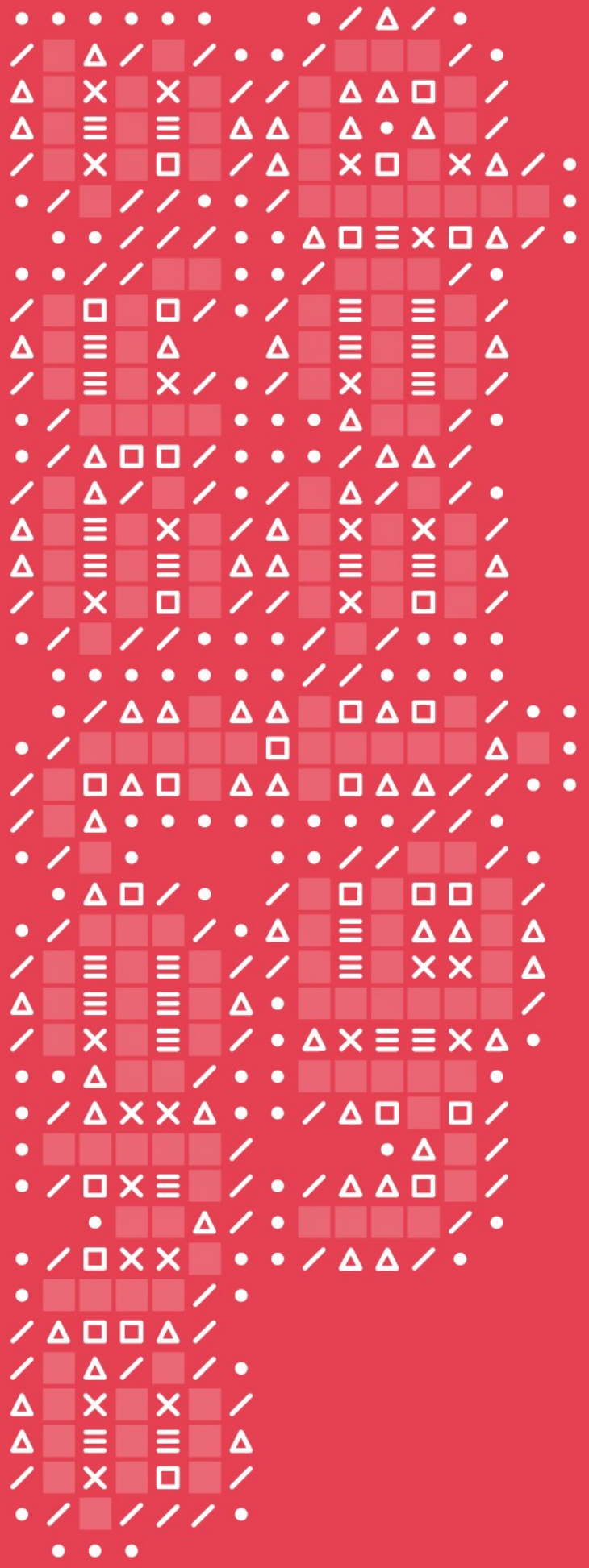
# Design Systems

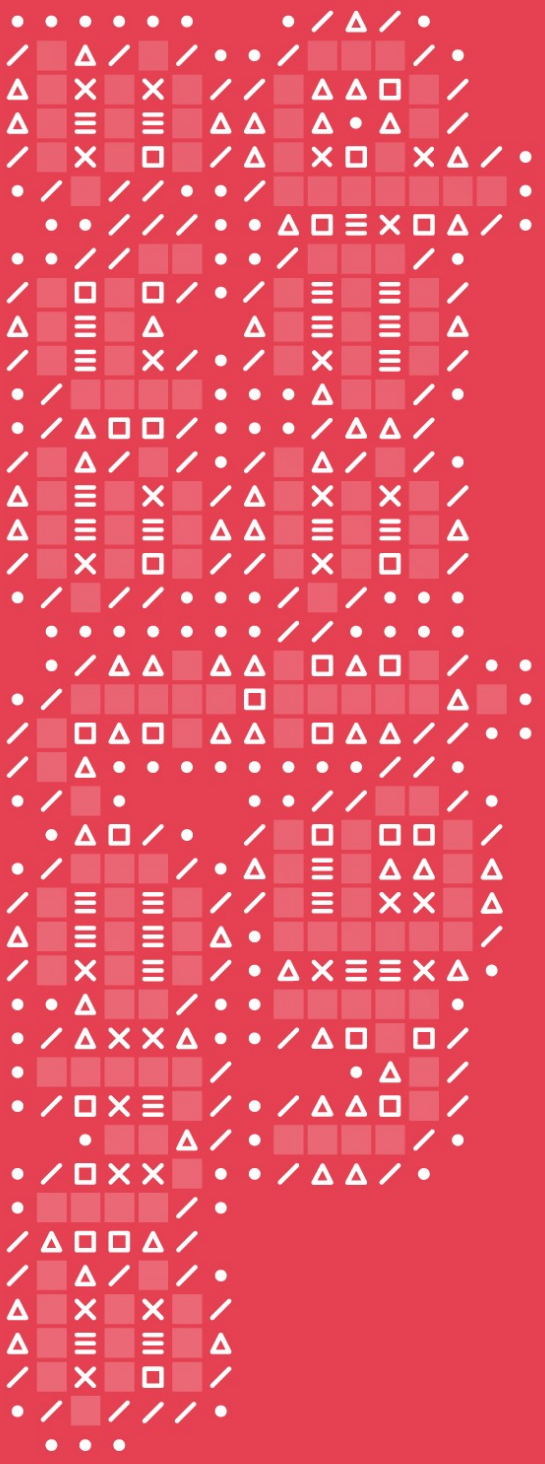
A practical guide to creating design languages for digital products.



by Alla Kholmatova

# Design Systems

A practical guide to creating design languages for digital products.



by Alla Kholmatova

# Imprint

Published 2017 by Smashing Media AG, Freiburg, Germany.

ISBN (ePUB): 978-3-945749-59-3

Cover Design: Espen Brunborg.
Proofreading: Owen Gregory.
eBook production: Cosima Mielke.
Print layout: Markus Seyfferth.

*Design Systems* was written by Alla Kholmatova and reviewed by Karen McGrane and Jeremy Keith.

Please send errors to: errata@smashingmagazine.com

*To Alyona*

# Table Of Contents

# About The Author

Alla Kholmatova is a UX and interaction designer with a nine-year experience of working on the web, for a range of products and companies. Most recently she was a senior product designer at an open education platform, FutureLearn.

She’s particularly interested in design systems, language, and collaborative ways of working. In the last two years this interest has led her to spend a huge amount of time working on and researching the subject. She’s been sharing her insights with people through articles, workshops, and projects. Alla contributes to design publications, such as A List Apart, and speaks at conferences around the world.

# About The Reviewers

*Karen McGrane* has helped businesses create better digital products through the power of user experience design and content strategy for the past twenty years. She is Managing Partner at Bond Art + Science, a UX consultancy she founded in 2006 and formerly VP and National Lead for User Experience at Razorfish. Karen teaches Design Management in the MFA in Interaction Design program at the School of Visual Arts in Manhattan. She co-hosts A Responsive Web Design Podcast with Ethan Marcotte, and her first book, *Content Strategy for Mobile*[1], was published in 2012 by A Book Apart.

*Jeremy Keith* is co-founder and technical director at Clearleft, a digital design studio based in Brighton, England. When he's not making websites, he talks at conferences about making websites. Sometimes he even writes books about making websites, like the web book at ResilientWebDesign.com that you can have for free. But he mostly spends his time goofing off on the internet, documenting his time-wasting on his website adactio.com where he has been writing for over fifteen years.

—

1. http://smashed.by/contentmob

# Foreword

If you have a moment, look up the work of artist Emily Garfield. She creates exquisite, intricately detailed maps in watercolor — each of them stunning, and each of them depicting a place that doesn't exist. Instead of depicting a city's real, actual landscape, she begins by creating a single, complex pattern — a knotted road or twisty river, or a compact grid of city blocks — and repeating it. Garfield iterates on that pattern, changing it slightly each time, spiraling out until her not-map is finished. As a result, her art has a generative, fractal-like quality: it's built from patterns, yes, but feels part of a cohesive whole.

In fact, Garfield once said, "I describe my process as *growing the drawing*." While reading this wonderful book by Alla Kholmatova — this book that *you're* about to read — I thought about that line a lot. Maybe you will, too.

In recent years, web designers have started embracing more modular, pattern-driven design practices. And with good reason: we're being asked to create compelling experiences for more screens, more devices, more places, more *people* than ever before. As a result, we've started to break our interfaces down into tiny, reusable modules, and begun using those patterns to build products, features, and interfaces more quickly than ever before.

But by themselves, design patterns aren't enough. They need to live within a larger process, one that ensures these little interface modules feel unified, cohesive, connected. Part of a whole. In other

words, they need a *design system* to thrive — and that's where Alla's book comes in.

In these pages, Alla shows us precisely *how* to create systems to support our digital designs. With clear writing, case studies, and detailed examples, Alla shows us how to establish a common, shared language among our teams, which allows us to more effectively collaborate; she'll tell stories of how different organizations have created their design systems, and put them into practice; and she'll discuss different models for evolving these systems over time.

In other words, this isn't just a book. Alla has drawn a clear, bright map for us, one that outlines a more sustainable model for digital design. And if we walk the paths she's drawn for us, we'll learn to grow better design systems — and with them, better designs.

*— Ethan Marcotte*

# Introduction

## What This Book Is About

As the web continues to change rapidly and become more complex, thinking of it in terms of static pages has become untenable, and many of us have started to approach design in a more systematic way.

And yet not all design systems are equally effective. Some can generate coherent user experiences, others produce confusing patchwork designs. Some inspire teams to contribute to them, others are neglected. Some get better with time, more cohesive and better functioning; others get worse, becoming bloated and cumbersome.

What are the key qualities of a well-functioning, enduring design system? This question intrigued me so much I spent a huge amount of time researching and thinking about it. My research and thoughts provide the basis of this book. Drawing on the experience of companies of various sizes and approaches to design systems, I set out to identify what makes an effective system that can empower teams to create great digital products. Throughout the book I'll share an approach that helps me every day with my work. I hope this will help with your work, too.

## Who This Book Is For

This book is aimed mainly at small and medium-sized product teams trying to integrate design systems thinking into their organization's culture. Everyone in the product team could benefit from reading this book, but particularly visual and interaction designers, UX practitioners and front-end developers.

## Scope Of The Book

This book presents a perspective on design systems based on my experience as an interaction and visual designer. I don't touch on other related areas, such as information architecture, content strategy or design research. Equally, this is not a technical book. You won't find any code samples or in-depth analysis of development tools and techniques, although there will be plenty of discussion directly related to front-end practices.

This is a design book, but it isn't about what to design. Neither is it an attempt to create a comprehensive guide to designing digital products.[1] It is about how to approach your design process in a more systematic way, and ensure your design system helps to achieve the purpose of your product and fits with the culture of your team.

## How This Book Is Organized

The book has two parts.

### PART 1: FOUNDATIONS

In the first part we'll talk about the foundations of a design system — *patterns* and *practices*. Design patterns are repeatable, reusable parts of the interface, from the concrete and functional (like buttons and text fields) to the more descriptive (like iconography styles, colors and typography). Patterns interconnect, and together they form the *language* of your product's interface. Shared practices are *how* we choose to create, capture, share and use those patterns — by following a set of principles, or by keeping a pattern library.

### PART 2: PROCESS

A design system cannot be built overnight — it evolves gradually with your product. But there are certain principles and practices that we can follow to make sure the system develops in the right direction and provide us some degree of control over it. The second part of the book focuses on practical steps and techniques to establish and maintain a design system, including: planning the work; conducting an interface inventory; setting up a pattern library; creating, documenting, evolving and maintaining design patterns.

# Terminology

Before we dive into the topic, let's establish the terms we'll use throughout the book.

**Pattern** or **design pattern**
I use the word *pattern* to refer to any repeating, reusable parts of the interface (such as buttons, text fields, iconography styles, colors and typography, repeating user flows, interactive behaviors) that can be applied and repurposed to solve a specific design problem, meet a user need, or evoke an emotion. Throughout the book, I distinguish between functional patterns related to *behaviors*, and perceptual patterns related to *brand* and *aesthetics*.

**Functional patterns** or **modules**
These terms are used interchangeably throughout the book, to refer to tangible building blocks of the interface, such as a button, a header, a form element, a menu.

**Perceptual patterns** or **styles**
These are more descriptive and less tangible design patterns, such as iconography styles or colors and typography, typically used to create a certain kind of aesthetic and strengthen an emotional connection with a product.

**Pattern language** or **design language**
A set of interconnected, shareable design patterns forms the *language* of your product's interface. A pattern language combines functional and perceptual patterns, as well as platform-specific patterns (such as the hamburger menu), domain patterns (such as

modules specific to an e-commerce site, or finance software, or a social app), UX and persuasive patterns, and many other types meshed together in an interface for a specific product.

**Design system** or **system**
There isn't a standard definition of "design system" within the web community, and people use the term in different ways — sometimes interchangeably with "style guides" and "pattern libraries." In this book, by design system I mean a set of *connected patterns* and *shared practices*, coherently organized to serve the purposes of a digital product.

**Pattern library** and **style guide**
A pattern library is a tool to capture, collect and share design patterns and guidelines for their usage. Creating a pattern library is an example of a (good) design practice. Traditionally, a style guide focuses on styles, such as such as iconography styles, colors and typography, while a pattern library includes a broader set of patterns.

## Design System Insights

The book is based on practical insights from real-world products. Most of them are drawn from my experience of working at FutureLearn[2], a medium-sized open education company in London. During my three years working there as a designer, I have had an opportunity to observe and influence how a design system evolves, from initial concepts to a mature system[3].

To have more depth and diversity in the research, I also closely followed five other companies of different sizes and approaches to design systems: Airbnb, Atlassian, Eurostar, Sipgate, and TED. Over the course of 18 months I've been interviewing members of their teams, to understand directly from the team, the challenges they face as their systems evolve. The companies who have kindly agreed to share their insights are as follows.

## AIRBNB[4]

When interviewed in August 2016, Roy Stanfield (Principal Interaction Designer) gave me plenty of detail about the Design Language System[5] at Airbnb.The distinguishing aspect of DLS is its strictness. Patterns are specified and used precisely and rules are followed closely. The team has placed a number of practices and tools in place to achieve that. They still have some challenges with adoption, speed of integrating new patterns, and with keeping art direction and engineering in sync.

## ATLASSIAN[6]

Jürgen Spangl (Head of Design), James Bryant (Lead Designer), and Kevin Coffey (Design Manager) shared their perspectives on ADG (Atlassian Design Guidelines[7]) in November 2016. While there's a dedicated team who curates the patterns, they also have an open source model for contributions. Everyone in the company is not only allowed, but actively encouraged to contribute to the system. The challenge with this model is to find a balance between giving people

enough freedom to contribute, yet making sure the system stays managed and curated.

## EUROSTAR[8]

Dan Jackson (Solutions Architect) was very forthcoming in August and September 2016 and in March 2017 about what they've been doing at Eurostar. At the time of writing, the team was in the process of building their first pattern library[9]. They initially experienced some challenges, particularly with prioritizing the project and encouraging everyone on the team to contribute. After a year, they were given the resources to allocate a dedicated team, which is now leading the work on the system.

## SIPGATE[10]

Tobias Ritterbach (Experience Owner) and Mathias Wegener (Web Developer) both gave me a lot of insight into their work in August 2016 and November 2016. The Sipgate pattern library[11] was established in 2015, but after a year the team found that there were too many patterns, mainly due to a lack of communication between the product teams. More recently, they were in the process of working on a new pattern library, with a goal to unify the design language across several product websites.

## TED[12]

Michael McWatters (UX Architect), Aaron Weyenberg (UX Lead) and Joe Bartlett (Front-End Developer) all provided input into

discussions in August and September 2016. Among the many people who support TED.com, a small handful of UX practitioners and front-end developers are responsible for design system decisions. The team has a deep shared knowledge of their patterns, which are documented in a simple way[13]. So far they haven't felt a need to establish a comprehensive pattern library.

## Acknowledgements

I want to thank everyone at FutureLearn for their support of this book, in particular: Lucy Blackwell, for reviewing the early drafts and for guiding and inspiring me to do my best work; Mike Sharples, for the thought-provoking feedback on the early draft and for challenging me; Gabor Vajda, for helping me to shape many of the ideas described in the book; Jusna Begum, for bringing some order and structure to my chaotic thoughts; and Sam McTaggart, Dovile Sandaite, Kieran McCann, Storm MacSporran, Katie Coleman, Nicky Thompson, James Mockett, Chris Lowis and Matt Walton, for taking the time to listen and for sharing their feedback.

Huge thanks to the Smashing crew and everyone who helped me shape this book and make it happen. A special thanks to Karen McGrane, Jeremy Keith and Vitaly Friedman, for the thoughtful and constructive feedback which made this book so much better; Owen Gregory, for editing the book; to Ethan Marcotte for the foreword; and Espen Brunborg for the beautiful cover design.

I would particularly like to thank the many people who kindly agreed to share their experiences and perspectives, many of which contributed to the material in the book: the teams mentioned in Design Systems Insights, as well as Sarah Drasner, Laura Elizabeth, Matt Bond, Trent Walton, and Geri Coady, and Joel Burges, Michal Paraschidis, Heydon Pickering, Léonie Watson, Bethany Sonefeld and Chris Dhanaraj (IBM), Amy Thibodeau (Shopify), and Joe Preston (Intuit).

Finally, I want to thank my family: my husband, Hakan, and my little daughter, Alyona, for the patience and understanding they gave me in the 18 months it took to reach a final draft. Writing a book while having a full-time job was an enormous amount of work and it would have been impossible without my husband's support. I'm sorry, Alyona, for all the times I couldn't play with you because I was busy working. I love you and I promise to make up for it!

