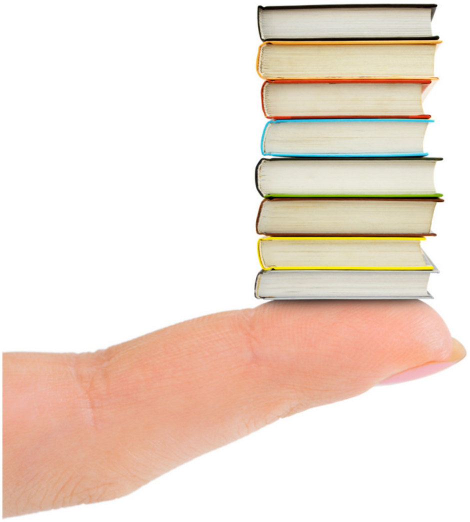
A Little Book of Requirements & User Stories



Heuristics for requirements in an agile world

Allan Kelly

A Little Book about Requirements and User Stories

Heuristics for requirements in an agile world

Allan Kelly

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

This version was published on 2017-04-04

ISBN 978-0-9933250-5-2

Also By Allan Kelly

Contents

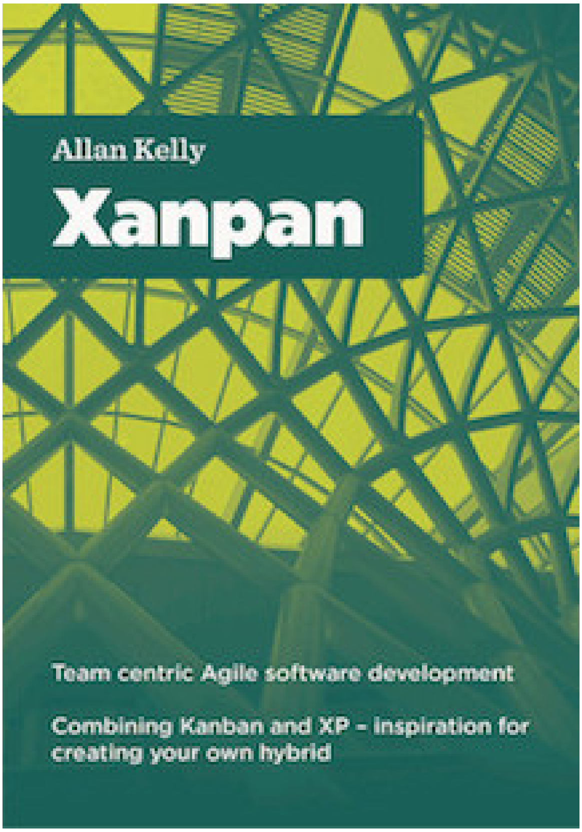
About the author

Allan Kelly has held just about every job in IT. London-based, he works for Software Strategy, where he provides training and consultancy in Agile practices. He specialises in working with software product companies, aligning company strategy with products and processes.

He wrote his first program at the age of 12 on a Sinclair ZX81, in Basic. He quickly progressed beyond the ZX81 and spent the mid-80's programming the BBC Micro in BBC Basic, 6502 Assembler, Pascal and Forth. As well as appearing in several hobbyist magazines of the time, he was a regular on BBC Telesoftware, with programs such as Printer Dump Program (PDP and PDR), Eclipse, Snapshot, Echos, Fonts, FEMCOMS and, with David Halligan, Demon's Tomb, and EMACS (Envelop Manipulation and Control System, nothing to do with its more famous namesake!).

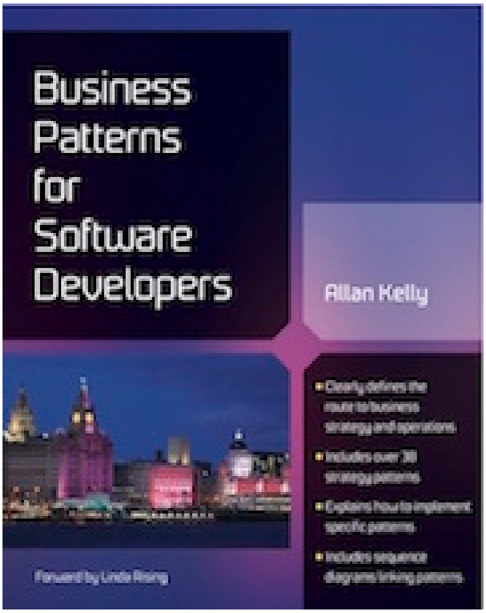
In addition to numerous journal articles and conference presentations, he is the author of *Business Patterns for Software Developers* (2012) and *Changing Software Development: Learning to be Agile* (2008), both published by John Wiley & Sons, and "Xanpan: Team Centric Agile Software Development" (available on LeanPub, Lulu and Amazon in eBook and printed versions). He is also the originator of Retrospective Dialogue Sheets (www.dialoguesheets.com).

