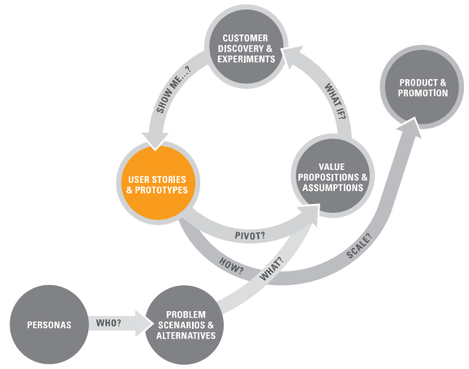
COWAN+

YOUR BEST AGILE USER STORY

Table of Contents

This article will show you how to create better user stories. You'll learn how to write them like a designer, test them like an entrepreneur, and use them to drive better discussions like an agile coach.

THE VENTURE DESIGN PROCESS
You are here.



OVERVIEW

GO DEEPER:

Want to make innovation an everyday thing?

RECOMMENDED:
Agile Development Specialization on Coursera

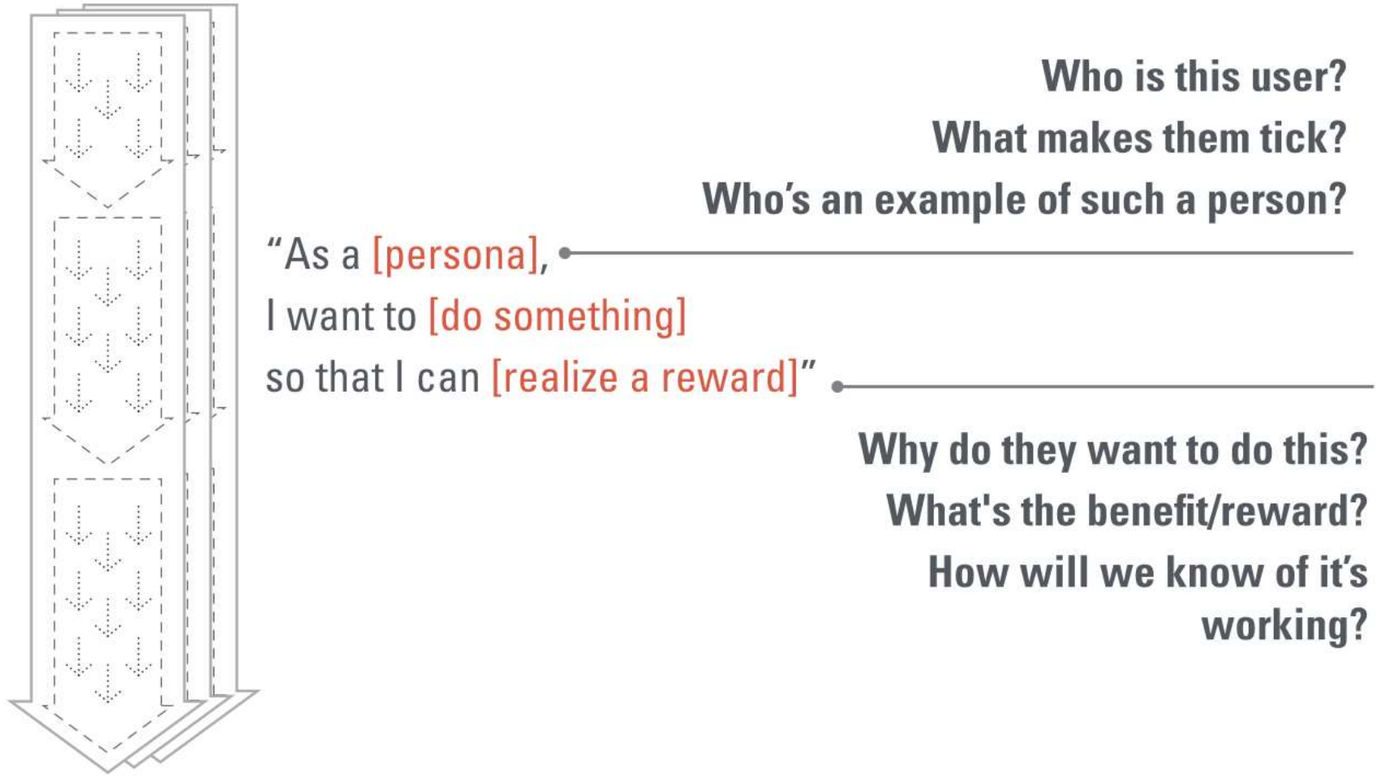
Creating a User Story

What is a user story?

User stories have a specific format:

> "As a [persona],
> I want to [do something]
> so that I can [realize a reward]"

This format is designed to help the story writer be descriptive and to drive better discussions about implementation with the rest of their team. Done right, the format helps prompt the following important questions:



If answering those questions seems like a lot of work, I'll mention here that taking

stock of new product failure rates, IT project failures, and the portion of features that actually are substantially used, something on the order of most or at least 'a lot' of software ends up lightly used or quickly scrapped. Skillful use of stories will improve your outcomes, your economics, and, most importantly, your team's sense that the work they're doing matters to the user.

If you're writing a story, your job is the fill in the (red) blanks. The persona is a vivid, humanized, yet operational description of your user (see also personas tutorial). The '[do something]' is a goal you assume the user has. The '[realize a reward]' clause is a *testable* statement of how you'll know if the user realized that goal. The [realize a reward] clause is the most neglected by most story writers and yet it's the most important. The single best litmus test for whether you have a good story is whether you could put a simple prototype in front of a user, ask them to act on the goal you're assuming, and see if they can achieve it.

Let's look at a specific example to see how that would work in practice. Enable Quiz is a fictional company I use in a lot of my examples. They're (fictionally) building an app that allows an HR manager to screen job candidates for a specific, technical skills sets relative to a job description. We might describe the goal of building a quiz to screen for a new open job description with this story:

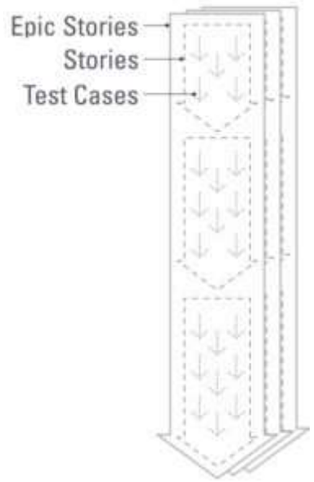
"As an HR manager, I want to match an open position's required skills with quiz topics so I can create a quiz relevant for candidate screening."

With this story, a designer or developer (or someone acting in that role) could put a sample job description in front of an HR manager, give them the goal of picking quiz topics for it, and see if they're able to realize that goal with the team's current prototype or working software.

There are schools of thought that the specific story format or the third clause [realize a reward] isn't that important. That just isn't consistent with my experience. Every time I talk to a team that's loose with story format, I'm seeing data that indicates they would benefit from front-loading value and driving to more specifics about user goals so they iterate to something valuable and avoid waste.

Pro Tip: There's a tendency to want to build software that's 'not specifically wrong' versus specifically right or specifically wrong. Resist it. Successful teams build in small batches with careful observation about whether they're right or wrong, but they build against tightly defined assumptions about the user's goals.

How do you organize stories?



Epic stories describe a user action and the child stories detail that action. Test cases and notes are a place to add further details and points to discuss relative to individual stories. Both types of stories have the same format- the format you saw above.

There's a school of thought that epics are really big, and they certainly sound big. My general advice is to keep them relatively small. I do a lot of training on story writing workshops and 95% of the first time child stories I see are actually what I would start with as epics (and the draft epics are overly broad in scope). Why? Well, developing and operating software is expensive. You might as well have good narrative coverage with stories on everything you're building to drive better discussions about what will be valuable to the user and how you'll test it. The alternative is to just guess and I can tell you how that works out- not well.

Let's look at a specific example. Returning to the narrative of an HR manager who wants to create a screening quiz for a new job description, we might start with this epic:

"As the HR manager, I want to create a screening quiz so that I can make sure I'm prepared to use it when I interview job candidates."

The child stories and their test cases them decompose that epic into more detailed steps (see Reference A for the full example). By the way, if you have the urge to start drafting, good on you and here is a Google Doc's template you may find helpful: User Story Template.

User stories are a type of specification, right?

NO! They're supposed to serve as a centerpiece for frequent interdisciplinary discussion about how to implement something valuable to the user. This is kind of where the use of stories and the practice of agile need to converge. If you're not familiar with agile, I recommend the aforementioned link or, hey, my agile course, but the basic idea is that instead of working in a sequential (waterfall) way from specification to development to test to release you're doing all of those things together in small batches. In the next section, we'll step through how to create stories like a designer, but I'll close with a few cautionary notes about what to watch out for with your user stories.

Why are most user stories so bad?

1. **Mistaken Belief that Software is Automatically Magic**
   I know it sounds silly, but you're probably working with people that implicitly think this to a degree. It's true- amazing things happen with software. But that software wasn't automatically magic; someone had to put the magic in it. That someone is you. The next section will show you how to conjure that magic.

2. **We Measure Our Efficiency Locally**

   To a degree, we all just want to finish our work and go watch Netflix. That's fine. That's natural. But, if you want to build great software, you have to want to consistently ask the hard questions of your colleagues and your work about whether what you're doing is still making sense. Is there a strong, testable case that it's going to valuable to the user? That's hard to do consistently. A lot of product managers just slam user stories into JIRA and call it a day. It's not that hard to run a more thoughtful, disciplined program, but it's not that easy either. You have to really want it.

