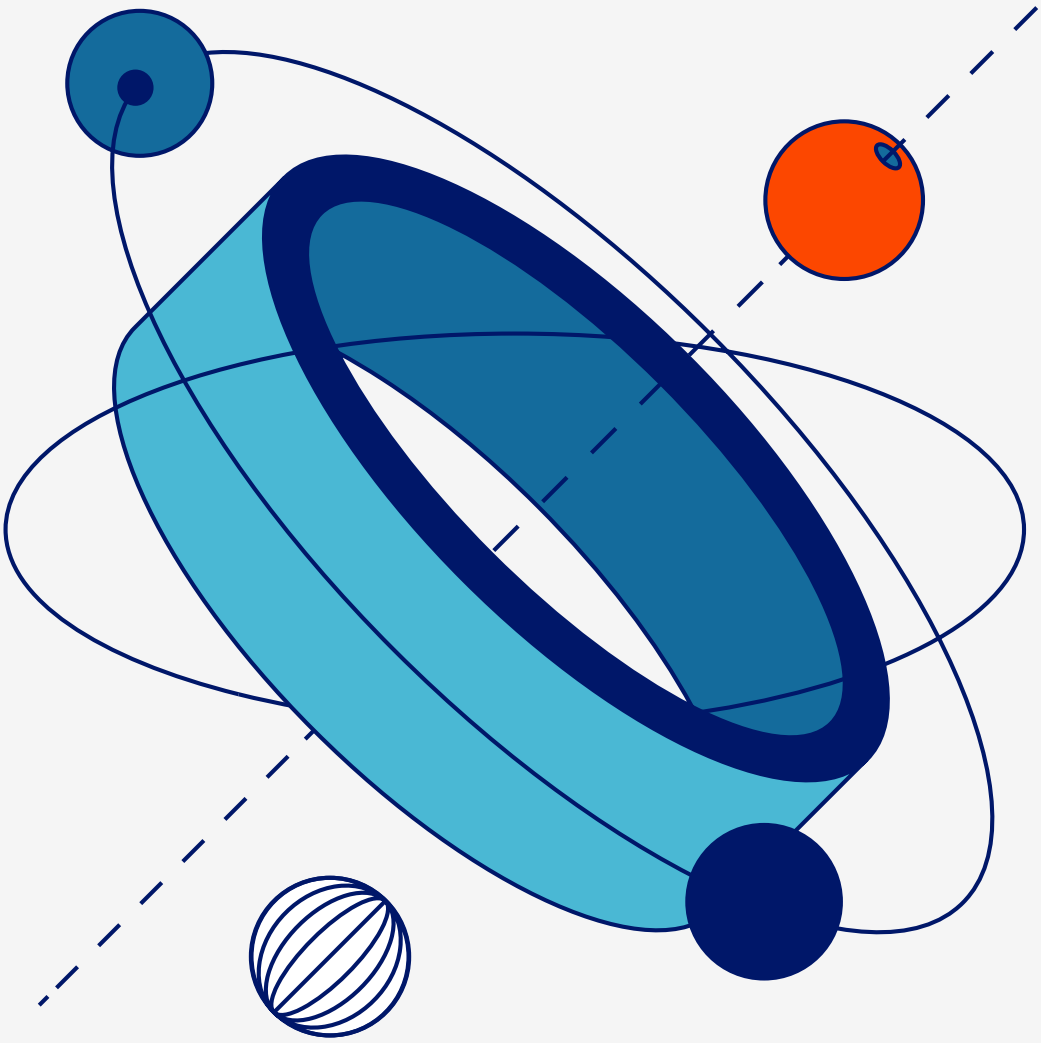
UXPin

Fixing the Enterprise UX Process

A Field Guide for Design Teams



by

Joe Natoli

UXPin

Fixing the Enterprise UX Process

A Field Guide for Design Teams

Index

Author

Joe Natoli has provided UX consulting and enterprise team training for Fortune 100, 500 and Government organizations for nearly three decades, in addition to writing, coaching, and speaking. He has launched two successful online UX training courses with Udemy, with more than 43,000 students enrolled to date. He has also written a book on UX Strategy and Digital Product Development, Think First. Joe's articles, advice and videos can be seen at his website, givegoodux.com.

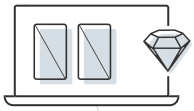
Design Systems in UXPin

UXPin

One platform for consistent design and development.

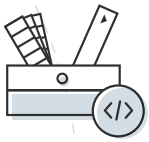
PayPal Sapient adidas HBO Microsoft

DESIGN SYSTEMS



Design Language

Sync Sketch with UXPin for a consistent design language: fonts, colors, icons, assets, and more.



UI Patterns

Scale designs consistently with Symbols and interactive components.

Automated Documentation

Documentation syncs everywhere and travels with library elements.

Modular design and development

Scale quickly with design system libraries.

One source of truth for everyone

Close your knowledge gaps. Formalize your design and code conventions.

Painless documentation and developer handoff

Eliminate busywork. Generate style guides, specs, and documentation.

Tracy Dendy
HBO

"My productivity and developer productivity have both increased. They love that they can collaborate and move quickly to a powerful experience.

To book a demo, call +1 (855) 223-9114 or email us at sales@uxpin.com

Foreword

The term “User Experience Design”, coined by Don Norman at Apple, has had a very broad meaning since its inception. As Norman explains:

> *I invented the term because I thought human interface and usability were too narrow. I wanted to cover all aspects of the person's experience with the system including industrial design graphics, the interface, the physical interaction and the manual. Since then, the term has spread widely, so much so that it is starting to lose its meaning.*

Unfortunately this historic, and by all means truthful, definition is easily forgotten.

Somehow it’s easier to close design in a silo, or limit UX to a drawing exercise. It’s easier to forget what UX means because the convoluted, minimized role of UX doesn’t challenge the organizational status quo.

I’ve experienced this misunderstood role of UX in my earlier career as an enterprise designer and UX director. The enormous frustration of ineffective product development inspired me to build UXPin – a

design platform that lets teams build products together. While our platform can increase design productivity (up to 60% as one of our customers reports), it can't solve all the problems.

No tool or platform can. Some of the painful issues are rooted in the way organizations think and act.

Our enterprise customers tell us that the fire of design genius is harder to ignite in corporations than in startups. UX loses its meaning in the battle of conflicting interests. Togetherness is often replaced by politics. Problem-solving is easily confused with the search for excuses.

Re-establishing UX as the collaborative spirit of product development requires two things:

1. Getting back to the roots of the field as described by Don Norman.
2. Thinking like a small company inside of a large organization.

Fixing the Enterprise UX Process by Joe Natoli does a great job of providing a common-sense framework to defeat both challenges.

Without further delay, let's explore the lessons learned over almost 30 years of design experience.

Marcin Treder,
CEO and co-founder of UXPin

Revisiting Enterprise Problem-Solving

Enterprise design and development teams know all too well the challenges of working in an environment where **the urgent trumps the important**. Speed and task completion are easily mistaken for value and success.

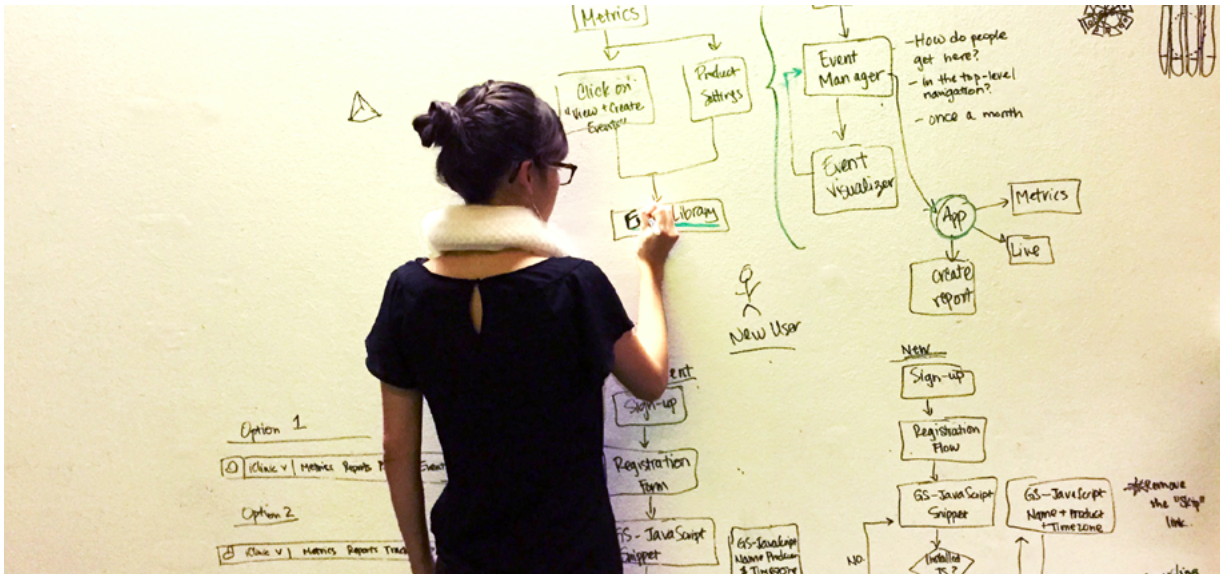


Photo credit: KISSMetrics

If you're a **Product Owner** or **Product Designer**, you're no doubt under tremendous pressure to deliver meaningful improvement – *without* spending the time or money you know it really takes to do that.

If you're a **Developer**, you're likely staring down a seemingly impossible coding workload – and now you're saddled with the responsibility of designing a better user experience, too.

From UI Design to Information Architecture and Interaction Design, you and your team are essentially expected to work miracles. Never mind that (despite your asking) you may have no training in these areas, or that some of these skills are in direct opposition to what you do best.

For far too many enterprise product teams, this is familiar territory. We're all living in a world where burnout factor is high, tempers are short, and the question of how to truly improve product quality and user experience often seems rhetorical.

But it doesn't have to be that way. I *do* think there's an answer, and it's successfully put into practice everyday – at **startups**.

For Startups, Problem Solving Equals Survival

Startups exist in a perpetual do-or-die scenario. Limited time, budget, personnel, and the need for speed offer no cushion. Every activity the product team undertakes must deliver **value**, because there's no time for anything else, no room for error.

Startup life is a constant struggle for the right to exist. **If you don't solve the *right* problem first, you won't survive long enough to solve the others**.

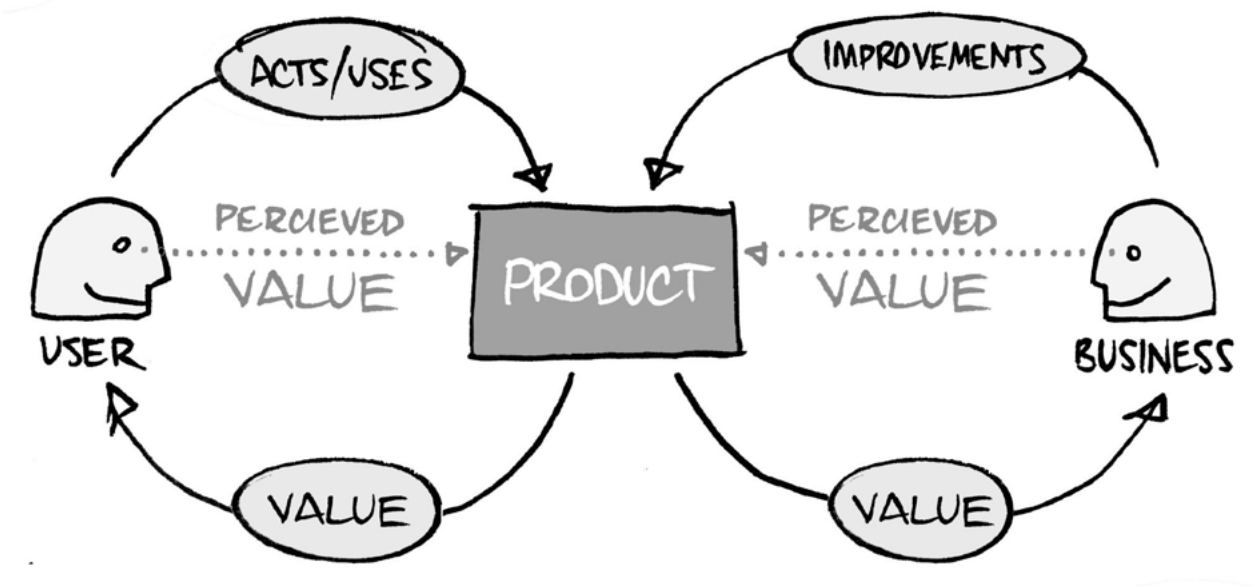
The goal here is not speed. It's not Minimum Viable Product, or dogmatic adherence to Lean or Agile, or any other approach to design and development efficiency. The goal is ***a relentless focus on problem solving***.