More about Allan at http://www.allankelly.net[1] and on Twitter as @allankellynet[2].

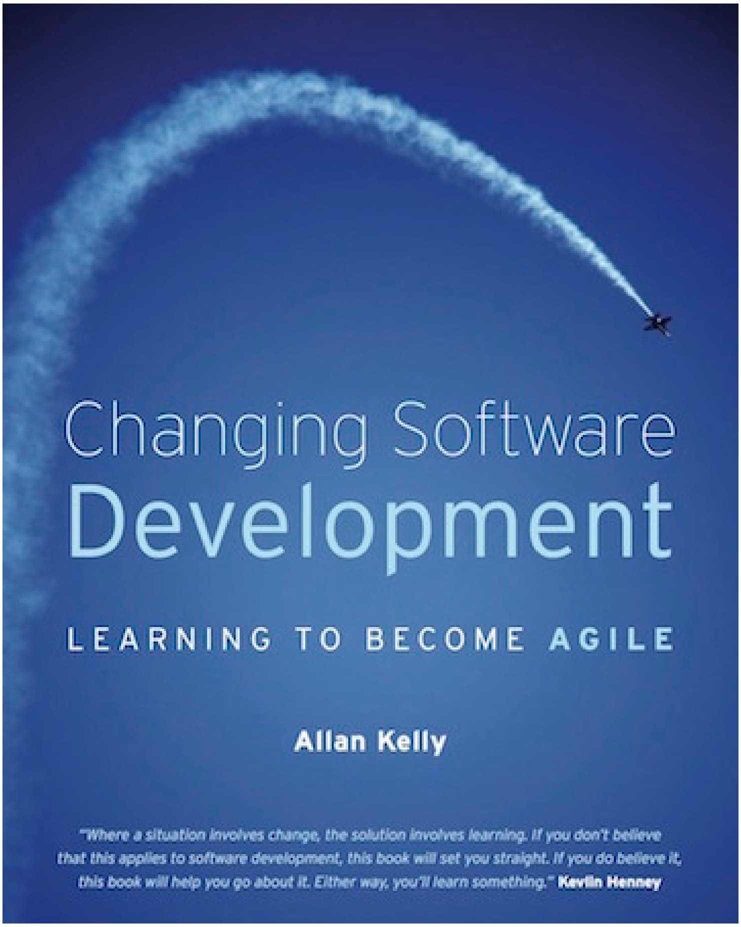

Allan Kelly
Xanpan
Team centric Agile software development
Combining Kanban and XP - inspiration for creating your own hybrid

Xanpan: Team Centric Agile Software Development

[1]http://www.allankelly.net
[2]https://twitter.com/allankellynet


Business Patterns for Software Developers
Allan Kelly
Clearly defines the route to business strategy and operations
Includes over 30 strategy patterns
Explains how to implement specific patterns
Includes sequence diagrams linking patterns
Forward by Linda Rising

Business Patterns for Software Developers


Changing Software Development
LEARNING TO BECOME AGILE
Allan Kelly
"Where a situation involves change, the solution involves learning. If you don't believe that this applies to software development, this book will set you straight. If you do believe it, this book will help you go about it. Either way, you'll learn something." Kevlin Henney

Changing Software Development: Learning to Be Agile

1. Conversation and benefit

User stories are probably the most widely used requirements technique in the agile world. This humble little who-what-why template was originally devised in 2001 by a team at Connextra in London, and it quickly gained widespread adoption:

> As a someone I want to do something So that some result or benefit

Simple, really.

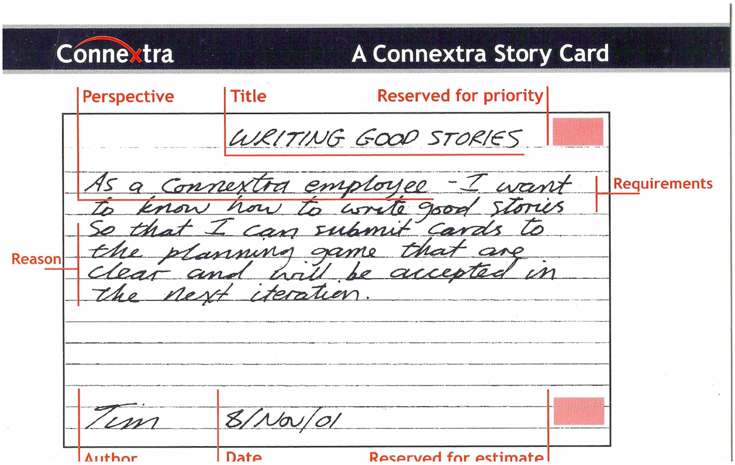
Many traditional requirements engineering and elicitation techniques are still valid in agile; it's just the results don't end up in a big document. Agile emphasizes just-in-time requirements rather than upfront preparation. The requirements person—be it the product owner, business analyst, product manager, or someone else—embodies the understanding of what is needed, and the user story represents the work that needs doing.

