Chapter 6. The Real Story About Stories

Story mapping is a remarkably simple idea. Using a simple map, you can work with others to tell a product's story and see the big picture form as you do. Then, you carve up that big picture to make good planning decisions. Underneath all that is the simple concept of Agile stories.

Kent's Disruptively Simple Idea

The idea of stories originated with a very smart guy by the name of Kent Beck. Kent was working with other people on software development in the late '90s, and he noticed that one of the biggest problems in software development sprang from the traditional process approach of using documents to describe precisely what we want—that is, the requirements. By now you know the problem with that. Different people can read the same document and imagine different things. They can even "sign off" on the document believing they're in agreement.

It's later, when we get into the thick of developing software—or even later than that, after it's delivered—that we realize we weren't thinking of the same things. Lots of people call this lack of shared understanding "bad requirements."

Let me vent a minute here. I have the pleasure of working with lots of teams. And we often start work together by talking about their biggest challenges. And, hands down, the one I hear most is "bad requirements." And then everyone points at that document. The document writer feels bad—as if he should have written more, or less, or used some cool requirements technique. Those who signed off feel bad at first, and then indignant. "Surely you didn't expect I'd read every detail! After all, we talked about this for days. I just expected you'd understood what I said. I can't understand your silly requirements document anyway." And the people building the software feel blindsided. They muddled their way through those cryptic documents and what they built was still wrong. Everyone hates the document in the end. Yet we still keep trying to write a better one.

> We can both read the same document, but have a different understanding of it.

But misunderstanding the document is only half the problem. We waste lots of time and money building what the document describes, only to find out later that what actually solves the intended problem is something very different. You heard me right. Those documents often accurately describe the wrong thing. Documents usually describe *what* we need, but not *why* we need it. If the person building software could simply speak with someone who understood the users who will be using the software and why they'll be using it, there's often a more cost-effective way to delight those users. Without talking, we just never know about it.

> The best solutions come from collaboration between the people with the problems to solve and the people who can solve them.

Kent's simple idea was to *stop it*—to stop working so hard on writing the perfect document, and to get together to *tell stories*. Stories get their name not from how they're supposed to be written, but from how they're supposed to be used. Let me repeat that with more emotion. Right now you should stop whatever you're doing and say this out loud:

> Stories get their name from how they're supposed to be used, not from what you're trying to write down.

Kent's idea was simple. If we get together and talk about the problem we're solving with software, who'll use it, and why, then together we can arrive at a solution, and build shared understanding along the way.

Simple Isn't Easy

A while back I began to notice that this entire story thing had gone a bit sideways; that is, lots of the people writing books, teaching, and using them focused on the activity of writing stories. If I had a dime for every time I've been asked how to write good stories, I'd have even more dimes than I collected a few chapters ago.

With all the focus on writing stories, I went back to Kent to make sure I wasn't missing something here. Over the course of an email conversation, Kent explained where the idea came from:

> What I was thinking of was the way users sometimes tell stories about the cool new things the software they use does. [For example,] if I type in the zip code and it automatically fills in the city and state without me having to touch a button.
>
> I think that was the example that triggered the idea. If you can tell stories about what the software does and generate interest and vision in the listener's mind, then why not tell stories before the software does it?
>
> — Kent Beck via personal email, Aug 2010

So the idea is *telling*, and you know you're doing it right if you're generating energy, interest, and vision in the listener's mind. That's big. And it sounds a lot more fun than reading a typical requirements document.[9]

But folks who start using stories for software development—and who still have a traditional process model in their heads—tend to focus on the writing part. I've seen teams replace their traditional requirements process with story writing, and then get frustrated trying to write stories to precisely communicate what should be built. If you're doing that now, stop it.

> If you're not getting together to have rich discussions about your stories, then you're not really using stories.