Every choice you make, whether it be a line of code or a button style all ultimately affects user experience. You are solving problems, from "how should this work," to "how can we improve data load speed," to "how can we present this in a way people understand?"

For those of us in the world of enterprise products, designing like a startup means focusing purely on **problems, solutions and opportunities**.

Leveraging the UX Value Loop

At the end of the day, every effort undertaken by an enterprise product team should directly correlate to the **UX value loop**:



Whether end-user or buyer, people need to **perceive value** before they'll act – even in situations of compelled use where they have no choice but to use the system. Once they *do* act, that **value must be confirmed** in order for them to try, buy, or continue using it. If any of those things happen, **value comes back to the organization**, usually in the form of money made or money saved.

The value back to the enterprise is the last critical inch. Depending on the nature of the product, the features we design must:

1. **Convince the user/customer to purchase** (e.g. after a trial)
2. **Keep the user/customer from abandoning it** (to strengthen customer loyalty)

3. **Ensure that the user/customer continues to use it** properly (to help users realize gains in productivity and cost savings).

In situations where use is compelled and UX is poor, people create workarounds. This hurts the organization in multiple ways, from duplication of effort to missed deadlines.

One way or another, the bottom line suffers.

Change Your Thinking, Change the Enterprise

UX isn't just about users, and it's not the sole responsibility of people with the words "design" or "UX" in their title.

In almost 30 years of work with enterprise teams, I've had the privilege of working alongside some truly extraordinary folks who took this idea to heart.

From front-end developers to database architects and UI designers, I've seen teams rise far above what anyone thought was possible within the constraints of time, budget, and personnel. They were able to make **quantum leaps in product improvement** simply by changing the way they *thought* about their work. By redefining what it means to design. By adopting and applying practices and principles that are an inherent part of startup life.

I've done it and they did it, which means *you* can do it, too. Let's dive into what it means to **design like a startup**, along with the mechanics of adapting these principles over the course of an enterprise project.

Here we go.

Joe Natoli,
Enterprise UX Consultant
Give Good UX

Stop Jumping to Solutions

You've probably experienced this scenario a few times.

A missive comes down requesting a new feature, function, section, etc. No matter who originates the ask (be it a product manager, marketing, a department head, etc), it's almost always delivered with a suggested (and often *required*) **solution**.

"It should look like ..."
"It should work like ..."
"It should have these features: ..."

The one thing that request doesn't do is thoroughly explain the **problem**.

That's called **solution jumping**, and for far too many organizations hurtling forward in the name of all things Lean and Agile, it has become an Olympic sport. There are two big problems with solution jumping:

1. **It doesn't consider *intent*, which sends people off in different directions**. When we humans start with the end state – the *solution* – the brain works to fill in the reasons why it's the right choice. This is neurological habit and reflex. And because everyone involved comes up with different rationales, they're all starting from different positions of intent. In other words, we suffer widespread disconnect from day one.

2. **It tricks people into believing they're now ahead of the game**. You'll hear something like "Folks, you don't need to waste time doing requirements or user research; we've got that already. This is what we need." Here's the problem: your speed or distance doesn't matter if you're *driving in the wrong direction*.

You know solution jumping is happening if, after suggesting that time should be spent understanding the problems at hand, you hear any of the following statements:

"Why do we need to understand the problem? We already have the solution!"
"Why are we moving backwards?"
"This feels like a waste of time."

Let's say your website users are complaining about slow load times and page load failures, to the degree where upper management is feeling the heat. So the Director of Product Marketing fires off an email requesting that IT implement incremental content loading, otherwise known as "infinite scroll." He's seen this work for Facebook,

plus, the company's largest competitor is doing it. That's a **solution jump**: *I know what's needed, here it is, go do it*.

Here's the problem with the solution:

1. The Director assumes the competitor site uses incremental loading to **solve the same problem** as his company's site.
2. The right question isn't being asked, which is do we really need to load ***all* of this content** in the first place? Do users really want it, or even *care* about it?

Turns out no one ever spent the time to think about what content a user would want or need, so the site is built to load *all of it*. At once. So how it loads isn't the problem, but rather a symptom of the *real problem*: that (a) we're loading too much data and (b) it may all be useless to potential customers.

Filtering results based on browsing history, or instituting user-controlled settings are both much more appropriate solutions. But the team never gets there if they jump to a solution before they really **understand the problem**.

For the enterprise, the question easily becomes "Can we build this product?" For a startup, the question is "*Should* we build this product?" Lean startups operate on the premise that their work is an experiment meant to answer that question. Startups also embrace the idea of the "five whys" a method for uncovering root problems (which I wholeheartedly recommend).

Here's an example of how – and *why* – it works:

1. **A new release disabled a feature for customers. Why?** Because the third-party module used wasn't compatible with all browsers.
2. **Why wasn't it compatible with all browsers?** Because it was implemented in the wrong way.
3. **Why was it implemented in the wrong way?** The developer who specified it didn't know how to customize the code properly.
4. **Why didn't he know?** Because he was never trained.
5. **Why wasn't he trained?** Because his manager isn't allowed the time or budget necessary for training.

So what started as a purely functional issue turns out to be a larger organizational issue. The technical problem can be fixed, of course, but it's a safe bet that future implementations will suffer until that larger organizational issue is addressed (and solved).

UX is Everybody's Business

When there's an issue with a digital product, the problem is almost *never* one thing. It's rarely *just* usability, rarely *just* the underlying technology, rarely *just* the UI design, etc.

It's almost always a combination of those things, all of which is usually the result of a larger team or organizational issue.

For example, during a kickoff or brainstorming meeting, it's pretty easy for everyone at the table to imagine a solution. And each solution is filtered through that person's experience:

- The **Business Analyst** is looking to statistical research for answers.
- The **Product Owner** is pushing established user stories and wants to create new ones to solve the problem.
- The **UX Architect** wants to go back to existing user research, plus start planning to do more.
- The **Developers** are thinking and talking about possible tools and processes to solve the challenge.

- The **UI Designer** is mentally designing screens in his head as he listens to everyone talk.

These are all solid, time-tested approaches. At the same time, they are all inherently flawed because each addresses only a **small slice** of the real problem. Each group has a very narrow perspective on the most appropriate solution. Everyone is doing what they do best.

What's missing is a shared focus on User Experience; a common lens with which to filter their activities. In the enterprise environment, these tasks, roles, and responsibilities are usually siloed by multiple departments and even locations.

Startups think differently about this issue of decision proximity, from how people work together to where people sit in the office. The open office concept popular with startups is a physical manifestation of an overall mindset: close, continuous collaboration. Every team member has a hand in making product decisions.

Enterprise organizations often resist UX improvement largely because of the way it's positioned to them. Every Fortune 500/100 organization I've ever worked with has relayed some version of how all the previous "UX folks" were hellbent on radically changing their methods and internal processes. You and I both know this never works.

To find common ground, everyone in the kickoff (and all subsequent meetings) needs to first ask themselves three critical questions:

1. **Is this worth doing?** Every minute spent researching, designing and developing a feature is an investment. Are we certain that every requirement on our list delivers a return on that investment?

2. **Does everyone agree on and understand what we're creating?** Does every member of the team share the same understanding of what's being built and why it matters?

3. **What specific value does it deliver?** Does this improvement, feature, or function facilitate a desired outcome for both customers and the organization? Do we *know* it will deliver that value, or are we guessing?

There are other questions, of course, but the simple act of stopping to think about these things will often change what happens next, and in a very positive way. This is what I emphasize while working with organizations. It's the reason I wrote my latest book, *Think First*.

If you're a startup, you must rely on **innovative, analytical thinking** first, because everything else is in short supply. And guess what? Even the largest global organizations are in the very same boat – it's just a *bigger* boat.

Until everyone considers how their decisions impact strategic issues, the proportion of effort to result remains grossly unbalanced. And while we'll certainly be moving fast, plowing through a monstrous amount of heavy lifting, we'll be doing it on a *treadmill*.

Get the Right People in the Room

Stop me if you've heard this one before: You're in the home stretch of development and heading toward launch when people in the organization *you've never heard from* suddenly swoop in and mandate changes to **everything you've done**.

Most of us know this scenario as the "swoop and poop."

It happened because, from day one – from initial planning to requirements sessions to prototyping – **they weren't at the table**. Perhaps they weren't invited, or maybe they aren't willing to make time to be "wasted" on project work.

Why do these hidden stakeholders have so much influence?

Because they carry the most **risk**. If the project fails, *their necks* are on the chopping block. They're usually removed from the development process, which means they're placing a very large bet on something they don't know much about.

If you fail to recognize this – and proceed without their input – the above scenario will play out repeatedly. You'll go down one, two, six, or eight sprints, and the scope keeps changing.

Involve stakeholders by calming their fears

It's not unheard of for designers, developers, and organizations to get stuck in a cycle of endless iterations.

"I'll know it when I see it" or "That's not what I envisioned" are all running jokes, at this point.

This is often a ping-pong game of guesswork: the team is guessing at what the stakeholder wants, while the stakeholder becomes increasingly frustrated that what they're seeing is *not it*.

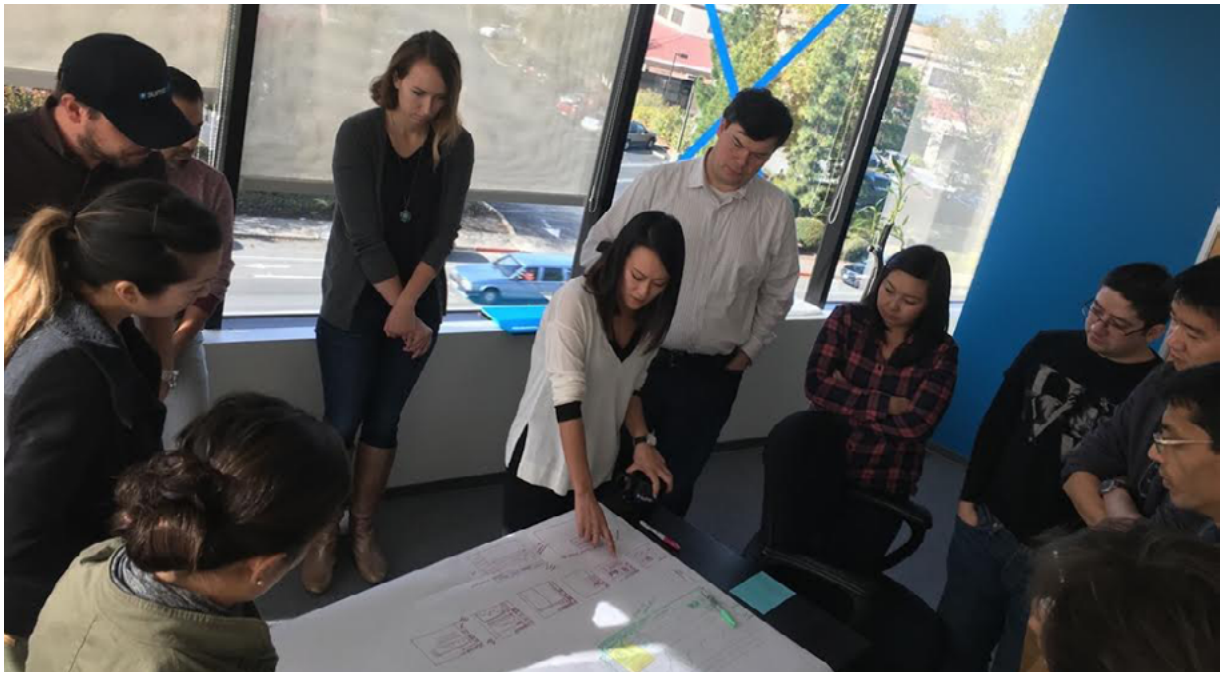


Photo credit: Sumo Logic

What happens next is that team members start excluding the stakeholder from the process. Sprint plans include fewer iterations, and designs or builds are simply thrown over the fence for review, instead of taking the time to explain decisions.

