Experience Design Notes: PRK 1-24-18

I find it really fascinating when I see some new blog post announcing a company’s new design system made in Sketch. And I cringe a little bit on the inside, because that project is going run into such problems when that design system is implemented in code, and there’s suddenly two sources of truth: Sketch and the code implementation. Then you’re suddenly faced with a situation where your teams have to do double work to keep the design system up to date.

This is why I was super excited to see Jon Gold at AirBnb release React Sketch App, which is one of the first projects I’ve seen that actually try to tackle this problem. It’s a library that allows you to automatically generate Sketch symbols based on React components, and auto-fill them with dynamic data from your code. So you can have a single source of truth, but still allow designers to make page designs in Sketch. The lesson here is that it’s really not that hard to whip up a design system, but it’s very hard to have it deliver consistency over time, and that only happens if you provide value for everyone.

There are 2 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.

[Books](https://www.designbetter.co/books)[Podcast](https://www.designbetter.co/podcast)[Workshops](https://www.designbetter.co/workshops)[SubscribeSubscribe](https://www.designbetter.co/design-systems-handbook/subscribe)

**Design Systems Handbook**

[01[](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)

**[Introducing design systems](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)**

**[The power of scale](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)**

[02[](https://www.designbetter.co/design-systems-handbook/designing-design-system)](https://www.designbetter.co/design-systems-handbook/designing-design-system)

**[Designing your design system](https://www.designbetter.co/design-systems-handbook/designing-design-system)**

**[Step by step](https://www.designbetter.co/design-systems-handbook/designing-design-system)**

[03[](https://www.designbetter.co/design-systems-handbook/building-design-system)](https://www.designbetter.co/design-systems-handbook/building-design-system)

**[Building your design system](https://www.designbetter.co/design-systems-handbook/building-design-system)**

**[A strong foundation](https://www.designbetter.co/design-systems-handbook/building-design-system)**

[04[](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)

**[Putting your design system into practice](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)**

**[Better together](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)**

[05[](https://www.designbetter.co/design-systems-handbook/expanding-design-system)](https://www.designbetter.co/design-systems-handbook/expanding-design-system)

**[Expanding your design system](https://www.designbetter.co/design-systems-handbook/expanding-design-system)**

**[More than the sum of its parts](https://www.designbetter.co/design-systems-handbook/expanding-design-system)**

[06[](https://www.designbetter.co/design-systems-handbook/future-design-systems)](https://www.designbetter.co/design-systems-handbook/future-design-systems)

**[The future of design systems](https://www.designbetter.co/design-systems-handbook/future-design-systems)**

**[To infinity and beyond](https://www.designbetter.co/design-systems-handbook/future-design-systems)**

[07[](https://www.designbetter.co/design-systems-handbook/appendix)](https://www.designbetter.co/design-systems-handbook/appendix)

**[Appendix](https://www.designbetter.co/design-systems-handbook/appendix)**

**[More resources](https://www.designbetter.co/design-systems-handbook/appendix)**

**DesignBetter.Co**

[Books](https://www.designbetter.co/books)[Podcast](https://www.designbetter.co/podcast)[Workshops](https://www.designbetter.co/workshops)[Credits](javascript:;)[Keep Learning](https://www.designbetter.co/design-systems-handbook/subscribe)

**InVision presents**

**Design Systems Handbook**

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.**

Go to Chapter01Go to Chapter

**Introducing design systems**

**The power of scale**

Repetition and reusability make scale possible. InVision’s Marco Suarez takes you back to the foundations of component-based design and the many ways design systems enable scaling. He also addresses a few common misconceptions about design systems.

Go to Chapter

Go to Chapter02Go to Chapter

**Designing your design system**

**Step by step**

Starting can be the hardest step. Design systems expert Jina Anne walks through what to consider as you start out. Involve the right people, find the right model, add the right pieces—and you’ll be well on your way to establishing a successful design system.

Go to Chapter

Go to Chapter03Go to Chapter

**Building your design system**

**A strong foundation**

Building a flexible, maintainable, stable, scalable design system creates a strong base for your product design team. Etsy’s Katie Sylor-Miller shares foundational knowledge learned building design systems across cross-functional teams.

Go to Chapter

Go to Chapter04Go to Chapter

**Putting your design system into practice**

**Better together**

Encouraging adoption of your design system forges connections and creates champions across teams. GitHub’s Diana Mounter lays out practical strategies for sharing design systems and growing adoption, including examples from her experience.

Go to Chapter

Go to Chapter05Go to Chapter

**Expanding your design system**

**More than the sum of its parts**

Components form the core of a design system, but teams can align around much more than components. Marco Suarez outlines how teams can round out their systems with vision statements, design principles, voice and tone guides, and more.

Go to Chapter

Go to Chapter06Go to Chapter

**The future of design systems**

**To infinity and beyond**

Design systems play a huge role in shaping our present, but we may only be scratching the surface. Airbnb’s Roy Stanfield explores a bold future of intelligent, adaptive, context-aware systems that could further accelerate design’s possibilities.

Go to Chapter

Go to Chapter07Go to Chapter

**Appendix**

**More resources**

Design systems continue to evolve, and every company’s system can offer something to our collective knowledge and understanding. We’ve curated a list of publicly available design systems and resources to help you continue your exploration.

