www.payton-consulting.com

# EVERYTHING
## YOU EVER WANTED TO KNOW ABOUT
# USER STORIES
## * BUT WERE AFRAID TO ASK

### WHAT YOU WILL LEARN:

- How to create and develop powerful user stories and acceptance criteria
- How to map, groom, group, manage, and prioritize user stories
- How to decompose user stories into tasks
- How to estimate user stories
- How user stories are different from requirements
- How to use your requirements as the foundation for your user stories
- How to run a user story workshop
- How to explain and present your user stories

**BONUS!**

AN ENCYCLOPEDIA OF **15** EXAMPLE USER STORIES WITH ACCEPTANCE CRITERIA ACROSS MULTIPLE INDUSTRIES

## User Story Creation


How to write a user story

How to write good user stories

How to develop user stories

How to write user stories for bugs

How to write user story acceptance criteria

# How to Write A User Story

For those of us who have been practicing agile professionals for some time, we sometimes forget that not everyone is familiar with the terms and jargon associated with agile and scrum. User stories, unfortunately, fall into the same trap. Everyone talks about them, but not everyone knows what they are. In this article I will show you how to write a user story.

I sat in my makeshift cube in a far corner of the second floor…where they put the consultants. A business analyst comes rushing into my area, “I have to write user stories!” I stared blankly… “and?” “And I have no idea how to write one or even what it is!!”

## Step 1: Describe who wants the feature

The purpose of a user story is to help everyone understand WHO wants WHAT and WHY.  When we talk about the WHO, its not just in terms of who the user is, but also things like

- What do they do day to day?
- What is their level of comfort with technology?
- What goals are they trying to accomplish?
- What are the daily pains they deal with?

Before you dive into user stories, make sure you understand who your users are and can answer the questions above for each user. Being able to describe your users in this way gives context to the user story and a foundation for the conversation.

Example User:

- **Online Holiday Shopper**, rarely shops online except during holiday season.
- Average comfort with computers and the internet, but throws hands up at technical jargon and javascript error messages. They want to shop online and avoid the lines and traffic of the brick and mortar experience.
- Top ecommerce pains: Having to double ship, slow shipping, opaque order processing, having to call customer service to find out the status of an order.

## Step 2: Describe what they want

The primary difference between user stories and other types of requirements is that user stories describe functionality in terms of **outcome**, while other requirements describe functionality in terms of system **activity**. This is part of what makes the user story so powerful.

Describe the desired outcome of the user in the user story.

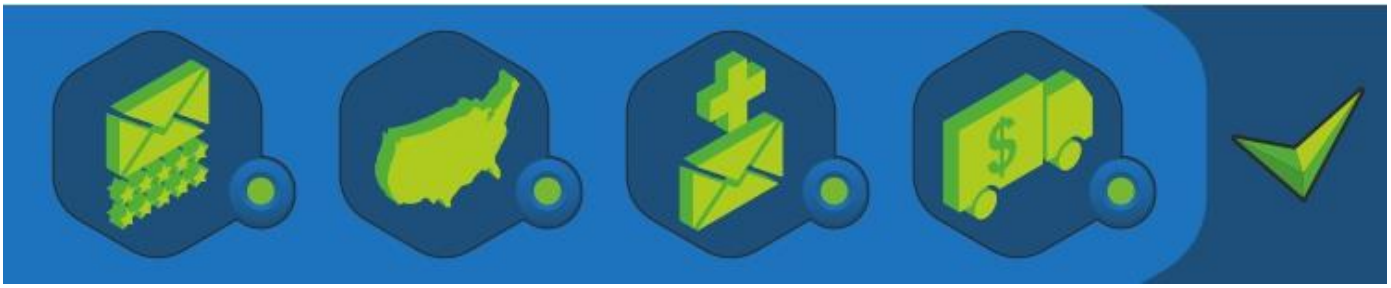
Example Outcome: Online Holiday Shopper **wants to be able to ship gifts to an address that is not their credit card billing address**.

## Step 3: Describe why they want it

Another big difference between traditional requirements and user stories is context, aka the WHY. Context is important because

- It helps us understand the value associated with the functionality
- It gives us the opportunity to explore other ways of reaching that goal

Example Why: Online Holiday Shopper wants to be able to ship gifts to an address that is not their credit card billing address **so they don't have to double ship their purchase**



## Step 4: Create acceptance criteria around the new feature

A user story is not complete without acceptance criteria. Acceptance criteria represent a way for us to validate the intent of the functionality by having test criteria defined up front.

Example Acceptance Criteria: Online Holiday Shopper wants to be able to ship gifts to an address that is not their credit card billing address so they don't have to double ship their purchase.

- **User can add up to 10 addresses in their 'address books'**
- **International addresses not supported**
- **User can add addresses in their 'account settings' or during checkout**
- **Shipping rates need to be recalculated based on shipping address chosen**

## Step 5: Engage in conversation to fill in the details

Remember, a user story is the promise of a future conversation, and are not meant to be a substitute for team members talking to each other. After you have the conversation, fill in additional agreed upon and discussed details.

There are a few key takeaways:

- The user story does not stand alone.
- The conversation is the most important part of the user story.
- User stories are progressively elaborated.

# How to Write Good User Stories

A question that comes up a lot is how to write good user stories. I can only assume because people have seen many examples of bad user stories out there.
"We can't use 'user stories' for our type of work. Its too complex and the user stories don't contain enough detail"
"Why can't you use user stories?"
"Because we cant build a system from 50 one-liners!"

Uh-oh. Maybe you have seen this situation before, maybe you have been in this situation before either as an engineer or as a product owner. Either way, its a tough spot to be in. Here are some times on how to write good user stories.

## Tip 1. There is no story without a goal. Start by outlining the goals of the users in the system.

Remember, user stories describe functionality in terms of the outcome, or the goal of the user. Once you understand what the user is trying to achieve, it makes it much easier to create a good user story.

## Tip 2. Include metadata or other artifacts with the user story.

There is a misguided conception that a user story can ONLY include the user story, and not additional artifacts as needed.

Realistically, you are still able to use anything and everything valuable at your disposal to help with the communication process. User stories are the beginning of the conversation, but not ALL communication.

### Tip 3. Make sure you have a set of cards with the different user personas described. If you don't have them, make some.

One of the reasons user stories do a poor job of communicating is because all users are treated the same. How many times have you seen "as a user…" on a user story? However, there is no system of a non-trivial size that does not have multiple types of users. Understanding who those users are, what their pain points are, and how we can address those pain points goes a long way toward being able to write good user stories for those users.

### Tip 4. Involve the customer in writing the stories.

When we sit down to figure out what needs to be done for the user (aka the customer), we often shy away from involving the customer in those conversations. This is because we think we should know everything. While we should be aware of the customers needs and goals, the best way to get that information directly is to talk directly to the customers about their wants and needs. Even better, if you can involve the customer in writing the user stories (at least the high level description), you stand a much better chance in delighting the customer with the product.

### Tip 5. Keep the stories short, remember, they're just reminders to have the conversation.

User stories have 3 parts, the card, the conversation, and the confirmation. The conversation is the most important part. This could mean conversations with stakeholders and customers to outline what they want, AND it includes conversations with the team to articulate the business need. There is no substitute for talking, and one of the positive aspects of Scrum is that it shifts the focus from documenting customer needs to actually talking about them. As a product owner, part of your "secret sauce" is to be able to communicate a vision, in both the big sense and the small sense.

## Keep an open mind

Some of the challenges teams have with writing and consuming user stories can be avoided by not having a rigid idea of what a user story is. The user story is just a starting point for the all-important conversation. Then you can flesh out and add details to reflect the shared understanding.

