CHAPTER 7

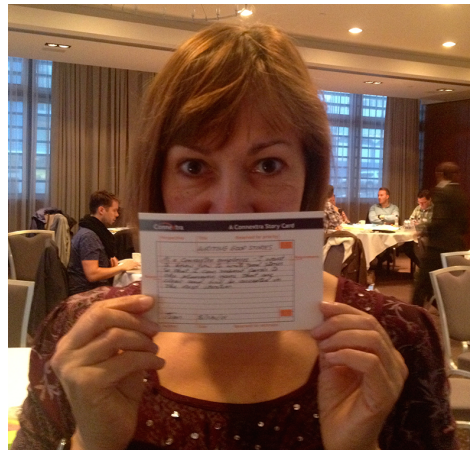
Telling Better Stories

The idea of stories is simple. Maybe too simple. For lots of people in software development, conversations like these feel very foreign…and a little uncomfortable. And people often revert to talking about requirements like they always used to.

When Kent Beck originally described the idea of stories, he didn't call them *user stories*, he just called them *stories*—because that's what he hoped you'd be telling. But very soon after the first books on Extreme Programming were published, *stories* picked up the more descriptive prefix *user* so that we'd remember to have conversations from the perspective of the people outside the software. Changing the name wasn't enough, however.

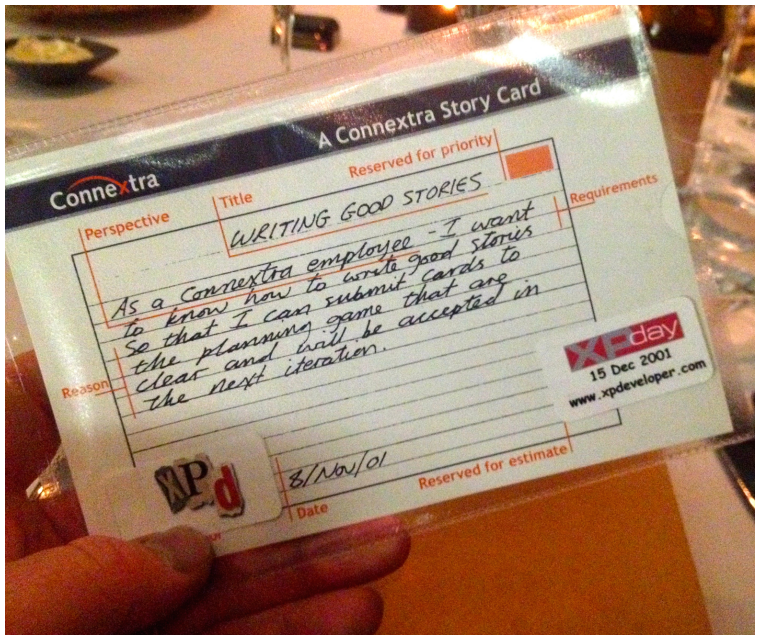
Connextra's Cool Template

This is my friend Rachel Davies. And she's holding a story card.



In the late 1990s she worked for a company called Connextra. Contextra was one of the earliest adopters of Extreme Programming, the Agile process where stories came from. When the Connextra folks started using stories, they found they ran into some common problems. Most of the people who wrote the stories at Connextra were from sales and marketing. They tended to write down the feature they needed. But, when it came time for developers to have conversations, they needed to find that original stakeholder and get a good conversation started—one that included who and why. Just having the name of the feature wasn't helping the team find the right people to talk to, or to start the right discussions. And, back then, there wasn't much guidance about what could or should be on a card. The template evolved over time at Connextra. It wasn't Rachel's invention specifically. Rather, it was the entire organization's desire to build things that mattered.

After using the template for a while, the folks at Connextra wanted to show off their cool new trick. They printed a bunch of example cards to show off at XPDay 2001, a small conference in London. That's what Rachel is holding. It's her last card, and might very well be the last card in existence, so that makes it a historical artifact.



The template goes like this:

> As a [type of user]
>
> I want to [do something]
>
> So that I can [get some benefit]

The folks at Connextra used this simple template to write the *descriptions* of their stories. Note that their stories still have short, useful titles. They found that writing this little extra bit to start to tell the story forced them to pause and think about: who, what, and why. And, if they didn't know, they might question whether they should be writing the story at all.

When they sat down to have the story conversation, they'd pick up that card and then read that description. That description started the conversation.

Use the simple story template to start conversations.

If you're picking up an individual card outside of a story map, the template is a good way to boot up the conversation. Remember in Chapter 1 that one of the cards in the body of Gary's map said, "Upload

an image." And that card was one of the details under the card "Customize my promo flyer." What I write on the cards in a map are those short verb phrases—the tasks the users are performing with my software. But when I pick one up by itself, I'll need to pick up the story in the middle of the user's big journey described in the map. I might say:

> "As a *band manager*, I want to *upload an image* so that I can *customize my promo flyer*."

That's a pretty cool trick. I can find the cards for different users hanging above the map, and the users' bigger goal is often in the card above the one you're viewing at any one time.

But remember—it's just a conversation starter. And, in fact, the conversation might continue like this:

> "Why would the band manager want to customize the flyer?"
>
> "Well, because it won't have his band's photo on it automatically, and he'd want it to be there. And he cares a lot about being original—he wouldn't want it to look like everyone else's."
>
> "That makes sense. Where do people like band managers keep those photos?"
>
> "Well, they're all over the place, really. They may be on their local hard drives, in Flickr accounts, or in other places on the Web."
>
> "Hmm…that's different than I was originally thinking. I assumed they'd just be on their hard drives."
>
> "No, lots of the people we've talked to have them spread all over the place. It's kind of a problem."

In fact, a lot of the conversation I had with Gary went like that. As we were talking, we'd write something down on a whiteboard or on the card itself. Before too long we'd be sketching ideas.

You can see that the very short template isn't nearly enough to be the specification. But when we start a conversation using that template, we end up in a much richer conversation than if we'd just talked about a file uploader.

