Design Systems Handbook

By Marco Suarez, Jina Anne, Katie Sylor-Miller, Diana Mounter, and Roy Stanfield

Design Better

DesignBetter.com

Design Better

Contents

By Marco Suarez, Jina Anne, Katie Sylor-Miller, Diana Mounter, and Roy Stanfield

A design system unites product teams around a common visual language. It reduces design debt, accelerates the design process, and builds bridges between teams working in concert to bring products to life. Learn how you can create your design system and help your team improve product quality while reducing design debt.

Chapter—01

Introducing design systems

The power of scale

By Marco Suarez

In the 1960s, computer technology began outpacing the speed of software programming. Computers became faster and cheaper, but software development remained slow, difficult to maintain, and prone to errors. This gap, and the dilemma of what to do about it, became known as the "software crisis."

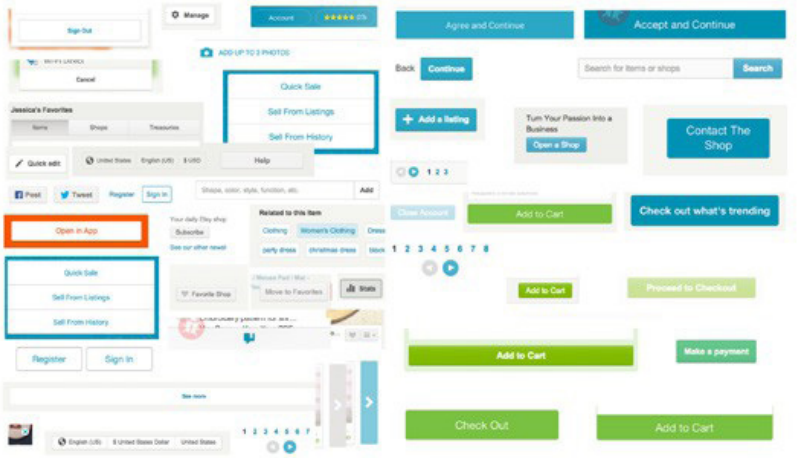


Figure 1: A UI audit collects the many permutations of simple UI elements to illustrate how deep in design debt your team is.

In 1968, at the NATO conference on software engineering, Douglas McIlroy presented component-based development as a possible solution to the dilemma. Component-based development provided a way to speed up programming's potential by making code reusable, thus making it more efficient and easier to scale. This lowered the effort and increased the speed of software development, allowing software to better utilize the power of modern computers.

Now, 50 years later, we're experiencing a similar challenge, but this time in design. Design is struggling to scale with the applications it supports because design is still bespoke— tailor-made solutions for individual problems.

Have you ever performed a UI audit and found you're using a few dozen similar hues of blue, or permutations of the same button? Multiply this by every piece of UI in your app, and you begin to realize how inconsistent, incomplete, and difficult to maintain your design has become.

For design in this state to keep up with the speed of development, companies could do 1 of 3 things:

01. Hire more people
02. Design faster
03. Create solutions that work for multiple problems

Even with more hands working faster, the reality is bespoke design simply doesn't scale. Bespoke design is slow, inconsistent, and increasingly difficult to maintain over time.

Design systems enable teams to build better products faster by making design reusable—reusability makes scale possible. This is the heart and primary value of design systems. A design system is a collection of reusable components, guided by clear standards, that can be assembled together to build any number of applications.

For more than 50 years, engineers have operationalized their work. Now it's time for design to realize its full potential and join them.

Scaling design with systems thinking

You're probably well aware that design systems have become a bit of a hot topic in the software industry these days—and for good reason. *Design is scaling*. Many businesses are investing in design as they recognize that the customer experience of their products offers a competitive advantage, attracts and retains customers, and reduces support costs.

Here are what things usually look like inside a company that's investing in design:

- The design team is growing
- Design is embedded in teams throughout the company, maybe in multiple locations
- Design is playing a key role in all products on all platforms

If you're a designer, this sort of investment in design may sound exciting, but with it comes many challenges. How will you design consistent UIs across platforms when many teams own various parts of your products? How will you empower all of these teams to iterate quickly? How will you maintain the inevitable design debt that will build up as many designers create new and tailor-made designs?

To understand how creating a design system can address these challenges, we must understand what design systems are. Design systems marry two concepts with individual merit, making something more powerful than its separate parts.

> It wasn't hard to get them to follow the guidelines, it was hard to get them to agree on the guidelines.
>
> *Lori Kaplan — ATLASSIAN*

Standards

Understanding not only the what, but the why, behind the design of a system is critical to creating an exceptional user experience. Defining and adhering to standards is how we create that understanding. Doing so removes the subjectivity and ambiguity that often creates friction and confusion within product teams.

Standards encompass both design and development. Standardizing things like naming conventions, accessibility requirements, and file structure will help teams work consistently and prevent errors.

Visual language is a core part of your design standards. Defining the purpose and style of color, shape, type, icons, space, and motion is essential to creating a brand aligned and consistent user experience. Every component in your system incorporates these elements, and they play an integral role in expressing the personality of your brand.

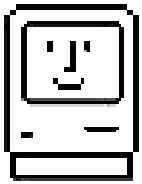
Without standards, decisions become arbitrary and difficult to critique. Not only does this not scale, it creates an inconsistent and frustrating user experience.

Components

Components are portions of reusable code within your system and they serve as the building blocks of your application's interface. Components range in complexity. Reducing components to a single function, like a button or a drop down increases flexibility, making them more reusable. More complex components, like tables for specific types of data, can serve their use cases well, but this complexity limits the number of applicable scenarios. The more reusable your components are, the less you need to maintain, and the easier scale becomes.

Learn more about building components in "Chapter 3: Building your design system."

> Having technical knowledge of the Macintosh user interface is a key factor in product design, but understanding the theories behind the user interface can help you create an excellent product.
>
> *Macintosh HIG — APPLE*



Component-based development reduces technical overhead by making code reusable. Standards govern the purpose, style, and usage of these components. Together, you equip your product team with a system that is easy to use, and you give them an understanding that clearly links the what with the why.

PRO TIP — Transcend platforms

Your visual language can transcend platforms to create continuity across web, iOS, Android, and email. Document and display your visual language in a prominent place within your design system's site. This will help inform system contributors about how components should look and behave.

For instance, Google's Material Design dives deep into every aspect of their visual language: Check out their page on color.

The value of design systems

Let's take a detailed look at the many ways a design system can be a much-needed painkiller for your growing pains.

Scale design

As teams grow, it's common for designers to concentrate on discrete areas of an app like search and discovery, account management, and more. This can lead to a fragmented visual language—like a Tower of Babel of design—with each designer speaking her own language. This happens when designers solve problems individually and not systematically.

With no common design language to unite the product, the user experience starts to break down, as does the design process. Design critiques become unproductive when there's a dearth of design conventions. To create alignment within teams, there must be a shared source of truth—a place to reference official patterns and styles.

Most often this is a static artifact, such as a design mock, but a static reference will almost immediately become outdated. That's why teams build monuments like Shopify's Polaris site—a design system site, built with the system, that documents all aspects of the system including the components, guidelines, and UX best practices. And because it is built with the system, it will always be up-to-date.

An internal design systems site is the best, most accessible source of truth for product teams. It provides the gravitational pull to keep team members aligned and in sync.

Jesse Bennett-Chamberlain (Shopify)

Listen Online
Circumstances For Investing In Polaris

Manage your debt

As applications and their teams age, they build debt. Not financial debt, but technical and design debt. Debt is acquired by building for the short-term. Design debt is made up of an overabundance of non-reusable and inconsistent styles and conventions, and the interest is the impossible task of maintaining them. Over time, the accumulation of this debt becomes a great weight that slows growth.

The act of creation does not inherently create debt—just like spending money doesn't inherently create financial debt. But using a design system will keep you on budget by keeping your design

and code overhead low, while still allowing you to grow and evolve your application.

Design consistently

Standardized components used consistently and repetitively create a more predictable and easy to understand application. Standardized components also allow designers to spend less time focused on style and more time developing a better user experience.

Prototype faster

Working within an existing design system allows you to piece together flows and interactions as quickly as pulling LEGO blocks from a bin. This allows you to build an endless amount of prototypes and variants for experimentation, helping your team gain insights and data fast.

Iterate more quickly

Whether evolving the style of your UI or making UX changes to a flow, using a design system reduces effort from hundreds of lines of code to as little as a few characters. This makes iterations quick and painless, and experimentation much faster.

Improve usability

Inconsistent interface conventions hinder usability. When CSS for countless unique interface elements and their interactions increase, so does cognitive load and page weight. This makes for a terrible user experience. It can also create conflicting CSS and JavaScript, potentially breaking your app. By using a design system, you're able to avoid these conflicts by building a holistic library of components, instead of per page, which means you'll spend less time in quality assurance.

Build in accessibility

Accessibility can be implemented at the component level by optimizing for those with disabilities, on slow Internet speeds, or on old computers. This is an easy usability win. In "Chapter 3, Building your design system," Katie Sylor-Miller explains how design systems can help improve your product's accessibility and compliance with your country's laws.

Myths of design systems

Even with all their benefits, buy-in for creating a design system can still be a hard sell internally. Designers can feel limited or restrained, but often these perceived weaknesses are the greatest strengths of a design system.

Let's debunk common myths you'll hear as you sell the idea of creating a design system.

Myth 1: Too limiting

Myth: Designers embedded in discrete areas of an app see qualities that may be different from other areas. Because of this, a universal system is perceived as being too limiting and might not serve the needs of these specific areas.

Reality: Designers often end up creating custom solutions to improve discrete areas of the app, adding to design and technical debt. With a design system, new solutions can be created and fed back into the system.

Myth 2: loss of creativity

Myth: If designers are restricted to using a design system, then designers will no longer be free to explore style. Front- end backlogs are often full of design style updates. Evolving the visual style of an app is typically no small task. This can also be a great risk, as it removes resources from new feature work and may negatively impact usability.

Reality: The components of a design system are interdependent. This means when a change is made in one location, the change will be inherited throughout the whole system. This makes style updates within a system trivial in effort but much greater in impact. What once was weeks— if not months—of work, can now be accomplished in an afternoon.

Myth 3: one and done

Myth: Once the design system is designed and built, the work is complete.

Reality: A design system is living, meaning it will require ongoing maintenance and improvements as needs arise. Because your application is powered by the reusable components of your system, however, the application automatically inherits improvements to the system, lowering the effort to maintain the application. This is the power of scaling that a design system offers.

> A lot of the original vision was about the visual identity...we started to understand that it had to be a system with really strong interaction design fundamentals as well.
>
> *Rich FULCHER — GOOGLE*

Conclusion

Design systems are not a fad or even an untested hypothesis. For design to find the scale necessary to match the rapid growth of technology, component-based design and development is a proven and dependable solution.

Now that you've seen the true value of creating a design system, let's dive into the actual design process in the next chapter.

Further reading

01. Software Crisis

02. Component-based Software Engineering

03. The Way We Build

04. Designed for Growth

05. Selling a Design System before asking for buy-in

06. The Design of Everyday Things

07. What is a Design Language... really?

08. Things you could be doing instead of designing and building that card component for the umpteenth time

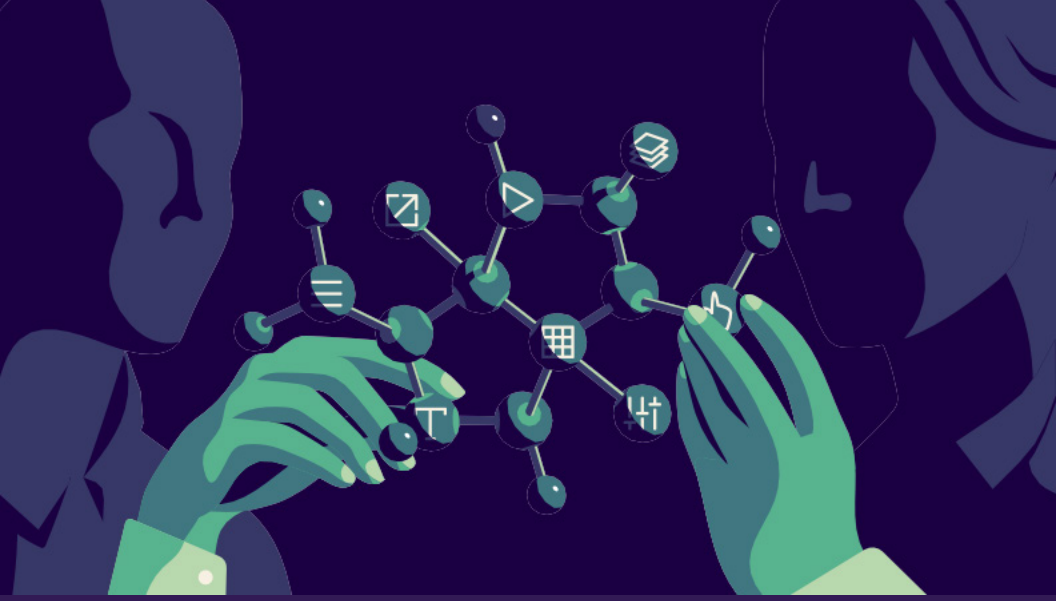
09. Website Style Guide Resources

10. Making Material Design

11. Material Design

12. Shopify Polaris

13. Starting a Design System



Chapter — 02

Designing your design system

Step by step

By Jina Anne

Starting a design system can feel daunting. There are so many things to consider: the design style, how to design for modularity and scalability, how it will be used by other teams, how to sell the idea to the decision makers in the company. Where is a designer to start?

Big problems are always more manageable when broken into smaller pieces. Before diving into the design process, start by considering who needs to be involved in the creation of your design system and how the team will work together.

Once you've got the right people assembled, you're ready to start thinking about the design language of the system, which will include color, typography, spacing, and more. Your visual design language will be the foundation of your UI library—a series of components that can be assembled to create an interface quickly.

Let's take a step-by-step look at how you can start designing your design system.

Who should be involved

Before beginning work on your design system, take a moment to think about the team you'll need to bring it to life. Who needs to be involved? Spoiler alert! You're going to need more than just designers.

> We have this program called the guild... essentially we take one person from each pillar...and for our design system we use them as our user research group.
>
> *Rachel cohen — LINKEDIN*

Here's a quick list of the disciplines that can be represented in your team to create an effective design system:

- Designers to define the visual elements of the system
- Front-end developers to create modular, efficient code

Accessibility experts to ensure your system conforms to standards like WCAG

- Content strategists who can help the team nail the voice and tone of the system
- Researchers who can help you understand customer needs
- Performance experts who can ensure your system loads quickly on all devices
- Product managers to ensure the system is aligning to customer needs
- Leaders (VPs and directors) to champion and align the vision throughout the company including up to executive leadership

Once you've got the right skillsets represented in the design systems team, identify strong leaders to represent each area who can drive decisions forward. Know who on the team can advocate for the areas of the design system.

With a team of experts guided by strong leadership, your next task is to establish the right team model to help you achieve your goals.

Choosing the right team model

The team model that brings people together is as important as the team creating your design system. In "Team Models for Scaling a Design System," design systems veteran Nathan Curtis outlines three popular team models used in many companies.

The solitary model: an "overlord" rules the design system.

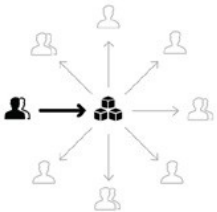


Figure 1: The solitary design system team model—one person rules it all. Image by Nathan Curtis, reused with permission.

The centralized team model: A single team maintains the design system as their full time job.

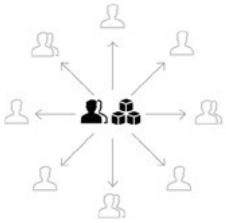


Figure 2: The centralized design system team model—one team rules the system. Image by Nathan Curtis, reused with permission.

The federated model: team members from across the company come together to work on the system.

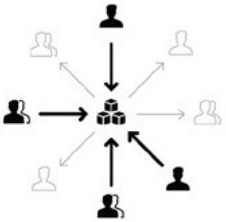


Figure 3: The federated design system team model—people from various teams assemble to manage and govern the system. Image by Nathan Curtis, reused with permission.

There are strengths and weaknesses in each of the above models. A solitary model is fast and scrappy, but with one person in charge of so much the "overlord" can become a bottleneck to the completion of many tasks. . A centralized team keeps the system well maintained, but they may not be as connected to the

customers' needs as they may be less involved in user research. And a federated team has great insight into what is needed for all the product features and user needs, but can be quite busy working on those areas already.

Many teams are moving away from the solitary model to the centralized or federated model because, as Nathan mentions in his article, overlords don't scale. The centralized or federated models are usually much better for scaling a design system.

I wrote about the The Salesforce Team Model in response to Nathan's article. When I was at Salesforce on the Lightning Design System team, we used a combination of the centralized and federated models. In an enterprise organization as big as Salesforce, a centralized design systems team was not enough on its own. With so many key players involved and the large amount of ground we had to cover across products and platforms, we needed an approach that would be more sustainable.

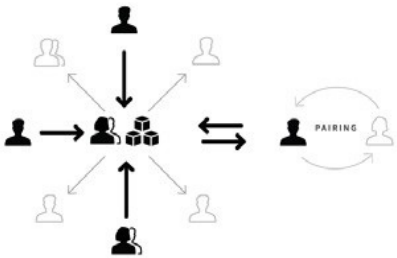


Figure 4: A hybrid design system team model that we used at Salesforce—a central team and members of other teams come together to manage and govern the system.

Though the Lightning Design System has a core team, there are also core contributors from many of the product and feature areas in the Salesforce ecosystem that act as a federation of practitioners who surface new ideas and make requests for the design system to evolve. Researchers, accessibility specialists, lead product designers, prototypers, and UX engineers work with the central design system team to both consume and help establish the patterns, components, and the overall design system. Engineers refine all code to make sure the system is performant and production ready.

Though the solitary model is less popular in most teams because the primary contributor can become a bottleneck, there are situations where it can work quite well. In the midst of a political campaign moving at breakneck speeds, Mina Markham had little time to bring in reinforcements as she developed new online assets for Hillary Clinton. She created a design system called Pantsuit to help many teams in many locations expedite design and production while maintaining consistency in the campaign brand. The solitary model let Mina focus on speed first and longevity second, which is a different tack than a typical enterprise might take.