3. **No Data**

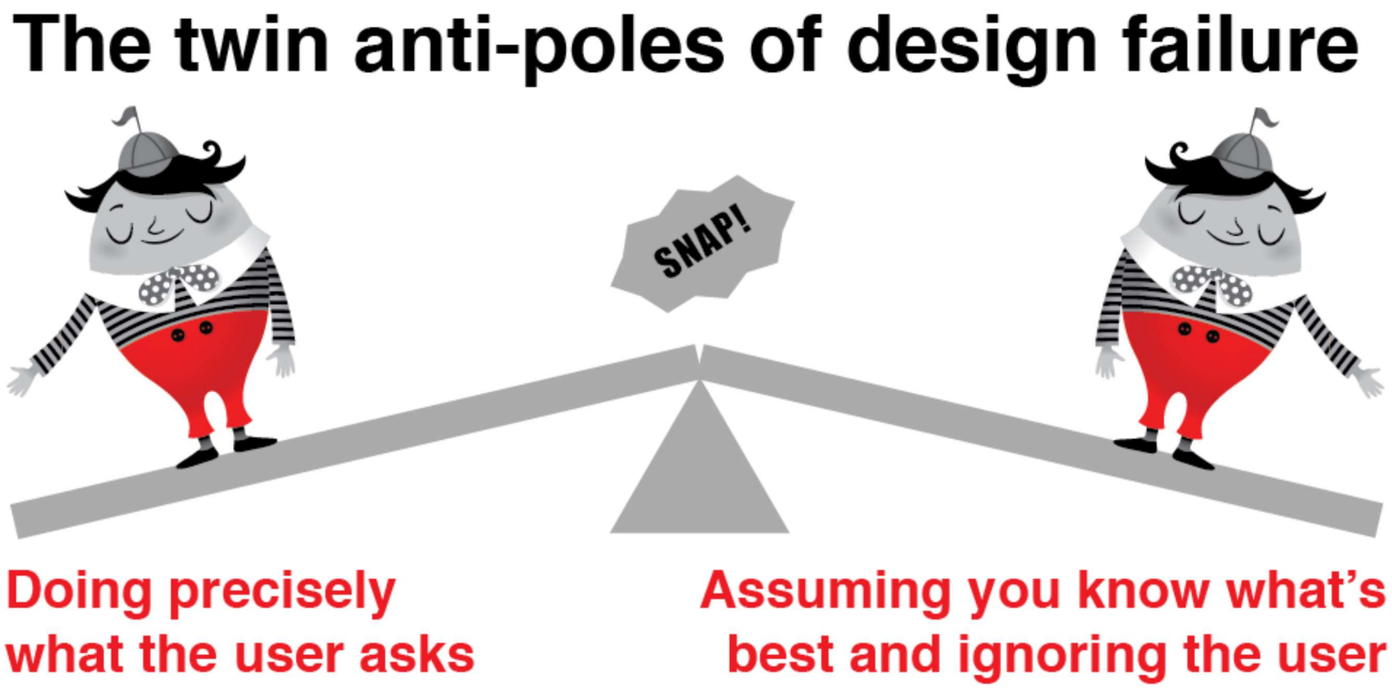
   You have to know your user and what's on their A-list to write good user stories. Without that, you're just going to be groping at some idea about what the software might do and then spending way more money than you need to to find out if you're right or not. There's an easier (and way cheaper) way, and that's what this next section is about.

Knowing If You Have a Story

Who are we talking about?

Remember when your high school English teacher told you to 'Write about what you know?'. That was always hard for me in high school because, really, I didn't know much. Likewise, if you don't know your user, writing good stories will be hard. This is not to mention that now what you're writing really matters- you're about to define lots of spending on creating and operating software.

The alternative is to end up on one of these two anti-poles of design failure:



Do precisely what the user asks and you'll probably end up in what my friend David Bland calls the 'Product Death Cycle': a) no one uses the product b) you ask customers what features are missing c) you build these 'missing' features d) go back to 'a'. Assume that you already have all the answers and you'll probably end up with a market size of close to one (you and maybe a few people just like you).

What's a conscientious product person to do? The good news is that there's an approach that consistently works. The bad news (it's not really bad ;)) is that it takes some work of a type that may be new to you. The tool I'd start with is the idea of 'personas'. These are humanized views of your customer, be they buyer and/or user of your product. It turns out this is the most actionable and testable way to build products and systems.

The basic idea is that you want to name the persona to humanize them (‘Helen the HR Manager’) and have an idea about a day in their life. What shoes do they wear? What’s the first thing they do in the morning? Then you want to dig into how they relate to your product or system. What do they Think about how things work now vs. how they’d like them to be? What do they See others doing that influences that perspective? How do they Feel about the job or habit you’re building software to improve? What do they actually Do in your area of interest? How much? How often?

You’ll probably find you need to go out and talk to a few of them to obtain this information, but if you have a strong Think-See-Feel-Do understanding of your user, I think you’ll find stories are a lot easier to write and to discuss.

For more depth and some examples see: Personas Tutorial.

For notes on how to interview users to create personas see: Customer Discovery Handbook (Persona Hypothesis).

For a structured tour of these, see Agile Meets Design Thinking on Coursera and the Personas section of Running Design Sprints.

What do they want?

Once we have a strong persona or personas, we need to consider and test what underlying jobs or desires we’re going to make better for this user. Do they really exist? Are they important? This is worth exploring since if you’re solving a problem that doesn’t exist, you can develop perfect software and still fail to deliver any value to the user.

How do you identify and describe these problem scenarios? First, simply consider what it is that you're actually doing for the user- helping them stay connected to friends? checking their accounts payable for suspicious patterns? Then make sure you can describe an alternative (ideally you've talked to users and already know). Maybe your user sends out a holiday card to stay in touch with their friends. Maybe they just look at large accounts payable and spot check/audit them. Make sure your definition of the problem scenario is general and could apply equally well to the current alternative as it could to whatever it is you're planning to do to make things better. This will help you avoid bias and stay focused.

Let's look at a specific example. For Enable Quiz (see above), their core problem scenario of interest might be:

'Screening technical talent.'

I said a good problem scenario can be applied equally well to the alternative or your new proposition. Helen the HR Manager's alternatives for screening technical talent might be: checking resumes, calling references, or just taking their word for it. I'd say the problem scenario definition works for those alternatives as well.

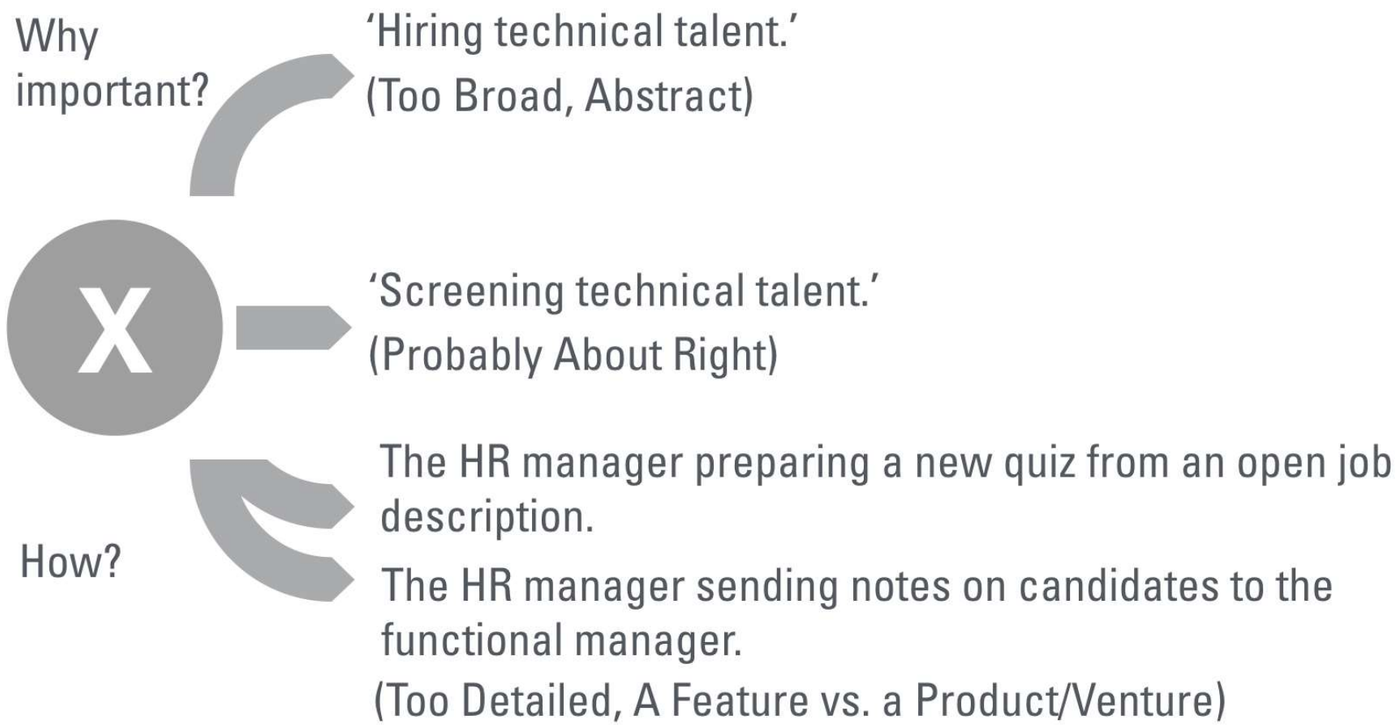
What if we took the problem scenario and went up a level of abstraction? Then it might be something like 'Hiring technical talent.' Now, that's probably too broad, but it's helpful to know so that a) you know you're at about the right level of abstraction with your core problem scenario and b) that you've identified the general area in which you're working.

What if we went down a level? Then we're probably get lots of more tactical problem scenarios like:

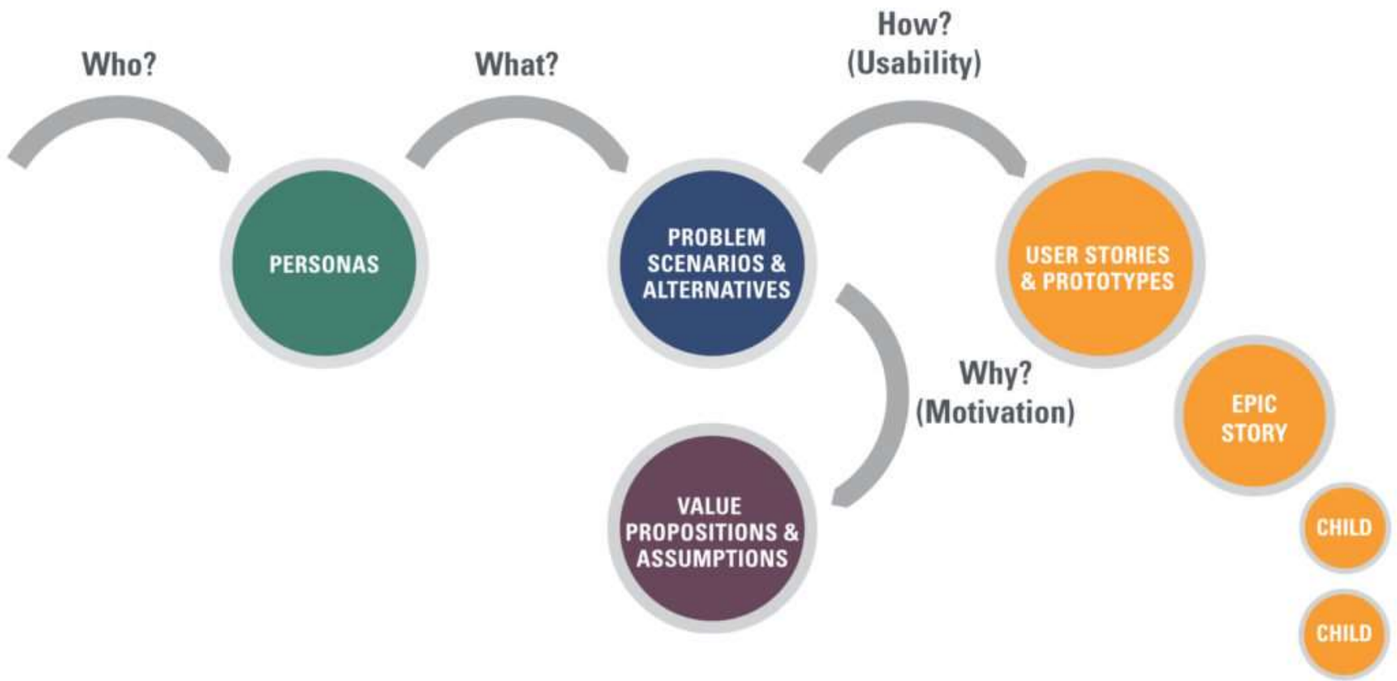
'The HR manager preparing a new quiz from an open job description.'