Here are some tips that have helped my enterprise teams avoid these issues:

1. **Invite them into the process early**. You'd be shocked at how many times stakeholders aren't involved simply because no one asked them. In at least half of the organizations I've worked with, they're excluded (often on purpose). But the more you exclude them, the more concerned they become–and the more influence they try to exert as a result. So start by asking them to be involved. If you feel alienated from them, I guarantee you they feel the same way.

2. **Explain that their vantage point is critical**. Tell them you can't do it without them. Invite them to share their pressure and pain, explaining that solving those things are part of the mission. Ask them what success means *to them*. In doing so, you are giving them a sense of control and ownership. They feel like they're being *heard,* which is fundamentally a human need.

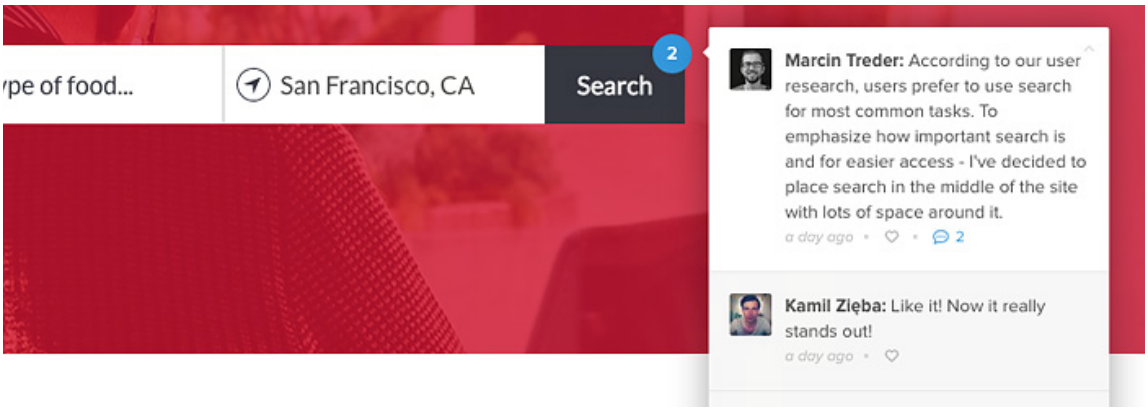
3. **Educate them about the design process**. Explain how their information integrates into your work. Fear comes from the unknown; fill in the blanks and the fear disappears. In many cases, it doesn't take much to change a stakeholder's position from critic to advocate. When that happens, they're less likely to re-

ject the results they see (and more likely to endorse your work to other decision makers).

Designer Pro Tip

Always support your design decisions by referencing user research and business logic. Instead of saying that a stripped-down layout creates a better impression, explain that it emphasizes the content and calls-to-action, which helps to persuade and convert users better.

For example, when working on a Yelp redesign exercise in UXPin, I commented with the user research supporting my decision to place the Search bar front-and-center (instead of on top). This way, marketing stakeholders could understand the choice as a business decision rather than a matter of taste.



Marcin Treder, CEO and co-founder of UXPin

When the Answer is Still “NO”

Yes, you will encounter situations where stakeholders will outright refuse to be involved. When this happens, you have one play left.

Request that **all department heads be present for the first working meeting** (and to me, they’re *all* working meetings).

With very few exceptions, you’ll hear some variation of “*that’s unusual… we don’t normally involve everyone in these projects,*” or “*I don’t think that’s something we can afford to do*.” You should expect this, so prepare to ask two very simple questions in response:

1. Over the last 3-4 internal projects, how many of them were **significantly late and over budget**?
2. How many of them **failed to deliver the desired outcome**, in the eyes of one or more of those stakeholders?

The answer to both of those questions is *almost always* some variation of “the majority of them.” When that’s the case, it’s also usually true that they weren’t involved upfront in any of those projects, either.

After posing those questions, tell them this is your collective opportunity to change the answer. With very few exceptions, this conversation will put you all on a path to capitalize on that opportunity.

Draw Hard Lines Between New and Legacy Functionality

Once you've improved collaboration, you still need to answer the lingering question of legacy vs. new functionality for a product.

If a planned redesign of one section of a system only represents 10% of overall use and there's no hard evidence anyone cares about it (or *will* care), **don't waste time redesigning it**. Improve speed, improve accuracy, make the screens cleaner and the language easier to understand. But *never* venture into full scale redesign unless you have evidence of expected value. And *quantifiable* evidence is always better.

For example, if you've done the math and it shows a redesign can reduce time spent on a task by 60% across 8,000 employees, that's a pretty convincing reason to do it. If the money the organization saves quarterly via that efficiency is $1200 per employee times 8,000 employees, it's a no-brainer.

Whatever you roll out must be *perceived* and *proven* to be so dramatically better (it doesn't matter what you or the stakeholders think) that users will forgive if **the rest of the system is still garbage**. It

has to deliver something that makes their lives significantly easier or simplifies a hated task (or eliminates it altogether). The resulting gains in efficiency or reduced stress buy you time to create a plan for fixing the rest.

The Difficult Tradeoff

Let's assume you're building an altogether new system to replace one of three systems in use. While you can wipe *this* particular slate clean, what you build still needs to integrate with the remaining two (woefully outdated) legacy systems.

Obviously, we've committed to a host of new sections, features, and functions, but what about **consistency**? The current scope didn't consider that some of what we're building will interface with these older systems. Users will see (and have to use) both, so do we:

- **Keep interactivity and UI consistent with 10-year old conventions**, poor interactivity, low usability, and bad design?
- **Design the new stuff with better visual hierarchy** and smarter functionality, yet *ignore* consistency with the older systems?
- **Try to be consistent with the last new section we designed** and forget about the 10-year old sections?
- **Work to find some middle ground**, where we have some consistency but the whole system isn't as good as it should or *could* be?

If you're nodding your head and laughing to yourself right now, I feel your pain. Every enterprise organization I've ever worked with is *still* wrestling with this.

Existing vs. New Users

Startups face the very same struggle with platform integration across multiple delivery channels.

The decision to build (or not) also rests upon your ability to figure out **who you really need to serve** – and which group matters *most*. The larger the organization, the more difficult this becomes. There are, however, some ground rules I strongly suggest.

Both the team and the organization must decide which of two audiences they should serve: an established, **existing user base** or a new and/or **prospective user base**? You can't be all things to all people, so you need to make some tough choices about who matters more.

1. **Existing Users**

   You design and build for an existing user base when the majority of your **revenue** comes from long-term users. Even if it sucks, they know how it *works*. So whatever new elements you propose builds on existing knowledge. The only exception is if any new workflow or UI convention is so good that the changes are a long-awaited *gift* in the eyes of these folks.

If you can't get to that level of greatness where users forgive the learning curve and follow along, then you seek the middle ground and make the choice referenced above:

- **Work to find some middle ground**, where we have some consistency but the whole system isn't as good as it should or *could* be. Improve what you can, but don't create anything that's a radical departure in terms of navigation, interaction patterns and general data presentation.

2. **New Users**

You design and build for potential users when the organization is **dead in the water** unless it gains a significant number of new customers. When I've had clients with buyers abandoning ship because they're tired of terrible legacy systems, the writing was on the wall – adapt or perish. That's almost an easier situation, because you have ample evidence that no one wants the status quo.

In this case, focus on the chunk of the system (or collection of systems) that represent the majority of (a) **use** and (b) **revenue** for the organization. *Ignore everything else*. So here, we're making this choice from the previous set of questions:

- **Design the new stuff with better visual hierarchy** and smarter functionality, and *ignore* consistency with the older systems. Navigation, interaction, instruction, workflows all need to be reimagined and redesigned. Your back is against the wall; your only choice is to radically improve the system. They won't use the system unless you give them reason to do so.

I realize that none of this is as neat and tidy as I make it sound here.

Like anything you do, the road to updating and modernizing existing systems can be rough. But it *is* entirely possible – with a shared focus on **desired outcomes**, a realistic approach toward **what's truly worth doing** and a clear understanding of the **tradeoffs** between value and cost of your efforts.

Designer Pro Tip

Rather than relying purely on expensive internal sales teams, new enterprise companies start by seeing a single team try out their product. Your user growth comes from employees at the company selling the product on your behalf. Over time, the bulk of your revenue growth comes from expansion rather than new customer growth. Your product must be compelling enough for that first team to recommend to the rest of their company.

Amanda Linden, Head of Design at Asana

User Research: Fast, Early, Often

Your mileage may vary, but my experience with enterprise organizations dictates there is usually little to no budget for **true user research** – meaning user interviewing, shadowing and/or direct observation of end-users.

Getting buy-in for these activities is time-consuming and difficult, and on a tight deadline we quickly realize there's no time to educate management on user research.

So instead of just a "Sprint 0" where we carve out time and personnel for heavy upfront user research, we try a different tack altogether: **we research early, and we revisit it often** to continually narrow and qualify what we think we know and what we've learned along the way.

Here's how.

The 1-Page Summary

By the time a project kicks off, you've probably already discussed how proposed features and functionality will be used. There's existing knowledge about user roles and responsibilities, and in most cases, some past use history that allows for educated guessing. This is **what we *think* we know**, because it has yet to be qualified.

Because it's part of the project, our current hypotheses about user needs are worth documenting. But that documentation should be nothing more than a quick, one-page summary. Many organizations love big documents, but the truth is **no one reads them**. And most executives won't read past the first page, *ever*.

Even if your organization mandates gargantuan documentation, create a table-based summary for each section. In terms of user research, my favorite approach is dead simple – a table with four columns:

1. **Problem/Issue** (what problems do we believe users have, that we intend to solve?)
2. **Proposed solution** (how will we alleviate those problems?)
3. **Expected Result** (what do we expect to happen [success metric] both for the users and the organization as a result of doing this?)
4. **Related Requirements/Tasks/Activities** (what do we think is needed in order to do this?)

If any of this takes up more than a single page, **your scope is too large**. You're taking on too much with lines of inquiry that simply don't matter.

In my mind, no matter what the document is or contains, these four components are what clearly delineate everyone's marching orders. They keep all research related activities focused on **value**, allowing us to narrow the areas we investigate.

Break everything down to common-sense level, **write so anyone can understand it**. Dispense with industry jargon and big words.