—

1. For that I recommend *About Face: The Essentials of Interaction Design* by Alan Cooper; *Lean UX: Applying Lean Principles to Improve User Experience* by Jeff Gothelf and Josh Seiden; and *Designing for the Digital Age: How to Create Human-Centered Products and Services* by Kim Goodwin.
2. https://www.futurelearn.com/
3. https://www.futurelearn.com/pattern-library
4. https://www.airbnb.co.uk/
5. http://smashed.by/airbnblanguage
6. https://www.atlassian.com/
7. https://atlassian.design/

8. http://www.eurostar.com/
9. https://style.eurostar.com/
10. https://www.sipgate.de/
11. https://design.sipgateteam.de/
12. http://www.ted.com/
13. http://ted.com/swatch

# Part 1: Foundations

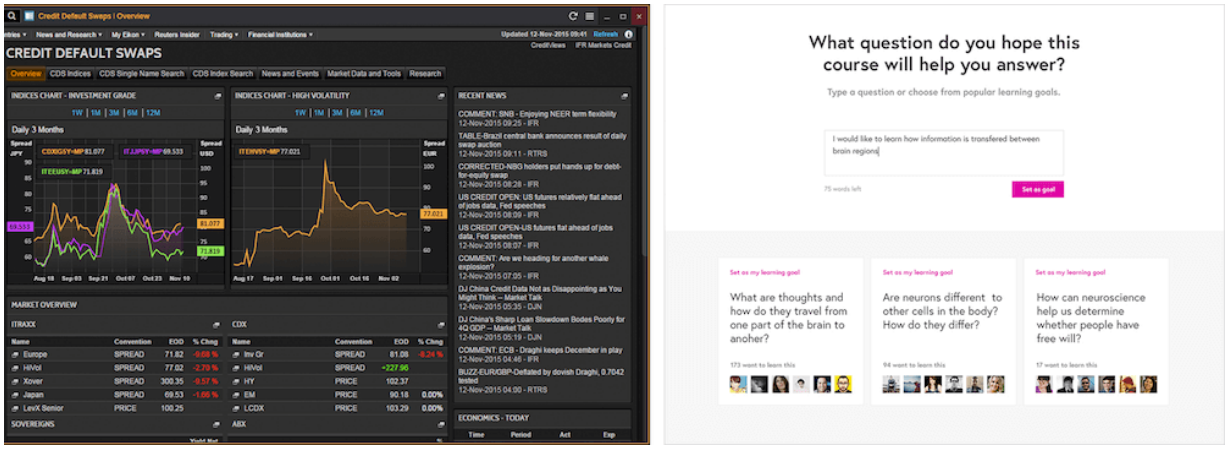
CHAPTER 1

# Design Systems

*There isn't a standard definition of "design system" within the web community and people use the term in different ways. In this chapter, we'll define what a design system is and what it consists of.*

A design system is a set of *interconnected patterns* and *shared practices* coherently organized to serve the purpose of a digital product. Patterns are the repeating elements that we combine to create an interface: things like user flows, interactions, buttons, text fields, icons, colors, typography, microcopy. Practices are *how* we choose to create, capture, share and use those patterns, particularly when working in a team.

Take a look at these two screens of unrelated products. One is from Thomson Reuters Eikon, a trading and market analysis application; the other is from FutureLearn, an open education social learning site.



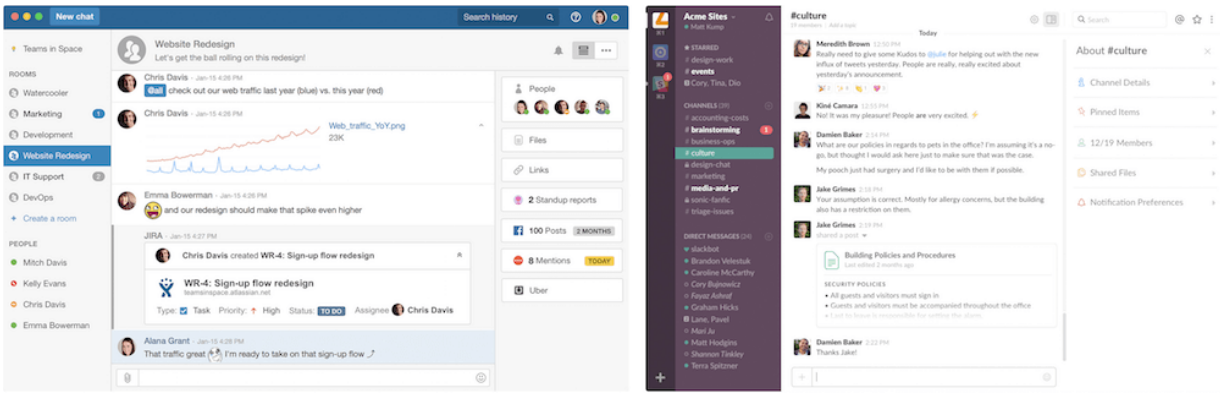

*A screen from Thomson Reuters Eikon (left) and a screen from FutureLearn (right).*

In each example the patterns work together to achieve different purposes. For Thomson Reuters, it's about handling data, utility, quick scanning and multitasking; for FutureLearn, it's about thoughtful reading, informal learning, reflection and connecting with like-minded people. The *purpose* of the product shapes the design patterns it adopts.

The Thomson Reuters layout is panel- and widget-based, to allow users to multitask. The design is dense, fitting large amounts of information on the screen. Density is achieved through tight spacing, compact controls, flexible layouts and typographic choices, such as a condensed typeface and relatively small headings. On the other hand, the FutureLearn layout is much more spacious. Each screen is typically focused on a single task, such as reading an article, engaging in a discussion, or completing an interactive exercise. The layout here is mostly a single column; there's high-contrast typography with large headings, chunky controls, and generous white space.[1]

The choice of design patterns is influenced by many factors. Some of them come from the *domain* the product belongs to, and from its core functionality: those are *functional patterns*. To use trading and market analysis software, for instance, you'd need to be accustomed to task bars, data fields and grids, charts and data visualization tools. For an online learning site, it could be articles, videos, discussion threads, progress indicators and interactive activities. An e-commerce site would most likely include a product display, list filters, shopping cart and a checkout.

The *ethos* of a product (or the brand, depending on your definition of a brand) also forms patterns which together shape how a product is perceived; throughout this book I'll refer to them as *perceptual patterns*. By that I mean things like tone of voice, typography and color choices, iconography styles, spacing and layout, specific shapes, interactions, animations, and sounds. Without perceptual patterns you wouldn't feel that much difference between products from within the same domain, which have similar functionality.




*Although HipChat and Slack have similar purposes and functionality, they feel quite different, largely due to how brand is expressed throughout their interfaces.*

Patterns are also shaped by platform conventions. A product can feel distinctly web-y or distinctly app-y because of a platform-specific design language. An iOS application for a product can behave and feel entirely different from its Android equivalent.

There are many kinds of patterns at play when it comes to creating a digital product. That's why design is hard. Patterns need to interact, connect, yet still work seamlessly together. Let's take a closer look at them.

# Design Patterns

The idea of design patterns was introduced by the architect Christopher Alexander in his seminal books, *The Timeless Way of Building* and *A Pattern Language*. One question that runs through the books is why some places feel so alive and great to be in, while others feel dull and lifeless. According to Alexander, the way places and buildings make us feel is not due to subjective emotions merely. It's a result of certain tangible and specific patterns. Even ordinary people can learn and use them to create humane architecture.

A pattern is a recurring, reusable solution that can be applied to solve a design problem.

---

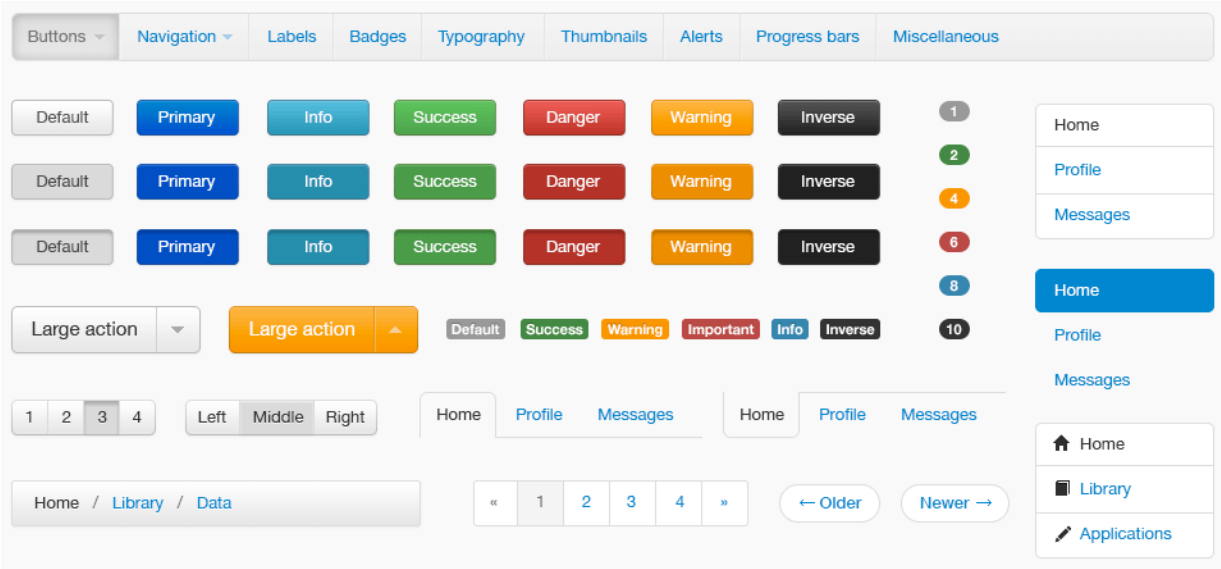
*"Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem."*
*— Christopher Alexander,* A Pattern Language

---

*A Pattern Language* contains 253 architectural design patterns, starting from the larger ones, such as a layout of a city and road systems, down to the smallest ones, such as lighting and furniture in a family house.

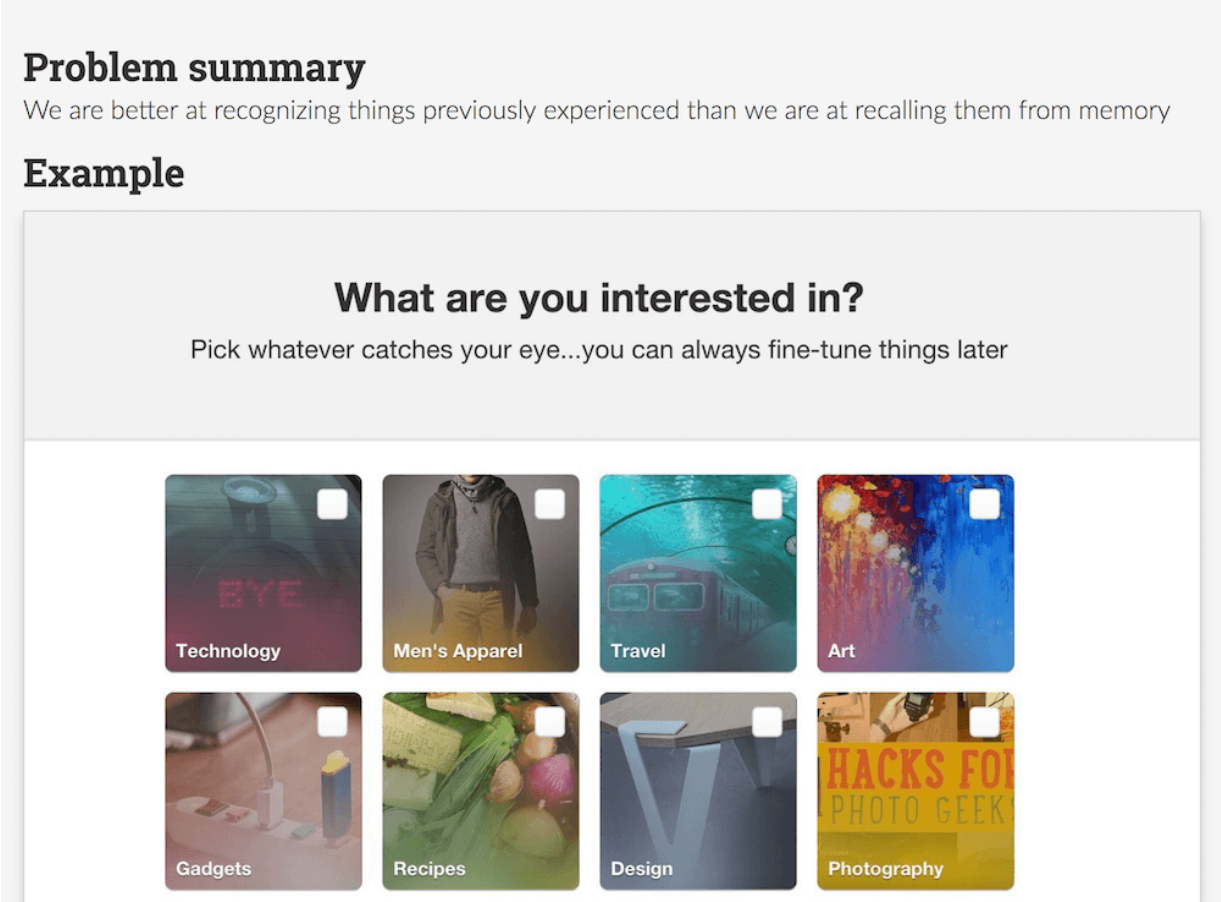
Similarly, when creating interfaces we use design patterns to solve common problems: we use tabs to separate content into sections

and indicate which option is currently selected; we use a dropdown to display a list of options at the user's request.[2]




*Some of the patterns from Bootstrap, a front-end framework for developing responsive websites.*

We use patterns to offer feedback, to show how many steps are left in a process, to allow people interact with each other, to view and select items. Design patterns can intrigue and encourage, make tasks easier, create a sense of achievement or an illusion of control.




*Example of persuasive pattern "recognition over recall" on UI Patterns*[3].

Most of the design patterns are established and familiar. They make use of people's mental models and allow design to be understood intuitively. Entirely new patterns require users to learn and adopt them first — they are relatively rare.[4] What makes a product distinct from its competitors is not the novelty of patterns it uses, but how those patterns are executed and applied, and how they interconnect to achieve a design purpose. A set of interconnected patterns forms the *design language* of your product's interface.

A design language emerges as we work on a product. We don't always know what this language is. Sometimes, effective and interesting designs are based on intuition, and we struggle to

articulate exactly how and why they work. Designers and developers might know it instinctively, but intuition is not always reliable or scalable. In his article "Researching Design Systems,"[7] designer Dan Mall noted that one of the main goals of a design system is "extending creative direction." If you need to get a group of people to follow a creative direction consistently, reliably and coherently, patterns need to be *articulated and shared*.

When you articulate your design language it becomes actionable and reproducible. You start approaching design with a system in mind. For example, instead of discussing how to tweak an item to make it stand out more, you can have a suite of promotional patterns, each one targeted to achieve a different level of visual prominence. The visual loudness guide[6] by Tom Osborne is an example of how buttons and links can be approached systematically. Instead of listing them individually, they are presented as a suite, each one having a different "volume" corresponding to its intended visual prominence.

## Loudness Guide

| LINK | TYPE | VOLUME | FREQUENCY | USE |
|---|---|---|---|---|
| Buy Now! | Graphic | Scream | Selectively | Promotional |
| Sign Up | Button | Yell | Rare | Brand, registration, help |
| Save | Button | Shout | Occasionally | Editing, actions |
| Submit | Button | Cheer | Often | Primary button |
| Cancel | Button | Murmur | Occasionally | Secondary button |
| Primary | Link | Whistle | Often | Primary text link |
| Secondary | Link | Whisper | Occasionally | Secondary text link |

Source: Viget.com

*The visual loudness guide by Tom Osborne.*

Articulating your language allows you to gain more control over the system. Rather than making small tweaks, you can affect it in much more profound ways. If you discover a small design change which made a positive impact on user experience, you can apply it to the pattern across the system rather than to one place. Instead of spending hours on designing a dropdown, you can spend that time with the users and domain experts, finding out if a dropdown is

needed in the first place. When the design language is shared knowledge, you can stop focusing on the patterns themselves and instead focus more on the user.

Throughout the book we'll talk a lot about articulating, sharing and documenting a pattern language for digital products. In particular, we'll look at two types of design patterns: functional and perceptual. Functional patterns are represented as concrete modules of the interface, such as a button, a header, a form element, a menu. Perceptual patterns are descriptive styles that express and communicate the personality of the product visually, such as color and typography, iconography, shapes and animations.

To extend the analogy with language, functional patterns are a bit like nouns or verbs — they are concrete, actionable parts of the interface; whereas perceptual patterns are similar to adjectives — they are descriptive. For example, a button is a module with a clear function: allow users to submit an action. But the typographic style in the label of the button, its shape, background color, padding, interactive states and transitions are not modules. They are styles; they describe what *kind* of button it is. From a front-end perspective, modules always have a basis in HTML, and perceptual patterns are typically CSS properties.

A design system contains many other kinds of patterns: user flows (such as completion of forms with errors and success messages), domain-oriented design patterns (like learning patterns for EdTech systems, and e-commerce patterns), and persuasive UX patterns.

The focus of this book is on the functional and perceptual patterns as the core building blocks of a design system.

But, of course, what matters is not only the patterns themselves, but how they are evolved, shared, connected and used.

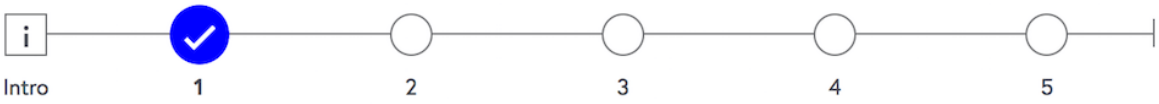
## Shared Language

Language is fundamental to collaboration. If you work in a team, your design language needs to be shared among the people involved in the creation of the product. Without a shared language, a group of people can't create effectively together — each person will have a different mental model of what they're trying to achieve. Let's return to the button example. Even such a basic unit of the interface can have different meanings. What exactly is a button? An outlined area that you can click on? An interactive element on a page that links somewhere? A form element that allows users to submit some data?

