

```
(function(nsp) {  
  'use strict';  
  
  var title = ;  
  'The Designer\'s Guide'+  
  'to Collaborating'+  
  'With Developers';  
  var uxpin;  
  
  window.addEventListener(  
    'load', function() {  
      uxpin = new UXPin();  
      uxpin.setTitle(title);  
    });  
})(window);
```

UXPin

The Designer's Guide to Collaborating With Developers

Copyright © 2015 by UXPin Inc.

All rights reserved. No part of this publication text may be uploaded or posted online without the prior written permission of the publisher.

For permission requests, write to the publisher, addressed “Attention: Permissions Request,” to hello@uxpin.com.

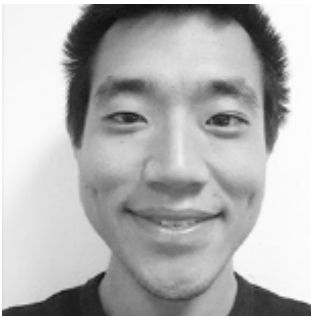
Index

Introduction	7
Less Painful Design Planning With Developers	9
Adopt Open and Honest Communication	10
Treat Documentation As A Dialogue	13
Engage in Feature Prioritization Exercises	14
Assemble a Realistic Plan	18
Wrap-Up	21
Wireframing and Prototyping for Collaborative Exploration	23
Wireframing for Developers	24
Design for Interaction	29
Test, Gather Feedback & Revise	32
Wrap-Up	34
Collaborating on Team & User Feedback	36
Sharing Feedback with Developers	37
Collaborating with User Feedback	41
Takeaway	46

Building Mockups Developers Won't Hate	47
Plan for Limitations	48
Dealing With Alternate Pages & Resources	54
Design Tips to Make Development Easier	58
Helpful Plugins	63
Wrap-Up	65
Collaboration Secrets for Designers	66
Follow Good Hygiene for Visual Design	67
Treat Style Guides as a Collaborative Tool	68
Share Ideas with Style Guides	73
Take an Interest in Development	75
List of Collaboration Tools	79
Wrap-up	81

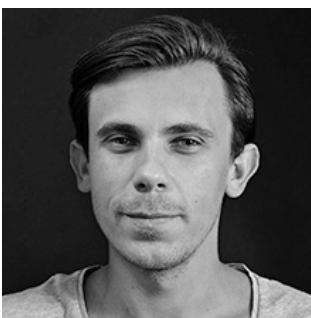


Jake is a writer and designer with a focus on interface design for the web. He often writes about W3C specs and the newest trends in web design. You can find out more on his [portfolio](#) view his latest design work on [Dribbble](#).



Jerry Cao is a content strategist at UXPin where he gets to put his overly active imagination to paper every day. In a past life, he developed content strategies for clients at Brafton and worked in traditional advertising at DDB San Francisco. In his spare time he enjoys playing electric guitar, watching foreign horror films, and expanding his knowledge of random facts.

[Follow me on Twitter](#)



Co-founder and head of product, Kamil previously worked as a UX/UI Designer at Grupa Nokaut. He studied software engineering in university, but design and psychology have always been his greatest passions.

[Follow me on Twitter @ziebak](#)

Introduction

Digital designers and developers represent two halves of the same coin.

While it seems like designers and devs speak two different languages, they're both an integral part of creative project work. The best creative projects result from splitting up the workload but maintaining a close relationship between everyone on the team.

In the enterprise, effective communication can be difficult considering the sheer size of teams and unfortunate reality of bureaucracy. But communication is a vital piece to every successful project. Although designers may not fully understand the development cycle, it's important that they treat developers as members of the same product team – otherwise your design will fall victim to unnecessary compromises and nasty surprises.

In this book, we'll cover practical tips for designers who want to work more closely with developers. Every creative project follows a series of stages from brainstorming to releasing the final deliverables. The

following tips analyze each stage of the process and help to bring teams closer together every step of the way.

For the love of UI design,

Jake Rocheleau & Jerry Cao
co-written by Kamil Zieba

Less Painful Design Planning With Developers

The most rigorous phase of brainstorming often occurs after receiving details for a new project. After all, most of the early ideation falls within the design team's jurisdiction since they're usually considered the "creative" minds.

But as dictated by modern [design thinking](#), you shouldn't restrict creative thinking only to design or marketing teams. Design is not a tactic, it is the art and science of solving problems for users. With the right guidance, usable design ideas can emerge from the roughest (and most unexpected) of origins.



Photo credit: "Every Design Challenge". Jo Quinlan. Creative Commons 2.0.

What this means is that developers are just as important to the design process because they actually build the ideas come to fruition. Never forget that developers are simply designers of code.

When starting any new project, invite as many designers and developers together into the process as early as possible. When it comes to [effective brainstorming](#), tackle issues related to both design and development so that everyone has a chance to air their grievances – but make sure only a small focused team has the final say.

With that in mind, let's examine a helpful framework for starting your design projects with the right people in the right direction.

Adopt Open and Honest Communication

The atmosphere in the room determines the success of idea generation. In order to make everyone feel open to suggestion without rebuke, create as objective a discussion as possible.

Brainstorming is a vulnerable process where people overthink and may take rejection too personally. Start off by speaking openly and letting everyone know their ideas are valued. Remind everyone to stay objective and focus more on the idea(s) rather than the messenger. Foster an environment of openness and camaraderie because it's always better to have too many ideas rather than not enough. Designers and developers may express their ideas differently, but remember that everyone has the project's best interest at heart.



Photo credit: [Sebastien Wiertz](#). [Creative Commons](#).

“Design may rely on aesthetics for its medium, and development may rely on code, but both draw on theories of efficiency to create effective output,” says Mozilla’s lead UX designer Cassie McDaniel’s in her [excellent Smashing Magazine article](#) on designer-developer collaboration. “Elegant code is efficient code, and elegant design is efficient design. This means that design and development share some core values of process.”

As we described in the first [Design Collaboration in the Enterprise book](#), set aside your ego and focus the conversation on the users. Developers and product managers are more receptive if you remind them of the user-based reasoning rather than its “just good design practice” (which is why personas are so important).

[Compromise is part of collaboration](#) and you can’t take anything personally. Time spent arguing over petty ideas is time spent chasing your tail. Pick your battles carefully and learn when it’s best to move on. When disagreements arise over prescriptive feedback, ask each

person to state the problem and then describe in detail how they arrive at the solution.



Photo credit: Ken Teegardin. Creative Commons.

As a cautionary tale of the importance of designer-developer collaboration, let's look at an example from Digg as described in [this piece from Smashing Magazine](#). During the early days of social bookmarking site, the lead designer came up with some new ideas for changing the vote button (which appeared simple in theory). However, the lead developer at Digg soon realized the changes would require a major reformat of the code architecture and server setup.

Luckily, in their case, the idea was killed after the early conversation. But imagine the ripple effects if they had each gone about their parallel paths, only to crash headfirst later in the process as they scramble to iterate and fix such a foundational error in assumptions.

While collaboration is important, you shouldn't compromise to the extent that you run into the issue of design by committee. Remember

that there is definitely a line between compromising on the aesthetics/interactivity due to technical constraints, versus **diluting the design** to satisfy all the stakeholders.

Treat Documentation As A Dialogue

First and foremost, you'll want to start out by reviewing all the top-priority ideas. As described in the *Guide to UX Design Process & Documentation*, the entire team will need to review some form of functional specs and technical specs documents.

Documentation is the starting point for conversation, not just a quick checklist for everyone.



Image credit: [Opensource.com](https://opensource.com), [Creative Commons](https://creativecommons.org/licenses/by-sa/4.0/)

At this stage, it helps to organize features into lists of required features, recommended features, nice-to-haves. You need to involve developers in these conversations because they are the ultimate reality

check. As the ideas evolve, keep the developer updated – otherwise you might find a “must-have” feature dictated by the product manager (and already built by designers as a hi-fi prototype) needs to be killed because it’s impossible to build.

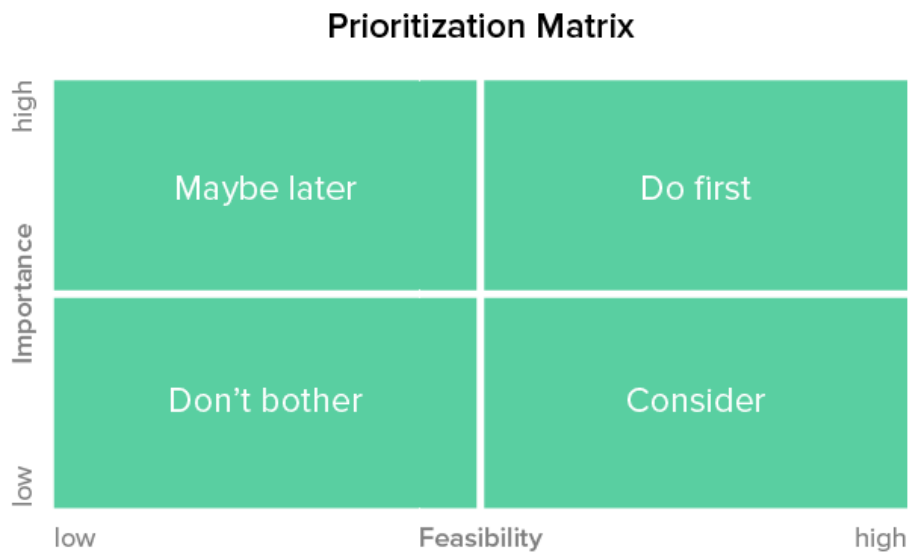
Everyone on the team should feel free to suggest other ideas beyond stakeholder suggestions. While the stakeholder’s suggestions may get the project rolling, these ideas are unlikely to form an exhaustive list. Open the floor to suggestions from designers and developers, which can include anything from big-picture layout design to smaller bits of [microcopy](#) or interactive elements.

Engage in Feature Prioritization Exercises

Once the team has a laundry list of ideas, you’ll want to organize them individually based on purpose and feasibility. Let’s look at the priority/feasibility plot, affinity diagramming, and a clever prioritization activity created by Pandora.

1. Priority/Feasibility Plot

The [priority/feasibility exercise](#) (originally developed by UX consultancy firm Adaptive Path) asks the team to score features on a scale of 1 to 5 for priority and feasibility. Everyone on the team casts their votes (Google Spreadsheet works well), then you plot the scores with feasibility on the X-axis and priority on the Y-axis.



Credit: UXPin inspired by Jason Bedell

To avoid design by committee, make it clear that this exercise is meant to just generate more data points for the product design team – it is not a substitute for expert decision-making.

2. Affinity Diagramming

An alternative method, the [affinity diagramming method](#) which is sometimes referred to as the **KJ Method**. It can be outlined as follows based on [Jared Spool's adaptation](#) (which we actually follow at UXPin):

- Start with a focus question, for example: “What new features should we build into our product?”
- Everyone suggests ideas on sticky notes or notecards. These ideas are then gathered into a single pile.
- Everyone organizes the notecards together into related groups. For example, these groups may refer to homepage elements, form attributes, branding, JavaScript effects, database features, etc.

- Once every idea has been grouped, the ideas are discussed at length and scrutinized accordingly. Some get tossed, while some may even spur new ideas.
- After you've decided which groups are important, prioritize the groups. Once you're done, you'll have a better idea of the rough feature set.

This organization technique is designed specifically for sifting through large collections of data, which makes it perfect for complex products.

3. Pandora's Workshop Exercise

Pandora's technique considers financial restraints by limiting features based on what can be afforded over a 3 month period. Although projects can run much longer or shorter, this methodology is highly adaptable for any creative project:

- About a week or two before the activity, take down ideas for suggested features. About a week before the activity, have everyone create a short slide describing their idea and its scope.
- Set a dollar estimate for the product team to design and build one feature over a certain period of time(1 month, 3 months, 12 months). These estimates can be used to calculate respective price limits for a budget. To keep it simple, Pandora decides that \$5 represents a feature that requires one month of work.
- Pick a group of people and give them post-its with their "budget". These post-its represent votes for how much should be spent

to build a particular feature. To prevent people from getting carried away, Pandora gives each person \$30.

- Encourage each person to spend their budget on the top features. Afterwards, the team can discuss which features seem most pertinent and feasible. We agree with Pandora's recommendation to give a small group of people (like the product team) the final say.

The above exercise helps Pandora cull down 80 rough ideas to a handful of feasible ideas. They've actually ran these 2-3 hour workshops every quarter for 8+ years now, which can't be all that bad considering their success.