Ron Jeffries and the 3 Cs

In the book *Extreme Programming Installed*, Ron Jeffries et al. (Addison-Wesley Longman Publishing) describe the story process best:

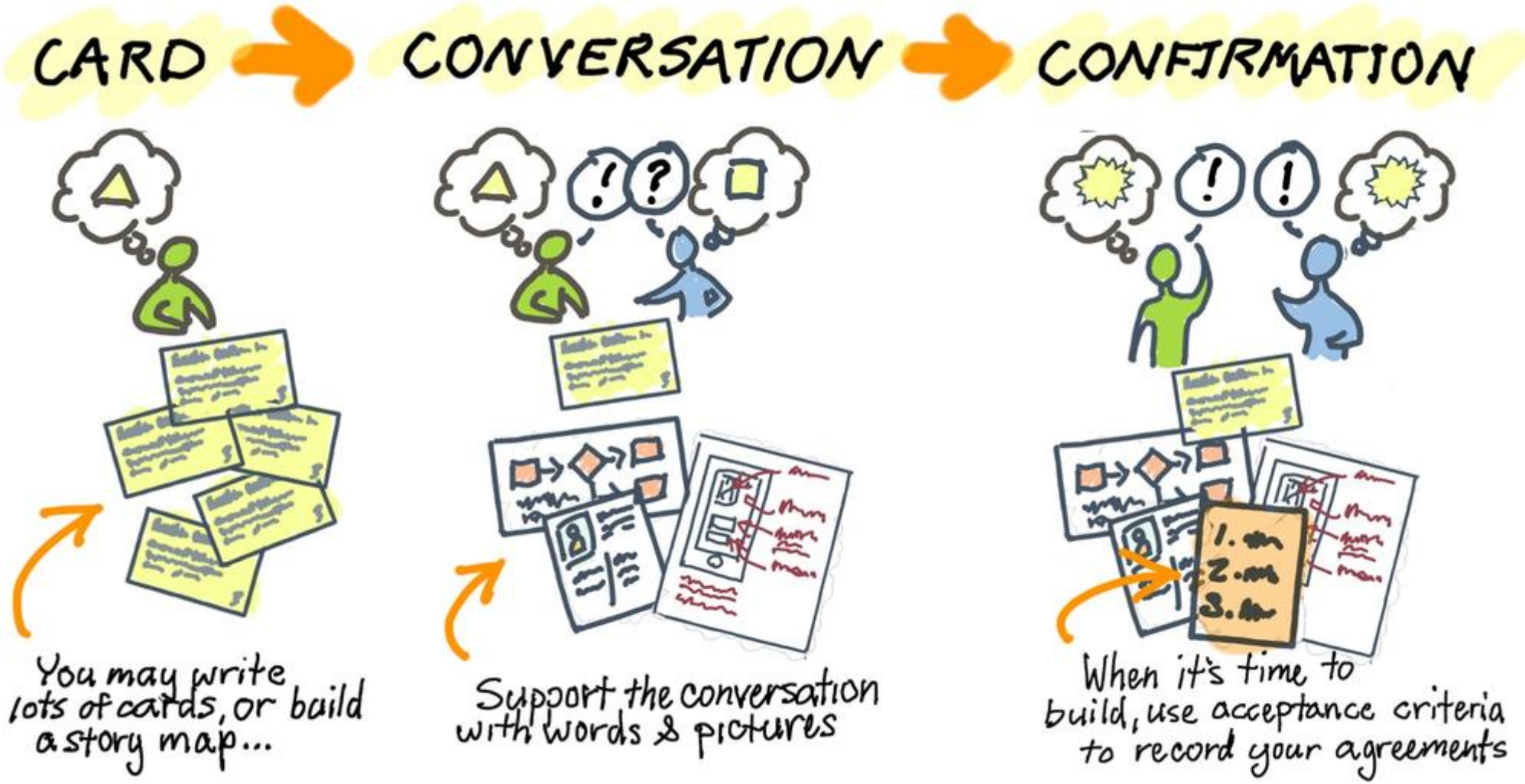
Card
: Write what you'd like to see in the software on a bunch of index cards.

Conversation
: Get together and have a rich conversation about what software to build.

Confirmation

Together agree on how you'll confirm that the software is done.

If it sounds simple, it's because it is. Just remember, simple isn't easy.



1. Card

Imagine you're responsible for working with a team to get some software built. Imagine that software as best you can. Then, for each thing users want to do with the product, write a card. You'll end up with a bunch of them. Kent's original idea was to write them on index cards because it's easy to organize a bunch of cards on a tabletop. It's easier to prioritize them, or organize them into a structure that helps you see the big picture—a structure like a story map, of course.

