Agile Story Essentials

Use cards as the tokens for the conversations you'll use to plan, design, describe, construct, and validate your product

Stories are for telling

In the late 1990's Kent Beck had a simple idea to solve one of the biggest challenges in software development: communicating the details of what to build. By simply getting together and "telling our stories" we could build shared understanding in the minds of everyone involved.

In the conversation we'd focus not only on what to build, but who would use the software and why. Our goal is to identify the most valuable thing we could most economically build.

Stories get their name from how they're used and not how they're written

Kent Beck, author of Extreme Programming Explained

What I was thinking of was the way users sometimes tell stories about the cool new things the software they use does:

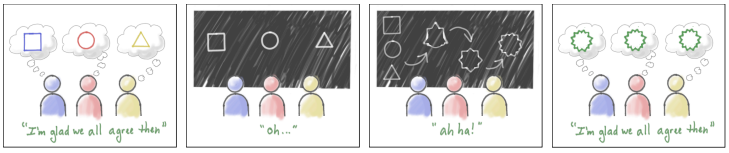
"I type in the zip code and it automatically fills in the city and state without me having to touch a button!"

I think that was the example that triggered the idea. If you can tell stories about what the software does and generate energy and interest and a vision in your listener's mind, then why not tell stories before the software does it?

A story is a token for a conversation

For every story in your backlog, put in three cards. The first is what you want, the second one is there to remind you to fix the first one. The third is to remind you to fix it again.

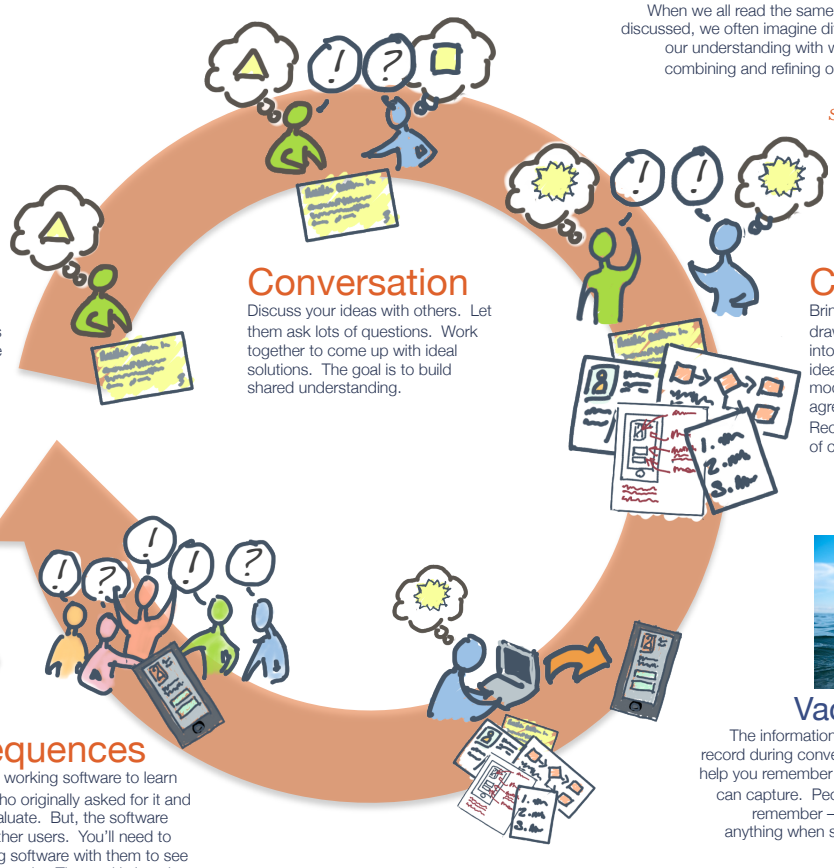
You've got to iterate or you're not doing it right.



Shared Understanding

When we all read the same document or hear the same discussed, we often imagine different things. It's describing our understanding with words and pictures, and then combining and refining our ideas that leads to shared understanding.

Shared documents are not shared understanding



Card

Write your product ideas on cards, one per card.

Conversation

Discuss your ideas with others. Let them ask lots of questions. Work together to come up with ideal solutions. The goal is to build shared understanding.

Confirmation

Bring models, personas, UI drawings or whatever you like into the conversation. Identify ideal solutions and draw new models. Work towards agreement on what to build. Record that agreement as a set of confirmation tests.

Vacation Photos

The information, drawings, and models you record during conversations are mementos that help you remember many more details than you can capture. People that weren't there won't remember – just like they wouldn't recall anything when seeing your vacation photos.

Consequences

Now we've got working software to learn from. Those who originally asked for it and the builders evaluate. But, the software was likely for other users. You'll need to test the working software with them to see if it meets their needs. The goal is learning.