# How to Develop User Stories

I sat in Doug's office. He was panting. He was sweating. As the Director of Product Management, part of his job was to ensure his product managers knew what they were doing. The transition to Agile was giving him a headache, because now all his product managers had to learn to become product owners. This meant learning how to write stories, but this was not the only problem.

The much bigger problem was that there was a major initiative put forth to turn the company around, and everybody, including the CEO, was dependent on this initiative to be a success. The project had already started, and the product owners were scrambling to figure out how to develop user stories. They tried sitting in a room with the business analysts creating requirements, but they weren't very good nor very actionable.

In this section, I will lay out 4 techniques on how to develop user stories.

## Technique 1: User Interviews

Most users have no idea what they need. If you press users, you will get answers, but most likely they will give only a superficial insight into what's needed. **Users are very good at identifying their problem, but not very good at identifying a solution.**

Conduct 1 on 1 and small group interviews to talk about users' pains. Pay close attention to the problems, but take proposed solutions with a grain of salt. Use judgment and understanding to take those problems and turn them into user stories.

## Technique 2: Questionnaires

Questionnaires are good if you have a sizable user population. I have had great success in using questionnaires to gather information for large ERP and PLM implementations. However, be careful in using questionnaires as the primary means of communicating with users to gather their problems. Compared to a 1 on 1 interview, there is little opportunity to follow clues and context. Your only tool is the response in the questionnaire.

Make sure your questionnaires are well written, with specific questions that can help you further identify trends among your user base.

## Technique 3: Observation

Direct observation is great, and when paired with user interviews, create a 1-2 punch that helps you develop your user stories in a more direct manner. This is easiest when you are developing in-house solutions, as on site observation for far flung customers could be cost prohibitive.

Go to where the user typically works, and observe their habits. Pay attention to the steps they take in the system, and compare that to what they are actually looking to accomplish on a day to day basis. Observe how comfortable they are with technology in general, and look for ways you can make them more successful in their day to day work.

## Technique 4: Workshops

A workshop to develop user stories is a meeting that has the complete team plus end-user stakeholders. During this meeting, users write down as many stories as they can think up. Conducting workshops is a very effective way to gather a large number of stories quickly.

After there is a large number of stories, you will want to map them out into user flows. As you walk through the user flows, ask questions like:

- What will the user want to do next?
- What mistakes could the user make here?
- What could confuse the user at this point?
- What additional information could the user need?

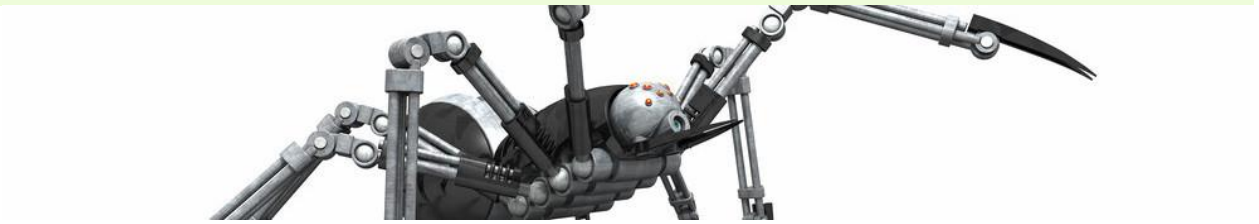
## Agile processes help you integrate new and emerging requirements throughout the process

At the beginning of your release, you should still take some time to conduct some sort of exercise to develop your initial set of stories. Rather than relying on one way to develop your user stories, you should use a combination of all the ones listed here, plus any others you may come up with.

What are some other way to develop user stories?

# How to Write User Stories for Bugs

I was finishing up a week of training with one of my clients, and by all accounts, it was a stellar week. People were excited and motivated to jump head first into Scrum, and the product owners were excited to start writing stories. One of the tech leads came up to me and said, “What about bugs?” “What do you mean?” “I mean, do we write user stories for bugs?” “If you want to. Its not absolutely necessary, but if you have 10 minutes we can chat about it.”



Should a team write user stories for bugs? You should do this only if expressing the bugs in user story form has value. User stories are useful but not always necessary.

## On Defect Management

Defect management is one of the areas where companies tend to trip over themselves trying to figure out “How agile says we should do it.” Take a step back from the process and look at the goal. The goal is to deliver working software that fulfills customer needs. In a lot of cases, a**dministrivia is a non-value added but necessary** component of delivery. We want to minimize this where we can, and when we can’t minimize it, ensure that it has value.

So instead of asking, “Should we write bugs as user stories”, ask yourself, “Would writing bugs as user stories add any value to us delivering software to our customers.”

When you frame this question, and a lot of other questions in this manner, the answer becomes obvious. Even if you pick the wrong answer, you can always retrospect and change it.

## Bugs as User Stories

So in this section I will show you how to write user stories for bugs, should you decide to go that route.

Each bug report should be considered its own story, especially if fixing the bug is likely to take as long as a typical story (2 days or so). For bugs that are minor and can be fixed quickly, these can be combined into a single bug story.

### Step 1. Use "Without" in your user story description

**Example: Administrator would like to log in from a mobile device without having the admin panel squished together and illegible.** What this does is outline the behavior you DON'T want (the bug) from the perspective of its impact on the user.

### Step 2. Outline "Steps to reproduce" in your story details

This reiterates the conditions that will help the developer understand how to create the bug and helps with testing.

Example: Administrator would like to log in from a mobile device without having the admin panel squished together and illegible. What this does is outline the behavior you DON'T want (the bug) from the perspective of its impact on the user.

- **Using android or iPhone, log into the admin panel**
- **Expected result: responsive panel**
- **Actual result: squished panel (unusable)**

### Step 3. Invert story description in your acceptance criteria

Example: Administrator would like to log in from a mobile device without having the admin panel squished together and illegible.

- Using android or iPhone, log into the admin panel
- Expected result: responsive panel
- Actual result: squished panel (unusable)

### Acceptance Criteria

- User can log into the admin panel on a mobile device and it will be usable
- The admin panel will accommodate various screen sizes
- Report viewing will be disabled on mobile, but the user has the option to send a pdf to their email

## A view from the field

Some teams like to write their bugs as user stories, and others don't. Like nearly everything, it depends on context, environment, industry, company size, and a lot of other variables. What about with your teams? Do you write stories for bugs?

## The 8 Step Process to Start Testing the Agile Way

In talking to companies about potentially adopting Agile, there is a lot of confusion with the idea of cross functional teams. In particular, there is much confusion around how testing works in an agile company. Bigger organizations, and organizations who are under regulatory scrutiny, have a lower risk tolerance than companies in other industries. As a result of their lower risk tolerance, they place a premium on quality and testing. A lot of the Agile literature talks about testing in very high level terms (use automation), but does not talk specifically about how to go about in testing with Agile while still keeping the quality concerns at bay.

## Why Test Planning is Important, Even with Agile

Cross functional teams are important. Short iterations are important. Close collaboration is important, and demonstrations of working product at the end of every sprint is important. However, quality and testing are also important, and companies need to understand how to address their quality concerns using Agile frameworks. Additionally, most companies do not have QA teams that can start coding day 1, either as part of a cross functional team or as part of an automated testing team. The transition to truly cross functional teams can take years. In the

meantime, you still have to deliver projects. As such, companies need a stop gap/interim solution to the test planning problem.

## Step 1, Deputize QA to Inject Quality Throughout the Process

Quality starts as a mindset. As such, QA needs to think of themselves not just as "testers", but as the voice of quality throughout the process. This means they need to be involved early and often, rather than waiting for code to be tossed over the wall to test.