In her book *How to Make Sense of Any Mess*, Abby Covert suggests that a shared language should be established before you think about interfaces, by discussing, vetting and documenting our language decisions. This idea could be applied to describing high-level concepts as well as the day-to-day language we use to talk about design decisions. Having a shared language would mean that we have the same approach to naming interface elements and defining design patterns, or that the same names are used in design files and front-end architecture.

Even that might not be enough. Sometimes, people in a group think they have reached a mutual understanding because they share the same vocabulary and use it expressively, but they still retain fundamental differences in understanding. For example, after a year of using the term “Scenario” as a key concept in a project, you might discover that different people are interpreting it in entirely different ways. It’s not only about developing a shared language — we need also to develop a shared *use of language*. It’s not enough to have a shared understanding of the term *button*. People must also know why and how to use a button, in what contexts, and the purpose a button can serve.

Suppose we use an element called “Sequence” in the interface. By presenting it as “Sequence” we aim to communicate to users that the steps should be looked at in a specific order.




*Example of “Sequence” module.*

Ideally, everyone involved in the creation of the product should know what this element is: its name and purpose, why it’s been designed that way, and how and when it should be used.[7] The stronger this shared knowledge is, the higher the chances that it will be used appropriately. Designers and front-end developers should have this knowledge, but it helps if people from other disciplines (content, marketing, product management) have some idea too.

It should be known to everyone as “Sequence,” not “Wizard control” or “Fancy bubbles.” If designers refer to it as “Fancy bubbles,” developers call it “Wizard control” and users interpret it as set of optional tabs, then you know your language doesn’t work. Why is the user’s interpretation important? We can remember here Don Norman’s pioneering work, *The Design of Everyday Things*, where he talks about the gulf between the system image (the interface) and the user’s model (the perception of the interface formed by the user through interaction with it). If the user has a mental model of the interaction that doesn’t fit with the system image provided by the design team, then the user will be continually challenged by unexpected behavior from the system. *An effective design language bridges the gap between the system image and the (assumed) user model.*

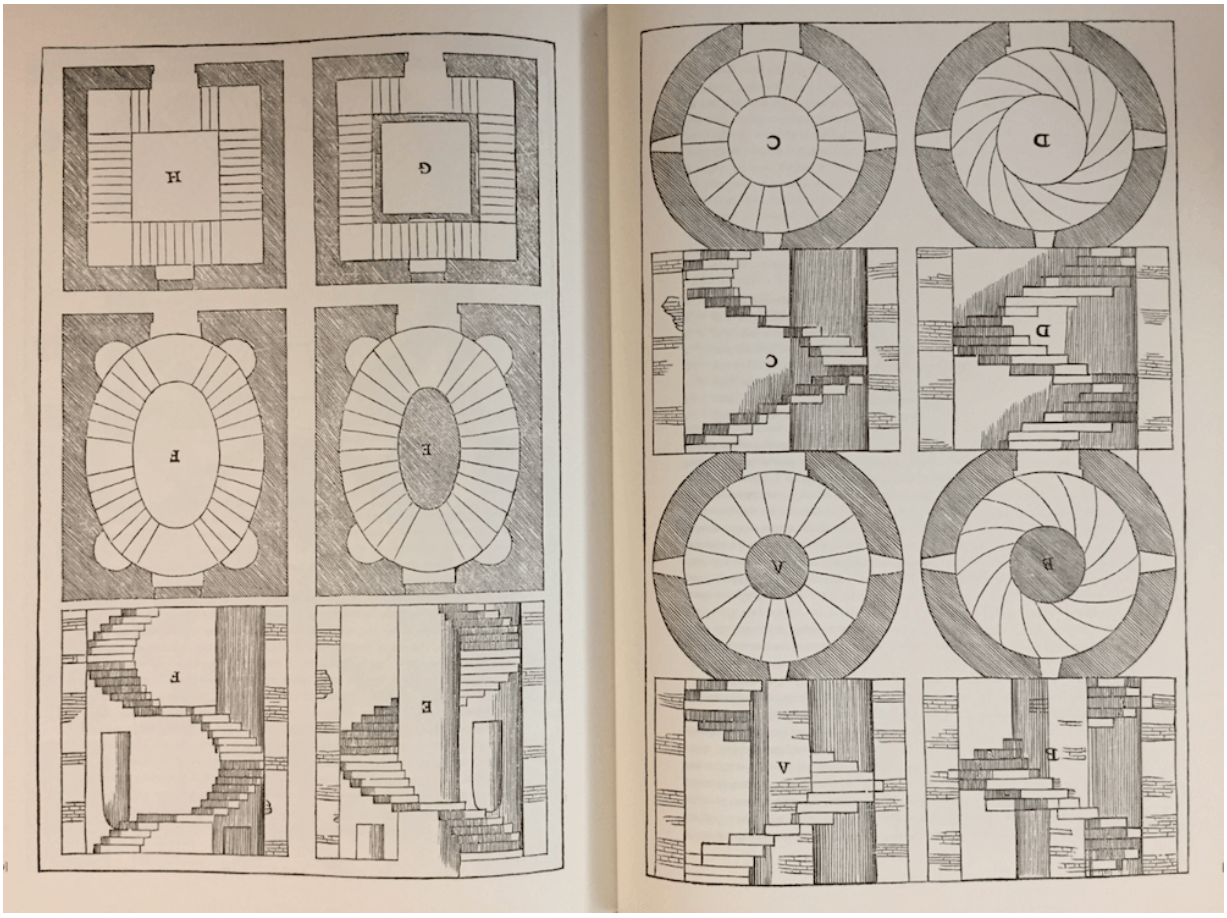
As your language becomes richer and more complex, you need an efficient way to capture and share it. On today’s web, a pattern library is one of the key examples of good practice in supporting a design system.

## Pattern Libraries And Their Limitations

A design system consists not only of patterns, but also *techniques and practices* for creating, capturing, sharing and evolving those patterns. A pattern library is a tool to collect, store and share your design patterns, along with the principles and guidelines for how to use them. Even though pattern libraries have become popular on

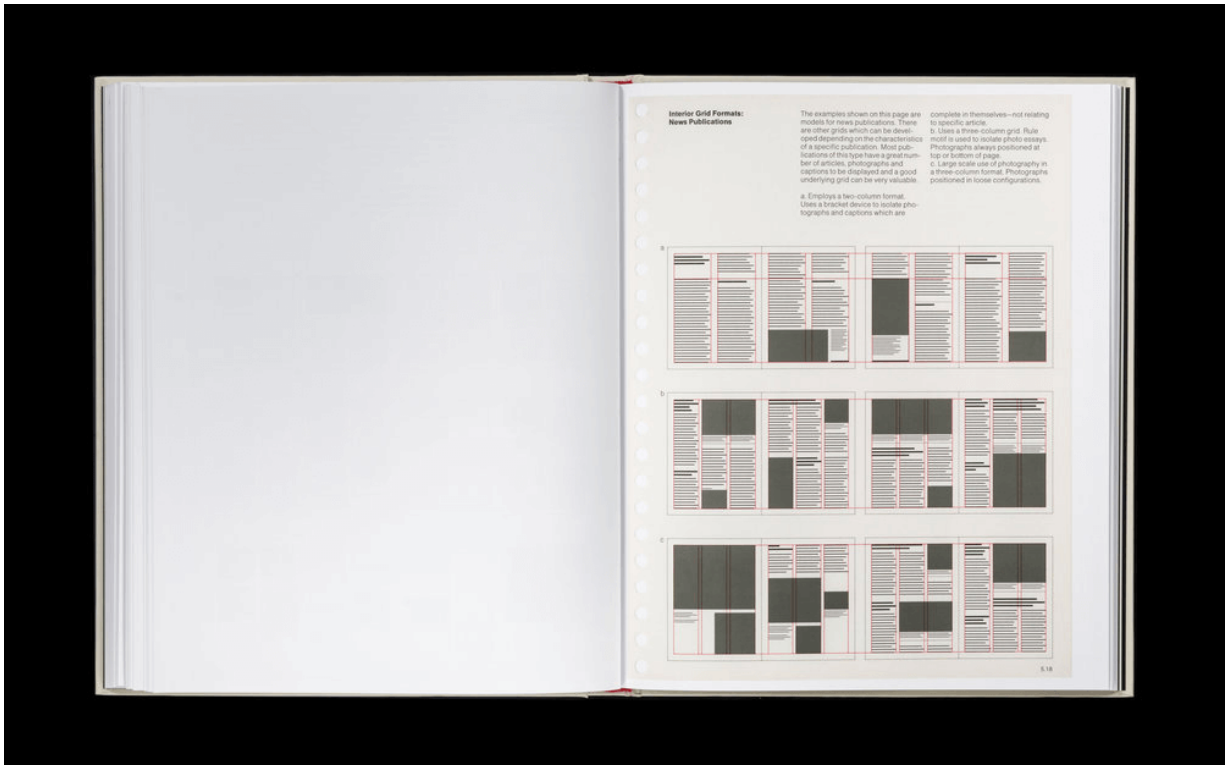
the web relatively recently, the concept of documenting and sharing design patterns in various forms has existed for a long time.

Palladio's *The Four Books of Architecture*, first published in 1570 in Venice, is one of the most important and influential books on architecture. It is also one of the earliest examples of system documentation. Drawing inspiration from Greco-Roman architecture, Palladio provided rules and vocabulary for designing and constructing buildings, including principles and patterns, with detailed illustrations and explanations of how they work.



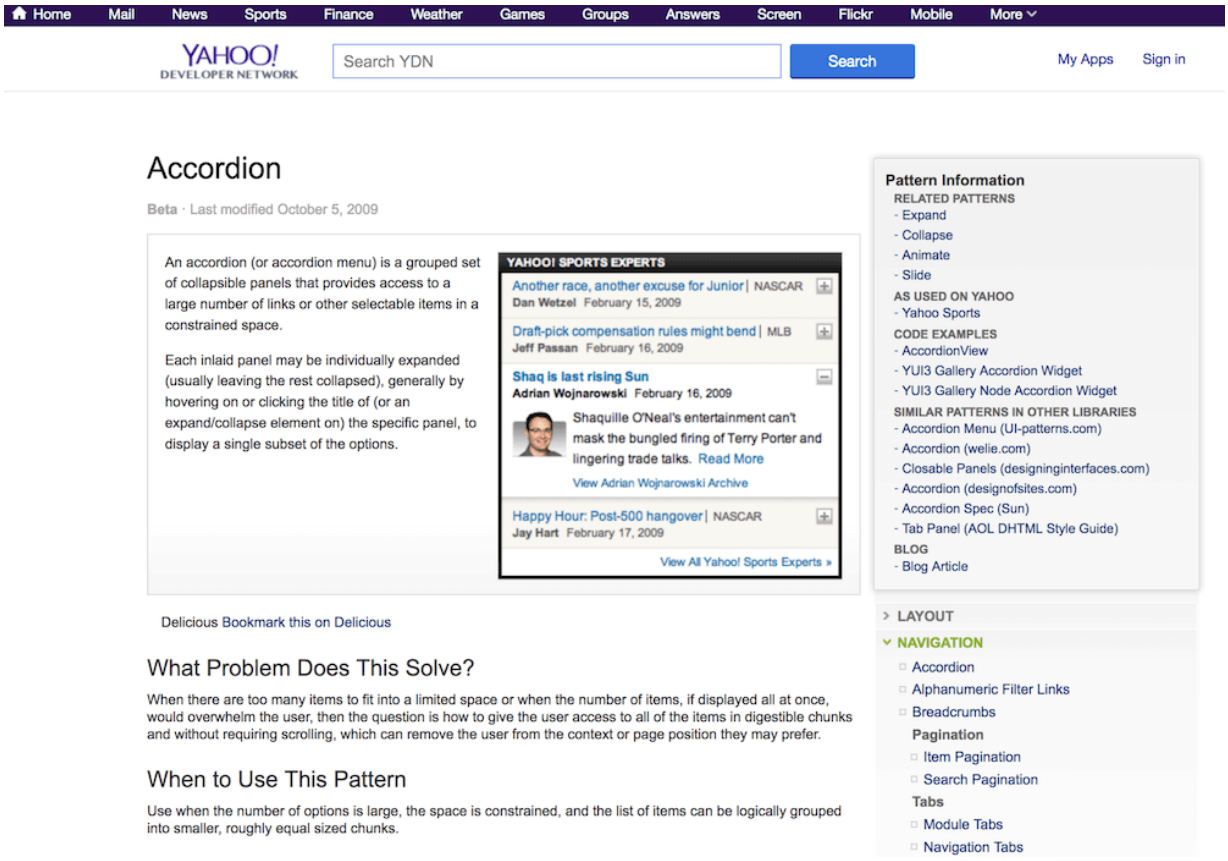

*Types of staircases: spiral, oval and straight. Palladio described how and when to use each type; for example, spiral staircases are suited for "very restricted locations because they take up less space than straight stairs but are more difficult to go up."*

In modern graphic design, systems have also long been documented, from early typography and grid systems, to Bauhaus design principles. For the last few decades, companies have documented their visual identities in the form of brand manuals, with NASA's Graphics Standards Manual from 1975 being one of the more well-known examples.



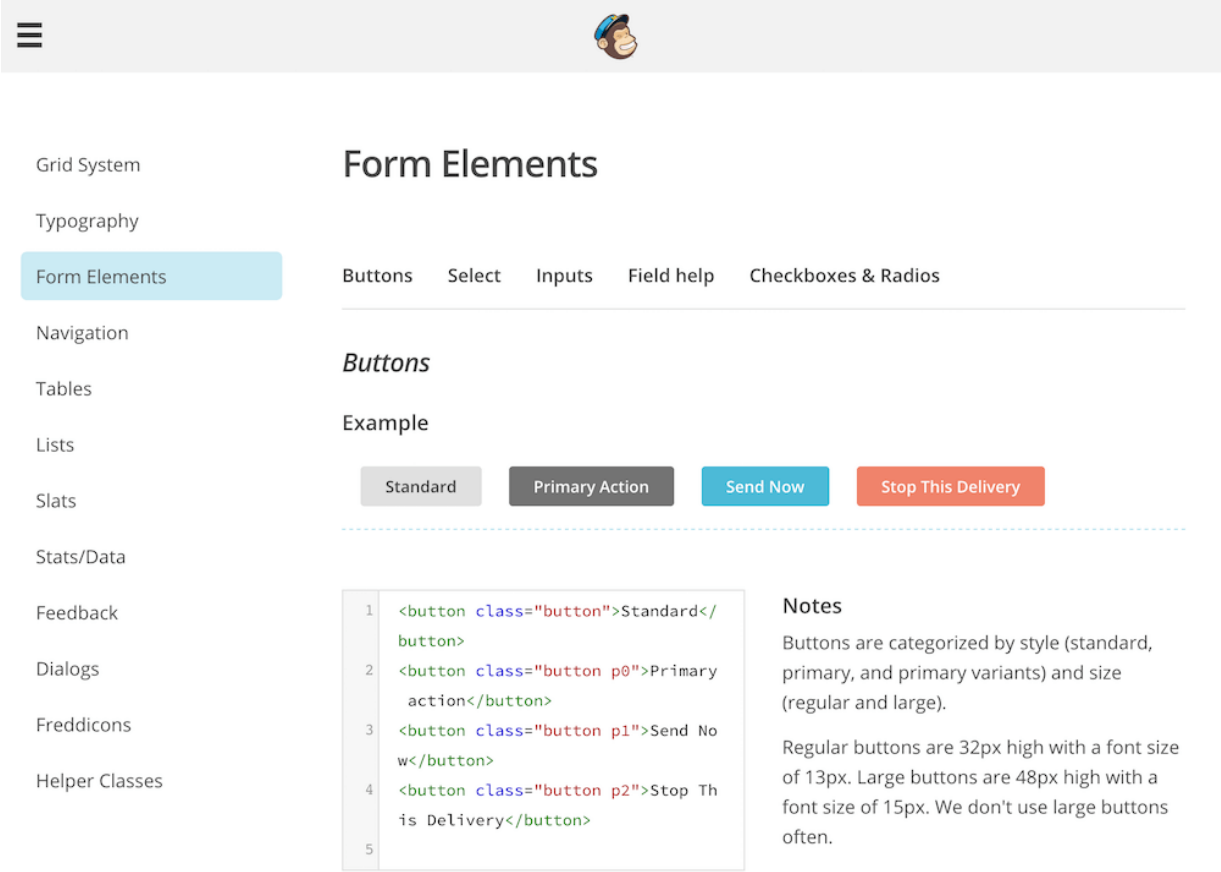

*Layout guidelines in NASA's Graphics Standards Manual.*

On the web, pattern libraries started as extended brand identity documents that focused on logo treatments, corporate values and brand styles, such as typography and color palettes.[8] They then extended to include interface modules, as well as guidelines for their use. Yahoo's pattern library was one of the first examples of documented interface patterns.




*Yahoo's pattern library was one of the first examples of documented interface patterns.*

For companies less resourceful than Yahoo, a pattern library would typically live in a PDF or a wiki, which meant that it was static and difficult to keep up to date. The aspiration for many designers and developers today is a more dynamic or "living" pattern library that contains the design patterns, as well as the live code used to build them. A living style guide or pattern library is more than a reference document — it's the actual working patterns used to create an interface for a digital product.




*MailChimp's pattern library*[9] *is one of the most influential early examples of living pattern libraries on the web.*

## THE LIMITATIONS OF PATTERN LIBRARIES

Pattern libraries are sometimes thought of as interchangeable with design systems. But even the most comprehensive and living pattern library is not the system itself. It's a tool that helps to make a design system more effective.

A pattern library doesn't guarantee a more cohesive and consistent design. It might help to correct small inconsistencies or make a codebase more robust, but a tool alone will have very little impact on the user experience. No pattern library will fix bad design.