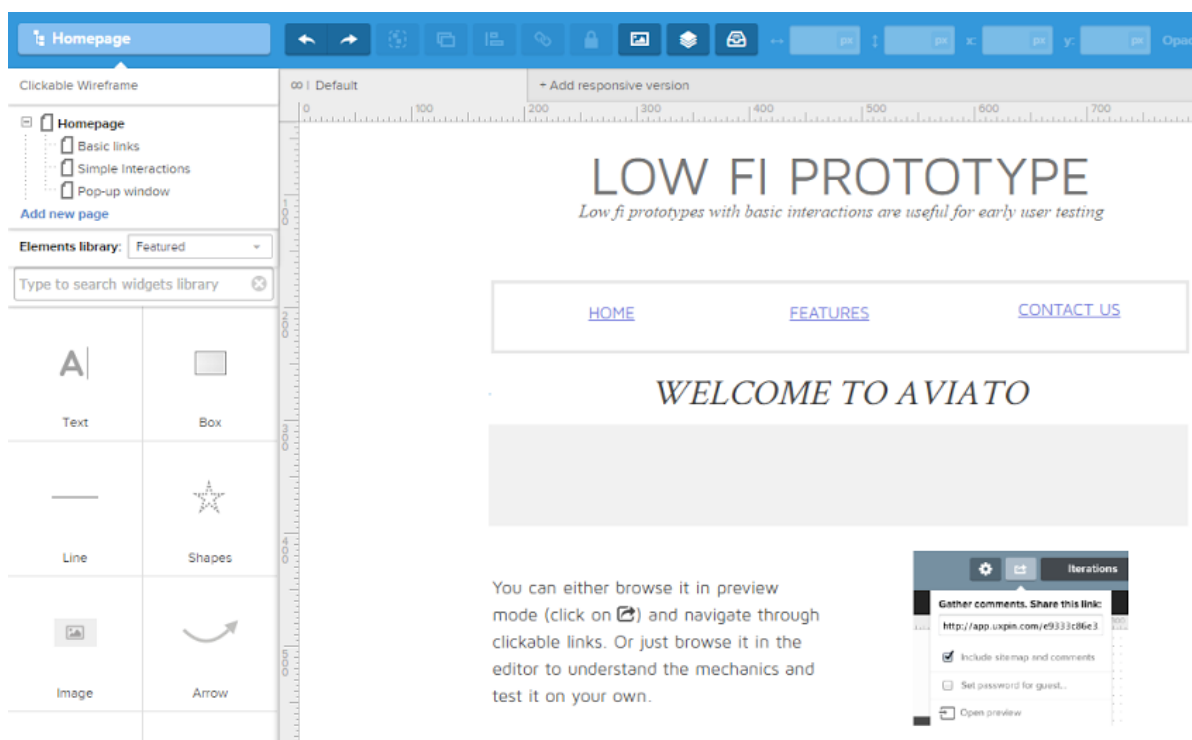
Source: "UX for Good Breakout. WIAD DC. Creative Commons.

Remember that it doesn't really matter which tactic you choose, as long as the final arrangement works and gets everyone thinking simultaneously about technical feasibility and design priorities.

Assemble a Realistic Plan

Once you've successfully prioritized ideas, the next step usually involves preliminary design work. You'll likely do some rough sketches, iterate them to wireframes, or perhaps just dive directly into low fidelity prototyping.

Although wireframing seems like a designer's task, developers can lend an ear for suggestions and an eye for critiques. For example, it helps for designers to get feedback especially on the UI patterns since they are a common component to almost every page on the site. The reality is that developers will need to fill in some design blanks, as [Paul Boag suggests](#). The earlier you involve them, the sooner they can see the "why" behind design decisions, which will help them exercise better judgment later.



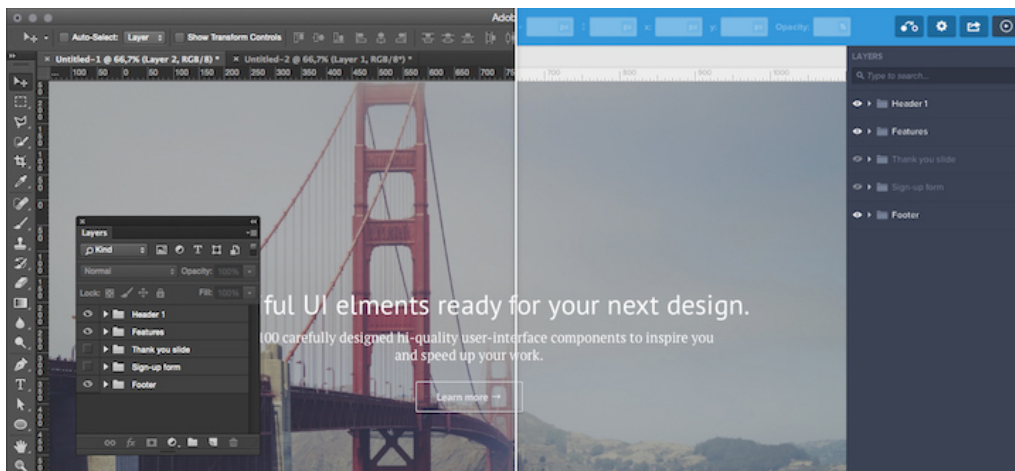
Source: [UXPin](#)

By outlining a plan of attack each stage can move along quicker and smoother. For example, the team could start by listing important phases of the project and what needs to be accomplished.

Critical phases usually include (not in linear order):

- **Wireframing** – Low-fi ideas, UX exploration, layout structure.
- **Prototyping** – Interactive elements, animated effects, page flow, UX testing.
- **Mockup design** – Full website mockup creation including icons, textures, colors, typography, and accurately-positioned elements. This may also include inner page designs and responsive changes based on viewport.
- **Frontend development** – Building design assets, coding the layout, browser support, responsive changes, complete CSS/JS animations.
- **Usability testing** – As described in the [Guide to Usability Testing](#), this ranges from simple hallway tests with 5 co-workers to unmoderated remote tests involving hundreds of users simultaneously.

Given the rise in popularity of [Agile UX](#) methodologies in which designers work a sprint ahead of developers, the key to success lies in validating ideas based on internal and external feedback. When possible, conduct a round of usability testing and internal feedback session (involving stakeholders and developers) every time you iterate the design or make dramatic changes to the code base.



Source: *Prototyping with Photoshop & Sketch*

For instance, you might start with a sketch or low-fidelity paper prototype. Even in this rough stage, your feature set is still tangible enough for feedback from developers on system feasibility. If you decide to use [UXPin](#) for the digital design, you can then craft a wireframe (or make it a prototype by adding interactions) and simply tag developers in your comments. As the fidelity of your design increases, so too must the precision of feedback. Once you've finalized the visual design and [imported your Sketch or Photoshop files](#) into UXPin, you can again invite developers to your project to ensure all the details are still feasible (before you refine the interactions).

Of course, this project roadmap cannot succeed without a solid [kickoff meeting](#). Make sure you involve developers in your kickoff meetings. Don't run a kickoff meeting as a 30-minute block to review documentation and timelines. Make it collaborative with [helpful exercises](#) like the [design studio technique](#).

Because design projects are completed as a team, they must always be started as a team.

Wrap-Up

Team collaboration isn't something that just spontaneously happens in an enterprise environment. Just like any good relationship, teamwork requires work.

When everyone on the team operates fluidly, it creates an atmosphere of one large team instead of smaller divided teams. That's exactly the environment you must create, since everyone really is **working on the same project** regardless of their role and each creative stage is just as important as the next.

As a designer, your role is not just limited to design. Developers may write the code but this is just a means to an end – this "end" being the final product, not the deliverables.

Wireframing and Prototyping for Collaborative Exploration

Although wireframing and prototyping are two different tasks, they both represent a web or mobile interface in its most fundamental stages.

Meant as an explorative exercise and an early visual specs document, wireframes are a natural part of the [design process](#). And as we described in the [Ultimate Guide to Prototyping](#), you can then turn a wireframe into a low-fi prototype by adding interactions or jump directly into a paper prototype.



Photo credits: [UXPin](#)

Just because the design team creates the wireframes and prototypes doesn't mean that developers have nothing to offer. In fact, developers can share early opinions while the wireframes are still rough, which helps catch small nuisances and even offer potential improvements for rough layout concepts.

In this piece, we'll explore some guidelines for wireframing and prototyping so that they communicate the experience as clearly as possible to other designers, developers, and stakeholders.

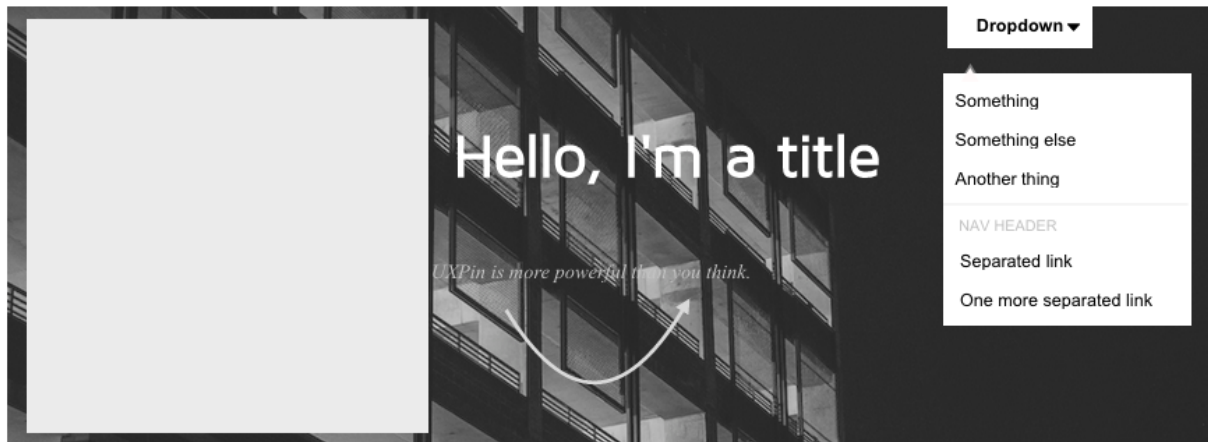
Wireframing for Developers

The best way to approach a wireframe is as a [blueprint for the product team](#).

Before adding any details, you must first lay out the headers, footers, content areas, and the relationships between these page sections. In our experience, we prefer to first wireframe the homepage (so we can start to think about content flow at the broadest level), then dive into landing pages and finally secondary pages (like About Us or Contact).

If developers are expected to code an HTML prototype straight from wireframes, then it's important to give them as much information as possible. In this case, you'll want to lean towards higher fidelity with crisp typography, richer colors, and high-resolution images since the wireframe will serve as the primary visual reference. If you prefer wireframing on paper, we recommend the ["sketching in](#)

layers” technique since it provides more structure and detail than standard freehand sketching.



TAKE ADVANTAGE OF GOOD TYPOGRAPHY

You can use a ton of beautiful fonts from Google Fonts library. Need something simple and nice looking, or maybe something more fancy? We've got them all. In total, there is 100 most popular Google Fonts to choose.

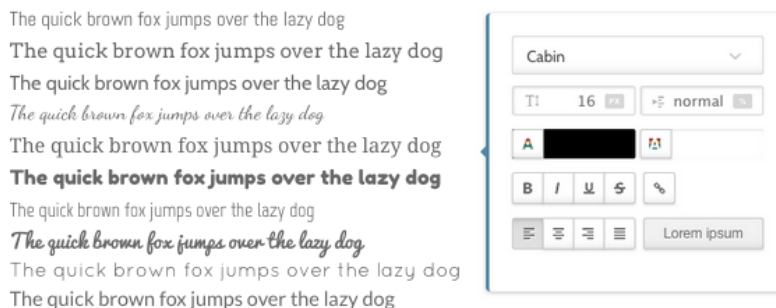


Photo credit: [UXPin](#)

On the other hand, if you're going to iterate the wireframe into a **rapid prototype** using a specialized tool, then the wireframe doesn't need to be as formal (although annotations are still helpful for later reference). In this situation, your prototype instead serves as a living representation of your technical specs.

