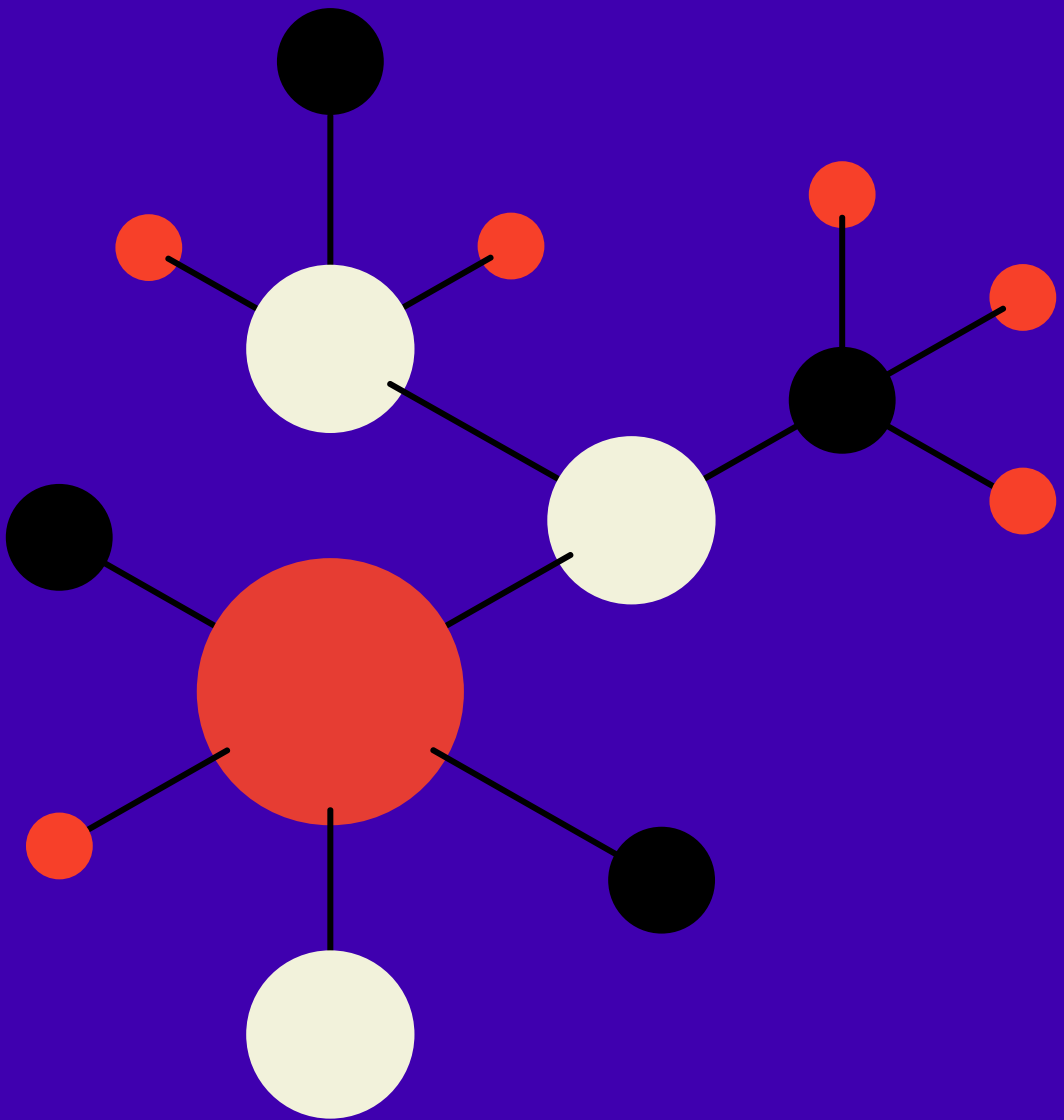
UXPin

The Actionable Guide to Starting Your Design System

The 100-Point Process Checklist



UXPin

The Actionable Guide to Starting Your Design System

The 100-Point Process Checklist

Marcin Treder, CEO and Head of Design Operations at UXPin

Index

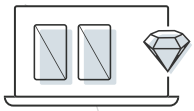
Design Systems in UXPin

UXPin

One platform for consistent design and development.