Patterns can still be badly designed, misused or combined in ways that don't work as a whole. As Michael McWatters, UX architect at TED, noted in an interview: "Even a SquareSpace template can be ruined by sloppy design thinking." And conversely, a product with a coherent user experience can be created without a comprehensive pattern library (as we will see in chapter 6 in the example with TED's design system).

A living pattern library is associated with better collaboration. Yet you might end up in a situation where only a small group of people uses it, or there might be too many disconnected patterns as a result of a lack of communication between the teams. When up-to-date, a pattern library is a glossary for the shared language. But that doesn't mean there isn't still room for interpretation. It's the discussions *around* the interpretation which are often the key to a pattern library's success or failure.

On the other hand, pattern libraries are sometimes blamed for stifling creativity, leading to generic looking websites. But again, a pattern library reflects the design system behind it. If your system is fundamentally rigid and restricting, the pattern library can reveal and emphasize the rigidity. If it allows plenty of creative experimentation, a good pattern library can make the experimentation easier.

With all the automated tools and style guide generators available today, setting up a library of components can be done much quicker than in the past. But without a foundation of a coherent design system that integrates the patterns *and* practices, its impact will be limited. When a pattern library is used to support a solid design

language foundation, it becomes a powerful design and collaboration tool. Until then, it's a collection of modules on a web page.

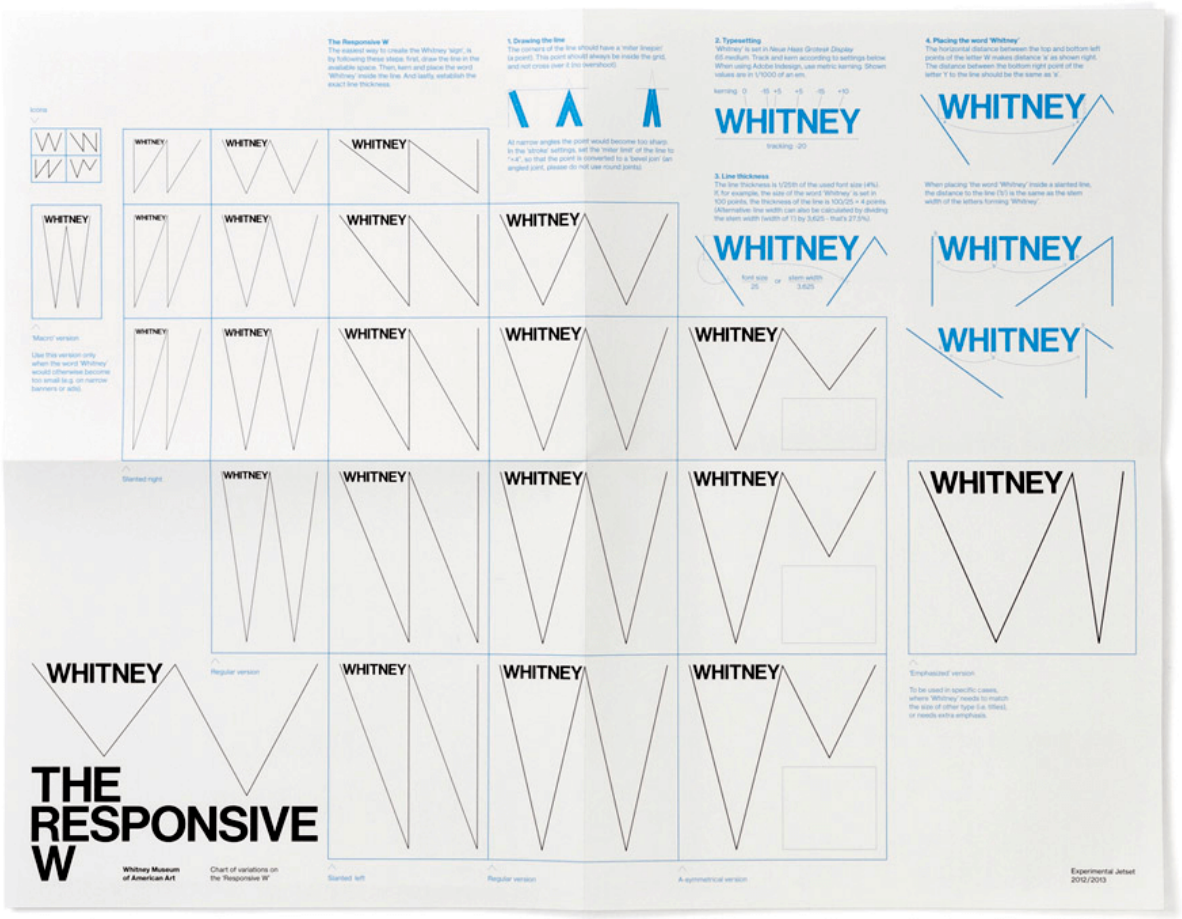
## What Makes An Effective Design System

The effectiveness of a design system can be measured by how well its different parts work together to help achieve the purpose of the product. For example, the purpose of FutureLearn is "to inspire lifelong learning in everyone, anywhere." We could ask, how effective is the design language of the interface to help us achieve this, and how effective are the practices of the team? If the interface is confusing and people don't achieve their goals, then the design language is not effective. If it takes too long to build a new area of the site because patterns have to be recreated from scratch every time, then we know our practices can be improved. *A design system can be considered effective when it combines cost-effectiveness in the design process, and efficiency and satisfaction of the user experience in relation to the product's purpose.*

### SHARED PURPOSE

In *Thinking in Systems*, Donella Meadows explains that one of the most important qualities systems have is how they're connected and organized: subsystems are aggregated into larger systems, which in turn form part of something larger. A cell in your liver is part of an organ, which is a part of you as an organism. No system exists in isolation. Your design system might have a smaller subsystem

within it: editorial rules to control the layout of the page; or it might include a method for scaling your logo responsively in a certain way. At the same time, it is also part of other larger systems: your product, your team, your company culture.




*The Whitney Museum of American Art's logo, a "dynamic W,"[10] is a simple yet remarkably adaptable system in its own right. The W responds to the artworks and words around it, allowing a huge range of flexible layout possibilities.*

In highly effective systems, we see that subsystems are connected and aligned toward a shared purpose: a design approach is mirrored in the front-end architecture; design patterns follow the guiding principles; the pattern language is applied consistently in design, code and the pattern library. We see harmony in the way

those systems function: their workflow is more efficient, and the user experiences they generate feel more meaningful and coherent.

### IDENTIFYING PROBLEMS

When there are gaps, it's also easy to see them. A fragmented design system leads to a fragmented user experience, full of conflicting messages. We want the user to sign up for our newsletter, but we also want them to check out our latest products. We want them to progress through the steps, but we also need to make sure they rate each recipe. "Sequence" is used to preview the steps in one area of the site; in another, it is suddenly a tabbed navigation. The interface is full of various shades of the same color and multiple versions of the same button. The team's productivity is also affected: making the tiniest change is time-consuming and fiddly because of the confusing, tangled up code. Designers spend their time copying pixels and reinventing solutions to the same problems, instead of understanding and solving real user needs.

How do we reduce the gaps and make our design system more effective? By understanding what it is and how it works. We'll start by looking at how a pattern language evolves in a new product, by taking a simple fictional ten-minute recipe website as an example.

## Example: A Ten-Minute Recipe Site

Imagine we are creating a website for sharing recipes with people who love healthy food but don't want to spend a lot of time cooking.

If we were to approach it with a system in mind and define design patterns early on, where would we start? Before we begin, let's make some assumptions. We understand the domain of cooking, and enough research has already been done to inform our design decisions. So what we're trying to do is not figure out what the experience should be, but see how we can establish design system thinking for this new website.

## PURPOSE AND VALUES

One of the first things we'd do is try to understand who our users are, their goals, needs and motivations. To keep it simple, let's say that we know they are busy professionals and their goal is to get a tasty, healthy meal without hassle and hours spent cooking. There are many ways we can help them achieve this goal: connect them with chefs, deliver food to their doorstep, build a conversational app. But we want to do it through a website with ten-minute recipes.

If we were to express the purpose in a single sentence, it would be along the lines of: *Motivate and empower people to cook delicious healthy meals in no more than ten minutes.* This purpose is the core of the product, and it should inform design and development decisions. The team should recognize that purpose and believe in it — it shouldn't feel forced.

Another important element is the *ethos* that captures the values and spirit we try to portray through the site. For us it can be simple healthy food and experimentation with common ingredients. For

other cooking sites it can be haute cuisine and mastery of culinary skills.

## DESIGN PRINCIPLES

To make sure the purpose of the product is expressed clearly through everything we do, we would need to establish a few grounding principles and values. They might be discussed informally or written as a manifesto — what's important is that the people involved in the creation of the product agree on those values and commit to them.

For example, everyone on the ten-minute cooking recipe site team understands the value of time. They are motivated by having a time limit on the recipes, and as a result they all strive to make interactions on the site short, simple, fast and smooth. This principle will be expressed not only through design patterns, but the performance of the site, its tone of voice, and even how the team operates.

These principles might not necessarily be official or even written down. But having an agreement and awareness in the team on what they are is essential to keep everyone's effort and priorities in sync. It can also help with decision-making: from which feature to build first and which approach to use, to working out UX flows, or how buttons should look and the choice of typeface.[11]

## BEHAVIORS AND FUNCTIONAL PATTERNS

We'd work out some of the key behaviors we want to encourage or enable in our users that will help them achieve their goals.

- We want people to always choose a healthy home-cooked meal over fast or microwaved food. This means that our meals need to look delicious and healthy, and be more enticing than microwaved food. Good imagery with irresistibly delicious yet simple-looking food can help us here.
- We want to enable people to achieve good results in under ten minutes. This means that we'll need to present the recipes in a few simple steps. The steps should be short, clear and focused. Perhaps we could have a timer on each step, to make sure we keep to the ten-minute promise.
- We want to encourage people to be spontaneous and feel like they can prepare something whenever they like. Start with what you have already, rather than with what you need to get — no need to shop for extravagant ingredients. In terms of UI, this could be supported by easily selectable ingredient thumbnails with clear labels.
- Finally, we want to encourage people to be creative and spontaneous, and to have fun. Show which ingredients are optional and what they can be replaced with. Show unexpected combinations that could be interesting to try.[12]

Naturally, the design details will change as we work on the site, but those key behaviors would remain the same. Those behaviors will inform the *core functional modules and interactions* of the site: ingredient thumbnails, recipe cards, step sequence, timer.

## AESTHETICS AND PERCEPTUAL PATTERNS

Around the same time, we'd need to work out how we want to be perceived by someone using the ten-minute cooking recipes site. Are we simple and down-to-earth or glamorous and sophisticated? Are we serious or playful? Bold or subdued? Utilitarian or emotional? What are the aesthetics that capture the personality and ethos we want to portray through the interface? We'd start thinking about the brand.

We're passionate about healthy food, so we want it to come through the site. Perhaps it would have an organic, warm, wholesome feel. We also believe that cooking doesn't need to take a lot of planning and preparation; it can be spontaneous and fun, and you can experiment and be creative in the ten-minute time limit.

At this point we would probably put together some mood boards and start experimenting with visuals until the brand feels right.[13] Once we achieve this, we can define core visual brand elements such as typography, color palette, tone of voice, and any distinguishing features of the brand; for example, illustrations, image styles, specific shapes, unique interactions that capture the essence of our service and present content in the best way.

Let's say we come up with a warm, earthy color palette, hand-drawn icons, typography with a focus on readability, quality photography of healthy food, and a few simple interface elements and controls. These styles will become our *perceptual patterns*.

## SHARED LANGUAGE

Alongside this process we will be making language decisions by choosing to refer to concepts in certain ways. What is a “recipe”? What do we mean by “steps”? What makes a “delightfully simple” interaction? The words we choose will influence how the team thinks. And indirectly, they will shape the design.

At first, the patterns and language choices will be shared informally. But as our team and the product grow, so will the language. Eventually we’ll need a more structured way to capture, share, and organize our design vocabularies.[14]

Now that we’ve looked at the design process in a nutshell using a fictional site, we can look at how systems evolve, using examples from real-world digital products and companies.

—

1. The patterns on FutureLearn are chosen to support reflective learning. The learner needs to focus on the task at hand and not be distracted by displacement activities. The goal was to create an atmosphere that feels calm and contemplative.
2. The website UI Patterns is a great source of common design patterns, grouped by purpose and the design problem it solves. For a comprehensive read on UI patterns, I also recommend *Designing Interfaces: Patterns for Effective Interaction Design* by Jenifer Tidwell.
3. http://smashed.by/patternsrecognition
4. Until the swipe gesture had emerged as a mobile pattern, no one would have known how to engage with it. Now we see whole products built on this pattern (such as Tinder). The transition to using what people know and exploring something new is very delicate; products live and die based on when and how they do this.
5. http://smashed.by/researchingsystems
6. http://smashed.by/visualloudness

7. The challenge is also not to impose one definition or conception of “Sequence,” but to understand and work with its context of use so that, for example, the UX team can support different types of sequencing (for FutureLearn, sequencing of courses, runs, steps, user actions, etc.) within a coherent framework.

8. Perhaps this is how we can distinguish style guides from pattern libraries. Style guides traditionally focused on styles, such as colour and typography. Pattern libraries can include styles, as well as functional patterns and design principles, among other things.

9. http://smashed.by/mailchimppatterns

10. http://smashed.by/whitneyidentity

11. In the next chapter we will look in more detail at the qualities of effective design principles and how they can form a foundation of your design language.

12. For further reading on understanding what people want, and shaping a vision for a new product, see “We Don’t Sell Saddles Here” by Stewart Butterfield, CEO of Slack.

13. More about the process of defining perceptual patterns in chapter 4.

14. In chapter 5, we will see how effective names and a collaborative naming process can become part of the foundation of a design language system. In chapter 10 we will look at pattern libraries as a way to capture language choices and establish a shared vocabulary.

CHAPTER 2

# Design Principles

*Solid principles are the foundation for any well-functioning system. In this chapter we'll discuss the qualities of effective design principles and look at some of the ways of defining them.*

Earlier we talked about the importance of starting with the purpose and ethos of the product when designing the interface. Having clarity on the purpose is paramount because all other decisions should be shaped by it, even if indirectly. How do we make sure that the purpose of the product is manifested through design? By establishing a few grounding values and principles.

In some companies, especially early on, trying to articulate shared guidelines can be hard. Design principles are not something that can be measured and quantified, and defining them can take a few iterations. There might also be some confusion about what principles are exactly. Depending on the company, they can be more focused on the brand, the team culture, or the design process. The principles at Pinterest[1] are more brand-focused ("Lucid," "Animated," "Unbreakable"), whereas at the UK's Government Digital Service[2] (GDS) they're directed more at how the team operates ("Do less," "Iterate. Then iterate again").

Sometimes principles are used for a limited time, for a specific project. Designer Dan Mall likes to write a "design manifesto" at the start of every project, to make sure creative direction and objectives are clearly expressed.[3] In other cases, principles are more long-

lasting, and their heritage becomes part of the company ethos. Take Jack Daniel's values of "confidence," "independence" and "honesty," which have remained the same for the last century.[4]

Larger companies might have separate sets of principles for the user experience, the brand and the design system.[5] Additionally, each team working in a company might also have their own team principles. While this works for some, others can find that having multiple sets of guidelines can contribute to a design system's fragmentation. At Atlassian, an enterprise software company, the principles for marketing and for the product were initially different. Over time, the team brought them closer together, and they are now working on a unified set of principles, with the goal of having a shared philosophy to bridge the gap between the disciplines of marketing, product and support.

---

*"It is **one** system. The principles are there to connect the dots."*
*— Jürgen Spangl, head of design, Atlassian*

---

Instead of having a separate set of principles for different teams and parts of the system, Atlassian aims to have a few key values — such as "Bold," "Optimistic" and "Practical, with a wink" — going *across* all the touchpoints in the customer journey of Atlassian products. While those values are the same throughout the journey, they are not represented with the same level of intensity at different stages.

There is a lot of “boldness” in the sales and marketing areas of the site, to showcase the products and help people understand their value. But once you get to the product itself and support areas, the experience becomes more about getting the work done and making sure people are able to be as effective as possible. So the boldness is toned down and the “practical” value is shifted up. As Kevin Coffey, design manager at Atlassian noted: “Nobody wants a bold support page.”

## Qualities Of Effective Design Principles

Approaches to design principles are unique to every company and can take many forms. Principles can be overarching or more granular, temporary or long-lasting. What matters is how effective they are at unifying design thinking and distributing creative direction in the team. In the context of this book, design principles are *shared guidelines that capture the essence of what good design means for the team, and advice on how to achieve it*; in other words, agreed criteria for what constitutes good design in your organization and for your product.

Regardless of your approach, effective guidelines typically have these qualities in common.

### 1. THEY’RE AUTHENTIC AND GENUINE

I’m sure you’re familiar with these principles: “Simple. Useful. Enjoyable.” They are ubiquitous, we hear them everywhere. There’s

no argument that products that are designed well follow a certain set of common principles (take Dieter Rams' ten commandments of good design, for example). But qualities like these should be a given — they should be done by design — along with other concerns, such as accessibility and performance. I've yet to see a consumer digital product which has "Complex," "Useless," and "Painful to work with" among its principles.

Knowing that your product should be useful and enjoyable is not going to be hugely helpful in guiding your design decisions, because these qualities can be interpreted in a variety of ways. What would make them more helpful is knowing exactly what those words mean to your team and your product. What does *innovative* entail? When is a design considered *useful*? How do you know if it's really *delightful*? Good design principles define qualities that can be interpreted in different ways, and ground them in the context of a specific product.

Let's take TED as an example. One of TED's design principles is *"Be timeless, not cutting edge."* The word *timeless* is specific not only to TED's interface but its entire brand and design approach. And this means they are not going to adopt a new technology or introduce a design element for the sake of following a trend. First and foremost, it has to serve a purpose, and it has to work for as many users as possible. For TED, timeless means not only simplicity but also being conscious of stylistic features that have no proven benefits to users. The team wouldn't introduce a parallax effect, for example, even though it feels very current, unless it solved a real design problem.

