46: Interface First

Design the interface before you start programming

Too many apps start with a program-first mentality. That's a bad idea. Programming is the heaviest component of building an app, meaning it's the most expensive and hardest to change. Instead, start by designing first.

Design is relatively light. A paper sketch is cheap and easy to change. html designs are still relatively simple to modify (or throw out). That's not true of programming. Designing first keeps you flexible. Programming first fences you in and sets you up for additional costs.

Another reason to design first is that the interface is your product. What people see is what you're selling. If you just slap an interface on at the end, the gaps will show.

We start with the interface so we can see how the app looks and feels from the beginning. It's constantly being revised throughout the process. Does it make sense? Is it easy to use? Does it solve the problem at hand? These are questions you can only truly answer when you're dealing with real screens. Designing first keeps you flexible and gets you to those answers sooner in the process rather than later.

The Orange Pen That Started Blinksale

> *As soon as I realized my frustration with off-the-shelf invoicing software, I decided to draw out how I would prefer my invoicing solution to work. I pulled out an orange pen, because it was the only thing handy that evening, and had about 75 percent of the ui drawn out within a few hours. I showed it to my wife, Rachel, who was ironing at the time, and asked, "What do you think?" And she replied with a smile, "You need to do this. For real."*
>
> *Over the next two weeks I refined the designs, and completely mockedup static html pages for almost the entire first version of what would become Blinksale. We never did any wireframes beyond those orangepen sketches, and getting straight into the html design helped us stay excited about how "real" the project was becoming, even though at the time we really didn't know what we were getting into.*
>
> *Once the html mockups were completed, we approached our developer, Scott, with the idea for Blinksale. Having most of the ui designed up front was extremely beneficial on several levels. First, it gave Scott a real vision and excitement for where we were going. It was much more than just an idea, it was real. Second, it helped us accurately gauge how much of Scott's effort and time it would require to turn the design into a functioning application. When you're financially*

> *bootstrapping a project, the earlier you can predict budget requirements, the better. The ui design became our benchmark for the initial project scope. Finally, the ui design served as a guide to remind us what the application was about as we progressed further into development. As we were tempted to add new features, we couldn't simply say, "Sure, let's add that!" We had to go back to the design and ask ourselves where that new feature would go, and if it didn't have a place, it wouldn't get added.*

—Josh Williams, founder, Blinksale

47: Epicenter Design

Start from the core of the page and build outward

Epicenter design focuses on the true essence of the page — the epicenter — and then builds outward. This means that, at the start, you ignore the extremities: the navigation/tabs, footer, colors, sidebar, logo, etc. Instead, you start at the epicenter and design the most important piece of content first.

Whatever the page absolutely can't live without is the epicenter. For example, if you're designing a page that displays a blog post, the blog post itself is the epicenter. Not the categories in the sidebar, not the header at the top, not the comment form at the bottom, but the actual blog post unit. Without the blog post unit, the page isn't a blog post.

Only when that unit is complete would you begin to think about the second most critical element on the page. Then after the second most critical element, you'd move on to the third, and so on. That's epicenter design.

Epicenter design eschews the traditional "let's build the frame then drop the content in" model. In that process, the page shape is built, then the nav is included, then the marketing "stuff" is inserted, and then, finally, the core functionality, the actual purpose of the page, is poured in to whatever space remains. It's a backwards process that takes what should be the top priority and saves it for the end.

Epicenter design flips that process and allows you to focus on what really matters on day one. Essentials first, extras second. The result is a more friendly, focused, usable screen for customers. Plus, it allows you to start the dialogue between designer and developer right away instead of waiting for all aspects of the page to fall in line first.

48: Three State Solution

Design for regular, blank, and error states

For each screen, you need to consider three possible states:

- **Regular**
  The screen people see when everything's working fine and your app is flush with data.
- **Blank**
  The screen people see when using the app for the first time, before data is entered.
- **Error**
  The screen people see when something goes wrong.

The regular state is a no-brainer. This is the screen where you'll spend most of your time. But don't forget to invest time on the other states too (see the following essays for more on this).

49: The Blank Slate

Set expectations with a thoughtful first-run experience

Ignoring the blank slate stage is one of the biggest mistakes you can make. The blank slate is your app's first impression and you never get a second...well, you know.

The problem is that when designing a ui, it's usually flush with data. Designers always fill templates with data. Every list, every post, every field, every nook and cranny has stuff in it. And that means the screen looks and works great.

However, the natural state of the app is one that's devoid of data. When someone signs up, they start with a blank slate. Much like a weblog, it's up to them to populate it — the overall look and feel doesn't take shape until people enter their data: posts, links, comments, hours, sidebar info, or whatever.

Unfortunately, the customer decides if an application is worthy at this blank slate stage — the stage when there's the least amount of information, design, and content on which to judge the overall usefulness of the application. When you fail to design an adequate blank slate, people don't know what they are missing because everything is missing.

Yet most designers and developers still take this stage for granted. They fail to spend a lot of time designing for the blank slate because when they develop/use the app, it's full of data that they've entered for testing purposes. They don't even encounter the blank slate. Sure, they may log-in as a new person a few times, but the majority of their time is spent swimming in an app that is full of data.