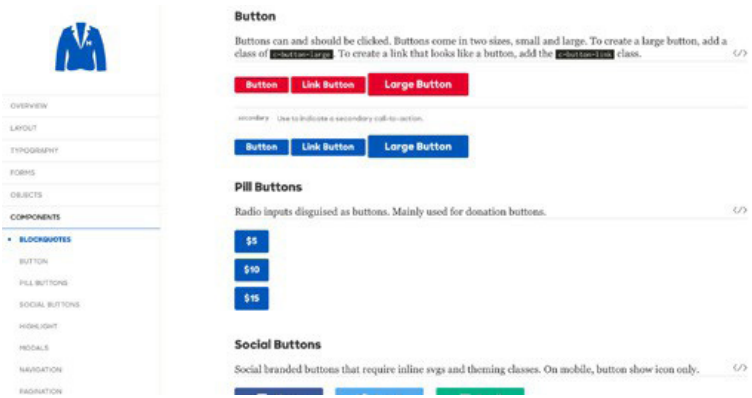


Figure 5: Pantsuit, a design system created by Mina Markham for the 2016 Hillary Clinton campaign.

As you determine what team model works for you, consider your goals. If you want to move fast, the solitary method is ideal initially, though some work may need to be done later to fully adopt it across other teams. If you want to move fast, but want to encourage buy-in from the start, consider the centralized team model. And to get the most buy-in and shared ownership, the federated model is a good option. In any case, remember that a design system is a product so staff it like a product instead of a project; you want people committed to maintaining and evolving it.

With the team and the model that organizes them established, it's time to start your design system just as you would any new product: by talking to your customers.

Interviewing customers

As with any product design process, it's important to do your research. Who will be using your design system and how will they use it? Your design system will get used much more often if you create it to fit into the workflow of other teams. By interviewing users, you can pinpoint problems ahead of time, define principles that will help others use the system properly, and focus your energies on the things that will be most important.

> **PRO TIP** — Building empowering style guides with practical research
>
> Isaak Hayes and Donna Chan delivered an interesting talk titled, "Building Empowering Style Guides with Practical Research (https://dbtr.co/empowering-style-guides)," at Clarity Conference (https://dbtr.co/clarity-conf). The talk proposes a series of useful techniques that can help you conduct research effectively for your design system. After the interviews, they use the data to create design principles, metrics, and user stories.

A less common group of people to interview are members of your open source community. This is likely in organizations that provide developer tools for their customer and partner communities. If you plan to open source your design system—a potentially bigger project—then you'll need to speak with potential contributors and consumers to discover what use cases your design system will need to satisfy.

And then there are the executives, leaders, and management. It is important to get their thoughts as well. You will need their buy-in to support and fund the system. Listen to their concerns and use them as actionable goals and metrics to achieve. Examples of requests might be faster shipping of features, better performance, and improved UI quality.

With insights in hand from customer interviews, it's time to take an inventory. There are two types of interface inventories to be created:

- An inventory of the visual attributes (such as spacing, color, and typography), which will help us create a codified visual language
- An inventory of each UI element (such as buttons, cards, and modals), which will help us create a UI library of components

Let's first focus on a global visual inventory. We'll get to the UI element inventory later.

Creating a visual inventory

Of course, if you're starting a design system for a product that doesn't yet exist you can skip this step and jump straight to creating a visual language for your new product. Lucky you!

Conducting a Visual Audit

As we start to take inventory, it's good practice to take a look at the CSS used to create all of those elements you just captured in your visual inventory. Use a tool like CSS Stats to see how many rules, selectors, declarations, and properties you have in your style sheets. More relevant, it will show you how many unique colors, font sizes, and font families you have. It also shows a bar chart for the number of spacing and sizing values. This is a great way to see where you can merge or remove values.

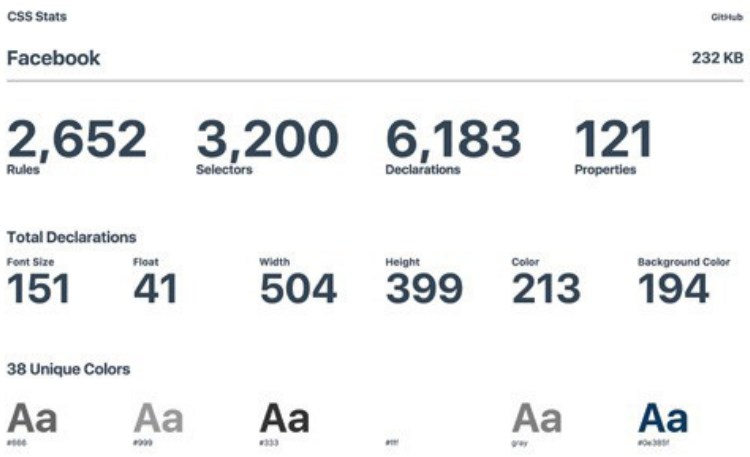


Figure 6: Facebook's 38 unique text colors found by CSS Stats.

If you're creating an inventory in Sketch use the Sketch-Style-Inventory plugin to aggregate all colors, text styles, and symbols quickly. It also gives you the ability to merge similar styles into one.

Creating a visual design language

I must admit, as an art school graduate the visual design language in a design system is my favorite part to work on. I love thinking about color theory, typography, and layout, which are at the core of any design system.

If we break apart each component of a design system we find these fundamental elements that make up its visual design language:

- Colors
- Typography (size, leading, typefaces, and so on)
- Spacing (margins, paddings, positioning coordinates, border spacing)
- Images (icons, illustrations)

Depending on your needs, you may also include the following to further standardize the user experience:

- Visual form (depth, elevation, shadows, rounded corners, texture)
- Motion
- Sound

Consider the role each of these design elements plays in a simple component like a button. A button typically has a background color, typography for the label, and spacing inside it. There may be an icon next to the label to create a visual cue. A border on the edge serves as simple ornamentation and may even round the corners. Finally, hovering over or clicking the button could trigger animation or sound as feedback to the user. Though a button may seem simple, there are many design decisions required to bring it to life.

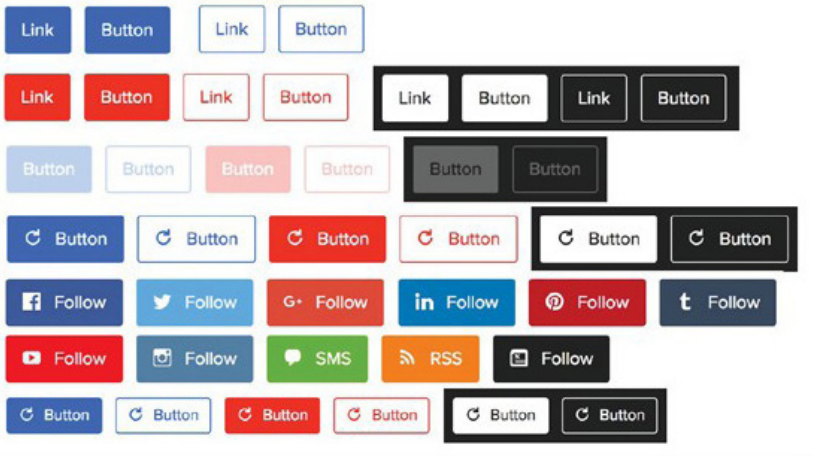


Figure 7: The many variations of buttons in the Buzzfeed Solid design system: buttons applied to both button elements and links, in the following modifiers: primary, secondary, transparent, negative, white, disabled, with icons, social, small, small with icons, small social, as well as a custom button that you can color to your needs.

Design tokens

Before we dive into visual design standards, I want to discuss design tokens. Design tokens are the "subatomic" foundation of a design system implementation. At its simplest, they are name and value pairs stored as data to abstract the design properties you want to manage. With the values for all design tokens stored in one place, it's easier to achieve consistency while reducing the burden of managing your design system.

Example: SPACING_MEDIUM: 1rem.

In design tokens you can store colors, spacing, sizing, animation durations, etc, and distribute them to various platforms.

Figure 8: Example of border radius design tokens on the Lightning Design System.

We'll look more closely at design tokens in Chapter 3.

Color

The colors you choose for your design system are more than just an extension of your brand. A UI uses color to convey:

- **Feedback:** Error and success states
- **Information:** Charts, graphs and wayfinding elements
- **Hierarchy:** Showing structured order through color and typography

Common colors in a design system include 1-3 primaries that represent your brand. If none of these work well as a link and button

color, then you may have an extra color for that as well. It's a good idea to use the same color for links and button backgrounds as it makes it easier for users to recognize interactive elements.

You'll likely have neutrals for general UI backgrounds and borders—usually greys. And finally, you'll have colors that are for states such as error, warning, and success. Group these colors to see how well they work together and refine as needed.

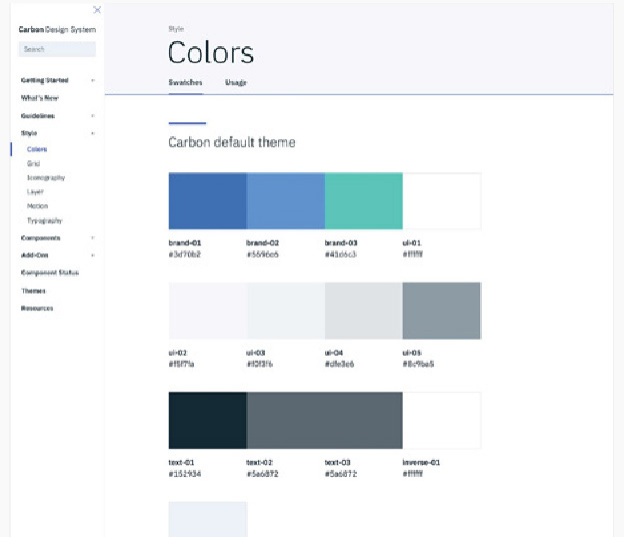


Figure 9: Color palette with design tokens in the Carbon design system.

PRO TIP

The in-progress book, Programming Design Systems by Rune Madsen, has some great chapters on color available online to read, including "A short history of color theory".

PRO TIP — Checking color contrast

There are a variety of color contrast checkers (https://dbtr.co/color-contrast) you can use to ensure your color palette works for everyone who will use your products. Be sure to check contrast ratios for background and text color pairings.

Larger design systems sometimes include colors for objects and products. For example, at Salesforce we had a color for contacts, for sales deals, or groups, and so on. We also had them for products: Sales Cloud, Marketing Cloud, Analytics Cloud, etc. Color can be a helpful wayfinding tool for your users.

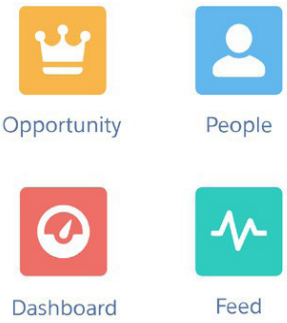


Figure 10: Colors used for objects in Salesforce.

Using color for wayfinding can be tricky to do while maintaining accessibility, as people who are color blind may not be able to discern some differences.

Depending on how strict you want to be with your palette, you may want to include a range of tints—a color mixed with white—and shades—a color mixed with black. Sometimes you may use other colors instead of white or black to avoid muddiness, such as an orange to darken a yellow so it doesn't appear brown.

These color variations allow designers to have choices. But be warned, having too many choices can lead to major design inconsistencies. Keep your inclusion of tints, shades, and neutral palettes slim to prevent misuse of the system while still giving designers the flexibility they need. You can always add more colors as you find the need.

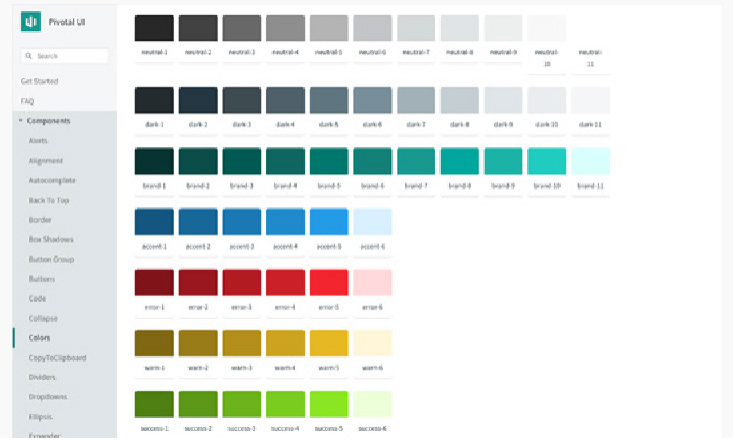


Figure 11: The Pivotal UI style guide chooses to give a wide range of color tints and shades with their design tokens. While I personally prefer to only give a leaner set of options (as seen in the Sass style guide), some design systems prefer to offer more choices. Consider which approach works for you as you balance concerns like creative freedom versus tighter consistency.

Typography

FONTS AND WEIGHTS

The fonts you choose have a high impact on both your brand and your user experience. Keep legibility in mind as you select the right fonts for your system. Keeping to common system fonts like Helvetica, Times New Roman, or Verdana can be a great shortcut, as they are familiar to the user's eye. Some companies prefer custom web fonts to better reflect their brand, but pay special attention to how you use them as performance can be affected.

Most design systems I've worked on include just two typefaces: one font for both headings and body copy, and a monospace

font for code. Sometimes there's an additional font for headings that compliments the body font. Most design systems do not have a need for more, unless you have a system that supports multiple brands. It's best to keep the number low as it's not only a best practice of typographic design, it also prevents performance issues caused by excessive use of web fonts.

> "Line spacing (leading) is at least space- and-a-half within paragraphs, and paragraph spacing is at least 1.5 times larger than the line spacing."
>
> ***WEB CONTENT ACCESSIBILITY GUIDELINES (WCAG) 2.0***
> *W3C*

Figure 12: Google's Roboto shown in varying weights.

These days it's trendy to use a font at a very thin weight, but be aware that legibility can become an issue. If you want to use light or thin weights, only use them at larger text sizes.

TYPE SCALE

When selecting the size to set your type, consider the legibility of the font you've chosen. In most cases, a 16px font size works well. It's the default font size in most browsers, and it's quite easy to read for most people. I like using 16px as it works with the 4-based metrics used by Apple and Google (and is gaining traction as the standard approach). I recommend this as your baseline, though I would use it in a relative format like 1rem for CSS-based systems.

You can use a modular scale for larger or smaller font sizes for other elements such as headings. A modular scale is a set of numbers in which you have 1 base number, and a ratio to generate

the next number. You keep applying the ratio to the new number to get yet another number.

> **PRO TIP** — Understanding modular scale
>
> Learn more about modular scales to create more meaningful typography.

Figure 13: The Modular Scale tool helps you find one that works for you. It even provides a Sass version of the tool, which you could add to your design token set.

As you design your type treatments, be sure to give thought to how it will respond to various screen sizes to maintain legibility. You won't want your headings to be too large for mobile devices. And for much larger displays, you have the room to bump up sizes.

A common method is to enlarge headings on larger viewports. You can also use viewport units to scale your type based on a percentage of your screen size.

LEADING

Leading, or line-height in CSS, can improve readability and aesthetics of your typography. While the best line-height can vary depending on the font face and the line length, a general rule of thumb is to have leading at around 1.4–1.5x the font-size. 1.5 is recommended by the W3C Web Accessibility Initiative.

It also makes your math more predictable, but you don't have to calculate it. You can define your line-height without a unit of measurement and the browser will do all the hard math for you.

For headings, tighten it up depending on your typeface. In most cases, I find a 1.25 or 1.125 ratio works quite well.

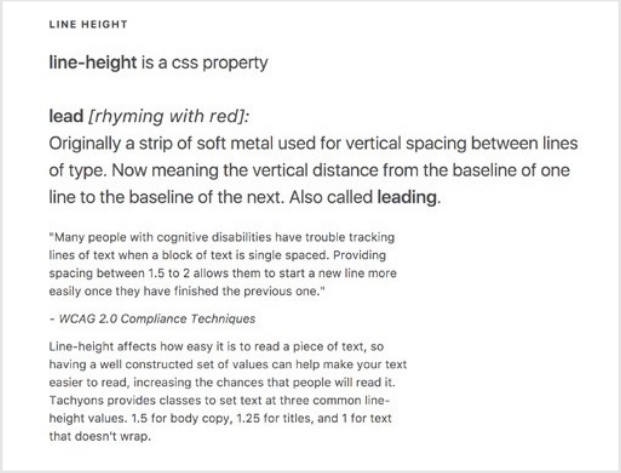


Figure 14: Tachyons also use 1.5 for body and 1.25 for headings.

SPACING AND SIZING

The system you use for spacing and sizing looks best when you have rhythm and balance. This means using numbers based on patterns and proportions. Using a consistent spacing scale also promotes maintainability through ratios by making layouts more predictable and more likely to "fit" and align well.

When I designed an Android app, I studied Google's design guidelines. I noticed a pattern of using 8dp between elements and 16dp for outer gutters. It broke me out of using a 10-based scale I was accustomed to, as I found that 4-based worked so much better.

A 4-based scale is growing in popularity as the recommended scale for many reasons. Both iOS and Android use and recommend metrics that are divisible by or multiples of 4. Standard ICO size formats, which are used by most operating systems, for icons tended be 4-based (16, 24, 32, etc.) so that they scaled more easily. The browser's default font size is usually 16. When everything is using this system, things are more likely to fit in place and line up. And finally, responsive math works out well.

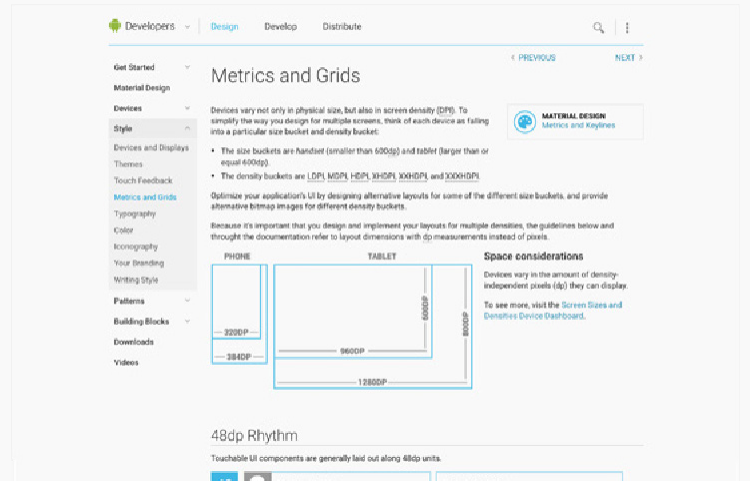


Figure 15: The Google Android design guidelines (before this site was replaced by material design). Studying these guidelines made me a better mobile designer.