This is sometimes difficult for QA organizations that are used to operating in a certain way. You will get the usual concerns about "productivity" (What is QA doing at the beginning of the sprint? They can't just sit around!) Of course not. At the beginning of the sprint, they should be

- Helping the Product Owner Create Acceptance Criteria
- Thinking about Testability of New Features
- Creating Positive, Negative, and Boundary Cases
- Working with Developers on Unit Tests

This serves to turn your "testers" into "Quality Advisers" for the rest of the team. Finding issues in the process during story authorship and development planning is much more effective than finding issues after code as been written.

## Step 2, Use The Acceptance Criteria As A Guide to Create Test Plans

### Use Acceptance Criteria As A Guide

When the Product Owner creates the initial set of acceptance criteria, its usually centered around 3-5 items the Product Owner will be able to inspect herself to see if the use story is complete to her satisfaction. However, there is usually much more going on under the hood than what the Product Owner sees, and this is where QA can really help the team shine.

QA should be able to help the Product Owner author additional acceptance criteria as part of story authorship by asking a few powerful questions like, "How will we know that works?" or "How will we test that?"

Product Owners sometimes fall short in creating verifiable acceptance criteria for their user stories, and QA can help make the acceptance criteria more verifiable and robust.

## Step 3, Expand on the Acceptance Criteria with Positive and Negative Test Cases

### QA Engineer Thinking Up Negative Test Cases

Most Acceptance Criteria is written as positive cases (when I do this, and do that, I should see this). One of the first places to start in test planning is to create a negative test case for every positive case.

Next, have a discussion with the Product Owner to gather more cases of acceptable behavior for the system, and ensure each of those positive cases have a negative test case. Once you have around 20 or so positive and negative cases, you have the beginning of something you can automate in order to create a smoke test and mini regression test. Do this for every story and now you have a regression suite from the beginning of the project.

## Step 4: Automate, starting at the Unit Level

### Unit Level Automation is Deep Within the Code

The first place to start in terms of automation is with the positive and negative test cases. Have QA work with developers to create the initial set of automated **unit** tests based on these cases.

QA can provide the blueprint in the form of input-process-output similar to how acceptance criteria are created. This will give development an easy way to create

the tests according to the plan. For companies who want to be more aggressive in creating cross functional skillsets, have QA create the automated unit tests under the guidance of a developer.

## Step 5, Automate at the Service Level

### Not Automating at the Service Layer Makes People Sad

A lot of software today is based loosely on a 3 tier structure, where you have a front end, a service layer, and a back end. Unfortunately, most of the time when I ask companies about test automation, it is exclusively at the front end, with nothing at the service level or at the unit level.

You can use tools like TestComplete to automate at the API level, or create a custom harness to exercise the API. Use your positive and negative test cases as a guide to exercise the business logic at the API level.

## Step 6: Automate at the UI Level

Typically, I am not a big fan of UI level automation, at least at first. This is because UI automation tends to be fragile, and even when the UI controls themselves are being instrumented for test, if there are major changes to the front end, that work tends to be wasted.

Once exception to this is input validation. When software relies on input validation (such as 0-9 or A-Z), the UI is usually the first place the input is validated. It makes sense to validate at this level if you already have unit tests and API/Service Layer tests in place.

Use the input validation rules as a guide to create negative and positive test cases based on acceptable and unacceptable inputs.

## Step 7,: Explore the Boundaries

The great tragedy of QA in most large organizations is that we rely on armies of people to run rote test plans that were written months ago as a way to say "Yes, its tested." Even worse than that, **QA almost never has enough time to manually regress everything**, so they do the best they can with the time they are given. However, rote work is mind-numbing and wastes the creativity of the people you have hired. If you have test plans that need to be run the same way every time, that is a great candidate for automation, either at the unit level, the service level layer, or at the UI level.

This frees up your humans to do work that humans excel at, and computers can't do well, creative work. You will get a much higher quality product by automating the rote work and using people do "try to break it" by doing things you didn't think of, but your users certainly will, like funky edge cases, improper input, exiting a form before its complete, partial saves, bad characters, and other items like that.

## Step 8: Add Bugs to Automation

When QA finds these strange edge cases (bugs), and they will, they need to collaborate with development to figure out where is the best way to automate testing for this regression again. In some cases it will be at the unit level, in some cases the service level layer, and in some cases the UI layer. In some cases, it will not be able to be tested automatically at all, which makes it a good candidate for the manual regression.

Hopefully this primer will give QA Managers, Agile Change Agents, and Technology Leaders a better understanding of how cross functional teams, QA, testing, requirements and development work hand in hand when using Agile frameworks. Just because you're moving faster and working differently does not mean you need to sacrifice quality.

# User Story Mapping and Management

How to create a user story map

How to groom, group, manage, and organize user stories

How to prioritize user stories

# How to Create a User Story Map

She stopped me in the hallway, lips tight, brow furrowed, hands a maelstrom of gesticulating madness. She didn't have to open her mouth for me to begin the conversation, "Hey Anita, how are you?" "Not good, I have all these user stories the business analysts created, but we have no idea how to organize them" "Ok, lets take a step back and make a story map, that will help us figure out where the gaps are." In this article, I will talk about how to write a story map.

### A User Story Map is a representation of a set of user stories along 2 dimensions:

- Sequence
- Priority

### At the top of the hierarchy is the Epic.

An Epic is a large activity that has user value, such as "Buy a Plane Ticket." Some prefer to use the user story format to create epics, but to me it really does not matter. We just want to use a phrase or sentence that captures the spirit of what we want to accomplish.

### Next, you have the workflow itself, which is divided into themes.

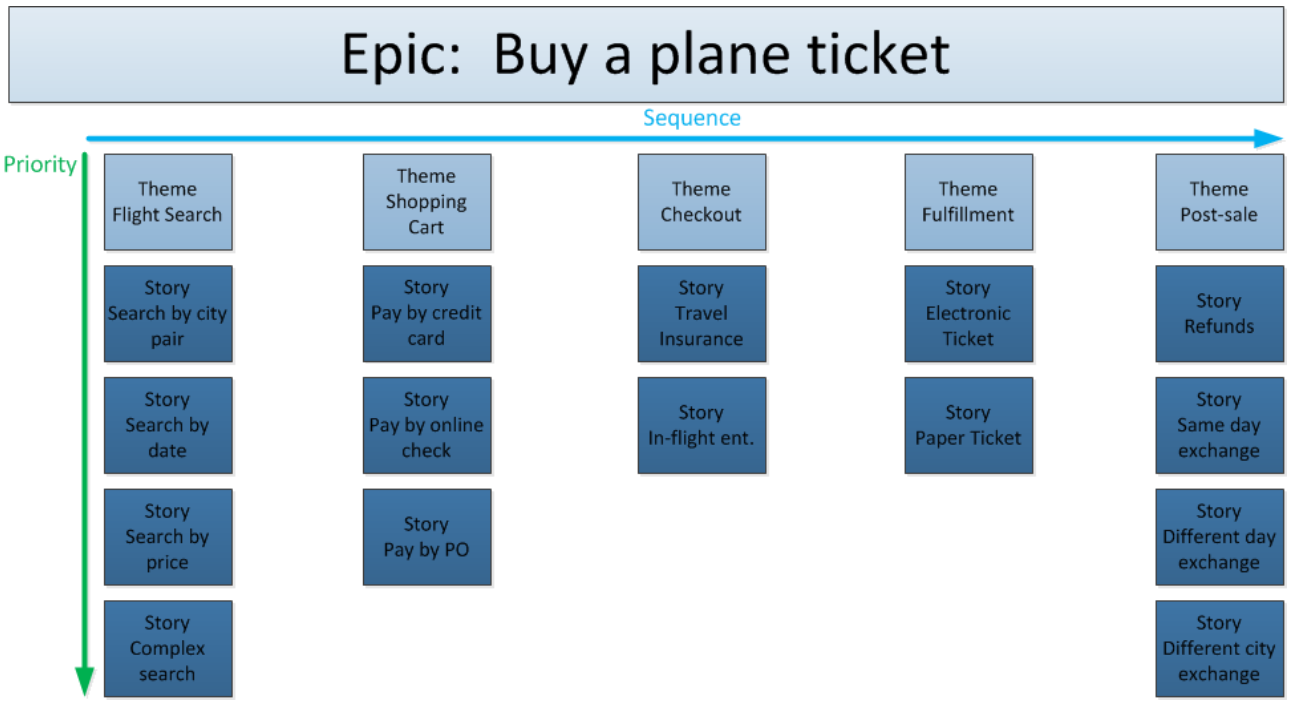
A theme is a collection of related stories, and the themes roughly correspond to workflow steps necessary to fulfill the value outlines in the Epic. Here is an example, using plane ticket purchases:

Buy a Plane Ticket

- Flight search
- Shopping cart options

- Checkout/payment options
- Fulfillment options
- Post-sale options

"Buy a Plane Ticket" is the epic, and the epic is broken down into a number of themes, such as "flight search" and "shopping cart options."




Each theme is then broken down into a set of stories that are arranged in priority order. The items near the top of the priority are more likely to be implemented first. The others are more likely to be implemented later. The top layer of the stories represents the minimal marketable features.

A story map is not a release plan. It helps us see the breadth and depth of stories to be implemented and merely **influences** the release plan.

A good story map:

- Shows us the flow of activities from the users' perspective
- Informs architecture/infrastructure needs
- Outlines user stories' relationship to each other

### Step 1: Outline the major goals (epics).

These are your highest level of user value. To come up with these major goals, think like a non-technical user. Users tend to describe applications in broad non-technical easy to understand terms that represent value.

- Facebook lets me stay in touch with family
- Mint helps me with budgeting
- Evernote allows me to take notes anywhere

These epics are the same language you would use to describe the functionality to your mother or a friend who doesn't work in the technology industry.

### Step 2: For each epic, outline the user flow from beginning to end.

Start by outlining the steps. Sometimes this is difficult. After you create a step, ask yourself, "Then what happens?" This will get you to the next step. Do the flow first before diving into the stories.

### Step 3: For each theme, outline and prioritize the user stories associated with that theme.

After you have outlined the steps, start thinking about the different ways to accomplish that step. Those will be your stories. For each story, consider "what happens if... " scenarios, that will deepen the user story list. There will be some stories that are obvious high priority, and others that will not. Often during story mapping new stories appear from looking at the problem from a different angle (horizontal and vertical).

### Treat the map as a living artifact

Congratulations, you now have a story map! Story maps are very useful to create at the beginning of a project, and helpful for release planning. Like all Agile artifacts, story maps are living artifacts. Make sure you revisit the story map every few sprints to add/update stories, and to reflect new stories and new lessons from project execution.

# How to Groom, Group, Manage, and Organize User Stories

Very often, I observe team members conversing in retrospectives, “we have to wait for the requirements to be clarified by the Product Owner, and it is eating up a lot of time and delays our work in the Sprint. In the "good old days" we had requirements specification document which was easily manageable. ” It is imperative that teams face an uphill task to manage hundreds of stories in their backlog.

I was pondering upon this problem for a while and understood that teams are missing a very important ceremony in the sprint called “Product Backlog Grooming” that would help in grooming, grouping, managing and organizing the user stories.

In short, the product backlog is the single source of requirements in agile, it is an ordered list of requirements, and the majority of the items in this list are the user stories. It needs to be continuously groomed whenever you see a need for it.

Here are few techniques for a team which can make their life easier when dealing with the user stories.

## 1. Groom the stories

The teams will have to spend at least ten percent of the time during their iteration, with the product owner to understand the existing stories in terms of clarity, feasibility and fitness in the iteration. If the story is big one, then they

need work with the product owner to break it down into smaller chunks, manageable in iteration. Mark the stories as ready for sprint after this meeting, so that there is no back and forth during the iteration.

**Remove** the stories from the backlog that are no longer needed.

**Clarify the stories** by elaboration the conditions of satisfaction as required.

**Estimate** the stories with the best known facts at that time.

**Adjust the priority** of the story with the permission of the Product Owner. The grooming activity is very much essential to maintaining the product backlog, otherwise it would become unmanageable.

## 2. Group the stories

We can group related together under one umbrella that helps us visualize the progress of stories. For example have all reporting templates related stories under 'reporting' group, all user management related stories under "user management" group and so on. This grouping of stories is called as "Themes" in agile.

Create high level groups when you start eliciting the requirements and try to group the stories accordingly as and when you add new stories to the backlog. Some tools like Rally, support a Parent-Child relationship and successor-processor relationship that supports user stories grouping.

## 3. Manage the Stories

Managing the stories is very simple. For each story in the backlog, the following attributes that would help in fetching the complete details of the origins of the story, that will make it very easy to track.

- User Story description
- Acceptance criteria
- Parent user story
- Priority

- Assumptions
- Dependencies
- Assigned to
- Estimate
- Team assigned to
- Group (if this story is grouped )
- Tasks associated
- Tests associated
- Status: Done, In-progress, completed, blocks etc… Impediments etc

**Follow the DEEP model, as Mike Cohn calls it. DEEP stands for**

***Detailed***

Appropriately Highest priority, more detail; lower priority, less, which are broken down into smaller stories for sprints/release.

***Emergent***

Growing and evolving overtime; Re-prioritized when items are added /deleted.

***Estimated***

The product backlog items are estimated. The estimates are coarse-grained and often expressed in story points.

***Prioritized***

Items that are slated for the next couple of sprints

**Organize the stories**

Organizing the stories is easy, have team-specific backlogs. Have the stories that are related to your team only in your backlog. Themes are also used to organize stories for releases.  Link your team stories on which you have dependency with other team, that way, you will come to know the status of other linked stories. Have frequent conversations with the Product Owner to prioritize and reprioritize

stories.  Help your Product Owner to remove the stories that don’t belong to your team.

Imagine your lawn in the back yard, how do you maintain it? It has to be maintained regularly? If not at some point of time it becomes unmanageable, grows haphazardly in all directions, making it difficult to maintain. Similarly, backlog grooming is not a onetime effort.

Do you think you need to consciously water your lawn, trim and manure it, so that it looks healthy?  If you didn't, imagine what might happen!

**Signal #5**. The person who created the original requirement left... years ago. If the person who created the requirement is long gone, but the requirement is still hanging around, it may be time for it to get voted off the island.

**Signal #4**. The platform that was needed to support it has been deprecated. That obscure bug for IE6 under Windows XP, get rid of it. The feature to support the Blackberry scroll wheel, delete. The module to receive faxes, gone.

**Signal #3**. Its no longer in line with strategic priorities. A few years back, we wanted to become the Kings of desktop automation software. In the last 2 years, we have pivoted toward cloud security products. Continuing to support and enhance the old products makes no sense.

**#2 signal.** It complicates the issue. Most of the time, when someone sees a problem or a gap, they immediately think of a solution. Proposed solutions are what typically make their way onto a requirements document or product backlog. This makes sense, because people tend to NOT think in terms of problems to be solved, they think in terms of solutions to implement. An example of this is when Henry Ford said, "If he asked the people what they wanted, they would say a faster horse."