Rediscovering the Small Conversation

Mat Cropper, ThoughtWorks

I was working as a business analyst on a ThoughtWorks project for a UK government agency. We were responsible for delivery, but also

for giving the client team some practical experience around Agile methodologies. That being the case, we were quite a big team of some 25 or so technologists and businesspeople. One room, 25 different air conditioning setting preferences—you get the idea!

To start off, the product owner and I would write stories, and then at the beginning of every two-week iteration, the entire team got together for planning. It was a necessarily big meeting, and it was a car crash. "Why are we doing it this way?" "These stories are way too big/small." "It doesn't make sense." "I have this strong preference for a particular technical implementation." These were common (and frustrating) discussions. To be honest, I left those meetings feeling pretty dejected. It felt like a personal failure.

Something had to be done to fix this, so we decided that rather than have one meeting with everybody to discuss everything, we would move to conversations with a tighter focus. Backlog grooming, for example, took place in week one of the iteration with a small group (project owner, project manager, business analyst, technical architect) who kicked the tires of the various stories so that when we did a grooming session as a full team later there would be much fewer distractions. The conversation was about tweaking and improving our stories, and we ignored things like prioritization, story points, and so forth. It worked.

We also made sure that we were much more constructively creating stories. I'd have an idea of the stories I was working on that week, and they'd be up on the card wall in the "In Analysis" column. Each day in our team's standup, I'd call out that I was working on a particular story, and could use some time from a developer pair to pull it together. We'd sit down, discuss what we were aiming for, usually touch on the technical aspects, and then get it all down on paper. We ignored Trello, which we were using for our digital card wall at the time, and focused instead on the face-to-face conversations, sometimes standing at a whiteboard. Working as a group, down in the detail, is actually pretty rewarding, and as it took only about 20 minutes each time, it wasn't too much of an overhead either. People were genuinely happy to pair and contribute.

As a happy consequence, our large backlog grooming sessions were a breeze, and we also found that story sizes were becoming more and more uniform. Iteration planning became less of a pain as a result. The quality of the conversations leading to the documenting of a story had improved, and so had the work we were producing.

Template Zombies and the Snowplow

The term *template zombies* comes from the book *Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior* by Tom DeMarco et al. (Dorset House). The name says it all, but I'll give you the authors' definition:

> **Template Zombie:**
>
> The project team allows its work to be driven by templates instead of by the thought process necessary to deliver products.

As simple as that template is, it gets abused quite a bit. I see people really struggle to force ideas into the template when they just don't fit. Stories about backend services or security issues can be challenging. I see people writing and thinking about things from their own perspective, not that of the people who ultimately benefit: "as a product owner, I want you to build a file uploader so that the customer requirements are met." Nasty things like that.

Even worse, the template has become so ubiquitous, and so commonly taught, that there are those who believe that it's not a story if it's not written in that form. Many people have even quit using titles on stories and write only that long sentence on every single card. Imagine trying to read through a list of stories written that way. Imagine trying to tell someone a story using a story map where every sticky is written that way. It's tough on the brain.

All of this makes me sad. Because the real value of stories isn't what's written down on the card. It comes from what we learn when we tell the story.

It doesn't need to be written in a template to be considered a story.

The person in this picture is learning to ski.[1] If you've ever learned to ski, and someone helpful is teaching you, you'll do what this person is doing. It's called a snowplow. It's where you put the tips of your skis together and lean on the inside edges of your skis. It's the easiest way to control your speed and stay upright when you've got two slippery boards clamped to your feet. It's the way I'd recommend learning to ski. But it's not best practice—it's a best *learning* practice. There's no Olympic snowplow event. You won't impress anyone on the slopes with your cool snowplow stance. It's nothing to be embarrassed of, though. If people see you skiing that way, they'll know you're learning.

For me, the story template works a bit like learning the snowplow. Use it to write the descriptions of your first stories. Say it aloud to start your story conversations. But don't get too concerned if you find that

1. This photo was taken by Ruth Hartnup, found on Flickr, and licensed under the Creative Commons Attribution License.

it doesn't always work. Just like the snowplow technique for skiers, it's not the best choice for difficult terrain.

My favorite template: if I'm writing stories on sticky notes or cards, and they won't be sitting inside a bigger story map, I'll first give them a short, simple title, and then under it I'll write:

Who:

What:

Why:

And I'll give a couple of lines in between each, because I'll want to specifically name all the different whos, say a little about the what, and make notes about the different reasons why. I'll want to leave room on the card to add extra information when we start talking about the stories. It's actually a pet peeve of mine when people write the title in the middle of the card, because it doesn't leave me any room to make notes when we start talking. But I'm nitpicky that way.

A Checklist of What to Really Talk About

- ▢ *Really talk about who*
  Please don't just talk about "the user." Be specific. Talk about which user you mean. For Gary, he could talk about the band manager or the music fan.

  Talk about different types of users. For many pieces of software, especially consumer software, there are very diverse types of users using the same functionality. Talk about the functionality from different users' perspectives.

  Talk about the customers. For consumer products, the customer (or chooser) may be the same person as the user. But for enterprise products, we'll need to talk about the people who make buying decisions, their organization as a whole, and how they benefit.

  Talk about other stakeholders. Talk about the people sponsoring the software's purchase. Talk about others who might collaborate with users.

  There's rarely just one user who matters.

- ▢ *Really talk about what*
  I like my stories to start with user tasks—the things people want to do with my software. But what about services like the kind way

beneath the user interface that authorizes your credit card for a purchase, or authenticates you on an insurance website? Your users didn't make a deliberate choice to get their credit cards pass: [authorized] or have their credentials verified. It's OK to talk about the services and the different systems that call them. It's OK to talk about specific UI components and how the screen behaves. Just don't lose sight of who cares, and why.

- *Really talk about why*
  Talk about why the specific user cares. And dig deeply into the "whys," because there are often a few, and they're layered. You can keep "poking it with the why stick" for a long time to really get at the underlying reasons why.

  Talk about why other users care. Talk about why the user's company cares. Talk about why business stakeholders care. There are lots of great things hidden inside why.

- *Talk about what's going on outside the software*
  Talk about where people using your product are when they use it. Talk about when they'd use the product, and how often. Talk about who else is there when they do. All those things give clues about what a good solution might be.