And your ideas for improvement start the cycle again.

Construction

Developers, testers, and others equipped with information from conversations and the shared understanding that comes with it build and test the software.

What's on the card

Use story cards, or items in backlogs they way you might cards in a library card catalog. Write just enough information on them to help you find the rest of the details when you need to. Use the card or list item to organize stories, prioritize, and plan.

On a typical card you'll find:

Short title — One that's easy to read in backlogs and easy say in standup meetings

If you catch yourself referring to the story by it's number, stop it

Description — If the title isn't enough, write a description. Try to include who, what, and why. The template could be handy here.

Meta-Information

- Estimated development time
- Estimated value
- Dependencies
- Status

"Talk & Doc"

You'll have many discussions around stories with team members in a variety of roles. Draw pictures and record details as you do.

Bring models like workflow models, use cases, UI designs, or anything else that helps you explain the story. But, be prepared to modify it during the conversation.

Draw on whiteboards, model with post-it notes, or record on flipchart paper during your discussions.

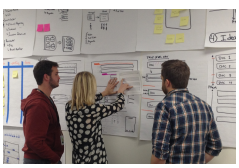
Keep models from your discussions as mementos to help you remember the details discussed.

Before you build, agree on what you're building

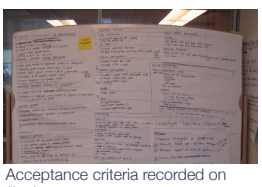
Before the team makes a commitment to build software described by a story, agree on acceptance criteria for the software. Record the answers to these questions:

- *What will we test to confirm that this story is done?*
- *How will we demonstrate this software at a product review?*

Story cards arranged as a map with UI sketches added in.



Story discussions supported by flip chart paper & drawings



Acceptance criteria recorded on flipchart paper

When writing story tests, use examples. See Gojko Adzic's **Specification by Example** for valuable tips.

Stories: Concept to Delivery

Progressively split and refine stories as you move them from vague idea through to working software

Opportunities

Create an **opportunity backlog** from product ideas, and customer, user, and stakeholder requests.

Discovery

Use discovery to elaborate, design, and validate product ideas. Your goal is to identify the smallest viable product you can. Discovery work results in a **product backlog.**

Delivery

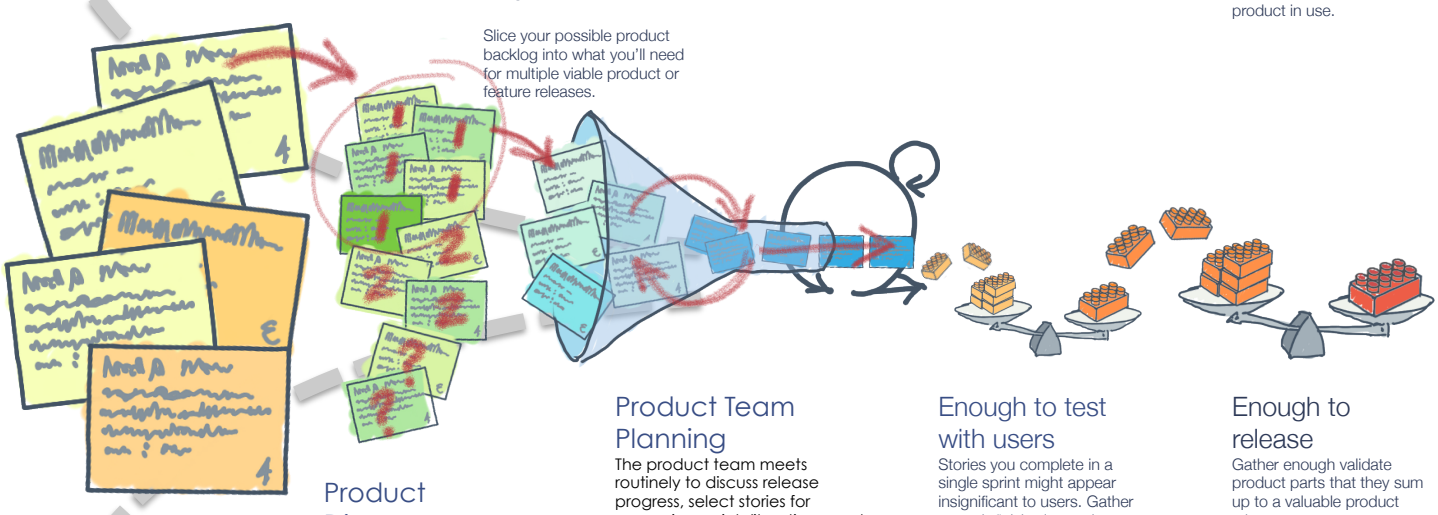
During delivery you'll focus on designing, decomposing, and describing backlog items.

