The Addison Wesley Signature Series

A Kent Beck Signature Book

User Stories Applied

For Agile Software Development

Mike Cohn

Foreword by Kent Beck

Table of Contents

Chapter 1. An Overview..... 1
What Is a User Story?..... 2
Where Are the Details?..... 3
"How Long Does It Have to Be?"..... 5
The Customer Team..... 6
What Will the Process Be Like?..... 6
Planning Releases and Iterations..... 8
What Are Acceptance Tests?..... 10
Why Change?..... 11
Summary..... 13
Questions..... 13

Chapter 1

An Overview

Software requirements is a communication problem. Those who want the new software (either to use or to sell) must communicate with those who will build the new software. To succeed, a project relies on information from the heads of very different people: on one side are customers and users and sometimes analysts, domain experts and others who view the software from a business or organizational perspective; on the other side is the technical team.

If either side dominates these communications, the project loses. When the business side dominates, it mandates functionality and dates with little concern that the developers can meet both objectives, or whether the developers understand exactly what is needed. When the developers dominate the communications, technical jargon replaces the language of the business and the developers lose the opportunity to learn what is needed by listening.

What we need is a way to work together so that neither side dominates and so that the emotionally-fraught and political issue of resource allocation becomes a shared problem. Projects fail when the problem of resource allocation falls entirely on one side. If the developers shoulder the problem (usually in the form of being told "I don't care how you do it but do it all by June") they may trade quality for additional features, may only partially implement a feature, or may solely make any of a number of decisions in which the customers and users should participate. When customers and users shoulder the burden of resource allocation, we usually see a lengthy series of discussions at the start of a project during which features are progressively removed from the project. Then, when the software is eventually delivered, it has even less functionality than the reduced set that was identified.

By now we've learned that we cannot perfectly predict a software development project. As users see early versions of the software, they come up with new ideas and their opinions change. Because of the intangibility of software, most developers have a notoriously difficult time estimating how long things will take. Because of these and other factors we cannot lay out a perfect PERT chart showing everything that must be done on a project.

So, what do we do?

We make decisions based on the information we have at hand. And we do it often. Rather than making one all-encompassing set of decisions at the outset of a project, we spread the decision-making across the duration of the project. To do this we make sure we have a process that gets us information as early and often as possible. And this is where user stories come in.

What Is a User Story?

A user story describes functionality that will be valuable to either a user or purchaser of a system or software. User stories are composed of three aspects:

- a written description of the story used for planning and as a reminder
- conversations about the story that serve to flesh out the details of the story
- tests that convey and document details and that can be used to determine when a story is complete

Because user story descriptions are traditionally hand-written on paper note cards, Ron Jeffries has named these three aspects with the wonderful alliteration of Card, Conversation, and Confirmation (Jeffries 2001). The Card may be the most visible manifestation of a user story, but it is not the most important. Rachel Davies (2001) has said that cards "*represent* customer requirements rather than *document* them." This is the perfect way to think about user stories: While the card may contain the text of the story, the details are worked out in the Conversation and recorded in the Confirmation.

As an example user story see Story Card 1.1, which is a story card from the hypothetical BigMoneyJobs job posting and search website.

> A user can post her resume to the website.

Story Card 1.1 An initial user story written on a note card.

For consistency, many of the examples throughout the rest of this book will be for the BigMoneyJobs website. Other sample stories for BigMoneyJobs might include:

- A user can search for jobs.
- A company can post new job openings.
- A user can limit who can see her resume.

Because user stories represent functionality that will be valued by users, the following examples do not make good user stories for this system:

- The software will be written in C++.
- The program will connect to the database through a connection pool.

The first example is not a good user story for BigMoneyJobs because its users would not care which programming language was used. However, if this were an application programming interface, then the user of that system (herself a programmer) could very well have written that "the software will be written in C++."

The second story is not a good user story in this case because the users of this system do not care about the technical details of how the application connects to the database.