Skip Traditional Personas and Get to Context Quickly

You'll find plenty of prescriptions for personas, but – unless you follow Alan Cooper's one true model – they're all essentially the same: laundry lists that suggest you can understand a person's motivation simply by checking boxes and asking questions related to behavior.

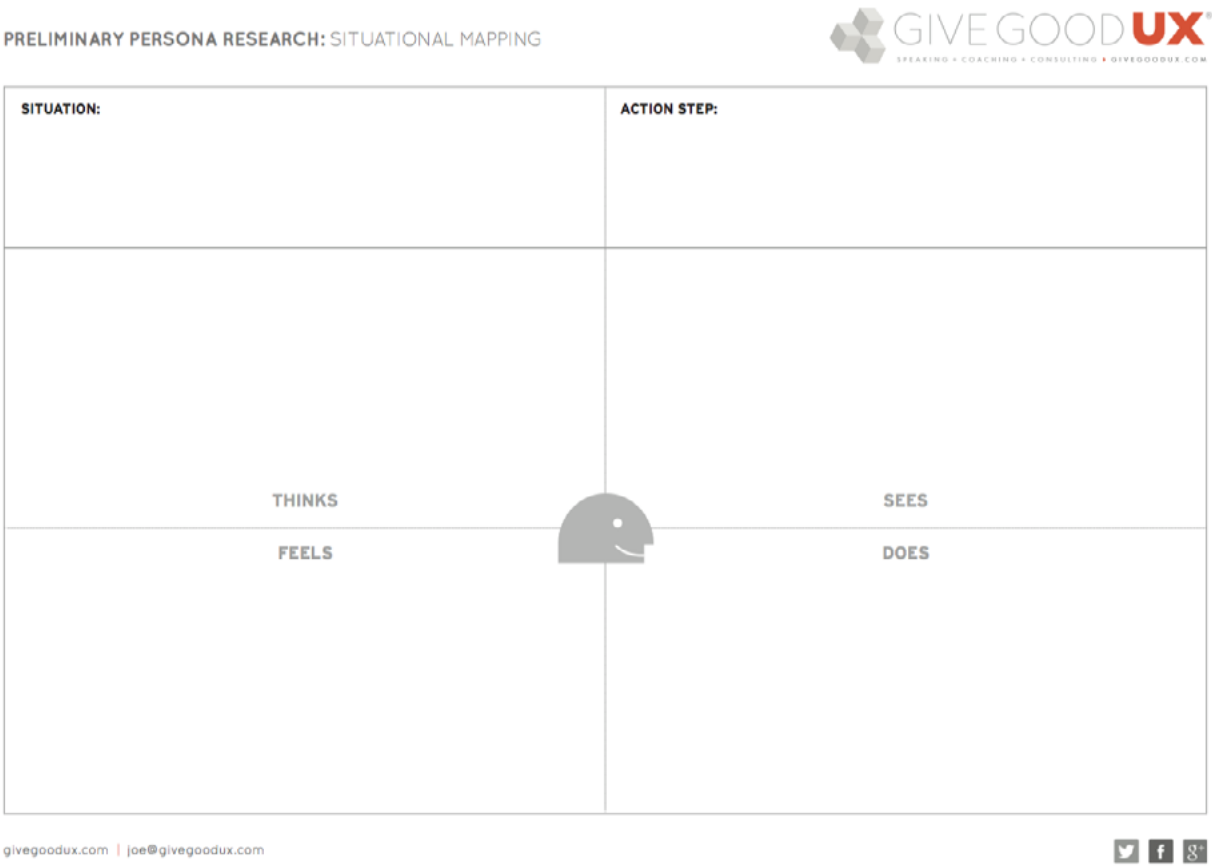
We don't have the luxury of time to start wide and rigorously work to separate fact from fiction. We need to be accurate in our first shot at describing user needs – which means it should be **contextual**. The process includes two key steps:

1. **First, develop empathy for the person**. Empathy goes far beyond demographics, likes, dislikes, job roles and responsibilities. Empathy is about understanding the *emotional drivers* that af-

fect the user's behavior, because **emotion will trump intellect** in almost every user situation. Design for the emotion and you're truly designing for a *person* instead of a collection of possible attributes. While that sounds intimidating, the truth is it can be done in 20 minutes – using what we call an Empathy Mapping Worksheet.

2. **Next, uncover behavioral attributes that motivate use, in the context of a situation**. What has the person just done or just finished doing when they encounter the product (site, app, tool)? What are they *thinking* and *feeling* at that moment, and how does that affect what they need to see or do, or how they act? What **stress** is present in that situation, and how does it affect the person's perception and action? Sounds like a lot of

work, but again, it typically takes no more than 15-20 minutes. The tool we use is called a Situational Mapping Worksheet.



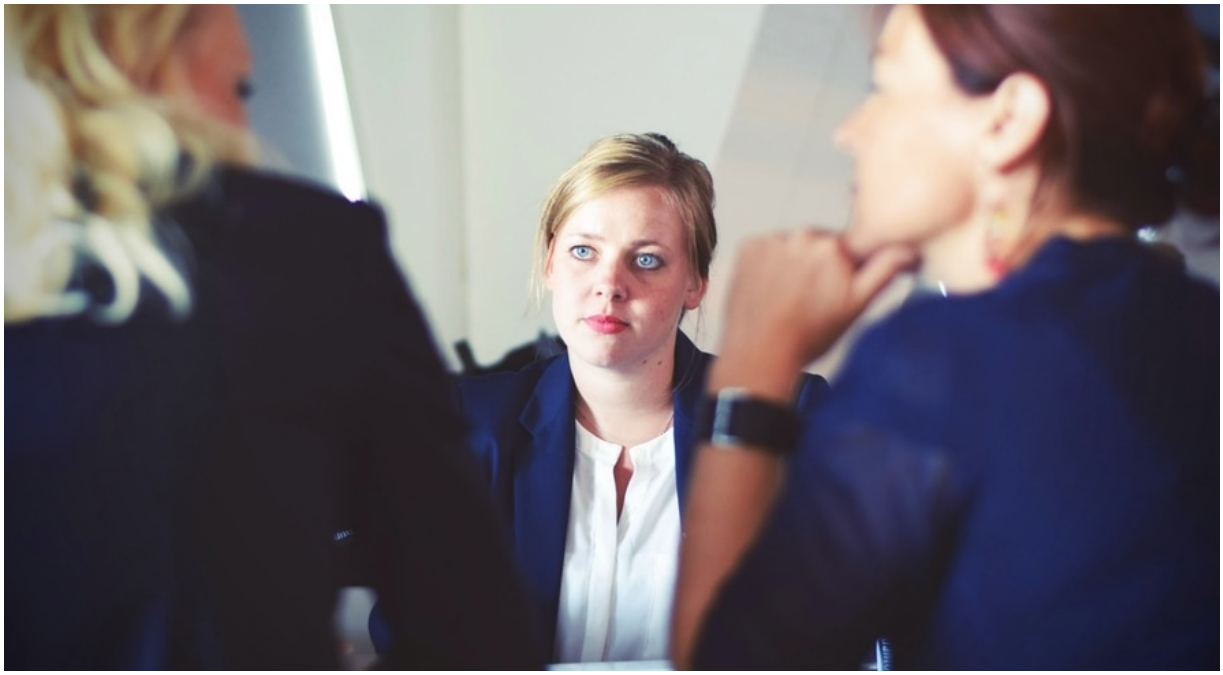
Both tools are extremely simple and virtually self-explanatory, and you can learn more about them here.

Because we're focused on **context of use** in both situations, our hypothesis are almost always more accurate than any traditional persona work. The enterprise teams I've worked with have found that combining this work with simplified user interviews (when possible) is a one-two punch that is *extremely* hard to beat.

Streamline User Interviews

When interviewing users, aim for the largest representative sample possible. The way you do that is by asking fewer, but more poignant and open-ended, questions.

Don't ask them about the system at hand, or it's features or functions. Those questions will focus the answers on the tool they're using – instead of the process they follow. Any number of factors unrelated to the specific tool or even the *task at hand* may contribute to the current issue or problem. People might use workarounds to avoid using the system altogether. And if you ask a narrow question that's tactic- or tool-specific, you'll never hear about any of those things.



Here are the 6 core questions that typically prompt people to tell the kinds of stories that uncover problems and desired outcomes:

1. What constitutes a good work day for you?

2. How do you go about doing (name a specific process, task or end goal)?
3. Can you show me how you do that (if time allows)?
4. How does this compare to other organizations you've worked for?
5. What are the biggest problems, obstacles or inefficiencies you deal with in doing this?
6. What other things do you do before, during and after this?

Are there other questions you can ask? Of course.

But **these six are worth their weight in gold**, because they invite the kind of stories that quickly get to the heart of real obstacles. You want to interview as many users as possible, as quickly as possible, so stick to these six.

Here are some quick guidelines to make the interview process quick, focused and easy to manage:

- **Interview at least 3 users, but 5-10 is ideal**. If you can't get to 3, don't do the interviews. Instead, start gathering a pool of people to test your prototype with. When you have less than 3 people, you won't have enough data to disqualify the emotions and preconceptions that skew responses. What's more, you won't see a wide enough example of the diversity inherent in enterprise users.

- **At the same time, you may need more than 10 if you have multiple, specialized user groups**. If a portal will be used by both accounting reps and development teams, these two groups have different motivations for use and different paths of use altogether.
- **Work to keep answer time for each question within 10 minutes** where possible, because in most cases anything after that is repetitive information or ancillary detail.
- **Don't expect to do all interviewing upfront**. Your first set of interviews should happen in the first 48 hours, but they don't end there. Interviews can and should coincide with post-requirements sprints, which often means two-week intervals.

If You Can't Access End Users...

When you can't get to the source itself, you need the next best thing – an informed opinion based on experience. Whether it's an IT Manager or an Account Representative, find out who's responsible for guiding end-users through the installation and configuration processes. Those people will be your user research champions, for two reasons:

1. They have **direct face time** with the people using the software, so they're hearing a whole lot about what users like or find useful (along with what they don't).
2. They have a **vested interest** in making sure the customer gets what they need.

People in these roles can help you connect the dots between the mechanics of product implementation and user needs. At the end of the day, it's their job to make sure that people get things done efficiently with the software.

Any organization can pitch a product, but if users are unhappy after implementation, or if they create workarounds to avoid using the product, these folks are the first to hear about it. Because when customers are unhappy, contracts don't get renewed. So in most cases, Account reps or IT Managers will be happy to have your help.

Offer to assist during the install, offer to make yourself available to answer any questions. Introduce yourself and relate that your job – just like theirs – is ensuring that employees are happy.

Validate User Research By Testing Lo-Fi Prototypes

Use cases and user stories should inform your requirements, but further testing/validation of those requirements should come from prototype testing.

A little later on, you'll read about low-fidelity prototyping for requirements generation. In order to serve this purpose, our prototypes should expose the key screens in core workflows.

They should describe the content (text, images, etc.) and interactive controls (links, buttons, menus, forms, etc.) appearing on each

screen. This is a two-step process: (1) **iterate** quickly and (2) **socialize** with users. And if you can't find a minimum of 3 users for testing, dogfooding with **5-10 coworkers** is equally valuable. Triangulated against the empathy and situation mapping worksheets, it's possible to make reasonably accurate judgments about what's appropriate and what isn't.

The specifics of prototyping are covered in the coming pages, but we're aiming for answers about the design's foundation. In general, we're asking if the **navigation categories** and **interaction models** we're proposing are easily understood. We want to know if the **visual hierarchy**, **information structure**, **navigation** and **workflow** we propose:

- Presents information users want and **expect**
- Presents information in a way where they can easily **find** what they need
- Uses labels users will readily **understand**
- Allows users to easily and accurately **predict** the outcomes of their actions

Invite your stakeholders to attend the usability testing sessions. You'll gain quicker buy-in for requirements decisions when they see firsthand where users struggle.