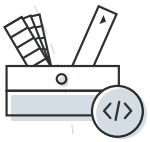
PayPal Sapient adidas HBO Microsoft

DESIGN SYSTEMS



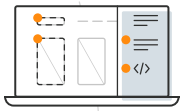
Design Language

Sync Sketch with UXPin for a consistent design language: fonts, colors, icons, assets, and more.



UI Patterns

Scale designs consistently with Symbols and interactive components.



Automated Documentation

Documentation syncs everywhere and travels with library elements.

Modular design and development

Scale quickly with design system libraries.

One source of truth for everyone

Close your knowledge gaps. Formalize your design and code conventions.

Painless documentation and developer handoff

Eliminate busywork. Generate style guides, specs, and documentation.

Tracy Dendy
HBO

"My productivity and developer productivity have both increased. They love that they can collaborate and move quickly to a powerful experience.

To book a demo, call +1 (855) 223-9114 or email us at sales@uxpin.com

Why Should You Have a Design System?

Design is more important than ever.

Companies fighting on a hyper competitive global software market can rely only on experience as a key differentiator. That puts enormous pressure on designers all over the world to take on more projects, deliver higher quality and work faster.

Unfortunately, these attempts are set up for failure. Design does not scale easily and efficiency required by the business is not going to emerge from thin air.

Scaling design through hiring, without putting standards in place, is a myth. With every new hire, new ideas for color palettes, typography and patterns appear in the product, growing the inconsistency and increasing the maintenance cost. Every new hire increases the design entropy.

There's only one way to stop the growth of the chaos. Committing to running a design system process. Gradual growth of a design system,

equals gradual decline of inconsistency and gradual increase of the speed of software development.

Design scales. But it scales only with a design system.

Why Should You Use This Checklist?

If you're a designer or a developer, feeling the pressure to deliver better experience – then this checklist is for you.

Built on the experience of creating an internal design system at UXPin and interviewing 40+ design and engineering leaders working on design systems, this checklist will help you:

- Create a foundation for a design system process
- Document inconsistencies in the interface
- Get organizational buy-in into the design systems process
- Build a small design systems team
- Organize design systems sprints
- Run the first 7 – 10 sprints
- Organize retrospectives

Start building your system today. It takes one committed person to kick off the process and change the way your organization builds software.

Time to scale!

Marcin Treder

The Inventory

To begin the process of scaling your design operations with design systems, you must understand the current state of your design and development ecosystem.

If you think the experience provided by your product is internally inconsistent and confusing, or the maintenance of code takes too much time – you must prove it to get your team and stakeholders on board.

The best way to start is by building an inventory of all the different patterns, colors, text styles and assets used in the product. The inventory is going to clearly show all the inconsistencies and hopefully let you build the case for a design system. Creating a complete inventory, as described below, takes time. You might want to select only few activities to get the team and stakeholders on board and continue when you'll have the full team focused on the task.

Create the Patterns Inventory

- [ ] Take screenshots of design patterns **or** collect design patterns from design projects, to form an inventory of all the different versions of patterns.

> Start by collecting different examples of patterns from your product(s). Either take screenshots and paste them into one place – for example your presentation tool, or your prototyping tool or collect project files and take patterns directly from there. You can read more about the inventory here.

- [ ] Check the frontend code and/or talk to developers about the architectural approach used across the codebase. Try to identify if the frontend architecture is *modular* and if you can use it to categorize patterns in the inventory.

> Quite often frontend developers work using modular architecture that can be used to organize patterns in the design system. Approach with care and a critical mindset, but definitely do not ignore what already exists. For example, at UXPin, our frontend developers divided CSS into: elements, modules and components. That modularity existing in the LESS files helped us build a strong foundation for the system and make the inventory more actionable as our reference document for the future sprints.

- [ ] **If** modular frontend architecture exists, categorize your patterns in the inventory using categories from the architecture.

