# Telling Better User Stories

# Mapping the Path to Success

by Jeff Patton

ISTOCKPHOTO

**The User Story** has emerged as a standard practice in agile development processes. While the idea of user stories is simple on the surface, there are challenges to working with them—particularly when working with the multiple stories required to create a successful product. User story mapping is a useful way to organize, decompose, and prioritize user stories. A good story map keeps conversation about your product alive and productive throughout its construction. But, to understand user story mapping, we first need to understand user stories.

## User Stories

### User Stories Are for Conversation

When I ask people for a definition of user story, the most common response I receive is "a token for a conversation." Extreme Programming (XP) popularized user stories and that particular definition, which came from Alistair Cockburn during some of the earliest discussions about XP. The name "user story" comes from the idea that we should tell the "story" of what's needed and why it's valuable and not just document requirements. In telling the story, we'll arrive at a better idea of what to build. Having that conversation is the most important aspect of the user story.

A user story is a boundary object—a point of agreement that different groups with different concerns use to collaborate. While every group uses the same name for the story, each tends to consider only its own concerns while discussing stories. Users think of the stories as descriptions of their needs. Business people think of them as product features that will generate return on investment. Developers think of them as the specifications for software to build. Testers think of them as things they need to validate. Project managers think of them as work to schedule into a release. All of these views are correct, but all of them are also incomplete.

A user story isn't an ideal user problem description, software specification, test script, or project management task. It's the token that allows these communities to think about their own needs while they collaborate over their mutual concern—the software they'd all like to see developed.

It's difficult to engage in conversations without emphasizing our own concerns. A story that works well as a boundary object has meaning to everyone involved with the project but isn't so specific that it loses meaning for any one person. Extra-specific details cloud the conversation and erode the value of the story.

Of course, individuals need all those details, and the details do need to be discussed. The trick is to keep very specific details to yourself. Understand that they are only relevant to some conversations you need to have, and keep them handy for those conversations.

### Organizing the Prioritized Story Backlog

User stories that describe only a small chunk of software are quicker to code and test and, when completed, show progress. Showing progress in the form of working software is an important characteristic of agile development. If I'm scheduling work into a short timebox or sprint, or figuring out the details of software to build and test, small is good. But small is a relative term. The ideal-size story from a user's perspective may be far too large from a developer's perspective.

A common goal in agile—and specifically in the Scrum community—is maintaining a "prioritized backlog." Organizing user stories into a backlog allows us to prioritize and schedule both the detailed conversations about the stories and their construction. But, when the backlog consists of hundreds of stories, it can be virtually impossible to create a mental picture of the product as a whole.

People looking at a prioritized backlog can have good conversations about the schedule, but other conversations about the product can be difficult. It's not enough to write stories and prioritize your backlog for implementation. A backlog that really supports conversation has three important qualities:

**Descriptive**—It helps us understand the users and their needs and how the software addresses those needs. It helps us visualize the entire product.

**Right sized**—High-level conversations have high-level stories; detailed conversations have more detailed stories. Conversations must be enabled at the executive level and at the detailed development level.