User Stories (And Requirements) Are Not Created Equal

If you asked me for the single, best piece of advice I have to offer, it would be this:

Not everything is worth doing.

Startup thinking is all about **value**–what matters most within the confines of reality.

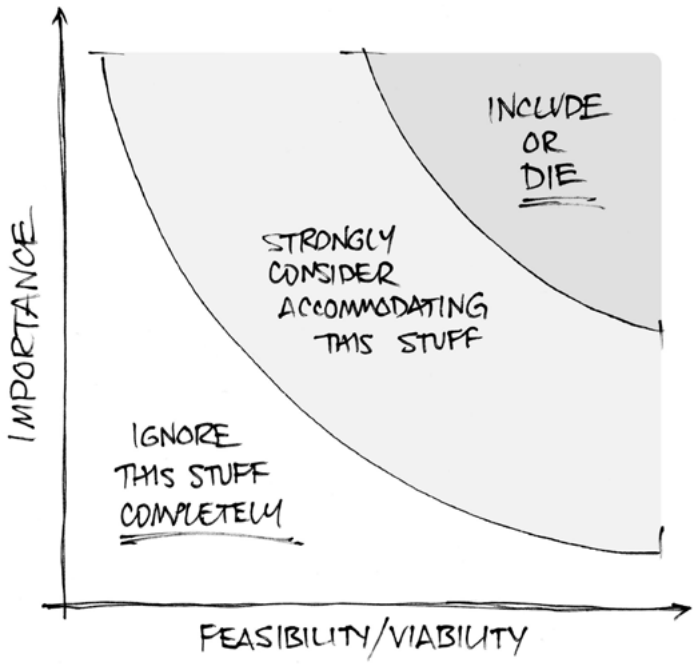
Compare that to enterprise product development, where the number one problem is wasted effort – *on things that don't matter.* We've all heard the excuses blaming time to market, customer responsiveness, Lean/Agile practices, or any number of misdirected justifications.

They're *all* nonsense, because the simple fact of the matter is this: being first *doesn't matter* if you're delivering something no one wants, needs, or is willing to use.

Once you've tested your assumptions against users, filter all the ideas through importance and feasibility/viability. Take an hour to figure

out what's actually worth everyone's time, effort, and the organization's investment. Seriously, it doesn't take any longer than that.

With the simple chart below, I've seen this exercise take all of 60 minutes in a room with 12 people and a list of 50+ requirements



On one axis you have **importance**.

- How important is the product as a whole to the business, to users, and to achieving the end goals?
- To what degree does each individual feature and function fulfill needs and deliver value?

The second axis represents **feasibility/viability**.

- What can we actually pull off in the time allocated?
- Are the budget and resources (read: people) sufficient?

- How much can we conceivably do? And if we get it done, how will we continue to maintain and improve the product?

Here's how it works up close.

You poll the room for each user story/possible requirement. You ask about the importance to users and to our stated business goals, and *can we do it*? You graph the answer and then decide what to do according to its position:

1. **Anything that lands in the lower left section is out**. As in *immediately*. Otherwise you're wasting time or money addressing things that (a) aren't important and (b) probably aren't possible. If a proposed requirement is of low importance and not feasible within your constraints, either postpone or scrap it.

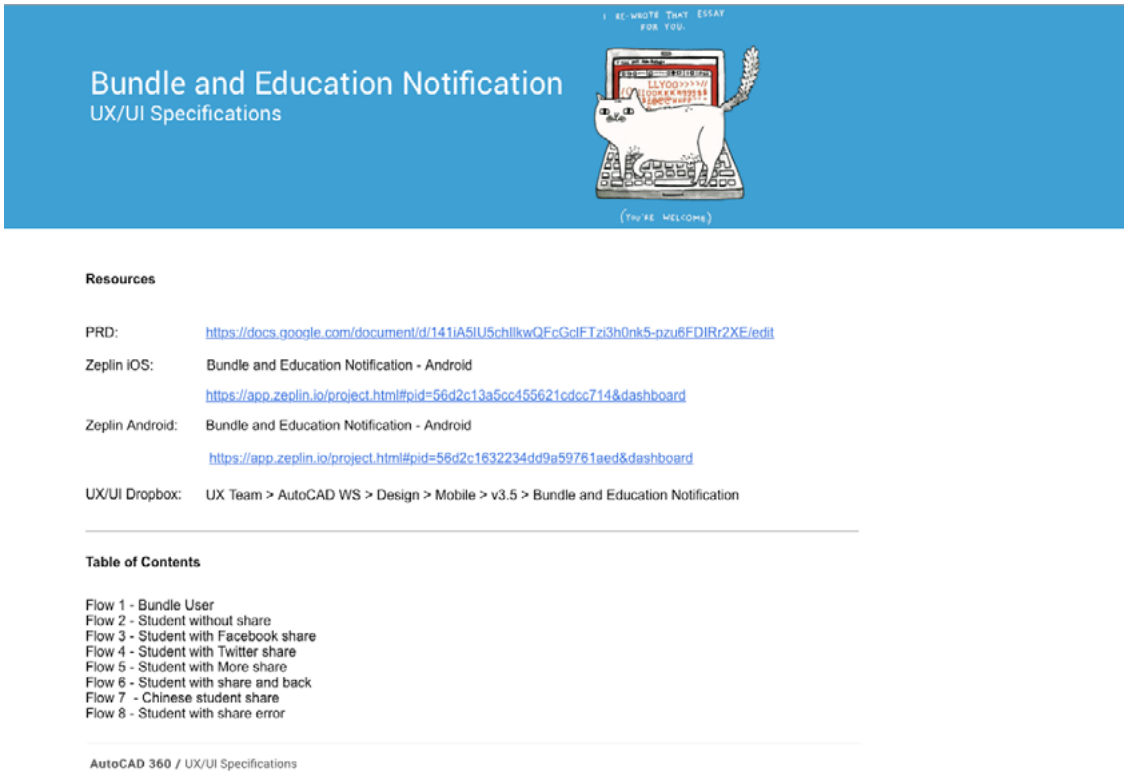
2. **Anything that falls in the middle section goes straight to the backlog**. You could accommodate these items, but you shouldn't spend the majority of your effort doing so. It's unlikely that anything landing here is critical for users and the business. You may even have doubts about feasibility. These items don't need to be perfect, just done.

3. **Anything that falls in that top right area is what you commit to doing**. That means you're damn sure it's important, and you're equally sure it's feasible. Design these items extremely well. Features and functionality in this category enable the product to serve as some kind of *answer to prayer* – and that's where you'll get the most return on your effort.

Even if this exercise takes up a full 8-hour workday, the wasted effort and potential disaster avoided makes it worth every minute.

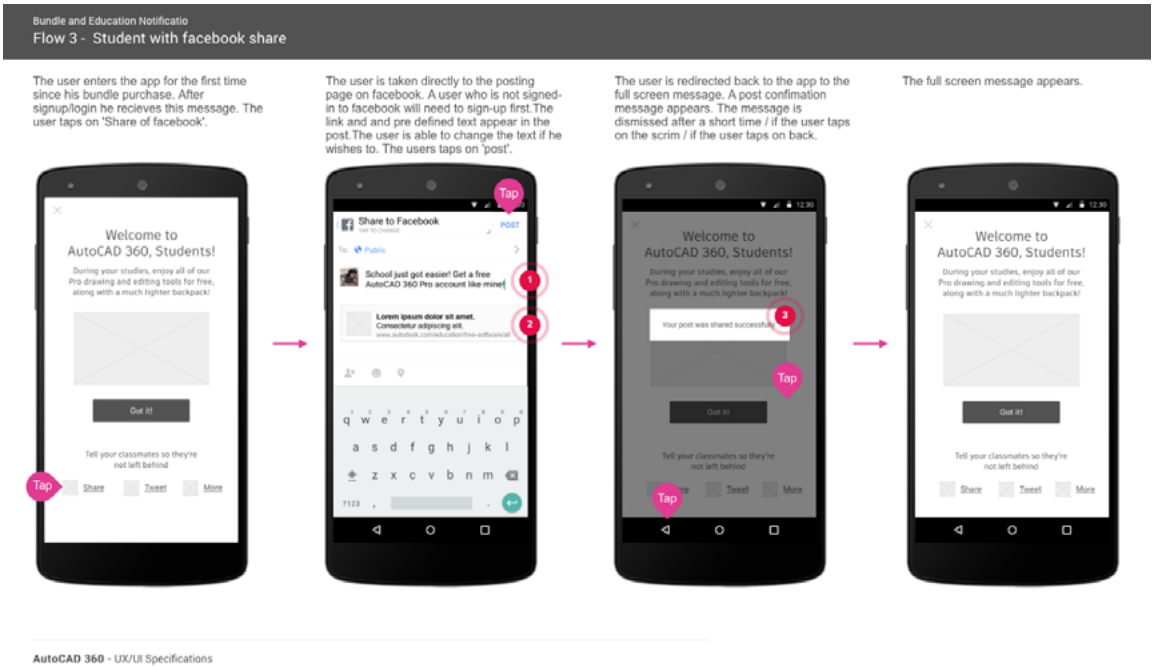
Designer Pro Tip

When mapping out product requirements, create your documentation as a knowledge hub rather than paper trail. Instead of writing out encyclopedic requirements, link out to artifacts with contextual documentation.



Lightweight PRD created in UXPin by Autodesk team

Autodesk's AutoCAD 360 team, for example, creates their product requirements document in UXPin as a living hub. They describe technical and UX guidelines, then include links to Zeplin and annotated user flows for the bulk of details. As the project progresses, the annotated designs reflect the latest prioritizations and technical needs.



Annotated UXPin user flow used by Autodesk team for detailed product documentation

Marcin Treder, CEO and co-founder of UXPin

Write Smarter User Stories

Most of you probably have some tried-and-true best practices that you lean on to formulate requirements and articulate what needs to be done.

In many cases, that may involve the creation of use cases and user stories. While both are valuable tools, neither go quite far enough in defining the problem and desired outcome. But as I think you'll see, a very simple change in the user story format will profoundly impact the finished product.

Designer Pro Tip

The greatest challenge I face as a designer is getting caught up in the design solution before understanding the why. For your

> *next project, I recommend writing a series of narrative-based user stories. A storytelling approach will help you dive into each step of a user's journey. It's a framework for you and your team to solve wicked enterprise problems.*
>
> Jessica Tiao, Product Designer at Crazyegg

What's Wrong With Traditional User Stories?

User stories aren't flawed; they're incomplete.

I'm not advocating doing away with user stories – I'm advocating that we make them more useful to our enterprise teams.

User stories don't describe a user's entire journey from start to finish, nor do they consider the motivations or needs that drive the journey. Typically no more than a couple of sentences in length, they stop short of explaining how users think and feel, and they don't address the business goals that should support every item on a list of requirements. As a result, user stories may inadvertently ask more questions than they answer.

I offer for your consideration a simple way to boost the power of your user stories: address the user's **goal** behind the action, as well as how that action solves a **problem** for the organization. In addition, this

simple change also illustrates the benefit the organization stands to receive from the user's action.

Instead of:

> *"As a **[user]**, I want **[function]**,*
> *so that **[action]**."*

Write it like this:

> *"The **[user]** wants **[function]***
> *to achieve the goal of **[user goal]**.*
>
> *The current inability to do this is causing **[adverse effect]** for the organization."*

You can also substitute that last sentence with:

> *"Enabling users to do this would deliver **[specific measurable value]** to the organization."*

This changes the user story from a statement of fact to a statement of **intent**. It opens the door to a new question: "*How do we do this in a way where we can track and measure its success?*"

Remember, we're talking about designing like a startup. All startups focus on the most important problem–the one with the potential to completely sink your ship, if left unsolved.