- *Talk about what goes wrong*
  What happens when things go wrong? What happens when the system is down? How else could users accomplish this? How do they meet their needs today?

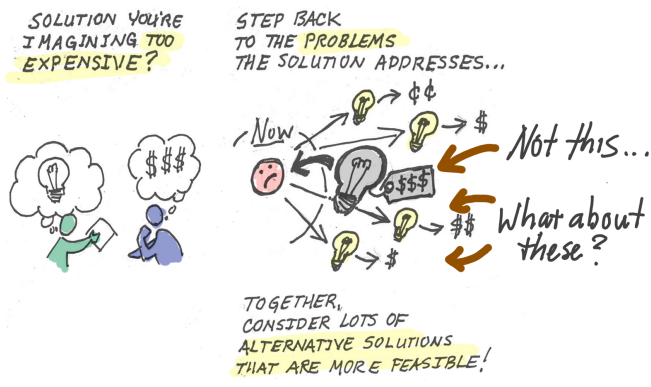
- *Talk about questions and assumptions*
  If you talk about all those things, you've likely stumbled across something you don't know. Identify your questions and discuss how important they are to get answered before you build software. Decide who'll do the legwork to get those questions answered, and bring them back to your next conversation. You'll find it takes lots of conversations to think through some stories.

  Take time to question your assumptions. Do you really understand your users? Is this really what they want? Do they really have these problems? Will they really use this solution?

  Question your technical assumptions. What underlying systems do we rely on? Do they really work the way we think? Are there technical risks we need to consider?

All these questions and assumptions may require deliberate work to resolve or learn. Make a plan to do just that.

- ▢ *Talk about better solutions*
  The really big win comes when those in a story conversation discard some original assumptions about what the solution should be, go back to the problem they're trying to solve, and then together arrive at a solution that's more effective and more economical to build.



- ▢ *Talk about how*
  When sitting in a story conversation, I often hear someone anxiously say, "We should be talking about the what, not the how!" By that they mean we should be talking about what users need to do, not how the code should be written. And I feel the same anxiousness when we talk about the "what" without talking about the "why." But the truth is, we're trying to optimize for all three in a good story conversation. What goes wrong is when either party assumes that a particular solution or the way it's implemented is a "requirement." Without explicitly talking about how (and if you're a developer, I know you're thinking about it), it's difficult to think about the cost of the solution. Because, if a solution is too expensive, then it may not be a good option.

  Be respectful of the expertise of others in the conversation. Don't tell a highly trained technical person how to do her work. Don't tell someone intimately familiar with users and their work that he doesn't understand. Ask questions, and genuinely try to learn from each other.

- *Talk about how long*
  Ultimately, we need to make some decisions to go forward with building something or not. And it's tough to make this sort of buying decision without a price tag.

  For software, that usually means how long it'll take to write the code. In early conversations, that might be expressed as "a really long time" or "a few days." Even better is comparing it to something already built—"about the same as that feature for commenting we built last month." As we get closer to building something, and we've had more conversations and made more decisions, we'll be able to be a bit more precise. But we always know we're talking about estimates here, not commitments.

Create Vacation Photos

Because there's a lot to talk about, and you won't want to forget it, make sure you're recording specific things that help you remember the decisions you made, or the questions and assumptions you'll need to look into. Don't forget to externalize your thinking so that others in the discussion see what's recorded.

If you write it down, you can pick it up and refer to it later. If it's posted on the wall, you can just point at it. And, if you're talking together as a team, you'll find you won't have to repeat everything so often, because people will remember—especially if you anchored your conversations with simple drawings and documents…those vacation photos.

This small group is having a story discussion. As they talk, they visualize their ideas and make notes about what they decide.

My favorite approach is to do exactly what they're doing. I record on flipchart paper or a whiteboard as we talk. I like making a note of who was in the conversation directly on the board, and then photographing the board when I'm done. I'll share the photo using a wiki or other tool. I know I can extract details or write them up more formally when I need them later. If I can't remember exactly what was said, one of those people in the conversation might. It's a good thing I wrote their names down.

It's a Lot to Worry About

It's daunting to think about how much there is you could be talking about in stories. At this point, you may want to go back to the good old days when all you needed to do was worry about understanding the "requirements." Back when it wasn't your job to really solve problems. Back when you just needed to build what you were told to, and it was someone else's problem to make sure it was the right thing to build. But I believe that you, and most people out there, really like solving problems. So now's your chance.

It may have occurred to you that with all these things to talk about, there will be a lot of information to keep track of. And all that stuff isn't going to fit on a sticky note or an index card. You're right. It won't. So let's talk next about what really does go on that card, and what doesn't.

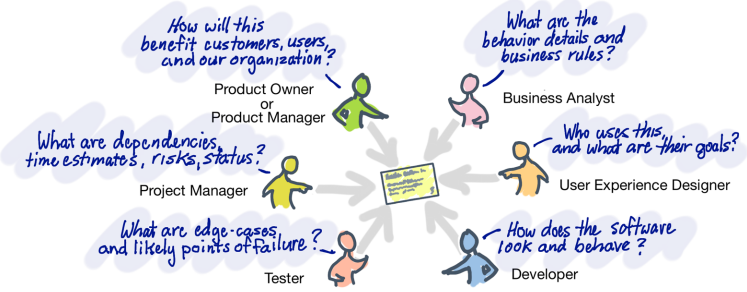
CHAPTER 8

It's Not All on the Card

Yes, the big idea was that short story titles on cards would help us plan and facilitate lots of conversations between the people who could build software and the people who understood the problems that needed to be solved with it. But, sadly, it takes more than a couple of people to get a finished piece of software out the door.

On a typical team you'll find project managers, product managers, business analysts, testers, user experience designers, technical writers, and some other roles I'm probably forgetting. They're all looking at the same cards, but the conversations they have are going to be different because they've all got different concerns to look after.

Different People, Different Conversations



If I'm a product manager or product owner, and I'm responsible for the success of this product, then I have to know a little more about my target market. I need to form some hypothesis about how many people

will buy or use this product, or how it's going to affect the profitability of my company. I'll want to talk about those things.

