**[Software Testing - User Stories](http://www.testinggeek.com/software-testing-user-stories)**

By **Anand Ramdeo** on **May 5, 2011**[1 Comment](http://www.testinggeek.com/software-testing-user-stories#disqus_thread)

User story is one of the primary development artifact for the XP project teams. A user story contains just enough information so that development team can reasonably give estimate about completing, tester can discuss how it will validated and customer can see its value.

One of the common question that we hear most of the time is, how user stories are different from use cases. User stories are much simpler than use cases. User stories are very easy to create, discuss and develop. They also do not contain any technical details.

Typically good user stories are defined in the following format

*As a I would like to do so that*

While discussing User Stories, you should make sure that they are not very big. A big user story leaves the chances of ambiguity and absence of clarity in the mind of development team. Ideally, you should be able to break user stories so that it can be finished by two person in a week's time.

If you are working as a tester in a agile project and see any story taking longer to finish, consider this as an opportunity to ask question and see if this can be broken by functionality or platform or in any other way.

Once a story is broken to the last level, this story is assigned some points based on how long it will take a pair of programmer to finish it. These user stories are than broken down in tasks and assigned to appropriate developers. Typically in the start of project, you create a deck of user stories and for each iteration or sprint you choose stories based on their priority and time it would take to complete them.

User stories are often written on the paper medium like index cards or post-its. This also ensures that there is no unnecessary details on the user stories. Once development team decides on the stories that will be developed in a sprint, it is further divided into tasks and resources are allocated accordingly.

Most of the time, these user stories are defined only in acceptance term. For example,

As a user I should be able to register with the site so that I can receive newsletter is a good example of user story. As a tester, it should be your job to find out negative scenarios and assumptions in this story. For example, what information user should provide to register? Are there any distinction between optional and required information? Should we validate email address of user before registering?.. and many more.

Remember, one of the main reason for using user stories is to have an invitation to further conversation. According to Ron Jeffries, every user story in XP have three components - Cards (Physical Medium) , Conversation (Discussion Surrounding them) and Confirmation ( tests that verify them). One of the widely used acronym for good user stories is called INVEST. Getting right user stories is essential for the success of any agile project and hence this acronym INVEST is used to justify the investment that you make in creating user stories.

According to this acronym,

* I stands for independent
* N for negotiable
* V for valuable
* E for Estimable
* S for small and
* T for testable

As a tester in agile projects, it is very important for you to make sure that every story you work on conforms to the characteristics defined by these terms. There are good reasons for doing this.

Stories should be independent so that it is easier to plan them. Consider the situation where in you need to decide stories to be included in the Sprint and most of the stories are dependent on each other.

Stories should be negotiable, user stories are not contract with the customer they are a platform to clear understanding between everyone involved. Any one should be able to challenge its intention as well as implementation.

User stories should be valuable to some one. You will have situation where in your stories will not bring direct value to your customer, but it should bring value to some one, it could be developer or tester. For example As a tester I should be able to access application below UI layer so that Automation is not dependent on the UI. As oppose to We should have functionality of accessing application with out UI.

Stories should be estimable, remember we are not saying accurate estimate, but it should be reasonable estimate. Since stories are negotiable, these estimates can change but initially it should be possible to estimate reasonable on how much time it would take to complete a story.

Most of the characteristics defined above make sure that user stories are clear and small. This is very important, most of time big stories are also vague. It is also possible to keep team moving with smaller stories as it gives a sense of accomplishing or finishing something. Typically, you should try to complete at least 2 stories per sprint (two programmers and one tester for two week).

Since user stories are requirements which drives the development you should make sure that these stories are testable. You can find this by looking for some characteristics like ambiguity, clarity etc which you might have used in verifying requirement in traditional development model.

Remember, user stories are the crux of agile software development methodology and as a tester you should make sure that your team INVEST in user stories.

**New to User Stories?**

http://www.scrumalliance.org/articles/169-new-to-user-stories

April 19, 2010 in Practices

Having coached traditional requirements, use cases, user stories, and agile development, I’ve fielded a lot of questions around the differences among the three major ways of specifying requirements, particularly by people migrating to user stories. To set the record straight on requirements, use cases, and user stories, I will describe each methodology and then compare the three against each other.

**Traditional requirements**

Traditional requirements are criteria to which the system or business must adhere. They are usually created before the coding begins and are nearly always written as text. They often are thought of as constraints, conditions, or capabilities to which the system must conform.   
  