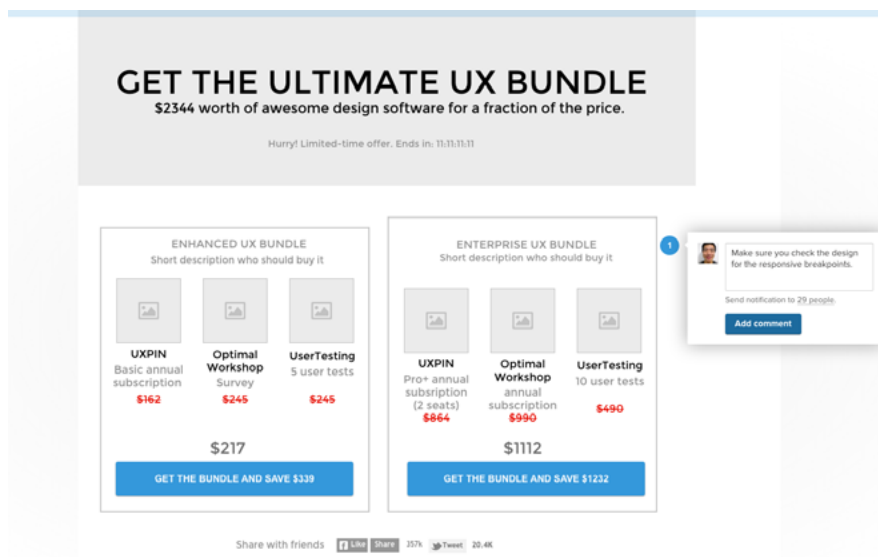


Photo credit: UXPin

When wireframing, clear explanations are vital because designers, developers, and non-technical stakeholders should be able to review your designs and quickly understand how they work. The best wireframes incorporate notes in the margins expressing which elements should be clickable, animated, or dynamic in any way.

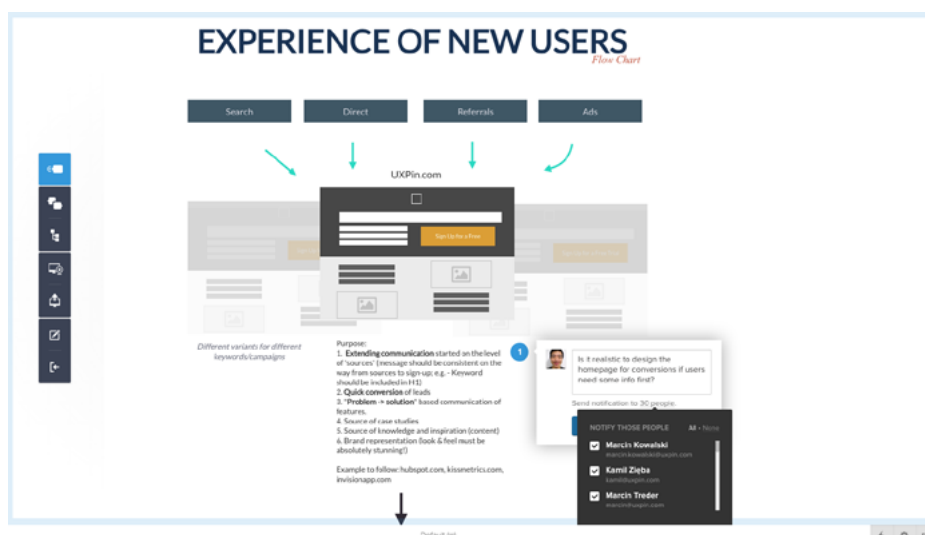


Photo credit: UXPin

But you can also create similar documents for telegraphing the intent of pages or page elements to developers. Here are a few common

supplementary documents (which we first described in *The Guide to UX Design Process*):

- **Storyboards** – Storyboards focus on the movement of elements and which links direct to which pages, creating a visual narrative for the entire team. Some links may not even load pages, but instead trigger effects like dropdown menus or modal windows. Created through [sketchboarding](#) or digital mediums, storyboards give developers a snapshot of the entire experience so they understand why some complex interactions might be necessary.



Photo credit: "Customer journey storyboard." [visualpun.ch](#). Creative Commons.

- **Flow charts** – Flow charts are similar to storyboards except they focus more on content itself. A [website flowchart](#) usually resembles a visual sitemap, except in a more multi-directional format that is more descriptive than the traditional tree structure. Detailed flow charts can be extremely helpful for developers since you could annotate how content is served (e.g. whether a lightbox is triggered via Javascript event or AJAX). If you lack the technical knowledge, map out the site pages in the flowchart, then collaborate on the technical notes with the developer.

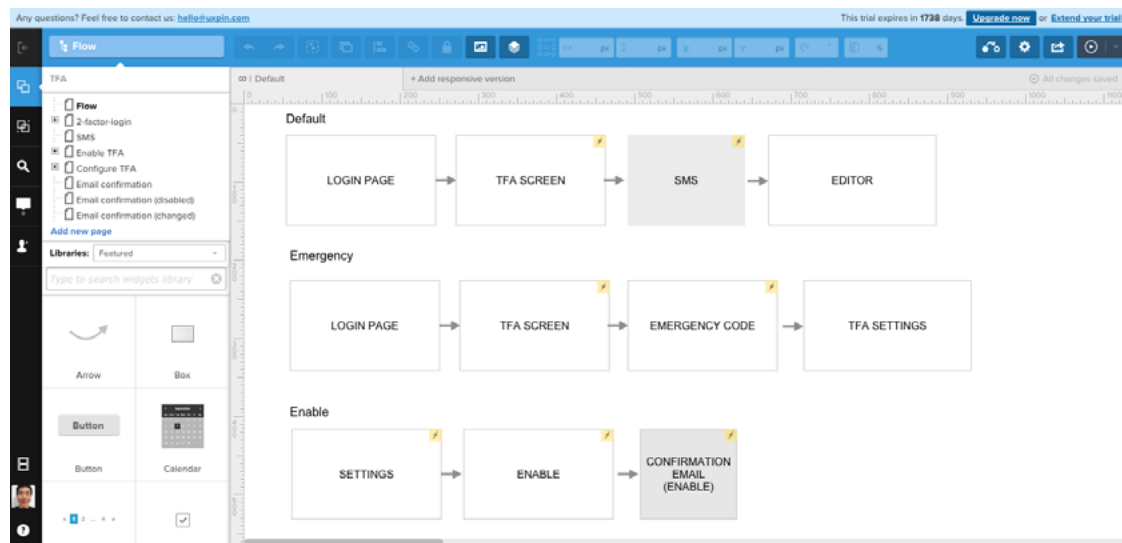


Photo credit: [UXPin](#)

- **Personas** – Like we previously discussed, the strongest way to align design and development is to focus everyone on user needs. Don't just include job titles and demographics. Dive into fears, ambitions, behaviors, goals, and habits. Create [lightweight personas](#), print them out for the product team, and let them become everyone's best imaginary friend.

JONATHAN VIZZIER

"Design isn't just how it looks, it's how it works."

Demographics:

- 27 years old
- Masters in Visual Design
- Visual Designer
- Single
- Earns \$85K per year

[list text](#)

Behaviors & Beliefs:

- Obsessive over visual quality
- Hates when product managers use the word "just" before describing last-minute tasks
- Wants to be as involved in the design process as possible
- Loathes jargon, wishes people would get to the point

Characteristics & Attributes (0 to 5)

- Design experience: 3
- Education: 4
- Tech Savviness: 5
- Ambition: 5
- Workload: 5

Goals:

- To build a strong portfolio, regardless of whatever job I'm at
- To start mastering UX design by the end of this year for a career transition
- To rise up in his company and start getting assigned larger-profile projects
- Wants to help the product team see the value of emotional design, not just "core KPIs"

Photo credit: [Persona Tool](#)

While you want to keep documentation as light as possible, don't underestimate its power for updating developers as fidelity increases. In our experience, developers tend to be highly logical people who enjoy the security of tangible visual requirements. When you treat design documentation as a collaborative tool rather than a "hand-off-and-pray-for-the-best" document, you'll find that developers will know what to build and designers will have a quick reference for popular and unpopular ideas.

Whatever your documentation methods, make sure they complement rather than supplement your design process.

Design for Interaction

When putting together digital interfaces, look at all the design features as a set of jobs. Each element on the page must help advance users toward completing their goal(s).

As described in *Interaction Design Best Practices*, it can be a good idea to think in terms of components rather than page elements. Which components on the page require action? What task does this action perform?

An interesting point to note is that developers speak the language of interaction. Developers review a wireframe/prototype and mentally formulate how it should be built. In some instances, developers might even mirror a rapid UX prototype with their own quick HTML/CSS/Javascript prototype to test the interactions.



Photo credit: "Brain Picture". [About Modafinil](#). [Creative Commons](#).

This is the beauty of designing for interaction: it helps everybody in the creative pipeline.

When designing for interaction, you'll spend most of your time in prototypes. It's harder to wireframe interactions since you'll need plenty of annotations, but it is certainly possible (and especially helpful for mobile interactions where you want to quickly show gestures).

Wireframes typically fall into two categories: low-fidelity (gray boxes and shading) and mid-fidelity (more detailed layout with crisp typography, images, and even real copy). Prototypes span the [entire spectrum of functionality and fidelity](#), ranging from paper prototypes on the low end to life-like representations on the high end.

Mid-fidelity wireframes are also used as a middle ground between static and interactive design and are usually less common in today's age of rapid prototyping. Mid-fi wireframes could include other documents like the storyboards we mentioned earlier. A traditional

workflow starts with low-fi wireframes which are then critiqued by designers and developers and iterated into low-fi prototypes and finally hi-fi prototypes.

Sometimes designers and developers prefer to jump right into a project by moving onto prototypes from basic sketches. Truthfully, there is no "right" workflow since every team will have their own unique style.

Regardless of your process, the most important thing to remember is that you design just the right level of fidelity. Otherwise, you might jump headfirst into a hi-fi prototype (dragging your developer with you on complex discussions around animations/interactions), when all you need is a low-fi prototype to explore structure and page flow.

Be proactive and set the right expectations around the design, because stakeholder feedback also indirectly affects developers. When possible, work with the product manager or project manager to outline a project map so everyone knows what to expect regarding the number of drafts, possible revisions, and levels of fidelity for prototypes & wireframes.

Test, Gather Feedback & Revise

Make sure you invite developers to all feedback sessions, whether it's a simple [user interview](#) a [moderated usability test](#), or an internal feedback session.

1. Conducting usability testing with developers

Not only can you both synthesize feedback in real-time, collaborative user research creates a stronger sense of joint ownership and creation. Interestingly enough, Xerox actually [sends their designers with service engineers during on-site visits](#) to deepen their understanding of how customers use the products.



Photo credit: “[Wikimedia User Testing](#)”. Blue Oxen Associates. [Creative Commons](#).

Of course, collaborative usability testing isn't just about sending pairs of people into the field to observe users. As we described in the [Guide to Usability Testing](#), it can be as simple as drafting up the core tasks together, then presenting the tasks to 5-7 users and asking them to think aloud as they interact with the prototype. Study which features draw the most attention right off the bat

and which receive negative reactions. Then after the test is over, ask follow up questions with your developer about where they felt most confused and experienced greatest difficulty.

2. Team Feedback & Fixes

Usability tests are meant to pinpoint issues or potential problem areas, but they don't always generate solutions. This is why teamwork and further discussion is required to flesh out the best solutions for individual problems.

Here are five UI/UX ideas to keep in mind regarding [usability 101](#):

- **Learnability** – Can people learn & adapt quickly?
- **Efficiency** – Are specific tasks easy to accomplish?
- **Memorability** – Does the interface leave a lasting impression?
- **Errors** – How are mistakes or inadvertent actions handled?
- **Satisfaction** – Does the interface provide a pleasant experience?

You can apply these questions to both the interface as a whole and to smaller features. For example, how learnable is your dropdown menu? Would it be more efficient to speed up the animation? Do users recognize that navigation bar links have dropdown menus?



Photo credits: “Question!”. Stefan Baudy. [Creative Commons](#).

Rather than scrounging up ideas on your own, it’s quicker to devise solutions as a team. Designers should always look toward developers for pragmatic suggestions. For example, if you’re on a tight timeline, the developer may have an elegant technical solution that needs some visual massaging.

Wrap-Up

Early design stages like prototyping and wireframing provide fertile ground for collaboration. These methods of visualization are meant to convey how the final product looks and behaves.

Ideas flow much easier when incorporating suggestions from both designers and developers. Developers can learn a lot about design and what makes the interface work. Developers who sit in on usability

meetings will also absorb new ideas that later improve their ideas in the development stage.

When it comes to the UX design process, everyone is responsible for keeping the team involved and moving forward.

Collaborating on Team & User Feedback

Giving and receiving feedback is the proving ground for how well you're working with your developers.

First off, how you communicate criticisms to each other, and to what degree these criticisms are implemented, is right away a marker of how well you collaborate together. On top of that you have feedback from others – most importantly the user. How you work together to prepare, conduct, and then analyze your usability tests is also telling of your compatibility. Luckily, neither of these aspects of feedback are set in stone, and you can make large leaps forward by using the best practices we'll list in this chapter.



Source: "Data portability escalator." [Todd Barnard](#). [Creative Commons](#).

Let's begin by discussing feedback within the team, then explore receiving feedback from users.

Sharing Feedback with Developers

Ideally, because of the same business goals, a designer and developer should be able to share and incorporate critique without incident, just another mechanism of a well-oiled machine. However, the reality is that we're all only human, and feelings and egos make sharing feedback a little messier. On the bright side, there are some simple strategies you can follow to ease communication and achieve the best results, all while remaining human.