For horizontal spacing, an 8-based scale works quite well. You can make margins and padding equal or in proportion to the font size. But for vertical spacing, I tend to use a 12-based system. This is due to the line-height I get of 1.5 (with the default font size of 16px) getting us to 24.

Occasionally, you may have to break this rule. If you've added a 1px border to something, this border can throw off alignment by a hair. So you might find yourself using a padding or margin that subtracts that amount. This is something that you do on a case-by-case basis.

You probably want elements to grow and shrink with the content. For general sizing, avoid setting widths and heights unless totally necessary. You can achieve responsive design much easier if you let elements flow to fill the space they're given in the layout.

Images

FILE FORMATS

For icons and illustrations, I find using a vector format (SVG) works best for scalability and responsive design. However, if you find yourself needing to use photography, you may need to use a rasterized image format like JPG or PNG.

For most photos, illustrations, and diagrams, you can allow the image to go 100% to the container or viewport and let the height automatically set itself by not defining it. This works best for responsive layouts. You may also want to define some preset widths for images if you don't want it to go full width (for example, half-width, a third, or a fourth). I recommend setting these as max-widths so that the image can rescale for smaller screens.

ICONOGRAPHY

Before drawing your icons, come up with your guidelines around them first. Will they be filled or outline? What is the line weight? Will they use more than 1 color? What sizes will they be? Is there an icon art boundary set inside an outer boundary?

You may have different styles for different icon types. For example, utility and action icons (like a notifications bell or a settings cog icon) may be solid and one color, while navigation icons may be multicolored and more creative. Clear guidelines will keep your icons unified.

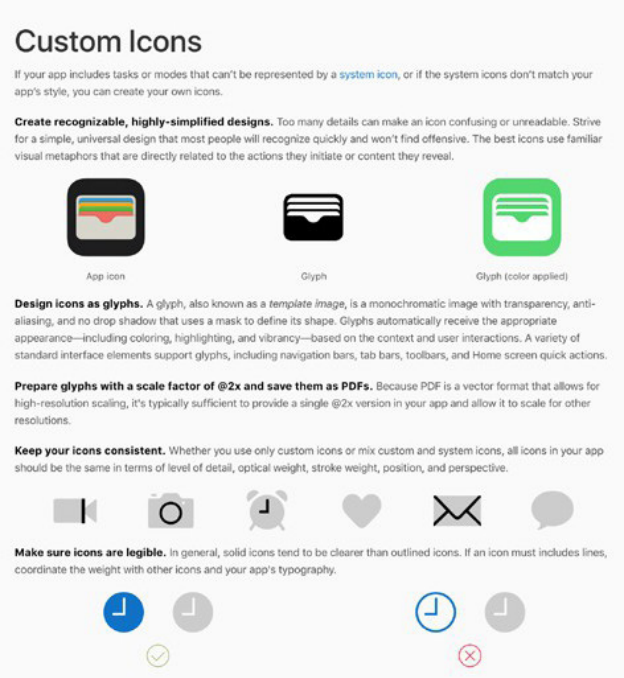


Figure 16: Apple shows the different icon types in their ecosystem: app icons, glyphs, and glyphs used on color.

ILLUSTRATIONS

Illustrations are a great way to add some character to your product. You can use these for empty states, loading screens, modals, and other components that invite visual interest.

Shopify went to great lengths to produce unique illustrations for all of the empty states of their platform, which conveyed a strong sense of brand personality.

Similar to icons, it's helpful to have guidelines for the style of your illustrations.

Figure 17: Illustration guidelines by AI Power.

Visual form

Visual form, or the material quality of your UI, is about the background images, gradients, and textures, shadows and elevation (z-indexes), rounded corners, and borders. These are visual qualities that help emphasize and decorate elements to add visual hierarchy and aesthetics. In any case, all are examples of ornamentation that need to be standardized.

Google does a great job indicating how depth and elevation work with layering of components.

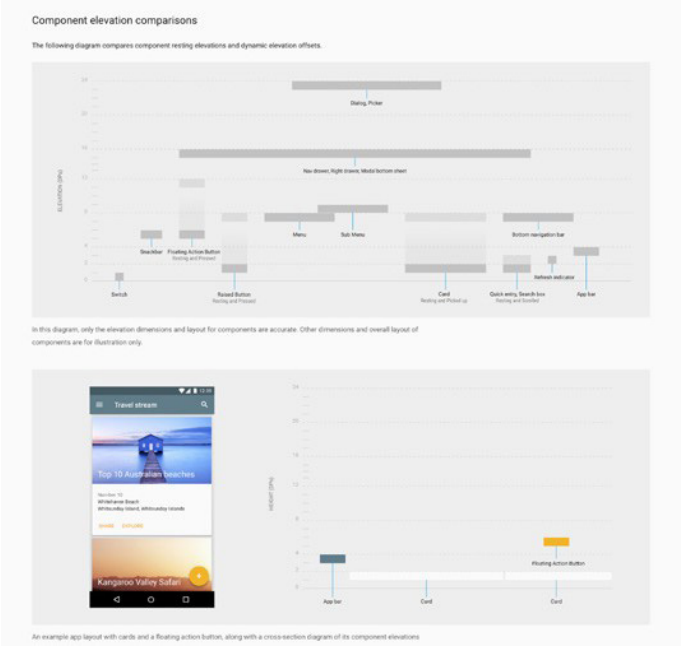


Figure 18: An example of depth through elevation in Google's material design implemented through z-indexes and shadows.

Motion and sound

When you define your visual language, motion and sound might not immediately come to mind. You experience these in a different way. But motion and sound can have a high impact on the experience of your app. You'll want to have that systemized as well for consistency. I personally haven't explored these areas as much as I'd like to admit, but there are some great examples in the wild.

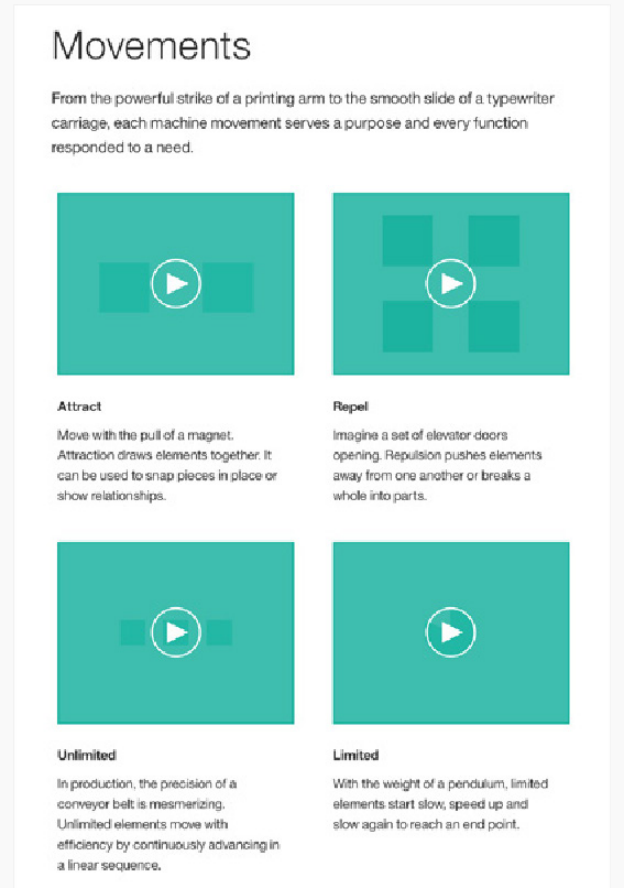


Figure 19: IBM's animation guidelines draw upon their rich history of products and technology.

Creating a user interface library

Before we conducted a visual inventory, which looked at the visual qualities of elements, such as color, spacing, and typography. Now, we will conduct a UI inventory, in which we look at the actual pieces of UI—like buttons, cards, lists, forms, and more. Where visual language is all about the visual approach and how things look on a global visual level, a user interface library (otherwise known as a pattern library) looks at actual components of a UI.

Let's take a look at each of these design elements and the role they'll play in your design system. Take stock of all interface elements in production to see just how much design debt you need to address and what elements are most commonly used. Warning! This can get a bit depressing, as most companies have an intense amount of inconsistency in their UIs.

To create an interface inventory simply open all products in production at your organization, screenshot all buttons, forms, various type styles, images, and collect them in a slide deck or on big posters where the whole team can see.

You can do this with cut out print-outs or through screenshots.

Gather the folks you're involving (as mentioned earlier in this chapter). Have them conduct this inventory with you, either through a shared presentation or via a hands-on activity. The idea is to gather the different components you're using and categorize and merge them.

Some like dividing the pieces into elements, components, regions, utilities, and so on. Atomic Design is a great example of this line of thinking, which is a great conceptual model.

But when it comes down to it, everything is pretty much a component, so at the end of the day, you could label all as such. But in general, what I see most design systems break things down into are:

- elements (or basics, or atoms)—these are small, stand-alone components like buttons and icons
- components (or molecules, or modules)—these are usually an assembly of small components into a larger component like a search form (which includes a form input, a button, and potentially even a search icon)
- regions (or zones, or organisms)—these are an area of the UI like a left-hand navigation
- layouts—how the pieces are laid out on the page (like a header region, followed by a sidebar and main content area, followed by a footer)

After you complete the inventory, you can merge and remove what you don't need (either in a spreadsheet or even directly in a code refactor if you want more immediate change). Also, document what the component is and when to use it. This will become your UI library (or pattern library, or component library, depending on what your organization chooses to call it.).

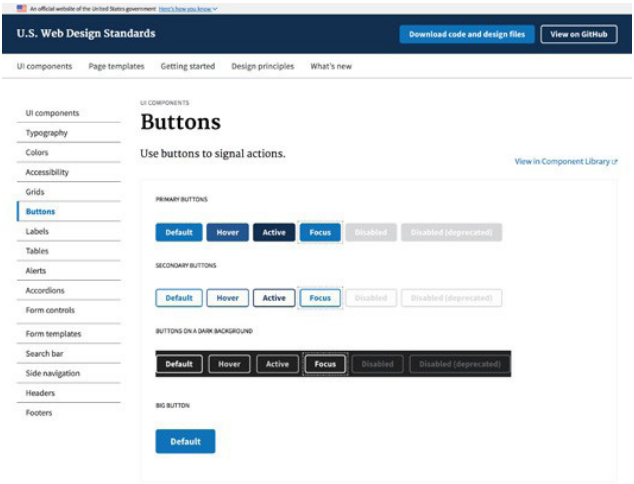


Figure 20: The US Government agency 18F has one of my favorite UI libraries: the U.S. Web Design Standards.

Most design system documentation includes the component's name, description, example, and code. Others may show meta data, release histories, examples, and more. What matters most is that you show what's necessary for your team to get your work done.

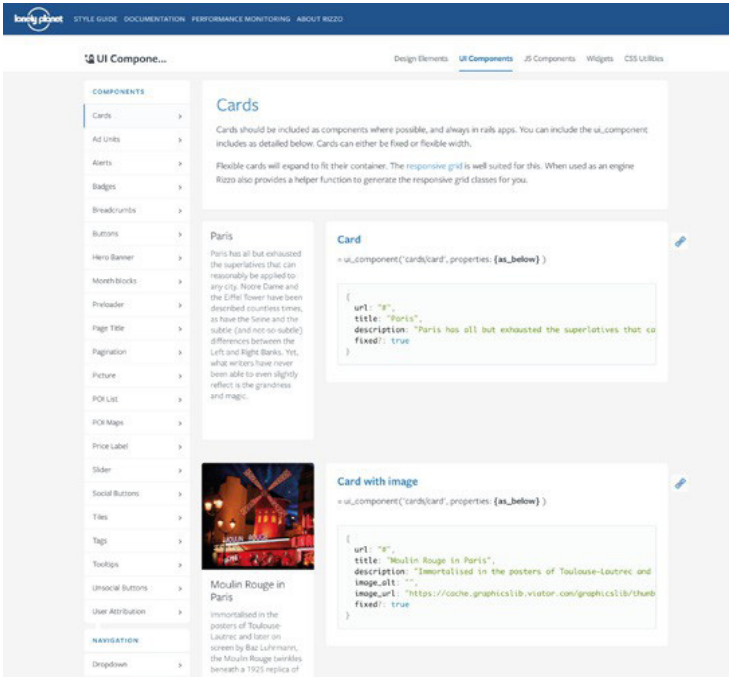


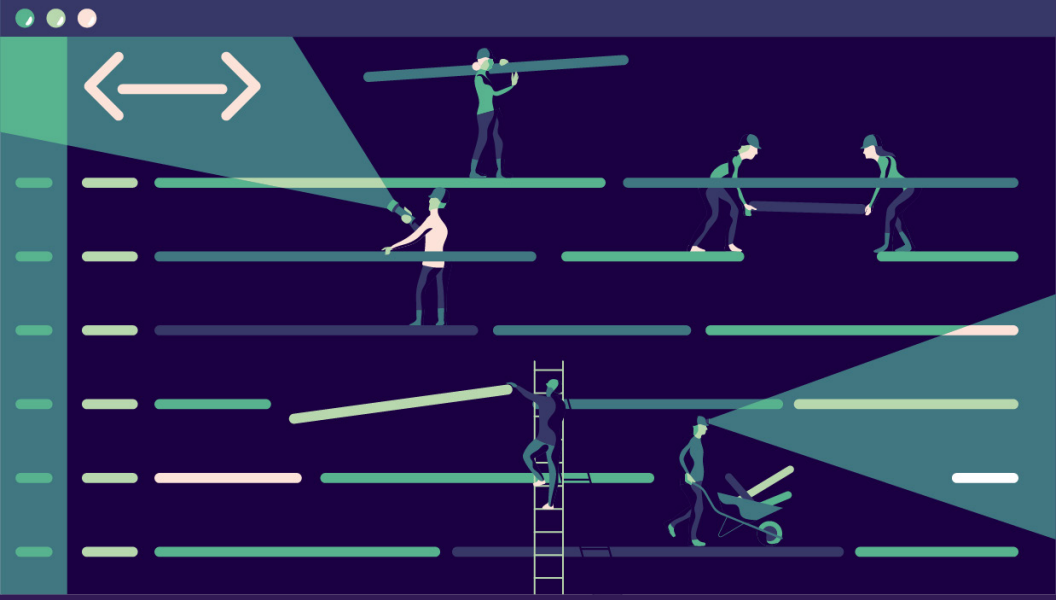
Figure 21: The Rizzo component library by Lonely Planet.

Conclusion

Creating a design system not only helps your team produce more consistent user experiences, it also builds bridges between design and development. By creating a common visual language codified through design tokens, and a set of components and patterns cataloged in a UI library, you'll vastly improve designer/developer communication. You'll also have fine-tuned control of the UI in a way that is manageable, scalable, and robust.

Further reading

01. Priyanka Godbole's Design System article series
02. Nathan Curtis's Design System article series
03. Marcin Treder's Design System article series. Start here
04. Brad Frost on creating interface inventories
05. Building a large-scale design system: How we created a design system for the U.S. government by 18F
06. Design Systems are for People by Jina Anne



Chapter — 03

Building your design system

A strong foundation

By Katie Sylor-Miller

Modern design systems are the result of many years of evolution in the way we write front-end code. When I started my career, most sites were built with single-use, inefficient, fragile, and inconsistent front-end codebases. Through hard- won experience and many years of collaboration and iteration, front-end developers have established more sophisticated practices for writing and organizing our code. Now, there is an explosion of front-end frameworks and tooling to help you write better, more maintainable HTML, CSS, and JavaScript.

This represents an exciting paradigm shift in front-end development, but the number of choices available can be overwhelming. A cursory glance at the table of contents for "Cody Lindley's Front End Developer Handbook 2017's section on tools" reveals a bewildering array of considerations. I couldn't possibly cover all of the technology choices available—the factors that go into your decision-making will be largely situational, and this is only a chapter!

Instead, I'll save you the headache of attending the school of hard knocks by walking you through what I've learned from building and contributing to 3 different design systems. First, I'll cover the technology-agnostic foundational principles that should guide the development of your design system. Then, I'll focus on some common pitfalls and how you can avoid falling prey to them. Throughout, I'll introduce you to some of the tools that will help you along the way, but remember this: Your technical approach doesn't matter as much as creating a living, breathing system that's flexible, maintainable, stable, scalable, and successful in the long-term.

Foundations

Regardless of the technologies and tools behind them, a successful design system follows these guiding principles:

- *It's consistent*. The way components are built and managed follows a predictable pattern.
- *It's self-contained*. Your design system is treated as a standalone dependency.
- *It's reusable*. You've built components so they can be reused in many contexts.
- *It's accessible*. Applications built with your design system are usable by as many people as possible, no matter how they access the web.
- *It's robust*. No matter the product or platform to which your design system is applied, it should perform with grace and minimal bugs.

Let's take a look at each of these principles in more detail.

Consistency

Your first, most important task when starting out is to define the rules of your system, document them, and ensure that everyone follows them. When you have clearly documented code standards and best practices in place, designers and developers from across your organization can easily use and, more importantly, contribute to your design system.

Code style guides

Code style guides provide the grammar rules of syntax and semantics for your code. Code syntax is the set of rules for structuring and formatting your code (e.g. curly braces always go on a new line). Code semantics provide the rules for making your code understandable (e.g. alphabetize CSS property declarations). But don't get bogged down fighting pointless wars over tabs versus spaces. The most important thing is to end up with consistently written code, not to achieve theoretical perfection!

Automating code style

To enforce your code standards and achieve consistency in your system, help your contributors write code that follows the rules through linting and tooling.

> **PRO TIP —** Front-end guidelines questionnaire
>
> Unsure where to start making decisions about your technical approach? Brad Frost has written a handy Frontend Guidelines Questionnaire to guide you.

> **PRO TIP —** A starting point for code style
>
> Start with an open source code style guide—I prefer Airbnb's "Mostly Reasonable" rules for CSS, JavaScript, and React—then modify it to fit your needs. Be sure to include your team's code style rules in your design system's documentation.

Linting is an automated process of analyzing code and raising errors when code either doesn't adhere to your syntax rules or is broken, buggy, or malformed. Linting tools such as CSSLint or StyleLint for CSS, and JSHint or ESLint for JavaScript, can be run manually as part of your local development process, as an automated pre-commit hook before code is checked into source control (the best option), or run as part of your build process.

> **PRO TIP** — Automagically pretty
>
> You can stop worrying about writing your JavaScript according to code syntax rules entirely by using Prettier to automatically reformat code without changing the underlying functionality.