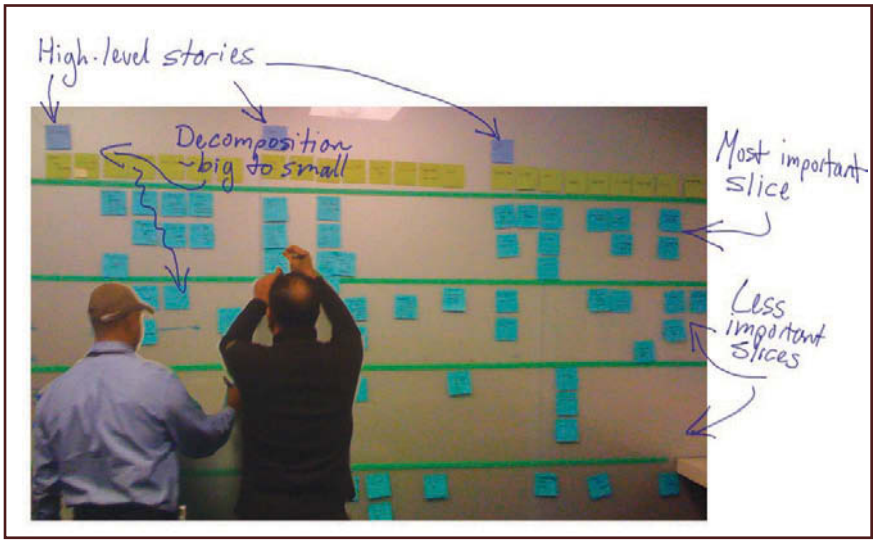


**Figure 1: User Story Mapping**

**Prioritized**—The most valuable items, the ones we need to focus on first, can easily be identified.

### User Story Mapping

A user story map places high-level stories in a meaningful order, typically left to right and top to bottom as shown in figure 1. Often, this reflects the user's workflow order or business process and allows us to tell a meaningful story about a significant part of the entire system

without getting lost in the details. These top-level stories form the backbone of our application.

Below each of the top stories is a similar left-to-right "decomposition" of that story into mid-level stories that break it down. The mid-level stories could be the steps in a business process or details of an activity in which a person engages. During a conversation, we can drop down to this level to dig into the details of any one top-level story. Below each of these mid-level stories are more stories that decompose into more detailed stories. To save space, we stack the detailed stories in a nice vertical column.

So far, the shape of the map has helped us get to the descriptive and right-sized qualities we need now to layer in prioritization. The map is "sliced" into product releases. We'll only slice the detailed stories since they're the smallest bits—the parts we're likely to build in a short iteration or sprint. Slicing the map lets us see what's in each release over time and how the top stories grow in capability with each release.

With a story map, it's easy to spot holes in our conversations—places where the story map is lacking and where more mid-level or detailed stories should be identified. In some cases, we'll identify planned omissions—places where we've purposely deferred functionality for later release.

## Working with Story Maps

Working with a story map follows a simple process:

### 1. Create a story map

Start by identifying user stories and arranging them into the shape of a story map. Focus on breadth, not depth. The fastest way to generate a lot of stories is to think through a user's experience. Let's consider "booking an airline flight," since most of us have had this experience.

The basic tasks to book a flight are:

1. Choose an origin and destination.
2. Choose the departure and return dates.
3. Search for flights.
4. Review the flight options.
5. Choose the flights.
6. Pay for the flight.

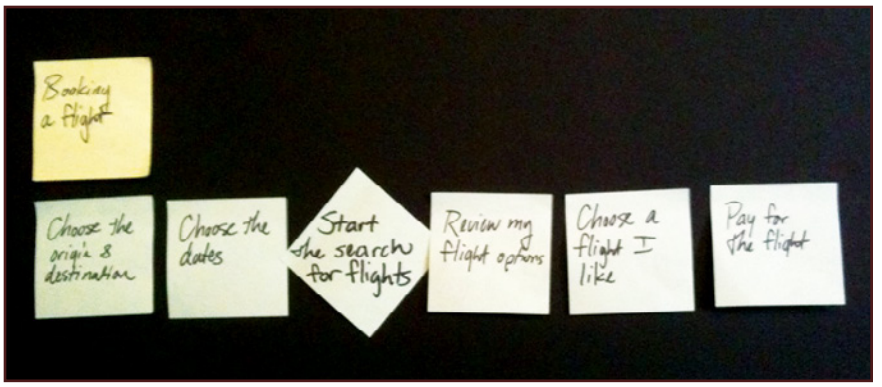


Figure 2

I'll write these user tasks down on stickies and lay them out left to right. On a different color sticky, I write "Booking a flight" and place the sticky on top of my neat row as shown in figure 2.

I used the phrase "user task" to describe things a user would like to accomplish. User tasks usually start with a verb to help us understand what they're doing. We'll call these "task-centric user stories."

To help us remember that user stories should be about users, this popular template has emerged:

As a [*type of user*],
I want to [*do something*]
so that [*I get some value*].

I like using this template to check my stories: "As a *traveler*, I want to *choose the flights I like* so that *I can fly at the times that suit my travel plans*." However, I don't write all those words because I don't need them. The "working words" are the short verb phrases such as "choose the flights." That's the least I can write and still be clear enough about what a user of a booking system would want to do. Writing too many words makes the map hard to read and diminishes its value as a conversation piece.

