Reading notes on design systems, components & tokens

Functions and the future of design systems

https://matthewstrom.com/writing/functions-in-design-systems/

Today, most design systems work like dictionaries, composed of a finite set of definitions. But there's a new page turning, one that ushers in the age of functions. Design systems are starting to use functions that recreate the design decision making process, responding appropriately to any input they are given. The early uses of functions do a great job of extending the reach of design systems. But the potential is vast.

For example: color palette or typographic scale generation.

Design System Communications

https://medium.com/eightshapes-llc/design-system-communications-ca679ffc36d3

For design systems, outcomes that matter include:

- Adoption incrementally deeper over time
- Awareness of new, upcoming features
- Satisfaction through open participation
- Trust through inclusiveness and transparency
- Alignment with other objectives and programs
- Feedback to redirect focus
- Celebration of adoption and contribution

Scenario-Driven Design Systems

https://www.youtube.com/watch?v=wxvu_lIUVS0 or PDF

A system is: elements, interconnectedness, purpose.
A good design system feels: cohesive, unified, connected.

Identify key moments for brand expression.

Do not start with individual components, be more specific.

Visual inventory and purpose inventory: What elements are presentationally similar, and which are semantically similar?

Only add layout when the content requires it, base everything on the content's needs.

Unify components that have the same goal, be specific, but keep patterns general and push back on visual variations. Visual variations are okay if they are critical to the brand.

Good variation:

– If there's a specific problem that we need a new pattern to solve
– Determined by user scenarios and content needs
– Strengthens brand voice in a way that serves our audience

Bad variation:

– Visual variation on components that serve the same function across brands
– Don't do much to strengthen brand voice

Design Systems Handbook

https://www.designbetter.co/design-systems-handbook

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

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

Criteria to find good candidates for design systems pilot projects:

– *Potential for common components*. Does this pilot have many components that can be reused in other products?
– *Potential for common patterns*. Does this pilot have many patterns that can be reused in other products?
– *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.
– *Technical feasibility*. How simple is a technical implementation of the design system? Is a large refactor required?
– *Available champion*. Will someone working on this product see it through and celebrate/evangelize using the design system (and even contribute to it)?
– *Scope*. Is this work accomplishable in our pilot timeframe of [3–4 weeks] (insert your timing here)?
– *Technical independence*. Is the work decoupled enough from other legacy design and code that there are clear start and end points?

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

Alan Kay:

> 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.

“I Made This. Does It Go in the System?”

https://medium.com/eightshapes-llc/i-made-this-does-it-go-in-the-system-3b67b9894531

Designers and engineers always confront challenges the system doesn’t solve. And then, they’ll solve it. Through blood, sweat and tears, they’ll develop fervent belief about the solution’s value. The system is their path to amplify that value. But is the system the right place for it?

Does It Belong in the System?

Systems are never the repository for every style, component, and feature designed and built for every product ever. Instead, a system should include what’s shared across many products and omit what’s not.

1. Is it relevant to any other product? If so, how many?
2. Is it consistent with the system’s vision?
3. How much will it cost to make and maintain?
4. Does it trigger momentum in a new direction?
5. How deeply can YOU guide its use?
“What guidance can you document now? Use…when? Configure…how? Alter content…in what ways? How does this work, for everybody?” If your contributor struggles to articulate ideas and you don’t know either, maybe the feature isn’t mature or ready for broad distribution. However, what if your contributor quickly cites scenarios of use, rattles off do and don’ts, and itemizes editorial and style considerations? Now we’re talking!
6. Is the timing right for contributor AND system?

What’s Left to Be Done?