Go to Chapter

01020304050607

**Design Systems Handbook**

[01[](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)

**[Introducing design systems](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)**

**[The power of scale](https://www.designbetter.co/design-systems-handbook/introducing-design-systems)**

[02[](https://www.designbetter.co/design-systems-handbook/designing-design-system)](https://www.designbetter.co/design-systems-handbook/designing-design-system)

**[Designing your design system](https://www.designbetter.co/design-systems-handbook/designing-design-system)**

**[Step by step](https://www.designbetter.co/design-systems-handbook/designing-design-system)**

[03[](https://www.designbetter.co/design-systems-handbook/building-design-system)](https://www.designbetter.co/design-systems-handbook/building-design-system)

**[Building your design system](https://www.designbetter.co/design-systems-handbook/building-design-system)**

**[A strong foundation](https://www.designbetter.co/design-systems-handbook/building-design-system)**

[04[](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)

**[Putting your design system into practice](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)**

**[Better together](https://www.designbetter.co/design-systems-handbook/putting-design-system-practice)**

[05[](https://www.designbetter.co/design-systems-handbook/expanding-design-system)](https://www.designbetter.co/design-systems-handbook/expanding-design-system)

**[Expanding your design system](https://www.designbetter.co/design-systems-handbook/expanding-design-system)**

**[More than the sum of its parts](https://www.designbetter.co/design-systems-handbook/expanding-design-system)**

[06[](https://www.designbetter.co/design-systems-handbook/future-design-systems)](https://www.designbetter.co/design-systems-handbook/future-design-systems)

**[The future of design systems](https://www.designbetter.co/design-systems-handbook/future-design-systems)**

**[To infinity and beyond](https://www.designbetter.co/design-systems-handbook/future-design-systems)**

[07[](https://www.designbetter.co/design-systems-handbook/appendix)](https://www.designbetter.co/design-systems-handbook/appendix)

**[Appendix](https://www.designbetter.co/design-systems-handbook/appendix)**

**[More resources](https://www.designbetter.co/design-systems-handbook/appendix)**

**Credits**



**Marco Suarez**

**Product Designer, InVision**



**Katie Sylor-Miller**

**Staff Software Engineer, Etsy**



**Roy Stanfield**

**Design Lead, Airbnb**



**Diana Mounter**

**Design Systems Manager, GitHub**



**Jina Anne**

**Design Systems Pioneer, Independent**

**Acknowledgments**

[Nathan Curtis](https://twitter.com/nathanacurtis), technical editor  
[Claire Karjalainen](https://twitter.com/clairekarj), copy editor  
[Aarron Walter](http://twitter.com/aarron), acquisitions editor  
[Eli Woolery](https://twitter.com/ewoolery), video and audio interviews

**WE WANT TO THANK EVERYONE WHO SPOKE TO US DIRECTLY, AND SHARED THEIR INSIGHTS FOR DESIGNBETTER.CO**

Devon Beard — Twitter, Jesse Bennett-Chamberlain— Shopify, David Carmona — LinkedIn, Jessica Clark — LinkedIn, Rachel Cohen — LinkedIn, Kalee Dankner — LinkedIn, Ashlie Ford — Twitter, Rich Fulcher — Google, Lori Kaplan — Atlassian, Lisamarie Kelly — LinkedIn, Tina Koyama — Twitter, Alicia Sedlock, Tim Sheiner — Salesforce, Nate Whitson — LinkedIn

The content of this website is the property of InVision, and may be reproduced with permission. Please [contact us](https://support.invisionapp.com/hc/en-us/requests/new) for details.

**© InVision 2018**

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](https://en.wikipedia.org/wiki/Software_crisis).”

In 1968, at the [NATO conference](http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF) 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.