User stories have three attributes that fit well within agile:

- Lightweight: They don't impose a lot of (upfront) costs on the team
- Easy to understand: You don't need a five-day course to understand them

- Easy to share: Objectives are simple to communicate between the technical team and customers

It is the third of these attributes that makes user stories my preferred choice: they are inclusive. Customers can engage with stories. Many other techniques are superior in terms of analysis, rigor, and completeness. But these advantages come at a significant cost: They create a barrier between those skilled in the approach (technical experts) and those who are not (customers.) Because user stories are so simple, they help create common understanding.


Connextra
A Connextra Story Card
Perspective
Title
Reserved for priority
WRITING GOOD STORIES
Requirements
Reason
As a Connextra employee - I want to know how to write good stories so that I can submit cards to the planning game that are clear and will be accepted in the next iteration.
Tim
8/Nov/01
Author
Date
Reserved for estimate

(Thanks to Rachel for sharing.)

A Placeholder for a Conversation

User stories are not, and should not be, complete requirements for software development. People call user stories a placeholder for a conversation, meaning the stories capture the essence of what is wanted, but they don't contain the detail. When the time comes to do the work, there will be a discussion about what the stories need.

I think of stories as tokens for work to be done. They are not the work itself—that has not yet been defined—but they represent the work. These tokens can be prioritized, shuffled, refined, changed, split, and more. When a token rises to the top of the pile, it is time to work on the story.

The first job is to understand what the job is. Conversations about stories are not just between the requirements person and the coder. If the team contains software testers, include them, too. Indeed, having the tester in the conversation is more important than having the coder.

User stories themselves need not be perfect; in fact, the biggest mistake with user stories is trying to make them exactly that. They should be transitory and short-lived: a means to an end, not the end itself.

When it is time to have that conversation, try conducting it in person instead of through documentation. Written documents are more expensive than commonly recognized. Each document costs twice: writing time plus reading time. There are other more effective, more efficient forms of communication.

Having a conversation about a piece of work can be cheaper, timelier, and more effective than communicating via documents or email. In a conversation, people can ask questions, skip a section, or discuss a section in depth. So, save time (and money) by talking instead of writing documents.

Each Story Has Business Benefit

Each story should have some business value. The story may earn revenue, reduce costs, attract customers, make employees more effective, or deliver value in some other way. Putting a value on a story may require sales projections or an understanding of time savings. Ideally, I'd like to see a financial amount on each story—a hard dollar number that states the monetary value of the story. Having a financial value on a story makes prioritization easy.

I encourage teams to estimate story value in the same way some teams estimate work effort: with poker cards. Sometimes I'll invoke the TV show Shark Tank or Dragon's Den and have one team pitch its stories to another team. The other team plays investors and tries to assign value to the story.

However, putting a number value on a story can be hard, and not just because estimating a value can be hard. Sometimes stories aren't quantifiable because they deliver something intangible, or because one story delivers a small piece of something much bigger. Some stories improve the user experience, some improve quality, and others are given as unquestionable mandates: "This story simply has to be done."

Even if a story doesn't have a quantifiable benefit, it should have some statement of benefit. I like to see at least a short sentence with the story, saying something like:

> "Fred says this story is beneficial because…."

Business benefit is anything that helps the business and business representatives accept the story as useful. might include learning and knowledge creation, enquiries into the market, and demonstrating commitment to a single stakeholder.

Someone, somewhere wants the story, and they should be able to express the reason as a benefit. If there is no identifiable benefit, then why build it?

User stories are far from perfect. But if I may borrow from Winston Churchill, I have come to believe that user stories are the worst requirements technique, except for all the rest.