This is because people tend to jump to the next logical step (in their minds) of what it would take to solve the problem. But this is not how innovation happens. Innovation happens when people look at the problem to be solved and create a novel solution to what the problem really is. Here are a few examples:

## Example 1

"We need the reports to automatically print at midnight on the last day of the month."

"Why?"

"So we can enter the figures into the state auditing system"

"So it sounds like the real problem is getting the data into the state auditing system, not printing them out. Printing them just complicates the issue. Let's implement a data feed that automatically feeds the data to the state auditing system"

## Example 2

"I need a slider so that the operator can adjust the screen size"

"Why do they need to adjust the screen size?"

"Because sometimes they are looking at it on their desktop, and other times they are looking at it on a projector"

"So it sounds like we need to solve the problem of how it looks on the projector. Rather than put in a slider for them to adjust to any arbitrary screen size, we will just put in a projector mode they can toggle. The slider complicates the issue"

When you look at a requirement from the perspective of a problem to be solved rather than a feature to be implemented, often the solution looks very different.

**#1 Signal** it's time to delete that requirement: The cost to develop and support it is not worth the ROI. As with the case from our friend at ESPN, we must weigh the costs to develop and support a new requirement against the ROI. For him, it wasn't worth 3 full time developers' salaries for an app that got, on average, 300 downloads a year.

When we are looking at the cost to develop a feature, its a good idea to understand the long term use patterns and whether or not the complexity, effort, and long term TCO is worth the additional customer engagement or revenue associated with that feature.

# How to Prioritize User Stories


His face was stern, and his voice had an air of seriousness, "I don't think you quite understand. In our business, everything is priority."

If I had a dollar for every time I heard this, I would be a very wealthy man. Yes, everything is priority. Everything is always priority because you have a large group of stakeholders and they all want their stuff first, thus everything is priority.

Realistically, you **cannot get everything done all at the same time**, so even though "everything is priority", some things will come first, and some things will come last. A softer word for priority is "order", but in this article I am going to show you how to prioritize user stories.

## Option 1: Prioritize by Knowledge Value

At the beginning of a project, what you don't know might hurt you. Removing the unknowns is a huge opportunity to positively impact the health of the project over time. In order to prioritize on knowledge value, we have to be willing to admit we don't have all the answers to the unknowns. After we do that, we can prioritize some of the "knowledge value" items on the backlog, such as spikes and prototypes.

When do you know it’s time to prioritize by knowledge value?

Here are a few signals to help you understand when it's time to stop thinking about functionality and start thinking about risk reduction:

- The team says, "We don't know if that will work..."
- The product owner says, "I don't know how the customer will react to that"
- The architect says, "I'm not sure if the platform will support this feature"
- The business analyst says, "I haven't figured out the requirements for that part yet"
- The tester says, "How will I test that?"

In each of those examples, there were clear signals that a lack of knowledge was preventing someone from having confidence to move forward.

### Option 2: Prioritize by Increased Revenue

This one is always a clear winner, but is heavily dependent on the sophistication of the product owner to be able to articulate the ROI associated with a feature or user story. Therefore, this needs to be one of the items thought about when prioritizing. An example of this would be payment options.

"15% of our revenue came from Paypal in the last version of this product, so it stands to reason that 15% of our revenue will continue to come from Paypal. On the other hand, only 5% of our revenue came from ACH debit, so the Paypal functionality will be prioritized higher than the ACH functionality."

### Option 3: Prioritize by Reduced Cost

This is another one that is easy to articulate and easy to defend, but requires some research and number crunching to have a sound basis. Cost reduction or reduction of "Total Cost of Ownership" is usually one of the driving forces behind new projects. An example of cost reduction would be changing platforms:

The old platform costs 10c per transaction, and the new platform costs 7c per transaction. Moving the functionality to the new platform will save us 30% per transaction, and we do over 1 million transactions per month.

In the above example, the cost reduction was easy and straightforward to calculate. Most of our real life situations are a bit more complex and messy. Here are some helpful tips for calculating cost reduction for prioritization

Anything that reduces time indirectly reduces cost, such as automating manual tasks. Investigate how much time your customers spend manually executing this task, and use a nominal "cost per hour" of this persons time to come up with a cost reduction figure.

Chopping out features can sometimes reduce costs. An example of this is when a company comes out with a "lite" version of software with only the core features. Fewer features = fewer support costs.

Creating an open API and allowing developers to create features can reduce costs. This is because feature development shifts to the development community, which means independent developers will be responsible for funding and supporting the add-ons.

### Option 4: Prioritize by Reduced Risk

There are all kinds of project risks to keep in mind: Technical Risk (can this be done?), Social Risk (can these people do it), and Execution Risk (will the marketplace accept this?). Items in your backlog that can point to a reduced risk can be prioritized according to the magnitude and likelihood of the risk itself. Here is an example:

The state fines us a surcharge depending on how accurate our claims payment processing is. This functionality will reduce claims errors by 30%, reducing the risk of non-compliance.

**It's always a guessing game**

Prioritization is always a guessing game, and you have to find the balance between getting a perfect priority and being flexible enough to let the work emerge. Don't ever expect to get a perfect priority. Some stakeholders will

always be unsatisfied with how it was done. As long as you have a framework to guide your thinking, you can defend the choices when asked.

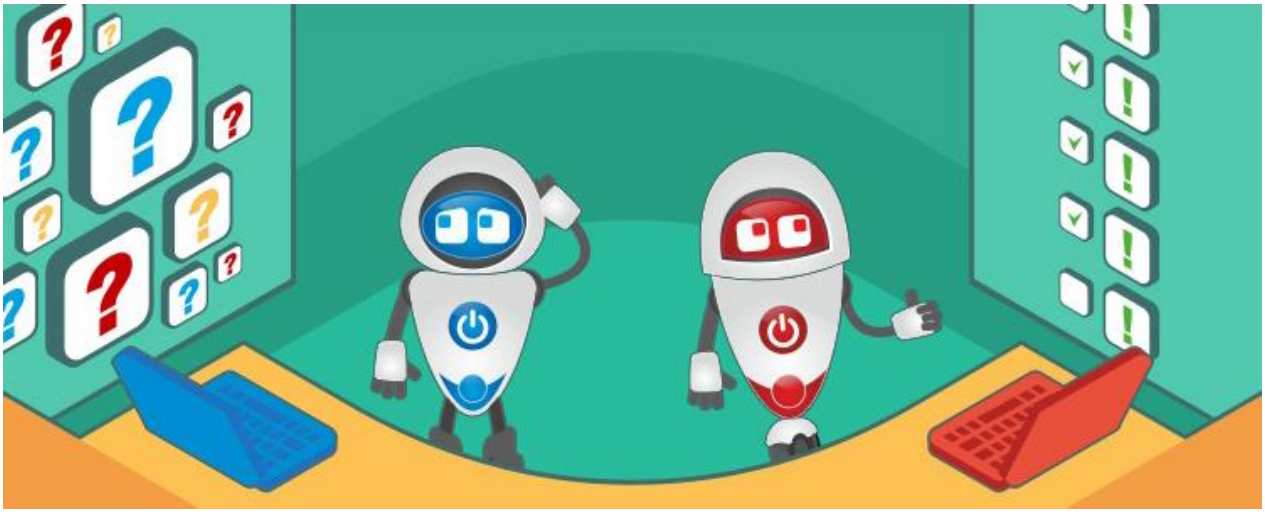
# User Story Splitting, Sizing, and Decomposition

How to break down user stories into tasks

How to estimate user stories

How to write user story acceptance criteria

# How to Break Down Stories into Tasks



Recently I was looking at the sprint backlog of a team, which just started their agile journey. When I looked at the task break down of the user stories, I noticed something like this.