1. Treat Developers Like Stakeholders

Apply the same principles of collaborating with stakeholders to your developers. For starters, this means treating them – and their opinions – with respect. You may not always see eye-to-eye, but you have to at least recognize that they may know things you don't.

Take this a step further and apply the same strategies to developer interviews as you would if they were stakeholders. This means eliciting their “gut feelings” – What are the real opinions and thoughts on a project? How do they feel about it deep down? Pushing this line of questioning will yield a lot of usable criticism and maybe even inspire some creative solutions.



Source: “VFS Digital Design Mexico Intensive.” [Vancouver Film School](#). [Creative Commons](#).

Another shortcut to building trust is to assure them some answers are off the record. Feedback is only as useful as it is honest. Alleviating the pressure of repercussions will allow your developer to speak candidly. This will open up a lot of doors about potential risks and genuine criticisms, both of which, while unpleasant, are what’s best for the project.

For more details on how to gather the best feedback from stakeholders – and by extension, developers – read [Kim Goodwin’s insightful checklist of interview questions](#), especially her recommendations for [interviewing engineering stakeholders](#).

As Goodwin suggests, you might find that some developers treat designers cautiously due to their previous experience of implementing near-impossible designs purely for the sake of originality. She makes an excellent point in recommending that instead of asking what’s doable and what isn’t, ask what is difficult and why. That slight tweaking of language can ease the minds of developers who might otherwise feel pressured to committing so early in the product development process.

2. Encourage Proper Communication Skills

Don't take smooth communication for granted. Expressing oneself to others is a skill that not everyone has, in which case it must be learned. We discuss this aspect of collaboration in detail in our *Design Collaboration for Enterprises: Book I*, but we'll reiterate some key communication tactics here:

- **Critiquing is a conversation, not a command** – Critiques should be the start, not the end. Good feedback inspires conversations and possibly enables brainstorming. Allow both sides to exchange their opinions, and look for a common middle ground, if it's available.
- **Phrase feedback as a problem, not a solution** – For example, “The ease-in animation doesn't feel as smooth as it should, which can impact conversion rates since it detracts from the feeling of sophistication,” is a lot more constructive than something like, “Make the animation ease in at 400 ms, and use the exact colors in the prototype.” Perhaps there's a technically nuanced issue that just arose during implementation, and phrasing feedback as a problem opens up a discussion that may satisfy design sensibility and feasibility.
- **Probe with follow-up questions** – To help developers articulate their feedback or pushback, try asking additional questions. The end goal is to hear honest opinions, but this doesn't come naturally to some speakers. [Dustin Curtis suggests asking three questions](#) to accurately isolate the heart of the criticism. We've

found his advice to be pretty realistic, since you don't want to start playing the question game and just keep asking "Why?"

Feedback sessions should feel casual, not like an interrogation or a court hearing. The same people skills apply at a product team meeting as they do at a cocktail party.

3. Express Ideas in a Practical Way

How you explain your feedback – not just in phrasing, but in context – will affect how it's received. One basic rule is to avoid too many details or lingo that may confuse each other. Designers and developers oftentimes speak different languages, so try to express your ideas in layman's terms so everyone can understand.



Source: "VFS Digital Design Mexico Intensive." [Vancouver Film School](#). [Creative Commons](#).

Another rule that's always helpful is to explain a point in the context of its implications on the user.

For example, you could waste your breath talking about how vertical alignment is off by a few pixels, but you'd just risk having the gravity of the point being misunderstood. However, if you explain

the same point as misalignment of content affecting readability (which affects time on site and therefore conversions), other stakeholders will start to pay attention. You'll find people's ears perk up once you connect design reasoning with business implications.

Collaborating with User Feedback

Each member of the team brings their own specific expertise, which means each member can provide feedback vital to the entire project. However, one person's feedback is by far more valuable than the rest: the user.

Collecting user feedback through usability testing and research is essential for the success of your product, no matter what you're building. The internal team members could sit and theorize about the best methodology all day, but none of it matters until it's verified by actual users. Whether for validation or new insights, it's best to get your answers straight from the source.

1. The Usability Testing Process

Collaboration must begin before the tests even start. As soon the plan for the test is completed, you should share it with your developer for their personal feedback. You're looking for their input in two areas:

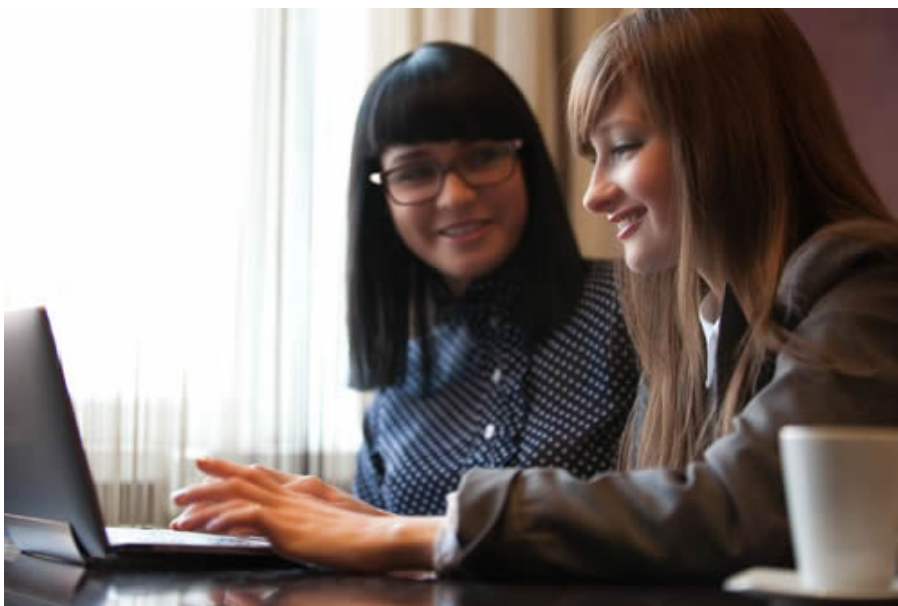
- **Suggestions on the existing plan** – Have them review the plan for any potential improvements, especially in the phrasing of

the tasks. At the very least, showing them the test beforehand will give them a deeper understanding of the core tasks the system must support.

- **Target data previously missed** – Developers may bring to light areas in which they need data, or at least would like some questions answered. Give them the opportunity to request certain data from the test before it's already over.

Once it's time for the actual testing, why not invite the developer to join in (if the tests are moderated)?

This first-hand observation will help them understand the user more fully, and give them the chance to pose questions directly. Just remember to select a single “leader” beforehand, so you don't confuse the user with two competing authority figures.



Source: [Damian Rees](#), via [Smashing Magazine](#)

If you're on a budget or strict timeline and can't do onsite tests,

we've compiled this list of our favorite usability testing tools for conducting remote user tests with multiple users simultaneously:

1. UserTesting
2. Usabilla
3. Lookback
4. Userlytics
5. Optimal Workshop

For more usability testing resources, check out [this list on Crazy Egg](#), and [this general guide from Mashable](#).

2. Collaborative Testing Activity: Rainbow Spreadsheet

As a way to strengthen team bonds and produce better feedback, Google UX Researcher Tomer Sharon recommends creating a [Rainbow Spreadsheet](#), an observational sheet that lets multiple team-members record their interpretations of the test in real time.

Starting with a Google Docs spreadsheet listing each user, encourage your team (or yourself, if they're too busy) to write observations in the left hand side. As your team notices these concerns in your user, they need only to fill in the colors accordingly.

	User 1	User 2	User 3	User 4	User 5
User feels interface is overwhelming	Red	Orange	Green		Blue
Prefers "search" over browsing the categories		Orange			
Requested that "Accepts Credit Cards" be a top-level filter	Red		Green		
Wants photo gallery accessible on results page to assess restaurant ambience			Green		
Bookmark feature was frustrating	Red	Orange		Yellow	
Needs clearer indication of price ranges	Red				Blue
Felt it was easy to sort restaurants by "Open Now"			Green		
Could not find the Events tab				Yellow	

Source: [UXPin](#) for Yelp design usability testing based on exercise [suggested by Tomer Sharon](#)

The colors make the results more comprehensible, especially to non-designers. The downside, though, is finding a time in which everyone can participate. If your company prefers a more formal report, you can always create one afterwards, using this as a summary document.

3. After the Testing

Once the testing is completed, properly collaborating on the results will allow everyone to process them more accurately.

The first step is allowing everyone access to the results. This means uploading all relevant documents – reports, graphs & figures, media such as videos of the tests, etc. – into a shared cloud folder if that’s available. If not, a mass email or even photocopies will do.

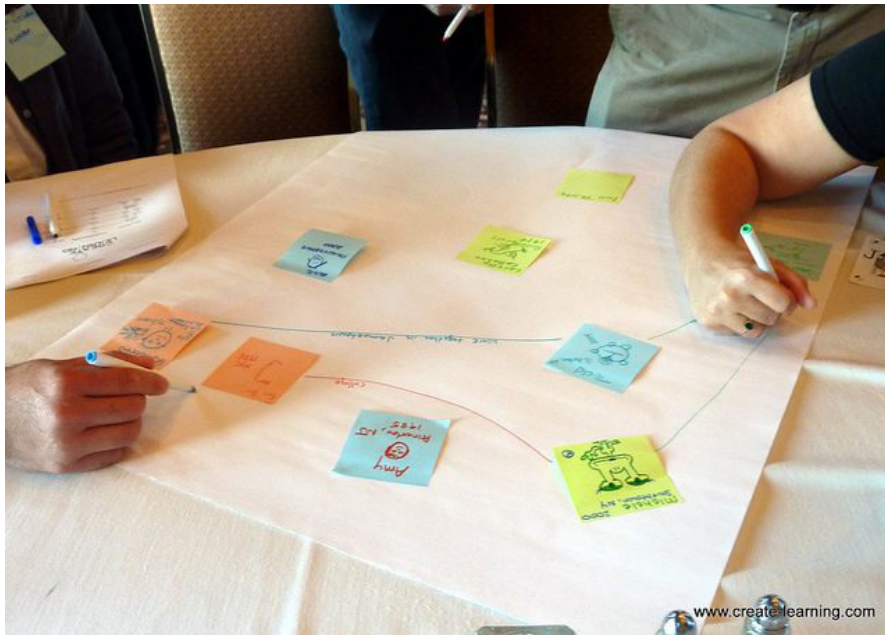
The more differentiation in the viewpoints and expertise areas of those analyzing the data, the less bias the interpretation will have. [As Alla Kholmatova explains](#), one study of 44 usability practitioners showed how the presence of a second evaluator increased problem detection by 30-43%.

4. “Cheese Day”

Collaboration doesn’t end once the usability test results are compiled and analyzed. [In an article for UXmatters](#), Roy Man explains the practice of “Cheese Day,” inspired by gaming leaders Blizzard.

“Cheese,” by his definition, is any UX annoyance that’s not quite a bug. Bugs are dealt with swiftly and without debate, whereas

cheese is a little trickier and requires some thought. That's why he suggests holding a cheese day, where designers, developers, and everyone else on the product team gets together to take care of these outstanding issues.



Source: "Team Building & Leadership w LawNY...Rochester, NY. Canandaigua, NY (16)." [Michael Cardus](#). *Creative Commons*.

A month before the scheduled Cheese Day, create a open document in which anyone can record potential usability issues. On the actual cheese day, gather everyone and set to work correcting each item on the list. Since solutions aren't so black-and-white as they are with bugs, varying perspectives across the team will be most beneficial.

Remember to uphold the same feedback guidelines while discussing and critiquing potential solutions.

Takeaway

Collaboration extends beyond a mere division of labor.

Everyone doing their individual jobs may be the bare minimum, but the real benefits of collaboration appear when the team improves how they socialize together. In this sense, people skills may be just as helpful on the final product as technical skills.

Be respectful of others feelings when giving and receiving feedback, and work together when conducting usability research and beyond. Having the same goal isn't enough: collaborating to meet that goal will give you the best chance for success.

Building Mockups Developers Won't Hate

For the best visual precision, every design project eventually requires fully-composed mockups for each page and interface. Informed by technical constraints, these mockups must act as collaborative visual design documents for [regular discussions](#) with developers – not just something you hand off as though the person on the other end is a WYSIWIG factory.



Source: *“Peer Review.”* AJC. *Creative Commons.*

Graphics, textures, typography, and other little tidbits all must be transferred from a simple graphic into a living interface. But just because you understand [how to design great mockups](#) doesn't mean that you understand how to design them for developers. In this chapter, we'll provide tips and considerations for building mockups that transition as smoothly as possible to development.

After all, if you want the project to move forward, you must always ensure that your works of digital art are clearly understood by developers who think in terms of frameworks and systems.