2. Small and beneficial

I have two golden rules for user stories. The first is that each story should be beneficial to the business[1]. Ideally, it should carry a statement of value—of course, not all benefits have a financial value, so it is better to talk about *business benefit* than *business value.*

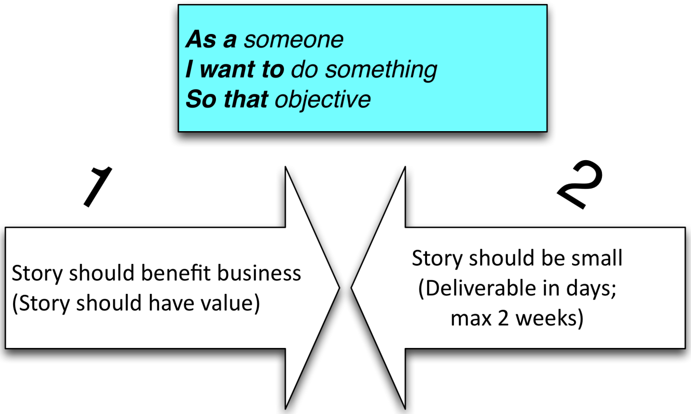
The second golden rule is: Each story should represent a small piece of work. While it’s tough to define *how* small “small” is, basically, the piece of work should be deliverable sometime soon.

How small is small will depend on many things, as a rough rule of thumb small should mean less than two weeks elapsed effort. That is, from accepting a story for development until it be ready for delivery is less than two weeks (14 days, 10 working days) should pass. Although I’m generous, I know people who think two days is a long time.

Of course there are exceptions but they should be exceptions. Indeed, I would hope that most stories end up delivered a good deal sooner than two weeks. Always strive for smaller stories.

Bigger stories need to be broken down into smaller ones. However stories do not break themselves down. It takes work, if you don’t try and make your stories smaller nobody else is going to.

[1]http://www.agileconnection.com/article/user-story-heuristics-understanding-agile-requirements



Two golden rules push against each other

A Balancing Act

There is tension between these two golden rules, and they often push in opposite directions. Getting stories that both exhibit business benefit and are small is difficult. The first golden rule of business benefit tends to push stories toward being larger, while the second golden rule that stories should be small tends to push stories toward being, well, smaller.

In the effort to make stories smaller, many stories lose their business benefit. When stories lose identifiable benefit, the business representatives lose interest in the stories—and in the work in general. This can also result in the technical team risk losing credibility.

When a product owner or other customer representative is part of story management and prioritization, he or she has the last say in what has benefit and what does not. Even if the technical team can see a way of breaking a story down into two independent chunks, if the requirements specialist cannot see benefit in each chunk, then the chunks do not have value and should not be split.

I know a team who split a story to preserve the system state into “Save data” and “Load data” stories. This makes sense from a technical point of view because each is an independent piece of work. But the business analyst said, “Neither has value on its own; only the whole has value to us.” Thus, the stories did not stand as good stories. The technical team is within their rights to split the “Store and restore system state” story into one “Save data” task and a second “Load data” task and then implement them, but they should not be made into separate stories.

The technical team could engage in discussion with the business analysts and point out that simply being able to demonstrate the system saving the data could have business benefit, by showing progress to a customer. But the business analysts have the right to stick to their original position.

How Small Is “Small”?

How small is “small” will depend on many things, but as a rough rule of thumb, “small” should mean less than two weeks’ elapsed effort—that is, from accepting a story for development until it is ready for delivery should take less than two weeks (ten workdays). This may be a generous definition; I know people who think two *days* is a long time.

Of course, there are exceptions, but they *should* be exceptions. Indeed, I would hope that most stories end up delivered a good deal sooner than two weeks.

However, the goal is to always strive for smaller stories. In terms of process, this is preferable because smaller pieces of work flow through a process more easily. In terms of forecasting, it is better because smaller requests can be accomplished more quickly, which results in a smoother flow and greater predictability.

Business benefit increases because delivering work sooner generates benefit sooner, so the return on investment is greater.

With smaller individual stories, it is also easier to spot risks and identify troublesome work. Risk is further reduced because when work is flowing smoothly, the system as a whole is less prone to disruption.

Finally, the impact of encountering a problem is less because the affected work is smaller. Failure of a hundred-thousand-dollar piece of work costs a hundred thousand dollars, but failure of one of ten ten-thousand-dollar pieces of work of work is only ten thousand dollars.

So, What's the Right Size?

Unfortunately, there is no one right size for every user story. The size of a story depends on a number of factors, and the right size for one team may be too big or too small for another team.

One reason is that the size depends on the knowledge the team creating the software has. A team who has been working in the same domain for a long time will instinctively understand a lot more than a team new to a domain. They will need less explanation of what needs doing, and they will know the language, domain, and context of the story.

I once knew an outsourced development team working for an airline. Few on the development team initially had experience in the travel industry at all, let alone the airline business. The lack of experience meant they needed more help from the customer team and lots of detail on stories, and the stories were so small that it was difficult to see the business benefit.

Contrast this with a team who has worked directly for an airline for years. Faced with the same user story, this team will immediately know the industry terms and language, plus many of the airline's existing standards for things like passenger name length, punctuation, and more. Such a team can work with less detail and bigger stories.