'The HR manager sending notes on candidates to the functional manager.'

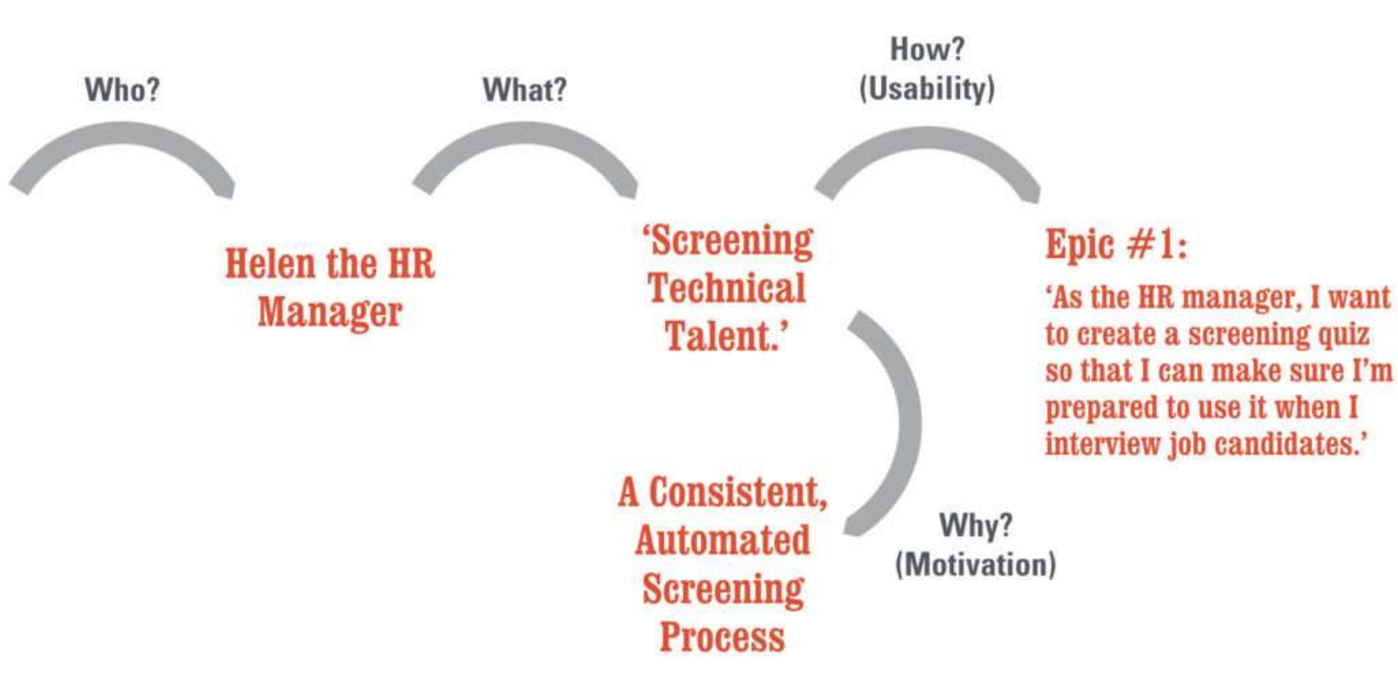
These all flow from the core problem scenario on screening, but we'd still want to make sure they're important before we start writing stories and consider investing in software. A summary of the first few problem scenarios for Enable Quiz might look something like this:



When you work forward to build up your understanding, the sequence goes something like this:



Here’s an example of what it might look like in the case of Enable Quiz:



Now, you may notice the note about ‘Why? (Motivation)’- I do also recommend testing whether the customer prefers your solution(s) to a problem scenario over their current alternatives. However, I don’t want to stray too far from the topic at

hand. If you're interested in the overall process, check out the Venture Design page.

For related tutorials, templates, and online courses, see the end of the section above on Personas.

Designing Better User Stories

What's the right level of detail?

Once you've learned about your user enough to know you really should build software, it's time to work through some stories. What you want to do to make their lives better? How will you know if you did that?

Your user stories should be detailed and they should be testable. Why? Well, because what you're really deciding is how much of the software/systems you're going to invest in building will be supported by carefully considered ideas about what's valuable to the user. Based on the stat's I cited at the intro. (that something on the order of half of all software ends up in a fail), I'd say odds are many teams have room for improvement. That's easy to say, but hard to do. I know I'm always working to do better. Let's take a look at a few specific examples.

If you've never written a story before, you'll probably start with something overly broad. For Enable Quiz, it might be something like:
'I want to properly screen engineering candidates so my company can get the best possible hiring outcomes.'
You could use this as a starting point for your next stories, but it's not immediately actionable for design or coding. Also, if you're only going to work with one

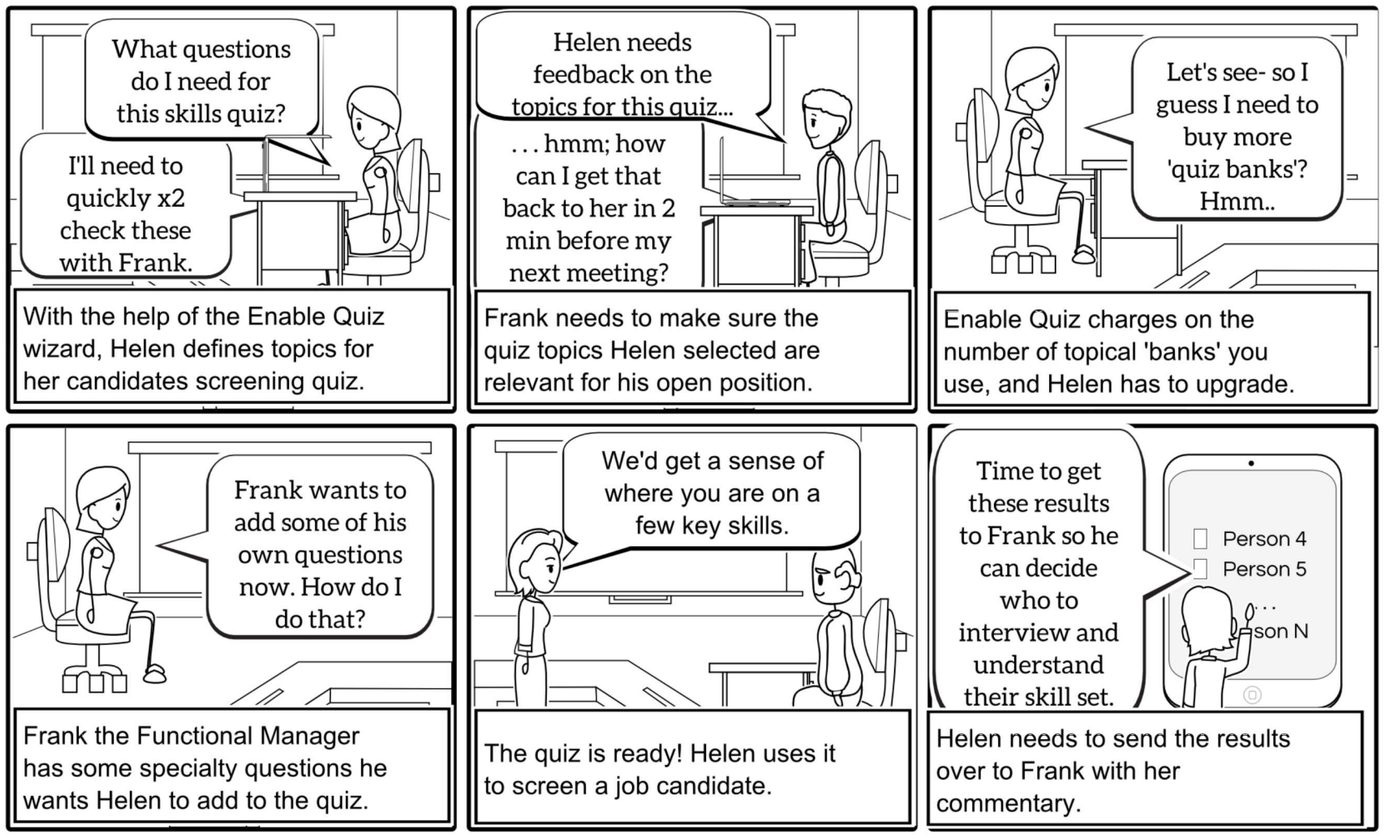
additional layer of abstraction in your stories (child stories under an epic), I think you're going to end up with stories that are too vague and a working environment that's not well backstopped by narrative. The story above is really analogous to the problem scenario we talked about in the last section: 'Screening Technical Talent'. For stories, I'd start with more actionable specifics (and tie them to problem scenarios if you want to make sure they're well supported by customer discovery).

Here's a more specific story:
'I want to easily screen an engineering candidate, so I know their skills profile.'
This story has some issues. Let's tighten it up! First, it's missing the first clause identifying the persona. That's an issue since there are at least two distinct personas involved: Helen (or Hank) the HR Manager and Frank (or Francine) the Functional Manager. We need to know who we're designing for and whether we should screen for subjects that match Helen or Frank when we're testing usability. (We may get very different results testing with Helen/Hank that has a background in HR vs. Frank/Francine that has a background in software). Here's an updated version of the story:
'As Helen the HR Manager, I want to easily screen an engineering candidate, so I know their skills profile.'

Do you see any issues with that story now? I always tell students to avoid normative terms like 'easily'. I mean, what does that mean to a designer? A developer? Did you think they were going to design it to make things hard? If you think particular moments in the customer journey are important in some particular way, capture that in a storyboard and/or get that up on a story map (see section 'Developing with Stories'), and really describe what's important and why and how you'll observe whether you're succeeding. (I know, designing thoughtfully is a lot of work!!) Let's remove 'easily' from the story:

‘As Helen the HR Manager, I want to screen an engineering candidate, so I know their skills profile.’