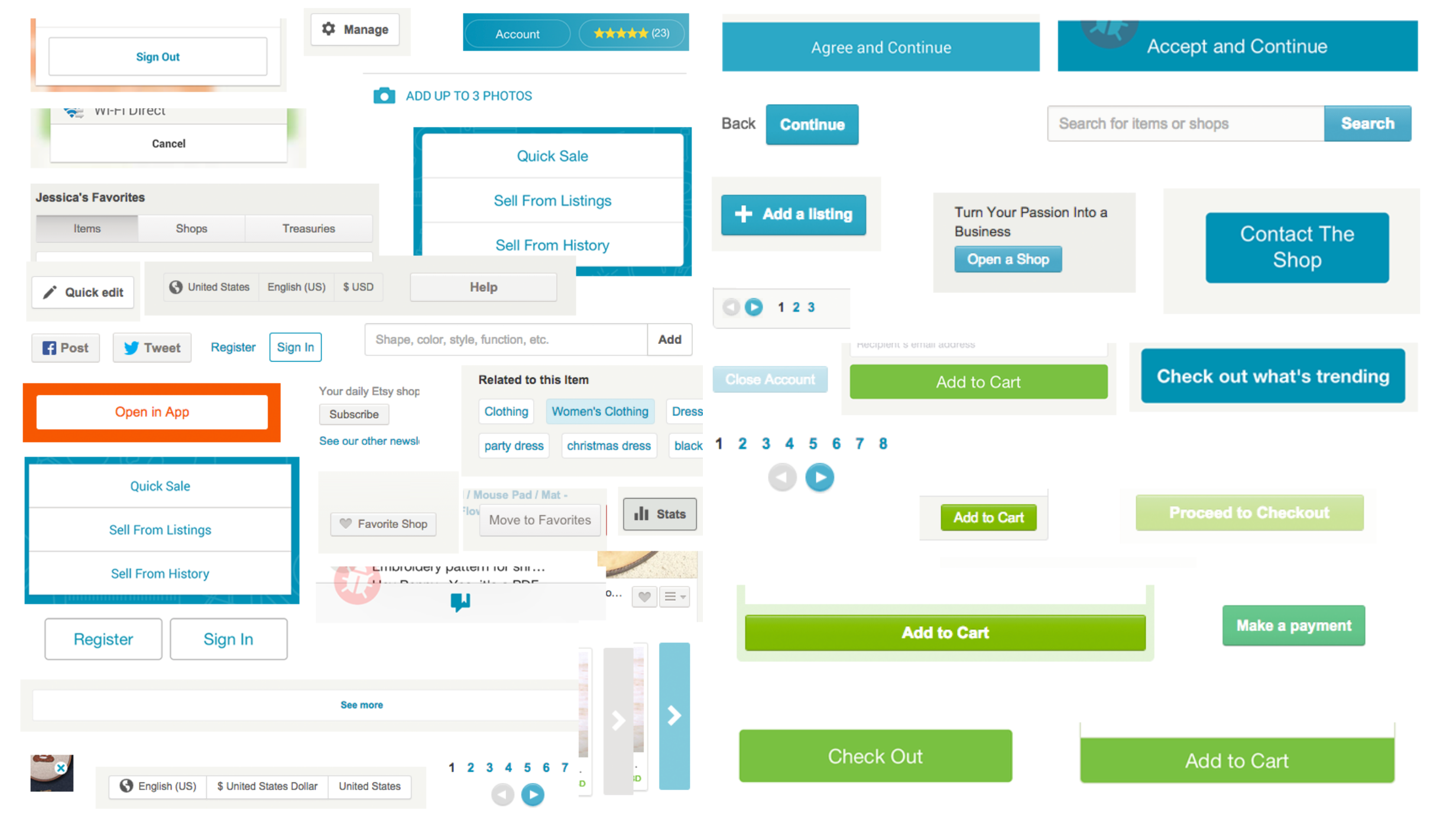


Figure 1. A UI audit collects the many permutations of simple UI elements to illustrate how deep in design debt your team is.

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

1. Hire more people
2. Design faster
3. 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 2 concepts with individual merit, making something more powerful than its separate parts.

**Standards**

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**

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.

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.

**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](https://material.io/guidelines/style/color.html).

[Play](javascript:;)

**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***

Lori Kaplan, co-author of the pioneering Macintosh Human Interface Guidelines (HIG), tells the story of their creation and impact on designers and developers.

**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.](https://www.designbetter.co/design-systems-handbook/building-design-system/)

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.

**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](https://polaris.shopify.com/)—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.

**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.

[[](https://www.invisionapp.com/blog/announcing-invision-design-system-manager)](https://www.invisionapp.com/blog/announcing-invision-design-system-manager)

[**InVision Design System Manager**](https://www.invisionapp.com/blog/announcing-invision-design-system-manager)

Create better products, at scale—with consistent experiences on every screen

[GET EARLY ACCESS](https://www.invisionapp.com/blog/announcing-invision-design-system-manager)

**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 making those improvements available to everyone.

**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.

**Conclusion**

[Play](javascript:;)

**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***

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.

[[](https://www.designbetter.co/subscribe)](https://www.designbetter.co/subscribe)

[**Get insider access to DesignBetter.Co**](https://www.designbetter.co/subscribe)

Be the first to get content from design innovators like Automattic’s John Maeda and more

[GET ON THE LIST](https://www.designbetter.co/subscribe)

**Further reading**

* [Software Crisis](https://en.wikipedia.org/wiki/Software_crisis)
* [Component-based Software Engineering](https://en.wikipedia.org/wiki/Component-based_software_engineering)
* [The Way We Build](https://airbnb.design/the-way-we-build/)
* [Designed for Growth](https://medium.com/etsy-design/designed-for-growth-cdc2a01a8a8a)
* [Selling a Design System before asking for buy-in](https://uxdesign.cc/selling-a-design-system-before-asking-for-buy-in-eeb45e88f66a)
* [The Design of Everyday Things](https://www.amazon.com/Design-Everyday-Things-Revised-Expanded/dp/0465050654/ref=sr_1_1?ie=UTF8&qid=1509400743&sr=8-1&keywords=design+of+everyday+things)
* [What is a Design Language… really?](https://medium.com/thinking-design/what-is-a-design-language-really-cd1ef87be793)
* [Things you could be doing instead of designing and building that card component for the umpteenth time](http://bradfrost.com/blog/post/things-you-could-be-doing-instead-of-designing-building-that-card-component-for-the-umpteenth-time/)
* [Website Style Guide Resources](http://styleguides.io/)
* [Making Material Design](https://www.youtube.com/watch?v=rrT6v5sOwJg)
* [Material Design](https://material.io/guidelines/)
* [Shopify Polaris](https://polaris.shopify.com/)
* [Starting a Design System](https://medium.com/eightshapes-llc/starting-a-design-system-6b909a578325)

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**

[Play](javascript:;)

**Imagine as if where I built this thing, gravity works one way, and when I install it in your office gravity works in another way**

***Tim Sheiner***

***Salesforce***

In his 1998 essay [*Interchangeable Parts*](https://medium.com/@tsheiner/interchangeable-parts-d2ca009eaa5b), Tim Sheiner examined the historical standardization of parts for physical products, and predicted how this will play out for digital products.

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.

**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*](https://airbnb.design/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](https://medium.com/etsy-design/etsy-studio-designing-a-human-centered-business-bd03000593b6). “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.