Perhaps you've read these stories and are screaming "But wait— using a connection pool is a requirement in my system!" If so, hold on, the key is that stories should be written so that the customer can value them. There are ways to express stories like these in ways that are valuable to a customer. We'll see examples of doing that in Chapter 2, "Writing Stories."

Where Are the Details?

It's one thing to say "A user can search for jobs." It's another thing to be able to start coding and testing with only that as guidance. Where are the details? What about all the unanswered questions like:

- What values can users search on? State? City? Job title? Keywords?
- Does the user have to be a member of the site?
- Can search parameters be saved?
- What information is displayed for matching jobs?

Many of these details can be expressed as additional stories. In fact, it is better to have more stories than to have stories that are too large. For example, the entire BigMoneyJobs site is probably described by these two stories:

- A user can search for a job.
- A company can post job openings.

Clearly these two stories are too large to be of much use. Chapter 2, "Writing Stories," fully addresses the question of story size, but as a starting point it's good to have stories that can be coded and tested between half a day and perhaps two weeks by one or a pair of programmers. Liberally interpreted, the two stories above could easily cover the majority of the BigMoneyJobs site so each will likely take most programmers more than a week.

When a story is too large it is sometimes referred to as an *epic*. Epics can be split into two or more stories of smaller size. For example, the epic "A user can search for a job" could be split into these stories:

- A user can search for jobs by attributes like location, salary range, job title, company name, and the date the job was posted.
- A user can view information about each job that is matched by a search.
- A user can view detailed information about a company that has posted a job.

However, we do not continue splitting stories until we have a story that covers every last detail. For example, the story "A user can view information about each job that is matched by a search" is a very reasonable and realistic story. We do not need to further divide it into:

- A user can view a job description.
- A user can view a job's salary range.
- A user can view the location of a job.

Similarly, the user story does not need to be augmented in typical requirements documentation style like this:

4.6) A user can view information about each job that is matched by a search.
  4.6.1) A user can view the job description.
  4.6.2) A user can view a job's salary range.
  4.6.3) A user can view the location of a job.

Rather than writing all these details as stories, the better approach is for the development team and the customer to discuss these details. That is, have a conversation about the details at the point when the details become important. There's nothing wrong with making a few annotations on a story card based on

a discussion, as shown in Story Card 1.2. However, the conversation is the key, not the note on the story card. Neither the developers nor the customer can point to the card three months later and say, "But, see I said so right there." Stories are not contractual obligations. As we'll see, agreements are documented by tests that demonstrate that a story has been developed correctly.

> Users can view information about each job that is matched by a search.
>
> Marco says show description, salary, and location.

Story Card 1.2 A story card with a note.

"How Long Does It Have to Be?"

I was the kid in high school literature classes who always asked, "How long does it have to be?" whenever we were assigned to write a paper. The teachers never liked the question but I still think it was a fair one because it told me what their expectations were. It is just as important to understand the expectations of a project's users. Those expectations are best captured in the form of the acceptance tests.

If you're using paper note cards, you can turn the card over and capture these expectations there. The expectations are written as reminders about how to test the story as shown in Story Card 1.3. If you're using an electronic system it probably has a place you can enter the acceptance test reminders.

> Try it with an empty job description.
> Try it with a really long job description.
> Try it with a missing salary.
> Try it with a six-digit salary.

Story Card 1.3 The back of a story card holds reminders about how to test the story.

The test descriptions are meant to be short and incomplete. Tests can be added or removed at any time. The goal is to convey additional information about the story so that the developers will know when they are done. Just as my teacher's expectations were useful to me in knowing when I was done writing about *Moby Dick,* it is useful for the developers to know the customer's expectations so they know when they are done.

The Customer Team

On an ideal project we would have a single person who prioritizes work for developers, omnisciently answers their questions, will use the software when it's finished, and writes all of the stories. This is almost always too much to hope for, so we establish a customer team. The customer team includes those who ensure that the software will meet the needs of its intended users. This means the customer team may include testers, a product manager, real users, and interaction designers.