Validation

Review finished software with the team and stakeholders. Validate product parts with customers and users.

Release

After your software is released, continue to measure the product's performance relative to its target outcomes. The most valuable opportunities come after seeing the product in use.



Slice your possible product backlog into what you'll need for multiple viable product or feature releases.

Opportunity Assessment

Before spending time going into details on any idea, discuss who the product, feature, or improvement is for, what benefit it will bring by building it, and how much it could cost if it's similar to other solutions we've built. Use the results of this conversation to prioritize opportunities, and to make go/no-go decisions.

Product Discovery

Product discovery is the work we do to determine what we should build. Use discovery to answer questions:

1. What problems are we solving, and for who?
2. What solutions will customers and users value?
3. What are usable solutions?
4. What's feasible to build given the time and tools we have?

Product Team Planning

The product team meets routinely to discuss release progress, select stories for upcoming sprints/iterations, and plan the work needed to get stories ready for the delivery team.

Story Workshop

Product team members meet with delivery team member regularly to work through story details and agree on acceptance criteria

Some call these workshops backlog refinement or backlog grooming meetings. But they're really the story conversations we need to have

Enough to test with users

Stories you complete in a single sprint might appear insignificant to users. Gather enough finished tested product parts to validate users can reach a meaningful goal before testing.

Enough to release

Gather enough validate product parts that they sum up to a valuable product release.

Release Strategy

During Discovery, try using a story map to slice a while product or feature into a series of viable releases.

When splitting stories, think cake

Use each story to describe an piece of software you can "taste." That is, once you've built it, you should be able to learn something from having done so. Whole features may have value to customers and users. But, it often takes a few stories to add up to a whole feature.
The steps for making software are development tasks.

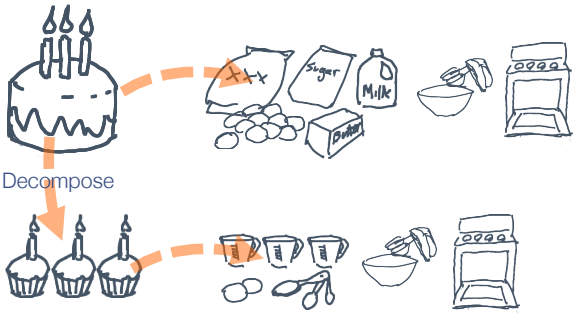
Demonstrable, testable software is the result of those tasks. If the software doesn't have user interface, you'll need to find another way to show that it works.

Stories

Stories describe something you can delivery and evaluate

Delivery Tasks

Delivery tasks give the "recipe" – describe the work someone needs to do to create the story



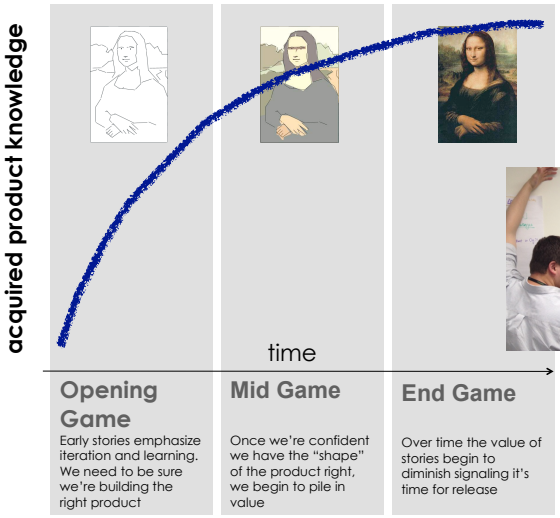
Decompose stories into smaller deliverable stories

Smaller stories often have similar recipes, just less of any one ingredient. For example all stories will have some testing, smaller stories should take less time than larger stories.

Work like da Vinci to finish on time

When managing a release budget, split larger stories into "opening game," "mid game" and "ending game" stories.

Try to get the "big picture" as soon as possible. Early versions that are fully formed but immature allow early functional and performance testing. They allow earlier validation that your concept is right.



Opening Game

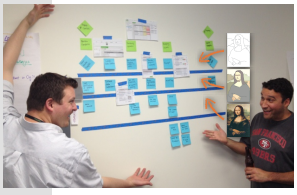
Early stories emphasize iteration and learning. We need to be sure we're building the right product

Mid Game

Once we're confident we have the "shape" of the product right, we begin to pile in value

End Game

Over time the value of stories begin to diminish signaling it's time for release



Development Strategy

Try using a story map to slice a feature into opening, mid, and end-game parts.