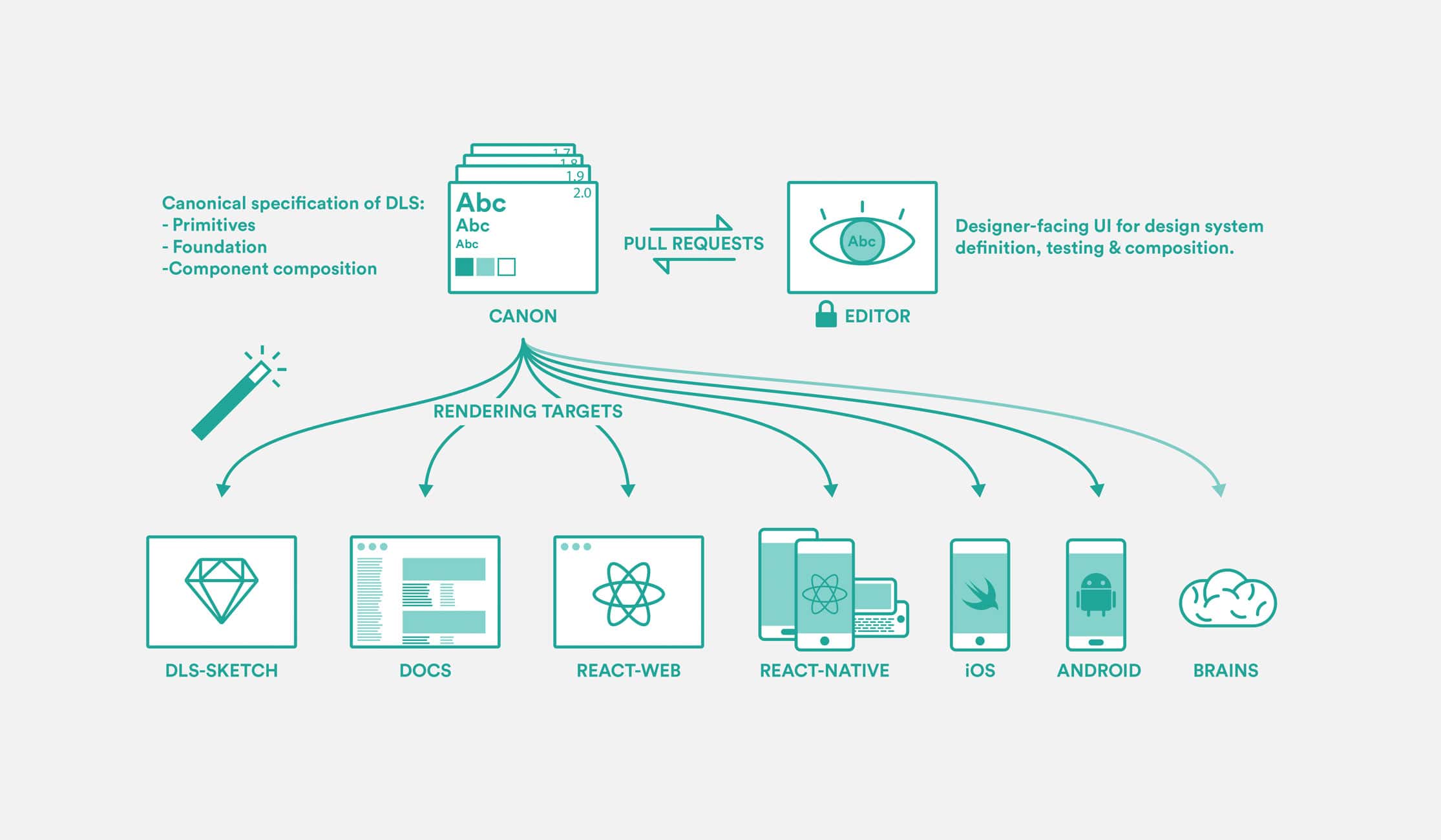


Figure 1. Single source of truth concept proposed by the Design Tools team at Airbnb, originally illustrated by Jon Gold.

**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](https://youtu.be/z5XxgxBz3Fo?t=7m).

**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](https://medium.com/eightshapes-llc/patterns-components-2ce778cbe4e8), 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 2 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](https://medium.freecodecamp.org/design-tools-are-running-out-of-track-94f21b6ae939), “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.

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.

# Painting with Code

## Introducing our new open source library React Sketch.app

Categories: [Behind the Scenes](https://airbnb.design/category/behind-the-scenes/), [Projects](https://airbnb.design/category/projects/) — Jon Gold

Today, we’re excited to share a tool we built to help bridge the gap between designers and engineers working on design systems at scale. [React-sketchapp](https://github.com/airbnb/react-sketchapp) is an open-source library that allows you to write React components that render to Sketch documents.

If you’re a designer or an engineer familiar with React, you should feel right at home with the new library, and you can [play with it right now](https://github.com/airbnb/react-sketchapp).

Here are some fun things we’ve already started doing with it:

**Seamless integration with your data**

Using the Foursquare API to search for places of interest, and the Google Maps API to generate dynamic maps.

Bring real data to your designs — including traditional APIs and GraphQL endpoints.

**Layout with flexbox**

Automatically testing our components on multiple screen sizes

Because React Sketch.app uses flexbox, its components can have the same rich layout as your real components. No more dragging rectangles by hand — everything works like your target layout engine!

**Automatic internationalization**

Using the Google Translate API to automatically translate text into multiple languages

We’re an international company, with users in every corner of the world — it’s vital that our designs work well for everyone. Our design tools should enable us to think beyond our native languages.

So, why did we want to put React in Sketch?