What Will the Process Be Like?

A project that is using stories will have a different feel and rhythm than you may be used to. Using a traditional waterfall-oriented process leads to a cycle of write all the requirements, analyze the requirements, design a solution, code the solution, and then finally test it. Very often during this type of process, customers and users are involved at the beginning to write requirements and at the end to accept the software, but user and customer involvement may almost entirely disappear between requirements and acceptance. By now, we've learned that this doesn't work.

The first thing you'll notice on a story-driven project is that customers and users remain involved throughout the duration of the project. They are not expected (or allowed!) to disappear during the middle of the project. This is true whether the team will be using Extreme Programming (XP; see Appendix A, "An Overview of Extreme Programming," for more information), an agile version of the Unified Process, an agile process like Scrum (see Chapter 15, "Using Stories with Scrum"), or a home-grown, story-driven agile process.

The customers and intended users of the new software should plan on taking a very active role in writing the user stories, especially if using XP. The story writing process is best started by considering the types of users of the intended

system. For example, if you are building a travel reservation website, you may have user types such as frequent fliers, vacation planners, and so on. The customer team should include representatives of as many of these user types as practical. But when it can't, user role modeling can help. (For more on this topic see Chapter 3, "User Role Modeling.")

Why Does the Customer Write the Stories?

The customer team, rather than the developers, writes the user stories for two primary reasons. First, each story must be written in the language of the business, not in technical jargon, so that the customer team can prioritize the stories for inclusion into iterations and releases. Second, as the primary product visionaries, the customer team is in the best position to describe the behavior of the product.

A project's initial stories are often written in a story writing workshop, but stories can be written at any time throughout the project. During the story writing workshop, everyone brainstorms as many stories as possible. Armed with a starting set of stories, the developers estimate the size of each.

Collaboratively, the customer team and developers select an iteration length, from perhaps one to four weeks. The same iteration length will be used for the duration of the project. By the end of each iteration the developers will be responsible for delivering fully usable code for some subset of the application. The customer team remains highly involved during the iteration, talking with the developers about the stories being developed during that iteration. During the iteration the customer team also specifies tests and works with the developers to automate and run tests. Additionally, the customer team makes sure the project is constantly moving toward delivery of the desired product.

Once an iteration length has been selected, the developers will estimate how much work they'll be able to do per iteration. We call this *velocity*. The team's first estimate of velocity will be wrong because there's no way to know velocity in advance. However, we can use the initial estimate to make a rough sketch, or release plan, of what work will happen in each iteration and how many iterations will be needed.

To plan a release, we sort stories into various piles with each pile representing an iteration. Each pile will contain some number of stories, the estimates for which add up to no more than the estimated velocity. The highest-priority stories go into the first pile. When that pile is full, the next highest-priority stories go into a second pile (representing the second iteration). This continues until

you've either made so many piles that you're out of time for the project or until the piles represent a desirable new release of the product. (For more on these topics see Chapter 9, "Planning a Release," and Chapter 10, "Planning an Iteration.")

Prior to the start of each iteration the customer team can make mid-course corrections to the plan. As iterations are completed, we learn the development team's actual velocity and can work with it instead of the estimated velocity. This means that each pile of stories may need to be adjusted by adding or removing stories. Also, some stories will turn out to be far easier than anticipated, which means the team will sometimes want to be given an additional story to do in that iteration. But some stories will be harder than anticipated, which means that some work will need to be moved into later iterations or out of the release altogether.

Planning Releases and Iterations

A release is made up of one or more iterations. Release planning refers to determining a balance between a projected timeline and a desired set of functionality. Iteration planning refers to selecting stories for inclusion in this iteration. The customer team and the developers are both involved in release and iteration planning.

To plan a release, the customer team starts by prioritizing the stories. While prioritizing they will want to consider:

- The desirability of the feature to a broad base of users or customers
- The desirability of the feature to a small number of important users or customers
- The cohesiveness of the story in relation to other stories. For example, a "zoom out" story may not be high priority on its own but may be treated as such because it is complementary to "zoom in," which is high priority.