  > Take patterns collected from products and group them into categories. For example, if the lowest level of the frontend architecture is an element then collect all the buttons and place them under a category ‘element button’. You might also want to document links to all the styling files associated with a given element. Not only will you communicate with frontend developers in the same language, you’re also showing the full scope of the inconsistencies.

- [ ] **If** modular frontend architecture, categorize patterns in the inventory

  > Even if there’s no modular fronted architecture in your ecosystem, you have to categorize all the patterns collected in the inventory. You **don’t** have to make the decision about the overall architecture of your design system just yet. Simply find the common categories for elements (buttons, form fields, sign-up forms…), so the team and stakeholders can quickly notice where the inconsistencies exist.

Create the Colors Inventory

A typical source of design inconsistency is the set of colors used across the product portfolio. For example, the inventorization of colors at UXPin showed a massive redundancy slowing down the development process. 116 colors variables with 62 shades of gray were used at UXPin before the emergence of our design system.

- ☐ Traverse the code files and list all the color variables and/or all the colors used in CSS files
- ☐ Take note of the number of places where a particular color appears
- ☐ Organize colors by a common denominator. Can be hue, shades, tones and tints or simple by similarity (all the grays, reds, greens etc.)
- ☐ Sum up the number of different colors, color variables and take note of interesting anomalies (such as the number of different shades of gray)

Create the Typography Inventory

In a complex project, typography can quickly get out of hand. Lack of consistent typographic scale used across the project makes the information architecture convoluted and hard to understand for

users, while it also increases the cost of product maintenance due to fragmentation of code. By inventorying text styles, you show your team and stakeholders where styles become inconsistent.

- [ ] Walk through the UI and check *all* the text styles through a browser console.

- [ ] Take notes of any inconsistencies of text styles in one project. Form a typographic scale by ordering text styles in the order of the informational importance (from h1 to small text). Create multiple scales if necessary.

- [ ] **If** CSS preprocessors are being used – take a note of mixins and variables used to generate text styles and try to match them with styles in the inventory

> In modern front end development CSS preprocessors are often used to generate ready text styles through the usage of mixins (e.g Sass mixins). Referring these simple functions in the inventory helps you communicate with developers and becomes very handy for implementation of a new typographic scale.

Create the Icons Inventory

Icons provide the necessary context for user experience and speed up the recognition of key actions in most of the interfaces. Inconsistent usage of icons can lead to extreme confusion of users and increase the maintenance cost. By creating an inventory of icons, you can help your team and stakeholders understand the pain of not having a design system in place.

- [ ] Identify all the icon libraries used across the product

- [ ] Mark inconsistencies across the icons (e.g. two icons from different families used in one UI, two similar icons assigned to different actions, two different icons assigned to the same action etc.)

- [ ] Check if there are different ways of implementing icons across the product portfolio

> There are multiple ways to use icons. Inline SVGs, SVGs as data-urls, icon fonts – talk to the development team, or browse through the code to understand which methods are used in your design and development ecosystem. If different methods are in use, it's worth taking a note since this is likely to increase inconsistencies and further slow down the software development process. You can learn about different methods of implementing icons and their efficiency, here.

Create the Space Inventory

Space is the key ingredient of any UI and designers and developers have to manage it efficiently. Add to your inventory different kinds of grids used across the products and perhaps also dive deep into paddings in the containers to understand any inconsistencies. Learn more about space in design systems: here.

- [ ] List and document grid systems used across the product portfolio (if there are any)

Get the Support of the Organization

Before you'll move forward with the system, you need the support of your team and stakeholders.

Presentation for the team

- [ ] Describe the process of building the interface inventory
- [ ] Present the key inconsistencies from every category
- [ ] Explain why these inconsistencies are detrimental to the experience of users or are negatively affecting the software development process
- [ ] Present building a design system as the answer to the challenge
- [ ] Explain that building a design system is an ongoing process and that you would like to build a small team which to manage this process

- [ ] Clarify that all designers and developers will have the right to contribute to the system

- [ ] Get a clear ‘yes’ from the team

Presentation for stakeholders

- [ ] Describe the process of building the interface inventory

- [ ] Present the key inconsistencies from every category

- [ ] Focus on numbers and the influence on the speed of the process (62 shades of gray, 16 types of buttons, 5 icon libraries etc.)