Plan for Limitations

The first step to proper implementation is predicting UI limitations that may (and usually will) arise during development.

In the design world, it seems like any beautiful idea should be possible to create in code. HTML5/CSS3 specs have certainly come a long way, but some ideas are still very difficult to build with full support.

When collaborating with developers in the wireframe/prototype stage, always keep in mind the limitations of [HTML5](#), [CSS3](#), or whatever your language of choice. That way, you'll know what to design and [how to design ideas visually](#) so that developers can actually build the interface.



Photo credits: [10 Best Practices for Sketch](#)

Keep a list near your desk (or in Google Docs) with a collection of hurdles, obstacles, and interface ideas to avoid. While it may be possible to add these features later, you should focus initially on prioritizing interface elements that are most feasible. As you progress in the design, review this list on a weekly basis with developers so that you don't get lost in a high-fidelity design of something that's actually a technical nightmare.

Now that you have a good starting point for design feasibility, let's examine some other methods to keep in mind as you design.

1. Expand on Interactions

When it comes to exploring interactions, prototyping is the most efficient platform for discussing feasibility. Aside from actually coding your designs, prototypes are far more effective than simply describing or annotating flat wireframes and mockups.

Even rough low-fidelity prototypes (also known as interactive wireframes) can show developers the overall framework of content, and the dependencies between content as users click through

the experience. While the visual details won't be anywhere near polished, developers can still provide feedback on design infrastructure.

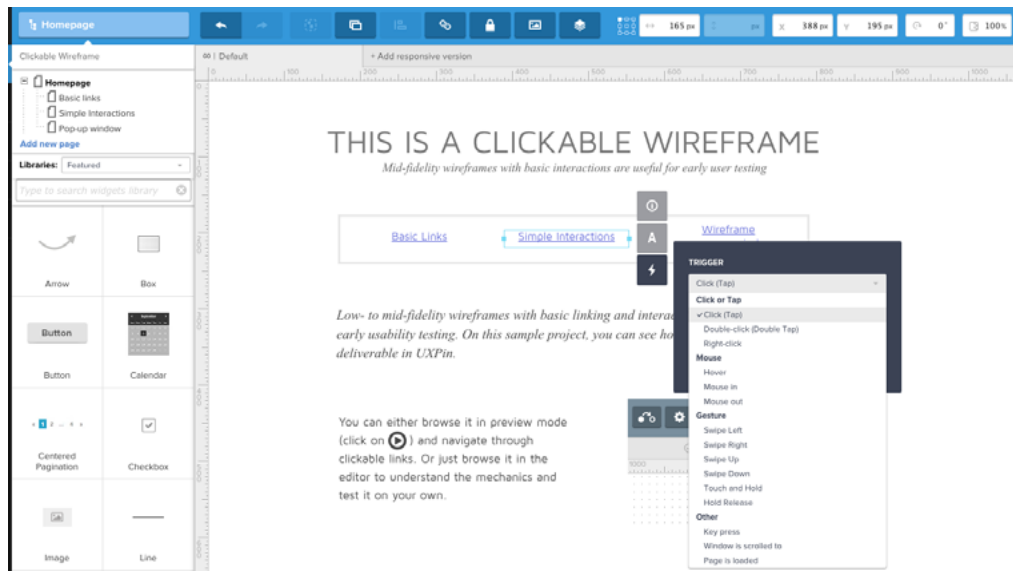


Photo credits: [Interactive Wireframe](#)

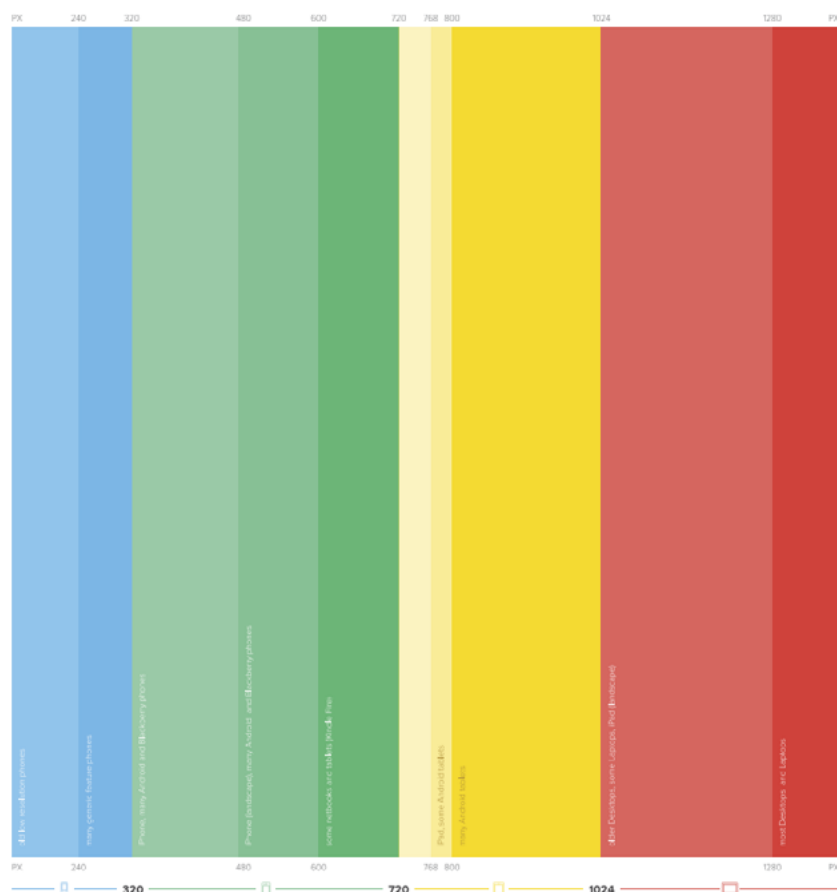
For example, you might be dead-set on an infinite scroll for a large blog design, but perhaps a cards UI pattern can still serve as an elegant compromise between usability and site load times. On a subtler level, developers can also provide you device-specific feedback. Your navigation header might work beautifully in the prototype, but it may be completely hidden by the address bar in certain browsers and devices.

You simply won't uncover these technical setbacks (some of which may require a complete revamp) unless you let developers play with your prototype. In fact, part of the reason we [integrate with Photoshop and Sketch](#) was to encourage more designers to cross over into the interactive design stage (even if they prefer to first add fidelity to the visual design).

2. Create Browser Fallbacks

The field of web design is quite different compared to mobile app design.

Android and iOS apps run on a single OS with various screen resolutions. Websites, however, introduce far more variables – they must run on different screen resolutions and on different operating systems, in different browsers, also with different screen resolutions (and devices).



RESPONSIVE WEB DESIGN CHEATSHEET

Photo credit: [Responsive Design Cheatsheet](#)

The process of designing a website interface feels similar to designing a mobile app interface. But development is vastly different between these two mediums. When designing a mockup, you'll want to consider the potential pitfalls and drawbacks of each feature.

This is especially true of newer CSS3 techniques that are not supported in all browsers(ex: transforms, reflections, [clip-path](#), and [masking](#)). While Photoshop and most other design programs let you insert notes next to your design, there is no substitute for a quick feasibility discussion with developers as you prepare for a major iteration.

Below, you'll find some of our favorite free open source scripts to help with browser compatibility. These resources may not help you directly when designing, but they are certainly an important conversation point when collaborating with developers.

- [Modernizr](#) – In the world of web standards Modernizr is a cherished asset for any project. It's a customizable library for cross-browser HTML5/CSS3 support. Use the [download page](#) to customize your own features or just grab the whole library [from GitHub](#).
- [Fallback.js](#) – This tiny JavaScript library is meant to control every possible fallback method and library. By centralizing all of your scripts it becomes easier to check which files aren't working and provide alternate solutions.

- [Selectivizr](#) – Selectivizr is similar to Modernizr but it focuses more on [CSS selectors](#). This JS library emulates CSS pseudo-classes and attribute selectors to improve support for older versions of Internet Explorer.
- [Video.js](#) – HTML5 video has taken the spotlight but Flash video is still a reliable fallback. Video.js makes it easy to embed any video format into an HTML5 player with a Flash fallback for older browsers.
- [IE7.js](#) – Older versions of Internet Explorer put up a tough fight against common web standards. IE7.js forces older browsers like IE6-IE8 to support typical rendering methods like transparent PNGs and RGBA() colors.
- [Detect Mobile Browsers](#) – This isn't so much a library as a code generator for mobile detection. You'll find snippets in all languages from ASP to jQuery for detecting a mobile browser. Developers can choose to execute(or not execute) code for mobile users based on these results.

Even if you don't understand how these scripts work, you should still bring them up with your developers. In our experience, we've found that developers appreciate the proactive gesture and don't mind spending some time explaining any compatibility issues. After all, it is in both of your best interests since late nights are usually a shared misery between designers and developers.

Dealing With Alternate Pages & Resources

While the homepage design may require the most effort since it's a portal to the entire site, you also need to apply the same precision to the inner pages.

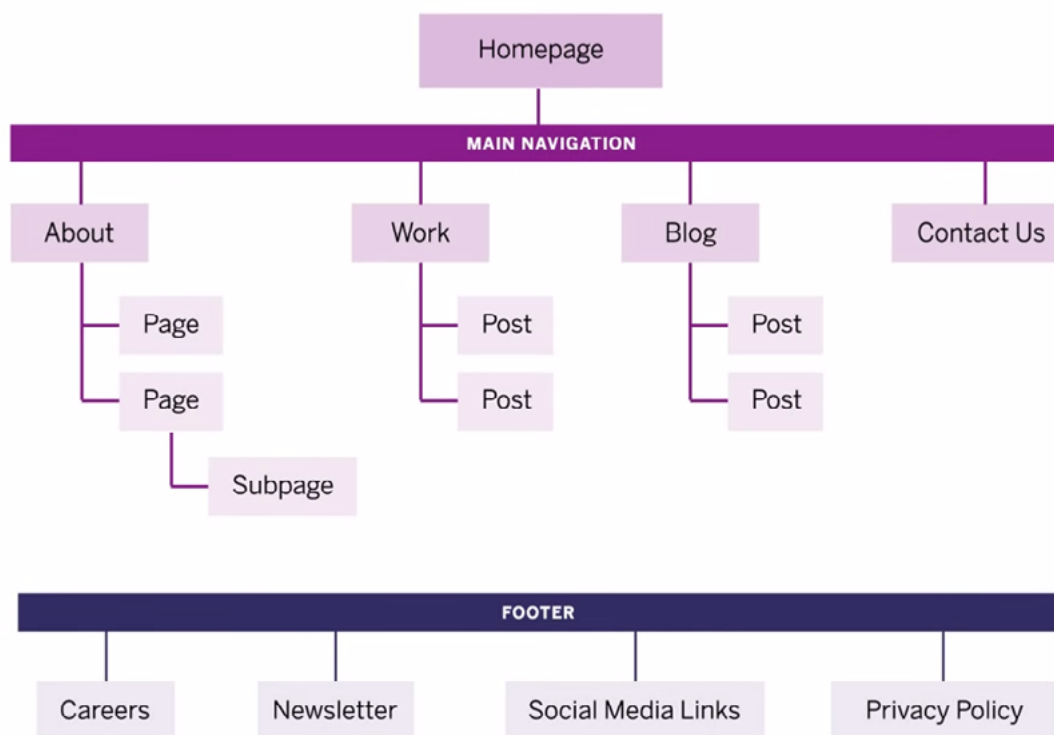


Photo credit: [Interactive Site Maps](#)

In some large companies, project managers demand full mockups for every single page due to strict internal processes. In other companies, some developers may have design experience, allowing them to build multiple similar pages from a single mockup (freeing up more time for you to work on unique pages). Adapt the level of mockup detail depending on your company protocol.

Now, let's examine some of the development considerations for your mockup elements.

1. Inner Page Graphics

A keen sense of graphic design is required for great mockup design.

For example, even if two pages are distinguished only by differences in icons, the safest option is to still create two separate mockups to avoid any confusion. Most of the time, you'll find it's safer to create all inner pages and then export icons separately – that way, developers can access the individual icon images and the page designs as reference material.