### Sources of truth

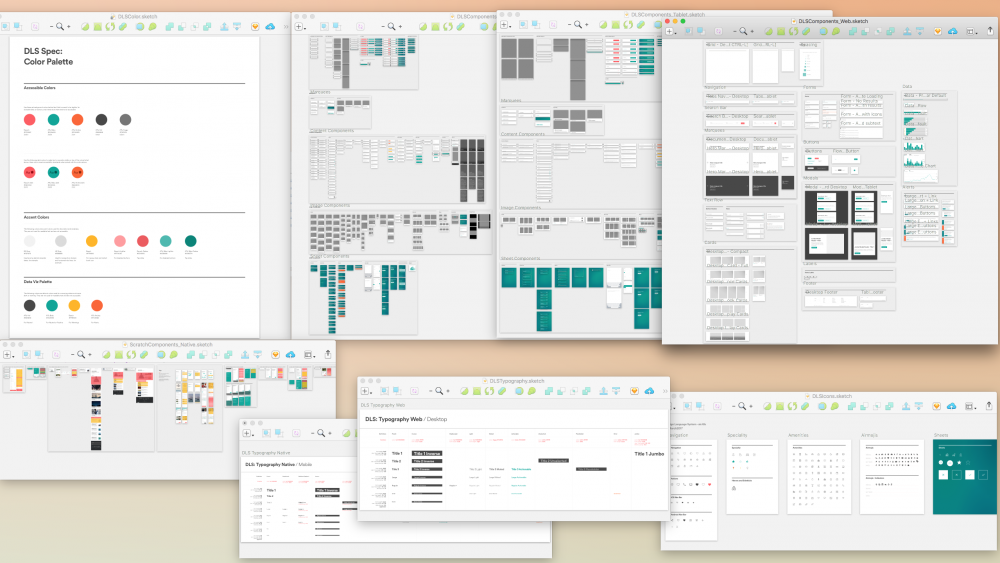
Design systems allow designers to reuse styles, components, and patterns, which in turn gives them time to focus on higher-level thinking. A good design system also empowers engineers to confidently implement new features without relying on redlines or back-and-forth pixel tweaking with a designer. And yet, comprehensive design systems in large organizations can introduce their own growing pains. In our case, one of our bottlenecks was in our Sketch templates.

I’m a big fan of Airbnb’s design system, [DLS](https://airbnb.design/building-a-visual-language/). It starts with common primitives — typography, color, and spacing — and grows into a rich library of components that work across platforms, screen sizes, and languages.

Of course, a design system is never complete. We’re constantly adding to DLS, changing existing components and optimizing it to make it usable by everyone in the world.

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.**

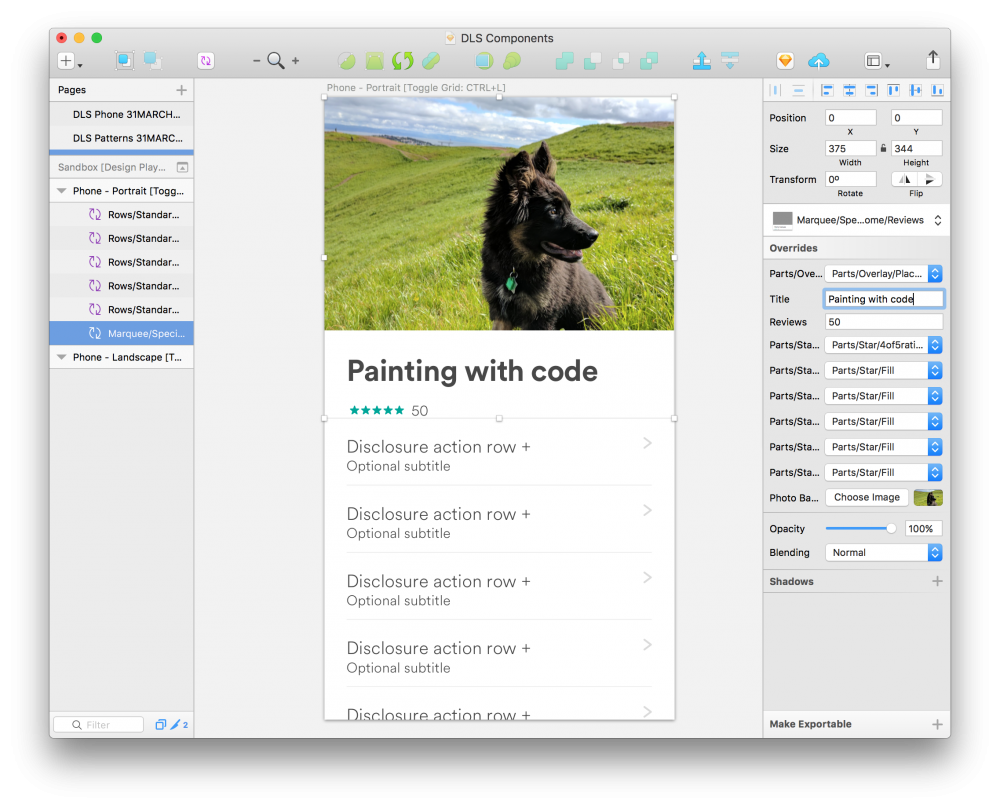
Code is relatively easy to tie together, and we already have an infrastructure that allows for version control and continuous integration of our applications. Until now though, maintaining Sketch templates relied on manually-intensive human touch points.