In my 10+ years of system work, I’ve never seen a contribution complete enough to be immediately merged into the system. Instead, there’s work to fine tune the contribution to:

1. Expand to a minimally sufficient feature set
2. Reduce scope to broaden relevance/omit contentious features
3. Always, always normalize to how system features are designed and built

The Most Exciting Design Systems Are Boring

https://bigmedium.com/ideas/boring-design-systems.html

With projects like this, there's often a strong temptation to throw out the old and start fresh with fancy new design components. "If we're going to establish standards," whispers the devil on your shoulder, "then let's get rid of all the old stuff. Let's blow it out with a new look, fancy interactions, and a shiny tech framework." (Or my favorite: "Let's make our own Material Design.")

Design-system builders should resist the lure of the new. Don't confuse design-system work with a rebrand or a tech-stack overhaul. The system's design patterns should be familiar, even boring.

The job is not to invent, but to curate.

Design systems are containers for institutional knowledge. They provide tested and proven solutions to design problems. When these solutions are held together by a consistent visual language and UX guidelines, they represent what good design looks like for the organization or platform.

When the design system is boring, it frees designers and developers to do the new stuff, to solve new problems. The design system carries the burden of the boring, so that designers and developers don't have to.

Components are commodities. The magic happens in the guidelines that come with them.

Design systems, like code frameworks, are no place for untested ideas. Extract proven ideas from production interfaces.

The GE Design System and Thoughts about Craft at Scale

https://vimeo.com/132580829

No one ever follows style guides. Maybe it's more like LEGOs than it is a set of rules. It's a kit of parts that snap together and that you can make things of. Tools not rules.

The Way We Build

http://airbnb.design/the-way-we-build/

Here's the simple truth: you can't innovate on products without first innovating the way you build them.

Christopher Alexander states that "nothing which is not simple and direct can survive the slow transmission from person to person."

Style Guide Best Practices at Beyond Tellerrand

http://bradfrost.com/blog/post/style-guide-best-practices-at-beyond-tellerrand/

Solid recap.

The 6 Flavors of Style Guide: Brand Identity Guidelines, Design Language, Voice and Tone, Writing, Code, Pattern Libraries *(**covered here**)*.

And if the boss says no, then do it anyways. I firmly believe this. I'm very much a proponent of asking forgiveness not permission. And this is especially true with establishing a pattern library and design system. Because guess what? In order to make the new website, you have to establish the parts of that whole anyways. Instead of this, which technically you can make some stuff out of a disheveled box of Legos. But by taking just a little bit of time to organize those pieces, then you're able to do that work a lot more efficiently. It doesn't seem like rocket science, but boy, does it not happen as much as it should.

So how do we actually create a sound UI design system? What does this actually look like? How can we create not just good final work, but also leave behind these great underlying UI design systems.

What I think needs to happen instead is this. Instead of thinking "we're making websites" we need to say "we're making design systems that are making that website". The website is one instantiation of our design system. That is a fundamental shift that I really think sets you up for long-term success because it forces you to go through your design system in order to make updates and changes to your site.

There's this concept of the Holy Grail, and this is something that anyone that's been talking about pattern libraries has been pursuing. What we want is this magical setup where we have our pattern library with all of our Lego bricks that make up our final website, and if we make a change to one of those Lego bricks, anywhere that Lego brick is included will just magically update. That is awesome. Very very few people have made this idea come to life. Ian Feather and his team at Lonely Planet have done exactly this. They created their design system, wrote an API for their pattern library, and their production environment eats their pattern library. Which is amazing. And what this allows them to do is crank out new pages and new features all the time. They can make tweaks, make performance tweaks, make accessibility tweaks, and they'll just roll out automatically to their production environment, which is amazing.

It's definitely not impossible, but it takes some thought and set up, and it's well worth your time.

In order to make style guides cross-disciplinary, they need to be attractive looking. This sounds like a no-brainer; well-designed things will get used more. So taking the time to create these good-looking, approachable style guides lead to more people using them. They become more valuable.

Their style guide serves as a huge recruitment tool.