Good requirements have the following characteristics:

* Complete. Requirements should be as complete as possible—no open-ended requirements.
* Testable. Must be able to create a test for all requirements.
* Consistent. Requirements must be consistent with each other—no conflicts.
* Design Free. Software requirements should be specified in the business perspective rather than the software perspective.
* Unambiguous. Use "shall" and other related words. Don’t be wishy-washy.

Traditional requirements focus on system operation. They typically are written in such a way as to limit interpretation and rarely contain explicit tests or acceptance criteria. These types of requirements are often written atomically; meaning that thousands of independent shall statements can comprise a software requirements specification. Traditional requirements are normally treated as a contract between the business and systems development and, as such, completeness, formality and rigidity in requirements is the rule.

**Use Cases**

A use case is a series of interactions by the user (Actor) with the system and the response of the system. Use cases consist of two main components: use case diagrams (which graphically describe actors, use cases, system boundaries, and the relationship between all of these) and the text of the use case itself. Use cases and use case diagrams focus on the user, with a use case text itself written in a call-and-response format that shows an action by the user, followed by the system’s response.   
  
Use cases focus on interactions and are written in such a way as to succinctly define the user/system activities and data that define the interaction. Use cases can be written atomically as well, but the use case diagram is meant to tie together the use cases. Use cases are intended to be drilled down in successive levels of detail, reducing the need for nailing down the details before coding.

**User Stories**

User stories are actually narrative texts that describe an interaction of the user and the system, focusing on the value a user gains from the system. A true user story is a metaphor for the work being done. It is not a highly documented requirement but rather a reminder to collaborate about the topic of the user story—in other words in agile development (good agile at least), the documentation is secondary to the collaboration. User stories aren’t agile in and of themselves. Instead, their underlying agile values—collaboration and just-in-time definition—make user stories a good agile tool. Formality is specifically removed from the mix and specification of the user story is pushed as late as possible.   
  
A good user story uses the “INVEST” model:

* Independent. Reduced dependencies = easier to plan
* Negotiable. Details added via collaboration
* Valuable. Provides value to the customer
* Estimable. Too big or too vague = not estimable
* Small. Can be done in less than a week by the team
* Testable. Good acceptance criteria

**User story–writing process**

The story-writing process is integral to understanding user stories.   
  
The typical template has 3 parts: the title, the description (or body of the user story), and the acceptance criteria. The title is used to reference the user story and should be kept very short, around 3 to 10 words. The description is where the meat of the user story is kept. It is the only part that can be explained as a reasonable template. The acceptance criteria are used to determine when the user story has met the goal of the user – a sort of test.  
  
Start by writing the title. It should be long enough to allow people on the team to differentiate it from other stories but short enough to fit on a 3” x 5” sticky card when written with a marker.   
  
Now write the description. You can use the following template:   
As a [user role] I want to [goal] so I can [reason].   
  
If the description becomes lengthy (more than will fit on an index card), you should revisit the user story. It is likely it needs to be split into several stories. You might also consider whether you are trying to include too much detail. Remember that the purpose of a user story is to encourage collaboration. A user story is a promise to have a future conversation; it is not meant to document every aspect of the work, as you might in a series of traditional requirements statements.   
  
When writing a user story, you might include some acceptance criteria, perhaps in the form of a test case or a brief description of "done," if those criteria help make the intent of the user story easier to remember. When the team is ready to work on that story, however, the team and the product owner must discuss the user story. This process will include (indeed must include) adding acceptance criteria so the team will know what done means. Later, as the team members begin to work on the story, they might contact the product owner and discuss new/different acceptance criteria as their understanding of the story grows – acceptance criteria enrich the understanding of the story, which in turn brings out new acceptance criteria and more meaningful conversations about what the customer really wants.

**Frequent Mistakes in User Stories**

There are three main problems I see in stories. First is too much information in the description. This leads to a loss of collaboration and a reliance on the old ways of documenting everything. A user story should be thought of as a "talking points", a "to-do list," or a "tickler that a conversation must occur about a topic." The user story is a placeholder for a conversation or series of conversations: it is only through collaboration that a user story works as an agile tool; otherwise it's just a requirement written on an index card.  
  
Too much information in a description can lead to the second problem: missing information in acceptance criteria. Before agreeing to work on a story, the team must understand the acceptance criteria. These are essential for knowing what needs to be done in order to satisfy the user story. Acceptance criteria should be detailed enough to define when the user story is satisfied, yet not so detailed as to quash collaboration. Writing acceptance criteria should not an afterthought – it is a crucial part of a user story.  
  
