User Story Feedback: Explanations of Basic Criteria

Role is Specific, Not General: it's tempting to use very general user roles in stories, but it is better if we can give a role that more accurately describes the kind of person who receives the value described in the story. So, generic roles such as "customer" and "manager" are to be avoided (while the role "user" is right out). Instead, try roles such as the following:

First year undergraduate student
Struggling masters student
Course team leader
First time customer
Repeat visitor to the site
Customer on a tight budget
Short-for-time customer

Remember that the role given in the story should describe the person who receives the value from the functionality, and is not necessarily the person who directly uses the functionality when implemented.

Functionality is an end-to-end slice: user stories should describe thin slices of functionality that can independently deliver business value when implemented. We want to be able to choose any story to implement, and (at the end of the iteration) to be able to deliver something that genuinely solves part of the customer's problem, albeit in a small way.

So, when assessing a story against this criterion, ask yourselves whether, if a system was shipped that delivered only the functionality described in the story, the user role named in the story would be able to get any of the stated value, or not. If a user signed on to the system, did just the thing described by the functionality, and immediately signed off again, would anything useful have been achieved?

User Story Feedback: Explanations of Basic Criteria

Goal Describes a Real Business Value: writing the business value part of a user story can be really difficult, especially for the first few projects in a new area. One common mistake that is made is to write a business value that just restates what the functionality part of the user story says, like this one:

As a registered fan, I want to get early access to concert tickets, so that I can buy tickets before they are made available to the public.

Here, the business value is just restating the same idea as the functionality. It doesn't add anything new to the user story, so it isn't really describing why the functionality is wanted. A better business value would be "so that I have a better chance of being able to attend popular concerts". Or, maybe, the real beneficiary of this story is the sales manager, who wants fans to get early access to concert tickets so that the time-limited opportunity encourages them to purchase more tickets earlier.

A good technique for checking whether you have actually written a value or not is to look for the word that shows a change in something. Our fan wanted to "have a **better** chance" of getting tickets. The sales manager wants **more** tickets purchase **earlier**. If the business value doesn't contain a word like this, then what is the impact (behaviour change) it is aiming for?

Customer appropriate language used: user stories are one of the key tools we have for fostering a genuine collaboration between customers and developers. It's vital that customers can read and understand the user story set we build up as the project progresses, so that they can give meaningful and useful feedback as early as possible. That is one of the reasons why user stories just take the form of ordinary sentences. There is no special notation to learn, no arcane keywords or diagrams. Just ordinary English/French/Bulgarian/Urdu/... sentences.

We won't get the benefit from this, however, if we pack our user story sentences with words that the customer is not familiar with. In particular, technical words (such as "database", "SSL", "array", "drop down menu") should be avoided. Instead, the words that the customer uses to talk about the domain should be used, even if they are unfamiliar to the developers. It's the developers' job to learn the customer's preferred vocabulary, and not the other way around.

User Story Feedback: Explanations of Advanced Criteria

Story is independent: one of the advantages of using end-to-end slices as the basis of our user stories is that they should deliver value in themselves, independently of any other story. In practice, we can't always achieve complete independence, but we would like at least to have stories that are technically independent of each other, even if there is some business value dependence. For example, it is technically possible to implement the stories for browsing upcoming events and for selling tickets to upcoming events in any order. But, we might not see any real business value from being able to sell tickets when the attendees can't find out what events tickets are available for.

We sometimes have to live with this business value dependence, but technical dependence between stories usually means that the functionality is not expressed as a true end-to-end slice. So, check that the stories can be implemented in any order, to make sure they are all technically independent.

Contains only useful ambiguity: you will probably have been taught on other courses that software requirements should be written in as unambiguous a way as possible. This is certainly important when you are relying on written specifications, where the people trying to write code from the specification may never actually meet the people who write it, or the customer whose needs dictate it.

The problem is that it is almost impossible to write anything more than a trivial few sentences in natural language without ambiguity. Luckily, user stories do not need to be unambiguous. In fact, some degree of ambiguity is a good thing, since it reminds the developers that they need to speak to the customer before doing any implementation. But too much ambiguity is a problem. If the developer has no idea who to talk to about a story, or even what it is broadly about, then there is too much ambiguity and the story should be clarified.

User Story Feedback: Explanations of Advanced Criteria

Story is estimatable: one of the things we will do next with our user stories is group them into meaningful releases and try to work out how many iterations it would take to produce each release. If a story doesn't contain enough detail to allow us to at least say whether it is a bigger or smaller story than the others we have, then it is not estimatable, and some key details need to be added to the story sentence. For example, is the following story large or small?

As a course team leader, I want the coursework for my modules to be marked automatically, so that we can use **more** of our teaching time for giving formative feedback to students

This could be a small task, if all the coursework is multiple choice, or a huge, impossible task, if it is an essay. These key details need to be added so the story's size can be estimated.

Story size: for this criteria, you can just note if the story seems small enough to be implemented within one or two weeks (by a team of 4 developers) or if it would take many iterations to complete (in which case it is an epic). Both sizes of story are okay.

Story is testable: one of the problems that plagues software development projects is the lack of an objective measure of when a piece of functionality is done. All too often, the developers on the team think the work is done, but the customer disagrees. What happens then?

On an agile team, we try to avoid this by making sure that all our stories have clear "acceptance criteria". We'll look at how to do that in a future lecture, but for now we should make sure our user stories are written in such a way that we can see what concrete actions we could take to see if they are done: that is, that they are testable. The problem arises when certain forms of ambiguity enter the story. Does the story contain enough information that we can guess at a set of concrete actions we could take to determine whether the story has been implemented? If not, it is not testable and needs to be clarified.