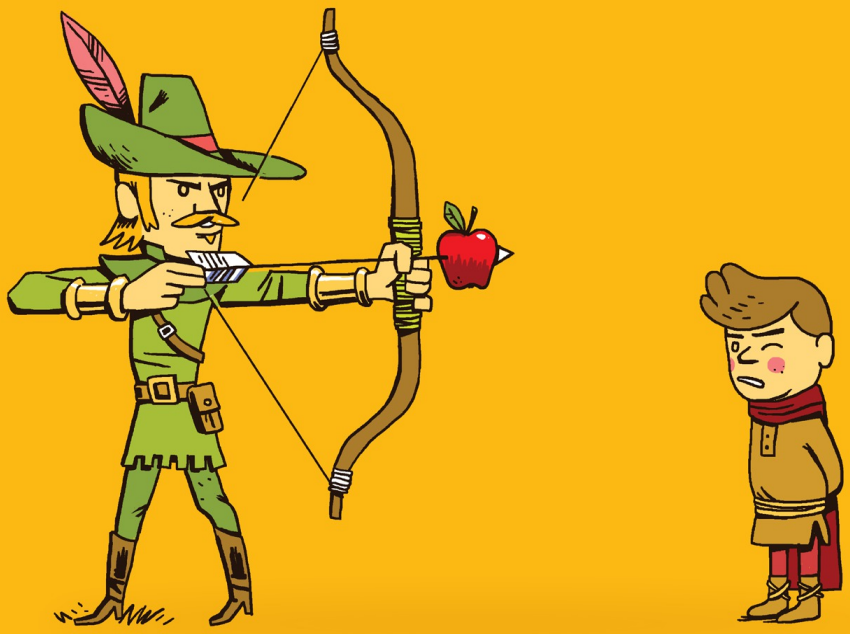
FIFTY QUICK IDEAS TO IMPROVE YOUR USER STORIES



by Gojko Adzic and David Evans

Fifty Quick Ideas to Improve your User Stories

Gojko Adzic and David Evans

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

This version was published on 2015-08-17

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 and David Evans 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

Also By These Authors

Books by Gojko Adzic

Fifty Quick Ideas to Improve Your Tests

Impact Mapping

Books by David Evans

Fifty Quick Ideas to Improve Your Tests

Contents

Introduction

This book will help you write better stories, spot and fix common issues, split stories so that they are smaller but still valuable, and deal with difficult stuff like crosscutting 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.

Who is this book for?

This is a book for anyone working in an iterative delivery environment, doing planning with user stories. The ideas in this book are useful both to people relatively new to user stories and those who have been working with them for years. People who work in software delivery, regardless of their role, will find plenty of tips for engaging stakeholders better and structuring iterative plans more effectively. Business stakeholders working with software teams will discover how to provide better information to their delivery groups, how to set better priorities and how to outrun the competition by achieving more with less software.

Who is this book not for?

This book doesn't cover the basics of stories. We assume that readers know what Card-Conversation-Confirmation means, what INVEST is and how to apply the basic strategies for splitting user stories. This isn't the first book you should read about user stories, if those terms are unfamiliar. There are plenty of good basic books out there, so read them first and then come back. Please don't hate us because we skipped the basics, but there is only so much space in the book and other people cover the basics already well enough.

What's inside?

Unsurprisingly, the book contains exactly fifty ideas. They are grouped into five major parts:

- *Creating stories*: This part deals with capturing information about stories before they get accepted into the delivery pipeline. You'll find ideas about what kind of information to note down on story cards and how to quickly spot potential problems.
- *Planning with stories*: This part contains ideas that will help you manage the big-picture view, set milestones and organise long-term work.
- *Discussing stories*: User stories are all about effective conversations, and this part contains ideas to improve discussions between delivery teams and business stakeholders. You'll find out how to discover hidden assumptions and how to facilitate effective conversations to ensure shared understanding.
- *Splitting stories*: The ideas in this part will help you deal with large and difficult stories, offering several strategies for dividing them into smaller chunks that will help you learn fast and deliver value quickly.

- *Managing iterative delivery*: This part contains ideas that will help you work with user stories in the short and mid term, manage capacity, prioritise and reduce scope to achieve the most with the least software.

Each part contains ideas that we've used with teams over the last five or six years to help them manage user stories better and get more value out of iterative delivery. These ideas come from many different contexts, from large investment banks working on internal IT initiatives to small web start-ups shipping consumer software. Software delivery is incredibly contextual, so some stories will apply to your situation, and some won't. Treat all the proposals in this book as experiments – try them out and if they help keep doing them.