Story size will also vary depending on the technologies in use. It appears that teams working in the more modern, high-level languages, such as PHP, Python, and Ruby, can handle bigger stories than those working in older system languages such as C, C++, and Java.

Iteration length and overflow rules will also play a role. Teams working on four-week iterations will be more willing to accept bigger stories, while those working on one-week iteration will challenge bigger stories more often. Given the advantages of small stories, the pressure created by shorter iterations can be beneficial.

Practice Makes Progress

Writing stories that are both small and beneficial is hard. It takes skill and practice, and it doesn't happen overnight or by attending a training course. (It won't even happen just because you read this article.)

Therefore, especially for people who are new to writing stories, every opportunity should be taken to make stories smaller. You will make some mistakes, but those mistakes will help you learn. Just don't lose sight of the goal of keeping stories small and tasks manageable—while retaining business benefit.

One day you may have too many small stories, when that happen it is time to start thinking about merging stories and making them bigger. Until then keep making them smaller - while retaining business benefit!

3. Assessing the Business Value of Agile User Stories

Some years ago I worked with an airline that was writing some new booking pages for its website. Following good practice, the airline's team tested the pages with sample users to see what they thought of the new design. On the whole the feedback was good, except one thing: The sample group of customers said the new pages made it harder to find the lowest price for a flight. The airline decided to go ahead with the new content anyway.

When the pages went live the airline's revenue went up. Customers were spending more money. What customers valued and what the airline valued were different things, there was no right or wrong.

Ideally, I'd like to see companies put a dollar amount on each planned business story, but to be fair, pinning down the financial value can be hard—especially in a corporate IT setting. And as this airline example highlights, it is not just a question of how much value is anticipated, but also how sustainable it is. One could argue that while the airline increased revenue in the short term, in time, customers would start to consider their flights more expensive and take their business elsewhere.

I want to look at ways to assess business value and some of the considerations to think about.

Calculating Business Value

The obvious way to put a business value on an agile user story is to consider what difference it will make and what financial benefit that will bring. Typically, we would expect new technology to either increase revenue or reduce costs.

Let's consider the following story:

> As a widget seller, I would like to sell my widgets online.

That might look easy, but some analysis is needed. For a start, how many widgets might we expect to sell online? That requires us to ask, do people buy widgets online? And if the answer is no, we need to ask, would people buy widgets online?

Once we get through these questions, suppose we determine that a hundred thousand people might buy widgets online. Then we need to ask what the selling price is likely to be. We might know what a widget sells for in a shop, but would people expect to pay less online?

When we eventually reach a figure, that breeds more questions, such as: If we sell widgets online, will we lose offline sales? This known as *cannibalization*: when one product or channel takes sales from another of your products or channels.

Answering these questions may well involve some guesswork, but herein lies another problem: Guesswork is open to questioning. If someone doesn't like your answer, they can attack the guesses you made to get to it.

The more background research and analysis you do, the less guesswork will be required, but eliminating guesses actually might not make calculations more accurate. In some cases, intuition and rough rules of thumb can be more accurate than complex formula and data. (Gerd Gigerenzer's book *Risk Savvy*[1] contains many such examples.)

If you get lucky, then as you do the analysis, you'll discover some driving forces that render all the others insignificant. For example, if in analyzing online and offline widget sales you were to find that over the last five years online sales were rocketing while offline sales were plummeting, then it becomes clear the future is selling widgets online.

Assessing Cost Savings

Making a rational assessment of the cost savings in an organization should follow a similar structure to the revenue example:

> As a supermarket manager, I want self-checkout lines so that I can reduce the number of cashiers.

Faced with such a story, we need to ask, how many customers will use the self-checkout? And how many cashiers might that replace?

One might also consider whether self-checkout might lead to increased theft or reduced opportunities to upsell additional products.

[1]http://www.economist.com/blogs/prospero/2014/05/qa-gerd-gigerenzer

Some of the same catches apply, generating the need for more guesses or further research. However, in situations like this, additional questions also come into play: Will the customer experience be better or worse? Such a qualitative change is even more difficult to value. Creating a worse shopping experience may not immediately affect the bottom line, but over time it can make a big difference.

A second question to ask is, how quickly will these changes happen? It might take years before customers are happy to use the self-checkout regularly and the staff can be reduced. Researchers have found cases where IT changes took decades to realize cost savings.

Time as a Business Factor

Time plays a still more insidious role in these calculations: The more analysis and research you do in order to tighten up your business case, the longer it takes, and time is money.

Initially, analysis and research costs money because someone gets paid to do it. It may also entail extra costs such as accessing research reports. But it also costs because such analysis and research delays development and release of a product into the market. Delaying product release means the benefits are delayed, too. Delay also leaves opportunities for competitors to release a product.

