Reading notes on design systems, components & tokens

Documentation Is a Living Thing: How We Talk Informs What We Make

https://www.supernova.io/blog/documentation-is-a-living-thing-how-we-talk-informs-what-we-make

Well articulated means and ends of design system documentation.

Documentation isn't a snapshot in time; it's a living entity. Regular updates ensure the documentation remains aligned with the current thinking, empowering teams to create innovative solutions while upholding the core principles. In this context, documentation isn't a mere record; it's an ongoing conversation that changes over time.

The lifecycle of documents: Temporary insights → Progressive foundations → Structured collaborations → Reflection and evolution.

Beyond design systems for interfaces: a mega system of design systems

https://medium.com/@mluvinh/beyond-design-systems-for-interfaces-an-megasystem-of-design-systems-4f87dd6d8df8

Interesting as a characterization of maturity. Not necessarily practical, as it doesn't map to structure or incentives of most orgs.

– **Level 5: Design systems for interfaces** which relate to how an organisation designs and builds its interfaces (based on the definition framed by Brad Frost in 2021). This could be a digital interface, but also another channel or medium if your services include physical touch points.
– **Level 4: Design Systems for products** which relate to how an organisation designs their products, whether consumer-facing or not.
– **Level 3: Design Systems for services** which relate to how an organisation designs and orchestrates its services. By services, I mean here large(r) end-to-end services in the economic/business sense (e.g. "online food delivery platform") rather than functional microservices (e.g. "authentication/logging in")
– **Level 2: Design Systems for brands** which relate to how an organisation envisions, designs and shapes their brand(s). Design systems for brands might include components and guidelines on how to create new (sub)brands but in many cases, there's only 1 brand at play. Thus the design system might focus on the different expression and direction of the brand, and all related guidelines. Brands might be internally-facing or externally facing to customers.
– **Level 1: Design approach of the organisation** which relates to overarching design-related guidelines that transcend every design system level.

The Best Design Systems Are a Shared Work Language

https://medium.com/demagsign/the-best-design-systems-are-a-shared-work-language-7f2daea7d967

From our experience, a design system creates a lasting impact when a shared language — including the vision, mindset, and process — is established from the beginning, then evolved with the input of a cross-disciplinary group of stakeholders.

At its core, design systems create a model to repeatedly create and ship efficient, meaningful, and sustainable products at scale.

While working with GoFundMe, we created dedicated sections for each discipline (product design, engineering, marketing, data, etc.) to personalize onboarding, contributions, and best practices for cross-disciplinary collaboration.

We often say that a design system should solve 80% of the basics and give guidance on how to build new patterns.

The correct method for governance (the workflows, internal processes, and organizational structure) is as unique as the organization itself. It is often influenced by factors like product scale, number of design team members, and product roadmap.

Your design system is a dashboard

https://twitter.com/vingar/status/1608141621933273092

Your design system is the work-in-progress dashboard of your organization's growth and concerns.

*Similaire, via **Dan Mall** :*

> The official story of how your organization designs and builds digital interfaces.

— Brad Frost, **Design systems are for user interfaces**

> Any set of decisions governed across an organization.

— Hayley Hughes, **Trust Between Teams**

Your design system contribution practice is doomed to fail

https://amyhupe.co.uk/articles/your-contribution-model-is-doomed

When Nathan Curtis interviewed a group of design system leads in 2020, they told him that although contribution was valuable for other reasons, they “do not reduce workload and do not make [a] system produce more”.

In fact, in *my* experience, contributions actually increase workload and make our systems produce less.

Contributions are often implicitly following a socialist model, whereas the org and the individuals work after a capitalistic one. Hence misaligned incentives, leading to poor results. Contributions can even reinforce systemic inequalities.

Truthish.

https://ethanmarcotte.com/wrote/truthish/

I’ve written before that design systems often widen the gap between design and engineering teams. In practice, this frequently means each “side” maintains its own workspaces, processes, and documents. I’d go so far as to say most design systems feel like several discipline-specific artifacts stacked underneath a single trenchcoat: inventories and audits conducted by the content team, a component library owned by engineers, a set of visual standards maintained by designers, and so on.

I’m not here to critique this multiple-artifact model, mind you. Until tooling improves to the point where cross-discipline collaboration doesn’t require expensive, custom-built solutions, designers and engineers will continue to work in the environments available to them. As they should!

Despite this, most design system teams position the component library as the “source of truth” for the entire design system.