If I'm a business analyst, I might be diving into a lot of details, so I need to understand what's going on in the user interface, and the business rules in the system that are behind the user interface.

If I'm a tester, I need to think about where the software is likely to fail. I need to have some conversations to help me put together a good test plan.

If I'm a UI designer, I don't want to be told what the UI looks like any more than a developer wants to be told the way the code should be written. I'll want to know who's using it, and why and what they're doing, so I can design a useful and usable user interface.

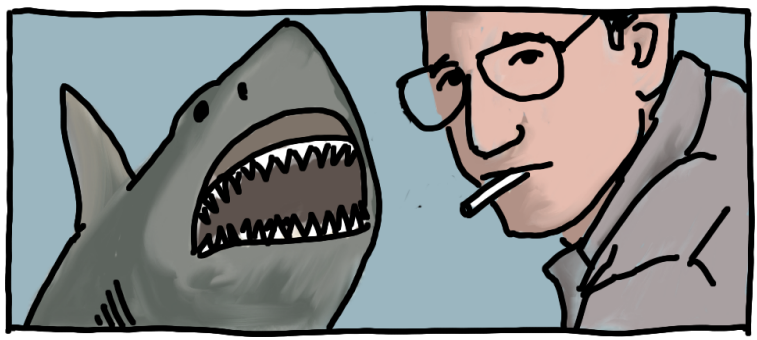
Finally, if I'm a project manager responsible for coordinating this group of people, I'll have to pay attention when all of them are talking to make decisions about all these details. I'll need to pay attention to dependencies, schedules, and the status of development when it gets started.

That's a lot of conversations. And some of them have to happen before others. And many of them happen more than once. So, to make it accurate, we'd probably have to add another dozen *C*s to the 3 *C*s. But happily, if you're really having conversations and building shared understanding along the way, you'll avoid lots of misunderstandings and course corrections.

There are many different kinds of conversations with different people for every story.

We're Gonna Need a Bigger Card

When you say the words in that heading, I hope you're thinking of the old movie *Jaws* when, after seeing the huge shark up close and personal for the first time, Police Chief Brody says to Quint, "You're gonna need a bigger boat."



You see, the original idea was also that I could pick up a card and write the title on the front, and then as we had conversations, I could flip it over on the back and write the details of all the things we agreed to that came up. I could sketch the user interface and write a lot of other information on the card. On some projects, it can really work this way. It's cool when it can, and it's usually a side effect of small teams working closely together with a lot of tacit knowledge. Those are the teams that don't have to write much to remember.

But I don't think even Kent and the folks who perfected the concept of stories actually thought that all these conversations between all these different people could be contained on just a single card, and in fact, they usually aren't.

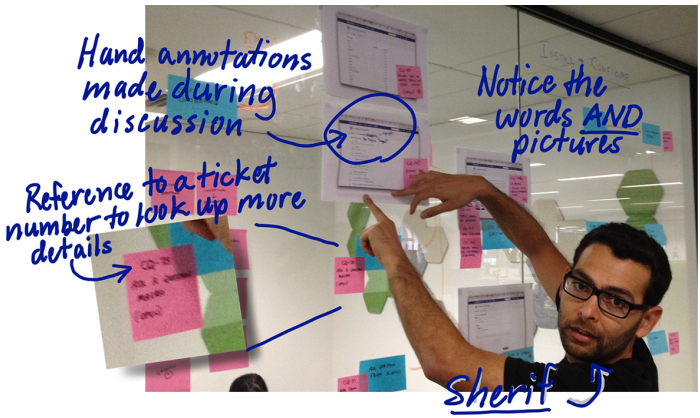
The metaphor that works for me is a card in a library card catalog, for people who are old enough to remember when libraries actually *had* card catalogs. Stories written on cards work a bit like those.

If I pick up a card from a card catalog, it's going to have just enough information for me to identify the book. It likely has a title, the author name, a description, the page count of the book, the category of the book—like "nonfiction"—and a code (remember Mr. Dewey's decimals?) to a location where I can actually find a copy of the book in the library. The card's just a token that's easy to find and organize. No one confuses the cards with a book. The card catalog is handy because it takes a lot less space than thousands of books would. And I can organize cards in different ways—by author, for example, or by subject.

Your stories will work the same way; that is, you may write them on cards, keep them in a list in a spreadsheet, enter them into your favorite tracking tool, or enter them in the tracking tool your company makes you use—you know, the one everyone grumbles about. In a library, you know there's a book out there somewhere, and if you have identified the right card filed away in the card catalog, it's easy to find it. Similarly, with a story, you know there's a growing amount of information out there somewhere. It grows and evolves with each conversation. And, hopefully, however your company chooses to keep track of the information, it's easy to find, too.

If you want to go really old school, keep the details of all those discussions taped onto big sheets of flipchart paper on the wall so you can keep talking about them whenever you want. But remember: you'll want to take them down when you've completed the work or else you'll run out of wall space. And you'll want to photograph them and keep those photos somewhere for posterity.

How Tool Creators Have Good Story Discussions



This is my friend Sherif, who's a product manager at a company called Atlassian. Atlassian makes Confluence, a popular wiki used in a huge variety of organizations to keep track of the knowledge those organizations accumulate, among other things. They also make JIRA, one of the more popular tools for managing work in Agile development. You'd think a company that focuses on building tools used for keeping and sharing information electronically would use its own tools— "eat their own dog food," so to speak—and you'd be right that Atlassian does. But it also understands how to have good face-to-face conversations.

When I walk around the Atlassian office in Sydney, I see the walls covered with sticky notes, whiteboard drawings, and screen wireframes. If you look closely, you'll see that the sticky notes reference ticket numbers in those tools the team relies on. They nimbly move back and forth from tools to physical space. When Sherif shows me what they keep in Confluence, I'm amazed at the combination of photographs, short videos, and back-and-forth discussions.