## 2. THEY'RE PRACTICAL AND ACTIONABLE

A principle should offer practical guidance on how to solve a design problem within the context of a specific product. Compare these two versions of one of FutureLearn's principles:

---

*"**Make it simple.** Make it so simple it's almost invisible! We should always work to remove friction on the platform, creating an experience that allows users real freedom to the content. If our platform is easy to understand, people can and will use it more."*

---

This statement makes perfect sense — no one can argue with the need to have a simple and usable interface. However, it's not clear from this advice exactly what simplicity means and how to achieve it. Compare it with this version:

---

*"**No needless parts.** Every design element, from the largest to the smallest, must have a purpose, and contribute to the purpose of a larger element it is part of. If you can't explain what an element is for, most likely it shouldn't be there."*

---

In practice, the question “Is it simple?” is much harder to answer objectively than “Does this contain needless parts?” The latter can be acted on by going through the interface and questioning the purpose of every element.

To phrase the principles in a more practical way, try thinking of them not as something that should merely sound good, but something that offers *actionable advice*. Imagine a new person joined your team and you’ve been asked to share five things that are most important when designing for your product. If you tell them “We like things to be delightful here. Make it delightful!” it’s probably not going to help them do their job. You’d need to define what delight means and share practical examples of what delight looks like in the context of your interface.

Let’s take a look at a couple of examples of design principles and how they can be made more practical.

Vague: “**Make it clear.**”
Practical: “**Only one number 1 priority.** What do you want your users to see and do first?”

Vague: “**Make it simple.**”
Practical: “**Make it unbreakable.** Just like a children’s toy, make sure it is designed for exploration and impossible to mis-tap.”[6]

Vague: “**Make it useful.**”
Practical: “**Start with needs.** If you don’t know what the user needs are, you won’t build the right thing. Do research, analyze data, talk to users. Don’t make assumptions.”[7]

But even the best-worded principle can still be interpreted in a variety of different ways. Nothing can make a principle more practical than pairing it with a real-life example to show how it can be applied in practice. Find specific parts of your interface where a principle is clearly represented and connect the two together. Can you point to a place which clearly shows having “only one number 1 priority”? Can you demonstrate how a pattern can be truly “unbreakable” despite having rich interactions?

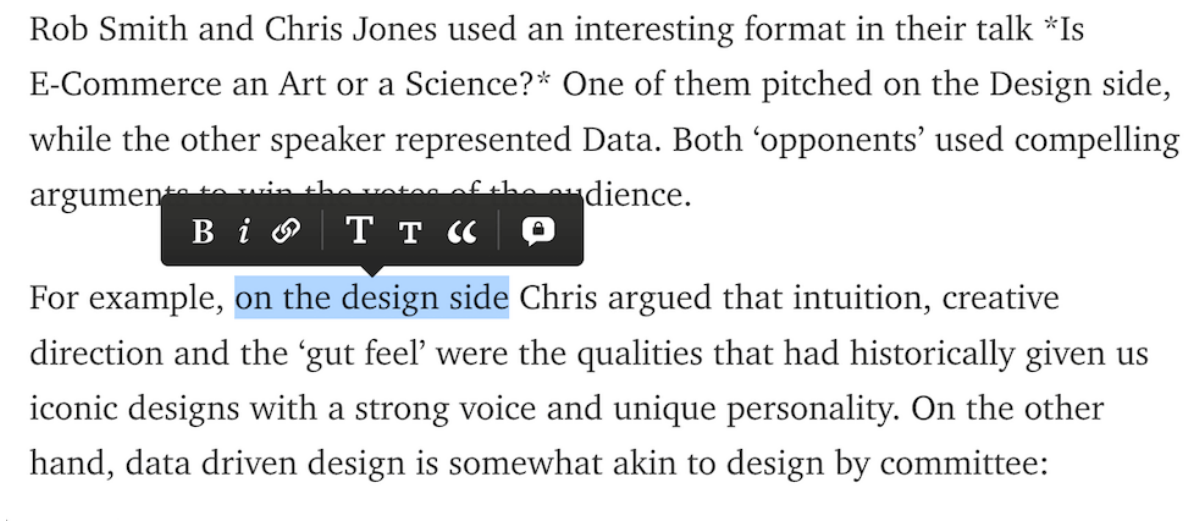
## 3. THEY HAVE A POINT OF VIEW

Design is shaped by the choices we make. Should this page be more visually alive or more utilitarian? Is it appropriate to be more playful or more serious here? Can we justify making this module more usable at the cost of making it less flexible? By achieving some things we often have to say no to others. Good design principles help to work out priority and balance, even if there are conflicting factors to consider.[8]

Take the Salesforce Lighting Design System[9] principles as an example: “Clarity. Efficiency. Consistency. Beauty.” It’s emphasized that they must be applied in the priority order above. Beauty should not be promoted over efficiency or consistency, and clarity should always come first. Ranking the principles in this way communicates to the team what should take priority when making design decisions.

It can be useful to acknowledge the conflicting values and suggest how to find a balance. One of the early design principles at Medium

was “Direction over Choice.” This principle was often referred to while the team was designing the Medium editor. They purposely traded a variety of formatting options for guidance and direction to allow people to focus on writing.[10]




*The few options available in Medium’s minimal editor clearly illustrated one of Medium’s principles, “Direction over Choice.”*

Good principles don’t try to be everything for everyone. They have a voice and actively encourage a designer to take a perspective. This idea has been emphasized by Dan Mall in “Researching Design Systems[11]”:

---

*“A design system should have guidelines for: perspective, point of view, extending creative direction to everyone that decides to build something with the design system. That stuff should be baked in. Otherwise, we all might as well use Material Design and call it a day.”*
*— Dan Mall*

## 4. THEY'RE RELATABLE AND MEMORABLE

Here's a fun test. Try asking people in your company what your design principles are. If no one can remember them, chances are they can be improved. To be memorable, the principles must be in constant use. They should be referred to in everyday conversations, included in presentations and design critiques, displayed where they can be seen. And to be in use, they must be genuinely useful, possessing the qualities above.

It also helps not to have too many of them. Human memory is limited and remembering more than four things at a time can be hard.[12] The optimal number of design principles — if you want them to be in use — is between three and five. When the teams at TED, Atlassian and Airbnb were asked about their design principles during interviews for this book, they were all able to recall them instantly. There wasn't a moment's hesitation; no one got confused or tried to look up the principles in a brand manual. How could they remember them so well? Their principles were simple, relatable, useful — and there weren't many of them.

Most importantly, the teams used them on a daily basis for making design decisions. Airbnb's four design principles ("Unified," "Universal," "Iconic," "Conversational") are deeply engrained in its design process:

*“When we design a new component, we want to make sure it addresses **all four of those**. If we didn’t have a set of principles it would be difficult to agree on things. We want to make sure each piece lives up to it.”*
*— Roy Stanfield, principal interaction designer, Airbnb*

The team at Spotify came up with the acronym TUNE (tone, usable, necessary, emotive) to make their design principles more memorable. Asking if a design is “in TUNE” during critiques and QA sessions has become part of Spotify’s design process.[13]

Making sure your principles possess the qualities above takes time, commitment and teamwork. But it’s worth the effort — a core set of principles are at the heart of any design system.

## Defining Your Principles

Expressing your design approach in five sentences is not easy. Every team arrives at their principles in their own way: some go through rounds of workshops, others might get input from their CEO or a creative director. Regardless of how you get there, the following tips can be useful to keep in mind.

### START WITH THE PURPOSE

Design principles must support the larger purpose of the product and help express the product's ethos. If you aren't sure where to start, look first at your company's overarching values or a product vision, and then try to work out how design principles can contribute to that larger goal.

The main purpose of TED's website can be captured in one sentence: "Spread the ideas as far and as wide as possible." In terms of TED's ethos and values, this means reaching as many people as they can, lowering the barrier to entry, and making the product inclusive and accessible. It means prioritizing performance and accessibility over flashy features, clarity of message over bold experimental design. Their "timelessness" principle encompasses that.

## FIND SHARED THEMES

If you're still in the process of defining your principles, a useful exercise is to ask a few team members (or everyone, depending on size of the team) to write them down individually. What, in their opinion, does good design mean for your product? How would they explain it in five sentences to a new member of the team, in a way that's practical and easy to grasp?

Ask them to find a practical example in your product's interface and include it alongside each principle.

Comparing your team's answers can reveal how much unity you have in your design approach. Are there many shared themes and overlaps? Have different disciplines ended up with similar

principles? It's always interesting to see everyone's responses, in particular how they differ between people who have just joined the team, and those who've worked on the product for some time. These answers can be a valuable starting point in further work on the principles, as you identify common themes and agree on priorities.

## FOCUS ON THE RIGHT AUDIENCE

A surefire way to end up with vague design principles is to have no idea who they're for. Are you writing for a corporate identity brochure? For the company website? A careers website? Potential partners and customers? Try writing your principles for yourself and for your colleagues, first and foremost: designers, developers, content producers, marketing professionals, domain experts — people directly involved in the creation of the product. Aim to come up with an informal agreement on what constitutes good design for your product, and offer practical guidelines for achieving it.

## TEST AND EVOLVE YOUR PRINCIPLES

As your product evolves, so will your principles. They will be shaping the design language, which in turn will be shaping the principles. Or they might start off being clear and focused, but over time become more diluted and lose their authenticity. To make sure your design principles continue to improve, they need to be constantly tested, evaluated and refined. This can only be done by being conscious of them and applying them in your work every day. By making your principles part of design critiques, for example, you

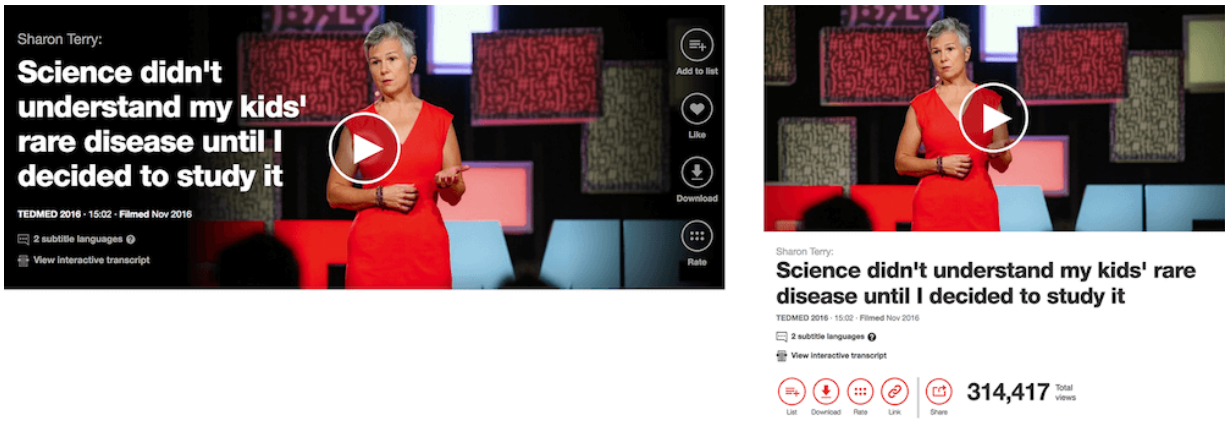
can continuously test if they help in your design process, and if not, continue iterating them.

## From Principles To Patterns

One of the challenges I've had in my work as a designer is working out how to materialize higher-level concepts, such as design principles and brand values, into concrete UI elements. How exactly are they embodied in the design patterns we create?

A lot of it is about choice and execution of patterns. For Medium, on the functional level a rich-text editor was required. It could have been any kind of editor, with any degree of complexity. But out of all the possibilities, Medium chose the simplest one, guided by its principal, "Direction over Choice."

For TED, clarity of message is valued over aesthetics. Trying to distill a talk into a single sentence can be hard, and sometimes titles can be long. It would be easier to clip the title, but for the team the message of the talk must always take priority. Instead of opting for an easier solution, they make sure that their design patterns can accommodate long titles.




*The hero banner pattern on a TED.com talk screen can accommodate long titles, which is in line with their design principles.*

There's a sense of prioritization also from the brand standpoint. For example, The TED team chose not to introduce a new image-rich home page until they developed a compression tool to minimize the impact such imagery would have on performance.

For the team at Atlassian, the "optimistic" principle is embodied in an "optimistic interface.[14]" In JIRA, for example, when a user has to move a card from "In progress" to "Done," cards are allowed to be moved right away, providing an instant response to the user, even though in the background there are a lot of checks and validations, and many things that can go wrong. They aim to express the "practical with a wink" principle through the friendly language of the copy, feedback messages, on-boarding, and other places throughout the site.

Design patterns are shaped by the core idea of how a product works. Think about how an ethos of "transparency and collaboration" is embodied through open channels on Slack; how the idea of "capturing life's unique moments" translates into

Instagram's photo feed and interactions; or how cards on Trello (a functionality that goes back to the seminal HyperCard system) encourage a certain type of workflow.

The choice and execution of the patterns and their unique combination are influenced by a product's purpose, ethos and design principles. You can view design principles as grammar rules for creating patterns and combining them in ways that make intrinsic sense. Equally, as the brand and functional patterns evolve and become more refined, they shape the design principles. Principles and patterns are refined and influenced by one another continuously.

Over the next two chapters we'll talk about design patterns in more detail, taking real-life products as examples. We'll see how design patterns emerge and how they are affected by a product's purpose and ethos, user behaviors, brand, business requirements, and other factors.

—

1. http://smashed.by/pinterestredesign
2. http://smashed.by/govukprinciples
3. "So…What Do I Make?" by Dan Mall
4. "When It Comes To Whiskey, America Knows Jack" by Avi Dan
5. Google has well known broad-brush design principles such as "Focus on the user and all else will follow," and a more specific set of principles for its material design language, such as "Motion provides meaning."
6. Example adapted from Pinterest design principles in "Redesigning Pinterest, block by block".

7. Example adapted from GDS design principle.
8. This point was inspired by the excellent article "A Matter of Principle" by Julie Zhuo.
9. http://smashed.by/lightning
10. "Creating useful design principles" by Dustin Senos
11. http://smashed.by/researchingsystems
12. For further reading on the limitations of human working memory see "The Magical Mystery Four: How is Working Memory Capacity Limited, and Why?" by Nelson Cowan.
13. "Design Doesn't Scale" by Stanley Wood
14. http://smashed.by/truelies

CHAPTER 3

# Functional Patterns

*In this chapter we'll discuss the role of functional patterns and why they should be defined early in the design process.*

Functional patterns are the tangible building blocks of the interface. Their purpose is to enable or encourage certain user behaviors.

On the ten-minute cooking site, some of those behaviors included selecting ingredients, choosing a recipe, and following steps within an allocated time. The functional patterns we design will be informed by those behaviors. Functional patterns, or modules[1], are largely shaped by the domain a product belongs to. The patterns in our cooking app would be quite different from, say, finance software. Instead of recipe cards we'd be dealing with task bars, data fields, grids, charts and graphs.

Functional patterns can be simple or they can combine to create more complex patterns. A recipe card is made of a meal title, image, ingredients, and an action button. Each module within the recipe card has its own goal: the title tells us what the meal is; the image provides a preview of the final result; ingredient icons allow us to scan the cards more easily. Together, those modules help to achieve a shared purpose: to encourage people to cook the meal shown on the recipe.

As the product evolves, so do the patterns. We might start allowing our users to rate the recipes, and the rating display will become part

of the recipe card. Or we might decide that the layout of the card can be improved, or that ingredient icons should be made clearer, or that we need to introduce a compact version of the card. We test and iterate the patterns to try to make them work better to achieve their purpose; that is, encouraging the behaviors more effectively.

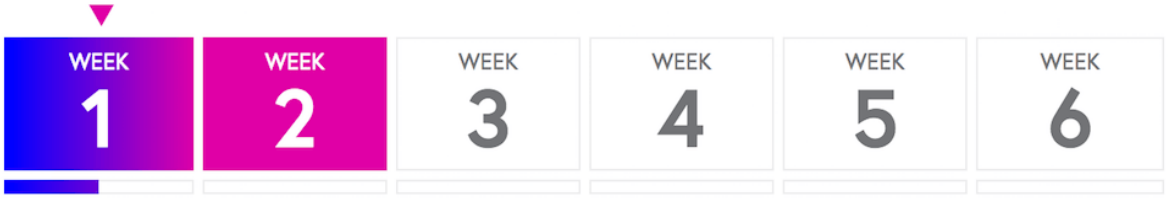
Articulating the purpose of the patterns early in the design process can help prevent duplication as the product grows. At first, it might not seem to be worth the effort; after all, a product can change too fast in its early days to be able to pin down all the interface parts. But do the core functional patterns really change that much? Let's take FutureLearn as an example and see how the interface evolved in the three years since the initial design.

## Patterns Evolve, Behaviors Remain

Since it was founded by the Open University in 2013, FutureLearn's vision has been to "inspire all to learn through telling stories, provoking conversation, and celebrating progress." To be able to do that, as a minimum we had to make sure people could discover and join an online course, motivate them to progress through, and make the learning experience rewarding and stimulating. This vision informed the initial functional patterns on FutureLearn.