Due to how design systems are composed, changes have to be tracked to see how they affect the system — changing a color reference might affect dozens of components. Each component must be redrawn to respect line-height and spacing. Niceties, such as converting colors for display in different formats, annotating specs, and calculating color contrast ratios for accessibility, all increase the potential for errors and inconsistencies. Graphics-based design tools are notoriously bad at version control. All these moving parts left us frequently uncertain about the state of the system. Questions, such as “What’s the font size for titles on mobile,” “Is this still what this component looks like,” or How does this component work on different screen sizes,” were all too common around the office.

These problems compound as organizations grow. What is fixable instantly when a freelancer designs and codes everything alone is a frustration for a three-person design team. Likewise, the inefficiencies that a medium team can roll with are catastrophically wasteful to a team of our size. Pruning human touch points from DLS is foundational to ensuring we can grow into an even more effective team in the next few years.

### Using the real thing



I’m obsessed with the tooling principle of just using the thing itself. A designer’s workflow in Sketch is actually pretty similar to using React components: she picks a component, and puts data in it. In Sketch, we use *symbols* and *overrides*, in React we use *components* and *properties*. **The concepts are so similar that it seemed silly not to unify them.**

We also wanted to minimize sources of truth. Why keep a separate library of components drawn by hand in Sketch once we have them implemented as the real thing used every day by millions of people?

The fewer sources of truth we have for a design system, the more efficient we are.

“Many hard problems are best solved when they are addressed backward.”  
— Charlie Munger, Vice-Chairman of Berkshire Hathaway

Design tools that seek to produce production-ready code are as old as the internet. As the industry has coalesced around Sketch, people have sought to generate code from Sketch. This is exciting, but for our challenges we wanted to do the *exact opposite* — **to keep DLS in sync we have to generate Sketch files from code.**

Having built several internal tools on top of Sketch in 2016, we knew the potential of scripting it to manipulate files as we wanted. However, its programming language, CocoaScript, isn’t the most familiar to most developers, and we wanted to be able to fully embrace the power of modern JavaScript to quickly and intuitively create documents.

“React is such a good idea that we will spend the rest of the decade continuing to explore its implications and applications”  
— [Guillermo Rauch](https://twitter.com/rauchg/status/801005961334943744), founder of ZEIT

At Airbnb, we’re big fans of [React.js](https://facebook.github.io/react/). It’s the perfect paradigm for thinking about and building componentized design systems, and we use it for large parts of our web and mobile applications. React is a much bigger idea than any one platform though — as well as the familiar React and React Native implementations, there have been interesting projects using the same principles for [virtual reality](https://facebook.github.io/react-vr/), [blinking LEDs](https://github.com/iamdustan/react-hardware), [music synthesizers](https://formidable.com/blog/2016/08/22/make-dope-beats-with-reactjs/), and [terminal applications](https://github.com/Yomguithereal/react-blessed).

**React as a paradigm is perfect for wrapping the complexity of underlying platform APIs and providing consistent and smooth tools to the developers using them.**

### The adjacent possible

“The strange and beautiful truth about the ‘adjacent possible’ is that its boundaries grow as you explore them. Each new combination opens up the possibility of other new combinations”  
— Steven Johnson

Over dinner on a rainy San Francisco evening, our team posed the question: “What if we used React to write Sketch files?”Shortly thereafter, we ended up with a pretty good proof-of-concept React renderer. It was perfect for laying out color swatches and type specimens, but we hadn’t yet figured out how we’d use it for more ambitious documents.

Using Sketch-style components, such as <rect>, <circle>, and <line>, meant we’d have to recreate each DLS component to be able to draw it to Sketch. While faster at scale than creating by hand, it was still (programmatically) drawing a picture of *the thing* rather than just *using the thing*. We’d inadvertently removed one source of truth and replaced it with another!

A breakthrough came when Airbnb engineer Leland Richardson proposed using React Native-style components. <View>, <Text>, etc. are the basic units of composition for design systems, so that semantically linked our Sketch components with their production counterparts. More excitingly, with Leland’s [React Primitives](https://github.com/lelandrichardson/react-primitives) project we could render *real React components* to Sketch, the browser, and our phones at the same time.

We began the project to reduce the time it takes to generate static assets. But through exploring its edges and adjacent possibilities, we’re unearthing exciting and novel ways of interacting with design systems in Sketch. Many tasks that were previously unfeasible, required massive human input, or relied on sketchy (sorry) plugins are now enabled with the same code that our engineers are writing day-to-day.

By lowering the bar for Sketch compatibility, we also open ourselves up to new and exciting experimental tooling, tailored to our workflow. We can prototype the smarter design tooling of tomorrow while maintaining full compatibility with the software that our team uses today.

### Moving forward

“We’re investing in code as a design tool. Moving closer to working with assets that don’t only include layout and design, but also logic and data. This helps bridge the gap between engineers and designers, thus reducing the need for design specs–or redlines–and the steps between vision and reality”  
— Alex Schleifer, head of design at Airbnb

The project wouldn’t have been possible without deep contributions from the community. Thank you to [Andrew Pouliot](https://github.com/darknoon), [Mathieu Dutour](https://github.com/mathieudutour), and countless others who wrote code, reviewed pull requests, played with demos, and wrote documentation. We’re excited to open source our work, and we look forward to more collaborations in the future 🙂

We hope you enjoy using [React Sketch.app](https://github.com/airbnb/react-sketchapp) as much as we do. We’re still in the early days of figuring out all its possible applications, but it’s already a great tool for wrangling design systems at scale, using real data, and bringing design and engineering closer together.