User Story:

- ✓ Coding
- ✓ Testing
- ✓ Check-in
- ✓ Build
- ✓ Demo

Looking at this task breakdown, it felt like a sequential process steps defined as tasks for a story. I also noticed that every story had the same task break down with different effort estimates. I was looking at their done criteria of a story. I could not really relate their Definition of Done (DoD) and their task break down. Immediately, I asked one of the team members, “How do you make sure that you complete every task listed in the DoD? “ He stared at me with a confused smile, and said “We just do it! Sometimes there were tasks, we forget few of them and will get them done in the next sprint!”

Ah! Here is the catch...

***I see that the task break down of the team just reflects a sequential process and doesn't convey anything meaningful about what is happening with that story! Moreover, the task "Coding" doesn't convey how much portion of coding completed for a specific story.***

I also found a similar pattern of task splitting with other team's sprint backlog too!

Somehow, I find that many teams are struggling to do an effective task down of a user story!

**A task is a piece of activity that is required to get a story done**

Here are some effective tips for breaking down a user story into tasks.

## 1. Create Meaningful tasks

Describe the tasks in such a way that they convey the actual intent. For example, instead of saying Coding, describe the tasks as "Develop the login class", "Develop the scripting part for login functionality", "Develop the password encryption for the login functionality", "Create user table and save the login data to DB" etc.. Such tasks are more meaningful rather than just saying coding and testing.

## 2. Use the Definition of Done as a checklist

Let us see what a DOD is very quickly. The DOD defines the completeness criteria of a story. It includes all items that have to be completed to claim that the story is done by the development team. A simple example:

- Acceptance criteria is verified during testing
- Coding tasks completed.
- Exploratory Testing completed and signed.
- Regression test reviewed and passed.
- Unit testing – written and passed.
- Code reviews conducted.

- Defects are in an “acceptable” state to the Product Owner.
- User story accepted by the product owner.
- Regression tests run and passed
- Smoke / automation tests run (if applicable)
- Check for memory leaks
- Automated unit tests are checked in

Now how do you ensure that all items of DOD are done by the team? One way is to use the DOD as a checklist to come up with tasks for the user story so that the team can take each one of them and complete without forgetting.

### 3. Create tasks that are right sized

Another syndrome I have seen is tasks which are very small, broken down to a minute level like, 10 min, 30 min, 5 min tasks, for example: Write Accept User Name Password, Validate Login, and Give Error Messages. Breaking the user stories with too many details is an overhead. What is the ideal size of the tasks?

One guideline is to have tasks that span less than 8 hours so that each one of them can be completed in at least a day.

### 4. Avoid explicitly outlining a unit testing task

If possible, make unit testing not a separate task but part of the implementation task itself. This encourages people to practice Test Driven Development as an approach. However, this practice may not be ideally suitable for new Scrum teams.

### 5. Keep your tasks small

Do not have tasks that span across days together. Its makes it difficult to know the actual progress.

In some mature teams, I have seen, they do not do the task break down at all. They just pull in the user stories and complete them, but it is a journey for new Scrum teams to get there, and requires a strong cohesive team, and many sprints of working together.

So, how often do you think, as a team, you have to revisit the DoD, so that your task breakdown may change?

# How to Estimate User Stories

## Ideal days

Ideal days are the most common way to estimate in traditional project management. When estimating in ideal days, the assumption is "This is how long it would take if I had nothing else to work on."

Ideal days are usually used in conjunction with some sort of "efficiency coefficient" whereby someone takes the ideal days and multiplies them by the number of actual ideal days you can get in a week. For example, in 5 business days I get 3 ideal days of work done, so the coefficient is 3/5. Therefore, if I estimate this work in ideal days and call it a 3, it will take me a week.

I'm not a big fan of using ideal days as an estimation method for a few reasons:

Anytime you use calendar time for estimating, no matter the caveat, it tends to be used as a commitment. **Estimates != commitments**, especially in complex-creative work.

**It creates a distracting vanity metric**: Getting ideal days to match calendar days (disguised as a quest for "efficiency"). With this, instead of focusing on delivering value, the teams focus on "making the numbers match"

## Story Points/Tshirt sizes

Story points are a way to decouple the estimation of size and effort from the duration of time. Instead of trying to estimate how long something will take, you estimate how big it is and let your velocity data tell you how long it will take.

If you don't have velocity data, take your best guess in terms of how much work a team can do in a sprint, then measure the actual when you have actual data.

I like story points as a sizing practice because once the teams get their minds around decoupling time and effort, it makes it easy to estimate large amounts of work quickly.

Be careful: It's not a good practice to try to map story points to days directly. The point of decoupling is to understand the change in the relationship between time and points, which changes as teams improve, add new members, get reconfigured, etc.

## Threshold-based sizing

Another form of sizing I like to use is "Threshold-based sizing." This is where we set a threshold (say 3 days) and when we estimate, we are only interested in getting stories under the threshold. This eliminates distractions around precision (2.3 days vs 2.8 days) and moves the stories toward a standard size.

For stories that are above the threshold, we split them, or decrease the scope, or cut functionality until they are below the threshold.

## My preference is to use a combination of tshirt sizes and threshold estimation

At the release level, I like to use chunky tshirt sizes: XL, XXL, XXXL

At the sprint level, I like to see smaller, more granular tshirt sizes: S, M, L

When we go into sprint planning, I take the stories slated for that sprint and apply the threshold method, as a double check. When applying the threshold, I use the rule of 1, 2, 3:

- 1 user story should take
- 2 people no more than
- 3 days

Maybe its a developer and a tester, maybe its a front end developer and a back end developer, or maybe its 2 full stack developers. The composition of the 2 people working on the deliverable doesn't really matter to me. What matters to me is that we are spreading knowledge around the team so we don't end up in a situation where "Only on person knows about this.."

# How to Write User Story Acceptance Criteria

When I am working with my clients who have already started adopting Agile, one of the first item I look at is their backlog. Why? Because the quality of the backlog is a leading indicator to how well the team will perform. Unfortunately, most backlogs created by beginning product owners are in no shape to be consumed by a team, and the number one reason for this is usually a lack of acceptance criteria in the user stories. In this article, I will talk about:

- What are acceptance criteria
- Why they are important
- Whey they work well
- How to create them

## What are acceptance criteria?

Acceptance criteria are statements of requirements that are described from the point of view of the user to determine when a story is “done” and working as expected.

This helps the team reduce risk by testing against the same criteria that were agreed upon when the team accepted the work. Acceptance criteria are

emerging and evolving and assumed to be flexible enough to change until the team starts working on the story.

Anyone in the team like business analysts, QA and developers can help the PO in both creating and reviewing the acceptance criteria.

## Advantages of Acceptance Criteria:

- Triggers the thought process for the team to think through how a feature will work from the end user perspective
- Helps the team to write the accurate test cases without any ambiguity to understand the business value.
- Eliminates unnecessary scope that will add no value to the story, in other words, it will keep the right content.

## Example of a User Story With Acceptance Criteria:

Customer would like to have an email sent to my normal email address when his account goes into overdraft so that I know that I need to put money into my account.