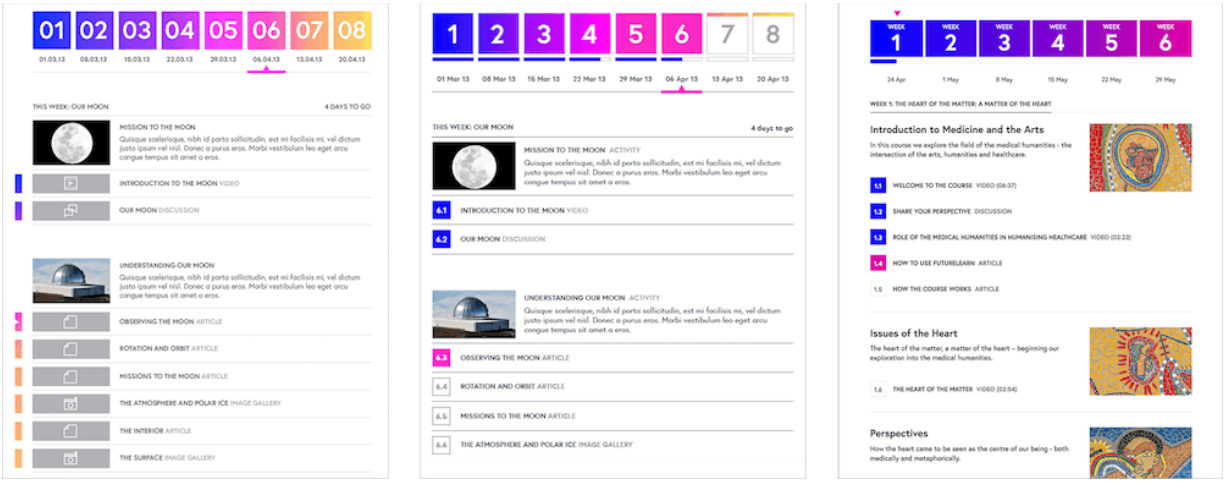
Courses are arranged in units and there's a linear flow to them — one part leads to another. On the interface level, this translates into a weekly structure. Each week is broken down into activities, and activities into steps. The course progress module is one of the core

functional patterns: it allows learners to navigate the content of the course, shows their progress, and where the course is currently active.




*Course progress module on FutureLearn.*

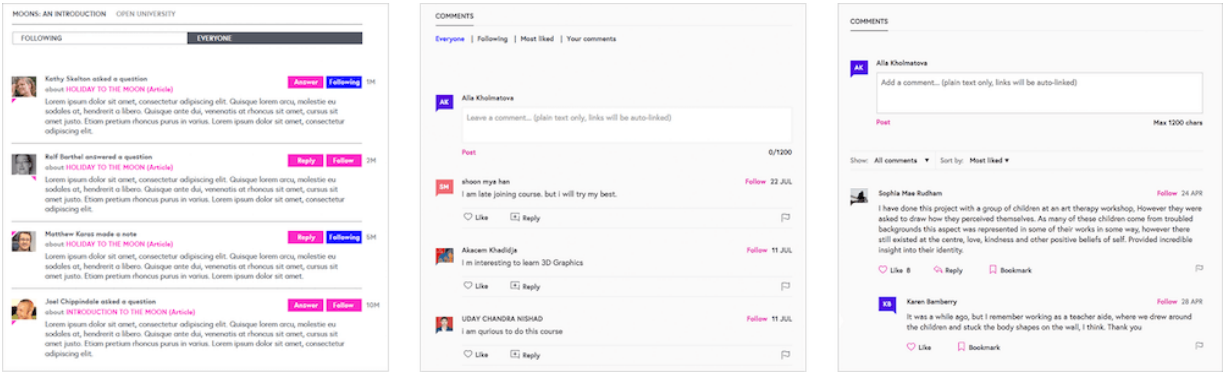
These patterns went through a few changes after they were first designed over three years ago. Their styles, and even functionality and interactions, have changed. Yet their purpose fundamentally stayed the same, as it's connected to the core idea of how FutureLearn works.



*The To Do page went through several revisions over the course of three years but the purpose of the core modules stayed the same.*

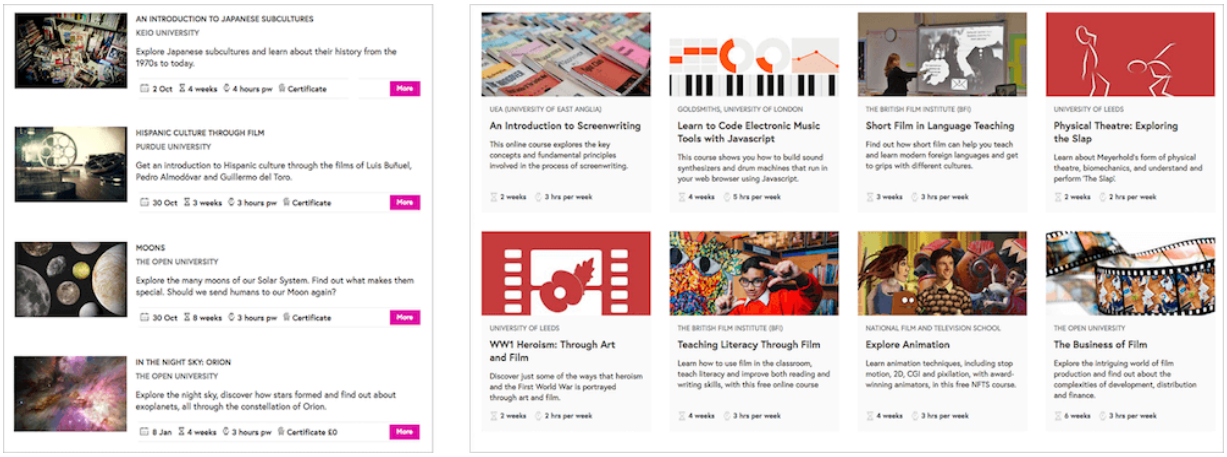
Similarly, discussion threads on FutureLearn have evolved over time, as the volume of people participating has increased: the layout of the threads, the interactions and the filtering features have

evolved, but their core purpose remains largely the same — to engage learners in conversation and allow them to learn from each other.




*Discussion pages went through several iterations once they were designed but the purpose of the core modules was unchanged.*

The core unit for displaying course details has also evolved over three years, to allow users to see a larger selection of course listings before needing to scroll down the page. But again, the purpose of the pattern remains the same — to allow people to discover and join the courses they're interested in.




*Course list evolved over years, to allow users to see a larger selection of course listings.*

As often happens in the early start-up days, because of time constraints and other priorities, many core functional patterns weren't defined. As FutureLearn's interface and educational functionality grew, patterns were duplicated. We ended up with several course progress modules, variations of a comment module, and a few different course blocks and course cards across the site. Perhaps this was inevitable. Or could some of those duplications have been prevented?

When a pattern is not defined and shared in the team, you start recreating it to accomplish similar goals: another promotional module, another news feed, another set of sharing links, another dropdown. Before you know it, you end up with 30 different product displays and pop-over menus.

Patterns are the physical embodiment of the behaviors we're trying to encourage or enable through the interface. Their execution, content, interactions and presentation can change. (In fact, patterns don't even have to be visual — they can be read out by a voice, or embodied in another way.) But the core behaviors they're designed to encourage remain relatively stable, as they stem from the purpose of your product and the idea of how it works. Being conscious of the purpose of your key patterns can help you understand how your system works and prevent it from fragmenting as it evolves.

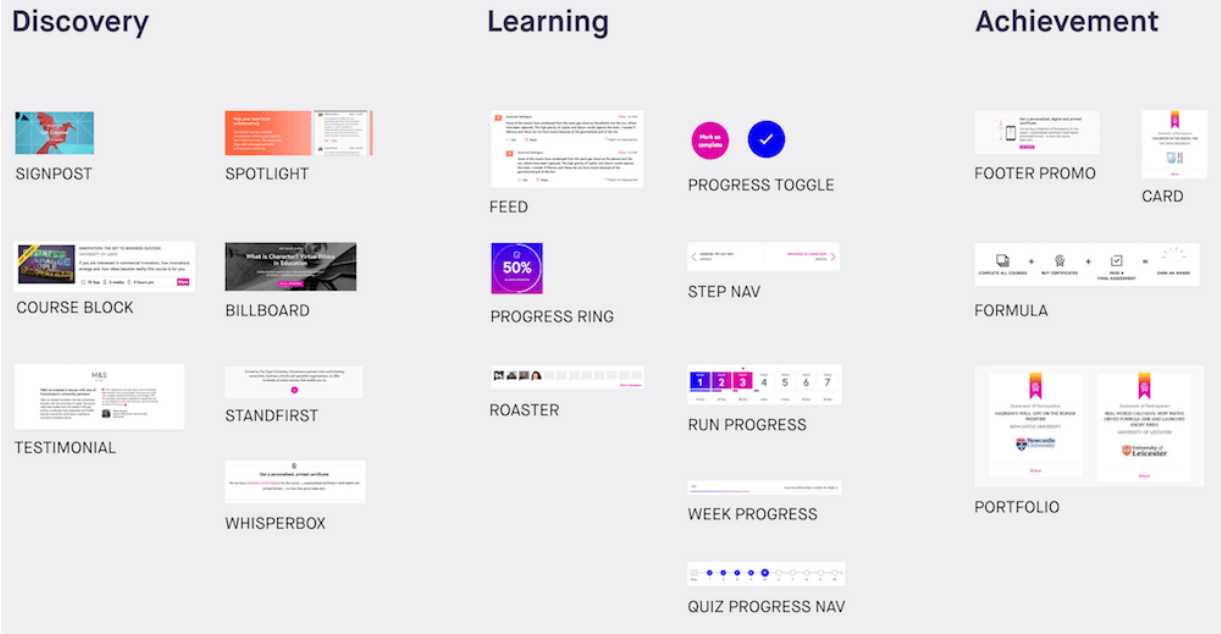
## Defining Functional Patterns

Defining patterns early in the design process doesn't have to take a lot of time. There are techniques that can be integrated into your process without extra effort. Here are a few that I find particularly helpful.

## CREATE A PATTERN MAP

To map out your customers' needs, goals and motivations you might have done customer experience mapping[2], "job to be done,"[3] or a similar exercise around customer journeys. The outcomes typically feed into the early design explorations and prototypes. At this point, there's usually a fairly clear understanding of the behaviors we want to encourage or enable in our users: learn about a course, join a course, engage in a discussion. But once we focus on the interface, sometimes we get lost in the details. We spend time making the most impressive page header, forgetting what this header is for and how it affects the user in different parts of their journey. In other words, we lose the connection between user behaviors and the exact patterns that encourage or enable those behaviors.

To see how your patterns fit into a bigger picture, try to map some of your core modules to the sections of a user journey. Think about what each section is for and what behaviors it's designed to encourage. You don't need to worry about capturing every icon or button at this point. Focus on the big picture: understand the parts of the system and how they work together. For FutureLearn, it was primarily focused on three areas: discovering content, learning on a course, and achieving a goal.



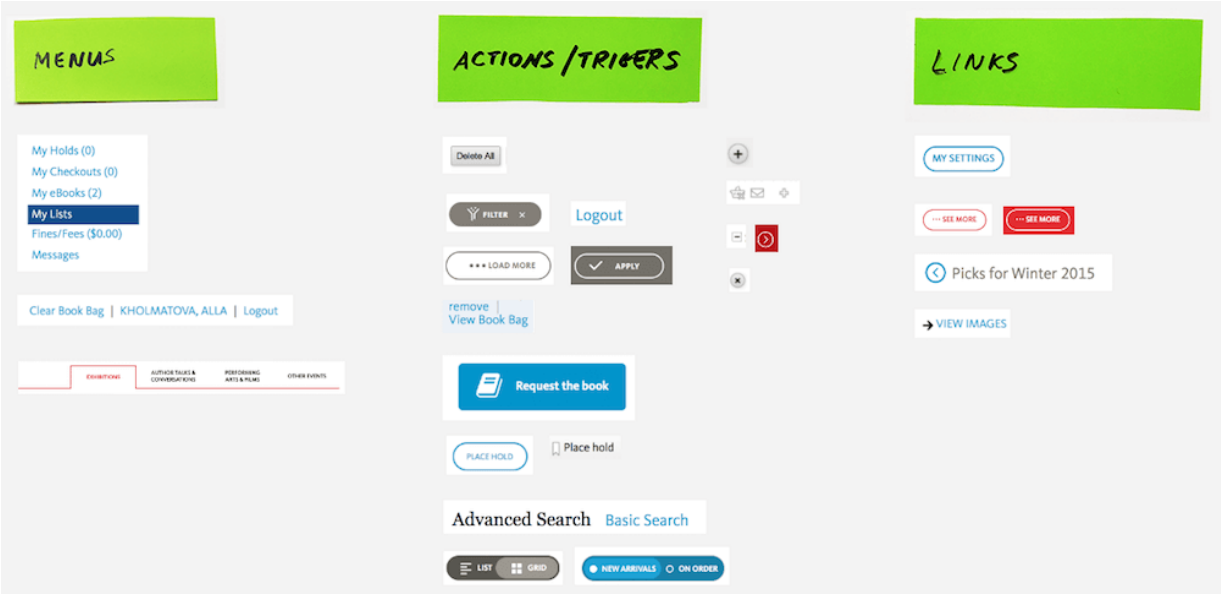

*Some of FutureLearn's functional patterns mapped to three key stages of a user journey.*

Keeping this map in my mind helped me to think in *families of patterns joined by a shared purpose*, rather than individual pages. For example, instead of designing a course list page, I'd think of the "Discovery" area as a whole. What are the behaviors we need to encourage at this stage of the user journey? What are the patterns that can support those behaviors? Where else on the site do they exist and how do they work there? If it's a new pattern, how does it contribute to the whole system? Thinking of all these questions is part of designing systematically.

## CONDUCT AN INTERFACE INVENTORY

The interface inventory process[4], described by Brad Frost, has become a popular way to start modularizing an interface. The idea is simple. You print out the screens of your interface on paper, or collect them in Keynote or PowerPoint. You then separate out

various components either by cutting them out or pasting them in Keynote.




*An interface inventory showing some of the interactive elements.*

You end up with a collection of parts which can be grouped into categories: navigation, forms, tabs, buttons, lists, and so on. Going through this process allows you to see duplicated patterns, and identify problem areas that need attention. This is when you discover that you have dozens of headers or pop-over menus, and start thinking about how to bring some order to all that.

An inventory doesn't have to encompass everything (although the very first one you do should be comprehensive). It can focus on one pattern group at a time, such as promotional modules, headers, or all the product display modules. You can do an inventory focused specifically on typography, or color, or animations.

To be most effective, interface inventories should be done regularly. Even if your team maintains a pattern library, new patterns will

emerge that need to be folded into the system. If you get into a habit of running inventories every few months, each time shouldn't take more than a couple of hours. And every time you do it, you understand your system better and can improve it.[5]

## VIEW PATTERNS AS ACTIONS

To understand the purpose of a pattern, try focusing on what it *does* rather than what you think it *is*. In other words, try to find an *action* that best describes the behavior a pattern is designed for. Describing a pattern with a verb rather than a noun can help you to broaden potential use cases for a pattern and define its purpose more accurately.

Say you've introduced a simple module to promote an online course. If you try to describe what it is, you might refer to it as "Image header" or "Course banner."


*UI component promoting an online course on FutureLearn.*

But by describing a pattern in this way, you could be making it too specific to its presentation or content. You might end up limiting its use to a particular context. On the other hand, if you define it in terms of actions — from your user's perspective as well as your own

— you can uncover its purpose: “*Promote* a course” and “*Discover* a course”; and “*Get inspired* to join a course” and “*Encourage* people to join.” By focusing on the action, you connect pattern with behavior and keep it open for various use cases. What else can this pattern promote? An online discussion? A new event? The name you give it should also reflect this. In the example above, we named the module “Billboard” to reflect its action-focused, promotional function.

## DRAW A PATTERN’S STRUCTURE

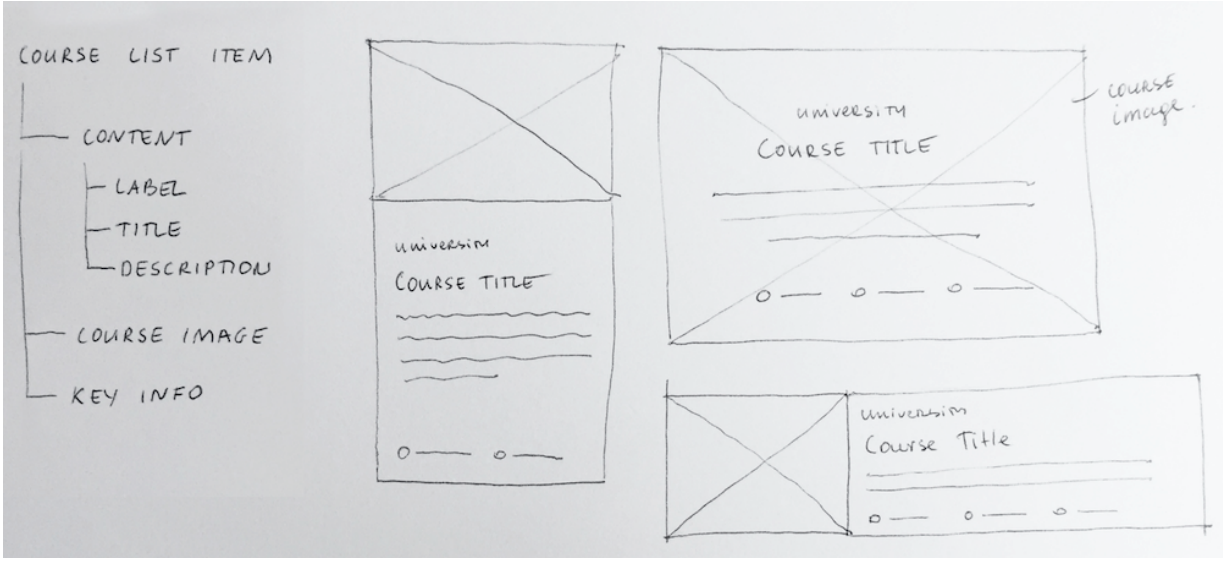
To get a shared understanding of how a pattern works, draw its structure: the core types of content a module needs to be effective.

Designers, developers and content strategists can do it together when working on a new module, or when refactoring an existing one. Start by listing the core content elements a module needs to be effective. For example, you might agree that in your interface a promotional module like “Billboard” needs:

- a heading
- a strong call to action
- an eye-catching background (solid color or image)

Next, try to determine the hierarchy of elements and decide how they should be grouped; for example: is the image part of the content? Is a label always necessary? While doing that, make a few sketches to visualize the structure. To give you an idea of what it

might look like, here's an example of the content structure for a course list item module on FutureLearn.