How about now? Is that about the right scope for an epic? I don’t know about you, but I have mixed results thinking that through in my head. I have to work through it on paper. One thing I find is really helpful for exploring an epic (particularly at the beginning of a new project) is to storyboard it. Here’s one view of what the major steps to the above epic might be:



Looking at this, I see a lot of stuff- creating the quiz, paying for Enable Quiz (if it’s the first time or requires a subscription change), consulting with a colleague (Frank), administering the quiz, and making the results available to Frank. I’d say just about each of those feels like an epic. What I think I’d do is start with an epic about just creating the quiz:

‘As Helen the HR Manager, I want to create a screening quiz so , so I know their skills profile.’

I’d say this is getting close, but that last clause about the reward, ‘so I know their skills profile’ still needs some work. That clause is so important, I think it deserves its own subsection! That’s up next.

Storyboarding is an excellent tool pushing yourself and your team to think substantially about your customers, what they want from you, and what they’re really doing in real life. Especially if you’re developing something relatively new, I highly recommend storyboarding your epic agile user stories.

The process of storyboarding will help you:

1. test the depth and scope of your understanding about the persona(s) and problem scenario(s)
2. stimulate interest and discussion in the story with your implementation team (and peers on the product side)
3. push you to think through the details so you’ve implementors don’t have to start from scratch there

This last one is particularly important if you’re new to product development. I coach lots of first-time entrepreneurs, and their #1 challenge at the product implementation phase is thinking through what they want in enough detail. Most of their stories are actually epic stories (or broader) and need more granularity and sequence- storyboards help. A classic failure mode is leaving your development team to design your product. This not to say your developers can’t think creatively or answer these questions, and part of the point of these inputs is to stimulate creative discussions with them so the team’s collective imagination

and expertise is maximized, but your job as the product person is to provide a well-articulated, thoroughly considered starting point.

For more on the process of storyboarding (including tools and templates), please see: Storyboarding Tutorial.

What's their reward?

The last clause of your user story should always contain a testable reward. Reward sounds exciting but all we're talking about is: a) does it describe the successful accomplishment of whatever goal first triggered the story and b) can observe in a user test?

Does the current 3rd clause ('so I know their skills profile') meet those criteria? I think we could test it- put a prototype in front of an subject, step them through creating a quiz, then administering it and seeing the results and ask them to tell us the skills profile. But is that reward even relevant anymore? It doesn't really make sense with the intent. What does Helen really want at this point? There are a few possibilities. Fundamentally, she wants to figure out, with the functional manager, who they should interview and who they should hire for the best possible (mutual fit). I'd say 'so that I can understand whether I want to send possible recruits to the functional manager' is a good candidate. Another more immediate possibility is 'so that I can make sure I'm prepared to use it when I interview job candidates'. Let's try that on for size:
'As the HR manager, I want to create a screening quiz so that I make sure I'm prepared to use it when I interview job candidates.'

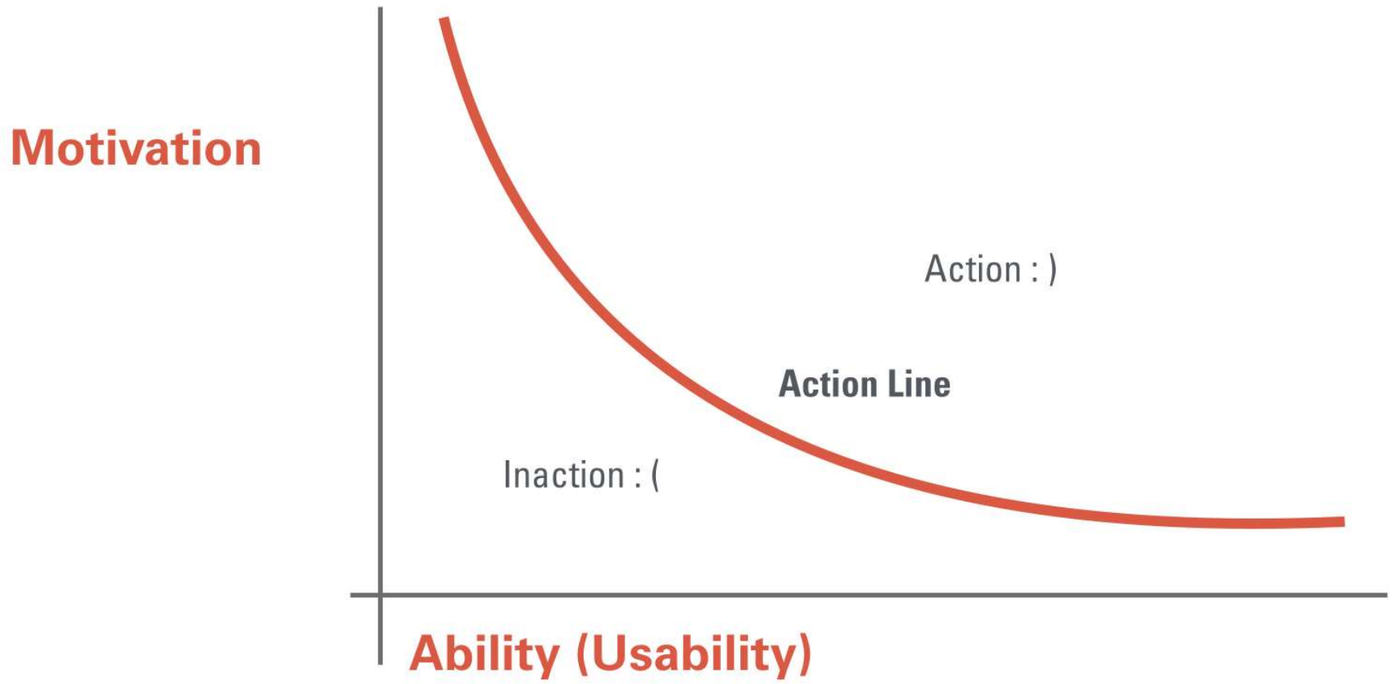
Now how does the clause do on our tests about a) describing their accomplishment and b) being observable in a test? I'd say it describes the HR manager's immediate goal- being able to use the test. Is it testable? We could

take the user through the creation of a new test and then ask them how they'd administer it and see if they have the right answer. I'd say it's testable.

Testing User Stories

What are we testing?

When it comes to testing, I think the most important thing is to a) draw a clear distinction between usability and motivation and b) understand the relationship between them. My favorite tool for dealing with this is BJ Fogg's curve:



source: adapted from BJ Fogg's Behavioral Model

Basically, what this is saying is that if the customer/user is really motivated to do something (imagine a point on the upper left), then it can be relatively harder for

them. If a user isn't that motivated (imagine a point on the lower right), then the action needs be relatively easier.

If you're building a product or feature, what this clearly implies is that you need to take a balanced view of motivation vs. usability. Most products and projects I see are very focused on usability. However, most of them haven't tested motivation: Will the user really prefer this product or feature over their current alternatives?

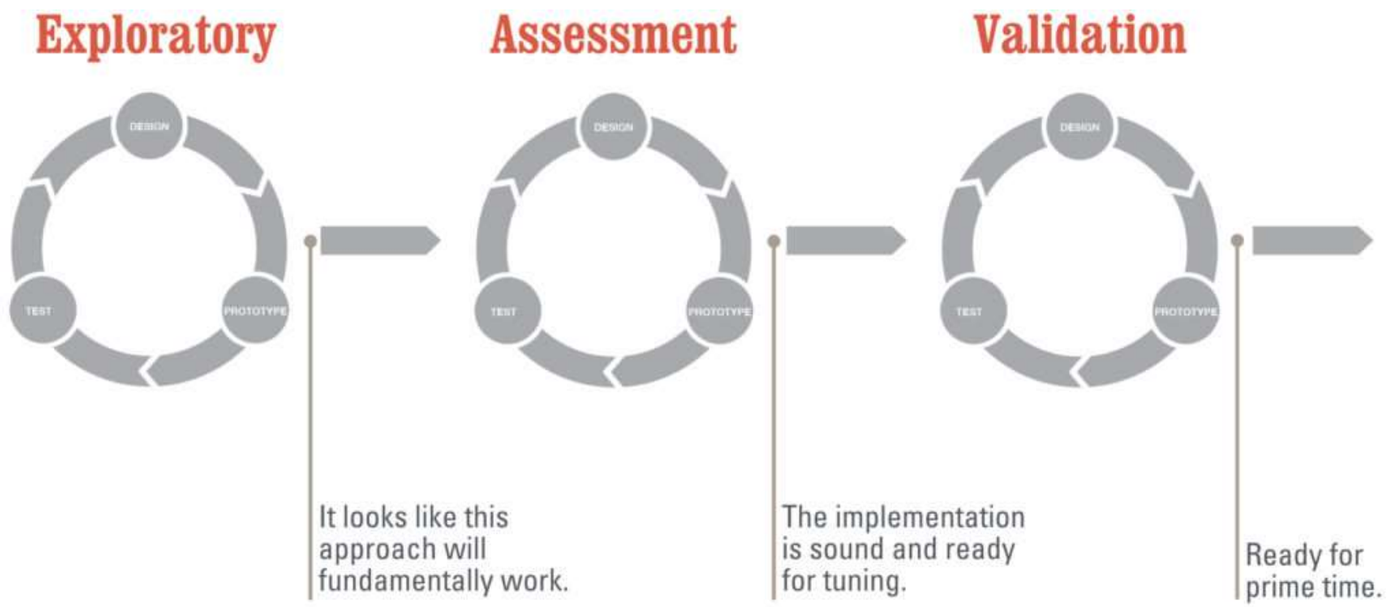
Instead, they often confuse it/mingle it with their usability testing, for example asking about a prototype 'Do you like this? Will you [use it, buy it]?'. That doesn't work- the subject will always give you a 'yes'. For more on why see specifically this post on the 'yellow Walkman problem' or more generally the here tutorial on Lean Startup. Suffice to say here that the kind of testing we do relative to user stories deals with usability- that's why they call it 'usability testing'.

What they means is that we're assuming they're motivated (controlling for motivation, if you will) and just testing for usability. What this means is that if we're testing a piece of software that allows users to paint virtual walls (bear with me), then we don't ask them 'Do you want to paint this wall?'. We don't even ask them 'What color would you like to paint this wall?'. We ask them 'Would you show me how you'd paint this wall yellow?' and then we're seeing how hard it is for them to do that and thinking about how we can make it easier.

How do we test stories?

Have you ever heard something like this: 'We don't do as much user testing as we probably should because by the time we can test the software we're already up against our deadline.' As sympathetic as I am to those deadlines, I just don't think this is a good reason not to be testing usability.