What should you include in a helpful blank slate?

- Use it as an opportunity to insert quick tutorials and help blurbs.
- Give a sample screenshot of the page populated with data so people know what to expect (and why they should stick around).
- Explain how to get started, what the screen will eventually look like, etc.
- Answer key questions that first-time viewers will ask: What is this page? What do I do now? How will this screen look once it's full?
- Set expectations and help reduce frustration, intimidation, and overall confusion.

First impressions are crucial. If you fail to design a thoughtful blank slate, you'll create a negative (and false) impression of your application or service.

You Never Get A Second Chance...

> *Another aspect of the Mac OS x ui that I think has been tremendously influenced by [Steve] Jobs is the setup and first-run experience. I think Jobs is keenly aware of the importance of first impressions...I think Jobs looks at the first-run experience and thinks, it may only be one-thousandth of a user's overall experience with the machine, but it's the most important onethousandth, because it's the first one-thousandth, and it sets their expectations and initial impression.*

—John Gruber, author and web developer (from Interview with John Gruber)

50: Get Defensive

Design for when things go wrong

Let's admit it: Things will go wrong online. No matter how carefully you design your app, no matter how much testing you do, customers will still encounter problems. So how do you handle these inevitable breakdowns? With defensive design.

Defensive design is like defensive driving. The same way drivers must always be on the lookout for slick roads, reckless drivers, and other dangerous scenarios, site builders must constantly search for trouble spots that cause visitors confusion and frustration. Good site defense can make or break the customer experience.

We could fill a separate book with all the things we have to say about defensive design. In fact, we already have. "Defensive Design for the Web" is the title and it's a great resource for anyone who wants to learn how to improve error screens and other crisis points.

Remember: Your app may work great 90% of the time. But if you abandon customers in their time of need, they're unlikely to forget it.

51: Context Over Consistency

What makes sense here may not make sense there

Should actions be buttons or links? It depends on the action. Should a calendar view be in list-form or grid-form? It depends where it's being shown and how long the time period is. Does every global navigation link need to be on every page? Do you need a global search bar everywhere? Do you need the same exact footer on each page? The answer: "It depends."

That's why context is more important than consistency. It's ok to be inconsistent if your design makes more sense that way. Give people just what matters. Give them what they need when they need it and get rid of what they don't. It's better to be right than to be consistent.

Intelligent Inconsistency

> *Consistency is not necessary. For years, students of ui and ux have been taught that consistency in the interface is one of the cardinal rules of interface design. Perhaps that holds in software, but on the Web, it's just not true. What matters on the Web is whether, on each individual page, the user can quickly and easily advance the next step in the process.*
>
> *At Creative Good, we call it "intelligent inconsistency": making sure that each page in the process gives users exactly what they need at that point in the process. Adding superfluous nav elements, just because they're consistent with the rest of the site, is just silly.*

—Mark Hurst, founder of Creative Good and creator of Goovite.com (from The Page Paradigm)

52: Copywriting is Interface Design

Every letter matters

Copywriting is interface design. Great interfaces are written. If you think every pixel, every icon, every typeface matters, then you also need to believe every letter matters. When you're writing your interface, always put yourself in the shoes of the person who's reading your interface. What do they need to know? How you can explain it succinctly and clearly?

Do you label a button Submit or Save or Update or New or Create? That's copywriting. Do you write three sentences or five? Do you explain with general examples or with details? Do you label content New or Updated or Recently Updated or Modified? Is it There are new messages: 5 or There are 5 new messages or is it 5 or five or messages or posts? All of this matters.

You need to speak the same language as your audience too. Just because you're writing a web app doesn't mean you can get away with technical jargon. Think about your customers and think about what those buttons and words mean to them. Don't use acronyms or words that most people don't understand. Don't use internal lingo. Don't sound like an engineer talking to another engineer. Keep it short and sweet. Say what you need to and no more.

Good writing is good design. It's a rare exception where words don't accompany design. Icons with names, form fields with examples, buttons with labels, step by step instructions in a process, a clear explanation of your refund policy. These are all interface design.

53: One Interface

Incorporate admin functions into the public interface

Admin screens — the screens used to manage preferences, people, etc. — have a tendency to look like crap. That's because the majority of development time is spent on the public-facing interface instead.

To avoid crappy-admin-screen syndrome, don't build separate screens to deal with admin functions. Instead, build these functions (i.e. edit, add, delete) into the regular application interface.

If you have to maintain two separate interfaces (i.e. one for regular folks and one for admins), both will suffer. In effect, you wind up paying the same tax twice. You're forced to repeat yourself and that means you increase the odds of getting sloppy. The fewer screens you have to worry about, the better they'll turn out.

No Separate Interface

> *The application is everything. Anything that can be changed, added, or adjusted can be done directly through the management area of the application. This allows us to see exactly what our customers see to help them through any problems or questions that they have. And our customers don't have to worry about logging into a separate interface to do different tasks. One minute they might be dealing with appointments for their clients and the next minute they might have to add a new employee. They can't be bothered with jumping between different applications and maintaining a consistent interface they're able to adapt to the application even quicker.*

—Edward Knittel, Director of Sales and Marketing, KennelSource