50 QUICK IDEAS TO IMPROVE YOUR USER STORIES

by Gojko Adzic

50 Quick Ideas to Improve your User Stories

Gojko Adzic

This book is for sale at http://leanpub.com/50quickideas

This version was published on 2014-03-25

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Gojko Adzic by spreading the word about this book on Twitter!

The suggested hashtag for this book is #50quickideas.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#50quickideas

Contents

Introduction

50 Quick Ideas to Improve your User Stories will help you define better stories, spot and fix common issues, split stories so that they are valuable but small, and deal with difficult stuff like cross-cutting concerns, long-term effects and "non-functional" requirements. Above all, this book will help you achieve the promise of agile and iterative delivery: to ensure that the right stuff gets delivered through productive discussions between delivery team members and business stakeholders.

The book is structured into four parts: Card, Conversation, Confirmation and Slicing. The Card part deals with organising information on user stories before they get accepted into the delivery pipeline. The second part, Conversation, contains ideas to improve a discussion between delivery team members and stakeholders once stories get into the pipeline. The third part, Confirmation, will help you define better acceptance criteria for stories and test them throughout delivery and after they are implemented. The final part, Slicing, deals with breaking stories down into small and valuable pieces, before and throughout delivery.

Each part contains ideas that I've used in with teams over the last five or six years to help them manage user stories easier and get more value out of iterative delivery. All

these ideas are just that - ideas - and not hard and fast rules. Software delivery is incredibly contextual, and what worked for one team might not necessarily work in other contexts, and ideas that failed in one context might actually turn out to be useful to many other teams. Treat all the proposals in this book as experiments - try them out and if they help keep doing more.

What to expect from the LeanPub version

By practising what I preach, this book will be delivered iteratively and planned adaptively. The book you are reading at the moment is an incremental slice, produced using early drafts on LeanPub. The goal with incremental slices is to get feedback early and consolidate the content later into a nice visual book, similar to Impact Mapping.

As an early draft reader, you get access to valuable ideas and practices earlier than people who will buy this book on Amazon, but they might be a bit rough around the edges. I hope this is a good trade-off between fast value delivery and fully formed book. Here is what to expect from the version you are reading now:

- Some chapters might not make it at all to the final version, so you are getting more content through LeanPub than in the final book

- Chapters will get shorter and more focused over time as I rewrite them using readers' feedback
- The book won't be copy-edited until most of the content is there, which means that unfortunately there will be some spelling and grammar errors.
- The illustrations and graphics will mostly appear in later versions of the book

Please help by providing feedback on these ideas, especially if you spot something that was not explained clearly, or something that you disagree with. You can contribute to the discussion by using the Google group 50quickideas

Divide responsibility for defining stories

One of the most common mistakes with user stories is to expect business stakeholders to fully define the scope. By doing that, delivery teams are effectively avoiding the responsibility (and the blame) for the ultimate business success of a solution. Although there are many reasonable arguments why this is good, there is also a huge unwanted side-effect: the people who are inexperienced in designing software products – business users – end up having the ultimate responsibility for product design. Unless business users have detailed knowledge of the technical constraints of your product, an insight into current IT trends and capabilities, and a solid understanding of your architectural choices, this is not a good idea. I can write a whole book on why this is a bad idea, but Anthony Ulwick beat me to it – read What Customers Want if you need convincing. The end result is often technically suboptimal, with lots of technical debt because things need to be hacked in, error-prone design and a huge waste of time and money on maintaining overcomplicated solutions.

The cause of this problem is a common misconception of the stakeholder role in agile delivery methods. The Product Owner or the XP Customer should be responsible

for deciding what the team will work on. But deciding isn't the same as defining, and this is where the things go wrong! I strongly believe that getting business stakeholders to design solutions wasn't the original intention of user stories – but many teams have fallen into this trap. If this situation sounds familiar, here's an experiment that can help you fix it:

- Get business stakeholders (sponsors, XP customer, product owners…) to specify only the "In order to…," and "As a …, " parts of a user story
- Get the delivery group (team) to propose several options for "I want…"
- Both sides together evaluate the options and the business stakeholders decide which one will be implemented

I've done this experiment with teams that misunderstand stories and business users fully specify everything in a task management tool, expecting developers to just do it without a discussion. By explicitly limiting the scope of what business users are allowed to specify, this technique can force a conversation. People will be able to see the benefits of face-to-face discussions instead of handing information over using a task management tool. Conversation is a lot more difficult to skip when one side can't write the whole story on their own. By making the delivery team come up with a solution, this technique can also help to provide a sense of ownership in the delivery team, and wake up their creative side.

Key benefits

The major benefit of this approach is that it forces both sides to have a conversation in order to decide on the actual solution. Delivery team members will have to explain several options, and business stakeholders will have to evaluate them, so this experiment can shake up teams where user stories come fully specified from the business. The collaboration also puts the responsibility for solution design on the people who are good at designing solutions – the delivery team.

Because business stakeholders are constrained in specifying only the role and the business benefit of a user story, they will typically think much harder about the impacts they want to cause instead of the features. That itself is a huge step towards preventing the user story stream of consciousness. The stories will move from a generic unspecified value ("in order to improve business", or "in order to sell more") to something very specific ("in order to monitor inventory 50% faster"). Having a clearer problem definition is always good. In particular, this helps to understand the dimension of the problem, and how much it is worth spending on solving it before you commit to a solution.