Code editor configuration

An often-overlooked but important corollary to linting is providing an Editor Config to enforce code style in your editor(s) of choice. EditorConfig.org (figure 1) provides a cross- platform format to define stylistic rules for most code editors and IDEs, so you can automatically convert your tabs into spaces—thus ending the tabs versus spaces war!

Figure 1: Example .editorconfig file from EditorConfig.org

Self-contained

Your design system should live in a source control repository independent from your main codebase. Although it will require more work to get up and running, a separate repository brings many long-term benefits:

- It enables versioned releases of your code (more on this later)
- It allows you to share code across multiple teams, products, and codebases
- It forces you to develop components in isolation so they're not tied to a single use case
- It provides infrastructure for a robust front-end testing architecture
- It forms a foundation for a living style guide website

A standalone design system repository functions as a single source of truth. There is only one place where components are defined, which then gets shared into other codebases as a discrete dependency. Because all usages point back to a canonical implementation, changes in a single place propagate through the entire system.

Ideally, all of the code for each component within your system is co-located: CSS, JavaScript, HTML templates, and documentation all live in the same place, possibly the same directory. If you're using React with CSS-in-JS, each component may even be encapsulated in a single file. The closer the pieces are to each other, the easier it is to trace and manage dependencies between bits of code, and the easier it will be to update and maintain.

Reusable

Successful design systems are highly reusable. Bootstrap, the most-used front-end library of all time, powers hundreds (if not thousands) of websites because it was architected with reusability in mind. Writing components to be reused in multiple contexts is vitally important, yet hard to do well— make components too focused for a single use case or too inflexible, and users will end up creating their own patterns.

To be reusable and scalable, patterns need to be modular, composable, generic, and flexible.

- Modular components are self-contained with no dependencies

- Composable components can be combined to create new patterns
- Generic components can handle multiple use cases
- Flexible components can be tweaked and extended to work in a variety of contexts

> **PRO TIP** — Stay DRY
>
> A fundamental best practice in software development is
>
> Don't Repeat Yourself (DRY). When two different pieces of code perform the same function, you double the possibility of bugs, unintended side effects, and the amount of time spent maintaining functionality. The goal of your design system is to DRY up your development and reduce duplication by creating reusable patterns.

Modular CSS architecture

Reusability and scalability in design systems begin with taking a modular approach to your code architecture. CSS, however, is not inherently modular. So over time, systems like SMACSS, OOCSS, and BEM have added structure and modularity to CSS. More recently, CSS-in-JS approaches such as Styled Components have solved the problem by defining CSS properties in JavaScript code structures.

Whether you use one of these systems or roll with your own, the fundamentals of any good CSS architecture are the same:

- It has clear naming conventions for components, variations, and utilities

- It's tightly-scoped and has low-specificity CSS that limits unintentional side effects
- It has utility classes that allow you to modify styles in a managed way
- It has rules for building modular, composable, generic, and flexible components

> "The Web is fundamentally designed to work for all people, whatever their hardware, software, language, culture, location, or physical or mental ability. When the Web meets this goal, it is accessible to people with a diverse range of hearing, movement, sight, and cognitive ability."
>
> ***WEB ACCESSIBILITY INITIATIVE (WAI)***
> W3C

To learn more about architecting CSS so that it meets these criteria, I highly recommend Harry Roberts' https://CSSguidelin.es/ then adding this type of documentation to your system.

> **PRO TIP —** Namespacing
>
> Working with legacy code? Choose a unique, short, and simple namespace to prefix your classes, e.g. `.ds-[component name]`. This will avoid class collisions when mixing multiple libraries on a page, and ensure you know that your `.ds-btn` class is different from the `.btn` class from Bootstrap.

Accessible

For too long, accessibility, or a11y, has been misunderstood as building sites for a small group of users of assistive technology—a blind person using a screen reader—and far too often dismissed as too complex, too time-consuming, or "not our customers." Accessibility, however, is not just for a single, small group, but for an estimated 15% of people worldwide—56.7 million people in America alone—with a wide spectrum of permanent or temporary visual, auditory, motor, and cognitive impairments.

> [Accessibility testing] gives developers a starting point to say 'here are some errors that I have tangible ways to go fix now.

Alicia SEDLOCK — Front-End Engineer & Accessibility Advocate

Thankfully these attitudes are changing and our industry is embracing a more inclusive definition of accessibility. Making your site accessible to users with disabilities improves the experience for everyone who visits your site. If that isn't enough motivation, improving your site's accessibility can help improve SEO, and it's becoming increasingly more important from a legal standpoint to avoid costly lawsuits.

In a landmark case against the grocery chain Winn-Dixie, a federal judge ruled that websites are subject to the provisions of the Americans with Disabilities Act (ADA). If you're just learning about accessibility, there are a lot of resources to help you get started.

I recommend reading the introductory articles from the W3C's Web Accessibility Initiative (WAI) WebAIM, and the A11y Project. You can inspect the current state of your site using Tota11y, an a11y visualizer bookmarklet by Khan Academy. Starting an accessibility practice where none has existed before can be challenging, but when you leverage your design system, it's easier than you might think.

Enforce a11y with your design system

To ensure everyone at your organization builds accessible sites, features, and apps, enforce accessibility best practices in your design system code.

- Test your color usage against established color contrast guidelines (figure 2).
- Build components to be keyboard and screen reader accessible by default. The Ebay Accessibility MIND pattern library is an amazing, thorough resource to help guide development of accessible components and best practices. Encourage contributors to build according to these guidelines and test their code using keyboard-only navigation and assistive technology devices like screen readers.
- Include in your documentation code standards and guidelines for common a11y best practices such as using larger, legible text sizes, always associating a form field with a label, and properly adding alt text attributes to images, to name a few. Salesforce's Lightning design system and Shopify's Polaris are great examples of accessibility guidelines in practice.

Jesse Bennett-Chamberlain Shopify

Listen Online

Implementing And Rolling Out Polaris

These accessible practices improve usability for everyone by making it easier to view, interact with, and navigate a site, improving form completion rates and reducing user mistakes.

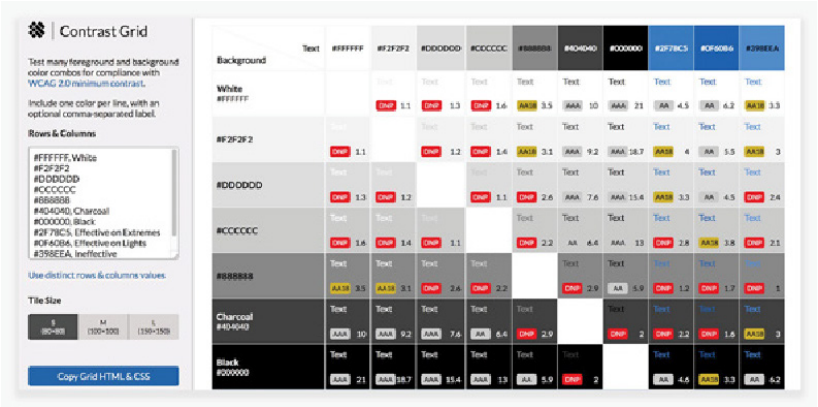


Figure 2: Eight Shapes Contrast Grid allows you to test color combinations for compliance with color contrast guidelines.

Use the power of your single source of truth to create a foundation for accessibility, thus relieving some of the burdens from product teams. It's much easier to build in accessibility from the start than to bolt it on after a feature has been designed and built.

Nate Whitson, LinkedIn

Listen Online: Accessibility

Robust

A robust design system has a strong foundation of tests behind it. Testing provides confidence in your code, which facilitates adoption. Users will know that they can upgrade or change their UI without it breaking in unexpected ways.

Additionally, your design system is uniquely positioned to form a foundation for robustly testing your front-end code.

> I like to think about it almost like...there's an implicit contract that this is always going to work.

Alicia SEDLOCK — Front-End Engineer & Accessibility Advocate

Test your design system instead of your complicated UI

Keeping tests up to date for pages, applications, and features—especially on a rapidly changing site or one with heavy experimentation—requires a lot of work. Tests get out of date quickly! You can narrow the scope of your tests and gain higher levels of confidence in your site-wide front-end code by heavily testing your design system components. You already need to generate example code for the different states of each component for your documentation—use those as your test fixtures.

Types of tests

There are 4 types of tests used for ensuring stability in your design system:

- **Unit testing:** These tests verify that small units of code (usually individual JavaScript functions) behave as expected. Unit tests execute functions with a set of predefined inputs, then verify that they return the expected output. Some popular frameworks to use are Mocha, Jasmine, and Jest.
- **Functional testing:** In functional tests, examples of your code, or "fixtures" are run in a virtual "headless" browser, then tested by performing simulated user actions, and checking the new state of the browser for the expected result of those actions. Functional testing frameworks include Nightwatch, Protractor, and Casper.
- **Visual regression testing:** These tests help catch unintended visual changes to component styles. The test framework takes screenshots of your fixtures both

before and after the changes, then compares them using an algorithm to detect visual differences. There are open source frameworks like Wraith, Gemini, and BackstopJS, as well as paid services like Applitools and Percy.io. Go to Kevin Lamping's excellent resource Visual Regression Testing for more information and options.

- **Automated accessibility testing:** Leverage tooling to ensure that your components are accessible. Some options for running automated a11y audits are Paypal's AATT and a11y by Addy Osmani, and aXe by Deque Systems.

Jesse Bennett-Chamberlain, Shopify

Listen online:
Challenges Implementing Polaris

Common challenges

No system is ever perfect. You will make decisions that you later regret, no matter how much time you put into designing your design system. You can, however, anticipate issues that arise as your system grows, and work to avoid or mitigate their effect on your project. There are 3 common challenges I've seen arise in multiple design systems:

- Keeping documentation up-to-date with your system code
- Handling breaking changes
- Avoiding performance degradations
- Let's look at each of these concerns in detail.

Maintaining documentation

The first time I built a front-end component library, my team decided it would be easier to create a documentation website with a codebase separate from our application. In hindsight, this decision broke the cardinal rule of Don't Repeat Yourself. Whenever a component changed in our main codebase, we had to remember to actually update the documentation, then do the tedious work of duplicating the changed code in 2 different codebases. Unsurprisingly, our documentation got out of date almost immediately!

Learn from our mistakes by reducing the distance between your documentation and code and using automation.

Minimize separation between library code and documentation code

Earlier in the chapter, we discussed storing your design system code in a separate repository that functions as your single source of truth. When documentation and code are co-located, it's more likely that you'll remember to update the documentation when a component changes. Consider adding a pre-commit hook to your design system repository to warn contributors when their code adjustments don't also contain updated documentation files.

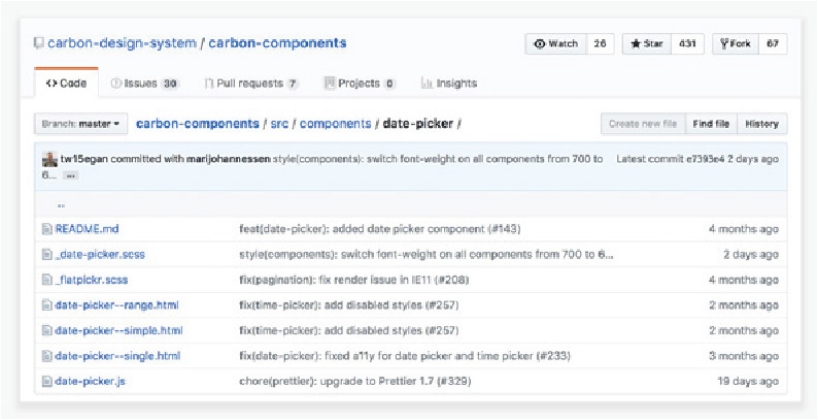


Figure 3: IBM's Carbon design system co-locates each component's CSS, JavaScript, documentation, and example HTML, which functions as both documentation examples and test fixtures.

Automate documentation

Start by documenting your system using simple, human- readable files written in markdown, co-located with each component. Github is already configured to render and display any file named

README.md when you're viewing a folder's contents—you might not need a flashy website at all!

If and when you do decide to create a full-featured documentation website, use automation to simplify the process. Instead of creating a new codebase for a separate website, use a tool that will auto-generate documentation for you—there are lots to choose from—reducing the amount of structural HTML you need to write and maintain.

PRO TIP — Cupper

Heydon Pickering of the Paciello Group has open-sourced Cupper, a documentation builder that creates fully accessible documentation. It's a progressive web application (PWA) under the hood, so you can save and view content offline on supported devices.

PRO TIP — Reducing Your CSS Payload

Consider running automated process to help reduce your CSS payload, then use that information to decide which components belong in your core file. UnCSS (https://dbtr.co/unCSS) removes unused selectors from your CSS and outputs a new, reduced file. Another approach is to automatically determine the CSS necessary to render critical, above the fold content, and embed that in the head of your page to improve render speed. Addy Osmani has compiled a helpful list of Critical path CSS tools (https://dbtr.co/critical-CSS).

Figure 4: The Cupper pattern library builder generates fully accessible documentation websites (formerly called Infusion Pattern Library Builder).

Handling change

As adoption grows and your design system becomes more widely used, you will invariably realize that you didn't get it all right the first time, and you will need a plan to handle breaking changes. A breaking change is a situation where necessary changes to a component's code will break existing usages of that component or class. The Morningstar design system provides guidance to contributors on what is considered a breaking change. If mishandled, breaking changes can be a major pain point for your users.

The wrong way: duplication

Initially, all of the CSS and JavaScript for Etsy's Web Toolkit lived in the same monorepo with the rest of the team's site code. This meant that whenever someone made a breaking change to a component, their commit making the change had to also contain fixes and updates for every single usage of that component. At first, when just a small team built and used the system on a subset of pages, finding and making these changes was relatively easy. But as adoption spread throughout the company, this quickly became unmanageable.

It became such a headache to make major structural changes to existing components that our team at Etsy started to duplicate and deprecate—when we refactored our Tab component to make it fully accessible, we created a new component named "Tabs2," and deprecated "Tabs" in the hopes that teams would take on the work to upgrade their code. But without clear guidance on how, why, and a timeline stating when to upgrade, most uses haven't been updated to use the new component. This kind of duplication is a code maintenance nightmare.

The right way: versioning

Breaking changes are easier to manage if you store your design system code in its own source control repository. This gives you the ability to do versioned releases of your code, which can be shared with other projects. Git allows you to tag a commit with a release version number, and package managers like npm and Yarn allow you to package up and publish multiple versions of your design system code.

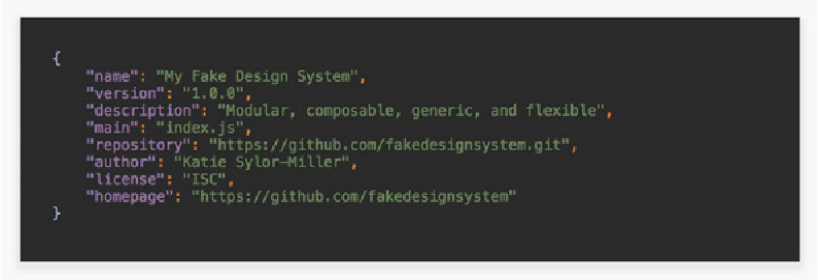


Figure 5: Example package.json, the package definition file format for npm.

With versioned releases, the adopters that integrate your design system code can target a specific version as a dependency and control when and how upgrades to new versions are handled. Make sure to publish release notes detailing changes so other teams can learn how upgrades will impact their codebase and better plan for upgrade work in their project timelines.

PRO TIP — Using semver

Many versioned projects use semver, short for semantic versioning, to distinguish between types of releases. Semver uses 3 integers separated by dots to indicate major.minor.patch versions, for example, v1.2.3.

- Major versions (1.0.0) contain breaking changes from prior versions.
- Minor versions (0.1.0) add new functionality that is backward compatible.
- Patches (0.0.1) contain bugfixes for existing functionality and are backward compatible.
- Semver adds another layer of confidence that certain upgrades should be seamless to users, while others may require regression testing or code changes.

Avoiding performance issues

As a design system grows over time, generally so does the file size of assets sent over the wire on each page. This can negatively affect your site's page load performance, which in turn negatively affects your company's bottom line. Help the products built on your design system avoid performance issues by taking a mobile-first approach and build your system with modularity in mind.

Mobile first

Poor page load performance will have the largest negative effect on mobile visits. According to statistics from Google and Akamai, more than half of mobile visitors will abandon pages that take longer than 3 seconds to load, yet the majority of mobile sites take longer than 10 seconds to load. As mobile traffic overtakes desktop, speeding up your page's performance is more important than ever to give you a competitive edge.

Build your design system mobile-first—test early and often on real devices with real hardware and a real network connection so you can understand the experiences of real users.

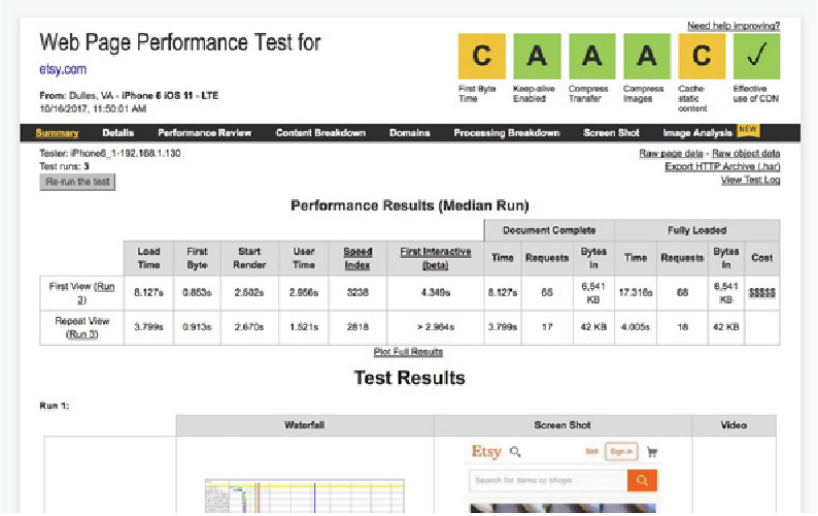


Figure 6: WebPageTest performance results for Etsy.com on an iPhone 6 over LTE network.

Leverage modularity

Initially, it made sense to bundle all of the Etsy Web Toolkit components and utilities into single files for CSS and JavaScript. While this is useful for prototyping, it adds unnecessary weight to production pages that don't use all the components. Now, we're working to avoid performance problems by better using the modularity inherent in the system. To do this, we are:

- Deciding on a set of core components and utilities that are most frequently used. This base file will be included on all pages that use the design system and can be cached by the browser across page requests to improve load times.
- Ensuring that all components are fully modular with no cross-component dependencies (unless they are explicit and managed by the system).

