



A TECHWELL COMMUNITY

menu

Defining Requirement Types: Traditional vs. Use Cases vs. User Stories [article]

By [Charles Suscheck \(/users/charles-suscheck/\)](/users/charles-suscheck/) - January 18, 2012

Summary: If you have recently transitioned to an agile team, you may have questions about the differences between user stories and use cases, especially how they differ from traditional requirements writing. In this article, Charles Suscheck defines each of these requirements types and uses a running example to illustrate how they differ in a real-world setting.

I've worked with a lot of teams transitioning to agile. In each situation, user stories always seem to be a sticking point, with a common question being, "What are the differences between traditional requirements, use cases, and user stories?" I'd like to answer this question with a description and example of each requirement type. I'll also use a running example: Imagine that we're writing software for placement firms, and one of the firms has requested the ability to search for candidates for a specific role by specialty within a geographic location. For example, "I want to find all business analysts who are Sarbanes Oxley (SOX) experts within fifty miles of New York City."

Traditional Requirements

Traditional requirements are usually thought of as capabilities and constraints of the system; the key term being *system*. All good requirements describe what the system can do or shouldn't do, but those requirements that focus intensely on the system tend to deemphasize user interaction or business context related to the user or business. To be fair, many traditional requirements do provide context for business and users, but that is usually not the main focus of the requirement, rather it's the system that is the focus.



The difficulty with having the system be the focus is that it's easy to make assumptions about what the user wants. I've seen requirements be the source of record for the system's operation. In such a case, interacting with the business or the user while the system is being developed reverted to a series of painful, negotiated change request. The work became a matter of giving the user exactly what she asked for, which may not at all be what she needed.

This is what really makes traditional requirements tough: They're written from the system perspective. Additionally, they're often written in the context of a process that enforces change control and a contract based on the requirements themselves. Throw in an ideology that encourages *written* communication between the business analyst and the development team, and you've got a tough job ahead when it comes to delivering value. Changes become a series of tightly controlled negotiations.

According to the International Institute of Business Analysts (IIBA), good requirements can be described via these criteria:

- Requirements are complete. They must be as complete as possible with no open-ended parts or opportunity for interpretation.
- Requirements are testable. One must be able to create a test or some sort of proof that the requirement has been met.
- Requirements must be consistent with each other with no conflicts between what they are specifying.
- Requirements must be design-free. Software requirements should be specified in what the system must or must not do, but not in how the software will ensure the requirement is met; that's design.

[Return to article details](#)

We use cookies to ensure you get the best experience on our website.

[Read More](#)
[Accept](#)

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250> The DB_FUL_SPEC_CAN database.

The geographic regions must be specified via city, state, and optionally a radius in miles from the city.

Use Cases

Use cases attempt to bridge the problem of requirements not being tied to user interaction. A use case is written as a series of interactions between the user and the system, similar to a call and response where the focus is on how the user will use the system. In many ways, use cases are better than a traditional requirement because they emphasize user-oriented context. The value of the use case to the user can be divined, and tests based on the system response can be figured out based on the interactions. Use cases usually have two main components: Use case diagrams, which graphically describe actors and their use cases, and the text of the use case itself.

Use cases are sometimes used in heavyweight, control-oriented processes much like traditional requirements. The system is specified to a high level of completion via the use cases and then locked down with change control on the assumption that the use cases capture everything.

Both use cases and traditional requirements *can* be used in agile software development, but they may encourage leaning heavily on documented specification of the system rather than collaboration. I have seen some clever people who could put use cases to work in agile situations. Since there is no built-in focus on collaboration, it can be tempting to delve into a detailed specification where the use case becomes the source of record rather than a mechanism for conversations.

Use Case Example

The placement specialist selects a candidate search.

The system retrieves a list of candidate specialties and populates the candidacy list.

The system retrieves a predefined list of candidate specialties and populates the specialty list.

The system retrieves a predefined list of geographic regions.

The system displays the candidate search screen.

The placement specialist selects a candidate specialty.

The placement specialist selects a candidate role.

The placement specialist selects a geographic region.

The system identifies the region and populates the subregion.

The placement specialist selects the geographic subregion.

The placement specialist selects the search initiation button or mechanism.