In this case, your overriding focus must be on the things that provide measurable *value* to users, and in doing so, back to the organization.

It really is *that critical* to your survival and success.

Task Completion vs. Success

Traditional user stories operate at a very tactical level; the focus is strictly on task completion, but I have a problem with that. **Task completion is not the same as success**. If you want better UX, you have to focus on success, which is where the value I keep harping on about lives.

Uncover and incorporate how users *feel* about the interaction, along with their intended **goal**, the reason they're doing it in the first place. You need to capture *why* a particular interaction or behavior provides a better user experience. You also need to capture why that experience, in turn, benefits the business.

Using the format above, your new user stories might look like this:

Task Completion

"As Jane the Bank Teller, I want a preset shortcut list of one-touch transactions so that I can complete more transactions."

Success

"Jane the Bank Teller wants a preset shortcut list of one-touch transactions, to achieve the goal of getting customers through the line faster to minimize their frustration and automate lengthy transaction sequences. The current inability to do this is causing a significant percentage of customers to leave our bank for a competitor."

For Jane, getting customers through the line and completing their transactions is **task completion**; she's *already doing that* now. Doing this same task better (in a way that is more accurate, efficient, and makes customers feel like the process was pleasant and *really fast*) is **success**. And when people feel like they're being taken care of, when their needs are met and expectations are exceeded, they remain *loyal*. When you shift your focus from what people do to why it *matters*, both your head and your feet are firmly on the path to creating powerful end-user experiences.

Designer Pro Tip

Today, enterprise users expect the same quality of experience as consumer products. But because stakeholders are not the target audience, they have a desire to continually add features for every perceived problem. Additional features don't always mean additional value.

Andy Vitale, Senior Interaction Designer at 3M Health Care

Use Low-Fidelity Prototyping to Generate Requirements

“That’s not how I imagined this would work when I wrote the requirement."

Most dev teams are resigned to hearing some variation of that statement at the *worst* possible point in the project. This experience usually leads to a great moment of self-righteous anger, during which they blame the stakeholder for not properly defining requirements.

As much as we all enjoy indulging that righteous anger, the truth is that the **blame for these situations lies squarely with us**.

Specifically, we’re failing to give stakeholders what they need to evaluate requirements for any gaps. Let’s talk about what typically happens here.

Requirements typically come from meetings where stakeholders talk about what they want and the design/dev team captures those items. Instead of drawing from actual **user research**, the meeting can be combination of personal opinion, political pressure, and knee-jerk

reaction to competitor features. This, to me, is the equivalent of placing a fast-food order for 12 million strangers.

There's only one way to prevent this. It's so simple that I am perpetually shocked that more organizations don't adopt it: **build a simple prototype as soon as humanly possible**, and iterate with stakeholders to generate requirements.

We are at a point in history where collaborative design platforms make it ridiculously easy for anyone to build a solid low-fidelity prototype in an hour or two for feedback. And even easier to export all of it in a few clicks to HTML that stakeholders can click through just like the real thing.

I've seen plenty of instances where a two-hour review session with my stakeholders (where iteration and review/discussion are happening simultaneously) has replaced one or more two-week sprints.

One quick note on prototyping – **low-fidelity means exactly that**. You *must* stick to the following constraints, otherwise you'll spend considerably more time than necessary, and you'll be inviting everyone who sees it to fixate on *what it looks like* instead of *what it does*.

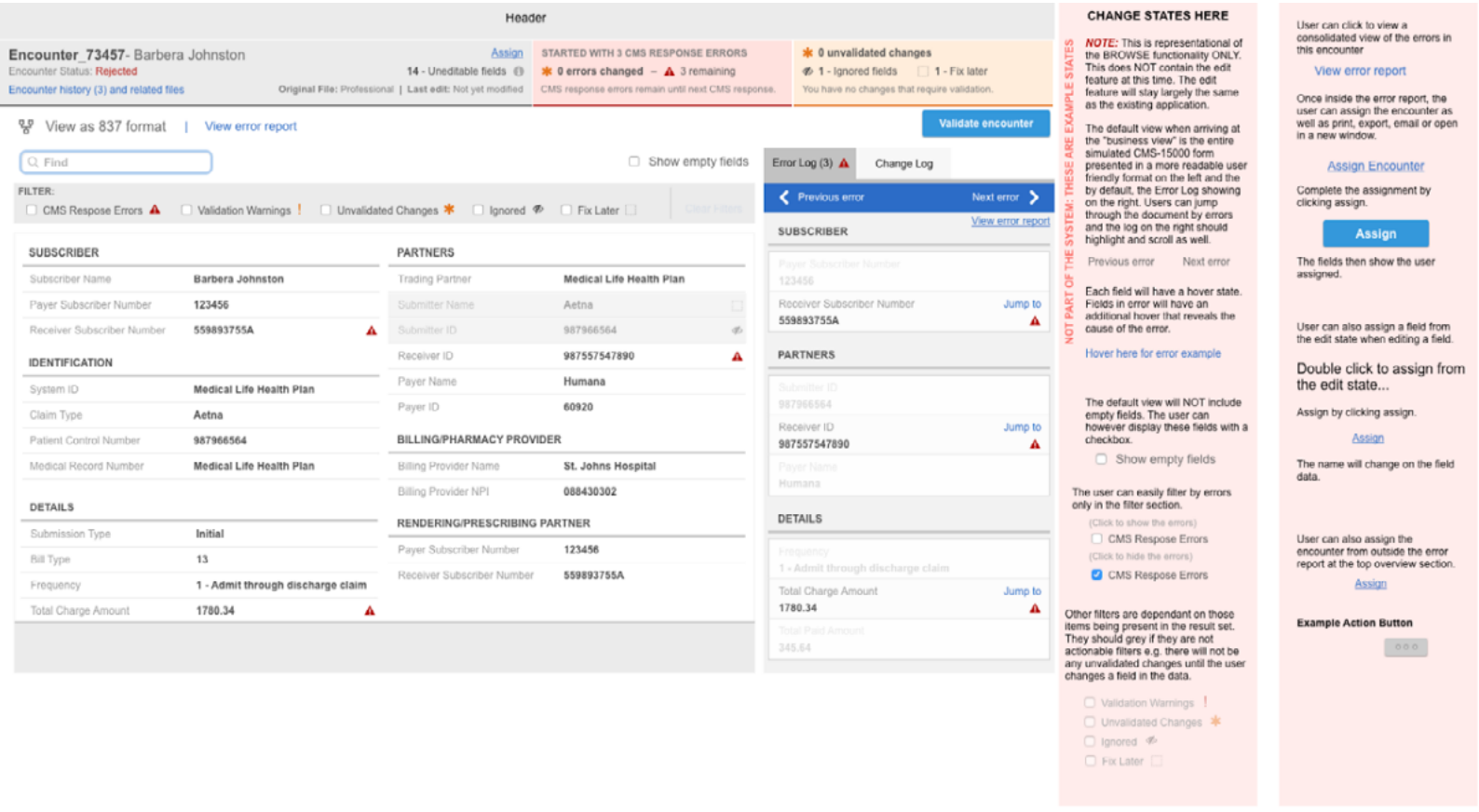
For startups, initial product prototyping and iteration has to be three things: **fast**, **inexpensive,** and **accurate**. They're facing extreme constraints, they need to get to reality quickly, and they can't afford to be too off target right out of the gate.

There is absolutely no reason an enterprise organization cannot operate the same way.

All it takes is following these constraints:

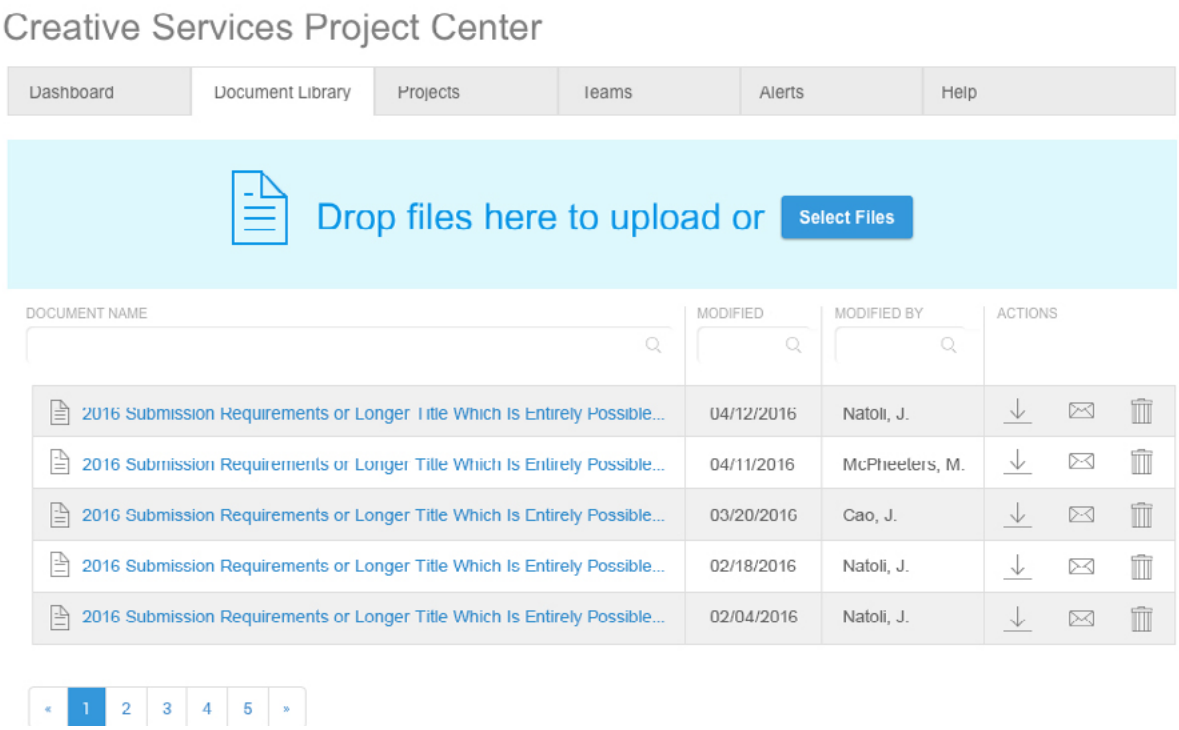
1. **Limited colors (use blue to indicate text hyperlinks)**. Everything else is filled with varying shades of grey or simple colors to indicate visual hierarchy.
2. **No images or graphical data displays**. The minute you introduce either, your reviewers fixate on them instead of casting a critical eye to screen layout, content structure, controls, interactivity, and workflow.
3. **No fonts other than Arial**. As with colors, fonts invite analysis and speculation.
4. **Real labels on all interactive components**. It doesn't have to be correct or final, but you definitely need feedback on navigation menu items, accordion content, data tables, buttons, form fields, etc. The only way to get it is to give people something to consider as a starting point.
5. **Real text content, to whatever degree possible**. Content goes a long way in establishing context, and context is the key to user experience. Design around content. A rough draft of your content is much better than "Lorem ipsum".
6. **Keep interactions simple**. Anything that requires significant, real coding for demonstration purposes is something you need to drop, re-think, or table it for your hi-fi prototype.

7. **Annotate to begin suggesting and documenting requirements**. It is your responsibility to communicate the rationale of all interaction decisions. Not only do annotated prototypes reduce misinterpretation, you create contextual rather than "paper trail" documentation. Try the **user story format** I suggest in previous chapters–it's the perfect format.