Back when new IT systems took months or years to develop—think about the days of COBOL and C programs—it made sense to do lots of upfront analysis. However, in a world of tools and frameworks with far higher productivity, such as Ruby on Rails and Amazon Web Services, it might be possible to develop a product, deploy it, and analyze the results in less time than it would take to do a detailed analysis.

The Importance of a Company Strategy

Answering some of the questions outlined here requires referencing your company's overall strategy, and perhaps the specific IT strategy. This provides one shortcut to evaluating user story requests: Does a story align with the business and technology strategy of the company? If not, you've saved yourself some analysis time.

However, this requires that companies have a strategy and that it is clearly communicated. Let me leave you with a story with three possible endings:

> As a taxi company, I want customers to be able to book a taxi using a phone app so that …
>
> a) I can reduce the number of people employed answering phones to save money.
>
> b) I can take more bookings for more taxis and generate more revenue.
>
> c) I can disrupt the business model of taxi companies and build a single global company.

Valuation of a story is going to be vastly different depending on which ending your story has, and the ending depends on your company strategy.

When presented with a new user story, these are all some major factors that should be considered when evaluating the important question of business value.

Quick User Stories FAQ

What is the right size for a User Story?

There is no universal “right” size for a User Story. There is no “right” number of words, no right length of work, or “right” business value. All will depend on the team and concerns of your environment.

Each story should be deliverable in a short space of time, anything that takes more than two weeks is probably too big but there are teams that would regard anything that takes more than two days as too big.

In general the more a team know about the thing under construction - the more they know about the domain, the customers, the business model, etc. - the fewer words needed and the “larger” the request can be. That is, if the team know little about the domain, have never met a customer and work for a supplier company in another country the product owner will need to invest more time and detail in both the user story and the subsequent conversation.

Under such circumstances - and particularly when the technical team is in a different location - there can be pressure to write bigger stories. The development team may ask for more information, in writing. The product owners peers and superiors may suggest adding more detail. Under such circumstances writing more can be self defeating, the product owner spends more time writing so has less time for conversation, the development team are more restrained about asking questions and superiors think having everything in writing provides some sort of guarantee.

Remember: the longer a document, or story, is, the less likely it is to be read. And, the longer a document, or story, is, if it is read then the less the reader will remember.

At the other extreme, when the development team is in-house, when the team have been working in the domain for some years and have met customers then the product owner can express larger ideas in few words because the team know the context. Plus, the conversation which accompanies the user story will be better informed and, hopefully, shorter.

When is the conversation?

User Stories are often described as "A placeholder for a conversation" which begs the question, *when does the conversation happen?*

The conversation is not necessarily a single event. The conversation is an on going dialogue.

The conversation may start before the planning meeting, perhaps in a pre-planning meeting, it may later resume in the planning meeting, and again later on during development. Testers and coders - and anyone one else! - should ask questions of the product owner, customers, users, and others as they need to.

The conversation isn't over until the story is finished.

How do I determine business value?

There are several techniques, one or more may be appropriate:

1. Ask the stakeholders: even if they can't give a number in a currency ask them to give a statement.
2. Do analysis of what the organization stands to gain from the story: perhaps new revenue, perhaps cost savings, perhaps an improvement in quality, which may, or may not translate to a financial gain later. Take the analysis and construct a financial model - this may not be complicated although it might need to be.
3. Estimate the value: you might make an estimate based on your own analysis, an educated guess.

Or you might consult others: perhaps in a "delphic" approach by asking stakeholders and knowledgeable experts what value they think a story will deliver.

Or perhaps in a "wisdom of crowds" approach, I like to play "value poker" similar to the better known "planning poker" used for effort estimates. (See the *Estimating Business Value* chapter in *Little Book of User Stories*.)

How do I make my User Stories smaller?

Again there is no one right way of splitting stories. The following techniques often help:

1. Look at every conjunction (AND, OR, BUT) and split the story into two, one before the conjunction and one after.
2. Refine the roles and write more specific versions of the same story about each role.
3. Meet a few real life users who fill your roles then develop *personas* for the key roles. Rewrite the stories using personas and take the opportunity to split them.
4. Look at the value of the story and consider how you may realise some of that value with a subset of the story
5. If they story doesn't have any acceptance criteria write some.
6. Consider the acceptance criteria: see if you can make a separate story each acceptance criteria. You may end up with a basic version of the story and several stories which follow to add the original acceptance criteria. Some of the stories derived from the acceptance criteria might stand alone, others might be dropped altogether.
7. Create a story map and see if you can split out smaller journeys.
8. Talk to others! Get other opinions, the technical team can bring more eyeballs and brains to the problem. So too can other stakeholders and peers.
9. Show the stories to actual customers and users and talk see if they have insights which can make the stories smaller, or change the acceptance criteria.
10. Consider the story fidelity: could you write lower fidelity "dirt road" version?
11. If the technical team are breaking the story down to tasks then maybe some of the tasks could be stories in their own right? Or maybe some of the tasks could, with a

little tweak, be made into stand alone stories which have business value.

