a task a user wants to do with a particular software. It briefly describes the user, the task, and what benefit the user gets from it.

On typical format of a user story looks like this:

> *As a <type of user> I want to <do something> so that I can <benefit>.*

A more concrete example would be:

> *As a Night Owl, I want to drink multiple cups of coffee in the morning so that I can be even slightly functional before noon.*

Or:

> *As a visitor to this website, I want to easily search for content so that I can find what I'm looking for quickly.*

The brevity of this format is helpful to force people to have actual conversations about the feature they're building. Once upon a time, development teams and business stakeholders needed to be encouraged to collaborate more, so this was a revolutionary tool.

So, obviously, this is a good idea. But In the course of the last year, we talked about user stories a lot and how they can help or hurt a development project.

Is everything really a story?

Stories are such an operating assumption that many of the software packages that help organize development projects start with 'Story' as a default issue type. They're so normalized in modern software that we've even had clients direct us that "everything needs to be a story" in their projects.

Explicitly or implicitly, there's an idea out there that since we're Agile (whatever that really means), we have to do user stories for everything. I've found myself writing 'As a user' stories uncritically, out of habit, or because I was instructed to do so.

However, there's often business-driven or system-level requirements, which are not user-focused at all, that have to be crammed into the user story format. It's easy to apply the user story format to everything, for the sake of consistency, but it's a bad fit in several common cases. Here's an example of a technical requirement stuffed into a user story:

> *As the system, I want to verify a user's OAuth credentials before granting access so that I can ensure secure connections.*

I've seen and written lots of stories like that. The problem is that this story personifies the system with wants and desires which it does not have. If it were truly a user story, the user would be the focus like in this example:

> *As an amateur chef, I want to log into the system because I want to access recipes behind the paywall.*

You'll note that the whole character of the story has changed. Security and technology standards are not the primary concern of the user, so they aren't reflected. It doesn't reflect the business requirement of HOW the authentication should happen, but then again the user doesn't care about that. It's honest but less than effective as a requirement.

As a user, I want you to take my money

This is also a problem for business requirements that are not actually user-focused. They are similarly ill-suited to the user-story format:

> *As a site visitor, I want to see advertisements so I can know about products and services that might interest me.*

We know that no user ever wanted that. They came to the website for the content, and the advertising was a distraction. So that user story is fundamentally NOT about the site visitor—it's about the revenue model of the site.

User stories function as conversation starters about the value of a piece of work and the ways in which that value might be realized. Our examples above don't need conversation or discussion. The imperative and authority to do the work come from the organization's need to provide security or earn revenue.

So how do you express non-user requirements?

In general, it's better to surrender to common sense and not put these kinds of technical requirements into the user's voice. Instead, write simple, imperative statements that declare what must be done.

> *Integrate Google AdSense into article pages.*
> *Require a valid OAuth token for access to the system.*

We like to surrender to the forces of common sense and call a user story that no longer involves a user what it actually is: a task for a developer to perform.

This might seem like a meaningless distinction—who cares if it's a story or a task or whatever? But if your mental model of the user is demonstrably false, what else are you getting wrong?

It's easy to write your biases into the user's voice and finding yourself keeping the status quo instead of doing something new. If you insist on shaping the conversation around your product from a false premise, how can you spot your real business problems and innovate to solve them?

To take our example from above, maybe traditional web advertising is a sub-standard way of generating revenue for your business, but you'll never have that conversation if you paint it over with false user requirements and benefit statements.

What are user stories actually good at?

The beginning of our process is a well-written, truthful story about a feature with benefits for specific kinds of users, accompanied by clear acceptance criteria.

Ideally, we're starting from a position where there's been actual user research and one-on-one interviews during the discovery process. That ensures we're building features that users are actually interested in.

> *As a content administrator, I want to be notified of new user account requests because I need to review and approve them quickly.*
> *As an authenticated user, I want to drill into search results using facets because I'm looking for something very specific.*
> *As a vacation planner, I want to visit a page that aggregates content about my country of interest, to help me decide what I might like to do while in that country.*

The conversation around actual user-based stories builds understanding between the business and our development team, and it sets us on the right path. It just needs to be recognized that the user story format itself isn't magic.

The life-cycle of a story

It's natural that stories would be initiated by the business stakeholders based on their knowledge of their users. In the same way, when the developers are done with their work, those same stakeholders will want to verify that the story is properly complete.