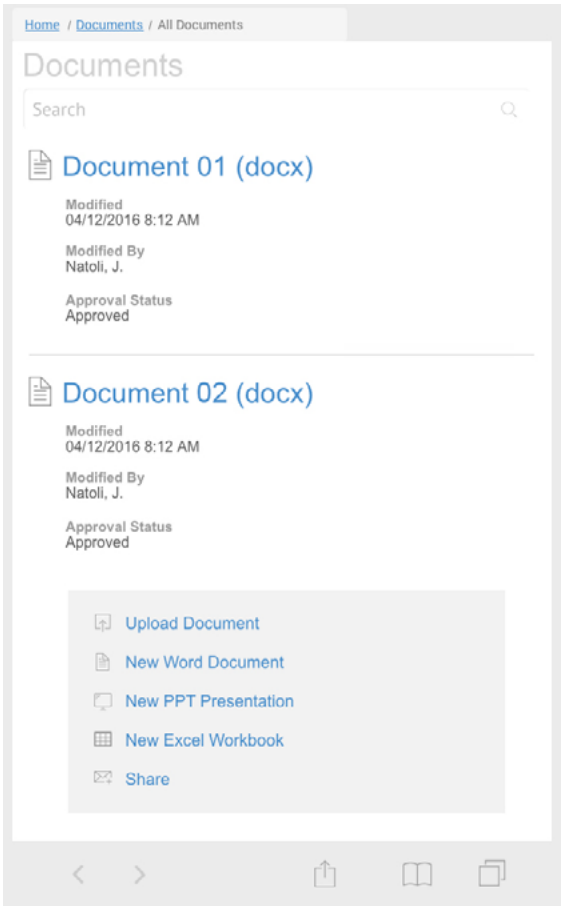


Instead of 30-page specs, LookThink creates annotated prototypes in UXPin to align design and development. Product shown is a client's healthcare data management platform.

To illustrate the fidelity level I'm advocating, here are a few prototypes I created of a possible enterprise document management platform:



Prototyped in UXPin



Prototyped in UXPin

After the First 48 Hours, Start Building Something

This one is simple, and means exactly what it says. If you're familiar with design sprints, then you understand the value of validating a prototype as quickly as possible.

At the outset of any project, we're always staring down significantly more questions than we can answer. That's the normal order of things, but it encourages a tendency to want to answer them all before doing anything.

The way to get clarity is to start working, start prototyping. For the teams I've worked with, the 48-hour rule usually plays out like this:

Days 1 and 2: Contextual Use Scenarios

We start with **stakeholders**, naturally. They're usually immediately accessible and are also itching to voice their concerns.

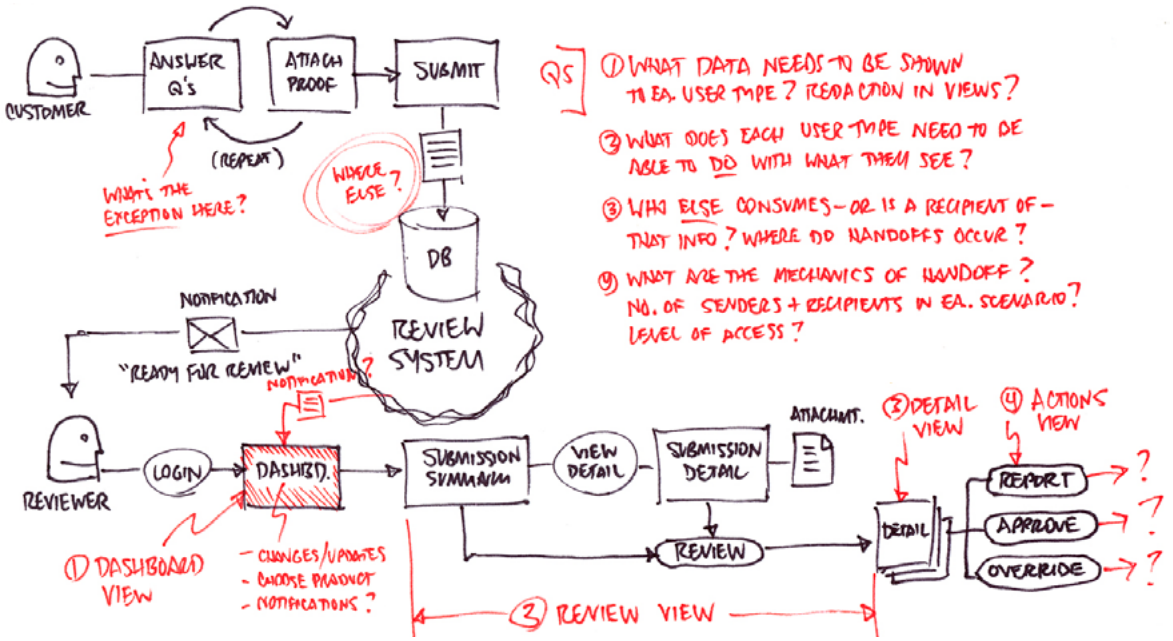
Our first order of business is to get the lay of the internal land, politically as well as tactically. We want to find out where each stakeholder is coming from, what they expect to happen and where their goals conflict with other departments. We also want to know what happens to each person's world if the project succeeds or fails.

The first 48 hours of the project is spent in meetings with stakeholders and users; each session focuses on a specific user group, or a specific department within the organization. If the product in question is used by employees, getting these folks to the table is fairly simple. If end-users belong to a customer, however, doing this may be a bit more challenging, as we discussed earlier. In both cases, we have anywhere from 6 to 12 people gathered around a table.

The day is spent walking through how people use the system now (if we're redesigning or updating) or how we believe they will use the new system (if it's a completely new product). In addition, we're **diagramming the process** as discussion occurs. The only tools necessary are a whiteboard and a table with a lot of seats. Boxes, arrows and questions, most of which consist of simply asking "why?" The drawing is what turns this into a working meeting instead of a verbal sparring match.

The sessions work like this:

We ask a user (or stakeholder/IT Manager/Account Rep) to **walk us through their daily work processes** verbally: “On any given day, how do you do your work?” If there are multiple tasks, activities and processes, we go through each one. As they talk, we draw people, boxes, and arrows on a whiteboard that describe what happens, which usually looks something like this:



The only deliverable from these meetings is a photograph of the whiteboard, shared with all involved. The **core process** is usually in **black** marker, with problem areas and related **issues** or **questions** in **red** or orange. The color separation allows us to focus on those areas quickly when we refer to it later.

The visual representation helps everyone get to clarity much quicker than if they had to imagine it in their heads. It also gives the dev team the ability to confirm and correct as we go: “does it work like this?”

For now, forget formal use cases, forget formal diagramming methodologies and rules. Just draw it out and label **who the players are** and **what they're doing**. As you draw, you ask questions:

- What *should* happen *here,* and what *actually* happens?
- What happens (or should) *next*?
- What can (or should) the *next person* in the process do with what they have?

The whiteboarding exercise is a very simple way of getting a baseline for what gets created, what gets acted upon, and how it all moves through any particular process. We're focusing on the people, their goals and how they do what they do every day.

When you're done, it's worth photographing the whiteboard for later reference.

At this point you may be wondering, "When does in-depth user research happen?" The short answer: **later**. While it's possible to work preliminary user interviews into the first 48 hour period, these are cursory shots across the bow: short, 15-minute conversations to get a baseline for who does what with the tool at hand. But if that doesn't happen at this point, that's OK – and here's why.

Some enterprise organizations are still reluctant to devote significant effort to user research and testing. We've know we're short on time and resources, so we need to make every every activity and decision count.

So instead of spending a lot of time with users up front, we'll break up our research into chunks, so (a) **we're not committing large blocks of time to this** at the expense of iterating design or development, and (b) **we're creating a low-fidelity prototype** to put in front of users, instead of asking them to *imagine* using it.

Day 3: Prototyping, Information and Action

While screen layout and interaction mechanics are legitimate concerns, the bigger fish here is the volume – and clear *separation* – of **information** and **action**.

Starting the prototype process early in a collaborative platform helps you come up with ways to separate the two as well. That matters a great deal, because this separation is the foundation of good UX and sound interaction design. Keep two things distinct:

1. Things the user needs to **know**, and
2. Things the user needs to be able to **do**.

Getting a handle on the relative importance of both is where you focus your effort. The latter is up-front and center; the former accessible but tucked out of the way and used only when needed – and in such a way that invoking it doesn't obscure or compete with the user action. You use the prototyping process to answer the questions that have arisen over the past two days of strategy discussions:

- How much **content and/or data** do we need to expose?
- How many different types or formats do we have (e.g. raw data, text, audio, video, etc.)?
- How much of the data is **static** vs. **interactive**?
- Where is it coming from – how many **sources** or systems?
- How will people **get** to all of it?
- How should it be **organized**, categorized and labeled?
- What matters most to users, and in what order of **priority**?
- How might we **stand it up** in the interface?
- What interaction patterns match what the user **needs** or **expects**?
- Is each pattern **appropriate** for the *volume* and *type* of content they're manipulating or viewing?

You and I both know how easy it would be to spend weeks trying to formulate answers to these questions. But we also know that isn't feasible, because we have neither the time nor the insight necessary to do so.

Instead, we quickly work towards a model that captures the volume of content and represents the ideal separation of that content from the mechanisms and controls. We're working quickly to iterate and explore data types, navigation, labeling and interaction patterns. We're socializing each iteration with our users, with the IT Manag-

ers or Account Reps responsible for helping them and/or with our stakeholders. We keep what works and throw out what doesn't. Try, evaluate, revise.

We'll use this artifact to guide everything that follows. We will absolutely revisit all of these questions as we test the prototypes with users, but **we don't wait for clarity before starting our prototype**. We need to put something in front of people as soon as humanly possible to determine whether or not we're on the right path.

If the Feature Remains Undefined After Two Sprints, Table It or Drop It

As a guiding principle, anything that moves more than twice through the review cycle I just described (requirements to prototyping) is too complex. Table it or drop it.

In my experience, if you can't figure it out after two passes, then it's either a complex issue deeply entrenched in associated processes or systems, or you're dealing with too many unknowns.

In either case, it waits. The *only* exception to this rule is a feature or function that is of **high importance** and **high feasibility**, as discussed in the previous chapter.

I'll give you an example from a project I worked on a short while ago.

Introducing The Mystery Messaging Feature

At the center of this scenario was a **messaging feature** – presented as simple, managed communication between user roles. It wasn't even part of the prototype, but quickly came up in a sprint planning session where two key stakeholders were in the room.

The spec from the stakeholders, in response to our barrage of specific questions, could be paraphrased like this:

- **We need messaging** – for evaluators to talk to approvers. But other roles might be involved. They should only be able to talk to each other, but there might be times when they need to talk to other departments. And clients. But only about certain issues.
- **It should work like Outlook**, except no mailboxes or sending anything. It could be like a comment thread, but not everyone can reply.
- **There shouldn't be a central messaging screen**, because it may need to appear on other screens.
- **What screens the messaging feature appears on depends on what's being asked**. And we need to notify everyone when a question is asked.

This is literally *all the information we got*. And I can assure you that all the clarifying questions you're asking in your head right now are the same ones we asked. I'm sure you're seeing the red flag lurking here. Anything this wide-open and undefined has **no business being**

a requirement. We didn't plan on including comment threads on every screen, and we weren't convinced that doing so even made sense. But since we could absolutely see the need for communication in context, we agreed to prototype it.

Sprint One Review

Two weeks later, we reviewed the prototype. The same stakeholder pipes up with this (again, paraphrased):

> *"When people log in, they should see a little badge, like on the iPhone, that tells them a message is waiting."*

Oh, a *badge*. Magically. Somewhere. It's all clear now. Remember, we haven't fleshed out this messaging feature at all. There was no plan to have any kind of central "messages" screen; messages were simply comments contextually related to a specific summary.