Radiators and Ice Boxes

In his book *Agile Software Development: The Cooperative Game* (Addison-Wesley Professional), Alistair Cockburn coined the term *information radiator* to describe how big, visible information on the

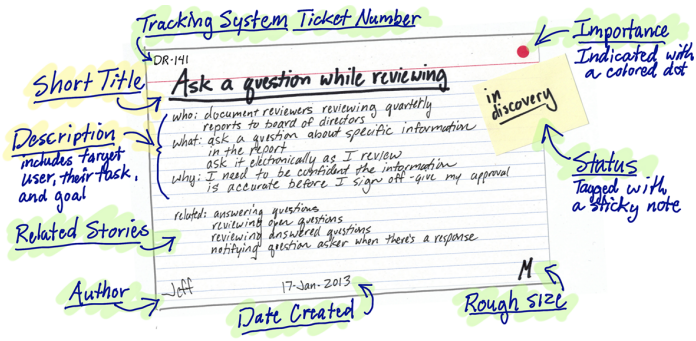
wall radiates useful stuff into the room. People walking by look at it and engage with it. When the information is alive and useful, lots of conversations end up at the wall, where people can point to and add to the information accumulating there.

When I walk into environments where the walls are clear, or even covered with pleasant artwork—or worst of all, motivational posters—it makes me sad. There's so much great collaboration that could be happening every day with those useful walls. If what's on the wall is an information radiator, some refer to the tools people use as an *information icebox*—because that's where information goes to be preserved—and potentially be crusted over with that thin layer of ice like that stuff in the back of your freezer. (I'm always shocked by what I find back there.)

That's what's really remarkable to me about Atlassian. They keep information alive and useful, both in and out of the tools they use.

What's Really on a Story Card?

Imagine a card from a library's card catalog. That card's got useful information on it to help you organize it and confirm you're talking about the right book. A good story card is a bit like that.



Common things you'd expect to find on a card are:

Short title
: One that's easy to insert into a conversation when you're talking about it. A good title is the most valuable part of your story. Don't be afraid to rewrite it if it's confusing people.

Description
: A sentence or two that describes what we're imagining. It's a good idea to describe who, what, and why—who uses or needs it, what they'll do with it, and what benefit they hope to get from it.

As you begin to discuss stories, you'll add information that summarizes some of your discussions. That'll include stuff like:

Story number
: When you get a bunch of these or put them into a tracking system, this will help you find them—sort of like the Dewey Decimal System in a library. But, whatever you do, *please* don't start referring to your stories by their number. If you do, it's a sure signal you haven't chosen a very good title. And even librarians don't refer to books by their Dewey Decimal numbers.

Estimate, size, or budget
: As you begin to discuss the story, you'll want a prediction on how long it might take to build the software. There are lots of terms for this, like *estimate*, *size*, or *budget*. Use the term your company uses.

Value
: You might have lengthy discussions about the relative value of one thing over another. Some might use a numeric scale. Some might annotate cards with *high*, *medium*, or *low*.

Metrics
: If you really care about results, identify specific metrics you'll track after the software is released to determine whether the software was successful.

Dependencies
: Other stories that this one might depend on or go with.

Status
: Is it planned for a particular release? Is it started? In progress? Done?

Dates
: Just as a book's card has the date it was published, you might keep the date this story was added, started, and finished.

You could scribble any other notes you like on the card. Or flip it to the back and write notes or bulleted acceptance criteria.

The only thing that's required on your card is a good title. All those other bits of information could be helpful, but you and your team get to decide which you'd like to use.

Not too much fits on the card, and that's good. Remember, it's just a token you'll use to plan with. You could use cards, or sticky notes, too. Having the physical cards lets you use handy words like *this* and *that* in conversations as you point to cards on a wall or tabletop. You can't do that with a thick document. With cards, you can shuffle them around a desk, rank them by importance, tape them to the wall, and wave them around while you're talking to make your point more emphatically. If you were doing that with a thick document, you could hurt someone—perhaps yourself. And, of course, you'll want to arrange bunches of cards into story maps to tell even bigger stories.

That's Not What That Tool Is For

A lumberjack comes across a man in the forest. The man is working hard trying to chop down a tree by hitting it with a hammer. The lumberjack stops the man and says, "Hey, you're using the wrong tool! Try this…" and hands the man a saw. The man thanks him, and the lumberjack continues on his way, happily knowing that he's helped. The man then begins striking the tree with the saw the same way he was with the hammer.

This joke reminds me that we can use the *wrong tool* for the job, and we can also use the *tool wrong*.

When I tell people how companies like Atlassian use tools, they're usually surprised. They're often surprised because they've been trying to use tools as a replacement for whiteboards and sticky notes. And, predictably, they've been struggling with that. It may be that they're using the wrong tool for the wrong job, or using the right tool wrong. To figure out what might be going wrong, it's best to look at the job first, and not the tool.

Building Shared Understanding

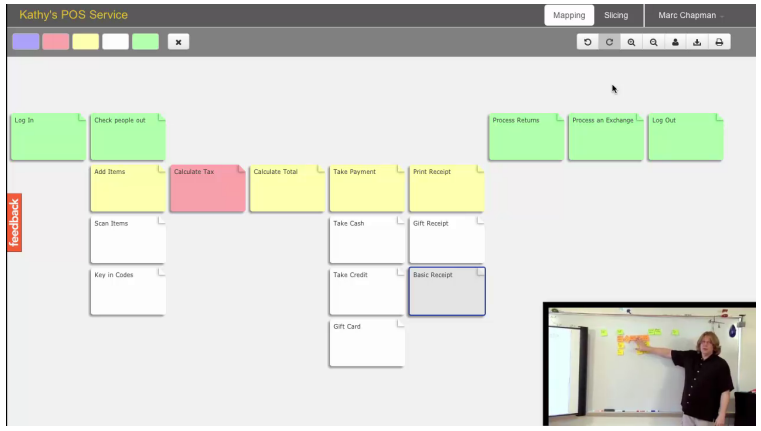
When we're working together to tell stories and make decisions about solutions to build, our first goal is to build shared understanding. This is where externalizing and organizing your thoughts is critical. And nothing beats face-to-face work in front of a whiteboard, armed with sticky notes. But, if you've got to accomplish this task with others

working remotely, this is tough. Video conference tools that let you see one another's faces don't help much, since it's not their faces you need to see—it's the ideas you'll be placing on the wall or tabletop.