A generally accepted (though not universal) view of user testing is that it's best to think of it having phases like you see below:



When you're doing early testing, rough prototypes (physical or virtual) will work fine- you don't need an engineer or (with the right process) even a formally trained designer to test the efficacy of different approaches. My advice is to maximize these opportunities early on in your story formulation. On a related note, I highly, highly (2 highly's) recommend anchoring your test items in user stories as opposed to interface-specific goals like 'see if the user understands our drop down menu's'.

For a quick overview of how to do this, I recommend the Usability Testing section of the Customer Discovery Handbook.

For a more in depth view, I recommend Running Design Sprints (course 2 in my agile specialization on Coursera) and Testing with Agile (course 4 of the same).

When is a story done?

I think you'll find this is a valuable question to answer with your team. I'd also bracket it with the other end of the larger question, which is 'When does a story begin?'

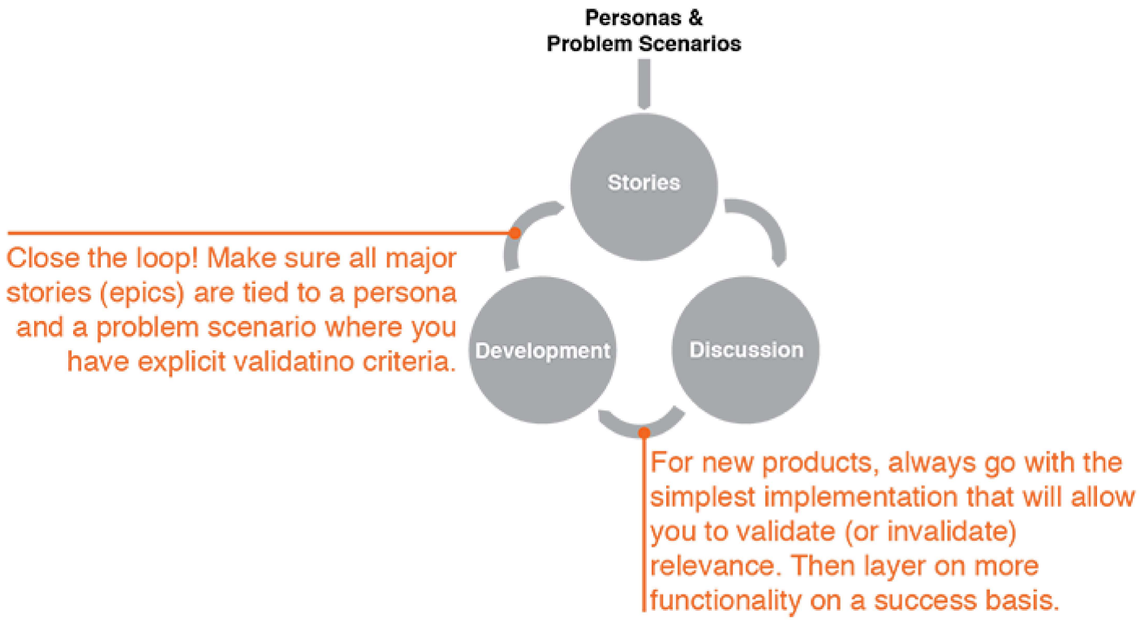
The prevailing thinking around modern software development is that you should treat your product, in fact, every feature within your product, as an ongoing experiment.

While it drew heavily on agile, it's really the body of work around Lean Startup that's popularized this. Their central heuristic is 'build-measure-learn'. I prefer the terms you see in the blue loop here but fundamentally they amount to the same thing.

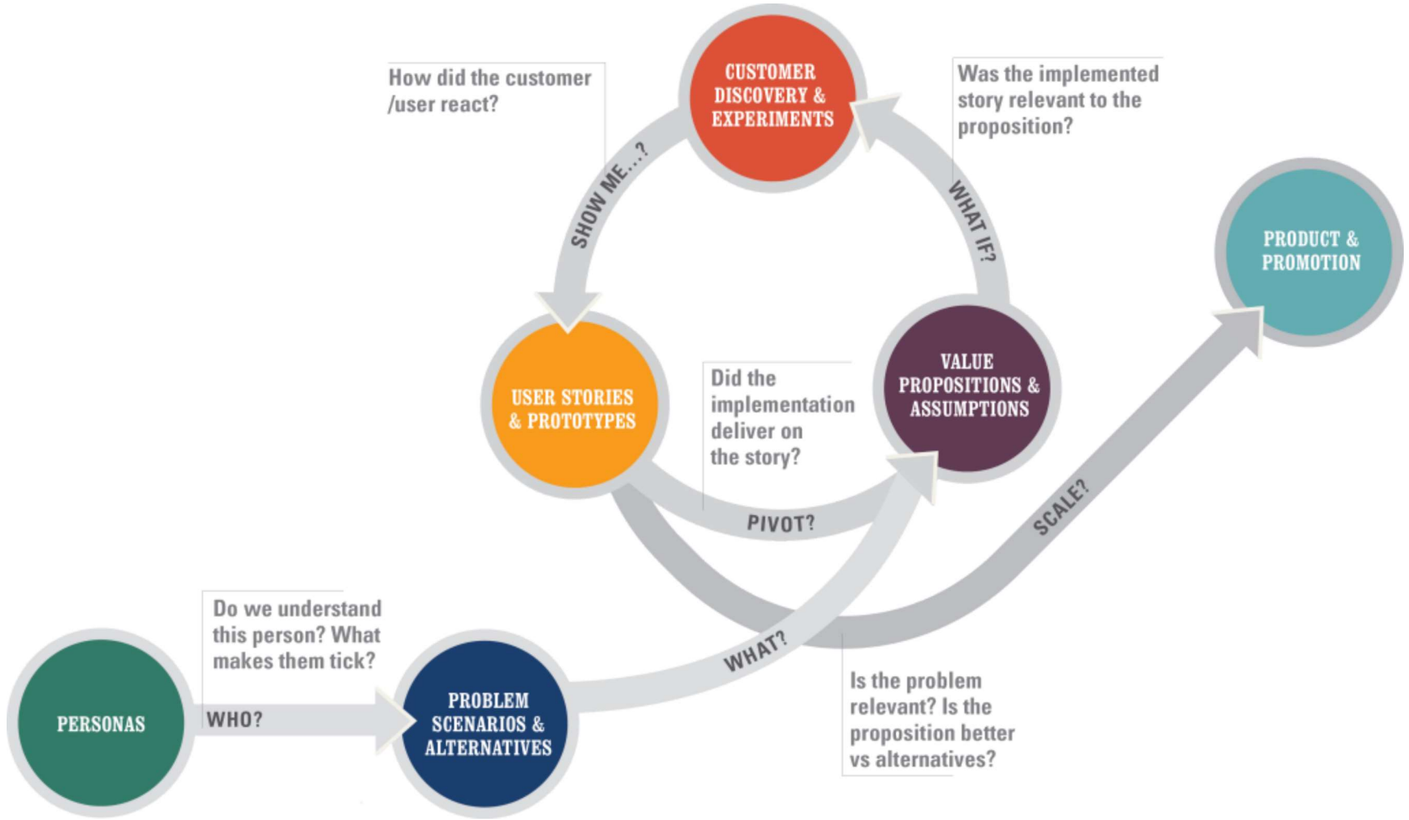
The beginning of any good answer to the question above begins with the acknowledgement that you just can't predict (with any consistency) what's going to be valuable for the user. Therefore, you need to work with testable ideas and closed-loop experiments to see if you're right or not.

Ideally your stories begin with some kind of observation about your user and the problems (jobs, desires) they have. Then you come up with a testable formulation of how you'll know if your solution idea is really better than what they have now. A very tactical examples from Lean UX is that you might put in a placeholder menu item or button for your new feature and just see if users click on it (at which point they get a 'coming soon message'). Once you've validated that some motivation

exists for your feature, you move on to actual user stories, testing usability as you go along. Then, once you release said feature into the wild, you have some metrics on usability and motivation that tell you whether, in fact, you really are right (and to what extent). Ideally, this feature you've implemented is the simplest possibly version (minimum viable product) that will allow you to learn whether you're right or not. Basically, you want to achieve something like this:



Yup, it's a lot of stuff. My particular approach to making it manageable and actionable is Venture Design, which is described in the diagram below. If you really want to dig in and make sure you're iterating to something valuable with a minimum of waste, I highly recommend taking a look. That said, let's move on to developing with stories!



Developing with Stories & Story Maps

Who writes user stories?

Odds are good that if you're doing agile you're using scrum or a scrum-informed take on agile. If you have someone in the 'product owner' (PO) role scrum prescribes, you've probably heard that it's their job to write the user stories. That is a reasonable starting point. That said, the best way of looking at it is probably that the PO is the lead on writing the user stories vs. solely responsbile.

WAIT! This is really important- it's not just touchy feely stuff. It might even matter more than the prescribed mechanics of scrum. Here are the specific reasons why:

1. The 1/40 Problem

Based on experimental results, it's possible (I would guess likely) that people are only actually understanding 1/40 of what we think we've told them. A PhD candidate at Stanford did an experiment where one person (a 'tapper') would tap out a song and another party ('listener'). The tappers guessed that they'd been understood 50% of the time where actually the listeners had actually understood only 2.5% of the time- off by 20x. Yikes!

What if that's true of the stories we're writing and the understanding we assume a designer and/or developer will have about how to build software against that story? Is that somewhat consistent with the kind of disconnects you see between 'businesspeople' and 'technical people'?

If you're slamming your user stories into JIRA and assuming they'll be understood, you're probably going to run into issues. By the way, I first read about this in the Heath brothers' excellent book 'Made to Stick'.

2. We Measure our Efficiency Locally

A certain part of us just wants to know in the simplest possible terms what our job is and how much of it we have to do. We crave certainty and an immediate sense of accomplishment. If someone tells me my job is to dig six holes, part of me just wants to do it and be done. It's fine- it's just human nature.

Unfortunately, you can't build software that's valuable to users this way (at least not with any consistency). If you don't co-create stories with your team, they're just not going to feel like they own them and their work won't be as thoughtful. Since the stories are probably your job I'll apologize in advance for saying so, but don't blame the implementation team- you'd act the same way in their position. Again, it's just human nature.

In the classroom, I often reference the ‘blue button moment’. In a low-functioning agile team where the implementors don’t understand and/or haven’t bought into the idea, it looks like this:

In a high-functioning agile team, it looks like this:

If you’re the lead on creating stories, part of your job is to solve for the two problems above and help elevate your team to a high functioning practice of agile.