The third big benefit of this approach is that it forces both business stakeholders and delivery teams to evaluate several solutions, reinforcing the idea of flexible scope and moving analysis from "did we understand this correctly?"

to "what's the best possible thing to do?". Expecting to deal with several options also reinforces the idea that there isn't much point in defining solutions in too much detail upfront.

How to make this work

If you decide to run this as an experiment, here are a few tips how to get the most out of it:

Communicate clearly upfront that this is an experiment and that you want to run it for a while and then evaluate with everyone (business stakeholders and delivery team). This will make it easier to get buy-in. Running process changes as limited, reversible experiments is a great way to avoid pushback and power-play politics. (For more on this, read Switch by Chip and Dan Heath).

Agree upfront that features are not allowed in "In order to…" – this is an easy way to cheat the experiment. The "In order to.." part shouldn't say anything about what the software or the product does, only what the users will be able to do differently. An easy way to avoid the problem is to enforce that part to specify a behaviour change, or enabling or preventing a behaviour.

Try to propose at least three options for how the software might provide the value business users expect. Faced with only two options, people often just focus on choosing one of the presented alternatives. With more possibilities, the discussion tends to be focused on the constraints, pros

and cons of different ideas, and often inspires someone to propose a completely new, much better, solution.

Let business stakeholders also propose options in the discussion – but not before. Presenting different options and their constraints will provide a better decision making framework for evaluating ideas, including those that business sponsors had in their heads even before the meeting (and they definitely had them). It's absolutely fine to adopt an option proposed by the business users after the discussion, if they still think that's the best solution.

The discussions should make everyone quickly understand that there is always more than one solution, and that the first idea is often not the best one. Once people are OK with this, and they see the benefits of collaboratively defining scope, you can relax the rules if you want.

Don’t worry too much about story format

There is plenty of advice out there about different formats of story cards. Some argue that putting the value clause first focuses delivery on business value, some argue that “So that I can” is a much better start than “In order to”, and I’ve heard passionate presentations about how “I suggest” is better than “I want”. I’m going to swim against the current here and offer a piece of controversial advice: Don’t worry too much about the format!

There are three main reasons why I think you shouldn’t trouble yourselves too much with the exact structure of a user story, as long as the key elements are there:

- A story card is ideally just a token for a conversation. Assuming the information on the card is not false, any of those options will be reasonably good to start the discussion. If the information on the card is false, they will all be equally bad.
- Although I’ve read and heard plenty of arguments for different card types, there wasn’t a single clear proof that choosing one format over another improved team performance by any reasonable measure. Show me where reordering statements on a

story card improved profit by more than 1% and we'll talk.

- As an industry, we love syntax wars. If you ever need proof that IT is full of obsessive compulsive types, look up "what is the best indentation level", the undoubtable superiority of tabs over spaces (or was it the other way around?), and the millions of web pages on the most productive positions of curly braces. Beyond the obvious argument about personal preference, there is value in choosing one standard way for writing code for the entire team, regardless of what gets chosen. But code is a long term artefact, and user stories are boundary objects that have served their purpose after a conversation. So the standardisation argument does not really apply here.

An interesting take on this is to intentionally experiment with different formats to see if something new will come out during a discussion. For example:

- Name stories early, add details later
- Avoid spelling out obvious solutions
- Think about more than one stakeholder who would be interested in the item - this opens up a nice discussion on splitting that story
- Use a picture instead of words
- Ask a question

Chris Matts, as always, has a nice example. He told me this story:

> My favourite story card had the Japanese Kanji characters ni and hon for Japan on it. Nothing else. It was the card for Japanese language translation.

The current milestone plan for the product I'm working on at the moment is mostly about helping users get information out easier. The user stories are examples of questions people will be able to answer, such as: "How much potential cash is there in blocked projects?" and "Average time spent on sales?". These are perfectly good stories, as they fulfil both important roles nicely: they allow us to schedule things and they spark a good discussion. Each question is just an example, and we discuss the best way of providing information to users to answer that and a whole class of related questions.

Forcing the statement into a three clause template just for the sake of it would not give us any more benefit, and it might even mislead the discussion as it would limit it to only one solution.

Don't get me wrong, I think that the Connextra card template ("As a… I want … So that") is a nice template for a discussion token. It proposes a deliverable and puts it in the context of a stakeholder benefit, which helps immensely with the discussions. But as long as the card stimulates a

good discussion, it will serve its purpose. Write down who wants something and why they want it in any way you see fit, and do something more productive with the rest of your time than filling in a template just because you have to. For example, make sure that the person in question is actually a stakeholder, and that they actually want what the card says.

Key benefits

Following a template just for the sake of it is a great way to build a cargo cult. This is where stories such as "As a trader I want to trade because I want to trade" come from, as well as "As a System I want the … report". Letting go of the template, and trying out different formats, can help to shake up the discussion

By trying out different formats, you might wake up some hidden creativity in the team, or discover a different perspective during the discussion about a story.

How to make this work

The one thing you really have to do to make this work is to avoid feature requests. If you have only a short summary on a card, it must not be a solution without context. So, "How much potential cash is in blocked projects?" is a valid summary, but a "Cash report" isn't. The potential cash question actually led to a pop-over dialog that presented a

total of cash by status for any item in the document, not to a traditional tabular report.

Focus on the problem statement, the user side of things, instead of on the software implementation. One sure way to prevent getting in trouble is to describe a behaviour change as the summary. For example "Get a profit overview faster", or "Manage deliveries more accurately".