The Language of Modular Design

http://alistapart.com/article/language-of-modular-design

As many of us move away from designing pages toward designing systems, one concept keeps cropping up: modularity. We often hear about the benefits of a modular approach; modules are scalable, replaceable, reusable, easy to test, quick to put together—"They're just like LEGO!"

Modularity might appear to be a simple concept at first, but making it work for your team demands significant effort and commitment.

The biggest challenges around modularity are all the decisions that need to be reached: when to reuse a module and when to design a new one, how to make modules distinct enough, how to combine them, how to avoid duplications with the modules other designers and teams create, and so on. When modularizing an existing design or building a new one, it's not always clear where to begin.

The biggest obstacle teams face is the lack of a shared language. To help establish that shared language, she suggests that we discuss, vet, and document our ontological decisions in the form of "controlled vocabularies."

The defining step toward thinking modularly was going through the process of building a pattern library as a team, which took several months (and is still in progress).

Name things collaboratively, based on their high-level function.

In the process of naming an element, you work out the function as a group and reach an agreement. It's not so much about giving something a great name (although, of course, that's an ideal to aspire to), but *agreeing* on the name.

Naming things together is a useful habit for your team to develop, because in the process of trying to give something a name that makes sense, you work out its function and, most importantly, reach consensus.

Make an effort to refer to the elements by the name you agreed on—no matter how strange this might sound in everyday conversations. It takes more effort initially to call something a "whisper box" (yes, we have an element called "whisper box") rather than "that thing with the lines and an icon in the middle." But until you start referring to an element by its proper name, it doesn't exist in your modular system as a solid, actionable block. Every time you use the name you agreed on, you strengthen the element you call on and evolve your design language.

Unless it's a part of your build

https://twitter.com/operatino/status/652414383759446016

Unless it's a part of your build (or dev process), your styleguide is just more documentation to maintain.

Component QA in Design Systems

https://medium.com/eightshapes-llc/component-qa-in-design-systems-b18cb4decb9c

Good inventory of the properties of quality components, such as Sufficient States & Variations, Content Resilience, Composability, *etc.*

Components

https://jxnblk.com/blog/components/

Arbitrary categorization (e.g. atomic nomenclature) doesn't help with composability.

A UI system that is made up of independent stateless components is extremely flexible. When individual pieces need to be swapped out or updated, those changes are isolated and don't cause other parts of a system to break.

Naming things is hard, there's no debate there, but when you start to categorize different parts of a UI into pages, views, flows, atoms, molecules, materials, or kittens, you've already started to undermine the concept of composability, and it probably takes more time and effort to get an entire team of people to "agree upon" your proposed naming conventions than it's worth.

The point of this is to think about everything as an interoperable system. You can slice and dice components in any way you see fit, and these components are likely to change and be fine tuned as a system is developed. Premature optimization is a trap that's easy to fall into. Embrace the chaos as you build. Patterns will emerge from the primordial goop of UI that is your product, and by consistently thinking about a composable system you'll probably come up with something more flexible and more robust than if one person dictates a dogmatic framework to work within.

Design Tokens Technical Reports

https://tr.designtokens.org/

Normative documentation for tokens. Solid and well-articulated; covering the "what" and the "how" more than the why. The action is really at ***https://tr.designtokens.org/format/****.*

Design Tokens Only Exist In A Pre-Processed State

https://carbonemike.com/design-tokens-only-exist-pre-processed

E.g. the initial token could store a transparency value, that would be lost on platforms supporting only hex values.

Once compiled and transformed, the output is no longer a design token. Instead, it's simply what it is: a key value pair fit for the platform. A variable in a form the platform can understand.

The reason for this conclusion is because the purpose of a design token is to exist as a value, surrounded by context, ready to be transformed. When the design token goes through that transform process, it will inevitably lose nearly all of its meta information. That meta information is very important: it provides context to the token and instructions for how that token should be transformed.

Documenting Design Tokens

https://dbanks.design/blog/documenting-design-tokens/

In this article I will show you all the cool things people are doing to document design tokens.