12. Repeat: writing user stories, and splitting them, isn't a one off activity. Revisit the stories on a regular basis and review them, ideally with someone else, and see if your thinking has changed.

This is not an exhaustive list by any means.

Remember: it is better to have more smaller stories than fewer bigger stories. But each story should deliver some business benefit (money, risk reduction, learning or something else) that the wider business can associate with.

When do I use a Story and when do I use and Epic?

Epics are placeholders for lots of stories. Epics will be broken down later into multiple stories. When that happens the Epic might exist as a "collecting" point or it might be removed all together - the stories need not have a common ancestor to be grouped (paper clips and rubber bands work too.)

Like stories an epic creates business benefit. It is even more important that epics deliver business benefit. The whole of an epics may deliver more value than the sum of the parts (the stories.)

Unlike stories epics cannot be delivered in less than two weeks. The large benefits they promise usually mean work will span multiple sprints and will be delivered a series of stories - each story will deliver some of the benefit although the whole (the epic) might be worth more than the sum of the parts (the stories.)

On the other side many teams break stories down to tasks. Tasks can be delivered very soon, days or hours but they do not normally deliver business benefit.

Epics are stories which deliver business benefits but are too large to be delivered quickly.

Tasks are mini-stories which can be delivered quickly but do not deliver business benefit. (Tasks are not normally written in User Story *As a…* format.)

Many teams don't use epics at all, and some teams don't use tasks. Most teams use two of the three available levels.

Finally, resist the urge to break epics down early. It is best to delay breaking the epics down until near the time when they are to be developed. Certainly there is little point in breaking an epic down which will not be done in the next three months. Any breakdown may well change during that time.

When epics are broken down it may be possible to delay developing some stories later. Indeed, some stories which originate in an epic breakdown may never be developed. If an epic breaks down to three stories and after the first two are built and delivered everyone is happy then why build the third?

How big should my backlog be before we start coding?

Big enough to keep the team busy for the coming iteration.

Big is not good for backlogs, small backlogs have many advantages: they are easier to manage, easier to reason about and keep overall cycle time low.

Some product owners want their backlog to include everything that might be done - something people forecasting an "end date" may well want know. Enlighten product owners recognise that no backlog can ever be complete and allow the backlog to evolve over time.

How do I write strong User Stories?

Practice.

Make sure the role in the story is real. Understand the role, perhaps add a persona.

Understand in detail how the customer will use the story. Understand how the story adds value.

Read other people's stories, have other people critique your stories.

But don't expect any story, no matter how strong to substitute for a conversation.

When is a User Story ready to go to development?

Now.

A story can enter development at any time if it is high enough priority.

It may be helpful to do some analysis on a story before development begins. It might be worth writing acceptance criteria for the story before it enters a development sprint. Some teams will place a value estimate and/or an effort estimate on the story in advance.

All of these things can make doing the story easier, perhaps faster, once it enters a development iteration but they are should not block work beginning. If any of this pre-work is need but not complete when a story enters an iteration then the work can be undertaken as part of the iteration.

For example, a team might write acceptance criteria for stories before accepting them into an iteration. However if an urgent story arrives the lack of acceptance criteria should not prevent the story from being scheduled into the iteration. If this happens then the first task in the iteration is for the acceptance criteria to be written.

Moving preparatory work on a story into the development iteration itself has several advantages: it minimises the chances that something changes between pre-work and development, it minimises the possibility that pre-work is undertaken for a story that is then cancelled, it allows the total amount of work on the story to be seen as part of the iteration.

That said many teams will set their own criteria for pre-work to be completed before a story enters an iteration. Therefore a complete answer to this question requires the asker to enquire into their working practices.

Acknowledgements and history

Many times I have looked at Mike Cohn's book (User Stories Applied) and thought "How can anyone get 200 pages out of User Stories?" They are simple, how much can you write about them?

Teams do find User Stories useful, they are a easy way to capture requests and contain three vital pieces of information: Who, What and Why. That is the obvious bit, the secret of User Stories is not the format, we could easily come up with another format, but the fact that they are easy to understand. Because they are easy to understand they help communication between all parties.