Use a document camera or web camera during a video conference to let remote people see what's being created on the wall.

I've worked with teams that have video cameras at both ends of a video conference, and the call focuses on the growing models on the wall, not the team members' faces.

If you use a tool to visualize, it's ideal if people on both sides of the conversation can add and move things around, just like they could if they were working together at a whiteboard. This is a screen from a tool called Cardboard.



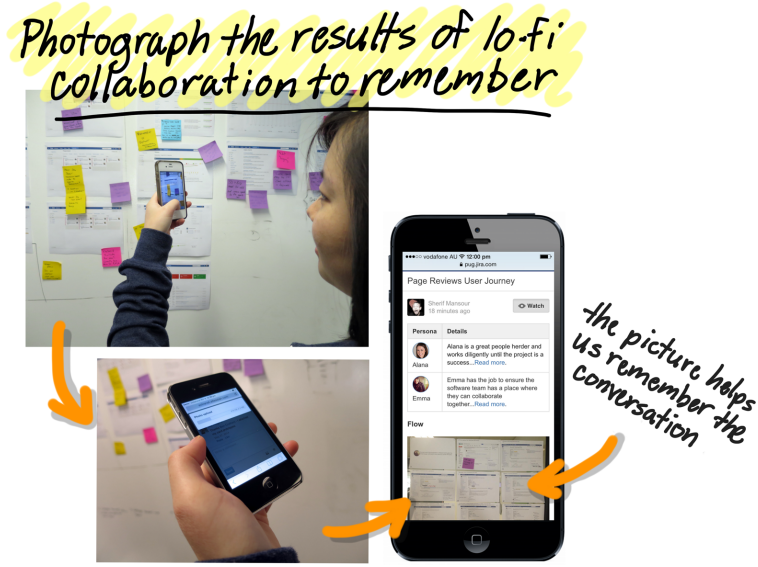
The person using Cardboard is creating a map at the same time as David Hussman, one of Cardboard's creators, maps on a wall. Others who are sharing the same map from other locations see it come together in real time. They can add, remove, and change cards, and everyone can see what everyone else is doing. You can virtually "step back" and look at the entire wall at once, which is handy because computer screens are just a tiny portal on what you could see if you were working at a wall.

When collaborating remotely, use tools that allow everyone to see, add to, and organize the model concurrently.

Happily, I'm seeing a lot more tools enter the market that understand and support working together to build shared understanding. This is a good thing.

Remembering

When we've worked hard together to get on the same page, we should be keeping copies of whatever models or examples we've created to use as vacation photos—to help us remember all the details we've discussed. Tools like Atlassian's Confluence offer a rich wiki for storing not just words, but also pictures and video. Taking and keeping pictures and videos after working together is one of fastest ways to document.



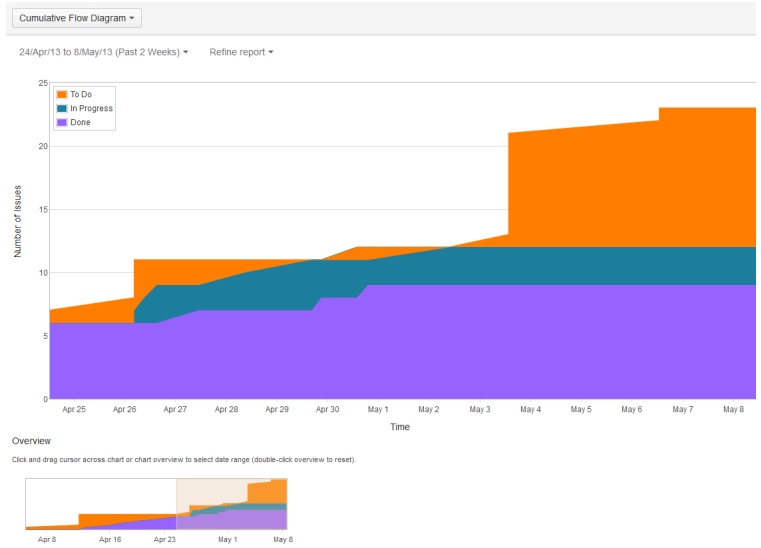
These folks at Atlassian are doing just that. They've snapped a picture after working at a wall, and uploaded it to their wiki for safekeeping.

Use tools to post pictures, videos, and text to help you retain and remember your conversations.

I personally like staying lo-fi and keeping information on the wall, but if I worry at all about a cleaning person taking it down at night, I'll photograph and keep it just in case. If I'm sharing information with people who couldn't be in the room, even virtually, I'll shoot a short video stepping through the model on the wall and post that where others can see it.

Tracking

One of the things that tools most excel at is taking all the work we've planned on doing, and letting us track its progress. Tools are great at keeping track of the numbers that are tedious for us—things like exactly when we started, when we finished, and how much we've got left to do. The better tools will generate useful insights for us as we routinely track what we're doing.



This is a cumulative flow diagram generated by Atlassian's JIRA product. It's a chart that shows the work we're doing and its state over time. And it's a chart I would hate to produce by hand.

For single, collocated teams and small projects, the wall will do just fine. But, if you've got larger teams working from different physical locations, and longer-running projects, use a tool to keep track of all the details.

Use tools to sequence, track, and analyze progress.

The trick is using the right tool for the right job. Don't try to use a really great tracking tool to build shared understanding. And don't struggle to do complex analysis on a whiteboard.