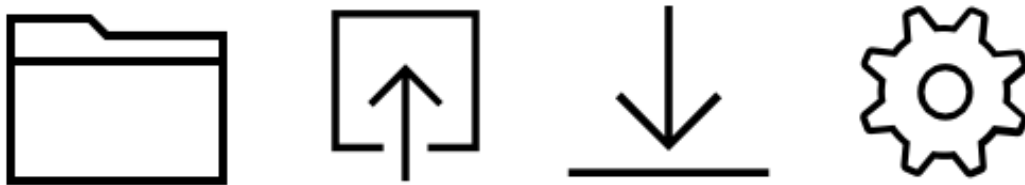


Photo credit: Icon Set in [UXPin](#)

Before wrapping up your mockup design, [follow along with this checklist](#) to see if you missed anything. Here's a quick list of some things to check for:

- Graphics should be sized exactly as they need to appear
- Illustrated different JavaScript interaction states (ex: dropdown menu open + closed states). Even if it's demonstrated in the prototype, it never hurts to leave a paper trail.
- Logged in & logged out states
- Label form fields, buttons, and inputs as needed
- Error/success messages

- Include separate graphic files for all images including the favicon, animated loaders, 404 page photos, etc.

On larger projects, we'd recommend including a short document for developer notes. Just be sure to clarify the referenced mockup (and page sections) which pertain to the notes. If you're working in [UXPin](#), you can add a note to the design itself and while you're in Preview mode, or you can also just upload a separate document into your project folder.

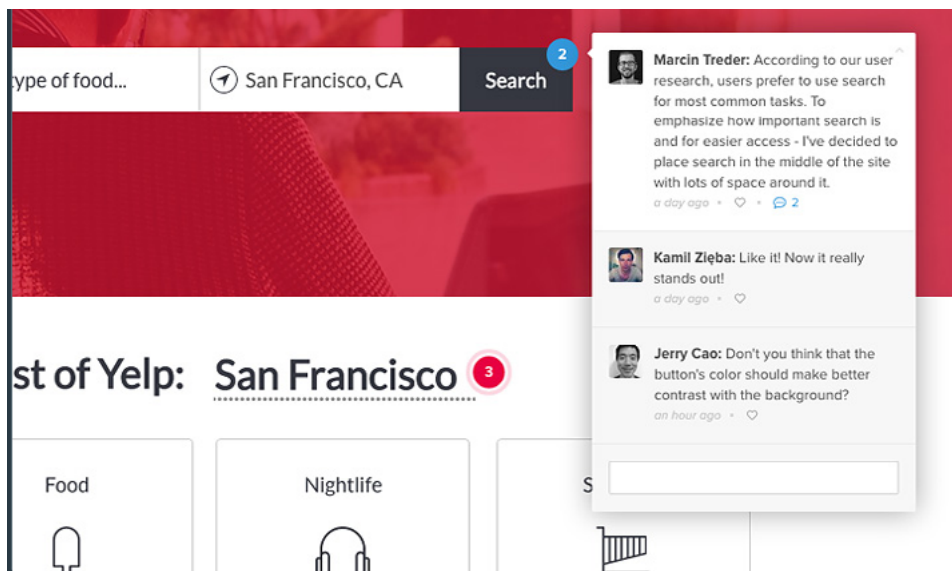


Photo credit: Collaboration via [UXPin](#)

2. 2. Separate Responsive Mockups

While mobile app mockups must consider landscape and portrait view, websites must support any view at any dimension from smartphones to widescreen monitors.

If your design is meant to be responsive, then we recommend creating different mockups [for each breakpoint](#) to show developers how the layout adapts.

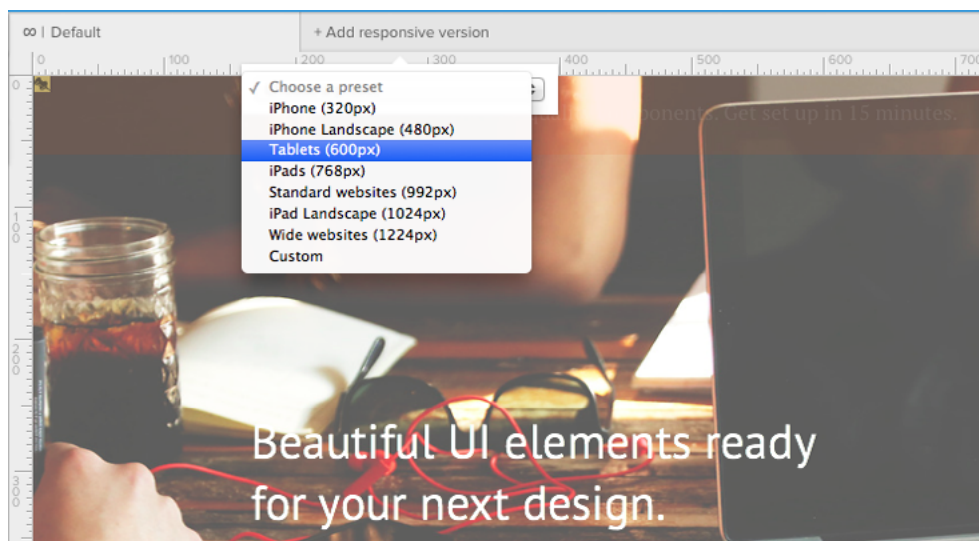


Photo credit: [Responsive Prototyping](#)

Consider a few of the following changes:

- Do logos, graphics, and/or icons change size or position?
- Should font size increase or decrease? What about line height?
- How does the main navigation adapt to smaller screens?
- Will sidebar columns drop below the main content or perhaps disappear entirely? What about footer content?

There are no right or wrong answers because each project needs to be handled individually.

When in doubt, the [mobile-first approach](#) championed by Luke Wroblewski is always a good starting point (once you've completed the initial user research). Design around the content for the smallest screen size, then scale it up as the screen size increases. By building upon the essentials first, you ensure a more enjoyable device-agnostic experience.



Photo credit: UXPin

To prepare for the transition to development, make sure you label your responsive mockups properly. Filenames must include the responsive breakpoint in pixels along with any other important details (revision date, draft number, retina screen, etc.)

Your goal is to design for every possible situation so the developer doesn't have to think.

As always, if you're unsure about how to deliver something, just ask one of the developers what format they prefer. Keep the communication lines open to reduce problems and avoid confusion.

Design Tips to Make Development Easier

Digital design programs come in many forms from Adobe Photoshop to Fireworks and the popular newcomer SketchApp. Despite the differences in capabilities, the overall mockup design process remains relatively uniform across these different applications.

The following tips are geared toward designers who create mockup assets for developers. Some of these ideas are easy to overlook on the surface, but they make a world of difference when it comes time to write code.

1. Mockup Design Best Practices

Mastering any design program takes time and switching up your workflow can be daunting. But once you try incorporating these tips, they'll feel like second nature. When you [design with developers in mind](#), it makes everyone's job a whole lot easier.



Photo credit: [Peter Morville](#). [Creative Commons](#).

Here's a few best practices we've learned through the years:

- **Organize and label your layers** – Developers may not always need to open your PSD/Sketch files, but the contents should always be organized. Use layer groups to rank similar layers together. Also, assign every layer a clearly recognizable name since mockups are very detailed and usually require a lot of digging to find exactly what must be changed or extracted.

- **Prepare a nice asset package** – Remember that asset preparation saves everyone a lot of time and stress. Once the mockups are finished, export graphics, icons, photos, and other assets into separate files. Developers may not feel comfortable exporting PNG/JPEG/SVG files and it's a lot easier if you hand off everything in one neat collection.
- **Use in-app export tools** – In Photoshop, you can export graphics using the [slice tool](#) or by manually creating new documents. Sketch includes its own [export options](#) designed specifically for interface graphics. Organizing your design files doesn't take much work and your developers will love you for it.
- **Show, don't tell** – If interactive elements require visualization, try converting your [PSD/Sketch files into layered prototypes](#). In doing so, you'll be able to actually show complex animations and interactions, leaving none of it to the risk of imagination.

For more practical tips, we recommend the [Photoshop Etiquette](#) site which includes advice for dozens of topics like effects, layer organization, and typography. To learn more about the anatomy and process of mockups, check out the free [Guide to Mockups](#).

2. Version Control for Designers

Every developer should at least know about [version control systems](#), which are like digital archives that store previous versions of a script, database, or an entire website. Aside from organizing files, version control is useful for rolling back changes or comparing differences between two(or more) files.

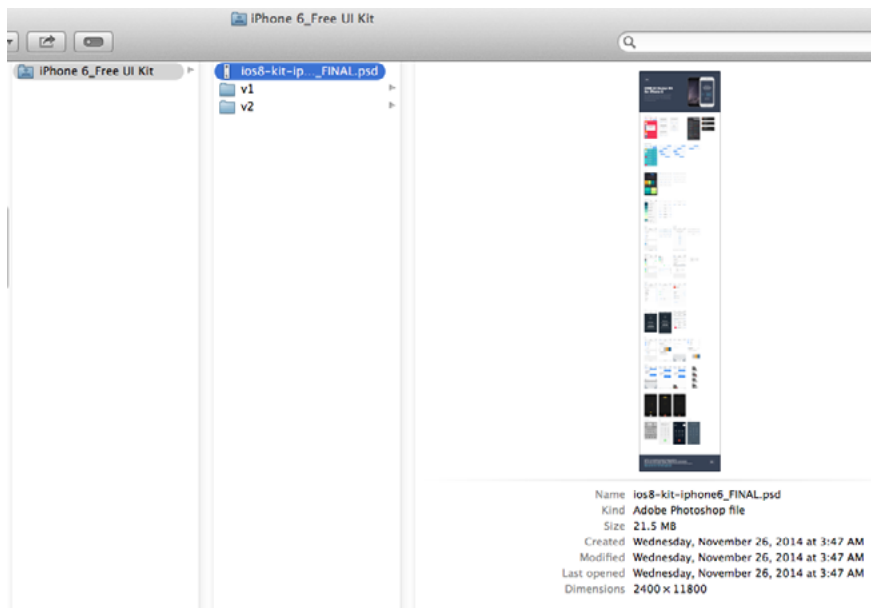


Photo credit: [Free iOS 8 UI Kit](#)

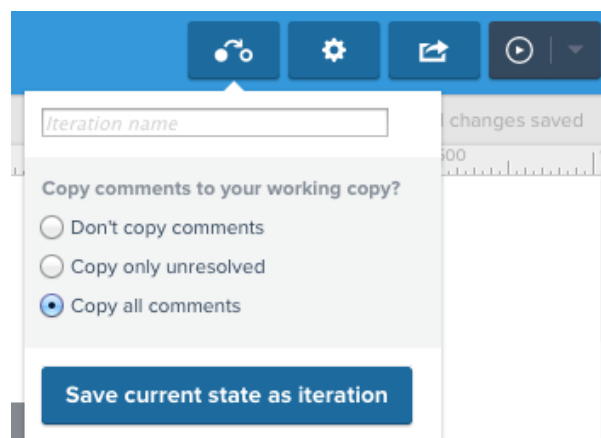
Although version control is mostly used in programming it has found small pockets in the design community. While design-based version control is still new, there are some great resources available. [Pixelapse](#) is an online storage application for managing design documents in the cloud. And while GitHub is primarily a code storage platform, they've recently [added support for PSD files](#).

Now it's possible for designers to use GitHub as their own version control system for PSDs. Granted, the site is open source and it may not be a great choice for private enterprise projects – but it is a good way to practice and learn about version control.

Here are some alternatives for design-centered version control:

- [Cornerstone](#) – The Cornerstone app is a [Subversion](#) client for Mac. This does require some initial setup but it can be great for localized work. Check out [this stack answer](#) for more info.

- [PixelNovel](#) – Adobe has released their own version control platform named PixelNovel. This also runs on Subversion but is much less technical to setup. It has a Photoshop plugin and even allows file sharing among multiple users.
- [GitHub for PSDs](#) – As mentioned earlier GitHub does support PSD diffing. While this can be a great method for learning Git, most GitHub repos are free and open to everyone. So this is great for practice but not for large enterprise projects.
- [Kaleidoscope](#) – This Mac OS X application doesn't offer traditional version control, but instead can be used for file comparison. Kaleidoscope is great if you don't need a full timeline archive and just need to compare the differences between files and/or folders.
- [Pixelapse](#) – The widely-known version control platform Pixelapse is great for any designer. They have free open source accounts and paid accounts that can support small freelancers and large teams. Pixelapse runs across all operating systems and even includes a project dashboard for team collaboration.



Source: [UXPin](#)

On a related note, we actually included 1-click versioning in [UXPin](#) since version control was always a pain point for us in the past. If you're working in UXPin, just click the below icon in the right-hand corner and a new version is automatically created.

Helpful Plugins

Photoshop and Sketch both support a wide assortment of plugins for automating tasks and improving typical design workflows. Even though Sketch was built from the ground up for web design, Photoshop has a larger selection of plugins simply because it's been around longer and focuses on a broader spectrum of tasks (photo editing, print work, UI design, etc).



Photo credit: Pixabay. CC0 Public Domain.

We chose the following plugins specifically for UI designers who create interfaces from scratch. These plugins will help you achieve pixel-perfect mockups with less time and effort than traditional workflows.