The third problem is to confuse acceptance criteria and test cases. Both are necessary for a user story. Acceptance criteria answer the question, “How will I know when I’m done with the story?” Test cases answer the questions, “How do I test and what are the test steps?” While both acceptance tests and test cases should be added to the user story via collaboration, only acceptance criteria are required. Test-driven development is often used to flesh out the test cases as the code is written.

**Comparing User Stories, Use Cases, and Requirements**

The basic difference between user stories and other forms of requirements specification has to do with perspective and intent, both of which affect the level of detail.  
  
Traditional requirements’ perspective is on system operations. They are typically written with the intent of limiting interpretation. They rarely have tests or acceptance as part of the mix – that’s another document – and are often written by someone other than the requirements author. The level of detail can vary, particularly if those writing the requirements are using a hierarchical approach to drill down to successive levels of detail. Traditional requirement focus on what the system should do, not on the user or business workflow.  
  
Use cases have the perspective of the users and their interaction with the system in mind. The use case diagram and associated text are intended to show the capabilities of the user and how these capabilities are met via a system response. Work flows or business flows can be derived from use case diagrams; the text of the use cases shows system and user interaction in a call-and-response format. Use case text typically contains more details than stories and traditional requirements. Test cases can often be written from the use case text, but are not typically part of the use case diagram per se.   
  
Both traditional requirements and use cases are meant to be successively refined into greater detail through detailed analysis and design into. They are both written as the primary communication media for the system capabilities.   
  
User stories focus on customer value. They don’t just assume a looseness of specification; they actually encourage it in order to foster a higher level of collaboration between the stakeholders and the team. A user story is a metaphor for the work being done, not a full description of the work. The actual work being done is fleshed out via collaboration revolving around the user story as system development progresses. As such the level of detail of a user story is ultimately higher than a use case but lower than a traditional requirement. In order to limit scope, user stories have collaboratively developed acceptance criteria which define when the user story meets the stakeholder’s expectations. Test cases are often developed as code (with test driven development) or documented as the code is developed.

**Comparison example**

**Problem Statement**  
The client has requested the ability to search for health care providers by provider specialty within a doctor selection site.   
  
Sample Requirements Statement:  
The provider search screen shall provide the ability to search for providers by provider specialty.  
  
**Sample Use Case**  
The client selects provider search.  
The system retrieves a predefined list of provider specialties and populates the specialty list. The system displays the provider search mechanism.   
  
The client selects a provider specialty and initiates the search.   
The system retrieves a list of providers that match the provider specialty search. [Alt 1]  
The system displays a list of providers that match the search. [Alt 2]  
End use case  
  
Alt 1: If there are no matches, the system displays a message indicating that no matches were found. End use case.  
  
Alt 2: If there are more matches than the user can view, the system will provide the capability to display multiple pages.  
  
**Sample User story**  
Title:  
Search for providers by provider specialty.  
  
Description:  
As a provider search user, I need the ability to search for providers by specialty so that I can more efficiently refer patients to specialists.  
  
Acceptance criteria:  
The provider search mechanism has the ability to enter a specialty.  
The specialty search will have a list of provider specialties from which to select.  
Searching via the provider specialty will return a list of matching specialists or a message indicating that there are no matches.  
If there are more results than can fit on one page, the system will provide the capability to view the list in pages or sections.

**Conclusion**

Are user stories better than other types of requirements specification? It depends. In the end, it’s the individual or team that makes a particular technique work (or fail). If a team is used to an iterative approach, use cases and user stories will be relatively easy to work with. If a team is steeped in waterfall and “big requirements up front (BRUF),” that team likely will bring that style to user stories and end up with traditional requirements that are user stories in name only. Adopting the underlying philosophy (concepts like collaboration and just-in-time requirements) is the key to moving toward more agile specification devices, such as user stories.  
  
User stories are not a highly documented series of requirements but rather a reminder to collaborate about the topic of the user story—in other words, in agile development (good agile at least), the documentation is secondary to the collaboration. The intent of user stories is to foster collaboration; the perspective is on customer value. If your user stories look more like requirements in disguise, you need to worry less about what specification you are using and more about how to increase collaboration. If, on the other hand, you have a collaborative environment already, user stories will enhance the collaboration more than use cases and traditional requirements can, in which case user stories are a better tool for you.