As if that wasn’t enough there’s one more reason why you should write stories as a team-

3. You Don’t Have All the Best Ideas

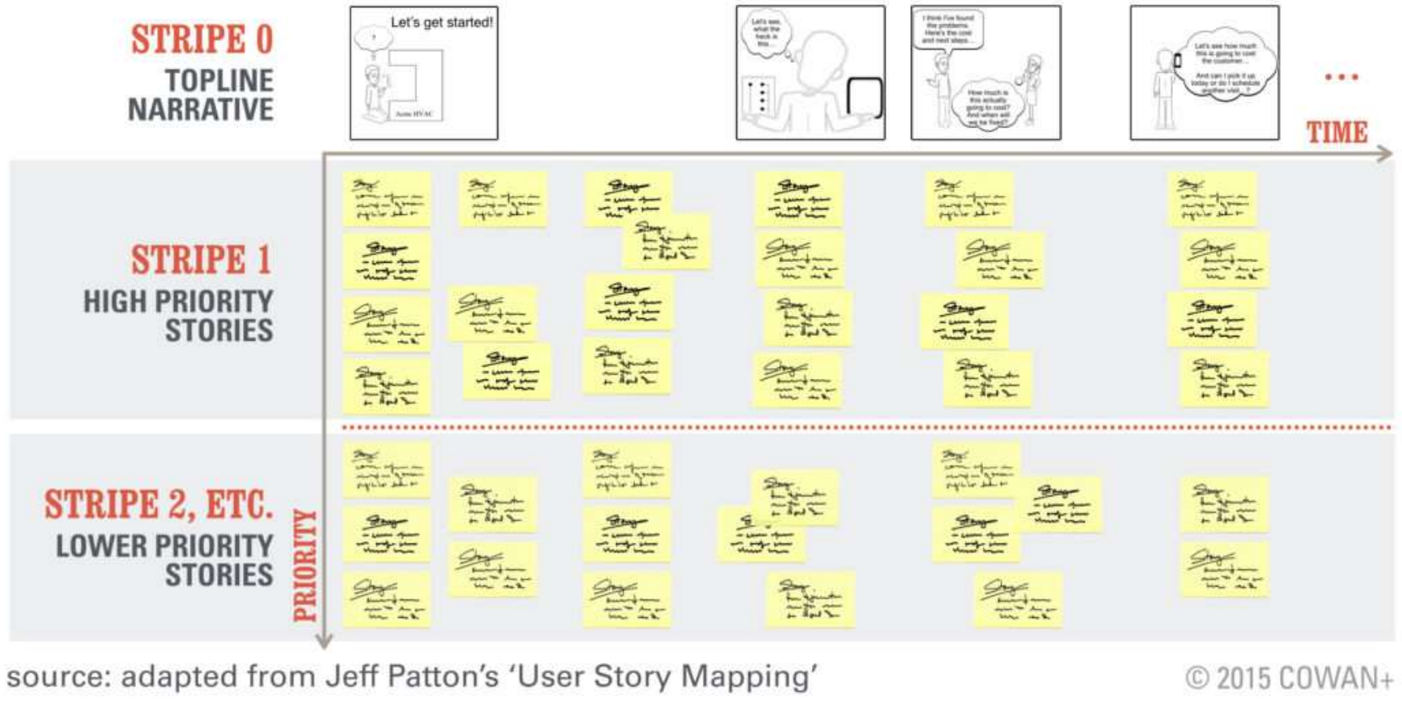
Naturally, others will have great ideas about what the user might want and how to

implement that in software. Beyond just doing your story writing with the extended team, this is the reason why the whole team is generally included in the increasingly popular practice of running design sprints.

One specific method of encouraging good outcomes around stories is having the PO/story lead run story writing workshops, which we'll cover in the next section.

How do you use stories within a team?

User stories are your anchor for working together on what you're going to build and how you'll know if it's working. In a perfect world, the whole team is conducting design sprints together to learn about that and cultivate a shared understanding. Regardless, to arrive at a strong shared understanding it's critical to formulate the stories together and make them highly visible to the team. On popular tool for this is the story map, popularized by Jeff Patton:



The story map does a few jobs for you: 1) helps you think through the right stories 2) helps you prioritize them and 3) serves as an ‘information radiator’ to keep the stories visible as you work on them over the course of weeks or months. If you’re in charge of managing your team’s story writing workshops, I think you’ll find it’s a very valuable tool.

Let’s talk about how to make one. The first stripe (0) defines the x-axis of the story map and represents the customer/user journey. It’s best built with storyboard squares. The reason why the seemingly silly details in the storyboard squares matter is that every single thing you can do to help keep your focus ‘outside the building’ and on the life of the user is going to help you drive to better outcomes more efficiently. Fine point: If you’re storyboard your epics (great idea!), then what you’d probably want to do on ‘stripe 1’ is to pick and choose among those to create a kind of summary. The reason this is a good idea is that a well articulated epic will usually have many more storyboard squares than you’ll want with the size and density of a typical story map.

Let’s talk about stripe 1+ and the y-axis. The y-axis describes relative priority, so stripe 1 is higher priority than stripe 2 and so forth. If you have a bunch of stories about how a user would search for a product, you’d put what you assume is the most common/important story in stripe 1, and then less common types of search stories in the same vertical space within stripes 2, etc. Careful prioritization on this axis relative to the x-axis/user journey is a subtle but important part of any high-functioning agile program.

The best explanation I’ve heard comes from agile thought leader Bill Wake: Let’s say you’re building a site where the basic user journey is a) search b) select c) purchase. It’s natural (but a terrible idea) to build out all the search functions and then move on to select, etc. The reason it’s a terrible idea is that you won’t be

able to do any meaningful user testing or market/release testing. The user can't do anything potentially valuable by just searching for stuff. In a high-functioning agile team, you're iterating through the thinnest possible slices of your story map, testing, assessing, and then building more software.

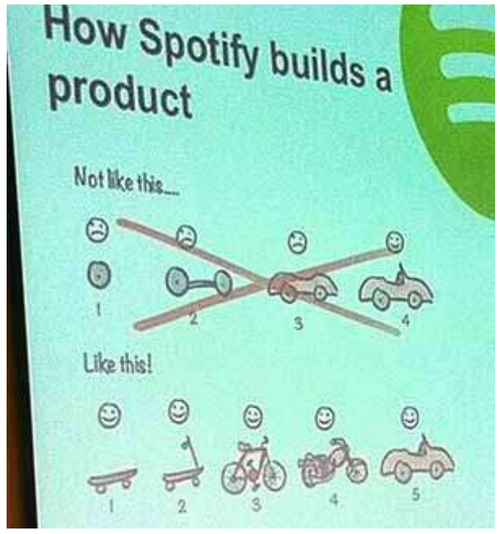
How do you use stories day to day?

In a high-functioning agile environment, when anyone is unsure of an implementation detail, they have a discussion around user stories- see the blue button moment above. I've shown you how to put yourself in a position to write strong stories that are anchored in the life of the user and highly testable. You've also seen how to organize and prioritize stories with a story map.

It takes practice to consistently write good user stories and apply them effectively to help your team work better. For overall breadth on how to think about this, my favorite heuristic on user stories is Bill Wake's INVEST acronym. Bill Wake introduced the INVEST mnemonic in his seminal post on creating better stories, suggesting they should be Independent, Negotiable, Valuable, Estimable, Small, and Testable. The INVEST checklist is a guideline, not a commandment, as Bill Wake makes clear in his post. When in doubt, always do what *you* think makes sense. All that said, here's an overview of INVEST:

Independent

Ideally, you should be able to implement your stories in any order while still realizing the benefit of seeing something working at the end. This is important for two reasons. First, you lose a lot of the adaptability agile's supposed to deliver if your stories have (many) interdependencies. Second, independence makes it much more likely that you can sequence what you do such that it's meaningful to users and testable with them by the end of an iteration.



Spotify is known for its success with agile and other adaptive techniques. This slide from one of their talks summarizes the value of Independent (and Valuable and Testable).

Negotiable

User stories are not requirements. It's a combination of stories and regular discussions that are meant to replace requirements. You should revise stories freely as you and your collaborators come up with better ideas.

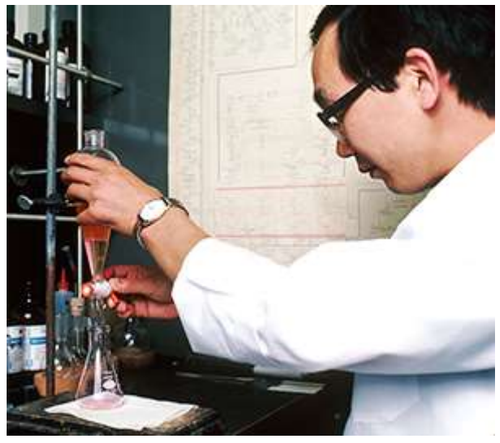
Valuable

Each story once implemented should be both functional and relevant to the customer/user. We've talked a lot in this post about how to achieve that.

A quote from Bill Wake's original post on why this is important in software development projects: *Developers often have an inclination to work on only one layer at a time (and get it "right"); but a full database layer (for example) has little*

value to the customer if there's no presentation layer.

Estimable



It should be possible for a developer with relevant experience to roughly estimate a story. That estimate's then used by the product person to prioritize their list of stories, taking account of both value and cost (in terms of developer time).

Estimating's kind of controversial in the agile community and not without good reason: It can lead unnecessary overhead and pointless recriminations when the final output doesn't 'agree' with the original estimates. These are all things agile strongly tries to help avoid.

If I were to offer three silver bullets on this, they would be: 1) Write good stories based on validated insights about the user. 2) Make sure everyone understands and agrees that the estimates are very approximate (like Small-Medium-Large) and for purposes of prioritization. 3) If you want to do post-mortems on estimates vs. actuals, make sure there's general agreement that it's to help the team in some actionable way and not just because someone's frustrated there isn't more implementation done.

Small

Your unimplemented stories are like a box of chocolates...ah, scratch that, reader; never mind.

Basically, your stories need to be Small for you to get the adaptability and reduction in overhead that agile offers. Big stories reduce opportunities for incremental testing and require more elaborate planning.