We’re excited to see what you make with React Sketch.app — feel free to reach us by [email](mailto:designtools+reactsketchapp@airbnb.com) with feedback, thoughts or suggestions. And if these projects sound interesting to you, our design tools team is hiring Design Technologists in San Francisco 🙂

**FedEx Enterprise XD Design System**

Web design systems fail by depending on images, pseudo code, design specs and other artifacts for design and for style sheets.

This virtually ensures that any change or addition to a project or style sheet require a cascade of work as images are redrawn, specs updated and styles revised. As a result, the system breaks down and become impossible to maintain, especially at scale.

But, the solution is simple, direct and obvious. Use real code to design software and to build a ‘live’ style sheet.

Working in code eliminates the uncertainty between what’s real and what’s not. And code goes beyond layout and design to include logic and data as well. Code is efficient, bridges the gap between design and development and is easy to tie together with version control and continuous integration.

Why keep a separate library of components drawn by hand when we have the real thing used every day by millions of people? A code based design system empowers engineers to confidently implement new features without relying on redlines or back-and-forth pixel tweaking with a designer.

In this system designers build real web pages with actual web elements and components using a simple and direct drag and drop design tool. This tool eliminates the guess work inherent in image based solutions. Once the design is complete developers have only to review and refine the code for testing and production.

Any modern design system must include a rich library of components that work across platforms, screen sizes, and languages.

Current design tools are inadequate. Most only 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 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.

Begin by defining top level goals that include organization, maintainability and scalability.

* Flexible modules rather than pages by building components outside of context
* Define codebase conventions and guidelines that’s scalable, modular and maintainable
* Use BEM(Block, Element, Modifier) for CSS coding
* Style guide driven development

As the number of viewports and environments proliferate, it’s become untenable to produce static mockups of every page of a web experience. As Stephen Hay quipped, presenting fully baked Photoshop comps “is the most effective way to show your clients what their website will never look like.”

Atmosphere describes the feelings we get that are evoked by colour, texture and typography. You might already think of atmosphere in different terms. You might call it “feel”, “mood” or even “visual identity.” Whatever words you choose, the atmosphere of a design doesn’t depend on layout. It’s independent of arrangement and visual placement. It will be seen, or felt, at every screen size and on every device. [Andy Clarke](http://stuffandnonsense.co.uk/blog/about/an-extract-from-designing-atoms-and-elements)

We’re not designing pages, we’re designing systems of components. [Stephen Hay](http://bradfrost.com/blog/mobile/bdconf-stephen-hay-presents-responsive-design-workflow/)

What makes up a design system?

* Brand identity
* Design language
* Voice and tone
* Writing
* Code style guides
* Pattern libraries
* *The hard part is building the machine that builds the product.”* [*Dennis Crowley*](http://techcrunch.com/2011/03/03/founder-stories-foursquare-crowley-machine/)

The cornerstone of pattern-based design and development is the pattern library, which serves as a centralized hub of all the UI components that comprise your user interface. As we discussed in chapter 1, the benefits of pattern libraries are many:

* They **promote consistency and cohesion** across the entire experience.
* They **speed up your team’s workflow**, saving time and money.
* They **establish a more collaborative workflow** between all disciplines involved in a project.
* They **establish a shared vocabulary** between everyone in an organization, including outside vendors.
* They **provide helpful documentation** to help educate stakeholders, colleagues, and even third parties.
* They **make cross-browser/device, performance, and accessibility testing easier**.
* They **serve as a future-friendly foundation** for teams to modify, extend, and improve on over time.
* In a traditional siloed design process, it’s typical to see lengthy wireframe and spec documents created, debated, and approved. These documents usually take the form of gigantic PDFs, which is unfortunate considering they often contain all sorts of valuable insights, instructions, and documentation about the design system. Sadly, these bulky artifacts are often thrown into a (virtual) trash can by the time the project makes its way into production.
* This shouldn’t be the case. A UI’s documentation should contain insights from every discipline involved in creating it, and—this is key—should be baked into the living, breathing design system. Effective pattern libraries carve out a space to define and describe UI components, articulating considerations ranging from accessibility to performance to aesthetics and beyond.
* Pattern Lab provides several ways to add pattern descriptions and annotations to a design system. Pattern descriptions can be added by creating a Markdown file that corresponds to the name of a pattern (e.g. *pattern-name.md*), which will show the pattern description in the library list view.

Working in HTML, CSS, and presentational JavaScript allows teams to not only create aesthetically beautiful designs, but demonstrates those uniquely digital design considerations like:

* flexibility
* impact of the network
* interaction
* motion
* ergonomics
* color and text rendering
* pixel density
* scrolling performance
* device and browser quirks
* user preferences

Crucially, jumping into the browser faster also kick-starts the creation of the patterns that will make up the living, breathing design system. More on this in a bit.

**Working in the browser allows teams to address layout issues across the entire resolution spectrum, design around dynamic data (such as variable character lengths, image sizes, and other dynamic content), demonstrate interaction and animation, gauge performance, factor in ergonomics, and confront technical considerations (such as pixel density, text rendering, scrolling performance, and browser quirks).** Static design comps cannot deal with all these considerations, so they should be treated merely as hypotheses rather than set-in-stone specifications. Only when transferred to the browser can any design hypothesis truly be confirmed or rejected.

*Let’s change the phrase “designing in the browser” to “deciding in the browser.”* [*Dan Mall*](https://the-pastry-box-project.net/dan-mall/2012-september-12)

**Once the designs are in the browser, they should stay in the browser.** At this stage in the process, the point of production shifts to team members adept at crafting HTML, CSS, and presentational JavaScript. Patterns should be created, styled, and plugged in wherever they’re needed. Designers can react to these in-browser implementations and can create spot comps in static tools to help iron out responsive wrinkles at the organism level. This back-and-forth between static and in-browser tools establishes a healthy loop between design and development, where the front-end code becomes more solid and stable with each iterative loop.