At the outset, we'll want to identify the full breadth of a potential system. Think of the other big activities that people might engage in with the reservation system. Based on my experience, my initial list is:

1. Change my flight.
2. Check in.
3. Book a flight I often take.

I could go on, and, if I were building such a system, I would.

These big activities bundle a number of smaller user tasks performed to achieve a larger goal. These smaller tasks often can be performed in varying order. For instance, I might stop halfway through booking a flight and go back to the beginning. I might always do some smaller tasks, like choosing flight dates, and rarely do others, like selecting special meal options. But, all those smaller tasks support a common goal—the user activity.

Because I know that when I'm engaged in an activity like "booking a flight" I might skip steps or back up, I don't get too concerned about the order in which I put the stickies. They're in the order that helps me best discuss it with others.

In Mike Cohn's book *User Stories Applied*, he coins the term "epic" to refer to a big user story, one that should be broken down further before placing the story into a next sprint or iteration. These user-centric activities that we've placed as top-level stories in our map are likely "epics." But, the task-centric stories we've placed as mid-level stories could be as well. At this point, they're all too big and too poorly understood to be built, so don't get caught up in the epic versus story distinction.

Thinking through the user experience as activities and tasks will give you a beginning story map a mile wide and an inch deep. Building the story map is often eye opening. For developers and testers who may be used to focusing on the details, this may be the first time they've seen the big picture.

## 2. Fill in and validate

Once the map looks right, start diving a bit deeper. Discussing the map with others helps you identify missing stories or alternative ways of doing things.

Let's go back to "booking a flight" and start with the task "choose an origin and destination." Thinking about the details, we can identify smaller sub-tasks inside that task: "enter airport code," "enter city name," or "look up airport code." Since I travel to New York City frequently, and three major airports serve that area, I often "choose to include nearby airports" in my search. All these are tasks I write on stickies and hang below the "choose an origin and destination" sticky. Doing this for all the mid-level, task-centric stories in the user-centric story at the top results in a more complete map.

Each kind of user is coming to our booking system with varying amounts of experience but with a goal in mind. These are user roles, and they overlap a bit. Think of roles like hats you wear and can change as your goal changes. For example, I'm often a frequent business traveler, but sometimes I'm a family vacation booker. And, because I'm a cheapskate, I always have a bit of price shopper in me.

If I put on my frequent business traveler hat and walk the story map, I remember my goals a bit better and add to my tasks. I often "check my current mileage balance" and "place my name on the first-class upgrade list." I hate spending too much time on planes so, while reviewing my flight options, I always "look at the number of stops" and "look for shortest flight duration."


**Figure 3**

But, I'm a frequent flyer and probably not the best example of all the types of users who might interact with this system.

I took a moment to brainstorm different kinds of users who could use our system and add them to my map, as shown in figure 3.

- Frequent business traveler
- Family vacation booker
- Price shopper
- Infrequent, perplexed traveler

When I put my family vacation booker hat on, I remember other tasks like "entering family member names and ages" and "saving flight itineraries to discuss later" with my wife, who needs to make this decision with me.

Understanding different types of users and walking the map from their perspective expand the map with things you might have forgotten otherwise.

It's also a good idea to begin walking the map with others on the team including engineers, testers, and other stakeholders. They begin to understand the user's experience and why it is the way it is. They can point out areas where writing software to support such a user task may be difficult. Or, better yet, when armed with a good understanding of the user experience, they can identify product solution ideas that would really help the users.

As you fill in and validate the map, you'll add, split, reorganize, and annotate stories with details and cool solution ideas. As you walk and talk through the story map with a variety of people, you'll gain confidence that it's complete.

When you're comfortable that you haven't left out anything big, it's time to move on to make choices about what to build now, what to build later, and, possibly, what never to build at all.

## 3. Plan releases

Armed with a strong understanding, we can begin to prioritize what's most important to implement in our software and what we might be able to do without. We then can slice our map into coherent releases of detailed stories.

When trying to identify what to build first, I focus on two levels of planning. First, I think about what I should release first to create a minimally viable product and what should come next in my product roadmap. This is called release planning. Knowing the goals of a first release allows me to focus on what to build in that first release. I'll target functionality that helps me learn fast about the product. I'll try to create a fully functional "walking skeleton" [1] of the product as soon as possible. Keep these two levels of planning in mind as you start to prioritize stories.