The bunch of cards that describes the whole product, or all the changes we'd like to make to a product we have, is called a *product backlog*. That term originated with the Agile process Scrum. Someone I know once said, "I hate that term *backlog*. We haven't even started to build software, and it already sounds like we're behind!" I'm not sure I have a better name for that bunch of stories, but if you can think of a good one, please use it, and let me know what it is.

2. Conversation

The conversation might start with you describing what you're thinking, and the person listening might form an idea in her head based upon what she heard. Because it's hard to explain things perfectly, and because it's easy to imagine different things based on our past experience, the listener likely imagined something different than you. But that's where the magic comes in.

Because this is a conversation, the listener can ask questions and it's the back and forth that will correct that understanding and help everyone arrive at some shared understanding.

In a traditional software process, the goal for the person who had the requirements is to get them written down correctly; and for the person who's going to build the software, it's to understand them correctly. Because this is a story-driven process, you each have a different shared goal. Your goal is to work together to understand the problem being solved by building software, and solve it as best you can. Eventually, you'll need to agree on what you should build that you both believe will help the users of the product.

Let me say this again, because it's an important point:

> Story conversations are about working together to arrive at a best solution to a problem we both understand.

3. Confirmation

All this talking is cool, but we've eventually got to build some software—right? So, when we feel like we're converging on a good solution, we'll need to start focusing on the answer to these questions:

If we build what we agree to, what will we check to see that we're done?
: The answer to this question is usually a short list of things to check. This list is often called *acceptance criteria*, or *story tests*.

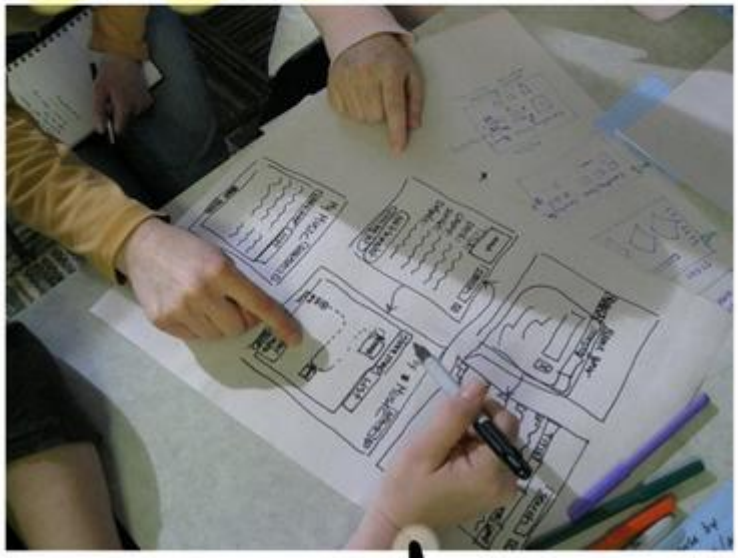
When it comes time to demonstrate this later at a product review, how will we do that?
: The answer to this question often reveals some holes. For example, you could make the software work, but to demonstrate it you'll need to get your hands on some realistic data. Discussing demonstration may add a few more bullets to your list of acceptance criteria.

Words and Pictures

The path to get to this agreement isn't just a single card and lots of hand waving. The conversation goes best if it includes lots of things such as simple personas, workflow diagrams, UI sketches, and any other traditional software model that'll help explain things. That way, you don't have to just wave hands—you can point a lot, too. Whatever we bring into the conversation we'll mark up, write on, correct, and change. We'll even create a lot of things on the spot during that