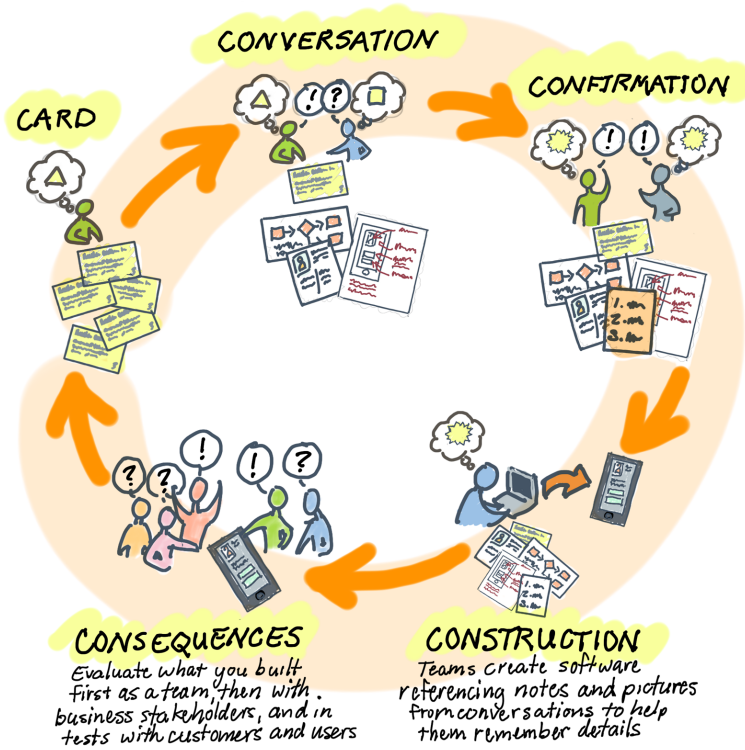
It's been an Agile ideal to keep things simple and fast, to stay working on index cards and whiteboards as much as possible. And I promise you if you can stay small and fast and avoid unnecessary tools, you'll be happier. But remember: those tools are just a means to an end. Next, we'll need to talk about what happens after the card.

CHAPTER 9
The Card Is Just the Beginning

The three *C* s are just the beginning.

I know I said earlier that I don't feel obligated to quote the Agile Manifesto, but I'm going to anyway—well, at least a small part of it. One of the value statements in the manifesto reads "Working software over comprehensive documentation." I could rephrase that as "working software over comprehensive conversation," and the meaning would be the same. All those conversations—and the documentation that helps us recall them—are just a means to an end. Eventually we'll need to build something.

If we finish the cycle, the model looks like this:



There are some gotchas that always manage to sneak in here after we've got shared understanding and agreement on what we'll build. Keep an eye out for them.

Construct with a Clear Picture in Your Head

After having conversations, and writing down details that'll help us remember those conversations, and writing down confirmation—that is, our agreement about the things we'll check to confirm we're done—we're finally ready to make something:

- Software developers can get started building the software.
- Testers can create test plans and test.

- UI designers can create detailed UI design and digital assets, if they haven't already done so as part of arriving at a shared understanding.
- Technical writers can write or update help files or other documents.

The most important thing here is that all these people are armed with *the same picture in their heads: the picture they built while talking together*.

I'm going to pause here for effect.

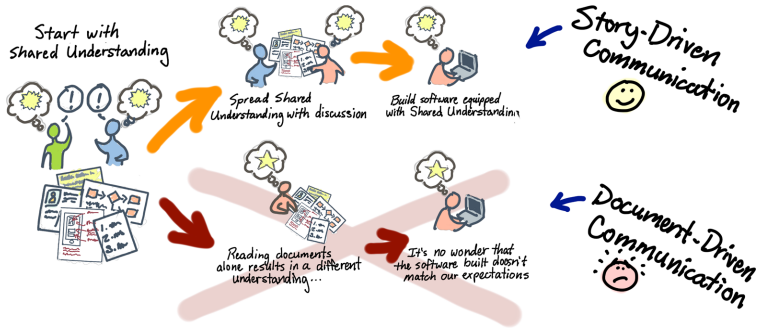
Now, I'm going to say this next part slowly, so you should read it slowly.

Handing off all the details about the story to someone else to build doesn't work.
Don't do that.

If you and a group of people have worked together to understand what should be built, and if you've documented all the important things someone needs to know to build it, you may be very tempted here to hand it off to someone else. After all, when you look at the information, it's crystal clear to you. But don't fool yourself. It's clear to you because your really smart brain is filling in all the details that aren't written down. Your brain is so good at it that it's hard for you to detect what could be missing. Remember, those details are *your* vacation photos, not *theirs*.

Build an Oral Tradition of Storytelling

Sharing stories is reasonably simple. Someone who does understand the story, and the information collected that helps tell the story, needs only spend a little time retelling the story to the next person who needs to learn. Now, this should go lots faster than early conversations where you worked together to make tough decisions, since hopefully you won't need to remake them. Use what's written to help tell the story. Talk and point to pictures. Let your listener ask questions and make changes to the pictures that help her remember. Help her turn the information associated with the story into her own vacation photos.



There's a nasty anti-pattern I often see here. Some think that since anyone in a team might pick up the story and do work on it, everyone on the team should be involved in every conversation. Perhaps you work at this company. You'll know it because you'll hear lots of people complaining that there are way too many meetings. By the way, "meeting" is often the euphemism we use for unproductive collaboration.

Effective discussion and decision making goes best with small groups of two to five. It's dinner conversation sized. You know that if you have a group of friends sharing a meal, it's easy to hold a single conversation when there are just a few of you. But any more than about five people is where it becomes a real effort.

Let small groups work together to make decisions, and then use continued conversations to share the results with everyone else.

Inspect the Results of Your Work

If you're on the team, you'll all work together armed with shared understanding about what you're building, and why. As you work together, you'll keep having conversations because you never think of everything. But when the software is done, you'll all come back together and talk about it.

This is a good time to congratulate yourselves on a job well done. It's pretty cool to see real progress. In traditional software development, the opportunities to see the results of your hard work can come a lot less frequently, and they're rarely shared as a team. In a typical Agile process like Scrum, you'll be sharing every couple of weeks at an end-of-sprint product review. In the healthiest of teams, team members get together frequently to inspect their work as it's done. But you'll need

to go beyond show and tell. After congratulating yourselves, take time for a short but serious reflection on the quality of the work you did.

When talking about quality, I start with discussions of these three aspects:

User experience quality
: Review the work from the perspective of its target user. Is it straightforward to use? Is it *fun* to use? Does it look good? Is it consistent with your brand and other functionality?