Testable

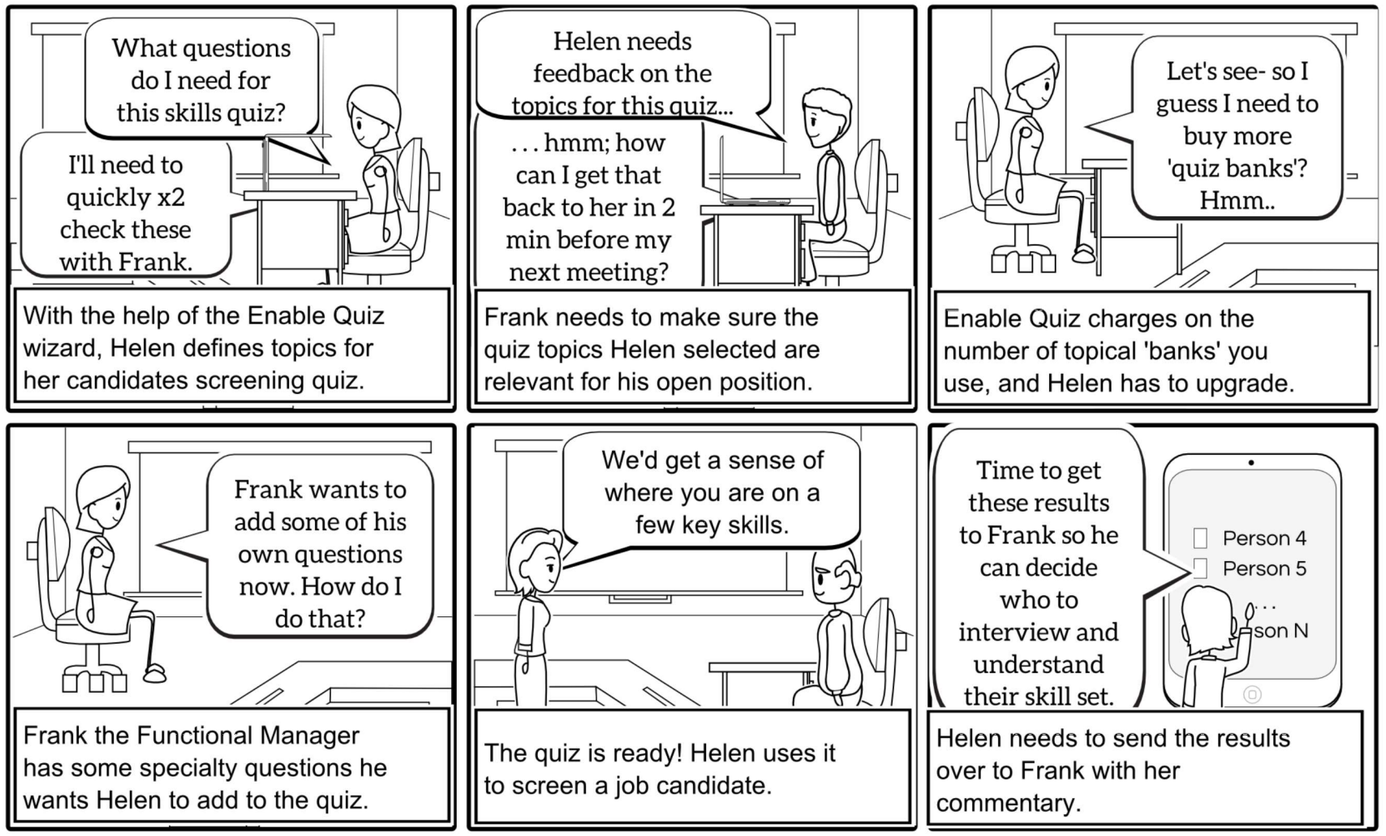
As the author of a story, you should be able to write tests that would allow you to validate that an implementation delivers on your intent. Many teams operate this way, but regardless of what everyone else does, you'll do yourself a big favor by writing some functional test cases in advance- you'll be much more likely to get what you want more easily.

Example A: Example User Stories from Enable Quiz (Startup)

Example Epic I

This epic story deals with the example company Enable Quiz and the HR manager wanting to create a quiz to screen engineering candidates. is organized in a more conventional fashion (vs. the epic above that's storyboarded).

Epic Story: "As the HR manager, I want to create a screening quiz so that I make sure I'm prepared to use it when I interview job candidates."



USER STORY	TEST CASES
As a manager, I want to browse my existing quizzes so I can recall what I have in place and figure out if I can just reuse or update an existing quiz for the position I need now.	Make sure it’s possible to search by quiz name Make sure it’s possible to search by quiz topics included. Make sure it’s possible to search by creation and last used date.

As an HR manager, I want to match an open position's required skills with quiz topics so I can create a quiz relevant for candidate screening.	Make sure the quiz topics are searchable by name. Make sure the quiz topics have alternate names, terms for searching
As an HR manager, I want to send a draft quiz to the the functional manager so I make sure I've covered the right topics on the screening quiz.	Make sure it's possible to add another user by email in this flow Make sure it's possible to include notes and customize the email Make sure it's possibly to just copy a link (for case where HR manager wants to send their own email)
As a functional manager, I want to send feedback on the screening quiz to the HR manager so I make sure I'm getting the best possible screening on candidates.	Make sure it's possibly to supply comments in-app. Make sure the above are quiz-specific by default but can also be general. Make sure it's also easy to copy the name or URL of the

	quiz for their own correspondence.
As an HR manager, I need to purchase an upgraded service tier so I can add additional topics to my quiz.	Make sure that users with billing privileges can upgrade the service. Make sure that If the users don't have billing privileges, they see a list of those that do and can contact them. Make sure the charges are correctly prorated against the billing anniversary of the account.
As an HR manager, I want to add custom questions to the quiz so we cover additional topics that are important to the functional manager.	Make sure the customer is not charged for this bank. Make sure the custom bank is owned by the client organization and not accessibly by any other accounts on the system.

’As the HR manager, I want to try out the screening quiz so that I can make sure it works as I expected and that I’m ready to both give it to candidates and share the results with the functional manager.’	Make sure there is a clear indication that the user can (and should) test the quiz Make sure there’s a ‘view as test taker’ mode available to the admin.

Here are a few other epics that might follow this one.

Example Epic II

‘As the HR manager, I want to give the screening quiz to a job candidate so I can assess their skill sets against the needs of the position.’

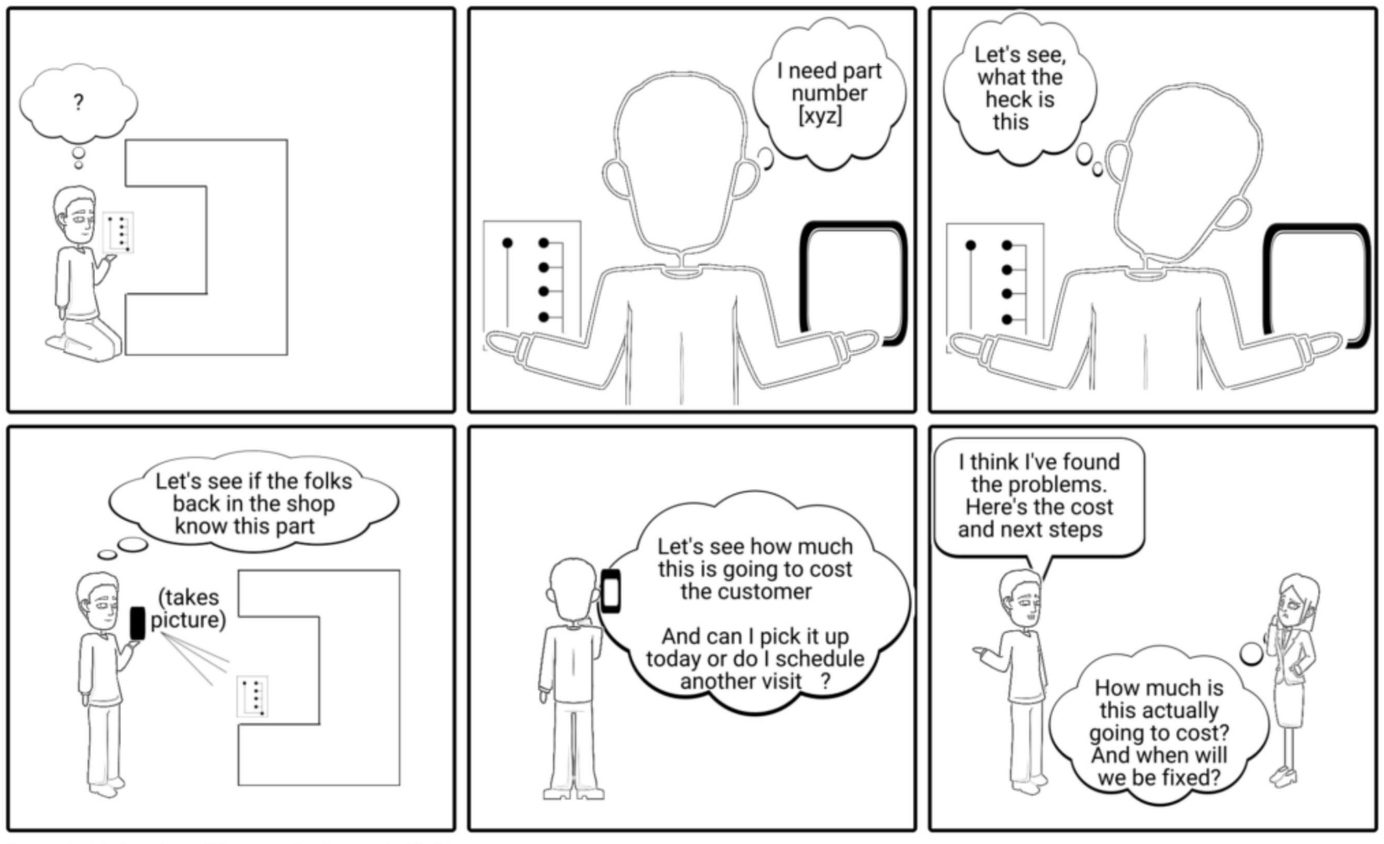
Example Epic III

‘As the HR manager, I want to share and explain the results of our screening with the functional manager so they can decide who they want to interview.’

Example B: Example User Stories from HVAC in a Hurry (IT Project)

Example Epic I

‘As Ted the HVAC technician, I want to know the pricing and availability of a part that needs replacing so I can decide my next steps.’



USER STORY	ANALYTICS
I know the part number and I want to find it on the system so I can find out its price and availability.	How often is this search used per transaction relative to the alternatives? How often does this search lead to a part order? – Searches of this type relative to

	others – Sequence of this search relative to other search types – Conversion to order from this type of search (%)
I don’t know the part number and I want to try to identify it online so I can find out its price and availability.	(see above)
I don’t know the part number and I can’t determine it and I want help so I can find out its price and availability.	(see above)
I want to see the pricing and availability of the part so I decide on next steps and get agreement from the customer	How well do tech’s that do this perform relative to others? – Conversion rate to order – Customer satisfaction per job of tech’s in a cohort that use the tool vs. baseline (mean customer satisfaction per job) – Billable hours for tech’s in this cohort vs. baseline (billable hours per week)

Citations and Image Credits

Pair Programmers: By Lisamarie Babik (Ted & Ian Uploaded by Edward) [CC-BY-2.0 (http://creativecommons.org/licenses/by/2.0)], via Wikimedia Commons
Chemist: By Linda Bartlett (Photographer) [Public domain or Public domain], via Wikimedia Commons
Present: By Kgbo (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
Chocolates: By Dwight Burdette (Own work) [CC-BY-3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons
Meter: By Iainf 05:15, 12 August 2006 (UTC) (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html), CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/) or CC-BY-2.5 (http://creativecommons.org/licenses/by/2.5)], via Wikimedia Commons

Filed Under: Product Development

ALSO ON ALEX COWAN

The business plan is dead. Long live the …
7 years ago • 5 comments
It's like watching the fall of the Berllin Wall in slow motion: as popularity of …

The Customer Discovery Handbook
7 years ago • 3 comments
This is a practioner's guide to conducting design research on personas, …

The 25 Minute Style Guide
7 years ago • 2 comments
Relevance and consistency are the top two contributors to a strong design & …

Less Hostess, More Sushi: 6 Future …
7 years ago • 1 comment
I'm working with a local college on some extension courses and the #1 thing …

Take a Close … that Duct Tap
7 years ago • 1 cc
Duct tape is enj… deserved resurg… of it from rising i…

Evan P. Epstein • 7 years ago

Great material Alex, this was helpful thanks!