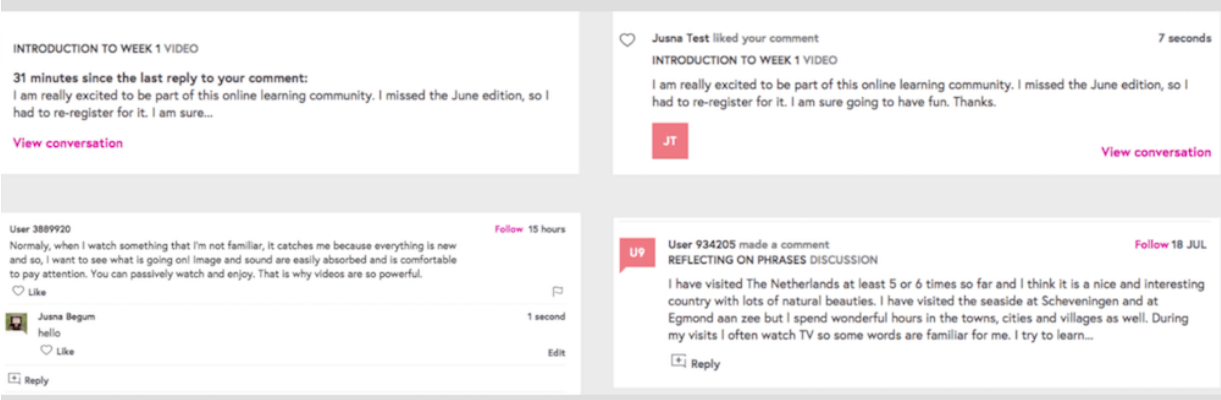



*Example of the content structure for a course list item on FutureLearn.*

At this point you might be thinking: "It's just a sketch or a wireframe. I do that all the time anyway." But it's a bit different. This is a sketch focused specifically on the content structure of a module, and the hierarchy and grouping of the elements.

Once you have a shared understanding of a pattern's structure, it's easier to make sure that the way the module is designed is reflected in the markup. A designer can work on the visual explorations, while a developer can start putting together a prototype (or both can prototype, depending on how you work). Designers understand how much they can push a pattern visually before it becomes something different. Developers know the reasons for the design choices and don't get unexpected designs thrown to them over the wall. Everyone is aware of how the pattern is constructed and how changing it will affect other places.

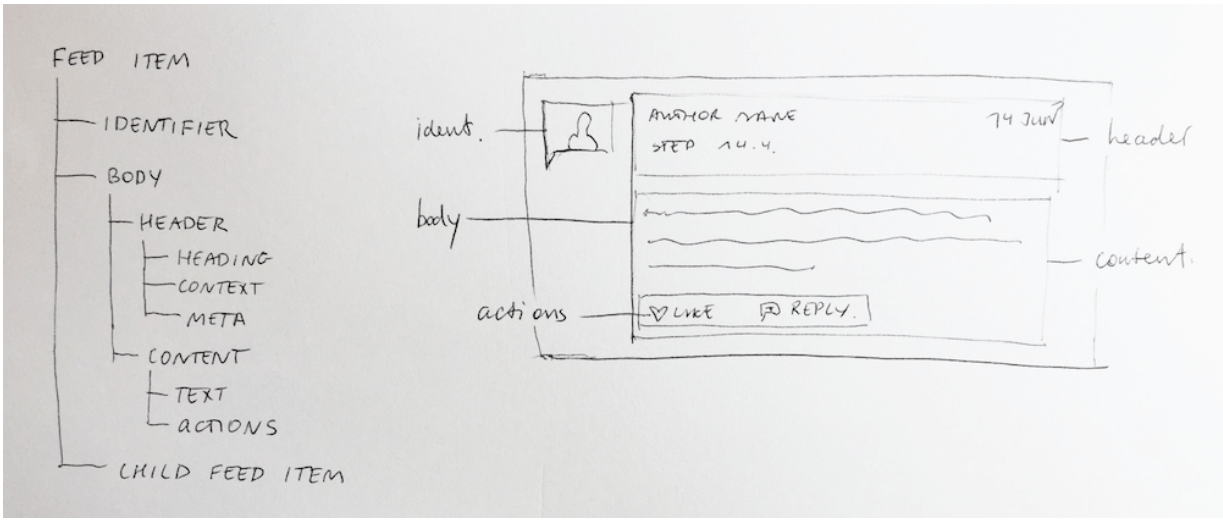
Here’s another example. On FutureLearn we used to have four different versions of a social feed in different areas of the site — two versions of “Comment,” a “Reply” and a “Notification.”




*Four different versions of social feed modules on FutureLearn.*

While at first glance they looked similar, they didn’t share the same styles; that is, if you changed one of them, the changes wouldn’t apply to others, there were visual inconsistencies in spacing and typography, and so on.

Breaking them down and drawing their structures allowed us to see if they could be unified into one pattern, and to design that pattern in a way that works in all four use cases.[6]

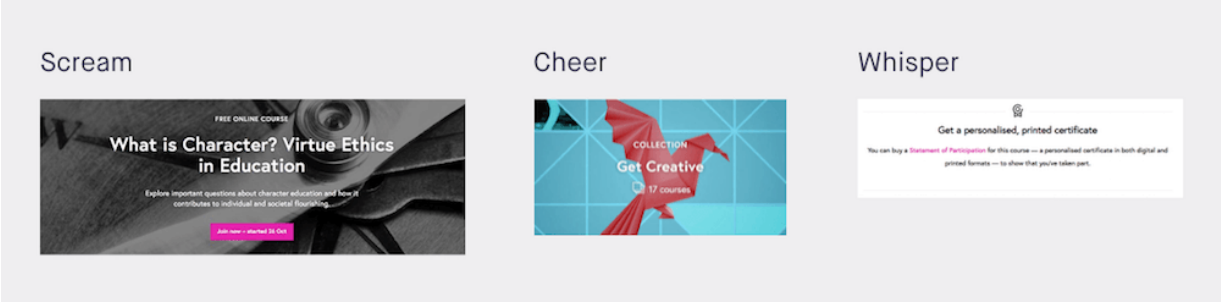



*The content structure for a “Feed item” module on FutureLearn.*

Content structure is closely linked to the purpose of a pattern, as these examples have shown. Knowing how a module is structured helps us understand how it works.

## PLACE PATTERNS ON A SCALE

Try placing similar patterns on a scale, in relation to one another. For example, there could be a few promotional patterns in your system, with varying degrees of intensity. Similar to the visual loudness scale[7] I mentioned in chapter 1, promotional modules can be compared with one another.




*Promotional modules can be placed on an imaginary visual loudness scale.*

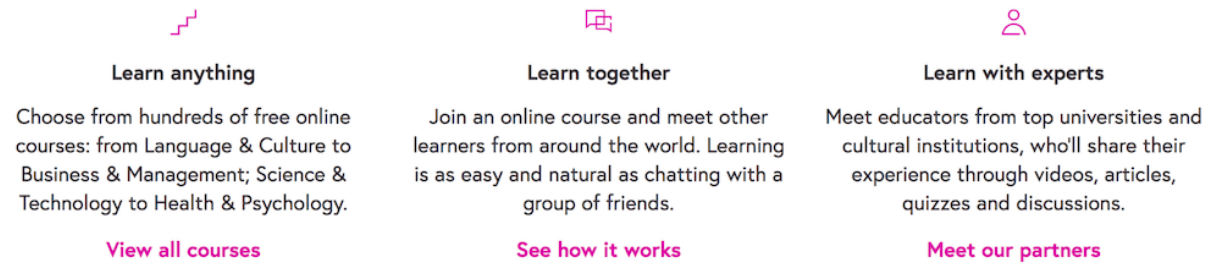
Placing patterns on a scale can help make sure they're used appropriately and don't compete for attention across the system. It also helps prevent modules being recreated needlessly, since you can see when you already have a module at the same "volume."

Another way to think about it is to imagine that your interface is not visual, but that it's being read out by a voice. When would that voice need to get louder and change intonation? Think about how that volume and intonation can be expressed visually, through the relationships between the elements within the modules, as well as their hierarchy in the overall design. Thinking of it this way, of course, also has an additional advantage of making it more accessible to screen readers.

## TREAT CONTENT AS A HYPOTHESIS

Here's a paradox. We're expected to design content-first, but at the same time we're expected to build modules in a way that can fit any kind of content. A way to do this is not necessarily by starting with content, but with the purpose. Then we can treat content not as a known asset but as a *hypothesis*. This allows us to test if we've defined the purpose of the module well, and if the design works for that purpose.

Suppose we have a module, which was designed for displaying product features.



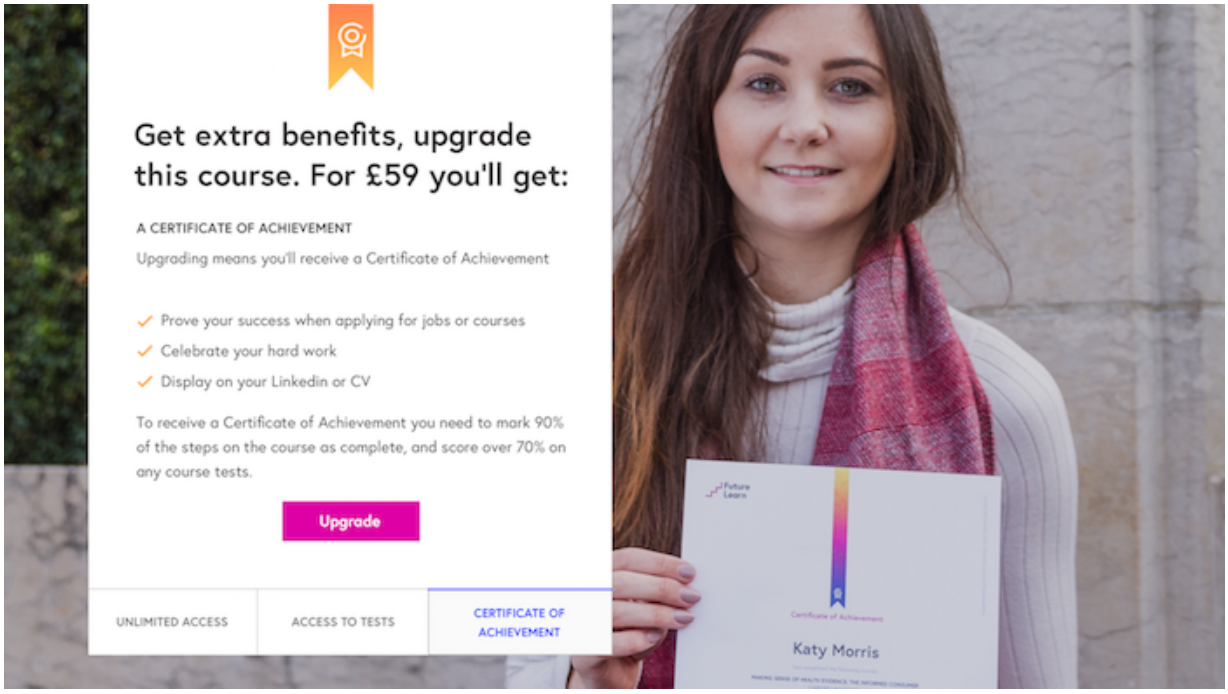

*An example of a module designed for presenting product features.*

We could define its purpose as “Supporting the main message with additional bits of easily scannable information.” The “bits” could be key features, short pieces of advice, or quick steps. We can build a pattern for content that fits this hypothesis (concise and scannable, supplementary rather than the main content on the page), and then test it.

If content consistently ends up being unsuitable for this pattern, it’s typically caused by one (or more) of these three reasons:

- We didn’t correctly define the purpose of the pattern. Go back to trying to understand the behaviors it’s been designed for.
- We didn’t design the pattern in a way that achieves its purpose best. Try a different design for this pattern.
- We’re trying to force the content into a pattern that’s not quite right for it. Consider revising the content, or try another pattern.

When we don’t start with the purpose and the structure, we can end up with modules that are too closely tied to their content. For example, we had a case at FutureLearn where the copy was pushing important tabs below the visible area.




*An example of a fragile module, which was too specific to its content.*

The tabs were meant to stay visible. To get around the problem, we almost introduced a custom smaller size heading, simply to nudge the tabs up a bit. But had we done that, we would have ended up with a module that wasn't robust. If the title got longer or if we added an extra line, we would have been stuck with the same problem. Had we started with the purpose of the module and its structure, the tabs would probably have been placed at the top, since they're such an important element of design.

These are just some of the tools and techniques you can try to define functional patterns in your interface. The most important thing is to understand how patterns link to the behaviors they were originally designed for: their purpose.

The purpose determines everything else that follows: the structure of the pattern, its content, its presentation. Knowing the purpose of

the pattern (knowing which behaviors it's designed to encourage or enable) can help us design and build more robust modules. It can help us know how much a pattern can be modified before it becomes something entirely different. And it can reduce duplication by giving the whole team a common point of reference, and connecting them with the original design intent. Being clear on the purpose also makes it easier to test the patterns to see how effective they really are.

If functional patterns are objects in the interface, then perceptual patterns are more like styles — they describe what *kind* of objects they are and how they feel. Let's take a closer look at them.

—

1. If we had to make a distinction between the two, I see functional patterns in a more generic way, as a kind of a Platonic ideal, and modules as the embodiment of functional patterns, which can be different in different interfaces.
2. http://smashed.by/mappingxp
3. "Job to be done" (JTBD) is an exercise that helps you focus on the underlying motivation behind using a product or service, rather on the product itself. For example, even though customers say that they buy a lawnmower to "cut grass", their higher purpose might be to "keep the grass low and beautiful at all times," which might indicate that a lawnmower is not the right tool for the job in the first place.
4. http://smashed.by/uiinventory
5. In chapters 7 and 8 we will look at the interface inventory process in depth, starting from the purpose of the system and going down to the smallest patterns, such as icons and colors.
6. In chapter 8 we'll discuss in more detail when to unify patterns, when to split them up, and when to create variants.

7. http://smashed.by/visualloudness

CHAPTER 4

# Perceptual Patterns

*In this chapter we'll discuss how perceptual patterns work and their role in a design system.*

Imagine we both have a house, with the same set of furniture: a table, a few chairs, a bed, and a wardrobe. Even though the furniture is the same, our houses can feel distinctly different: it could be because of the style of the furniture, the materials, colors, textures, the fabric on the bed covers, the style of the ornaments, how we lay out the room, the lighting, or even the music we play inside. I refer to such attributes as *perceptual patterns*. Because of them, your house might feel like a bohemian lair, and mine like a warehouse.[1]

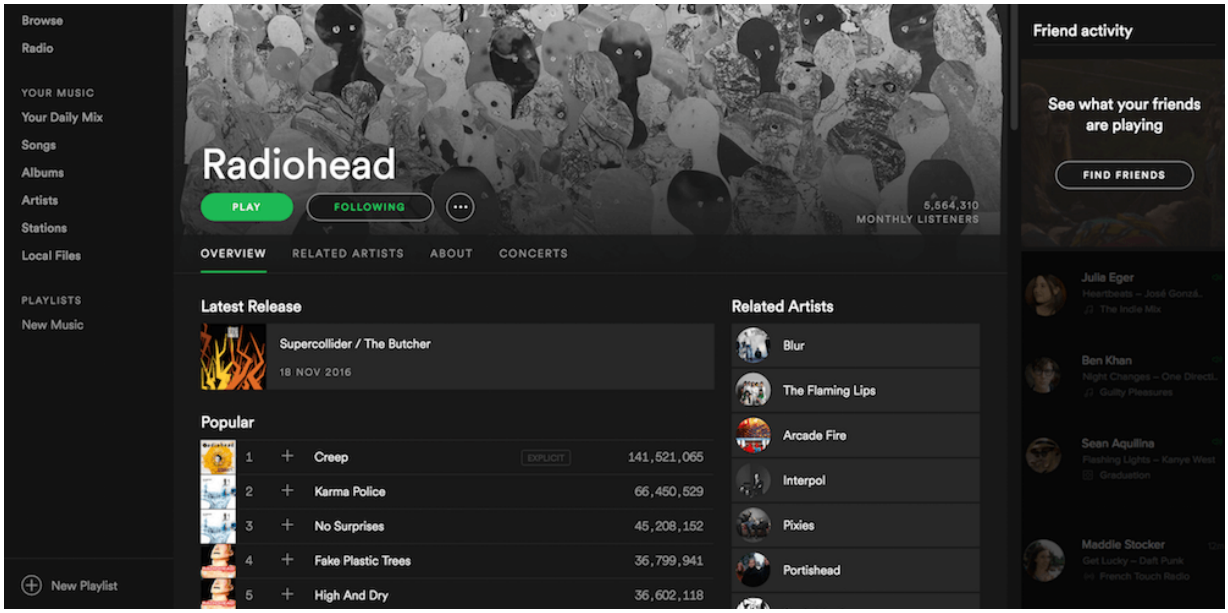
Examples of perceptual patterns in digital products include tone of voice, typography, color palette, layouts, illustrations and iconography styles, shapes and textures, spacing, imagery, interactions or animations, and all the specific ways in which those elements are combined and used in an interface. Perceptual patterns are always present, even if they're not purposefully designed. Even a purely functional tool has an aesthetic.

Sometimes such qualities are seen as styling or the skin of a product — a layer on top. But to be effective they must live not only on the surface but *at the core of the brand*, and they should evolve with the product. When effective, perceptual patterns become powerful product differentiators.