So two weeks later, our scope has expanded exponentially: we are now building some kind of messaging app, complete with alerts and notifications. Which no one is really sure is necessary, let alone how it might work. So we ask:

> *"Shouldn't there be some kind of alert or prompt that a customer sees when checking their status that this is requested/needed? How do we know they'll bother to check messaging?"*
>
> *"They will."*

Let's pause here for a minute. A team of 12 people will each work a minimum of 80 hours to prototype something wildly undefined which will probably exceed schedule and budget. We're spending almost $70,000 in internal cost based on two simple words: *They. Will.*

We voice these concerns as thoroughly and diplomatically as possible. We suggest it's entirely possible that this messaging component is big enough to warrant a **separate system** and project in and of itself, with the potential to be very time consuming and equally expensive.

But you know the end result: we run the next sprint.

Sprint Two Review

Two weeks later, we're reviewing again. We now have a messaging dashboard in place similar to Outlook's approach to email. Threads, filters, address book, the whole nine yards. In the middle of the review session, this gem comes out:

> *"We're not allowed to store these conversations."*

Wait, *what?*

> *"These examples could potentially allow customers to expose details that, by law, we cannot store on our servers."*

So we need three-way communication between evaluators, approvers, and customers, it must include or prompt the exchange of proprietary information, but we *cannot store that communication*. Which, as you

might imagine, makes it more than a little difficult for someone to *view* it.

A long conversation ensues with no answers. So we pivot and explain the internal cost of pursuing this for another two weeks against the remaining project budget, which we cannot exceed under *any circumstances*. After much gnashing of teeth, it's painfully clear this is a **black hole** of epic proportions.

At the very least, it suggests that a canned process for alerting and collecting missing data needs to be considered in detail, in a way that doesn't expose the customer or the organization to any degree of legal liability. No small thing, to say the very least.

We collectively agree to table it until after launch of version 1.0.

Speak Up – Or Face the Consequences

Over the past 30 years, I have seen multiple variations of this story play out across a multitude of products, projects, and industries. And we've all learned, by way of having our noses broken more than once, is that when something remains grossly undefined after two sprints, it needs to be tabled.

So when that voice in your head pipes up and says *this is going to be trouble*, listen to it. Voice your concerns clearly, thoroughly and, of course, politely.

One of the best pieces of advice I ever received is that **silence equals agreement**. If you say nothing, you are not only agreeing that something should be done, you are also agreeing to *do it*.

The price of that agreement can be very painful for everyone involved.

Dispense with Dogma

A great number of daily stand-up meetings are little more than protocol.

Everyone reports in, everyone else listens, and everyone has one foot out the door. No meaningful discussions, no problem solving, no meaningful action. Any meetings where people do nothing but stand around and listen should *never* happen again.

Status can and should be shared–by any one of the online or offline solutions available. Update it when things change–and that's it. Anyone who needs a meeting just to hear someone else's status is someone who is (a) too lazy to check the project site, and (b) unlikely to act on anything they hear.

A significant number of enterprise organizations that I know of have an **unhealthy, counterproductive attachment to dogma**.

Designer Pro Tip

We adapted practices from Extreme Programming, Agile, and Lean Startup. We didn't follow "agile with a capital A". The most steadfast 'rule' was that every single person on the team is doing UX, whether they're a back-end developer, front-end developer, or designer. A Database Engineer is just as important to UX as a Visual Designer.

Jeff Veen, Design Partner at True Ventures
(former Typekit CEO and VP of Design at Adobe)

Dogma is defined as "*a principle or set of principles laid down by an authority as incontrovertibly true.*" In this case, that dogma is typically of the misinterpreted Lean/Agile variety, and (paraphrasing my point man for all things software development Sander Hoogendorn) usually presents itself like this:

- **We must have daily stand-up meetings**, even in the face of overwhelming evidence that no one is prepared, paying attention, or acting on anything that is discussed.
- **We must follow every single rule/process/axiom in the Scrum Guide**, to the letter and at the expense of all else. I once read a sto-

ry about a developer with herniated vertebrae who was verbally shamed into standing by his PM and team, despite the excruciating pain he experienced doing so.

- **We must use a Project Management tool** (e.g. Mingle, Trello, Speedbird9, Jira Agile, etc.), even though the entire team sits in the same room and Post-It notes on a Kanban board would probably be infinitely simpler and 100X more efficient.
- **We must set up a modeling environment** using Enterprise Architect, even though this is a six-week mobile project.

That list can go on and on, and I am certain that some of you are compiling the rest of the list in your head right now. The truth is, no theory or principle or dogmatic practice can possibly account for the **infinite number of variables** pertaining to your people, your products, your project, your organization, and your industry. Pretending it can is a surefire way to paint yourself into a corner at every turn.

If you have clear evidence that some mandated principle, process, practice, or tool isn't working, *stop doing it*.

And keep in mind that this advice also applies to this guide: not everything here will apply to your situation or your organization, so don't be afraid to adapt, rework or even ignore these guidelines as necessary.

Designer Pro Tip

When running your sprint retros, consider a flexible format with 3 questions:

- *What should we start doing that we haven't already?*
- *What did we try that we should stop doing?*
- *What worked so well that we should keep doing it?*

To scale for larger teams, appoint key people as "Investigators" instead of asking everyone. Before the retro, give the appointed members a day to review the recent sprint. Encourage them to share insights with other team members prior to the retro. Treat the actual retro as a "meeting of the minds".

When it comes to standups (which we hold 2x a week), a dedicated Slack channel helps our product team minimize physical meetings.

Marcin Treder, CEO and co-founder of UXPin

Shorten the Path to Certainty

Obviously, there are some dramatic differences between enterprise organizations and startups. But when it comes to digital product design and development, they share the very same pain points and pressures.

We have an ever-shrinking window of opportunity to uncover obstacles to end-user efficiency. We have limited visibility into which of their complaints actually grind productivity to a halt.

Use is and will always be tied to motivation, and motivation will always come from some kind of reward – a sense of accomplishment, a feeling of increased competence or a specific goal achieved. Unlocking the things that drive this motivation and enabling it via solid UX provides the secret to startup success – and the key to enterprise longevity.

I've seen the practices I've shared with you here adopted successfully by enterprise organizations of all sizes across multiple industries – from small startups to massive companies like Autodesk and Paypal.

Those organizations put these principles into practice for one single, solitary reason: **they *work***.

UX improvement for big sites and systems begs equally big questions. But no matter your enterprise-driven constraints, I assure you that the principles we've explored here will dramatically shorten the distance to the answers.

If you found this guide useful, check out the case study in the next section to see how Sumo Logic improved up its enterprise UX process as it scaled.

Enterprise UX Process Case Study

Speeding Up Design Reviews By 300%

THE CHALLENGE

Based in the Bay Area with 250+ employees and $161 million in venture capital funding, Sumo Logic serves some of the top enterprises in the world. The company's analytics platform visualizes more than 100 petabytes of data per day, helping businesses harness the power of machine data to streamline operations.

In 2015, Sumo Logic hired their first UX team comprised of design leaders, interaction designers, visual designers, and UX architects.

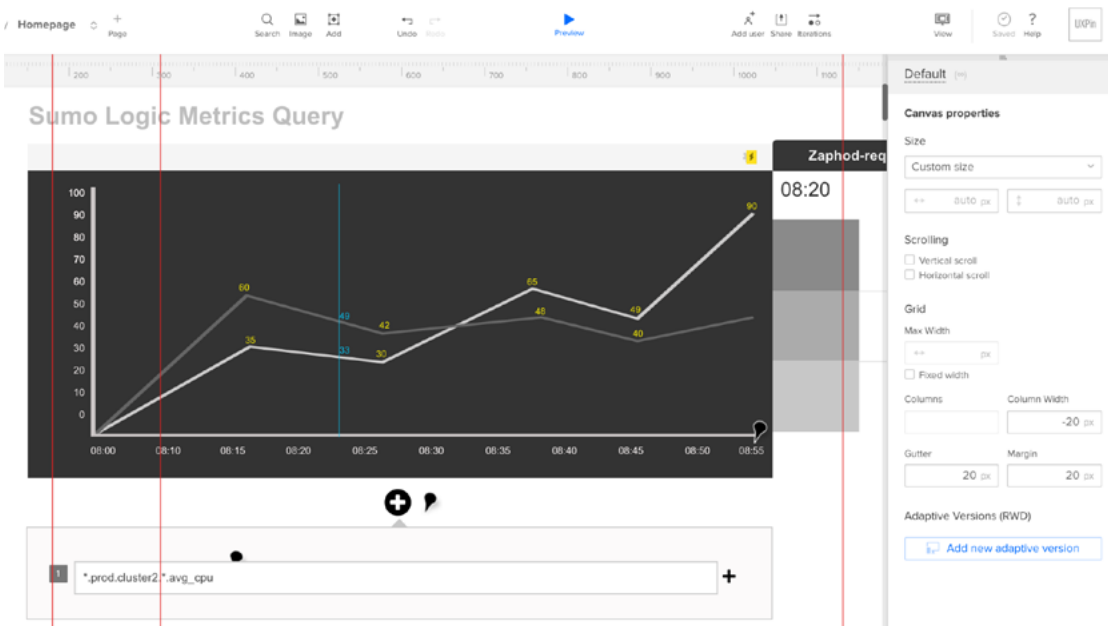
The company had been using Axure for wireframing but Design Director Daniel Castro quickly found that the solution did not allow for easy design modification and did not encourage collaboration.

Tired of sharing PDFs back and forth, the company needed a collaborative UX platform that could scale along with their teams and processes.

THE SOLUTION

For the first six months, Design Director Daniel Castro and his team spent much of their time holding happy hours and offering show-and-tells of great UX design.

In a culture already used to collaborative tools such as Slack, the slow process of emailing thoughts on static designs was stifling. Castro had begun using UXPin at his previous job, and knew it would offer his Sumo Logic team the collaboration tools they needed.



Sumo Logic prototype created in UXPin for their Unified Logs Metrics product

> *"We are constantly collaborating with engineering and product managers and it used to take a significant amount of time to work together going back and forth," Castro said. "UXPin allows us to easily show the flow and main components of our projects. We can share a link and everyone can communicate with our key stakeholders, expanding on each other's comments and allowing us to manage feedback contextually without redundancy. It's like a visual version of our thought process. We can even make comments on a pixel level. This has made our review process three times as fast."*

> *"UXPin has played a vital role in creating a design-oriented culture at Sumo Logic," Castro added. "The team is great to work with, and I'm excited to see what we can do next."*

THE RESULTS

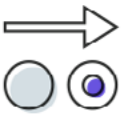
- Design modification is **quick and simple** with UXPin, instead of the limiting modifications possible with Axure wireframing.
- Design reviews are **three times as fast and now contextual** using UXPin to collaborate instead of emailing static PDFs.
- UXPin is "like gold" when trying to get approval from stakeholders on projects, **halving the effort** needed to communicate with stakeholders.

Want UXPin to help your team? Start a free trial now.

The Full-Stack UX Platform

Your entire UX process in one place

Design:
Create lifelike prototypes quickly with Photoshop and Sketch integration.

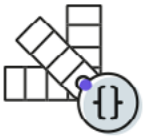


Iterate:
Built-in version control improves efficiency and eliminates confusion.

Document:
Cleanly annotate your designs. Insert custom code snippets that travel with elements.

Collaborate:
Get feedback and co-design on any project anywhere.

Scale:
Automate consistency and documentation with design systems (syncs with Sketch).



Implement:
Auto-generate style guides, assets, and specs for developers.

Try UXPin now