**Code is the key….its the thread that holds it all together…design, development, devops, etc. We need to give designers the tool that works for them but silently builds code behind the scenes.**

## The problem with current Style Guide Solutions

### A Static HTML Style Guide

Style Guides built with static HTML are standalone representations of your UI components with no direct link to your codebase. Once you change or refactor your HTML or CSS you need to update the Style Guide if you want it to reflect the latest version.

The main difficulty with keeping this up to date with your application is having to update both versions of the same template. It requires diligence on the part of all developers to manually keep it up to date.

Read more: <http://engineering.lonelyplanet.com/2014/05/18/a-maintainable-styleguide.html#ixzz55DzF14MN>

For the holy grail you must build both your style sheet and your applications at a component level with a component based architecture. To do this you must decouple the user interface from the application, and split them into components. This has a positive effect on workflow and codebase. You need to create a simple API to fetch the UI components from the component layer. You need to maintain the mapping between the latest version of the component and the application and not have developers copy and paste component code.

Having a single version of the component, accessible via an API, worked perfectly with unit testing too so we could ensure that the contract between the API parameters and the returned template was solid. We could modify and extend the component based on the data we passed it and assert on the returned result. This also allowed us to add accessibility helpers and microformat attributes as standard and ensure that they weren't forgotten when used in new applications.   
  
Read more: <http://engineering.lonelyplanet.com/2014/05/18/a-maintainable-styleguide.html#ixzz55E3B04ne>

Your style guide simply becomes another application with a component architecture.

Paypal API: <https://github.com/paypal/api-standards/blob/master/api-style-guide.md>

[**https://www.paypal-engineering.com/2017/09/08/announcing-the-latest-api-style-guide/**](https://www.paypal-engineering.com/2017/09/08/announcing-the-latest-api-style-guide/)

[**https://www.invisionapp.com/blog/guide-to-design-systems/**](https://www.invisionapp.com/blog/guide-to-design-systems/)

**single source of truth…multiple touchpoints…channels and platforms.**

[**http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api/**](http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api/)

[**https://readwrite.com/2015/11/16/how-to-build-an-api-amazon-lambda/**](https://readwrite.com/2015/11/16/how-to-build-an-api-amazon-lambda/)

[**https://schoolofdata.org/2013/11/18/web-apis-for-non-programmers/**](https://schoolofdata.org/2013/11/18/web-apis-for-non-programmers/)

[**https://stackoverflow.com/questions/4973156/how-to-write-a-rest-api**](https://stackoverflow.com/questions/4973156/how-to-write-a-rest-api)

[**https://www.smashingmagazine.com/2016/12/styling-web-components-using-a-shared-style-sheet/**](https://www.smashingmagazine.com/2016/12/styling-web-components-using-a-shared-style-sheet/)

[**https://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/**](https://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/)

[**https://caniuse.com/#search=components**](https://caniuse.com/#search=components)

[**https://www.webcomponents.org/community/articles/introduction-to-shadow-dom**](https://www.webcomponents.org/community/articles/introduction-to-shadow-dom)

From: <https://developers.google.com/web/fundamentals/web-components/shadowdom>

Shadow DOM v1: Self Contained Web Components

Eric Bidelman (google)

HTML, CSS and JS are brittle and problematical. Theres no tellif if HTML id/class with conflict and bugs creep up with CSS specificity a big issue….style selectors grow and performance goes down.

HTML templates

Shadow DOM

Custom elements

HTML imports

Benefits of Shadow DOM are CSS scoping, DOM encapsulation and composition).

Shadow DOM is designed as a tool for building component-based apps. Therefore, it brings solutions for common problems in web development:

* **Isolated DOM**: A component's DOM is self-contained (e.g. document.querySelector() won't return nodes in the component's shadow DOM).
* **Scoped CSS**: CSS defined inside shadow DOM is scoped to it. Style rules don't leak out and page styles don't bleed in.
* **Composition**: Design a declarative, markup-based API for your component.
* **Simplifies CSS** - Scoped DOM means you can use simple CSS selectors, more generic id/class names, and not worry about naming conflicts.
* **Productivity** - Think of apps in chunks of DOM rather than one large (global) page.

When the browser loads a web page it does a bunch of interesting stuff. One of the things it does is transform the author's HTML into a live document. Basically, to understand the page's structure, the browser parses HTML (static strings of text) into a data model (objects/nodes). The browser preserves the HTML's hierarchy by creating a tree of these nodes: the DOM. The cool thing about DOM is that it's a live representation of your page. Unlike the static HTML we author, the browser-produced nodes contain properties, methods, and best of all...can be manipulated by programs! That's why we're able to create DOM elements directly using JavaScript:

Polyfils: <https://www.webcomponents.org/polyfills>

<https://www.webcomponents.org/about>

<https://dmitriid.com/blog/2017/03/the-broken-promise-of-web-components/>

\https://www.linkedin.com/pulse/native-html5-component-development-using-web-components-kumar

<https://www.upwork.com/hiring/development/web-components/>

<https://speakerdeck.com/1000ch/the-state-of-web-components> (video)

<http://blog.teamtreehouse.com/author/mattwest>

<http://blog.teamtreehouse.com/working-with-shadow-dom>

https://www.html5rocks.com/en/tutorials/webcomponents/imports/

**.**