Photoshop Plugins

- [GuideGuide](#) – Setup guides and grids based on columns and pixel values.

- [Cut&Slice Me](#) – Cut and export graphics to different devices like smartphones, tablets, and computers.
- [PNG Hat](#) – A faster way to slice & export Photoshop mockups.
- [CSS Hat](#) – Export any Photoshop layer into CSS code.
- [Renamy](#) – Dynamically rename your Photoshop layers in seconds.

Sketch Plugins

- [Sketch Generator](#) – Export all of your design assets with a single keystroke.
- [Text Styles](#) – Export Sketch text styles from a mockup into CSS code.
- [Measure](#) – Obtain the exact dimensions and coordinates of any graphic in your mockup.
- [RenameIt](#) – The best way to rename layers in Sketch.

Wrap-Up

Mockup and hi-fidelity prototype creation are the last major tasks for the design team before developers take over. To keep your design on track during the transition, offer as many resources as possible and clarify everything. As we've recommended before, it's always better to over-clarify rather than under-clarify.



Photo credit: i a walsh. Creative Commons.

By designing mockups with developers in mind, you'll be giving yourself a constant reality check. It can definitely be frustrating when you move from your own routine into a more developer-friendly process. But when you're working on design projects as part of a larger team, it's better to compromise on your process than it is to unnecessarily compromise on your design due to breakdowns in communication.

Collaboration Secrets for Designers

When working with developers you really need to see things from their perspective. By learning the basics of how development works you'll gain a better foothold to expand your ideas into dev-related territory.

As a designer it's your job to not only create pixel-perfect interfaces but to clarify how they work. Strive to build sanitized mockups that anyone could understand. In fact, you should even maintain a style guide to act as documentation for each project.

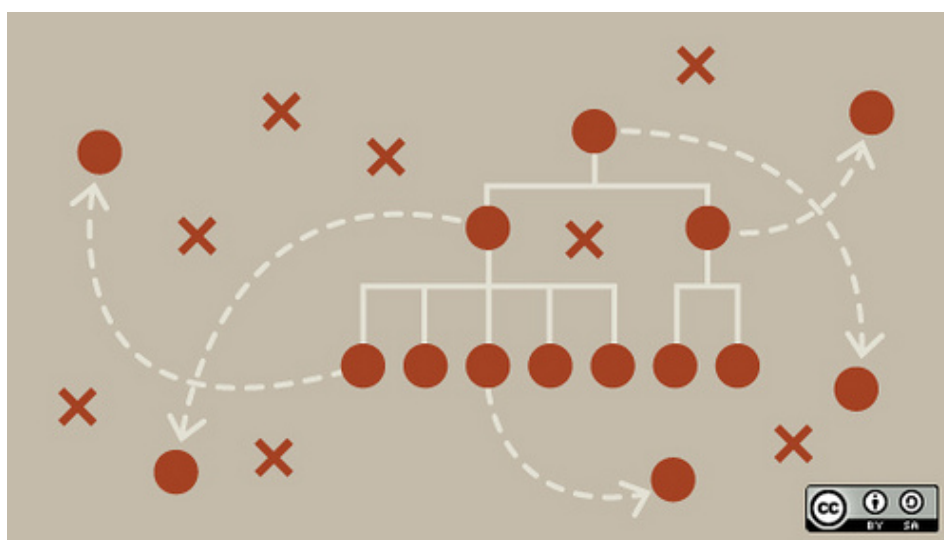


Photo credits: [OpenSourceWay](#). [Creative Commons](#).

This piece covers more "big picture" design tips along with some tools for improving designer & developer communication. All parties should actively work towards clarifying their intentions and seeing each problem from the others' point of view. The following tips & tricks should help you see design from the developer's side and foster a more free-flowing collaborative environment.

Follow Good Hygiene for Visual Design

As a designer, you're obviously expected to create high-quality flat and interactive designs, but [the delivery method](#) affects the quality of final implementation. When it comes to good design hygiene, make sure you're thinking about what developers need (not just your own organizational preferences).



Photo credit: [Brett Jordan](#). [Creative Commons](#).

Organize individual resources into folders so that developers know where to find items like icons and font files. Beyond just web-based resources, you should also consider adding HEX codes for colors and possibly demo content with actual text (no Lorem Ipsum).

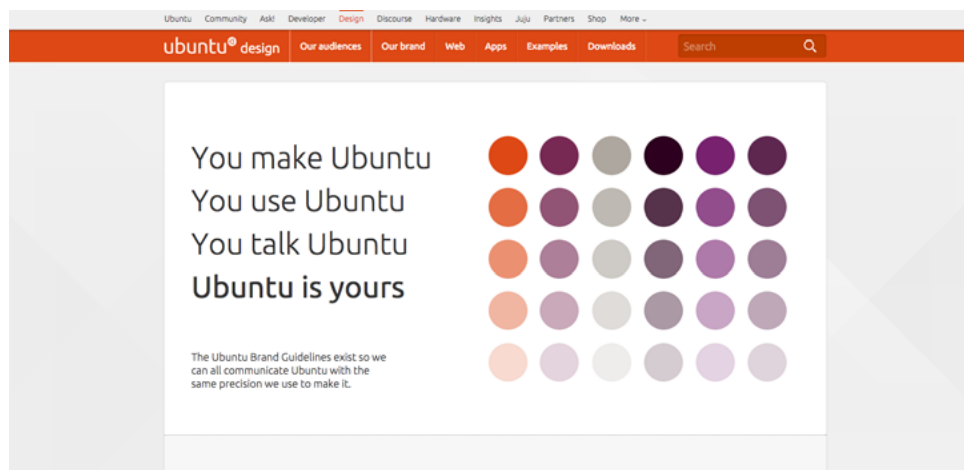
Here's a small list to get you started:

- Include hover states and describe animations as clearly as possible (either verbally in a quick over-the-shoulder meeting, or as notes affixed to early wireframes).
- Create diagrams and/or [storyboards](#) to convey where each link should lead.
- Label each font family and include 3rd party files for any webfonts. Check that all paid fonts also have proper licenses for web use.
- Graphics, tiles, icons, and photos should be appropriately named and organized. It's better to format these files yourself rather than expect developers to do resource management.
- Include HEX codes for all colors and gradients.

Treat Style Guides as a Collaborative Tool

When it comes to implementing these details consistently in your designs, you'll find that style guides are an excellent tool for designer and developer collaboration.

Designers can see how to create reusable UI patterns (down to the [atomic components](#) like button treatments and labels), while developers have a quick visual checklist and can even speed up their implementation if the style guide includes reusable code snippets.



Source: [Ubuntu](#)

Style guides are a solution to design pandemonium. These guides are created as either HTML pages or PDFs which contain information related to a company's brand, color scheme, graphics, fonts, and other similar features.

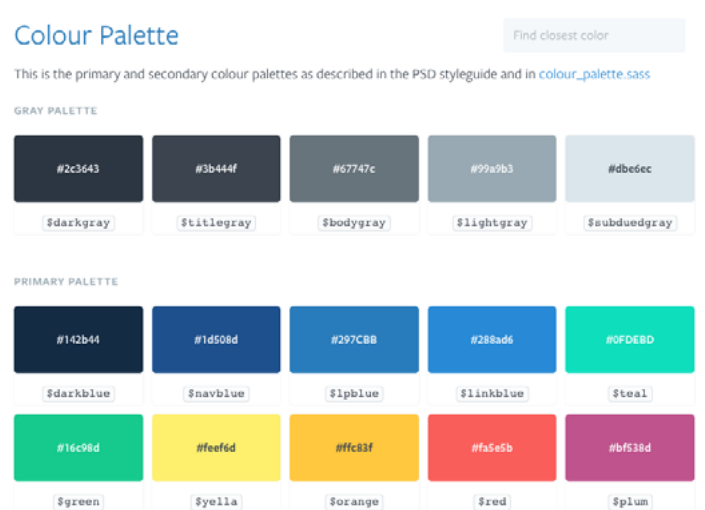
Here's some of the benefits of creating a style guide:

- **Consistency** – Style guides organize a cohesive design that reflects a common visual language. Visual and interaction design elements like color palettes, font families, and animations contribute to a unified user experience.
- **Context** – Specific ideas can be expanded upon at length to codify design choices. Great style guides won't just show a tab-based page and scroll-based page but will also explain the proper context for both. As an example, Jawbone uses a [scroll-based page](#) for storytelling and a [tab-based page](#) for product customization.
- **Collaboration** – A reference manual for each member of the team ensures that everyone is on the same page. Collaboration becomes

easier with fewer repeat questions. Style guides also streamline communication by creating a project vocabulary (i.e., defining what a “widget” or a “module” should be).

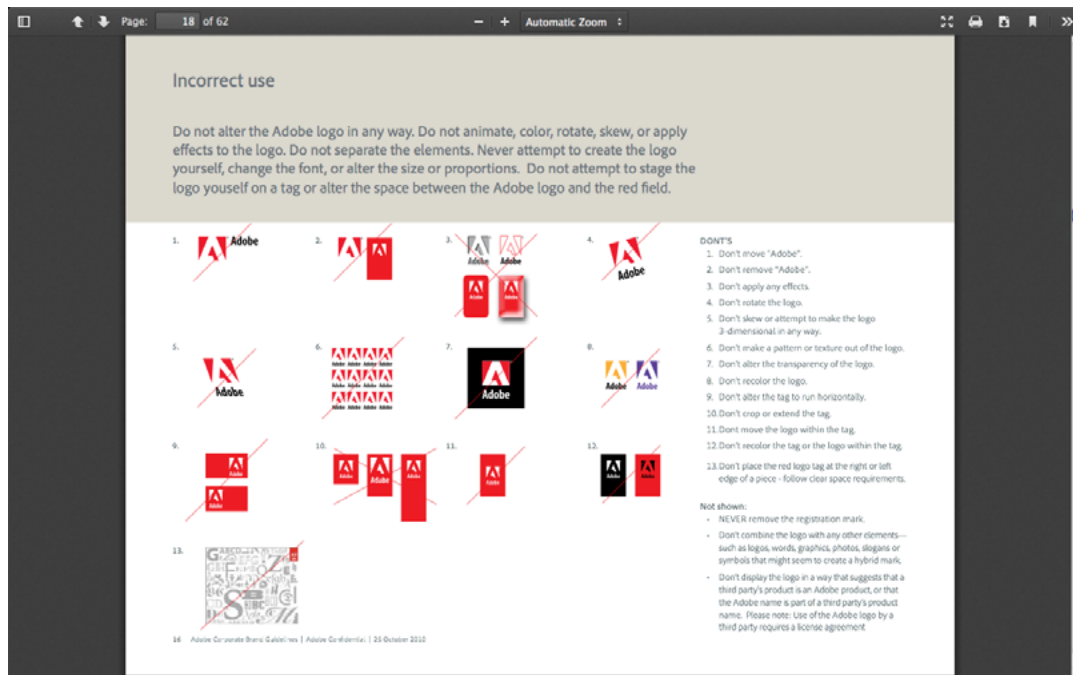
- **Code standardization** – Front-end style guides help to standardize HTML, CSS, and JavaScript so that designers and developers can test if a new idea deviates from established standards. These guides also provide vocabulary and mini-templates for adding similar interfaces onto different pages.

When debating a [PDF vs HTML style guide](#), you must consider the project’s medium. All web/mobile UI projects typically choose the [HTML route](#) because it’s much easier to update and they feel more natural to developers. In fact, you can take your HTML front-end style guide a step further and make it a [live style guide](#) that actively updates your site’s code as you modify the style guide. For developers and designers, live style guides are therefore less of a separate document that requires maintenance and more of a connective step for design iteration.



Source: [Lonely Planet](#)

On the other hand, PDF style guides aren't as useful in a web and mobile context – you usually see them used for projects like magazines, advertising, video production and/or animation studios. PDF guides also require digital resources but they're typically stored in .zip files or burned to CD-ROMs/ISO files.



Source: [Adobe](#)

While PDF guides can be hard to come by, there are plenty of HTML style guides to be found online. Take a peek at these front-end style guide examples to see how they're designed and how content is organized on each page.

- **WooThemes** – The beauty of WooThemes' style guide is that everything feels connected and relatable since its all on a single page.
- **MailChimp** – This style guide's interface is palatable to both designers and developers. The live examples and code snippets accentuate the free-flowing nature of this guide's content hierarchy.

- **Disqus** – Information is condensed together and visible from a single page with dynamic hover effects.
- **Primer CSS** – All content is heavily structured and the vertical menu makes navigation quick and effectual.
- **Lonely Planet** – A technically elegant version of a live style guide that updates the site via API call.
- **Mapbox** – The Mapbox style guide features everything from CSS styles to animation and even instructions for writing page copy.