If I had a nickel for every time I heard someone say, "Prioritization is my biggest challenge when working with stories," I'd have a lot of nickels. The trick to prioritization is not to work with the stories first but to work outside in the product's context. By context, I mean the customers and users of our product and the business strategy that motivates building the product.

We have previously identified a few user types such as frequent business traveler and family vacation booker. Start by prioritizing those.

Now, whether business travelers or vacation bookers are more important to our organization depends on what we're trying to do with the product. Are we trying to build the best business travel site on the Web? Or, do we think that the vacation planner is an underserved market and we'd do best to focus on them? These sorts of decisions are strategic business decisions. Don't try to prioritize details about what to build until you're clear on the business strategy.

Business stakeholders ideally will make strategy decisions informed by market research, revenue- or cost-saving goals, or other considerations. Once those decisions are made, we can see more clearly which users are most critical to realizing our business strategy. Given that, we can begin to prioritize what those users need to do with the system.

The left-to-right organization of the story map supports good discussion. The top-to-bottom organization shows decomposition. When it comes time to build specific bits of sofware, it's the smallest detail stories that we'll want to build, so they're the ones we'll prioritize into releases.

I like to do prioritizing as a group so we're all on the same page. First, post and discuss the business strategy and organize the user types into priority order. Then, create slices in our story map by adding tapelines left to right to create horizontal slices, each representing a release.

Together, we'll begin moving the smallest stories up into their respective release slices. The first release slice on the top should contain all the stories needed to deliver our minimal viable product. Each slice below that will add a layer of capability to build up the product one release at a time.

Don't get tangled up in prioritizing one high-level or mid-level story against another. It doesn't do any good to debate "What's higher priority: choosing origin and destination or entering travel dates?" It's an irrelevant question since we know the product needs to support the activity of booking a flight and both tasks are critical for that. Focus on the detail-level stories and what slice they belong in. Focus on identifying the smallest first slice that will support your target user and business strategy.

While planning, we may split stories, add more stories, reword stories to make them clearer, or reorganize the map to better support discussion.

At the end of this exercise, we have a release roadmap ordered top to bottom where each slice should be in a coherent product release. We can test a slice's coherence by walking it left to right and talking through a user's experience. If we start with our highest priority user and describe his experience, pointing out stories as we go, we do so only by discussing stories inside the release slice. If we can't tell the big story of our user's experience with the first release without dipping down into second-release functionality, then we know we have not included sufficient functionality.

When you're comfortable with a release plan, you can summarize each slice or release by giving it a name, writing a couple of short sentences expressing the business benefit for building it and what the user's benefit is when using it. This list of names and benefits—along with a few bulleted, high-level stories—is our product roadmap.

### 4. Construct Software

It's time to get to work building something. Working with the detailed stories and moving story by story, we'll choose what we'd like to implement in the upcoming development cycle. Over time, we see the stories in our release being completed.

During product construction, keep the story map visible so it can help give context to further detailed discussions. Ignore the slices for releases other than the one on which you're working. I remove them from the story map completely, setting them aside to discuss as we get ready for the next release. Break the current release again into three thinner slices. Using a chess game metaphor, I call the slices "opening game," "mid game," and "end game."

The opening game slice contains the simplest possible functional version of the product. Remember, we're not releasing yet, so it doesn't need to be release ready. But, it should help us validate our functional scope, our architecture, and, at the end, be able to support end-to-end performance and load testing. It ideally contains a little bit from every major activity in the system. I refer to these activities as the "backbone" and this very thin, not quite releasable product, as our "walking skeleton."

Mid game stories focus on getting the system to full functional completeness. It's important in mid game to ramp up the amount of validation we do with users and with the system architecture.

End game stories focus on polish, fit, and finish. We'll need to make sure we have time during end game to accommodate all the "predictably unpredictable" work that comes in as a result of the user validation and other testing we've been doing.

## Keep Your Eye on the Prize

Keeping your eye on a coherent, growing system and constantly evaluating it against the benefits you targeted in your release plan are critical. And, it's not something one person does alone. The entire team needs to help keep an eye on these things.

Keep visible your business strategy, target user types, release roadmap, and user story map. Mark off stories that are done. Discuss how ready your product is to release at a high level, user activity by user activity. Use the map to pull your attention away from the minutia of day-to-day work and back to the big picture of the product you're growing. {end}