- [ ] Explain why these inconsistencies are detrimental to the experience of users or increase cost of software development (hours involved multiplied by average salary)

- [ ] Present building a design system as the answer to the challenge

- [ ] Explain that building a design system is an ongoing process and you’d like to build a small team to manage this process
  Emphasize that building a design system will help the company deliver better experience to the market faster

- [ ] Get a clear ‘yes’ from the team

Build a Multidisciplinary Systems Team

You can build the inventory on your own, but you need a team to run the design system as an ongoing process. The team not only plans the system, but also build it, implements it, and promotes it across the team. Start by planning who you need to efficiently correct inconsistencies listed in the inventory and then consider what skillset will help manage this process long term. Initially, you might not have a team of full-time contributors, so be ready to manage the team with partial time allocation. Learn more about building Design Systems teams here.

- [ ] List all the skills needed to successfully fix inconsistencies listed in the inventory
- [ ] List all the skills needed for a long term success of the systems (maintenance + governance)
- [ ] Find people with the necessary skills

> You'll probably need well-rounded designers and front-end engineers, but some teams also need help of DevOps to implement certain tools in the build process, or PMs to run a sprint. There's no one correct structure, so you'll have to think about your team, product, and external constraints.

- [ ] Check the realistic time allocation that you can get from your team

> Many organizations, right at the beginning, do not commit employees full time to the team running the design system process. You need a realistic time allocation that you can expect from every member of the team.

- [ ] Clarify the roles on the team and the decision making process

> Make sure that the team understands who's leading the team, who's making the key decisions about different aspects of the system and how the overall decision making process will look like.

- [ ] Decide the length of the sprint

- [ ] Decide when the team will meet for planning and post-sprint meetings

Finalize by bringing clarity to the agile process needed for a design system. Are you going to work in weekly or two-week long sprints? When is the planning meeting going to happen? How are you going to summarize the sprint? When are the planned retrospectives?

Make Key Decisions and Establish Key Rules and Principles

Before your first sprint, gather the team and discuss the most important decisions that have to be made as part of the Design Systems kick-off. After finding a solution to every problem, make sure that the entire company understands where you stand.

- [ ] Decide whether you're building the system from scratch or treating one of the products as the foundation of the system.

> Some teams like to tie a kick off of a design system process with the redesign of a product. Others start with a standard that should be improved and distributed across the product portfolio. If there's a new part of the product, tested with users, that you feel should serve as a standard – start with it and expand your approach. This is the approach that we've used at UXPin, when the newest part of our UI became the foundation for our design system.

- [ ] Decide whether you're going to build using existing technologies or introduce a new technology.

> The design system might use a tech stack used across the entire product portfolio, in one of the products or some completely new technology. Naturally, the less popular the technology, the more difficult it's going to be to implement the system and get teams on board. An example would be – writing a React.js based system with CSS in JS, when a company uses Angular with Less preprocessor. Opinions about what's the right approach vary, so please discuss your situation with the team and commit to one approach. At UXPin, we're building the system on React.js and Less, which means that the Angular part of the application will have to be eventually refactored.

- [ ] Decide how you're going to distribute the system.

> Distribution of the system is crucial. Are you going to start with one team and one product? Or work across the product portfolio and take care of particular features? Make a decision and commit to one approach.

- [ ] Decide what are the KPIs of the system

> Discuss the measurable goals for the system. You can read more here.

- [ ] Formulate your design principles

> Design principles can be a useful tool to bring the entire team on the same page with shared values. What are you trying to achieve with the system? Higher consistency of interactions?

> Better craftsmanship? Faster implementation? More accessible UIs? Discuss and list the common principles.