The developers have different priorities for many of the stories. They may suggest that the priority of a story be changed based on its technical risk or because it is complementary to another story. The customer team listens to their opinions but then prioritizes stories in the manner that maximizes the value delivered to the organization.

Stories cannot be prioritized without considering their costs. My priority for a vacation spot last summer was Tahiti until I considered its cost. At that point

other locations moved up in priority. Factored into the prioritization is the cost of each story. The cost of a story is the estimate given to it by the developers. Each story is assigned an estimate in *story points*, which indicates the size and complexity of the story relative to other stories. So, a story estimated at four story points is expected to take twice as long as a story estimated at two story points.

The release plan is built by assigning stories to the iterations in the release. The developers state their expected velocity, which is the number of story points they think they will complete per iteration. The customer then allocates stories to iterations, making sure that the number of story points assigned to any one iteration does not exceed the expected team velocity.

As an example, suppose that Table 1.1 lists all the stories in your project and they are sorted in order of descending priority. The team estimates a velocity of thirteen story points per iteration. Stories would be allocated to iterations as shown in Table 1.2.

Table 1.1 *Sample stories and their costs.*

Story	Story Points
Story A	3
Story B	5
Story C	5
Story D	3
Story E	1
Story F	8
Story G	5
Story H	5
Story I	5
Story J	2

Because the team expects a velocity of thirteen, no iteration can be planned to have more than thirteen story points in it. This means that the second and third iterations are planned to have only twelve story points. Don't worry about it—estimation is rarely precise enough for this difference to matter, and if the developers go faster than planned they'll ask for another small story or two. Notice that for the third iteration the customer team has actually chosen to

include Story J over the higher priority Story I. This is because Story I, at five story points, is actually too large to include in the third iteration.

Table 1.2 *A release plan for the stories of Table 1.1.*

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, J	12
Iteration 4	I	5

An alternative to temporarily skipping a large story and putting a smaller one in its place in an iteration is to split the large story into two stories. Suppose that the five-point Story I could have been split into Story Y (three points) and Story Z (two points). Story Y contains the most important parts of the old Story I and can now fit in the third iteration, as shown in Table 1.3. For advice on how and when to split stories see Chapter 2, "Writing Stories," and Chapter 7, "Guidelines for Good Stories."

Table 1.3 *Splitting a story to create a better release plan.*

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, Y	13
Iteration 4	J, Z	4

What Are Acceptance Tests?

Acceptance testing is the process of verifying that stories were developed such that each works exactly the way the customer team expected it to work. Once an iteration begins, the developers start coding and the customer team starts specifying tests. Depending on the technical proficiency of customer team members, this may mean anything from writing tests on the back of the story card to putting the tests into an automated testing tool. A dedicated and skilled tester should be included on the customer team for the more technical of these tasks.

Tests should be written as early in an iteration as possible (or even slightly before the iteration if you're comfortable taking a slight guess at what will be in

the upcoming iteration). Writing tests early is extremely helpful because more of the customer team's assumptions and expectations are communicated earlier to the developers. For example, suppose you write the story "A user can pay for the items in her shopping cart with a credit card." You then write these simple tests on the back of that story card:

- Test with Visa, MasterCard and American Express (pass).
- Test with Diner's Club (fail).
- Test with a Visa debit card (pass).
- Test with good, bad and missing card ID numbers from the back of the card.
- Test with expired cards.
- Test with different purchase amounts (including one over the card's limit).

These tests capture the expectations that the system will handle Visa, MasterCard and American Express and will not allow purchases with other cards. By giving these tests to the programmer early, the customer team has not only stated their expectations, they may also have reminded the programmer of a situation she had otherwise forgotten. For example, she may have forgotten to consider expired cards. Noting it as a test on the back of the card before she starts programming will save her time. For more on writing acceptance tests for stories see Chapter 6, "Acceptance Testing User Stories."