***Acceptance Criteria:***

| Input | Process | Output |
|---|---|---|
| Valid Email Address | Email Validation | Message sent to email address |
| Invalid Email Address | Email Validation | Flag online profile as incomplete, kickoff snail mail message. |
| Valid Email Address | Marketing Messaging | Marketing message copy matches copy provided by marketing |
| Valid Email Address | Marketing Messaging | Marketing message design matches the specs provided by marketing |
| Valid Email Address | Marketing Messaging | Message contains email link that allows the user to navigate to online banking |

| | | |
|---|---|---|
| Valid Email Address | Email Validation | Message sent to email address |

In the above example, Acceptance criteria are a set of statements that represent the requirements “conditions of satisfaction”. It also contains boundaries and parameters that determine when a story is completed and ready for acceptance. It expressed clearly in simple customer language without any ambiguity on what is expected as outcome. It must be easily actionable and translated into one or more manual/automated test cases.

When the development team has finished working on the user story they demonstrate the functionality to the Product Owner, showing how each criterion is satisfied.

## Creating Acceptance Criteria

Acceptance criteria consists of 3 parts: input, process, and outcome. A useful way to think about acceptance criteria is: “When I <input> X and <process>Y, I will check for <outcome>Z as the result”.

The **inputs** of acceptance criteria are things like “entering a value and pushing a button” or “entering a command and checking results”

The **process** of acceptance criteria is the actual computation being checked. Usually when we create a user story, we want something to happen for a given set of inputs by a user. That process, while not usually directly observable, is verifiable for a given set of inputs and expected outputs.

The **outcome** (results) of acceptance criteria should always be testable with minimal ambiguity.

When people think about user stories, they usually think in terms of the user story description. However, the user story is not complete until it has verifiable acceptance criteria. Acceptance criteria also help the team quickly size a user story, because once they know how the story will be verified, they understand they effort needed to make it happen. Use acceptance criteria with every user story.

# User Stories vs. Requirements


How user stories are different from requirements

How to create user stories from requirements

# How User Stories are Different from Requirements

I was packing up my things, ready to leave after a long week of training. Susan, the lead business analyst, had been dutifully attentive all week. She took excellent notes, asked probing questions, and seemed to be integrating the new knowledge smoothly.

"Hey Tirrell, I have a question for you…"

"Shoot"

"So exactly how are user stories different from requirements?"

When people say "requirements", especially in my neck of the woods, they are referring to IEEE Standard 830 "The system shall" style of writing requirements. Here are some examples:

- 1.2) The system shall allow a user to buy a cake with a credit card
- 1.2.1) The system shall accept Visa, Mastercard, and American Express
- 1.2.2) The system shall charge the credit card before the cake is made
- 1.2.3) The system shall give the user an order number

You get the idea. One problem with documenting requirements this way is that its very very tedious and time consuming. Another problem is that its boring to do, and even more boring to read, which is why long detailed requirements documentation rarely gets read. Lastly, because requirements are written at such a granular level, its difficult to understand the big picture.

## User Stories are Goal Oriented

Traditional requirements are created from the perspective of the system and its associated activities. Problems with interpretation of these activities lead to pain and missed deadlines because they tend to be interpreted as edicts rather than

points of discovery. In other words, they tend to over specify. This leads us to a situation where highly paid intelligent engineers don't get the opportunity to solve user's problems, they are stuck implementing a sub optimal solution.

Unlike traditional requirements, user stories tell is what the **user** is attempting to achieve. This is important because it gives context to how we view the requirements. Since there are multiple ways to help a user accomplish the goal, it ensures the solution meets the goal (or the problem the user is trying to solve).

### User Stories Allow for Quick Level of Effort Estimates

In waterfall, there is a traditional requirements gathering phase that may be 20-30% of the overall project timeline. Therefore, the team doing the work can't give estimates until after the specifications have been written. This leads to friction because the teams are often railroaded into a timeline that doesn't match with the requirements, but the timeline and budget were decided before the specs were written. Catch 22.

With User Stories, a team can look at what the user is attempting to achieve, and give an associated estimate usually within minutes or hours rather than the weeks or months associated with traditional requirements.

### User Stories Allow for Negotiable Scope

Traditional requirements tend to be "all or nothing" and have no sense of prioritization. Therefore, the scope of traditional requirements is not negotiable. This can lead to what is known as "value engineering", which is the practice of creating quick, but fragile solutions as to get all the functionality completed on time.

With user stories, if the level of effort estimate is different from what the project sponsors thought, the team and product owner can take a look at the goals of the users in the stories, and think of other, less feature rich (but still usable) ways to help the user achieve their goals.

### IEEE Standard 830 was last updated in 1998

A lot has happened in software development since then, but many companies have not yet revisited how they create requirements, so I still see a lot of this in

the field. Often, its because a decision was made 15 or so years ago to write the requirements this way, and armies of business analysts were hired to support this decision. In short, continued use of IEEE 830 style specs is just as much a function of inertia as it is conscious decision making.

How do you write specs at your company?

# How to Create User Stories from Traditional Requirements

When I coach teams, many times, my attention goes to the requirements analysts, because they feel discomfort trying to understand the difference between a user story and traditional requirements. I was talking to a requirements analyst last week, and he was complaining, “My company started to go the agile way, and I was told to write requirements as user stories. I have no idea on how to write user stories. Do we still need requirement documents? I see the concern expressed is one of the most common one that makes people nervous.

It is hard to figure out how to break the big requirement specifications into smaller chunks of work doable in iteration.  Here are few tips and tricks that help to slice the big requirements documents to user stories.

Traditional requirement documents have features specified by each module or milestone.

## Tip1: Pick one feature at a time and prioritize

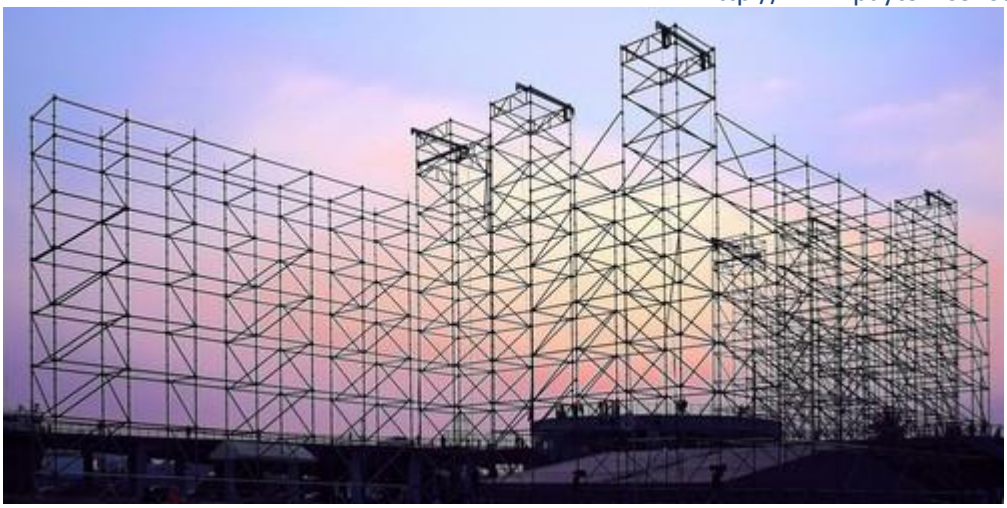
Take each module or big feature from the traditional document, and understand

- Who is the user of that functionality
- What is the purpose of that requirement
- Why does he need that functionality

The "Why" in the last point will tell you the actual business value the end user gets from the feature. It also helps you to assess whether the end user really needs that feature. If you see that there is no real business value in building feature, then push it to the bottom of the stack. The aim is to identify the most important big rocks that are useful for the end user to build his system.