In the old days, before Agile, traditional requirements documents aimed to capture what was wanted and to promote co-operation between different groups (e.g. programmers, testers, customers, business sponsors) but, and this is a big but, because traditional documents were written in a very boring language ("The system shall...") or very precise language - sometimes even predicate logic - these documents actually formed a barrier between technical and non-technical staff working together.

Over the years I found myself helping more and more teams using User Stories. And I found that I kept giving the same, or at least very similar, advice again and again. After a while I started to think "I should write this advice down" but there was always something more important to write - most recently Xanpan.

The more I gave the same advice the more I wanted to write it down. Then in May 2015 I had a flight from London to Dallas. This was my opportunity, nine hours in the air with nothing to do. Shortly after take-off my laptop came out and I started. I don't know what the other passengers thought of me, tapping away furiously on my machine but I know that hours later when I came to put the machine away there I had 10,000 words. Far more than I expected.

I surprised myself not only that I had so much to say about user stories but that there was just so much to say about them! This also presented me with a problem.

10,000 words is a lot but it is not a book - not a traditional book anyway. Yet it is still far too big to be a magazine or journal article. Even the most generous online journals won't publish more than a few thousand at a time.

It was even too big to edit.

So I sat on it and wondered.

Then I remembered Johanna Rothman and the [Agile Connection|https://www.agileconnection.com/] online journal. Johanna had published a few of my pieces in the past and had offered to publish more. Maybe I could serialise the material?

Johanna jumped at the chance. But she also set me three constraints. First, as with previous articles she wanted each piece to be at most 1200 words. OK, a challenge but I can do that.

Second she asked me to add some stories, not user stories, war stories. Arh, I had one or two but I wasn't sure I could meet this. I could see how readers would like it - after all, how many news paper stories start "Joe Smith was sitting back in his big arm chair remembering the day..." ? But could I do it?

I decided to wing it, that is an English expression meaning: chance it, improvise, make it up as I go along. With a little luck Johanna would forget this request quickly.

To my surprise once I started trying to think of stories I had them! Some of them may need rough edges knocking off, and the odd one might even be be embellished a little to make a point clear but I had them!

Third, Johanna asked me to write in "the active voice." This was potentially the biggest challenge. I don't know what active voice it. Or rather, I can read a definition, I can read a contrast with passive voice and I can understand it for two minutes. But please don't ask me again. If you do I need to look up the definition again. To me its like those adjectives, verbs, noun things, o and the pronouns, prepositions and the rest. I don't know what they are and even when I learn I forget fast. In fact you can say that about English grammar altogether. As I tell my wife: I don't do grammar, I just write.

When I was learning to read and write the English education system didn't care much about grammar. In fact, they didn't care much about English. I was originally taught to read and write something called ITA. A seriously bad idea. Eventually my dyslexia came through and I spent four years out of mainstream school learning to read and write all over again, twice. actually.

So active voice was a big challenge. Luckily Johanna pointed me at a plugin for TextMate which highlighted all passive voice (its something to do with using words ending in "ed" or prefixed with "be".).

All in all I owe a big debt to Johanna. Since I started feeling the need to express myself I've been lucky enough to find people who have challenged me, and mediums which forced me to change. John Merrells as editor of ACCU Overload originally encouraged me to write and I developed a certain energy with writing. Blogging forced me to change and then writing patterns and attending pattern conferences helped me improve massively. Now Johanna and serialisation have challenged me again and initiated change, hopefully for the better!

So a big thanks to Johanna. A big thanks too to Beth Romanik of Agile Connection for copy editing my pieces, fixing the remaining voice, grammar, Americanising my English and getting the things online.

Not all the material here appeared in *Agile Connection*, the appendix, *Requirements and Specifications* originally appeared *Methods and Tools* as "Something Old, Something New: Requirements and Specifications" in the Fall 2014 issue. Many thanks to Franco Martinig, the M&T editor, for carrying this piece and many others.

Finally, the magic of LeanPub. Do you know using LeanPub you can accidentally write a book? Xanpan was an accident. As the User Stories pieces mounted up it became obvious they we ripe for recycling in a book. This is it.

allan kelly

September 2015

This is a sample

This free sample contains three chapters and one appendix from the full book. The full book contains 14 more short chapters and an additional, longer, appendix.

You can buy the full book on LeanPub:

https://leanpub.com/userstories

The additional chapters are:

- As a Who?
- Stories, Epics, and Tasks
- Defining Acceptance Criteria
- Acceptance Criteria, Specifications, and Tests
- Definition of Done
- Working with Nonfunctional Requirements
- Stakeholders
- Estimating Business Value
- Effects of Time on Value
- Maximising return on investment
- Writing stories - where do I begin?
- Backlog structure
- Alternatives
- Last words of advice

Appendix: Requirements and Specifications