Packaging and sharing the system's code so that it can be consumed as individual CSS and JavaScript modules added as dependencies only when a component is actually used on the page. We use an in-house system to define dependencies in our front-end code. Build your design system modules so they work with the dependency manager that your team uses. Webpack is a popular option.

Forward-thinking

Design tokens

For most of this chapter, I've focused on building design systems for web applications and sites. However, that's not the full picture. Modern organizations face unique challenges with their design systems at scale. Today, we build for multiple web and native platforms that need design consistency. Larger organizations may have multiple sub-brands that want the shared support, functionality, and organization that a design system brings, but each needs a different, brand-aligned look and feel. The Salesforce UX team introduced a solution to both of these problems: design tokens.

Cross-platform sharing

Design tokens are a way to abstract design details like colors, fonts, rounded corner radius, etc., into a format that can be shared across platforms using Salesforce's Theo tool. Instead of defining your main brand color as a SASS $variable in your web app, a UIColor in your iOS app, and a textColor in your Android app, you define a single design token in a shared JSON file that gets compiled into platform-specific code. Decide to change all of your rounded corners from a 3px to 5px border- radius? Change the value once in your tokens file, and it propagates to all of your apps automagically.

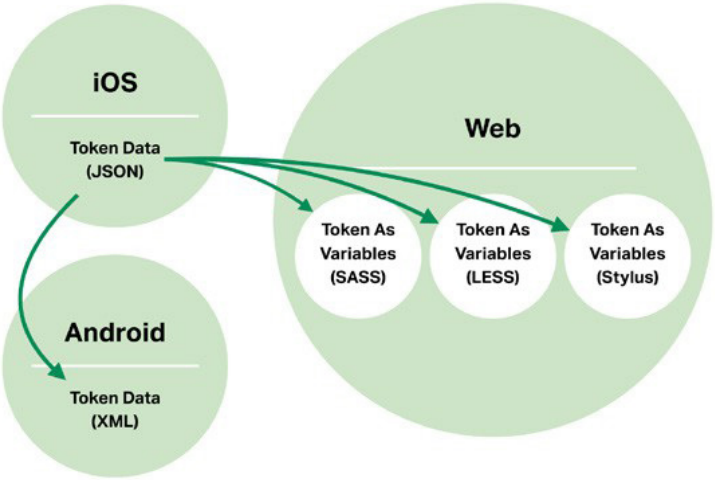


Figure 7: How design decisions are propagated via tokens in design systems.

Multi-product theming

You can also use tokens to "theme" the same structural styling for multiple brands. One brand wants orange buttons and the other wants blue? No problem! You can define different token values for each brand, then combine each to the same, base CSS to output themed versions of your design system. That way, all of the classes are the same, and all of the accessible JavaScript functionality you worked so hard to build can be used as-is with no modification for each brand.

Conclusion

Systematizing your front-end code benefits everyone in your organization: developers move faster, designers don't have to reinvent the wheel, and ultimately, your users have a better experience. But starting out can feel overwhelming!

Thankfully, there is a wealth of resources and a growing community of folks sharing their own best practices to help you build your system. Anna Debenham's "Styleguide Resources" lists nearly 500 public design systems/style guides/pattern libraries to use as inspiration. While they look and feel very different, each shares a common goal and is built upon foundational principles that our industry has developed through many years of trial-and-error.

A flexible, maintainable, stable, scalable, and successful design system begins with a strong foundation that goes beyond frameworks or tooling. If I've learned anything in my career, it's that the only constant in front-end development is change. Chances are, within a few years, the technology driving your front-end will look very different than it does right now. If you build for the long-term and plan to handle change, then you can implement new, better practices as your system grows.

That's not to say you should aim to create another Bootstrap right out of the gate. Even if your team can only implement a few of the best practices we've discussed in this chapter, it's still worthwhile—some of the techniques I recommend haven't yet been implemented in the design system that I work on every day! Design systems are not a one-and-done thing, but a continual process of iteration and change as we make mistakes, learn from each other, and create new and better approaches to front-end code.

Further reading

01. Website Styleguide Resources

02. Styleguide Driven Development

03. Styleguide Best Practices

04. CSS Architecture for Design Systems

05. Front-end Architecture for Design Systems (book)

06. Accessibility for Everyone (book)

07. Web Accessibility Resources

08. Front-end Performance Checklist

09. Designing for Performance

10. Rebuilding Slack.com

11. Tokens in Design Systems

Chapter — 04

Putting your design system into practice

Better together

By Diane Mounter

How you develop your design system will influence how you share and encourage adoption of the system. Broadly speaking, there are 2 general approaches to developing and rolling out a new design system: incremental and large-scale redesigns.

In this chapter, we'll walk through the 2 approaches, highlight the pros and cons of each, and discuss adoption strategies.

Large-scale redesign

When a team takes this approach, it often means they spend more time designing the system before rolling it out everywhere, including doing a visual refresh or consolidation of components. This allows the team to develop a fuller system—from the primitive layer of colors and typography to components, page layouts, and interaction flows.

Some teams approach their new system by creating imaginary products to help them step away from the constraints of working within a real application. Other teams design their new system alongside the redesign of a real product or feature.

Testing with pilot projects

At Etsy, I worked with a team of designers and engineers on the redesign of seller tools. This project provided a great opportunity to test a new approach to our CSS and refresh our visual styles. We built a new style guide alongside the seller tools redesign that documented implementation and design guidelines.

This became a pilot project for testing the new design system. Rebuilding a real part of the product gave us a great playground for testing new components, responsive layouts, and new typography styles.

"In From Design Systems: Pilots & Scorecards," Dan Mall writes about the criteria he uses to find good candidates for design systems pilot projects. Though our approach at Etsy was more opportunistic, many of the attributes Mall lists happened to match the seller tools project:

01. Potential for common components. Does this pilot have many components that can be reused in other products?

02. Potential for common patterns. Does this pilot have many patterns that can be reused in other products?

03. High-value elements. Even if uncommon, is there a component or pattern with high business value at the heart of this project? We're talking about elements that are integral to a flow or audience with unusually high value for the organization.

04. Technical feasibility. How simple is a technical implementation of the design system? Is a large refactor required?

05. Available champion. Will someone working on this product see it through and celebrate/evangelize using the design system (and even contribute to it)?

06. Scope. Is this work accomplishable in our pilot timeframe of [3–4 weeks] (insert your timing here)?

07. Technical independence. Is the work decoupled enough from other legacy design and code that there are clear start and end points?

08. Marketing potential. Will this work excite others to use the design system?

> "We don't really have one team doing all the work...pretty much every designer and engineer in our organization is a part of the system." https://dbtr.co/twitter-horizon
>
> 
>
> **Tina Koyoma**
> *Twitter*

There was one downside to building the new design system alongside a product—it ended up being biased to the needs of that product. Afterward, it needed further development to work for the whole Etsy application. Working in a silo with minimal outside feedback allowed us to make progress quickly, but meant that we had to work harder to encourage other teams to adopt the design system.

Showing value through a sandbox environment

While working on the new Etsy style guide, we set up a simple sandbox environment that allowed us to quickly prototype HTML/CSS mockups. As we prepared to share the new style guide with other designers, we realized the sandbox could be helpful with adoption.

One of our goals was to reduce the time designers and developers had to spend writing CSS, so they could spend more time iterating on designs. The new approach we took with CSS, combined with the sandbox environment, made prototyping designs in the browser fast and easy.

The best way for people to see value is to experience it.

We set everyone up with sandboxes for training. This approach empowered people to prototype and experiment with the style guide using the sandbox, revealing value through hands-on use.

To set people up for success, we wrote tutorials teaching them how to apply different styles—such as atomic vs. component classes, build mobile-first responsive layouts, and create complex views such as a search results page—without writing CSS. To help people understand the why behind our decisions, we included a presentation on our design principles with example scenarios, code samples, and live coding demos.

The sandbox's CSS environment mirrored production, so a designer's prototypes easily translated to a developer's production work, avoiding costly and unnecessary new CSS. Since code prototypes are written in the same language, they gave developers a better sense of the intent than a static mockup. Developers didn't have to translate like they do with comps from design tools. This further helped with adoption. A design system that both designers and developers love has a greater chance of success.

Documentation is key

Another key to adoption is up-to-date documentation. As design systems peers say, "If it's not documented, it doesn't exist."

When styles go undocumented you run the risk of people writing new but duplicate code. Documentation becomes more important when introducing a new design system because old patterns can outweigh new ones. Documentation helps promote those new patterns, reduce the need to write new code, and makes implementation easy with code examples and guidelines.

Outdated documentation can cause damage too since it can lead people down the wrong path and cause frustration. It's worth investing time in developing practices that help you keep updated documentation.

Find opportunities to check documentation accuracy—such as before and after an onboarding session, or as you add new styles. Writing docs on the fly helps you test your code and avoid building documentation debt. If you can, add tests that check documentation when patterns are added and updated, and try to make it easy for people to report inaccuracies.

> **PRO TIP —** Documenting styles
>
> Document styles as you add them. It's easier to do when you're writing the code while it's fresh in your mind than waiting to do it later.

At Etsy, some engineering teams created onboarding projects for new engineers. This usually involved building a small feature from the backlog that would help them get familiar with the engineering stack. I was stoked to see them implementing UI layouts without help from designers, just by following style guide documentation.

Post-rollout follow-up

Design systems are never done. The launch of your new system should be thought of as version 1.0, with many iterations will follow.

Whether you form a full-time team to evolve and maintain your system or not, you likely have a few groups of people interacting with it—makers of the system, users of the system, and users who also contribute to the system. Whatever form your team takes,

makers need to stay in touch with the needs of users and the organization.

A few months after rolling out the new design system at Etsy, we ran working sessions with staff from a cross-section of teams. Our goal was to learn how to improve the experience for people using and contributing to the system, how successful we'd been with promoting the system, and where it was lacking. We took a qualitative research approach so that we could be open to discovering things we didn't realize were problems—this meant a lot of face-to-face discussions with light agenda's to stimulate conversation.

> "Clients, colleagues, and stakeholders should embrace the pliable nature of the digital world to create living design systems that adapt to the ever-shifting nature of the medium, user needs, and the needs of the business." (https://dbtr.co/frost).
>
> 
>
> **Brad Frost**
> Author, Atomic Design

> **PRO TIP —** In-person feedback
>
> Make time for in-person feedback sessions with internal stakeholders, it will expose you to insights that might otherwise never be uncovered.

CONTRIBUTORS

We ran a user-journey mapping workshop with first-time contributors to understand their experience. We suspected people had mixed experiences—some seemed to struggle or give up on their contribution entirely, and some had taken a really long time to add their new pattern.

Findings: We discovered how deflating the experience could be even when the contribution was successful. This led us to develop a more collaborative pairing approach and take an encouraging mindset toward code review for first-time contributors.

ENGINEERING EARLY ADOPTERS

We met with early adopters on product teams with a particular focus on understanding the engineering perspective. Most engineers hadn't taken part in training sessions and had learned to use the new design system by reading the style guide.

Findings: We learned a lot about out holes in our documentation. For example, tutorials didn't lend themselves well to engineers, utilities were useful but buried, and people felt concerned about how to use styles safely in experiments. This led us to reorganize and elevate important information up-front, improve search and navigation, add tutorials that appealed more to engineers, and provide more clarity on style usage.

PRODUCT MANAGERS

We ran a session with product managers to walk through the new system and discuss the impact of refactoring and building responsive layouts as part of feature development. Building a

design system that was responsive by default was a core goal—and new for many parts of the Etsy web application.

Findings: We learned that most product managers trusted their engineering managers to guide them with impact on scope—this confirmed communication with engineers was a priority. PMs understood there might be extra work initially, but moving to the new design system would make front-end implementation easier in the future. We showed them they could scope work and make progress iteratively.

Collaboration creates investment in adoption

It might not be logistically possible to involve your entire company in design systems decisions, but some level of collaboration is worthwhile.

Before Etsy formed a dedicated design systems team, a working group made of a cross-section of designers drove the evolution of the system. This group met regularly to plan and prioritize projects, share work, and get feedback.

Group members collaborated with engineers and specialists where needed, in situations like running experiments on high converting pages and measuring performance.

These collaborations spread adoption even further and resulted in design system champions across the company. People who spent time contributing to the system were invested in it and promoted it to others. Working in a silo had helped us for the short term, but extending contributions to outside the team set us up for success in the long term.

Nate Whitson, LinkedIn

Listen Online:
The Origin Story of Art Deco

Incremental rollout

Not all design systems teams are able to take the time to develop a fully-fledged system before rolling it out. Many teams have to roll out parts of a design system at a time.

On the plus side, with an incremental rollout, teams can adopt new parts of the system as they become available, which can feel less daunting and disruptive. However, teams will also need to give more thought to communication and promotion of the system. Without a launch, there isn't a big moment to get everyone's attention. Instead, you need to find and create occasions to introduce people to your design system.

When I joined GitHub and began exploring how the small, then-part-time team might turn Primer into a more robust design system, I knew we weren't able to go away for a long period of time and develop a complete system. There was a ton of work in flight, and no planned re-design or siloed feature we could use as a pilot project.

We needed to figure out the biggest pain points, reduce them to their smallest part, and roll out Primer updates incrementally.

Solve problems and win early adopters

One of the biggest pain points at GitHub was the amount of time people were spending writing CSS, particularly for things that should have been systematized, such as spacing and typography styles.

Our solution ended up being the introduction of utilities (single purpose classes, often referred to as atomic or functional classes) based on system variables. We weren't able to start with auditing components and page layouts—we had to start small and test the foundation of the system before extending to larger parts. Adding utilities enabled us to make styles available without refactoring tons of UI. Designers and developers could start to use the primitive styles of the system as soon as we rolled them out.

Once people discovered they could use utilities instead of write new CSS, Primer started to catch speed. It solved a real problem while allowing us to test the primitive layers of the system. From there, we could start to replace and update the component layer of styles.

Improve documentation and findability

Like at Etsy, documentation was key to adoption at GitHub. Tons of widely used patterns in the GitHub codebase weren't in Primer and thus weren't documented.

We sorted through piles of custom CSS and created a directory in the GitHub codebase that formed a "waiting room" for patterns that should be moved into Primer. We focused a lot of effort on getting as much as possible documented, whether it was in Primer or not. We also focused on adding more layers of documentation, including describing our class naming conventions, accessibility

principles, recommendations for tooling to work with our system, how to run linters, and an overview of our style organization and packages.

Just having styles documented wasn't enough, though. We needed to prioritize the style guide navigation and search for findability.

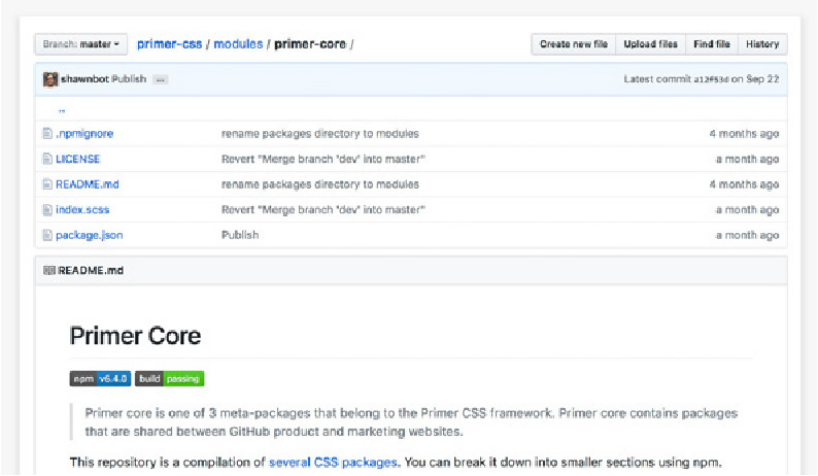


Figure 1: GitHub's Primer, components and behaviors for building things on the Web.

We originally modeled the navigation to match our package organization—core, product, and marketing—prioritizing that information over the findability of searching for specific styles. The style guide also lacked search and hid styles behind several layers of navigation, forcing multiple clicks before people found what they were looking for.

I found myself acting as a human style guide search, with people pinging me to find what they were looking for rather than finding it themselves.

With research outlining people's struggles to find styles, and the fact that we were wrestling with an aging web app we originally used for the style guide, we decided it was worth the effort to rebuild and redesign.

The new documentation site listed all our styles in the navigation, and we added a contextual search that helped people find documentation with similar keywords—such as finding "color" within utilities vs. support variables.

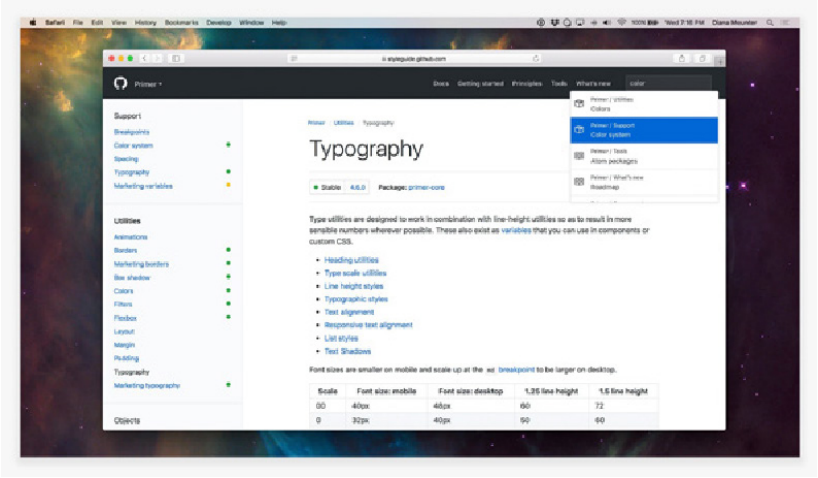


Figure 2: Primer documentation and search.

Getting a documentation site set up early on was important. Looking back, I don't think it was the wrong choice to repurpose an old web app at first—it may not have been ideal, but it did what we needed. It is important to re-evaluate the quality of your documentation site over time though, to ensure it meets user needs, and is scaling with your design system as it grows.

Grow adoption through many touchpoints

When you don't have that big moment to launch your design system, use every opportunity to share its value. You're building systems to solve real problems and you know what your goals are—take some time to share those goals and how you plan to reach them. These details are helpful when you're trying to show how changing something like a few lines of CSS matters.