The system retrieves a list of candidates who match the candidate specialty search. [Alt 1]

The system displays a list of candidates who match the search. [Alt 2]

End use case

<http://www.addthis.com/bookmark.php?v=250> ng a good <http://www.addthis.com/bookmark.php?v=250> article/de
ion of the s <http://www.addthis.com/bookmark.php?v=250> fining-
to be analyzed as the <http://www.addthis.com/bookmark.php?v=250> require
nents are particularly <http://www.addthis.com/bookmark.php?v=250> ment
types-
tradition
al-vs-
use-
cases-
vs-user-
stories)

inge in the user
ble, even small

ographic region.



<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

an exercise in writing out an inordinate amount of detail. This is counter to the [Agile Manifesto's](http://www.agilemanifesto.org/) (<http://www.agilemanifesto.org/>) principle that “the most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”

To sum up the differences: Traditional requirements focus on system operations with a tendency toward detailed system specification; use cases focus on interactions between the user and the system with a similar tendency of detailed specification; and user stories focus on customer value with a built-in imprecision meant to encourage communication.

User Story Example

Title:

Search for candidates by role, specialty, and geographic location.

Description:

As a candidate search user, I need the ability to search for candidates by specialty so that I can more efficiently refer patients to specialists.

Acceptance Criteria:

The candidate search mechanism has the ability to enter a role.

The candidate search mechanism has the ability to enter a specialty.

The specialty search will have a list of candidate specialties from which to select.

Searching via the candidate 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 on the situation, but in a collaborative environment, my experience leads me to say yes.

User stories will not make your project agile, and the lack of them will make it challenging to become agile. However, user stories encourage a number of principles from the [Agile Manifesto](http://www.agilemanifesto.org/): (<http://www.agilemanifesto.org/>)

- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Business people and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.

If you have a decidedly non-agile environment, will user stories help? Again, it depends on the team. If a team is steeped in waterfall or heavy iterative processes and uses the “big requirements up front” approach, that team likely will influence the user stories, turning them into traditional requirements. User stories are somewhat vague and lend themselves to successive

www.addthis.com/bookmark.php?v=250

[/http://www.addthis.com/bookmark.php?v=2501](http://www.addthis.com/bookmark.php?v=2501)

[/http://www.addthis.com/bookmark.php?v=2501](http://www.addthis.com/bookmark.php?v=2501)

[/http://www.addthis.com/bookmark.php?v=250](http://www.addthis.com/bookmark.php?v=250)

<http://www.addthis.com/bookmark.php?v=250>

(system requirements). The example given is a system requirement. I would expect to see two or possibly more levels of problem statement before you get to the requirement here. In fact, the requirements I would want to see would look not dissimilar to the user story example.


The approach is, to use a saying from winemaking, simple, but not easy. Define your overall goal, work out what must be achieved to reach the goal (these are capabilities and might need several levels, two is common), then work out what the system must do to provide the capabilities (these are functions, and again might need several levels). At each point, do not lose sight of non-functional requirements such as performance, security, etc.

User stories and use cases are great ways of identifying these capabilities, of course.

January 20, 2012 - 6:51am



Anonymous

 It is a fallacy to state that traditional requirements are focused on the system. There is a clear divide between requirements that state the problem (stakeholder requirements) and those that define (but not design) the system (system requirements). The example given is a system requirement. I would expect to see two or possibly more levels of problem statement before you get to the requirement here. In fact, the requirements I would want to see would look not dissimilar to the user story example.


The approach is, to use a saying from winemaking, simple, but not easy. Define your overall goal, work out what must be achieved to reach the goal (these are capabilities and might need several levels, two is common), then work out what the system must do to provide the capabilities (these are functions, and again might need several levels). At each point, do not lose sight of non-functional requirements such as performance, security, etc.

User stories and use cases are great ways of identifying these capabilities, of course.

January 20, 2012 - 6:51am



Anonymous

 It is a fallacy to state that traditional requirements are focused on the system. There is a clear divide between requirements that state the problem (stakeholder requirements) and those that define (but not design) the system (system requirements). The example given is a system requirement. I would expect to see two or possibly more levels of problem statement before you get to the requirement here. In fact, the requirements I would want to see would look not dissimilar to the user story example.