- [ ] Communicate key decisions, rules and principles to the entire company

> Gather all the decisions, rules and principles and communicate them clearly to the rest of the company. Consider putting them all in a wiki, or listing directly in the documentation of your design system (for example in UXPin).

Build the Color Palette

Kick off your design system process with sprints devoted to unifying and implementing the color palette. Colors affect all the parts of the system, so you have to organize them first.

- [ ] Use the color inventory to identify the primary/base colors

> To identify base colors, check which colors are associated with the brand and which colors are the most prevalent in the UI. Aim at forming the full palette in which there is no accent colors not associated with a primary color and there's no color in the UI not represented in the palette.

- [ ] Decide on the naming convention

> There are different approaches to naming colors in a design system. You can name colors using abstract names (e.g. #b9b9b9 – pigeon), actual names (e.g. #b9b9b9 – silver), numbers (e.g. #b9b9b9 – silver-1) or functional names (e.g. #b9b9b9 – silver-base). At UXPin, we've decided to go with the functional approach. Names are also used as variables in preprocessor files.

- [ ] Decide on the system of building accent palette colors

> You can build the palette of accent (secondary) colors by making a series of arbitrary decisions (which in some cases might be very time consuming and difficult) or use some sort of a functional approach. At UXPin, we've used LESS functions (darken and lighten) to derive accent colors from base colors. For example a 15% darker color than our base-silver (#f3f3f3), would be called @silver-darken-15 (#CDCDCD). This approach let us build an extensive palette with human readable, clean, naming.

- [ ] Test the color palette against the colors in the inventory

> To assure your new palette will serve the entire design team well, make sure you either match, replace or merge all the colors used in the current version of the UI. There should be no color in the UI outside of the color palette as stated in the design system.

- [ ] Implement new color palette in CSS (consider using a preprocessor and build a list of variables) on a test server

- [ ] Test how the new palette affects the interface

> Plan an extensive QA checking how the new palette affected the UI.

- [ ] Check the contrast between colors in the new UI. Make sure you comply with WCAG guidelines

> Make sure that after the introduction of the new palette all the elements of the UI still have sufficient contrast as regulated by the Web Content Accessibility Guidelines.

- [ ] Present the new palette and the UI affected by the palette to all the product designers

- [ ] Invite product designers to test the interface and suggest changes to the palette

> Let the product designers participate in the process by testing the new palette and suggesting changes. You need them onboard to assure them that the new palette is actually going to be implemented and used in the future projects. Remember – design system not used by the product team is dead and useless.

- [ ] Finalize the palette

> After tests and gathering feedback, finalize the palette and communicate it to the company. Add the palette to your design system documentation (e.g. in UXPin).

- [ ] **If** you're using variables in a CSS preprocessor, make sure that names of variables are documented in the documentation of the design system.

- [ ] Deliver the new color palette to the tools used by product designers

> Make sure that the new palette is going to be implemented in design tools used by your team (UXPin, Sketch, etc). For example, you can use the UXPin design systems library to assure and manage consistency.

Build the Typographic Scale

The next part of the system that affects all the other parts is typography. Not only do you have decide which typefaces will become a standard, but also you have to build a consistent scale, which is going to build a predictable information architecture across the product portfolio.

- [ ] Choose the approved typefaces

- [ ] Build a consistent typescale

> There are different approaches to building a typographic scale (read more). You can optimize the scale to serve existing styles, or you might try to build a harmonious scale using the golden ratio or major second. Ultimately though your goal is to build a scale that will be implemented and will fix the existing and future inconsistencies. When building the scale, don't forget that you're not only setting the size of the font, but also weight, line-height and other properties.

- [ ] **If** you're using CSS preprocessors Build Mixins generating text styles

  > Mixins (in Sass, LESS) can be a terrific tool for managing the typographic scale and generating the right styles in code.

- [ ] Test the new typographic scale against the text styles in the inventory

  > Test whether you covered all the existing styles. You should either match, replace or merge existing styles, so the new scale can be implemented across the UI without leaving any text-styles outside of the scale.

- [ ] Implement the new typographic scale palette in CSS (consider using a preprocessor and building mixins) on a test server

  > Build a test environment for the new scale.