Why Change?

At this point you may be asking why change? Why write story cards and hold all these conversatons? Why not just continue to write requirements documents or use cases? User stories offer a number of advantages over alternative approaches. More details are provided in Chapter 13, "Why User Stories?", but some of the reasons are:

- User stories emphasize verbal rather than written communication.
- User stories are comprehensible by both you and the developers.
- User stories are the right size for planning.
- User stories work for iterative development.

- User stories encourage deferring detail until you have the best understanding you are going to have about what you really need.

Because user stories shift emphasis toward talking and away from writing, important decisions are not captured in documents that are unlikely to be read. Instead, important aspects about stories are captured in automated acceptance tests and run frequently. Additionally, we avoid obtuse written documents with statements like:

> The system must store an address and business phone number or mobile phone number.

What does that mean? It could mean that the system must store one of these:

> (Address and business phone) or mobile phone
> Address and (business phone or mobile phone)

Because user stories are free of technical jargon (remember, the customer team writes them), they are comprehensible by both the developers as well as the customer team.

Each user story represents a discrete piece of functionality; that is, something a user would be likely to do in a single setting. This makes user stories appropriate as a planning tool. You can assess the value of shifting stories between releases far better than you can assess the impact of leaving out one or more "The system shall…" statements.

An iterative process is one that makes progress through successive refinement. A development team takes a first cut at a system, knowing it is incomplete or weak in some (perhaps many) areas. They then successively refine those areas until the product is satisfactory. With each iteration the software is improved through the addition of greater detail. Stories work well for iterative development because it is also possible to iterate over the stories. For a feature that you want eventually but that isn't important right now, you can first write a large story (an epic). When you're ready to add that story into the system you can refine it by ripping up the epic and replacing it with smaller stories that will be easier to work with.

It is this ability to iterate over a story set that allows stories to encourage the deferring of detail. Because we can write a placeholder epic today, there is no need to write stories about parts of a system until close to when those parts will be developed. Deferring detail is important because it allows us to not spend time thinking about a new feature until we are positive the feature is needed. Stories discourage us from pretending we can know and write everything in advance. Instead they encourage a process whereby software is iteratively refined based on discussions between the customer team and the developers.

Summary

- A story card contains a short description of user- or customer-valued functionality.
- A story card is the visible part of a story, but the important parts are the conversations between the customer and developers about the story.
- The customer team includes those who ensure that the software will meet the needs of its intended users. This may include testers, a product manager, real users, and interaction designers.
- The customer team writes the story cards because they are in the best position to express the desired features and because they must later be able to work out story details with the developers and to prioritize the stories.
- Stories are prioritized based on their value to the organization.
- Releases and iterations are planned by placing stories into iterations.
- Velocity is the amount of work the developers can complete in an iteration.
- The sum of the estimates of the stories placed in an iteration cannot exceed the velocity the developers forecast for that iteration.
- If a story won't fit in an iteration, you can split the story into two or more smaller stories.
- Acceptance tests validate that a story has been developed with the functionality the customer team had in mind when they wrote the story.
- User stories are worth using because they emphasize verbal communication, can be understood equally by you and the developers, can be used for planning iterations, work well within an iterative development process, and because they encourage the deferring of detail.

Questions

1.1 What are the three parts of a user story?

1.2 Who is on the customer team?

1.3 Which of the following are not good stories? Why?

a The user can run the system on Windows XP and Linux.

b All graphing and charting will be done using a third-party library.

c The user can undo up to fifty commands.

d The software will be released by June 30.

e The software will be written in Java.

f The user can select her country from a drop-down list.

g The system will use Log4J to log all error messages to a file.

h The user will be prompted to save her work if she hasn't saved it for 15 minutes.

i The user can select an "Export to XML" feature.

j The user can export data to XML.

1.4 What advantages do requirements conversations have over requirements documents?

1.5 Why would you want to write tests on the back of a story card?