As leader in a Fortune 100 Portfolio and Program Management Office, and a CSM/Agile executioner myself, I would say that among my top challenges I have encountered in the past is managing through uncertainty and working with stakeholders to create higher quality inputs/stories. The links to Lean Startup and “design thinking” are useful and I can see their application in helping teams through the curve.

6 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ Evan P. Epstein • 6 years ago

Hi Evan,

Thanks! I'm glad you liked the material.

^ | v • Reply • Share ›

Nathanael Coyne • 6 years ago

Thanks Alex, appreciate the insight and examples as I'm trying to figure out how I want to transition to Scrum and user stories, and dialing back on the wireframes and UI specifications.

The hierarchy of epics and stories is something I'm struggling to get my head around; it seems very ... organic? Some epic-level stories have as much detail as I'd expect to produce while other epics could be broken down into a dozen sub-stories and even some of them could be broken down futher to ensure they're testable.

Also, with your example epics I'm not sure if these are expected to have user stories under them but I'd have thought there would be more test cases that test the [derive a benefit] clause of the stories; I felt that the lack of test cases that explored that aspect of the user stories was a missed opportunity to increase the likelihood of delivering a successful product.

4 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ Nathanael Coyne • 6 years ago

Hi Nathanael-

First off, sorry for the slow response.

Second- thank you so much for your thoughtful comments & questions. I invest a lot of time maintaining this material and I'm thankful for every single visitor. That said, notes and questions likes this are so crucial to advancing the material.

Now, to your actual questions:

- Hierarchy and scope of stories: The above stories are examples I created for flexibility and congruence with the rest of the material on the site. It's possible (scratch that- likely) that they'd evolve during the discussion and implementation process.

I think the best way to scope epics+their children is to draft early and often, getting them in front of your

implementation team (or proceeding with the implementation yourself) as soon as possible. Not to suggest that this is what you're doing, but one of the biggest failure modes of agile user stories is rendering them as another format for a written 'spec'. The key thing with stories is that they're not just a new syntax- they're a new way of doing things (vs. waterfall, etc.). Their role is to serve as an input & anchor for discussion. It's that discussion that's the really important part.

- Why aren't there more test cases to test the 'derive a benefit clause'? Excellent question! In my classes and workshops, we do this through an integrated curriculum and the Venture Design template has provisions for that. I'm going to think about how to layer this on to the material in this specific tutorial without overwhelming. That's for the thoughtful, provocative question, and I agree it is a missed opportunity. That said, there's some material for this in the Customer Discovery Handbook: http://bit.ly/cdhandbook.

- Note on wireframes: I know you said 'dialing back' and not 'abandoning', but in practice I use a lot of wireframes to supplement my user stories- I just keep them rough and ready for change.

Thanks again for the questions!

^ | v • Reply • Share ›

Greg Cohen • 7 years ago

Great stuff Alex. This really places all the key pieces in context, especially how design thinking, lean startup and agile all fit together. And I appreciate you giving focus to the problem space because I see so many failures from solution first thinking.

4 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ Greg Cohen • 6 years ago

Thanks Greg! I'm so glad you found it useful.

^ | v • Reply • Share ›

Jim Stover • 7 years ago

Outstanding post Alex. One thing I sometimes like to add to the end of the 'INVEST' criteria is an 'D for Demonstrable. At least for functional user stories, I find it helpful for the team to keep in mind that at the end of each sprint one of the goals is to demo the completed functionality to the customers / stakeholders / etc...
Adding onto the challenges or failure modes of agile, in my experience it aligns very closely with the results from VersionOne's State of Agile Survey: Company culture at odds with agile principles and values; external pressures to follow 'the norm' and what is known and understand in an organization (traditionally Waterfall - I have grown a gag reflex to saying Waterfall haha; and generally just broader organizational issues impacting an agile transformation.

3 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ Jim Stover • 6 years ago

Thanks Jim! I'm so glad you liked it.

Demonstrable- I love it! Can we draw it on the whiteboard? Do a quick mockup? Storyboard or generally talk through the user experience? Demonstrable's a great criteria.

^ | v • Reply • Share ›

David Siegel • 6 years ago

I am an agile outsider. I have always had a crush on agile as a concept but never worked in a context that allowed me to explore it or actually learn it. Before I read this article, I had a vague notion of user stories. I thought of them a way to get in the skin of the worker to create functionality that addresses real needs. This article helped me refine that notion and was a perfect primer for someone who had only passing understanding of agile concepts.

I know you advocate this tool in the broader context of a total agile environment but I can use these concepts to make my requirement specs better. (Don't gag.)

I am glad I found this. Thanks for your contribution.

2 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ David Siegel • 6 years ago

Hey David-

I'm so glad you found it helpful and that's great!

I think all the core pieces of agile are helpful, together or individually. One of the worst things that's happened to agile is that (in some quarters) there's this perception that it's a monolithic, all or nothing proposition. Good on you for trying it out in small bits!!

^ | v • Reply • Share ›

KF • 3 years ago

who is responsible for creating the requirements? Does it mean a good story can replace requirement?

1 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ KF • 2 years ago

Hi KF- In agile, there's usually a designated role called 'product owner' and they are the lead on user stories. In practice, what that should probably means is that they draft a few of these and then facilitate sessions where the teams creates a set of working stories. With this process and regular communication within the team, yes, this can replace requirements.

Shameless but I think relevant: This online course I do Agile Meets Design Thinking gets into the specifics of doing all that.

^ | v • Reply • Share ›

Saikiran Yerram • 3 years ago

Great post Alex. The article really hones in on story writing. One thing that you didn't cover is use cases & UX.

User stories aren't enough for dev team to build features. Where do designs (UI/UX) fit with user stores. Second, where do describe user interaction with the system (aka use cases or how part e.g. form submission, checkout process) to achieve the users story.

Finally, how do you manage in sprints. Does the sequence below work?

For e.g. is the process
sprint 0
- create user stories
- get ux/ui team design prototypes
- concept test
- iterate
- user story, use case - points score
---
sprint 1
- pick user story + use case from backlog
...

...

1 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ Saikiran Yerram • 2 years ago

Hi Saikiran- These are great questions. I'm going to refactor them a little to hopefully get at a better answer. Please, please, reply, etc. if this leaves you with more questions.

How do user stories relate to the overall UX program?
I think they should be the anchor. From the user stories, the team should iteratively, dynamically, and, above all, responsively (based on what they think makes sense):
1. Pull comp's for ideas on UI patterns and/or prototype concepts
2. Draft usability test plan(s) and test the above with clickable prototypes
3. rinse and repeat as needed
4. move to code when they think it's time

Should a team have a standard process of going from user story to prototype to working software?
No, I would say, and with a lot of emphasis. In practice, what I see happening in those situations is the team drifting back into a more waterfall-ish, serialized, silo'ed way of doing things (opposite of Agile Manifesto). User stories are a **provocation** to push everyone to continually think about what they're building from a user perspective and whether it makes sense or not. The is a much harder way to work- we all find it easier to measure our efficiency locally. But it's a better way to avoid waste and maximize user outcomes.

1 ^ | v • Reply • Share ›

Inna XITE ➚ Saikiran Yerram • 2 years ago

Hi, I second this question but also: how do you manage Epics themselves? Is there separate flow for them? Like draft-ready-in work-in review-done?

Also, in reality, teams often have technical debt. What is the way to deal with it? Would it be a user story representing developer or QA that is initiating refactoring/improvement? Let's say, something internal that does not change app behavior but, for instance, is more reliable/faster/etc? Or would it be "Task" issue type?

1 ^ | v • Reply • Share ›

Alex Cowan Mod ➚ Inna XITE • 2 years ago

Hi Inna- What practical and relevant questions.

On the epics, I would think about them primarily as a way of organizing your drafting of the user stories. But that doesn't specifically answer your question. In practice, I think the Story Map (see above) is probably overall the best way to maintain a flow for those epics. They do that jobs of 1) presenting everyone with the big picture when they want to look at it and 2) relating that specifically to the individual stories you have in play (or out of play). In terms of the epics themselves, you could annotate them on the storyboard panels or just archive them wherever if you find that job is being done better by the storyboard panels.

Technical Debt: If the tech debt is about refactoring the code to make it more manageable, in practice tasks are probably better for that. If you're writing stories like 'As a developer, I want to refactor XYZ so that it's easier to maintain.', then obviously that's not super useful for anyone vs. a simple task (and it kind of distracts/confuses the important role that user stories should be playing).

If it's about performance, I would set a benchmark and work to that.

The one big exception in this area is that if you're going to build a service/API as part of your reduction of tech debt, then I would consider drafting stories from a developer using that service/API.

• Reply • Share ›

Inna Shevchenko ➚ Alex Cowan • 2 years ago

Makes sense, thank you. This is exactly how I'm shaping tech debt reduction, so good to confirm with the best practices from your experience.

2 • Reply • Share ›

Alex Cowan Mod ➚ Inna Shevchenko • 2 years ago

You bet! Thanks for posting the question. I love seeing these.

• Reply • Share ›

christian kaefer • 5 years ago

Just came across this looking for user story material to share with my team and this is great. I don't see any reference to acceptance criteria as part of the story, where does that fit in?

1 • Reply • Share ›

Alex Cowan Mod ➚ christian kaefer • 5 years ago

Hi Christian- I'm so glad you liked it and thanks for writing! On your question--

Short Version: I would use the 'Test Cases' you see described here to help think about more details if that seems like a good idea.

Longer Version: While there's nothing inherently wrong with them, I'd approach acceptance criteria with caution. Very often (in my experience, at least) teams that spend a lot of time on acceptance criteria tend to be looking at their efficiency locally and not at how to deliver something valuable to the customer (and hence their firm). Here's a more nuanced view of how the discovery process might be executed by a team: http://www.alexandercowan.c....

Not to say how much this may or may not apply to your team, but I'd be sure to balance acceptance criteria with the idea that 'collaboration > negotiation'.

• Reply • Share ›

LET'S STAY IN TOUCH

JOIN ME ON:

Tweets by @cowanSF