Functional quality
: Does the software do what you agreed it would without bugs or errors? Testers and other team members have hopefully spent time testing and you've fixed any bugs already. But good testers can often tell you that there are likely more bugs lurking in your product that may emerge later. Or hopefully they can say that it feels rock solid.

Code quality
: Is the software we wrote high quality? Consistent with our standards? We may own this stuff for a while, so it'd be good to know if we think it'll be easy to extend and maintain, or if we've just added a pile of technical debt we'll need to address later.

I have some bad news for you here. You're likely to find things you believe should change about what you've done.

It'll help everyone's sanity to separate out two concerns. First: *did we build what we agreed to build?* And then: *if it's what we agreed to build, now that we see it, should we make some changes?*

Everyone will have worked hard together at the outset to figure out what to build that would solve users' problems and be economical to build. You'll have done your best to identify the things to check to confirm that it's done. Check all those things, and congratulate yourselves if you've accomplished that much. You got exactly what you agreed you wanted to get.

Now, here's where some old Rolling Stones song lyrics play in my head. If you know the song, hum along: "You can't always get what you want. But, if you try sometime, you just might find, you get what you need." The irony with software is that it's exactly the opposite.

You'll work together to agree on what you want. And, if you're working with a competent team, you'll see that you can get pretty good at getting

it. It's only after seeing it, though, that you can better evaluate if it's what you need. This sucks. But don't blame yourselves—that's just the way it works.

You do, however, have a way of fixing it. And it starts by writing a card with your ideas about what to change in the software to fix it. This, of course, sucks if you'd planned on being right the first time. Maybe Mick Jagger was right after all. Maybe what you really needed was to learn that being right the first time is a risky strategy—especially in software.

It's Not for You

I'm sorry. I've got even more bad news.

In reality, the person who originally wrote the card and who started this entire cycle is likely not the person who will use the software every day. The person who originally wrote the card, and the entire team who worked together, may believe they've nailed it—that they've built the perfect solution to the challenges their target users have.

Don't fool yourself.

If we're on this team together and we're smart, we'll take the software out to users and test it with them. This isn't show and tell, either. We'll test by watching them use the software to reach a real goal they'll normally need to accomplish using the software.

Have you ever sat with someone as she uses software you've helped build? Think back to the first time you did. How did it go? I wasn't in the room with you at the time, but I'm willing to bet that it didn't go the way you expected.

If you've ever sat next to a user as she uses your product, you know what I mean. If you've never done this, then do it.

You'll need to test with the people who'll actually buy, adopt, and use your product at some regular frequency. I often wait until I've built up a bit of software—enough that they could use it to accomplish something they couldn't before. Whatever frequency you adopt, don't let more than a couple weeks go by without seeing a genuine user interact with the software.

Everyone on the team doesn't need to be there with users. In fact, it'll kind of creep the users out if everyone is. But being there builds

empathy that you won't get any other way. It's a powerful motivator to see people struggling to use your product, especially when you were so confident they wouldn't need to. If you were there, share what you saw with others by telling stories back to them.

After testing with users, you'll identify problems to fix and obvious ways to improve the software. And for each one of those things, you should write a story card with your ideas for improving the software.

Build to Learn

If you'd labored under the belief that using stories would stop your team from writing bad software, you were at least half right. In fact, all the conversations between smart people focused on understanding the problem—and how what we're building solves it—go a long way toward making a much better product. But we need to acknowledge that building software isn't the same as working on an assembly line. You're not just building one more widget like the one you built a few minutes ago. Each new story we create software to support is something new.

One of the luminaries in the Agile development community is my aforementioned friend, Alistair Cockburn, who once told me, "For every story you write, you need to put three into your backlog of stories."

I asked him why, and he said, "You just do."

I asked, "What should I write on the other two?"

"It doesn't matter what you write."

"What do you mean?" I asked, "I have to write *something* on them!"

Alistair replied, "Well, if you have to write something on them, then write what you want on the first card, and on the second card write 'Fix the first card.' Then on the third card, write 'Fix the second one.' If you aren't going around this cycle three times for each story, you're not learning."

In a traditional process, learning gets referred to as *scope creep* or *bad requirements*. In an Agile process, learning is the purpose. You'll need to plan on learning from everything you build. And you'll need to plan on being wrong a fair bit of the time.

Eric's strategy used in Chapter 3 helped him build smaller solutions and continue to iterate them until they were viable. Eric counted on learning from every release.

The *Mona Lisa* strategy used by Mike and Aaron in Chapter 4 helped them slice every story down into smaller, thinner, undeliverable bits so they could learn sooner and manage their delivery budget wisely to finish on time.

These are both great learning strategies. Try those. Invent your own. But please don't assume you're always right. I promise you'll be disappointed.

It's Not Always Software

In 2011, Kent Beck—the creator of the story—opened one of the first Lean Startup conferences with his revision of the Agile Manifesto. If I'd done it, it might have been blasphemy. But he's one of its creators, so he should know. He revised the value about working software to read:

> *Validated learning* over working software (or comprehensive documentation)

If you remember from Chapter 3, validated learning is the super-valuable concept that comes from the Lean Startup process. The key word there is learning. What makes it *validated learning* is discussing what we want to learn as part of making something, and then going back and considering the *consequences*—reflecting on what we learned or didn't learn. And one of the things we're realizing is that we don't always need to build software to learn. But we do usually need to make or do something.

I like using stories to drive the work we do to build simple prototypes, or to plan the work we're doing to interview or observe users. I like talking about the who, what, and why for those things, too. I like agreeing on what we'll make before we make it. And I look back at the consequences of having done it to consider what we've learned.

Try using stories to drive the making of anything, whether it's software or not.

Plan to Learn, and Learn to Plan

Story maps are useful for breaking up our big product or feature ideas into smaller parts. Chapter 3 and Chapter 4 were about slicing up those smaller parts into buildable chunks where each chunk was focused on learning something. But there's a different way to break things down that you need to be aware of, and to keep separate in your head. It's the work we do to break down a story into our plan to make something. That's what we'll talk about in the next chapter.