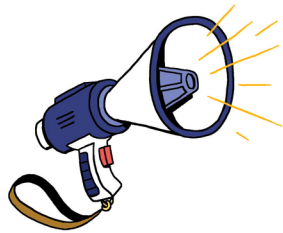
Join the conversation

There is only so much space in a book, and some of the ideas described deserve entire books of their own. We provide plenty of references for further study and pointers for more detailed research in the bibliography at the end of this book. If you're reading this book in electronic form, all the related books and articles are clickable links. If you're reading the book on paper, tapping the text won't help. To save you from having to type in long hyperlinks, we provide all the references online at 50quickideas.com.

If you'd like to get more information on any of the ideas, get additional tips or discuss your experiences with peers, join the Google group 50quickideas.

There is, of course, one more important aspect of user stories: agreeing on the right confirmation criteria for testing. To prevent scope creep, we decided to put ideas about this topic in a separate book. If you are interested, head over to 50quickideas.com and grab a copy of *Fifty Quick Ideas to Improve Your Tests*.

Tell stories, don't write them



User stories are often misunderstood as lightweight requirements, given by the business stakeholders to the delivery team. This misunderstanding leads to stories being collected in a task management tool, with a ton of detail written down by business representatives. Except in the very rare case where the business representative is also a technical expert and has a great vision for the product, this division of work prevents organisations from reaping the benefits of user stories.

To make things crystal clear, if a team passively receives documents in a hand-over, regardless of what they are called and whether they are on paper, in a wiki or in a ticketing system, that's not really working with user stories. Organisations with such a process won't get the full benefits of iterative delivery.

User stories imply a completely different model: *requirements by collaboration*. Hand-overs are replaced by frequent involvement and discussions. When domain and technical knowledge is spread among different people, a discussion between business stakeholders and delivery teams often leads to good questions, options and

product ideas. If requirements are just written down and handed over, this discussion does not happen. Even when such documents are called stories, by the time a team receives them, all the important decisions have already been made.

Effective discussions about user needs, requirements and solutions become critically important with short delivery phases, because there just isn't enough time for anyone to sit down and document everything. Of course, even with longer delivery phases documenting everything rarely works, but people often maintain a pretence of doing it. With delivery phases measured in weeks or days, there isn't enough time to even pretend. When a single person is writing and documenting detailed stories, the entire burden of analysis, understanding and coordination falls on that person. This is not sustainable with a rapid pace of change, and it creates an unnecessary bottleneck. In essence, the entire pipeline moves at the speed of that one person, and she is always too busy.

Try telling stories instead of writing down details. Use physical story cards, electronic ticketing systems and backlog management tools just as reminders for conversations, and don't waste too much time nailing down the details upfront. Engage business stakeholders and delivery team members in a discussion, look at a story from different perspectives and explore options. That's the way to unlock the real benefits of working with user stories.

Key benefits

Discussions allow business representatives not only to explain what they want, but also to ensure that the delivery team members understand this correctly. Misunderstandings between different roles are a major problem with any kind of hand-over. Explaining a story face to face prevents problems from falling through knowledge gaps.

The second benefit is faster analysis. When the entire team is en-

gaged in a discussion, functional gaps, inconsistencies and unclear requirements get discovered faster than when a single person needs to write down the details.

The most important benefit of discussions compared to hand-overs is that they produce better solutions. To be able to design good solutions, people need to know business plans and opportunities, understand the problem domain, have in-depth knowledge of technical constraints and an awareness of potential new technologies. Engaging a group of people in analysis from different perspectives helps the team benefit from shared knowledge.

How to make it work

There are several common reasons for writing down detailed stories. Most of these needs can be met without document hand-overs. Here are the most common excuses:

- When regulatory requirements or the political environment require formal sign-offs, written details serve as a record of what was approved.
- When different business stakeholders have to agree or approve plans, having something written to send out is useful. Geographically distributed organisations often have this need.
- If stories depend on the specialist knowledge of people who aren't available to participate in story discussions, written details are a good way to transfer their knowledge.
- Where third-party dependencies or legacy systems require time-consuming analysis and investigation, involving the entire team in that would be a waste of time. Written details are a good way to capture the outcomes of the investigation.

The most common excuse for handing over documents is insisting on formal approval of scope. Without going into whether formal

approval is right or wrong, if you must have it, postpone the sign-off until after story discussions. Get each story signed off as you discuss it. We've worked with several teams in regulated environments where due process demanded that a business sponsor approves scope. In such cases, business sponsors have signed off on specifications with examples produced as a result of story refinement and analysis discussions.