- [ ] Test how the new typographic scale affects the interface

  > Check how the new scale affected the UI. Is everything easily readable? Are all the text styles covered by the new scale? Does the new scale reinforce the right information architecture?

- [ ] Present the new typographic scale and the UI affected by the scale to all the product designers

- [ ] Invite product designers to test the interface and suggest changes to the scale

> Just like in the case of the color palette – involve product designers in the process. Invite them to review the scale and the UI affected by it. Ask them for feedback.

- [ ] Finalize the typographic scale

> After gathering the feedback, testing and iterating on the scale – finalize it, make part of your design systems documentation and communicate it to the company.

- [ ] **If** you're using mixins, make sure that names of mixins are documented in the documentation of the design system.

- [ ] Deliver the new typographic scale to the tools used by product designers

> Make sure that the scale is reflected in design tools used by your team (UXPin, Sketch....). You can use UXPin's Design System Libraries to save all the text scale and make them easily actionable in UXPin and Sketch.

Implement Icons Library

Icons are a very important part of the visual language and should become part of the system. Finalize the library, choose the implementation method and make sure that all team members have easy access to all icons.

- [ ] Decide which icons from the interface inventory should become part of the system

- [ ] Decide which method of managing icons should be used to implement icons in a design system

  > Discuss with the team the ideal method of managing and implementing icons in your design systems. You can learn more about different options and their efficiency here.

- [ ] If you're changing the technology used to implement icons, implement icons on a test server and thoroughly test

- [ ] Finalize the icons library to be used in a design system and add it to design system documentation

- [ ] Deliver icons to tools used by product designers

> Make sure that icons are easily accessible from design tools used by your team (UXPin, Sketch….). You can use UXPin's Design System Libraries to save all the icons in an easy-to-manage toolkit available in Sketch and UXPin.

Standardize other style properties

The process of adding grid, space definition, and basic style properties is going to be nearly identical with the process described for standardizing color palette, typography, and iconography. When running it in your sprints make sure that:

- ☐ Every standard part of the system solves problems of inconsistency as presented in the inventory
- ☐ Every change is thoroughly tested on a test server before the implementation
- ☐ Testing involves product designers
- ☐ Finalized part of the system becomes part of systems' documentation and is communicate to the entire company
- ☐ Every part of the system is available directly in tools used by product designers (Sketch, UXPin, etc.)

Build the First Design System Pattern

Once all the building blocks of the system are built, tested, implemented, and accepted by product designers and developers, you can start building up the patterns. This is the task that will never end. Patterns should always either reflect the truth about the product, or reflect the aspirational state of the product in the near future. Products constantly evolve and so should patterns in the design system. Don't aim at finalizing all the patterns in a single sprint or even a series of sprints. Take them one by one and make sure they are implemented by product teams so you can gradually correct all the inconsistencies and increase the speed of product development.

- [ ] Decide the best architecture for the patterns in your design system

> You can use different models of architecture, or create your own. A popular way of organizing patterns in a system is Atomic Web Design. At UXPin, we've divided our patterns into elements, components and modules, which reflects the modular architecture of our front-end code. No matter your naming convention, make

> sure the architecture is modular and vocabulary is understood by the design systems team and the product team.

- [ ] Choose one pattern you'll work on during this sprint (example – button)

- [ ] Make sure that the pattern is using the correct color, typography, iconography etc. Make changes if necessary.

- [ ] Review the code. If necessary, make changes to make sure that the component rendering the pattern is fully encapsulated and can be implemented in any part of the product.

- [ ] Review the code again. Make sure that all the coding standards are given justice. Make changes if necessary

- [ ] Ask members of the design system and developers working on the product for a code review

- [ ] List all the patterns that are going to be replaced by the pattern in the system

- [ ] Consult product developers and designers and see if there are any suggested changes

- [ ] Implement the pattern on a test server and thoroughly test with designers and developers from the product team

- [ ] Finalize the pattern
- [ ] Add pattern to your design systems documentation
- [ ] Document the usage and implementation guidelines
- [ ] Test the documentation with members of the product team to make sure that everything is easy to understand
- [ ] Make sure that the pattern is available in design tools used by product designers (Sketch, UXPin)

> Make sure that the pattern is easily accessible from design tools used by your team (UXPin, Sketch….). You can use UXPin's Design System Libraries to save patterns as symbols.