The approach is, to use a saying from winemaking, simple, but not easy. Define your overall goal, work out what must be achieved to reach the goal (these are capabilities and might need several levels, two is common), then work out what the system must do to provide the capabilities (these are functions, and again might need several levels). At each point, do not lose sight of non-functional requirements such as performance, security, etc.

User stories and use cases are great ways of identifying these capabilities, of course.

January 20, 2012 - 6:51am



Anonymous

It is a fallacy to state that traditional requirements are focused on the system. There is a clear divide between (the system) and (the design) the system

We use cookies to ensure you get the best experience on our website.

[Read More](#)

Accept

5/9

<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

<http://www.addthis.com/bookmark.php?v=2501>

only need to change relatively often and rapidly, the lack of documented requirements enables the rapid unnecessary changing of requirements that can lead to unnecessary requirements churn and floundering. Volatile requirements, when change is not really necessary, has led to many cost and schedule overruns.

4) User stories can be an excuse for the customer, stakeholders, and SMEs to be sloppy in considering what they want/need. Although it is true that many people don't really know what they want until they can see it, finding out that you don't like the system until it is delivered (or at least a prototype is available) is too late to decide that it's not what you wanted after all. That is especially true for large, complex, expensive systems including systems that consist of more than just software (e.g., hardware, equipment, facilities)

So although user stories have their place and value and should be part of the requirements engineer's toolbox, they have their limitations and disadvantages as well as their benefits.

My problem is when groups of people pick single techniques as their only way of solving non-trivial problems (like engineering requirements). That's when using things like textual requirements, use cases, and (yes even) user stories become a matter of dogma and ideological purity. A good carpenter has many tools in their tool box providing a choice of multiple ways of doing something (it is better to have both flathead and Phillips screwdrivers as well as hammers/nails and wrenches/bolts). Software developers should be no different; selecting and using the right tool for the job at hand given the job and its constraints.

January 25, 2012 - 11:45am



Anonymous

The author notes that good requirements should have the following two characteristics:

- 1) "Requirements are complete. They must be as complete as possible with no open-ended parts or opportunity for interpretation."
- 2) "Requirements must be unambiguous. No wishy-washy statements nor (conceptually) anything that can be interpreted differently than intended."

However, the author also states that:

- 1) "user stories focus on customer value with a built-in imprecision meant to encourage communication."
- 2) "User stories (when used as intended by experts like Cohn and Cockburn) are just too loosely formatted to lend themselves to full documentation."

Clearly, user stories are intentionally not complete as possible and are ambiguous. Thus, user stories are *not* requirements but rather a verbal person-to-person communication technique for eliciting verbal requirements that are not documented, but rather rapidly implemented in code and test cases.

This leads to several limitations of user stories:

- 1) They require close person-to-person collaboration between developers and stakeholders/SMEs/customers. This is often not possible in practice, especially when dealing with large, expensive systems that are not developed in house.
- 2) A lack of documented requirements makes it very difficult to cost and schedule a large and complex system, especially when customers and developers belong to different companies so that financial contracts are an unavoidable fact of life.
- 3) While being agile is great when requirements truly need to change relatively often and rapidly, the lack of documented



<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

<http://www.addthis.com/bookmark.php?v=250>

January 25, 2012 - 11:45am



Anonymous

The author notes that good requirements should have the following two characteristics:

- 1) "Requirements are complete. They must be as complete as possible with no open-ended parts or opportunity for interpretation."
- 2) "Requirements must be unambiguous. No wishy-washy statements nor (conceptually) anything that can be interpreted differently than intended."

However, the author also states that:

- 1) "user stories focus on customer value with a built-in imprecision meant to encourage communication."
- 2) "User stories (when used as intended by experts like Cohn and Cockburn) are just too loosely formatted to lend themselves to full documentation."

Clearly, user stories are intentionally not complete as possible and are ambiguous. Thus, user stories are *not* requirements but rather a verbal person-to-person communication technique for eliciting verbal requirements that are not documented, but rather rapidly implemented in code and test cases.