## Tip 2: Break the big feature into small chunks

Now this big rock cannot be obviously built in a 2 weeks or 4 week iteration. It needs to be broken down into smaller pieces. Start splitting the requirements into smaller pieces. As you identify each smaller requirement, try to come up with the functionality aka the acceptance criteria that tell us whether we are building the right feature.



### Tip 3: Support it with other artifacts

One myth that many have in their minds is that a user story and acceptance criteria is only a bulleted list of sentences. However, that is not true.

**While writing the user story and acceptance criteria, it is Okay to support it with all additional artifacts that are needed**. For example: any wire frame, business rule document, any architecture diagram etc. but ensure you attach the bare minimum things without investing lot of time and the developer understand the intent of the story.

### Tip 4: Make sure the requirements are clear

- Ensure that conditions of satisfaction for each story is identified aka the acceptance criteria; That helps the team build the right functionality.
- Write down the tests that can be run verify whether the right functionality is really implemented, aka the acceptance tests, that confirm the acceptance criteria is met. Running these acceptance tests would qualify the story as a DONE and accepted story by the end-user.

- Map dependencies across the stories and have them specified in the same user story for reference. As an example, the "send email" story is dependent on "create email" story.  Ensure the top level feature is outlined, i.e the big rock, so that we know what we are looking for once all the user stories are created.
- Outline any assumptions around how will you demonstrate the small rock to the end user, for example any test container that you may want to use until the actual test environment is ready.

## Tip 5: Repeat and rinse

Repeat all the above steps until a feature in your traditional document is covered.

**Decomposition is the same, you are just making it outcome driven instead of activity driven**

To summarize, take the a big requirement specification document, break it down into feature, break it down into smaller requirements, prioritize them, take the most important ones from top of the stack, capture enough details in the form of acceptance criteria aka the condition of satisfaction. This way you are nailing down on the most important requirements, releasing them to team early to build them, so that ROI can be realized faster.

# User Story Gathering and Communication

How to run a user story workshop

How to explain and present user stories

# How to Run a User Story Workshop

Running a hands-on user story workshop is one of the most valuable skills a product owner can learn.

## 7 Steps to a successful User Story Workshop

### Step 1: Form a group of 3-5 people who understand the purpose of the product

3-5 seems to be the magic number. Any less and you might miss some ideas. Any more, and it slows the process down as well as diminishing returns on the quality of ideas generated.

Let the team to come up with a vision of own dream product. Propose to them that they write high level features for the vision. Clearly explain what you mean by a feature. (Example: Login page for a portal is not a feature). Take each feature and identify the high level requirements on color sticky notes.

### Step 2: Introduce the phrase “EPIC” and tell teams to break features into epics in different color sticky notes

Make a wall map and help the teams to paste the epics exactly below the features. When teams establish features and epics relationship, I would work with them to make team write high level requirements for each epic. Introduce User Story with an example and its intent. Help teams understand various formats of

User stories. Convey the importance of identifying User Personas at this stage. Show some examples of splitting user stories. Give few guidelines of dos and don'ts as well as pitfalls and traps when writing user stories.

### Step 3: Start the exercise by asking teams to write high level one liner as requirements for each epic in silence

Each person takes the same colored post-it and silently writes down one user story per post-it. Once everyone has finished writing their post-its, have each person read their post-its aloud. If anyone has duplicates, consolidate.

Depending on the size of the epics it can take 3-10 minutes to get all the major user stories. You can watch the body language to see when teams are finished. You can observe that the group goes from huddled in at the beginning to standing/leaning back at the end. It is likely that each post-it starts with a verb. (eg. Compose E-mail, Create Contact, Add User, etc) which forms the "walking skeleton" of the map. Ask teams to stick all user stories exactly under the related epics. This might be their first 'aha' moment for silent brainstorming.

### Step 4: Ask the team to group the sticky notes in silence.

Ask team to move similar things close to each other and dissimilar moved farther. Use silent grouping because it is faster than grouping them out loud. If duplicates are found, remove them. Groups will form naturally and fairly easily. Once again, body language will help you see when they are done – usually 2-5 minutes.

### Step 5: Introduce acceptance criteria with an example

Help teams write acceptance criteria for individual stories. Now talk about the non-functional requirements. Ask the team to come up with non-functional requirements for the same stories. Arrange all the stories on the wall, and help teams order them, ask them use slice the stories either by date or scope.

### Step 6: Explain sizing of stories using story points and help teams size all the stories

The product owner explains the sizing constraints and facilitates a story points sizing exercise using planning poker with the team.

## Step 7: Take all the user stories into the first release, then start slicing stories to make them as thin as possible

This is so that the stakeholders get a solid understanding of vertical slices, and so that we will more accurately be able to measure progress.

I see there is lot of value and motivation when team s to come up with their own vision and write down the stories. The essence of the workshop may be lost when I give pre-cooked user stories to the team. What do you think?

# How to Explain and Present User Stories

When I coach teams and their product owners, one of the first areas I want to help them correct is the backlog refinement sessions. Most of the time these are hours-long grueling meetings where the teams are wondering, "Why can't we just go code?" Just like many things, the pain that people feel in a new process is because of inefficiencies upstream. In this case, the product owners often have no idea how to present the information to the team to be able to size, refine, split, and ultimately deliver the story. In this article, I will explain "How to Explain and Present User Stories."

## Step 1: Present the background and business case

A big mistake I see product owners make, especially with new teams, is to jump directly into reading the story to the team without giving background. In the case of a project that already has some momentum, this us understandable. You certainly don't want to reinvent the wheel. However, for teams that are starting out a new project, it is very important that you present the background and business case associated with this project.

Presenting a background and business case does not mean reading a 40 page MRD or PRD to the team. It means helping them understand the problem from a big picture perspective and the solution to fix the problem.

### Step 2: Present the problem and feature area

Problems that are big enough, complex enough, and important enough to require a software team generally have "sub problems" associated with them. These sub problems have feature areas to solve for them. The sub problem and feature area are important because its another step down in granularity. This gives additional context to the conversation.

### Step 3: Present the user story and acceptance criteria

Now is the time to present the user story, which is fine, because the team has mental boxes to put it into:

- What is the big picture problem we are trying to solve
- How have we decomposed that problem into smaller problems
- What is the specific small problem we are trying to solve with this user story in particular

Reminder: When you present a user story, make sure you have a clear idea of who the user is. Use personas to outline basic information about your different user types, such as age, pain points, general goals, and level of comfort with technology.

As you present the user stories, you will also need to present the acceptance criteria for the story. Invariably, the team will have questions about the acceptance criteria, specific use cases, and other scenarios not outlined in the user story. Make sure you (or your Scrum Master) update the story with this additional information. The conversation is the most important part of a user story.

### Step 4: Ask the team if they have enough information about this user story to size it

The first couple times, the team will be taken aback at the directness of this question. But this question is very important to the progress of the meeting. As

new stories and requirements are presented, sometimes the conversation degrades into an architecture discussion, or a trip down memory lane reminiscing about the last time they developed a similar feature. This is fine, but left unchecked, the conversation will go on forever. So you will need to ask the team every so often "Do you have enough information about this user story to size it?" If they don't, your next question should be "What else do you need to know about this user story to be able to size it?" Continue to ask this until you get a reasonable size.

## The big picture

As I always like to say, "Context is everything." The difference between how I suggest product owners present user stories and how it's done in real life is that my method gives full context. If we want to leverage the collective brainpower of the team, everyone needs to understand what kinds of problems we are looking to solve, for whom, and what is the business impact. That will enable us to make smart decisions and tradeoffs when it comes to delivery.