If you're looking for more examples, take a look at [StyleGuides.io](https://styleguides.io) which has a large gallery of web-based guides (both traditional HTML style guides and more cutting edge live style guides).


Share Ideas with Style Guides

Style guides are equally helpful for designers and non-designers.

Developers can take full advantage of these guides because they're like documentation for design resources. The [best style guides](#) cover only what's necessary by providing guidance for anyone from development to marketing and even management.


How we design

There's a method to our madness: we have techniques and tools to make sure we're asking the right questions, and come up with great design solutions that our users really need.



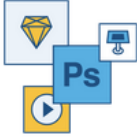
Design approach

For every step in your design journey, there are workshop techniques that help you make the right decisions, and make you understand better what you need to do.



Personas


Our products put our users into the center of the design process. We've established core personas that help you make the most suitable design decisions for them.



Resources


Whether you're just dabbling in screen design for a presentation, or you're a hard-core visual artist demanding pixel-perfect design, we're providing component libraries for your most popular tools.

Our guiding design principles




Our users should kick ass

We build products for our users, and we value their choices. We aim to use consistent patterns so that our users don't need to re-learn them every time. And we always play to the strength and flexibility of our products, combined with sensible defaults.



Just enough is more

Atlassian products are about getting your work done as effectively as possible. Inspired by Bauhaus and its focus on functional design, we aim to provide just enough design to create an awesome experience. When in doubt, leave it out.



Developers should kick ass

Our aim is to rationalize our design decisions as much as possible so that others can quickly build on them. The design guidelines are built with the Atlassian UI framework (AUJ). The AUJ flat pack is great for rapid prototyping.

Source: [Atlassian Design](#)

As months and years pass by, employees switch positions or even leave a company. Teams tasked with updating an older project won't always be able to just lean over and ask the original designer about

colors or typographic choices. When you dedicate time to creating and updating a style guide, you drastically reduce the amount of repetitive questions and discussions down the line (which naturally improves collaboration).

When creating a style guide, consider the following topics:

- **Logo Styles** – Proper size, colors, placement, white space, and positioning of the logo. Include alternate logo designs and icons/brand mascots if needed.
- **Color Schemes** – List all color choices and how they relate to the design. Include codes to reference these colors in either RGB, CMYK, or Pantone®.
- **Navigation** – Include all rules related to navigation, such as the use of search boxes, sidebars, links (if not covered in the typography section), and menus and drop-downs (i.e., what is the appropriate range of items to include).
- **Fonts** – List which typefaces fit into the design and in which locations. Try to include a character chart for all letters A-Z, a-z, 0-9, and symbols. Fonts that require licensing should include links to online font foundries.
- **Graphics** – Include any graphics related to the brand such as icons, buttons, symbols, favicons, tileable backgrounds, or vector artwork. If these items are listed in the style guide be sure to reference where actual PSD/AI files are located in the cloud or server database. Also, make sure to describe the details of execution: for

example, the alignment and position of images, how to overlay text on images, etc.

- **Layouts** – Print and web style guides often include sample templates for grids, margins, and different compositions.
- **Reusable Code** – Code doesn't have to be an individual section (you can attach the snippets in the relevant sections). Style guides should include both CSS classes and IDs in the snippet samples. Ideally, the developers should be able to copy and paste what they need.
- **Identity & Personality (extras)** – Include details regarding [voice & tone of the copy](#), the overall [design persona](#), and anything else that describes the overall emotional goal behind the design.

The larger your product team, the more you'll want a [detailed style guide](#) as a rallying point for designers and developers (which really comes in handy when they might not even work on the same floor anymore). In some cases, you might prefer to build separate guides relating to individual areas of design(ex: copywriting, print, web, etc).

Take an Interest in Development

If you design interfaces, then you're in the business of creating usable designs. Even if you never expect to learn development, it's still a good idea to take an interest in the process. While you may not be wrist-deep in code, you're still working in a field that requires coding.

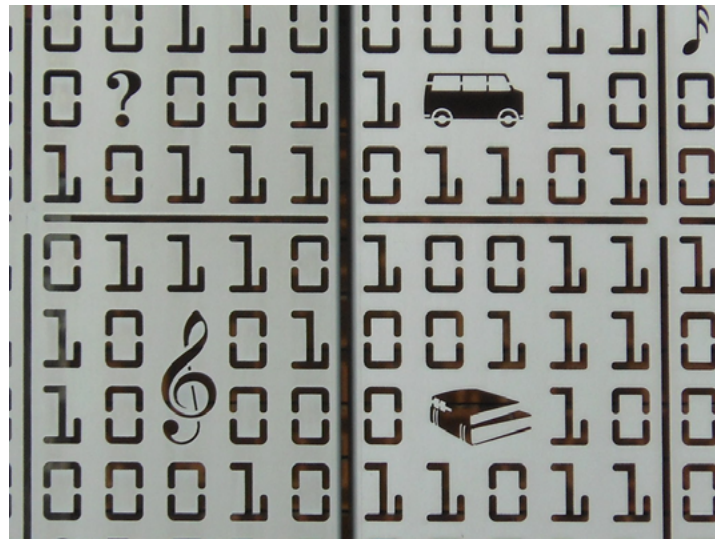


Photo credit: [Michael Coghlan](#). [Creative Commons](#).

A comparable analogy for learning about development could be learning about computer hardware. Although you may never install a new graphics card or upgrade your RAM, it's good to understand how these things work and how they affect your workflow. So while you may never fully understand how to write CSS or JavaScript, you should understand how they affect your workflow.

Another analogy comes from [this UX Stack question](#) discussing the value of a UI/UX designer learning JavaScript:

An architect that only knows how to use the pen can still be a good architect, but one that also knows how to pick up a hammer tends to have better insight into the types of building solutions one can come up with.

A better understanding of development naturally improves collaboration. For example once you understand the [CSS box model](#), you'll be forced to see mockups as if they're built with CSS box containers.

Then you'll be able to craft more specific design ideas and convey them in a native tone that developers understand.

Let's take a look at a few ways to understand development as a designer and how to communicate developer-friendly ideas.

1. Understanding Development as a Designer

There's a careful balance to be found between learning how code works and learning to code. As a designer, you should mostly be interested in the former topic by studying the core fundamentals of development.

When designing mockups you should be able to "see" [how they can be built](#) from a developer's perspective. For example, an input field is created using an HTML tag and styled with CSS. You don't need to know how to write this code but you **should** know the basics of how form inputs are generated.

If you're stumped on a particular subject don't be afraid to reach out to developers for support. Consider studying a few of these ideas to help you speak in a way that developers understand:

- How HTML tags are nested and rendered in the browser
- The [basics of CSS](#) and how it differs from HTML
- How CSS properties are used to create effects like box shadows
- Which elements can be created using pure CSS vs. which elements require images

- How [breakpoints work](#) in responsive website layouts

If you love UX design, then it's a good idea to expand into JavaScript as well. You don't need to understand how to write variables or functions, but you should know a little about how JavaScript interacts with HTML elements on the page.

2. Communicating Dev-Friendly Ideas

Talking about ideas can be one of the most confusing aspects of the creative process – but it doesn't need to be that way.

Developer-friendly ideas must be described in a practical manner. Brad Frost's method of [atomic web design](#) (composed of 5 tiers from atoms to complete pages) breaks down each interface into logical building blocks. This way, you can explain how users should interact with each individual block, and how these interactions affect others elements on the page.

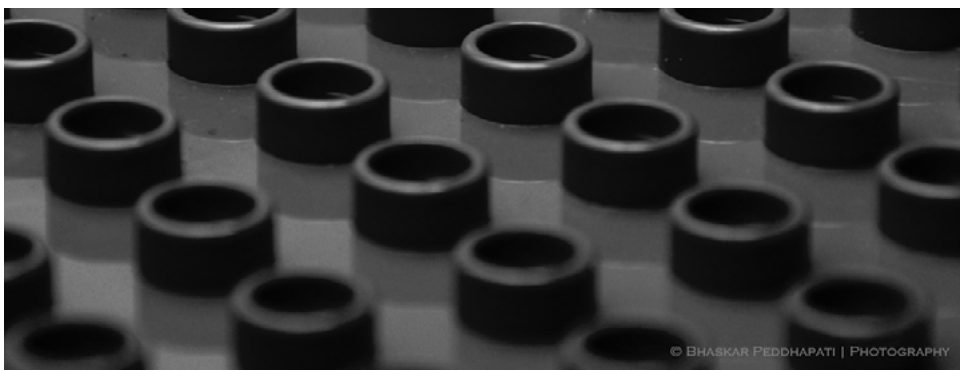


Photo credit: [Bhaskar Peddhapati](#), [Creative Commons](#).

Most developers are able to look at a live interface and understand how it works. We always recommend explaining your idea(s) with

a live example to help developers better visualize how the final interface looks and behaves. Since most developers know how to rebuild any interface from scratch, live examples improve comprehension and retention rate for future discussions.

To learn more, we recommend [this piece](#) by Smashing Magazine which elaborates on 6 practical tips for communicating with developers.

List of Collaboration Tools

Team collaboration is a complex process requiring communication, brainstorming, task management, and design presentation. While it's possible to coordinate these efforts separately it can be much easier to rely on tools specifically crafted for these individual tasks.

We've compiled some of our favorite tools below.

- [Google Drawings](#) – Visual notes and ideas can be organized simultaneously with Google Drawings. This plugin runs in Chrome but can also work offline without Internet access. This can also tie into Google Drive for sharing ideas with other team members.
- [Cage App](#) – This in-depth dashboard app provides everything for smooth project management. Teams can share files and comments during every stage of the process while staying up-to-date with the latest changes.

- [Slack](#) – Large teams need large communication tools. Slack is one such tool built around 21st century communication among creative teams. All chat logs are saved and searchable from the web dashboard and proprietary mobile apps.
- [UXPin](#) – Of course, we’re a little biased in our assessment here, but we’ve really tried to design UXPin to make collaborative design available to everyone. Our app allows for low-fi to high-fi design in one tool, turns wireframes into prototypes easily, [integrates with Photoshop and Sketch](#), supports in-app usability testing, and allows for easy commenting and design presentations.
- [Trello](#) – Many teams and individuals consider Trello the best free task management tool. Set schedules, share responsibilities, leave notes, and manage project files all within the same dashboard – for free!
- [Asana](#) – Asana is a more detailed project management tool. Their platform integrates with all smartphones and includes support for multiple teams with multiple projects.
- [Conceptboard](#) – Ever needed an online scratch board for taking notes? Conceptboard provides interactive workspaces with draggable elements and multi-user commentary. It’s great for everything from early brainstorming to final presentations.

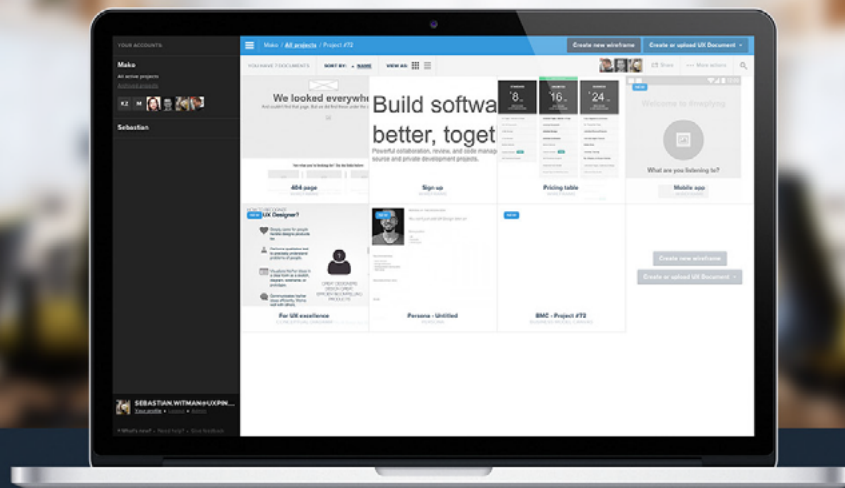
Wrap-up

When working on large enterprise projects everyone on the team should be involved in all stages throughout the entire process. The best way to achieve smooth collaboration is through the use of tools and relatable terminology.

Every designer should familiarize themselves with the development process. An underlying knowledge of how interfaces really function can improve the quality of design mockups. Similarly developers should become familiar with design language and style guides to communicate their ideas visually.

Open collaboration and communication is the best way to push through project work by maintaining a concrete level of focus. Hopefully these tips and resources will help you & your team members work together in unison with greater focus on each phase of a creative project.

[Design collaboratively in UXPin \(free trial\)](#)



- ✓ Complete prototyping framework for web, mobile, and wearables
- ✓ Collaboration and feedback for any team size
 - ✓ Lo-fi to hi-fi design in a single tool
- ✓ Integration with Photoshop and Sketch

UXPin

www.uxpin.com