At GitHub, we used lots of small interactions to promote Primer and the design systems team. Here are a few ways we made our piecemeal approach successful:

CREATIVE COMMENTS IN CODE REVIEW

We started to jump in and comment on pull requests in the GitHub app that touched CSS or design patterns we were trying to improve. This gave us an opportunity to suggest changes in line with the new design system, and point people to our documentation site so they knew it existed.

We spread the word about our team and our availability to help by adding simple comments to pull requests, such as, "Ping the design systems team," or, "Find us in the design systems Slack channel."

Over time, people started to CC us on issues and pull requests, and we took advantage of features like Code Owners so we got automatically requested for review when someone made CSS changes.

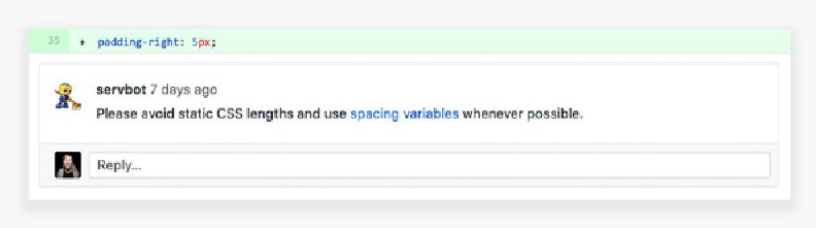


Figure 3: Servbot comments on pull requests to recommend using system variables instead of static values.

Recently, we've started adding bot scripts that comment on pull requests with simple feedback. With lots of points of communication about our team and Primer, we're growing awareness and adoption of design systems.

SHOW, DON'T TELL

We noticed when teams updated the GitHub UI, they would go to another part of the website and literally copy and paste the markup and CSS. Refactoring away old instances of patterns is our best fight against the continued reuse of old patterns. It is a long-term initiative, but we can prioritize the most popular and troublesome patterns.

RESPOND QUICKLY TO SUPPORT REQUESTS

Most teams at GitHub have an on-call duty rotation called First Responder. It means one or more team members are on call to triage issues, respond to support requests, or provide code review.

As support requests grew for design systems, we adopted this process to help respond to people in a timely manner. Over time, we've iterated on the process and created a number of automated

scripts that help track notification items that need attention. Responding to people quickly increases the likelihood that they react positively to our team and recommendations.

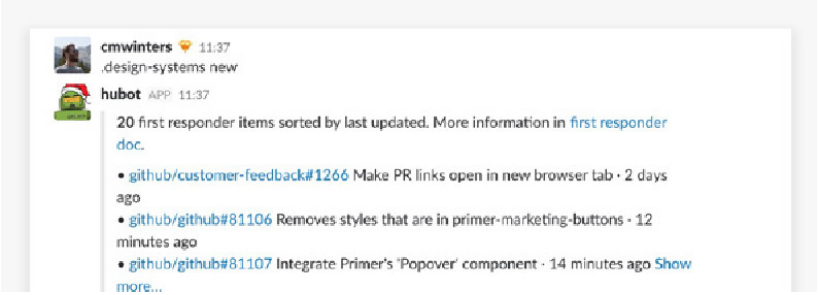


Figure 4: First responders use a hubot script to see which items need their attention.

PRESENCE AND RESPONSIVENESS

Particularly early on, we made the simple effort to be available via Slack to answer questions, pair on code, or jump on a call to walk through something. It takes time away from deeper work, but allocating time to this is worth it to help win friends and champions on other teams.

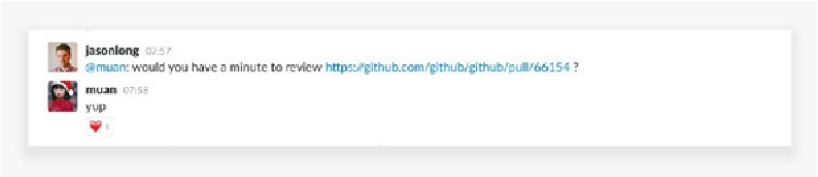


Figure 5: The GitHub Design Systems team are available to help in our slack channel.

Publishing and distribution at scale

Most of my recent experience working on design systems has been on large-scale web applications like Etsy and GitHub. The scale to which those systems need to be distributed and published usually requires more complex infrastructure than a small company with a small system and small team.

However, many companies intend to grow the size of their team and scale of their product over time, so understanding how to set your design system up for scalability will benefit you in the future.

Let's walk through methods of style organization, distribution, and public vs. private documentation and code, to help you consider what's right for you and your team.

Package management and organizing modules

Design systems should be built for change, large or small. This isn't just important for how you code your system or use things like variables or design tokens—it's important for how you organize modules, and version, and distribute them too.

At GitHub, we iterated on how we organize and package styles multiple times. Like Etsy, we pulled our design system out of the GitHub monorepo into its own repo. We wanted to have more control over the changes made and when those changes made it back into GitHub.com.

We also use Primer for more websites and applications beyond GitHub.com. Conference websites like git-merge.com are built with Primer, as well as community websites such as developer.GitHub.com and opensourcefriday.com.

Not all these sites need the full suite of styles we use for GitHub.com, and GitHub.com doesn't use all the marketing- oriented styles in the core application. This led to taking a modular approach with style organization and influenced how we packaged and distributed Primer.

Nate Whitson, LinkedIn

Listen online:
Versioning

Versioning the entire system vs. versioning by module

Versioning the entire system means everything within the system belongs to just one version number and can only be installed in its entirety. This could be likened to browser or software versions. When you update to a new version of Google Chrome or your phone's OS, for instance, you're updating the whole piece of software in one go.

If you were to version your design system in the same way, it would mean everything within the design system would be updated too.

For example, you may have updated your font styles, added a new navigation component, or deprecated an old grid layout. When a user of your design system chooses to upgrade, they get all

of those changes together. This still gives teams the flexibility of when to update the system, but a more granular approach can be helpful as your system scales.

Versioning individual modules means having a version number for every component or group of styles within the design system. So if you put your button component into one module and your utility styles into another, they would each have their own version number, such as primer-buttons@2.4.0 and primer-utilities@2.8.0.

You can still maintain a package that includes the entire set of the modules and version that too. For example, primer-css@9.5.0 includes primer-buttons, primer-utilities, and all the other Primer modules. So you can provide the best of both worlds—the option to use the entire system, or just use individual modules.

This modular versioning approach does require more effort for initial setup because you need to work out what to boil each module down to. For instance, should you have a module for layout utilities as well as a grid system, or combine them all into one layout module?

The benefit of this approach is that users of your design system can choose to upgrade just the bits they need. Giving teams the option to selectively update when they have the time to do it can mean they're more likely to iteratively keep up to date—whereas the overhead of updating the entire system in one go can act as a barrier.

Versioning by package enables a design systems team to push updates more frequently without the pressure of forcing an update to the entire system.

Compared with versioning the entire system in one go, versioning by module creates more flexibility around major design systems

releases, can lead to a culture of continuous development, and enable your systems team to move faster overall.

Releases, branches, and version numbers

Design system teams often have to balance competing priorities. Attending to bug fixes or other commitments has to be balanced with intensive projects requiring more research and planning, such as developing a new color system or bringing in a newly supported feature like a CSS grid.

Your publishing workflow needs to work for both the users and the maintainers of the design system. Maintainers want to have confidence in what they test and ship. Users of the system want to have clarity on status and what type of updates they're getting.

At GitHub, we found the combination of versioning and organizing releases with Git branches gave us the right amount of flexibility in planning and shipping new design system releases.

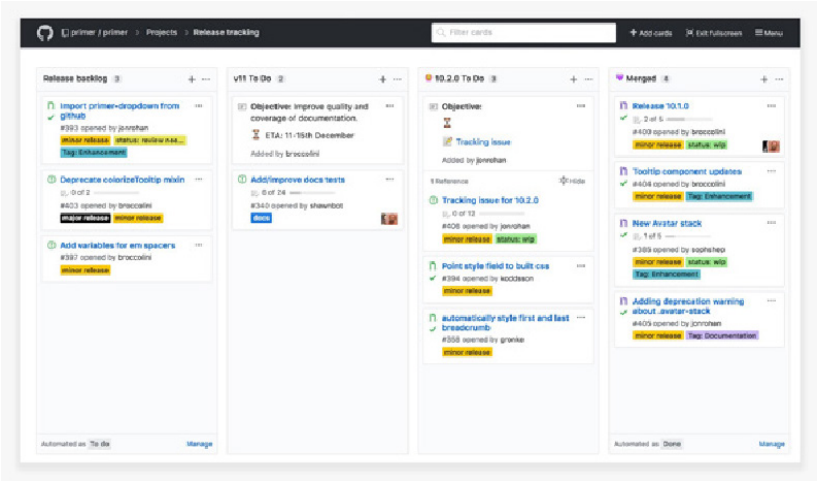


Figure 6: Tracking multiple Primer releases with GitHub Project boards.

We maintain a "dev" branch that includes work in progress and creates a new branch for each individual release. Since Git allows us to create multiple branches of the design systems code, we can work on a minor or patch release at the same time as a major release. This means we can take our time with a major release that might include breaking changes, while also shipping timely updates and bug fixes in minor and patch releases.

Versioning our system and using Semver makes this workflow possible. We can clearly point to a specific version number for each release, and know by the Semver system what type of updates are included with it, and therefore what type of testing we need to do.

Since Semver is a well-recognized standard for versioning, it helps clearly communicate to users of Primer—internally at GitHub or externally—what updates they're getting and what type of testing they may need to do in order to use the updates.

Nate Whitson, LinkedIn

Listen online:
Building a Public vs. a Private System

Public vs. private

As your system and its number of users grow, you might find yourself considering whether to make your design system public

to some degree. There isn't a one-size-fits-all solution to this—it should be based on what's right for your team. Making a system public doesn't have to be an all-or-nothing approach, there are multiple ways of breaking down what's public.

Many pioneering teams have created and published beautifully designed documentation, such as Material Design, Lightning Design System, and Shopify's Polaris. We shouldn't assume the solution that worked for these companies is right for everyone else. These are companies making a significant investment in design systems. Plus, they have the broader objective of wanting external developers to use these systems to build on their platform.

When deciding what, if anything, is made public, you should prioritize based on your company's needs, rather than standards set by other companies.

Here are a few ways you can make your design system public:

PUBLIC DOCUMENTATION ONLY

You might decide that making the source code public isn't right for your team, but you want to make the documentation public.

Marvel's style guide and MailChimp's patterns are examples of public documentation sites that don't (yet) share their source code.

OPEN-SOURCE DESIGN SYSTEMS

Many companies open-source their design system. This means the general public can open issues to request new features,

give feedback, or let the maintainers know of bugs. Maintainers can also choose to accept contributions in the form of code or documentation changes via pull requests. If the maintainers choose, they can make the design system available for modification and reuse by adding a license.

The US Web Standards, Primer, Help Scout, and Solid are all examples of design systems open-sourced on Github.com.

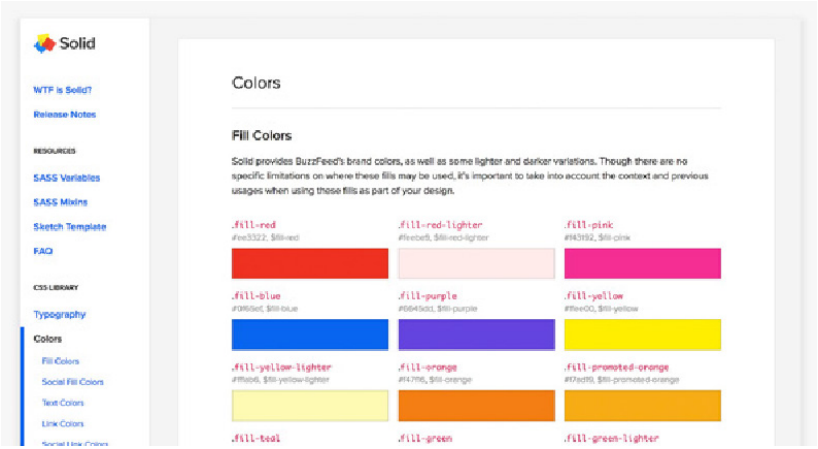


Figure 7: Buzzfeed's design system, Solid, is open-sourced on GitHub.

OPEN-SOURCE THE DOCS WITH YOUR CODE ON GITHUB

You don't necessarily have to share a crafted documentation website—you can just share documentation written in markdown format and it will be rendered and styled on GitHub. Another option is to publish the docs via GitHub pages. This can be a low-barrier way to share documentation since GitHub hosts the documentation, giving you a default public URL if you don't add a custom domain.

SHARE A ZIP FILE TO DOWNLOAD

If you want to keep the source-code and/or the npm packages private, you could choose to share the code for your design system so it can be used by others. This might be a nice option if you don't want to make in-progress work on your system public, or add the overhead of managing open-source contributions, but are still happy to share the code for use by others.

Projects like Lab by Compositor (a design tool for creating React components for design systems) use the Releases feature on GitHub to share release notes and a ZIP file download of the software. They also use the repo to host documentation and issues to get feedback from users.

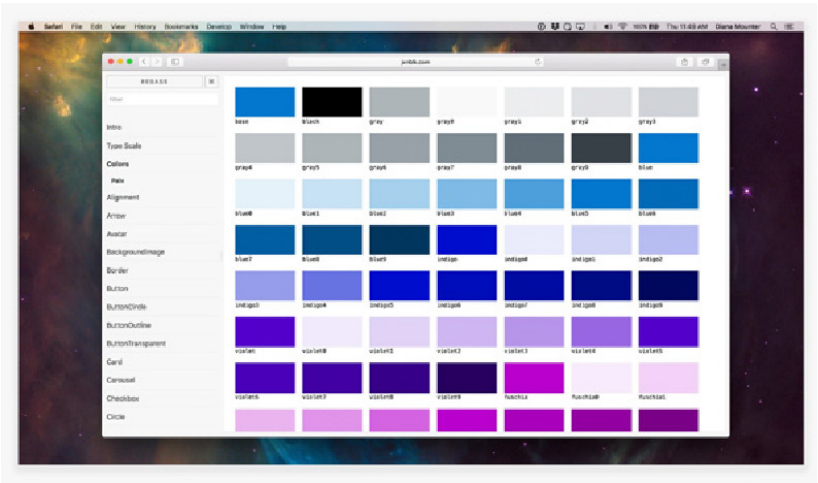


Figure 8: Color swatches in Storybook.

PUBLISH A STORYBOOK OF YOUR UI COMPONENTS

Storybook has become a popular tool for design systems teams, especially for those using React, though it can also be used with

Sass or CSS-based design systems too. As Brad Frost describes, Storybook is a "workshop" tool designed to output rendered examples of your styles and components, which you can use in development for testing changes. Style guide documentation is your "storefront" and includes crafted and detailed documentation with information like usage guidelines and code-style principles.

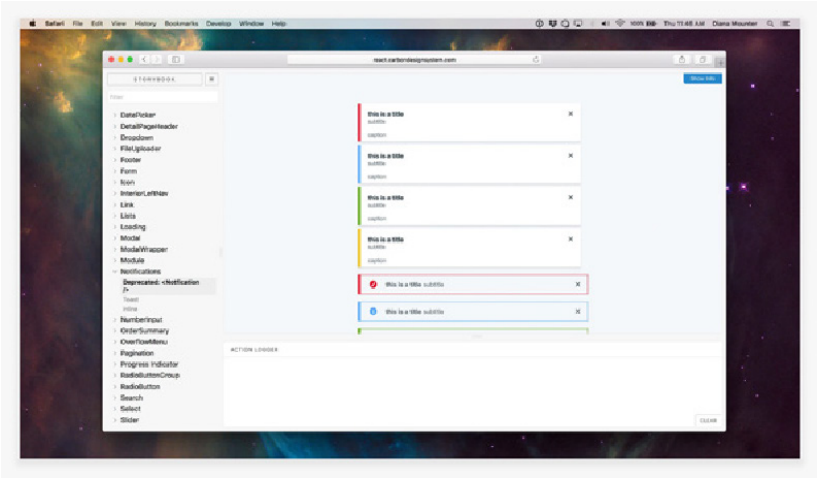


Figure 9: Notification styles in Storybook.

Providing rendered examples of your components, even without detailed documentation, is still very useful to users, maintainers, and would-be contributors of your design system. It provides a catalog of what's included in your design system.

Pattern Lab (created by Brad Frost), Fractal, and React styleguidist are other tools that provide similar features to Storybook with options for documentation and code examples. All these tools can be used without sharing your design system's source code, and give you options as to what level of documentation you share.

As you consider varying degrees of access to your system, also consider some of the reasons teams make their systems public:

NO AUTHENTICATION BARRIERS TO SHARING

Anything behind authentication, like outside VPNs or firewalls, means one more barrier to access. I know this firsthand since we have a new, in-progress documentation site for Primer that requires staff login. People frequently think the style guide had been taken down or don't even know it exists.

Authentication also makes it difficult to give external contractors access. It's a small barrier, but it can have a negative impact and mean people who could be using your design system documentation can't or won't because it's too much hassle.

HELPS WITH RECRUITING THROUGH PREVIEWING DESIGN MATURITY

Design systems are a popular topic in the design and front- end community right now and might be part of a prospect's decision to choose one company over another.

Design systems can be a sign of maturity for a team, and also provide insight into what it might be like to work at that company. Especially if you're trying to recruit directly for your design systems team, making the documentation public is an obvious way to attract people and set expectations.

OPENS YOU UP FOR INPUT FROM THE COMMUNITY

Making your documentation and/or your source-code public opens you up to external contributions, whether via simple feedback comments or direct contributions to code. This risks overhead, particularly if your system becomes popular, but the benefit is that you get feedback and insights from a wider range of people.

Primer is open-sourced on GitHub. We've been increasingly working in the open and sharing issues and pull requests so people can see how we work. We've noticed a gradual increase in contributions from the community and internal GitHub contributors. People generally feel proud to contribute to an open-source project that can benefit anyone.

Conclusion

There's a lot that goes into the rollout of your design system, and much of it will extend naturally from how you decide to build your system, who you're working with, and the best way to spread adoption inside your company.

Creating a perfect rollout strategy is less important than ensuring you're involving the right people, being clear about your end goals and how your strategy supports those, and documenting your plan methodically through each step.

Design systems are always evolving, and the way you share and encourage adoption of new iterations will evolve along the way as well. An intentional, thoughtful strategy with room to scale will go a long way to making sure adoption is treated as a crucial part of the building process.