Run a Sprint Retrospective

You should organize a regular retrospective of your design systems sprints. Make sure the team keeps learning and improving.

- [ ] Summarize the KPIs for first sprint

Questions for Team Discussion

Ask team: what did we do well?
Ask team: what did we learn?
Ask team: what should we do differently next time?
Ask team: what could help us be successful as a team?

Questions for Individuals Sharing

Ask: how did you do this sprint?
Ask: what is your biggest impediment?
Ask: if you could change one thing, what would that be?
Ask: what was keeping you awake at night?

Congratulations!

Bravo! If you got to this place on a checklist, you're running the design systems as a sustainable project. Hopefully you're already seeing increases in software development velocity and user satisfaction. Keep it up!

> *A design system is a process and therefore is simultaneously always ready and never done.*

Recommended Reading List

Books

The Timeless Way of Building by Christopher Alexander
A Pattern Language by Christopher Alexander
Modular Web Design by Nathan Curtis
Atomic Design by Brad Frost
Frontend Architecture for Design Systems by Micah Godbolt

Articles

Nathan Curtis
Design Systems Leaders and Managers
The Principles of Designing Systems
What Will Your Design Systems Deliver?
Contributions to Design Systems
Component QA in Design Systems
Patterns ≠ Components
Light & Dark Color Systems

Reference Designs for Systems
Tokens in Design Systems
Buttons in Design Systems
Color in Design Systems
Picking Parts, Product, and People: A Team Activity to Start a Design System
Right-Sizing the Rectangle: Grappling With Hierarchy in Design Systems
A Design System Isn't a Project, It's a Product Serving Products
Reduce, Reuse, and Recycle
A Design System's Reach
The Component-Cut Up Workshop
Balancing Platforms in a Design System

Marcin Treder

Design Systems Sprint 0: The Silver Bullet of Product Development
Design Systems Sprint 1: The Interface Inventory
Design System Sprint 2: One Color Palette to Rule them All
Design System Sprint 3: Managing the Basics
Design System Sprint 4: Design Principles
Design System Sprint 5: Managing Typography
Design System Sprint 6: The Fastest Icons on Earth
The Minimum Viable Design System
Design Systems Are a Language. Product Is a Conversation

Brad Frost

Atomic Design: The Online Guide

Dan Mall

Researching Design Systems

Selling Design Systems

Cooking With Design Systems

WeWork

Selling a Design System at Your Company

The Plasma Design System

Salesforce

Living Design Systems

The Lightning Design System is the Next Generation of Living Style Guides

Introducing Design System Ops

React JS and the Lightning Design System

Airbnb

Design Ops at Airbnb

Building a Visual Language

The Way We Build

Spotify

Design Doesn’t Scale

GE

The Predix Design System

Intuit

Intuit Design System Overview

Bottomline Technologies

Creating and Scaling Enterprise Design Systems

Beautiful Seams: Creating a Coherent Experience Across Products

Other:

How to Construct a Design System

Design Systems in 2016

The Current State of Design Systems

Design Systems and Postel's Law

How Designers Can Use Unit Testing to Build Resilient and Happy Design Systems

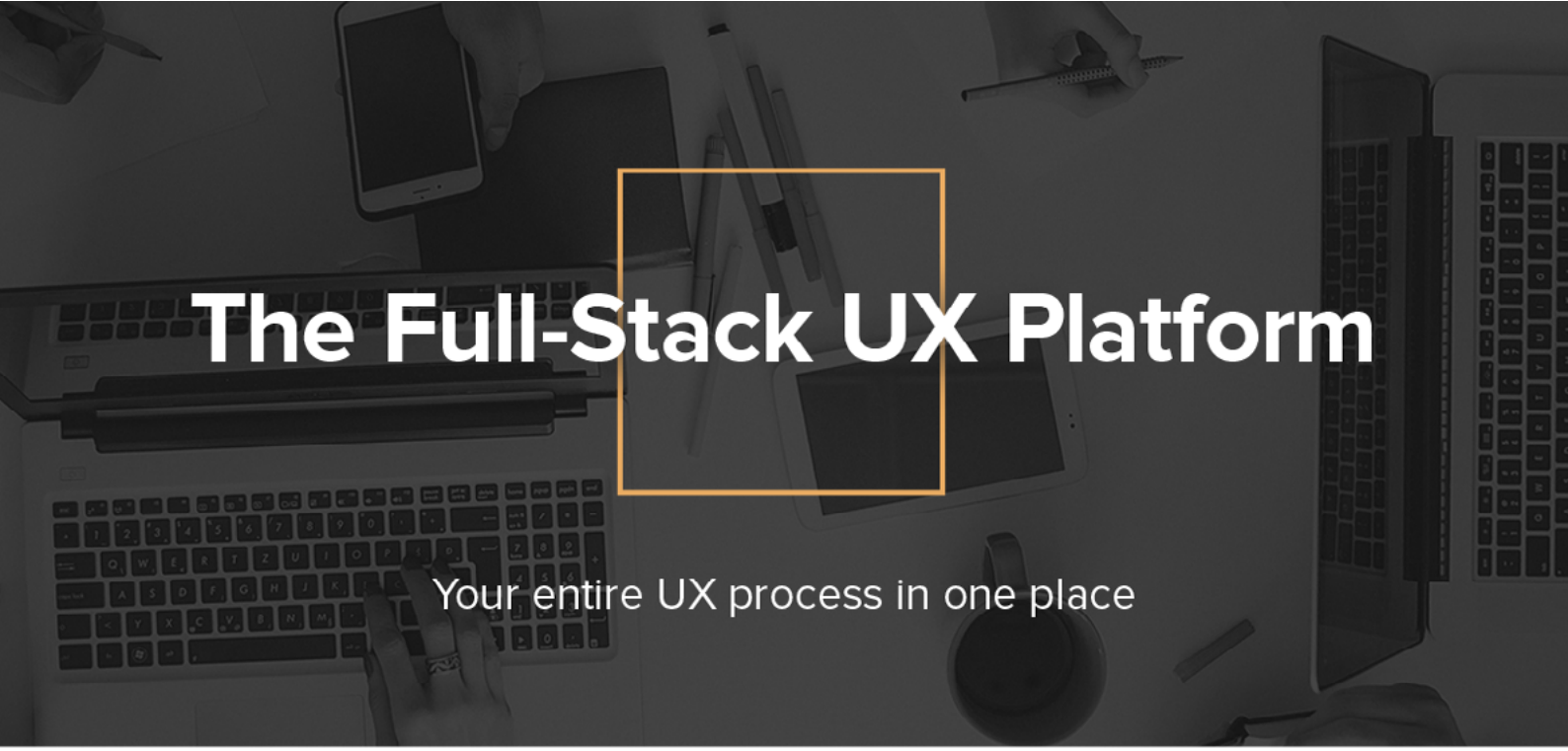
What is Design Operations and Why Should You Care?

Component Workshops: Our One-Two Punch for Kicking Off a New Design System

Object-Oriented UX

The Most Exciting Design Systems Are Boring

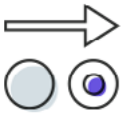
The Full-Stack Design System



The Full-Stack UX Platform

Your entire UX process in one place

Design:
Create lifelike prototypes quickly with Photoshop and Sketch integration.



Iterate:
Built-in version control improves efficiency and eliminates confusion.

Document:
Cleanly annotate your designs. Insert custom code snippets that travel with elements.

Collaborate:
Get feedback and co-design on any project anywhere.

Scale:
Automate consistency and documentation with design systems (syncs with Sketch).

Implement:
Auto-generate style guides, assets, and specs for developers.

Try UXPin now