This leads to several limitations of user stories:

- 1) They require close person-to-person collaboration between developers and stakeholders/SMEs/customers. This is often not possible in practice, especially when dealing with large, expensive systems that are not developed in house.
- 2) A lack of documented requirements makes it very difficult to cost and schedule a large and complex system, especially when customers and developers belong to different companies so that financial contracts are an unavoidable fact of life.
- 3) While being agile is great when requirements truly need to change relatively often and rapidly, the lack of documented requirements enables the rapid unnecessary changing of requirements that can lead to unnecessary requirements churn and floundering. Volatile requirements, when change is not really necessary, has led to many cost and schedule overruns.
- 4) User stories can be an excuse for the customer, stakeholders, and SMEs to be sloppy in considering what they want/need. Although it is true that many people don't really know what they want until they can see it, finding out that you don't like the system until it is delivered (or at least a prototype is available) is too late to decide that it's not what you wanted after all. That is especially true for large, complex, expensive systems including systems that consist of more than just software (e.g., hardware, equipment, facilities)

So although user stories have their place and value and should be part of the requirements engineer's toolbox, they have their limitations and disadvantages as well as their benefits.

My problem is when groups of people pick single techniques as their only way of solving non-trivial problems (like engineering requirements). That's when using things like textual requirements, use cases, and (yes even) user stories become a matter of dogma and ideological purity. A good carpenter has many tools in their tool box providing a choice of multiple ways of doing something (it is better to have both flathead and Phillips screwdrivers as well as hammers/nails and wrenches/bolts). Software developers should be no different; selecting and using the right tool for the job at hand given the job and its constraints.

what they
ding out th
not what you wanted
more than just
toolbox, they have
lems (like
1) user stories
iding a choice of
ammers/nails and
at hand given the


article/de
fining-
requirement-
types-
tradition
al-vs-
use-
cases-
vs-user-
stories)

<http://www.addthis.com/bookmark.php?v=250>
<http://www.addthis.com/bookmark.php?v=250>
<http://www.addthis.com/bookmark.php?v=250>
<http://www.addthis.com/bookmark.php?v=250>
<http://www.addthis.com/bookmark.php?v=250>
<http://www.addthis.com/bookmark.php?v=250>

[article/defining-requirement-types-traditional-vs-use-cases-vs-user-stories](#)


d, in my opinion, be
s focus on value. A
mall part of the big
ne. Because use-
ends on the

January 26, 2012 - 2:42am



Anonymous
Hi Charles,
A short reaction on your great post from a requirements specialist in The Netherlands.
I agree mostly with you, traditional requirements are to much system focussed while requirements should, in my opinion, be user focussed. And both use cases and user stories are user focussed. And it is also true that user cases focus on value. A problem with user stories I've seen in practice is that user stories can be fragmented, users only see a small part of the big picture and sometime users miss the overview. In these situations story boards don't do the job all the time. Because use cases are more narritive users can better see the cohesion, the big picture.
In my experience you can use both use cases and user stories in an agile project. Which of the two depends on the situation, I agree on this point with you.
Great post, it made me think and cleared my opinion on this point.
Keep up the great work!
Jan Jaap Cannegieter
January 26, 2012 - 2:42am

About the author



</users/charles-suscheck>
Dr. Charles Suscheck is a nationally recognized agile leader who specializes in agile software development adoption at the enterprise level. He is one of the few [scrum.org](#) trainers certified to teach the entire Scrum.org cirriculum. With over 25 years of professional experience, Dr. Suscheck has held positions of Process Architect, Director of Research, Principle Consultant, Professor, and Professional Trainer at some of the most recognized companies in America. He has spoken at national and international conferences such as Agile 200X, OOPSLA, and ECOOP on topics related to agile project management and is a frequent author in industry and academia. Dr. Suscheck has over 30 publications to his credit.

More Like This

- » [Why DevOps Still Needs Release Management](/article/why-devops-still-needs-release-management) [article]
- » [Lessons Learned in Jenkins Configuration Management](/article/lessons-learned-jenkins-configuration-management) [article]

and-systems-article/defaulting-requirement-types-traditional-vs-al-vs-use-cases-vs-user-stories)

4) [article]

use-
cases-
vs-user-
stories).

the resources, Reckiton helps you develop and deliver great software