That makes stories an ideal artifact to drive QA and acceptance testing. Developers and QA team members both benefit from business-level acceptance criteria elaborating the story to help guide their work.

We like to use Gherkin (https://github.com/cucumber/cucumber/wiki/Gherkin) as a way to write acceptance criteria using a particular format consisting of 'Given, When, Then' statements.

- **Given** expresses the preconditions of the acceptance criteria. This might be authentication, the existence of some data, or the completion of a business step that must precede the feature under discussion.
- **When** expresses the action a user takes.
- **Then** expresses the result of the action.

All three of these keywords can be combined with conjunctions like 'and' or 'but' to layer conditions onto the acceptance criteria. Here's a simple example:

> *Given a user has requested an account in the system,*
> *and the account request has been reviewed by an admin,*
> *and the admin wishes to approve the request.*
> *When the admin approves the request,*
> *then the account changes state from 'pending' to 'active,'*
> *and the user is notified by email that their account request has been approved.*

This kind of acceptance criteria is great to work up as the story is being discussed, where developers can ask questions of business stakeholders. Gherkin is actually used in software like Cucumber (https://cucumber.io/) and Behat (http://behat.org/en/latest/) to drive automation of these test criteria.

Not every client is interested in this kind of test automation, but even if those tests will never end up in code, working with Gherkin to write acceptance criteria has a way of clarifying everyone's thinking.

When the conversation is done

The story and the acceptance criteria are really valuable for everyone involved. But, when you've reached a stopping point and the story is 'ready for development,' there's still more that has to happen. Even with great acceptance criteria, the story still might not express the level of detail that a developer needs.

Beyond the story and acceptance criteria, developers may rely on technical documentation, design artifacts, or architectural planning to fill in the gaps. This is especially true when a story is a single unit of value to the business, but the implementation crosses disciplines and teams.

Stories for everyone, subtasks for developers

It's helpful at that point to break the story into subtasks which can be assigned and sized for an optimal developer workflow — bite-sized chunks that take no more than 1-2 days to accomplish, and which are often unit testable but aren't always good candidates for QA.

Our account approval example from above can be broken into several tasks. Most of these features are off-the-shelf with Drupal, but we'd probably want to make some adjustments and customize the language in the notification emails.

Again, our task format is short imperative statements, possibly with additional notes as needed in the body of a ticket.

> *Grant permission to site admins to allow account approvals.*
> *Build a list of open account requests.*
> *Customize account approval emails.*

You may or may not need distinct acceptance criteria here. Some tasks are self-explanatory, while others have the need for more guidance as we describe above.

In any case, QAing these shorter tasks can be cumbersome and confusing for the QA team because the total work for that story is not sufficiently complete. Any single task is only part of the full acceptance criteria, and they're often interdependent.

However, we're not without validation for those subtasks. Unit testing and peer review between developers work very well for checking them along the way, especially when we write custom code where errors might be introduced. When all the tasks are done, the complete story can move forward to QA and acceptance testing for the complete feature.

All of this is to say that user stories are an important part of Lullabot's software development process—but they're not the only thing we need to get the job done.

Get more background

I like to know the background and history of these techniques we use to manage our projects. The user stories and Agile development

practices have been around for about 25 years, but it sits in the context of the whole history of software. Knowing the history and the thinkers behind it all set you up to use the practices selectively and use them well.

In that light, and to give credit where credit is due, these are some of the folks that did the big thinking around user stories and other Agile practices. If you're a practitioner of Agile in some way, check out their history and opinions:

- Kent Beck (https://en.wikipedia.org/wiki/Kent_Beck)
- Alastair Cockburn (https://en.wikipedia.org/wiki/Alistair_Cockburn)
- Mike Cohn (https://en.wikipedia.org/wiki/Mike_Cohn)
- Martin Fowler (https://en.wikipedia.org/wiki/Martin_Fowler)
- Ron Jeffries (https://en.wikipedia.org/wiki/Ron_Jeffries)

And, here a couple of books that have been helpful:

- User Stories Applied, by Mike Cohn (https://www.amazon.com/User-Stories-Applied-Software-Development/dp/0321205685)
- User Story Mapping, by Jeff Patton (https://www.amazon.com/User-Story-Mapping-Discover-Product/dp/1491904909/ref=sr_1_1?ie=UTF8&qid=1520606721&sr=8-1&keywords=story+mapping+jeff+patton)

Darren Petersen