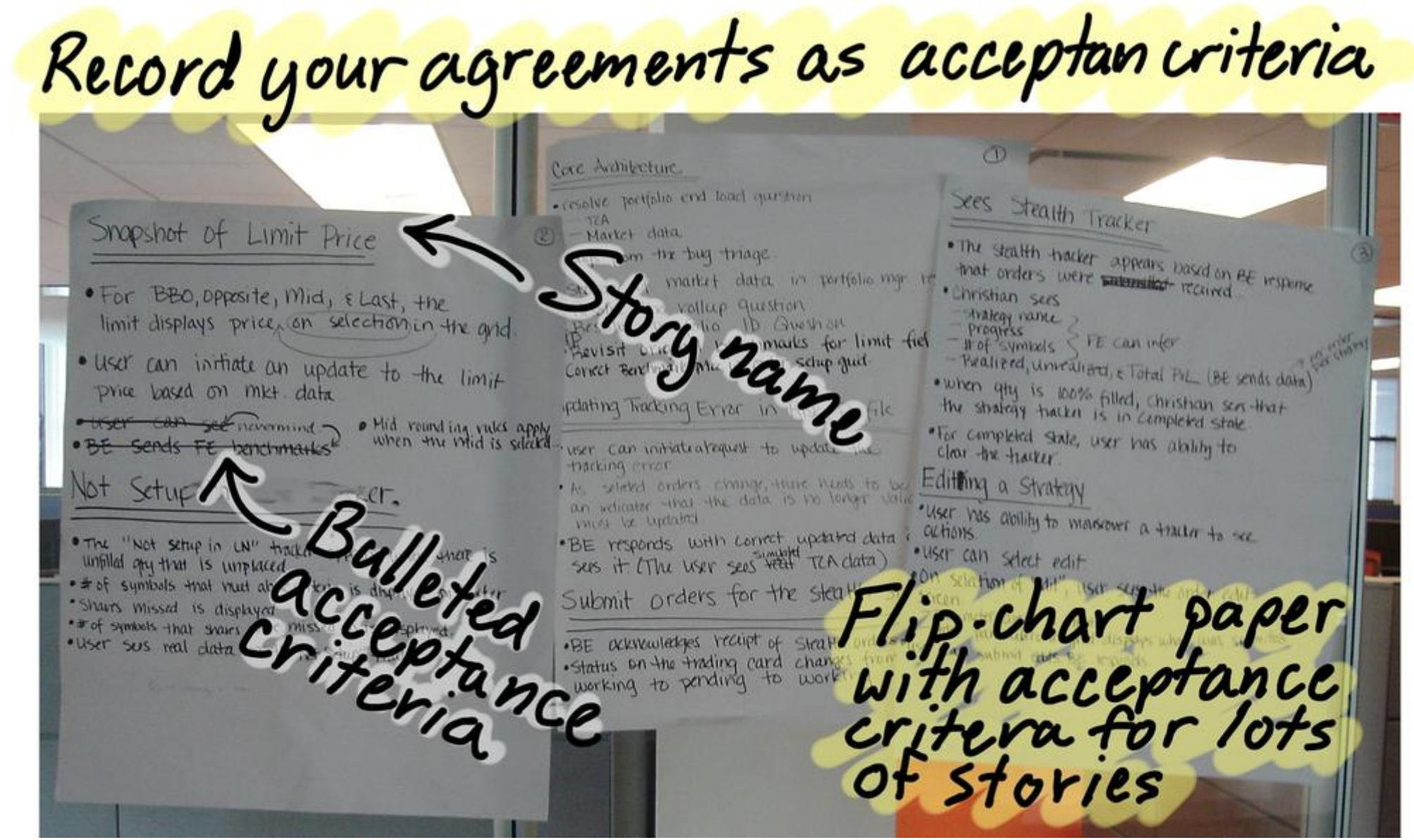
conversation. Use a whiteboard or flipchart paper. And don’t forget to take some “vacation photos” before you leave. Photos of what you’ve created will help you recall all the details of the conversation that would be difficult to write down.

Use words and pictures



sketch,
tell stories,
write down
facts & decisions

Good story conversations have lots of words and pictures.



Keep the acceptance criteria you decide on big and visible during the conversation. This team uses flipchart paper to record as they go.

That's It

That’s all there is. That’s Kent’s disruptively simple idea. And, if you do it, I promise you it’ll change everything.

Unless it doesn’t.

For people used to working the old-fashioned way, this conversation can go really badly. They fall back into their old patterns of the person bringing in stuff, trying to work hard to communicate exactly what he wants to the other person, and the other person working hard to try to understand and then punch holes in what the other person is saying. If this goes on too long, they often feel like punching holes in each other, which is not conducive to getting things done.

There are some things to keep in mind to help these conversations go better. Happily, that’s what the next chapter is about.

---

[9] There’s some evidence that a fact wrapped in a story is much more memorable than the fact presented alone—22 times more memorable, according to psychologist Jerome Bruner.