Further reading

01. A successful Git branching model

02. The workshop and the storefront

03. Maintaining Design Systems

Chapter — 05

Expanding your design system

More than the sum of its parts

By Marco Suarez and Sophie Tahran

As mentioned in Chapter 1, early software programming was limited by the speed and capability of hardware. But over time, hardware became increasingly faster and cheaper, and the inefficiency of software became even more noticeable.

As product teams are expected to move faster and faster, speed and efficiency is an ever-growing concern. However, design systems enable teams to find speed, efficiency, and consistency through reusability. As you experience these benefits, other areas of inefficiency that were once unnoticeable become glaring.

Alignment is often one of those areas. Aligning team members, teams, and a company around things like direction, expectations, and quality is critical to moving and scaling quickly. Without alignment, friction increases and velocity slows—something product teams cannot afford.

A design system is a fantastic way to create alignment around interface design and implementation. But beyond components, teams can build alignment with vision, design principles, process, and voice and tone.

Vision

A vision statement moves everyone toward a common destination. Vision is your North Star. It's a reusable statement that gives context to your work to help your team stay on track and in sync. A vision statement declares what your team, product, or company is attempting to achieve and why it's worth achieving. It's comprehensive yet memorable, elevating yet attainable.

Vision creates clarity, making non-essentials easier to identify. Momentum is far easier to build once everyone knows where they're going and why.

A few years ago, the Starbucks product team was reorganizing and needed to clarify—for themselves and the entire company—their reason for existing. In a corporation as large as Starbucks, it's vital to clearly and succinctly express the purpose and value of your team.

"We create digital products that make our customers happy and our partners proud." This statement became their rallying cry. The vision grounded decision-making to improve the experience for customers and partners (baristas, managers, office employees) as well as achieve business goals.

> **PRO TIP —** Creating your vision statement
>
> To create your vision statement, describe what your team, application, or company should look like in two to five years. Does the culmination of your present work add up to that description? Is that description worth spending the next several years realizing? If not, adjust your aim. Maybe you need course-correcting or maybe you need to aim higher.

> "It helped us create a system that, we think, is clear and scalable across the various global markets we serve while still being unique."
>
> *Jason Stoff — STARBUCKS*

Informal vision statements for discrete projects also became useful. "Clarity first, then elegance if possible," was the guiding principle of a recent redesign.

> When we had a few design options for a given feature to discuss as a team, we referred back to our goal of clarity at the expense of all else, then worked to find a way (if possible) to make it elegant.
>
> *Jason Stoff — STARBUCKS*

Nick Grossman of Union Square Ventures gives several examples of how a North Star impacts company alignment

in his talk, "Purpose, Mission and Strategy." One example he gives is about Foursquare's product focus moving from the consumer app to the direction of being a location data platform that could power other applications. This was a massive shift in direction. It was the vision "making cities more delightful" that provided a much-needed North Star for employees to accept and embrace this new model. Grossman says, "It was a watershed moment for aligning the company around a new direction."

It's helpful to assess your progress as a team. At Etsy, we held retrospectives after every major launch. A retro is a meeting to celebrate the things that went well and identify the things that didn't for a particular launch. It's also an opportune time to check your alignment with your vision.

Design principles

Designers often operate with their own implicit set of standards to evaluate the quality of their work. But when teams grow, explicit standards become necessary to unify all with a shared language and a guide to evaluating the work. How do you define good design? How do you know what's essential and what's arbitrary? How do you know when something is ready to ship? How do you know if you're on the path to achieving your vision?

PRO TIP — Setting the groundwork for your principles

Creating your principles can seem like an overwhelming task. Before getting to work, establish a set of goals your principles must achieve. Here are a few examples:

- Should they describe the output or process?
- Should they be a sentence or a phrase?
- Who should they benefit?
- How should they be used?

Begin your work with a large amount of raw data collected from your team through discussions, survey, or interviews. Then, work through a process of combining, refining, and evaluating until you've arrived at a comprehensive and distilled set of principles. Etsy has a great article about their own principles creation process (https://dbtr.co/etsy-design-principles).

> Principles are important for design systems primarily for describing how it was created, and how the makers of it would intend it to be used.
>
> *Rich FULCHER — GOOGLE*

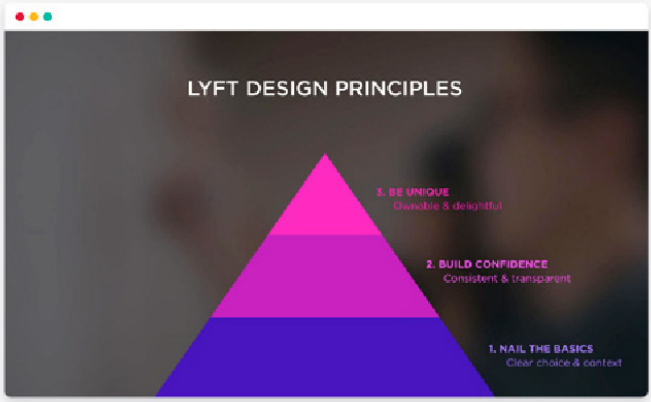


Figure 1: Lyft's design principles build upon each other.

A system of principles can provide the answers. Design principles act as a reusable standard for designers to measure their work. They replace subjective ideals with a shared understanding of what design must do for users. Just as guardrails keep you safe and on the road, design principles keep teams on the path to achieving their vision.

Posting your principles on the office walls, creating a computer desktop wallpaper, adding them to your design systems site, or printing them on notebooks are all great ways to keep them top of mind. Task your most senior designers with using them in every

design critique to ensure the new language is adopted. This new system of language will become a vital part of your design system because it creates alignment, enabling your design team to scale.

> Disagreements happened less and less because we were now aligned.
>
> *Steven Fabre — INVISION*

While designing InVision Studio, the design team realized far too much time was spent debating solutions. So a set of principles was drafted to help navigate the complexity of designing software. These principles were used to guide individual decision-making, help reach agreements faster as a team, and acted as a transcendent standard to create harmony and cohesion throughout the product. The principles were modified over time as Studio evolved and needs changed, but these principles aligned the team and helped them keep the velocity they needed.

Jesse Bennett-Chamberlain, Shopify

Listen online:
Principles And Why They're Important

Process

Providing a clearly defined process for how user experience problems are approached and solved builds alignment within product teams. This consistency helps to remove friction and build velocity.

> It is only in the repetition of the craft that he or she masters the art.
>
> *Samuel Avital — MIME*

Process guides designers through a series of clearly defined milestones, each with different objectives and deliverables. These milestones act as interdependent components that, when used together, improve the probability of arriving at optimal solutions.

A repeatable process offers several key benefits:

01. It provides clear expectations within each step, allowing you to focus solely on the tasks at hand without worrying about what to do next

02. It builds data within each step that can be referenced and used to inform future iterations or course-correct low performing launches

03. It creates an understanding of the roles and responsibilities of each team member involved—bringing the right people in at the right times, making everyone's involvement beneficial to the quality of the output

04. A repeatable process will ensure progress is smooth, efficient, and predictable while also improving the quality and consistency of your work.

At InVision, our process has 6 steps:

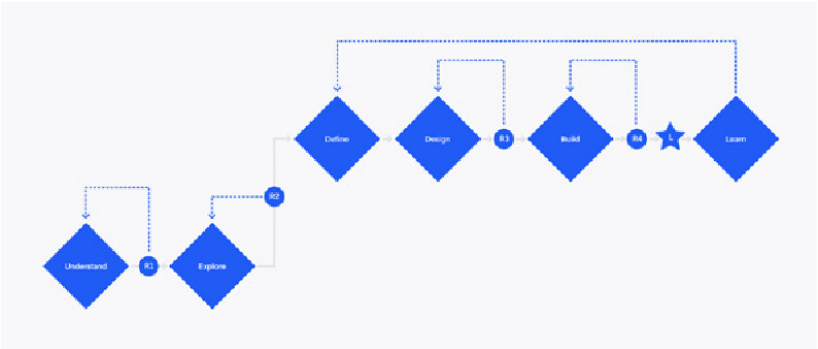


Figure 2: The InVision design process.

01. **Understand:** First, we use customer research and insights to gain a deep understanding of the problem space and identify how it aligns with our business goals. Product management leads this phase and works with the research team to conduct interviews, gather data, and perform competitive research.

02. **Explore:** The design team then ideates and explores possible solutions. They work with product and research teams to produce wireframes, core flows, and user journeys.

03. **Define:** Once a potential solution has been identified, the product team works to align everyone on what success looks like. The output of this work is a problem statement and defining measurements of success.

04. **Design:** Refining the solution is design team's responsibility. They work with the research, product, and engineering teams to develop core flows, prototypes, and identify tech requirements.

05. **Build:** Engineering translates the design and prototypes into reality. The product team orchestrates quality assurance, support documentation, and the sales and marketing teams.

06. **Learn:** At this point, we observe the effectiveness of our launched solution. The product team collaborates with the research, design, and engineering teams to gather insights to measure against our success metrics. We use these insights to inform our next steps and start the process over again.

Voice and tone

Sophie Tahran, InVision's UX Writer, weighs in on the value of writing guidelines—and how to create them.

Great writing is an essential part of great design, but even in-house UX writers aren't able to edit every single word in a product. That's where writing guidelines come into play. A source of truth promotes good communication, credibility, and consistency—no matter who's writing content.

Every writing guide should cover both voice and tone. While your voice generally stays the same (like a personality), your tone shifts according to the situation (like an attitude). Both are an essential part of communication—and you can't ship a product without effectively communicating it.

Because voice identifies who we are and defines our relationships, building writing guidelines early on is one of the best ways to gain credibility with your users. They'll begin to recognize you, and know that they can trust you.

Writing guidelines also help evolve your voice. Just as your personality matures over time, your voice will evolve as your company grows. Guidelines define what you should sound like right now, so when you do steer away from them, you'll know that you're doing so intentionally. ("I'll just throw an emoji in this subject line," turns into, "Hey, let's test how emoji perform and see if they're worth adding to our writing guidelines.") Without thoughtful writing guidelines, teams risk allowing their voice to be determined by patterns instead of users' needs, and end up making decisions solely based on what they've done in the past.

Jesse Bennett-Chamberlain, Shopify

Listen online: Voice and Tone in Polaris

Building writing guidelines

Start with an audit. Talk to people across (and outside) the company: designers, writers, support agents, co- founders, users, etc. Your goal is to get their impression of the company's personality. Consider not only conducting interviews, but also creating mood boards—visual collections of colors, people, places, and more—to represent more emotional, intangible qualities.

DEFINE THE FINDINGS

Have a core group of people, normally writers, narrow down the best words to describe what you found. Is your voice human, real, and bold? Or is it kind, experienced, and empowering? Take into consideration not only how you sound now, but also what you aspire to be and avoid ("InVision is never clickbait-y or pandering").

Tone can be more difficult to hammer down. It shifts according to the situation, so it's sometimes more helpful to provide direction than it is to identify characteristics. State your key priorities, like, "Always provide clarity," and, "Consider the user's emotional and mental state." This is also a good place to determine whether humor has a place in your writing, and where you fall on the formality spectrum.

Once you've identified the more nebulous aspects of your voice and tone, carry that same thinking into the tangible. What should your different channels look like? Do you use sentence case or title case?

You may find that you have a need for multiple guidelines— voice and tone within your design system, a company-wide style guide

focusing on words only, and brand guidelines that also encompass visual design.

GET BUY-IN

Potentially more important than building voice and tone guidelines is encouraging everyone to follow them. Anyone who works with content should have a chance to sign off.

FIND A HOME

Your guidelines need to be easily found and integrated. If you're building voice and tone into your design system, follow MailChimp and Shopify's lead in giving writing its own section, separate from components, principles, and layers. You can also go one step further by creating a separate list of copy patterns (like Mailchimp did) for grab-and-go words.

DEVELOP

Writing guidelines are living, breathing documents that require maintenance as your company grows. From the start, use an editable Paper or Google Doc—and better yet, schedule monthly check-ins.

Invest time in writing guidelines from the start. Most companies have more people representing their voice than you'd expect, from product designers to support agents. By having a source of truth—and creating it together—you can offer users a better experience.

> When you have guidelines that are in conflict, then you have to go back to the principles and decide which are more important.
>
> *Lori Kaplan — ATLASSIAN*

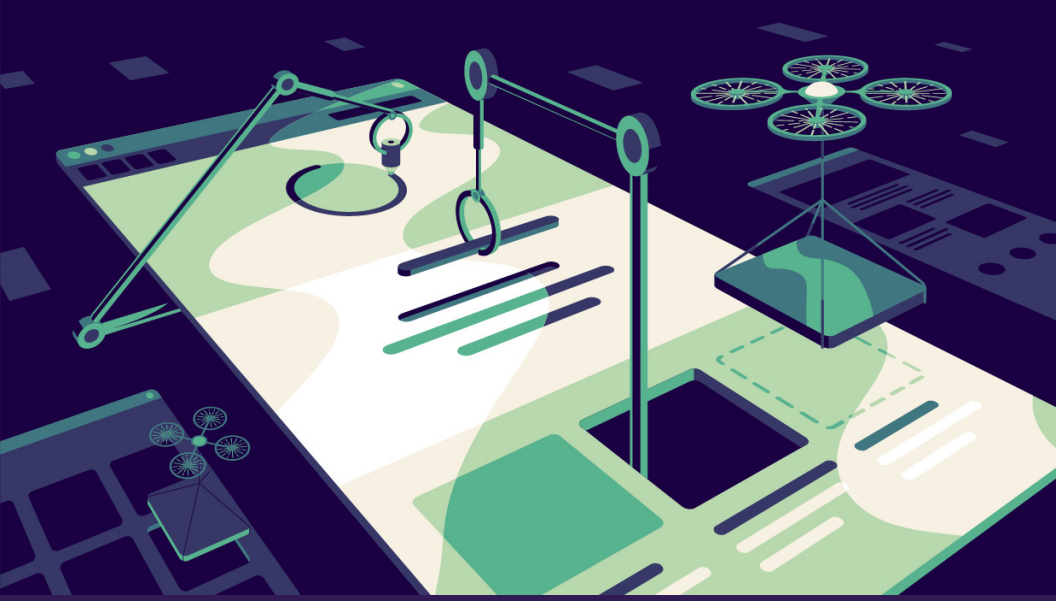
Conclusion

Just as design is far more than the sum of its parts, your design system can serve as far more than a components-only guide. Vision, principles, process, and voice and tone expand your design system into additional areas where reusability can also increase speed and efficiency.

Design systems have changed the way we design and build applications—so much so that we must look at the future of design through the lens of systems. In a future where design systems gain wide adoption, design could take on many new, exciting forms.

Further reading

01. The Importance of Product Vision to Product Leaders
02. 7 Problems Growing Design Teams Face
03. IBM Design Thinking
04. Design Principles Behind Great Products
05. Creating Etsy's Design Principles
06. Design Doesn't Scale
07. A Matter of Principle
08. A Study of the Design Process
09. UX Tools for Every Step in Your Design Process
10. Nicely Said
11. How Snow White Helped Airbnb's Mobile Mission
12. MailChimp's VoiceandTone.com



Chapter — 06

The future of design systems

To infinity and beyond

By Roy Stanfield

I find it exciting to see design systems empower design teams to scale and consistently produce solid products, but I know we're only scratching the surface of our potential. There's so much more we can accomplish.

At Airbnb, we've been pondering how we might push our design system in new directions, and we're inspired by design- forward companies that share our desire to craft the future of design systems.

In this chapter, I want to introduce you to a new way of thinking about design systems. Design systems can transcend the walls of a single company to exist as shared standards and customizable tooling with help from the open source community, which can accelerate development and eliminate the need to start systems from scratch. If we're really bold, we could be creating adaptive, intelligent systems that are context-aware and compose themselves—reducing our workload and ultimately unlocking AI-powered design.

But, I'm getting ahead of myself. Let's start with nuts and bolts.

Building a common foundation

A designer's familiarity with the concept of a design system is based on the systems they've encountered, what platforms they've been tasked to support, and where they've worked.

Books like this one help us converge on a high-level definition for design systems, but a more rigorous definition will ensure the utility and flexibility of our work.

Finding a standard that supports shared goals will involve decoupling a design system from its implementation, cataloging common UI and associated states, and more strictly defining design primitives and components. This could then be expressed in a file format that's able to define a component or design system more completely.

> Imagine as if where I built this thing, gravity works one way, and when I install it in your office gravity works another way.
>
> *Tim Sheiner — SALESFORCE*

An example of differing goals

Currently, existing systems reflect the specific needs of the companies that created them. Since each company is building an entirely independent system, design system development starts from scratch—possibly with help from a web toolkit like Bootstrap, relying on the internal knowledge of the team, and only focusing on top-level needs. As a result, even the best systems contain flaws and lack the necessary tooling to speed development and track results. And if a company's priorities shift, its design system must shift, causing another section of the design system to be built in the same limited way.

For example, one of the reasons Airbnb created DLS was to minimize and sync differences in UI between our supported Android, iOS, and web platforms. In The Way We Build, VP of Design Alex Schleifer writes, "Universal and Unified define the system's approach we apply when defining patterns. Is it part of a greater whole? Does it work across devices?"

In our idealized vision, a mockup easily ports between platforms—creating a better design and development experience. This cross-platform UI would give Airbnb guests and hosts alike the same end-product experience as if they jumped between mobile app and desktop web.

In contrast, supporting multiple device platforms was initially less of a concern at Etsy, where the main priority was to scale its web platform. During my time there, the team built the web toolkit with the core website as its primary focus. Later, Etsy expanded upon its toolkit to support different branding elements for other internal web initiatives. Karyn Campbell describes what it was like modifying the Etsy design system while making Etsy Studio. "While we made a conscious decision to depart in some instances with the etsy.com UI in order to birth this new brand, we also retained many underlying components that our design systems team had created."

A priority at Airbnb was having the same functional and visual voice across platforms. A priority at Etsy was to support multiple web products with varying brand initiatives. Both were valid needs. A shared standard for design systems would need to ensure a solid foundation so that both these and other real- world priorities could more easily be achieved.

Gathering examples of different design system priorities will help create a checklist to make sure your design system standards address real concerns. Any company adopting these suggested standards could be assured their design system development focuses on immediate company needs, as well as adhering to standards they plug in to a growing body of open source code and tools that support most operational transformations that could be encountered.

But what should be in the design system standard?