There’s an interesting tension here. Because practically, this means most design systems contain *multiple* sources of truth, while formally acknowledging only one. It’s one of the surest ways that hyperobject feeling can creep in.

Consistency vs Flexibility: The Myth of Creative Restraint in Design Systems

https://sparkbox.com/foundry/design_system_consistency_versus_flexibility_design_system_constraints

But the work is not only technical, it’s cultural. If you want to build a systematic design practice, you’ll have to let go of the tools and the techniques and focus on the people. Consistent output comes from organizations that are culturally consistent—aligned on values and principles. In other words, an organization is successful when it aligns people behind a shared vision, so that everyone’s “wants” match the needs of the business.

Your design system can be a catalyst for that alignment, but it won't create it on its own. That is the unspoken and deeply challenging mandate of a design system team. The reward is a system flexible enough to enable innovation alongside digital products that are authentically consistent. It will take time to get there, but don't settle for anything less.

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_llUVS0 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.

All the user-facing states

https://ericwbailey.website/published/all-the-user-facing-states/

User-facing state is what someone experiences when they interact with an element in some capacity.

30+ states!

Boringness in Design Systems

https://daverupert.com/2023/05/boringness-in-design-systems/

The component must be defensively designed and built against any permutation of author-supplied content. What was once a humble visual display component is now a content workhorse. *Une douzaine de props, sans parler des slots*.

Any component has the potential to become a little machine filled with a myriad of rules and requirements. In Storybook and Figma these lil' state machines manifest as knobs and dropdowns. Knobs provide an interface for the large underlying feature matrices of properties, showing the programmable pieces allowing experimenters to flip them on and off or change properties at will. These if-statements have a cost as each component swells with atomic business logic. If you're lucky, someone is keeping track of these intents in a test suite, otherwise it will be even harder to change later.

Quick and radical changes are always harder in a complex system. It's tough to "be funky" in the confines of a strict system.

All these complexities dilute the component towards a bland defensive design, averaging itself against every possible permutation of content. As utility goes up, so does boringness in a design system.

« Boringness » est une tension, un équilibre ; pas nécessairement une mesure de non-qualité ni un marqueur d'échec.

Customization vs. Configuration in Evolving Design Systems

https://engineering.atspotify.com/2021/04/customization-vs-configuration-in-evolving-design-systems/

Customization — Custom styles are added external to the component. These styles reference HTML elements and touch CSS properties directly. A low level of abstraction.

Pros: Autonomy, speed, innovation

Cons: Lack of coherency, loss of maintainability, potential duplication

Configuration — The original component is made more flexible. Additional parameters are passed to the component for more varied behavior. A high level of abstraction.

Pros: Consistency, contribution, maintainability

Cons: Can become a bottleneck, rigidness, vocabulary awareness

When evolving a design system, there is a range of strategies you can take. A more abstract configuration approach can increase consistency and maintainability, but at the risk of the system being a bottleneck for outgoing features. The less abstract customization approach enables quicker feature development; however, overall consistency of the product can suffer as a result.

The more mature a product or feature is, the more beneficial and feasible a configuration approach is. However, the iterative and low-level nature of customization makes it more suitable for prototyping and features which are bespoke, or are still subject to change.

How to decide which approach to use : feature maturity, product maturity, timeline, reusability.

Component naming problems

https://twitter.com/Amy_Hupe/status/1603113982671429634

Design system component naming problems are nearly *always* scoping problems in disguise.

99% of the time, when we're struggling to name a component, it's because we're not yet clear on what its purpose, parameters and applications are.

Component QA in Design Systems

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

Solid 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. The action is really at **https://tr.designtokens.org/format/**.*

This document describes the technical specification for a file format to exchange design tokens between different tools.

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.

Design tokens — What are they & how will they help you?

https://lukasoppermann.medium.com/design-tokens-what-are-they-how-will-they-help-you-b73f80f602ab

- Design tokens are a methodology to extract design decisions into a separate space. They are stored technology agnostic to be used on any platform.
- The two types of design tokens are choices & decisions.
- Choices are the available primitives like brand colors
- Decisions are choices applied to a context, e.g. what color is used for a primary button
- Continued interest in design tokens will bring better design software integrations and improved development tooling.
- Design tokens help have 4 main benefits
    - Design consistency across products
    - Improved maintainability

- Help scale products to other platforms
- Improve documentation of design decisions

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.