If the final scope has to be approved by several different business stakeholders, have the conversation a few days before officially starting to implement a story, and then coordinate with the stakeholders. For example, a team we worked with in an insurance company needed to get the details approved by all country managers, so they discussed stories a week ahead of the iteration. Their product owner then collected the results of the discussions, refined them into specifications, and sent them out to all business stakeholders to agree.

Effective teams with third party dependencies, or those that rely on external specialist knowledge, generally dedicate one or two people to investigate those dependencies a week or two before starting a story. The results of the investigations are a great starting point for the story analysis discussions.

Some teams analyse stories twice as a group: once a week or two ahead of the iteration to collect open questions and focus the upfront investigation, and the second time just ahead of the iteration to communicate and agree on the details. In such cases the details are often collected in the same task management tool as the stories, but teams treat them only as notes to start a discussion, not as formal requirements for sign-off.

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 this, delivery teams are effectively avoiding the responsibility (and the blame) for the ultimate business success of a solution. Although a case can be made for this approach, there is also a huge unwanted side-effect: 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. We could write a whole book on why this is a bad idea, but Anthony Ulwick beat us to it – read *What Customers Want*, if you need convincing. The end result is often technically suboptimal and buggy, with lots of technical debt because of bad design decisions. Delivery teams then have to waste a huge amount of time and money maintaining the overcomplicated solution.

The cause of this problem is a common misconception of the stakeholder role in agile delivery methods. The product owner or

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 things go wrong! 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:

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

We've done this experiment with teams that misunderstand stories, where their business users fully specify everything in a task management tool, expecting developers to just code without a discussion. Explicitly limiting the scope of what business users are allowed to specify can force a conversation. People can 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 have to explain several options, and business stakeholders have to evaluate them, so this experiment can shake

up teams where user stories normally come fully specified from the business side. 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 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 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'). This helps everyone to understand the dimension of the problem, and how much 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 it work

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 an effective way to avoid pushback and power-play politics. (For more on this, read *Switch* by Chip and Dan Heath).

Agree at the start that features are not allowed in the 'In order to…' part – 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 have a rule that it must specify a behaviour change, or enable or prevent 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, and 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 provides a better decision-making framework for evaluating ideas, including those that business sponsors have in their heads even before the meeting (and they always do have them). It's absolutely fine to pick an option proposed by the business users, if they still think that's the best solution at the end of the discussion.

The discussion 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.

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 business value statement first focuses delivery on business value, some argue that ‘So that I can’ is a much better start than ‘In order to’, and we’ve heard passionate presentations about how ‘I suggest’ is better than ‘I want’. We’re 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 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 the formats is good enough to start the discussion. If the information on the card is false, they are all equally bad.
- Although we’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 a significant amount. Show us 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 on the web the best indentation level, the undoubted superiority of tabs over spaces, or the most productive position for 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 discussion reminders that have served their purpose after a conversation. So the standardisation argument does not really apply here.

The Connextra card template ('As a… I want … So that') is a great structure for a discussion token. It proposes a deliverable and puts it in the context of a stakeholder benefit, which helps immensely with the discussion. But that's not the only way to start a good conversation. As long as the story card stimulates a good discussion, it serves 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.

An interesting take on this is to experiment with different formats to see if something new comes out during 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 options for splitting the story

- Use a picture instead of words
- Ask a question

Chris Matts, one of the agile business analysis thought-leaders, has a nice example:

> My favourite story card had the Japanese Kanji characters Ni and Hon (the name for Japan in native script) on it. Nothing else. It was the card for Japanese language translation.

When we wrote this, Gojko was working on a product milestone that was mostly about helping users obtain information more easily. Most user stories for the milestone were captured as examples of questions people would be able to answer, such as: 'How much potential cash is there in blocked projects?' and 'What is the average time spent on sales?'. These are perfectly good stories, as they fulfil both important roles nicely: they allow delivery teams to schedule things and they spark a good discussion. Each question is just an example, and leads to a discussion on the best way of providing information to users to answer a whole class of related questions. Forcing the stories into a three-clause template just for the sake of it would not give the team any more benefit, and it might even mislead the discussion as it would limit it to only one solution.

Key benefits

Letting go of a template, and trying out different formats, can help to shake up the discussion. This also helps to prevent fake stories. 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'.

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 it 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-up 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. The way to avoid trouble is to describe a behaviour change as the summary. For example 'Get a profit overview faster', or 'Manage deliveries more accurately'.