Imagine a tool that can specify which design primitives, (e.g., fonts, spacing, color—more on that below), components (and their states), platforms, and what documentation and testing are needed to have a fully formed design system. The tool would also allow the designer to specify which components were not yet needed and which platforms could be added later. With this tool, a designer would have a framework stating what aspects of the design system were completed or outstanding. A product manager could export documentation, and a developer could easily export UI and UI tests—no longer needing to translate UI from Sketch to code, or from web code to native implementation.

If created today, not only would this tool provide industry- wide savings, but it would start to standardize the low-level definition of a design system. Working backward, let's now imagine what kind of definitions such a tool would need in order to exist.

First, decouple the design system from any specific implementation. We're not creating React components (nor other web implementations), nor Android UI, iOS UI, or even Sketch files. Instead, our system is an abstraction that can be deployed to any target implementation. We're going to need a file format to describe this abstracted design system. The exported format could be rendered into views by open source modules specific to each target implementation.

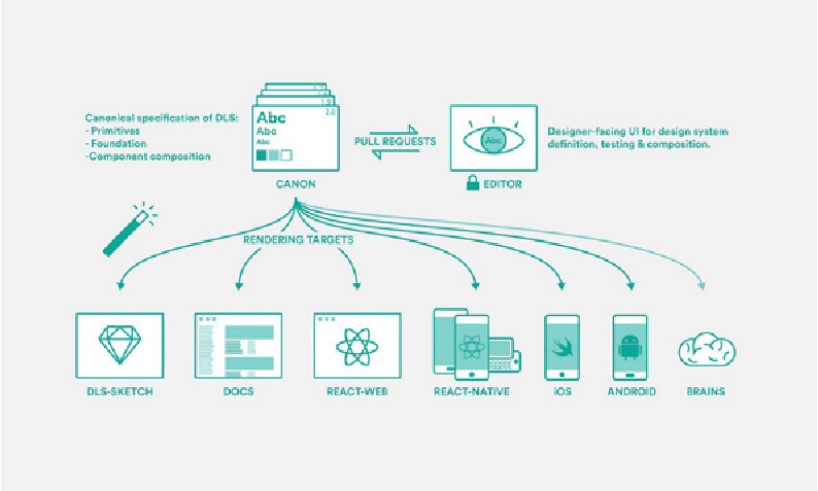


Figure 1: Single source of truth concept proposed by the Design Tools team at Airbnb, originally illustrated by Jon Gold.

Next, codify the definitions of design primitives and components so they are fully expressed in the design system format. Dang, .dsf extension is already in use! Guess we'll have to settle for the .dang file extension!

Design primitives are the building blocks of a UI. These include specific predefined colors, fonts, spacings, and more. They are foundational visual elements that can be combined into components. Changing primitives is echoed throughout a given design system's components, and doing so changes the overall feeling of a brand. Additionally, what are components? We'll also need to codify those. Components are mostly views composed of design primitives and smaller components whose minimal internal logic is mapped exclusively to state and state change. Benjamin Wilkins, Design Lead on the Design Tools team at Airbnb, describes

the difference between primitives and components in minute 7:00 of his talk, “Thinking in Symbols for Universal Design.”

Next, we’ll need to catalog all of the common UI components in use today. Just as a typeface may have its unique take on the letter “A” (the letter’s visual appearance may vary between typefaces while its meaning is maintained), a .dang file would have a text input component that varies in visual representation but not functionality. The catalog will need to group components with their accompanying states (selected, focus, on-tap, error, etc.) and detail interactions to distinguish between mobile, desktop, and TV UI.

What are the benefits of this catalog? To start, functional tests for common components could be easily automated through contributions from the open source community. In many cases, UI engineers would no longer need to write their own tests. The cataloged components would also enable a marketplace of boilerplate design systems that can be installed interchangeably, and against which custom UI can be built and substituted. This means bootstrapping the creation of every design system is no longer necessary.

Lastly, we need to allow for the evolution, growth, and extensibility of design systems built upon the shared standard. Just because we’re aware of which components are needed today doesn’t mean that we’re able to predict all the elements needed for future innovation. A process for modifying existing components or creating wholly new ones is in order. Thoughtfully standardizing our collective knowledge will produce a more consistent user experience, accelerate development, decrease investment needed from individual companies, and enable open source and collective development of next-generation design tools that conform to shared conventions.

Creating a single source of truth

The elements that make up a design system—principles, UI components, patterns, and documentation—create the human-computer interaction layer for our apps. Product designers and system designers are directly responsible for this layer, and therefore should own the design system and its representation in the codebase.

There are two hurdles to achieving a single source of truth. First, our current design tools are inadequate. Most only allow us to produce images of UI and prevent designers from achieving product level fidelity. Second, if the implementation of a design system is spread across multiple repositories (Android, iOS, React Native, React, etc.), collected in a Sketch file, and documented on a website, then there really is no single codebase to represent a truthful account of the system.

Lacking a single source of truth, the design system—spread out over multiple codebases—becomes an amalgam of sources that easily fall out of sync.

Our tools

Designers use tools like Sketch, Illustrator, or Photoshop to draw pictures of UIs, yet these are actually just representations of interactive components that look different, behave differently, and contain different data depending upon the state of the app at a given time. As Colm Tuite notes in the article "Design tools are running out of track. Here's how we can fix them," Think of the number of simple interactions which are commonplace in almost

all of our products yet cannot be communicated through our design tools."

Tuite then mentions interactions and states such as hovering over a button, focusing an input, checking a checkbox, and identifying scroll areas. He points out that our design tools aren't prompting designers to think with product level fidelity, and so a designer's work is usually missing some of its most important details.

> "For each change or addition to the system, a cascade of work is created. Documentation has to be updated, each of our apps' code has to be changed (in Swift, Java, and JavaScript), and Sketch templates have to be redrawn. These efforts have to be coordinated, and each source must always be in sync with the others. Code is relatively easy to tie together, and we already have an infrastructure that allows for version control and continuous integration of our applications. Until now though, maintaining Sketch templates relied on manually-intensive human touch points."
>
> 
>
> **Jon Gold**
> Airbnb

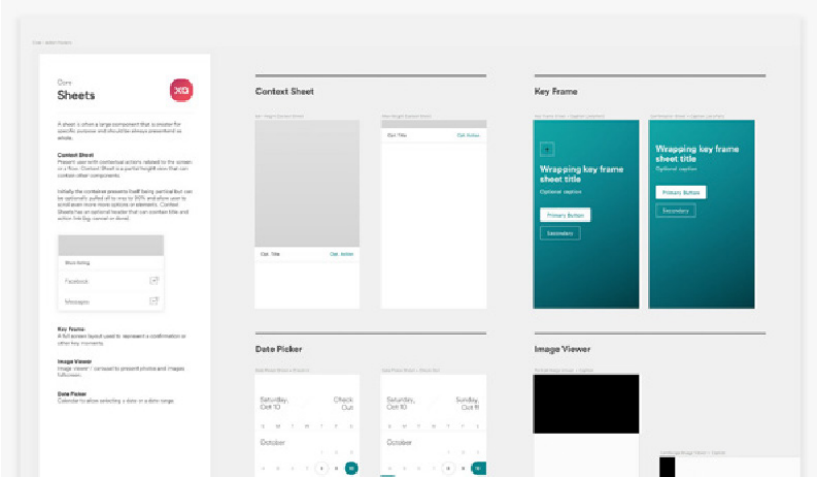


Figure 2: Detail of the Airbnb DLS system Sketch file. DLS, UX Platforms, Infrastructure. © Airbnb, Inc.

Sketches of apps are then handed off to developers who have to translate them into working UI. Between the designer and the 4 developers that it takes to convert the design into Android, iOS, React Native, and React, it takes 5 different members of the team begin to bring the design up to product- level fidelity. Since the original sketch was missing details about state and interactions, a back-and-forth conversation between designer and multiple developers is needed to make the designs production-ready. And because the implementation is coded by 4 different humans, it's likely that unwanted variation creeps into each implementation.

For similar reasons, many designers have focused on sharpening their coding skills for at least 1 platform. There are many advantages to this, but if you're a systems designer creating components for cross-platform use, coding those components for a single implementation is not yet enough.

Multiple implementations

The attempt to reach a single source of truth is further complicated when working on cross-platform design systems. Jon Gold identifies places where Airbnb's DLS workflow could be improved in "Painting with Code."

At Airbnb, Gold has taken some exciting first steps toward solving this problem with his project React Sketch.app. With this tool, layered Sketch files can be generated from the React codebase. This means components in the Airbnb React repo, which already have product-level fidelity, can be populated with real data and rendered to a Sketch file. It's another reward for those adventurous designers willing to learn React!

It's also a touchstone technology, pointing us toward understanding mockups not as the source of truth, but instead as another target for automated output. With these generated files, we get a clear picture of which components are sitting in the repo. Best of all, a product designer relying on Sketch does not have to change their workflow and can use more accurate files (generated by the codebase, not by hand) to compose their work. At last, we can have confidence in how the components look at product-level fidelity and in how these components behave with real data.

> The key in making great and growable systems is much more to design how its modules communicate rather than what their internal properties and behaviors should be.
>
> 
>
> **ALAN KAY**
> LEGEND

React Sketch.app is great because it syncs the React and React Native repos and Airbnb's design system Sketch files. But what about the Android and iOS implementations? How can designers make sure these are in sync? React Sketch.app points the way, but we'll have to go further.

Here, we can learn from tools and WYSIWYGs of the recent past like Dreamweaver and Interface Builder. Tools in this category allow users to combine elements of UI, hook the UI up to data and interaction, and then export deployable code. Unfortunately, these software produce code that's not maintainable by humans, and so few companies use them as an official part of their process. Did these tools promise too much and deliver too little?

Luckily, design system components are simple views with minimal logic. Learn more about views and MVC architecture here and here. Unlike the promise of tools like Dreamweaver and Interface Builder, components are easily exported view files that developer partners can incorporate into existing workflows.

Solving two problems with one tool

To create a designer-controlled source of truth, we'll need next-generation design system tools to enable the composition of components and to automate the output of the design system to any number of target clients (codebases, vector files, and documentation sites).

As outlined in the section above, if we standardize an export format that contains production-level fidelity for components, then other interpreter modules can be freely built to compile a single component file into all the various flavors of production code. Then, varied implementations of any given component

would flow from a single source of truth and would plug right into existing workflows—saving resource-intensive and error- prone human interpretation of mocked UI. Product designers and design system designers would finally control the UI they have always been responsible for.

At Airbnb, we're tackling this challenge in a variety of ways, including with Lona, our tool for defining design systems and using them to generate cross-platform UI code, Sketch files, images, and other artifacts. A highly experimental prototype, Lona represents our exploratory approach to discovering what's next for design at scale. Taking a research-based approach to the future of design systems encourages experimentation and collaboration.

Intelligent systems compose themselves

If you've not heard of Alan Kay, make sure to look him up. He's credited with inventing the graphical user interface, Object Oriented Programming, and—with his concept of the Dynabook—even the tablet. Given that design systems fit firmly inside of the world that Kay and his peers built, it's worth listening to him.

Once the magic of design system standards enables both private and open source development for cross-platform use, design systems will most likely reach new heights of functionality and popularity. Whereas siloed development requires talented people to reinvent the design system anew for each employer, standards-based design systems might easily plug into a passionate community that will add capabilities to our nascent tools.

Taking inspiration from Kay and the PARC maxim—"The best way to predict the future is to invent it"—I'll sketch out a stretch goal for the future of design systems and tools in the paragraphs that follow.

Virtuous cycle—plugging into the feedback loop

Often, our design systems contain components that aren't tracked in any special way. Without special consideration and considerable development effort, the design systems team is blind to the usage statistics and performance metrics directly associated with the system. To gain insight into design systems, teams must manually track which components each product team uses, and then try to glean usability information from product team UX research and performance metrics.

At Airbnb, we see data from the large proportion of our native apps, which now use design system components. Connecting these stats to user metrics lets us know which components might be underperforming and deserve special attention.

Similar metrics also have an internal benefit. Dashboards containing usage data aid the internal perception of the system among product designers and engineers alike. It's easy to imagine that tracking the use of a design system can go beyond recording end-user impressions (views) and interactions. Even analyzing the usage of documentation, design files, and other internal tools can lead to insights that better enable product teams.

But why should such insights be limited to just a few companies with the resources to create similar tooling? A module could be built for just this purpose. It would track where canon components

are used in product and then apply a flag to that component. These flagged components could be polled, and usage stats could come straight back to the design systems team. The insights would then be used to improve internal tools and the system itself.

Layout aware

Another promising direction would be for our components to become somewhat layout aware, and then declare their intended use to both the system at large as well as sibling components. Given this new super power, design system components could now communicate where they should generally be used and then share information about what types of data they generally contain. Even a simplified implementation of this functionality would help designers instantly swap abstracted data sets to, for instance, see how translation might affect the layout of a particular component given a longer language like German or a right-to-left language like Arabic.

This awareness might even go a step further with a given component stating the kinds of screens or interactions preceded by its own display, and again which kinds of interactions or screens should follow upon user interaction. This awareness would enable predictive assembly.

Predictive assembly—a pattern tool

Because each component now sends messages about where it's typically used and what type of data it usually contains, we should also be able to see existing components predictively assembled into screens. A future tool might allow the product designer to view a particular component, click a randomize button, and see

the component in situ with other components arranged in an order that more or less semantically makes sense.

By allowing designers to up-vote particularly useful component groupings (or patterns), a layout aware tool featuring predictive assembly—perhaps better described as a design system pattern tool—could become an instrumental way in which product designers quickly discover repeatable patterns in the system. With just a single click, a designer would be able to compose an effective solution made of many components and export the basic design for any project. Moreover, with the advent of standardization, design systems could become plug-and-play with compositions composed by predictive assembly, allowing for a quick way to assess the integrity of a given design system.

Predictive assembly would enable the product manager, developer, or product designer to provide a set of data and ask for a menu of pre-assembled screens that display the data. We could then choose the best option, only correcting the layout if needed.

It's worth noting that if we get particularly good at predictive assembly and consult the right minds to take it further, we will soon find ourselves in a world where machine learning provides artificial intelligence for plug-and-play design systems. Among many other things, we might witness the birth of a world where phone users could eliminate branded apps, preferring a locally installed design system that could spin up UI without needing to load templates over the network.

As designers, we continue to rely on intuition, performance metrics, and user research to create products. If a large percentage of our users can complete a task and proceed to the next stage in the flow, then we generally call the design successful. We use templating languages to add options to a design so even more users

reach their goals. But by connecting design systems to artificial intelligence, we could achieve a step change in customizable UI. Instead of designing products that work for most users, systems design and AI will apply specific solutions for specific individuals.

Designers should prepare now by discovering a bit about machine learning, reading about the breakthroughs enabling AI today, and finally, thinking about how ML could impact design tools.

> There is this tremendous value to writing things down in a way that others can understand it.
>
> *Richard FULCHER — GOOGLE*

Conclusion

As digital product designers, we're asked to ride the waves of change in our industry. People just like us helped small development teams create apps for desktop computing in the 1980s. In the late 1990s, other designers formed a coalition that pressured large software companies to unite front-end web development through the web standards project. In the mid-2000s, we focused on communication and user-generated content while developing patterns for creating web apps, and just a little later, all our work was available in mobile form factors like the iPhone. Now, after nearly 4 decades of designed computing, design systems and AI are shaking hands—presenting us with new opportunities for innovation at scale.

Every designer will now be asked how they might change custom, one-off design solutions into reusable components that grow the system. We'll have to think more holistically, and work collectively to develop and learn a new set of tools.

How can we work together to improve our workflows and better scale design? What should be in a design system standard, and what kinds of tools could be built if such a standard existed? How will design systems and AI combine to create extremely customized interactions for end users? There are many questions, and each answer points to a possible future. What I love about design is that it enables each of us to explore the ideas we feel may have the biggest impact, and project our ideas outward and into the future. Innovation is certain.

Further reading

01. Thinking in Symbols for Universal Design

02. How to Improve UX with Machine Lea[r]ning: A Wonderful Lesson from Netflix and More

03. Algorithm-Driven Design: How Artificial Intelligence is Changing Design

04. CreativeAI: On the Democratisation & Escalation of Creativity — Chapter 01

05. Project Phoebe

Chapter — 07

Appendix

More resources

There are a number of great resources available for diving further into the world of design systems. The folks at StyleGuides.io have not only assembled a comprehensive list of example style guides and design systems, but have also included relevant articles, books, podcasts , talks, and tools. Alex Pate has assembled a list of Awesome design systems on Github. And DesignGuidelines.co has a list of examples, as well as readings and tools.

From these resources, we've curated a list of design systems and style guides you may find helpful:

01. Airbnb
02. Atlassian
03. BBC, Gel
04. Buzzfeed, Solid
05. FutureLearn
06. Github, Primer
07. Google, Material
08. IBM, Carbon
09. IBM, Design Language
10. Lonely Planet, Rizzo
11. MailChimp, Patterns
12. MailChimp, Voice & Tone
13. Microsoft, Fabric
14. Microsoft, Fluent Design
15. Nordnet
16. Salesforce, Lightning
17. SAP, Fiori
18. Shopify, Polaris
19. Ubuntu
20. WeWork, Plasma
21. Yelp

About the Authors

MARCO SUAREZ

Marco Suarez has helped shape the visual language, design systems, and customer experiences of companies like Etsy and Mailchimp. His work extends beyond design to influence internal processes that scale across organizations. He is also the co-founder of Methodical Coffee.

JINA ANNE

Jina Anne is a leading advocate for design systems, having contributed to the evolution of systems at Amazon, Salesforce, Asana, and Google. Her work has shaped foundational concepts such as design tokens. She is also the co-founder of the Clarity Conference.

KATIE SYLOR-MILLER

Katie Sylor-Miller is Principal Engineer and Web Tech Lead at Square, and was formerly Frontend Architect at Etsy. She specializes in design systems, web performance, accessibility, and frontend infrastructure, and is passionate about the craft of Staff Engineering. As an invited expert on the W3C Web Performance Working Group, she helps guide browser development to improve Web Performance APIs.

DIANA MOUNTER

Diana Mounter is Head of Design at GitHub. Her approach blends systems thinking, inclusive design, and the perspective of code as a material in the design process.

ROY STANFIELD

Roy Stanfield has played a pivotal role in developing design systems at Airbnb and Etsy. He currently serves as Principal Product Designer at Shopify.

Design Better

About Design Better

Hosted by **Eli Woolery** and **Aarron Walter**, *Design Better* explores creativity at the intersection of design and technology. Go deeper into design at **designbetter.com**