## PERCEPTUAL PATTERNS HELP TO EXPRESS BRAND IMAGE

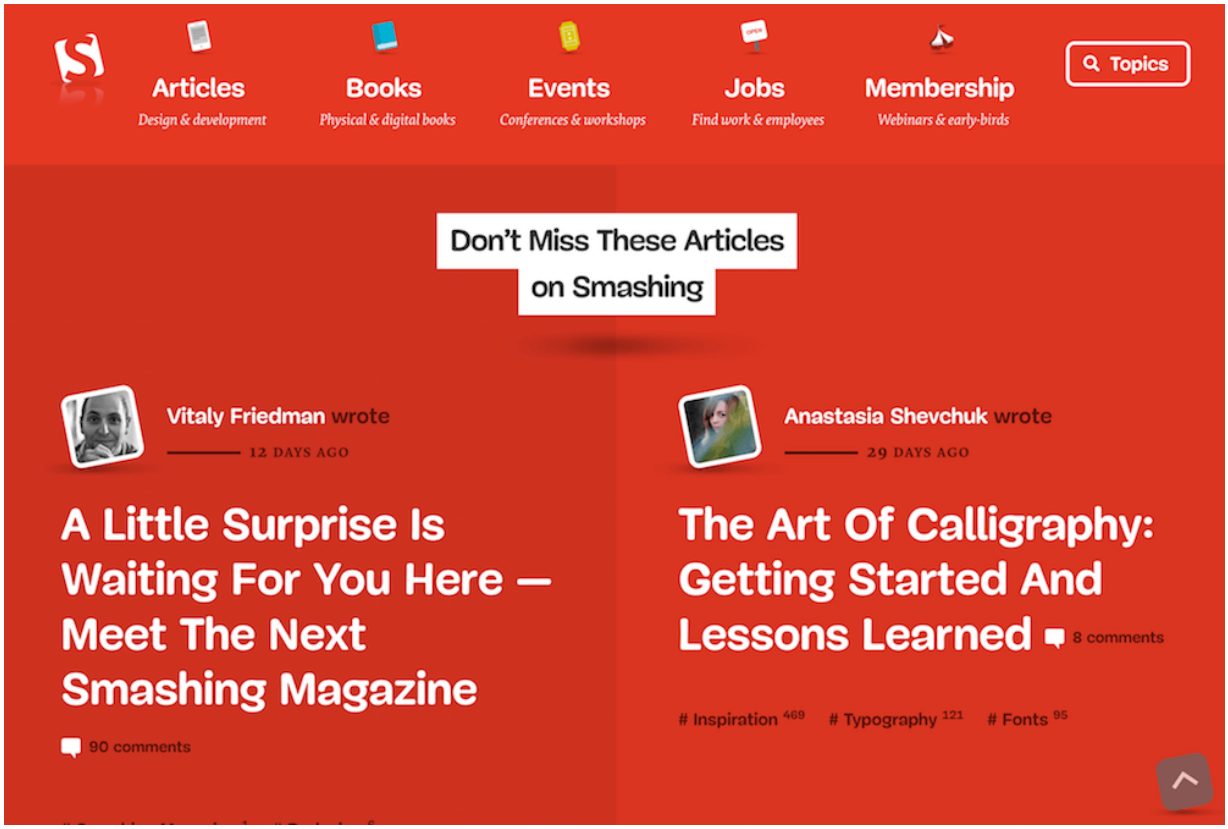
Products can feel different, even if they belong to the same domain and have similar modules. To write this book I tried dozens of word processors with very similar functionality. Only a couple of them, including the one I'm using now, had the kind of writing environment that helped me focus and be most productive.[2] I can describe it as crisp and calm, exhibiting a distraction-free aesthetic with an accent on important things, such as the display of the document outline, or the small circles which turn green as I approach my "writing goal." This environment is created through a combination of certain patterns, although at a first glance it's not easy to pinpoint what they are.

Let's take another example: Spotify. To me it feels warm and personal. What exactly are the patterns that create the ambience of intimacy in this digital music service interface with over 100 million monthly users? As well as the functionality, it's largely due to the imagery styles, color combinations (particularly the ratio of green to black), the subtle and calm feel of interactions, and the typography choices.[3]



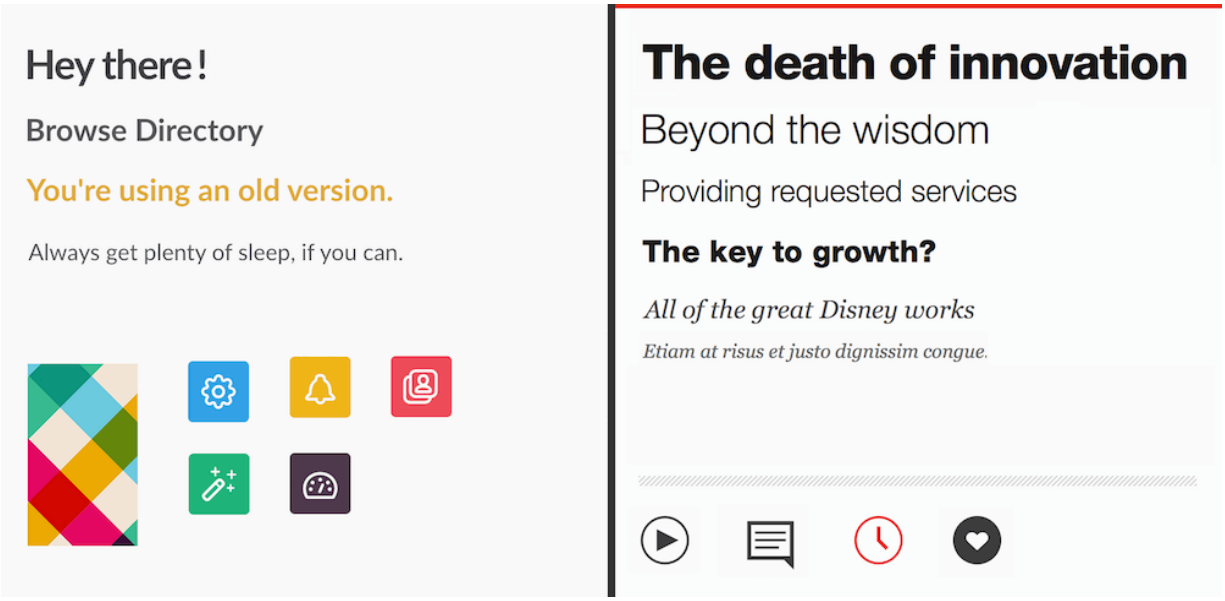

*The ambience of intimacy on Spotify is the result of a combination of perceptual patterns, such as subtle interactions, subdued imagery and color accents.*

On the other hand, the playful, creative, openly enthusiastic and slightly offbeat character of Smashing Magazine is conveyed through a different set of patterns — from the bold color palette and illustrations, down to the smallest details, such as a slight angle on some of the interface elements.




*The character of Smashing Magazine is conveyed through a variety of perceptual patterns — from typographic treatments to the playful angle of the images and icons.*

Perceptual patterns express the brand through the interface and help to make it memorable. Take a look at the images below: can you recognize which products they represent?




*Can you recognize which products these two images represent?*

There isn't a lot of information in these images: typography styles, some shapes and colors, and a few icons. But those visual cues are so distinct that you might still have been able to recognize which products they belong to (Slack and TED). By being applied purposefully and repeatedly, they form patterns, which is why we can recall and associate them, even when those patterns appear out of context.

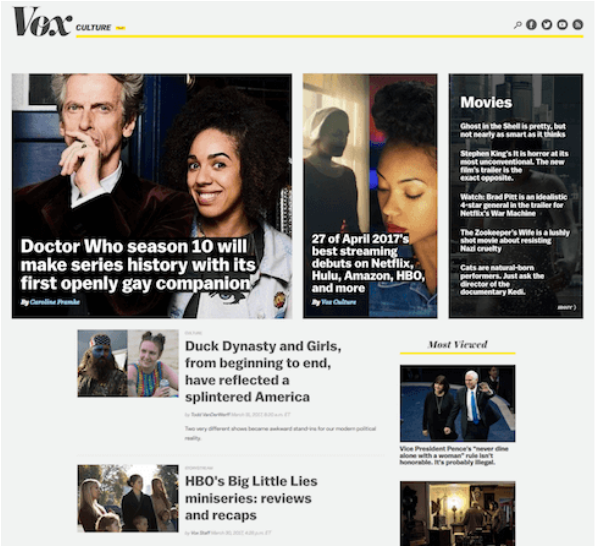
Perception is influenced not only by the building blocks such as color palette and typeface, but the *relationships* between them. It's not enough to use headings and colors across the modules consistently. We should also be aware of the unique proportions and combinations that make the product feel a certain way. How do the colors relate to one another? How does the imagery relate to the typography? How does the typography relate to the spacing?

Michael McWatters, UX architect at TED, shared in an interview how important it is for TED's brand that the color red appears in the right proportions: "Red should be used sparingly and thoughtfully. Use it in the wrong places, or too much, and it doesn't feel like TED anymore."

## PERCEPTUAL PATTERNS CONNECT THE SYSTEM

In modular systems, achieving visual coherence and seamlessness can be tricky. Modules are created by different people for different goals. Since perceptual patterns permeate through different parts in the system, they *connect* the parts. When the connections are effective, users perceive the unity that links the modules together.

Look how Vox and the Guardian apply perceptual patterns to bring together different elements, integrating them in a cohesive whole. At Vox, eye-catching imagery overlaid with large headings, crisp typeface pairings and generous white space all come together to convey the feeling of a lifestyle magazine under the Vox banner — it's spacious, informal and perhaps even a little rebellious. In contrast, typography, spacing, imagery and color in the Guardian create a denser, more credible feel, more fitting for a source of serious journalism and opinion.[4]





Connections can be made not only between the modules but across different platforms and contexts. Platform-specific standards such as material design take quite an authoritative view on how patterns should be designed and built. When companies follow native platform conventions strictly, they are very much reliant on perceptual patterns to make the product feel like part of the same brand.

Sometimes it's the smallest things that can help make a connection. Even though there are differences between Twitter's apps for web, native and third-party platforms, small details such as the "Like" interaction of the heart, help to propagate Twitter's pattern language.

*A still from Twitter's heart animation, introduced in 2015 on Twitter for iOS, the web, Android, TweetDeck, Twitter for Windows 10, and other platforms.*

## Exploring Perceptual Patterns

If functional modules reflect what users want and need, then perceptual patterns focus on what they feel or do intuitively. Rather than coming from user behaviors and actions, they are influenced by the *personality* or *ambience* a product strives to project.

In *Designing for Emotion*[5] Aarron Walter suggests how, in a simple way, teams can capture key personality qualities by creating a "design persona," which embodies the traits they wish to include in your brand. Walter recommends basing a persona on an actual image of a person, to make it feel less abstract. If that's difficult, I sometimes find it helpful to think of a place and its ambience instead, rather than someone's personality traits. For example, what is the ambience that suits focused writing as opposed to relaxed social chatter?

Whether you use a person or a place as a starting point, the goal is to end up with a few (Walter recommends around five to seven) traits that best describe your brand, along with the traits to avoid. For MailChimp, those traits include “Fun, but not childish. Funny, but not goofy. Powerful, but not complicated. Hip, but not alienating. Informal, but not sloppy.”[6]

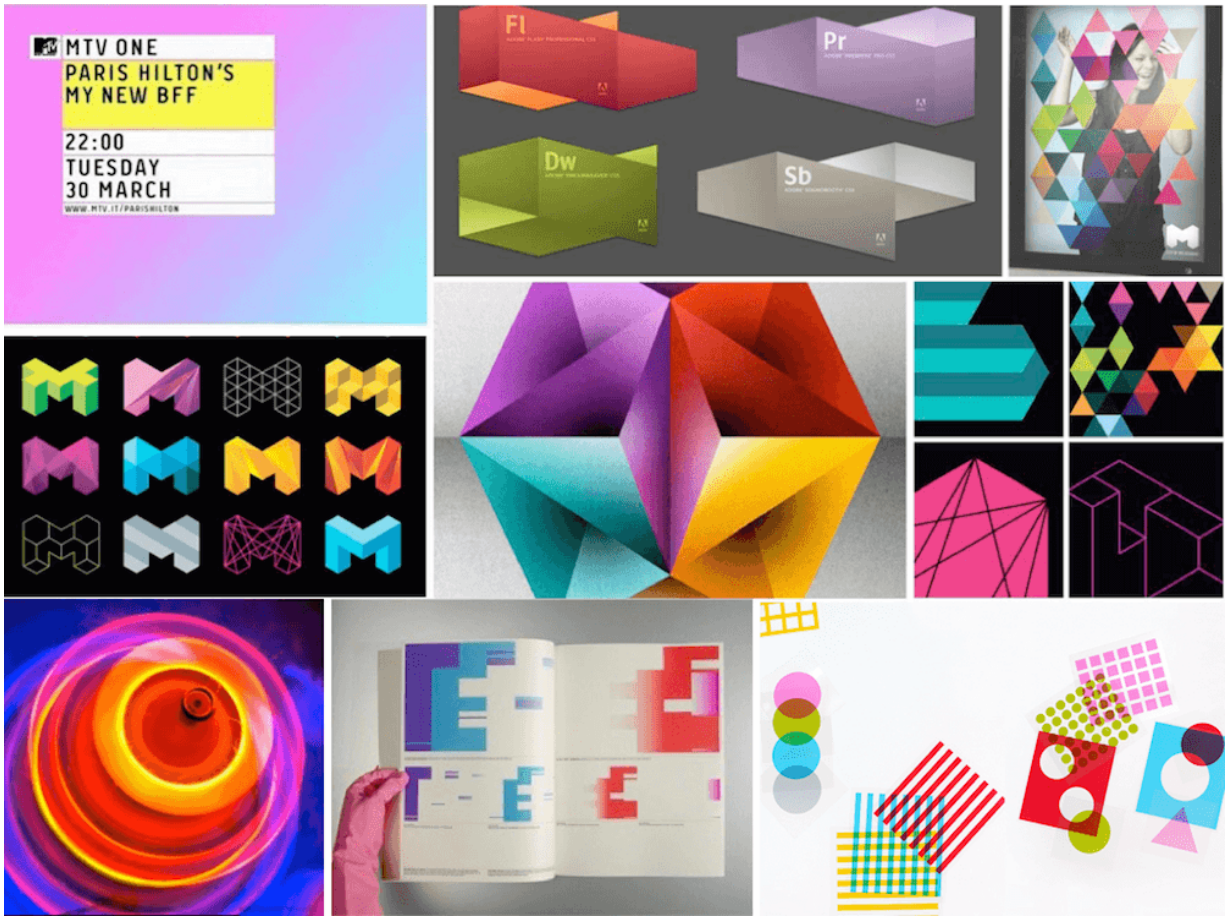
Teams can then identify ways to bring those traits to life through the interface: the tone of voice, visually, and in other ways, such as interactions and sounds. For MailChimp, some of the visual perceptual patterns (or a “visual lexicon,” as Walter refers to it) included a bright yet slightly desaturated color palette, simple sans serif typography, and flat interface elements with soft, subtle textures in places, to warm up the space.

Here are some of the popular design techniques that can help explore perceptual patterns.

## MOOD BOARDS

Mood boards are a great tool for exploring different visual themes. They can be created digitally (Pinterest[7] is a popular tool for creating digital mood boards) or assembled physically on a large board, using cut outs from magazines and other printed material.

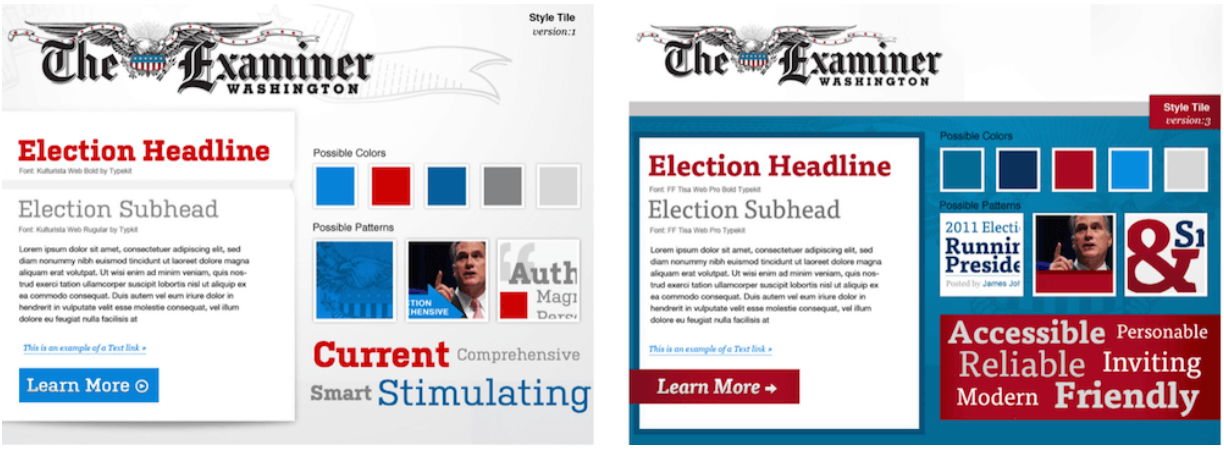
Some people use mood boards to research current trends or gather ideas, while others generate them to work out what their brand might feel like. Mood boards can be broad, or they can focus on exploring specific parts of the brand, like color or typography.




*An example of a mood board that explores color and gradients.*

## STYLE TILES

Once a general brand direction is defined, style tiles can be useful for exploring several variations in more detail. The concept of style tiles was introduced by Samantha Warren[8], who describes them as "a design deliverable consisting of fonts, colors and interface elements that communicate the essence of a visual brand for the web."[9]




*The Washington Examiner 2012 Campaign Site.*

Like mood boards, style tiles can provide valuable discussion points with users and stakeholders, and gauge their initial reactions towards specific design directions. Comparing and contrasting style tiles against each other can help you make a choice of one direction over the other.

## ELEMENT COLLAGES

Riffing on style tiles, Dan Mall[10] suggested the idea of an element collage: an assembly of interface pieces that explore how branding works in the interface. As a design deliverable, they can feel more concrete and tangible than style tiles; yet element collages don't come with quite as many expectations as complete visual mock-ups. Element collages explore not only a general brand direction but how brand is expressed through the interface.


*Element collage for RIF.*

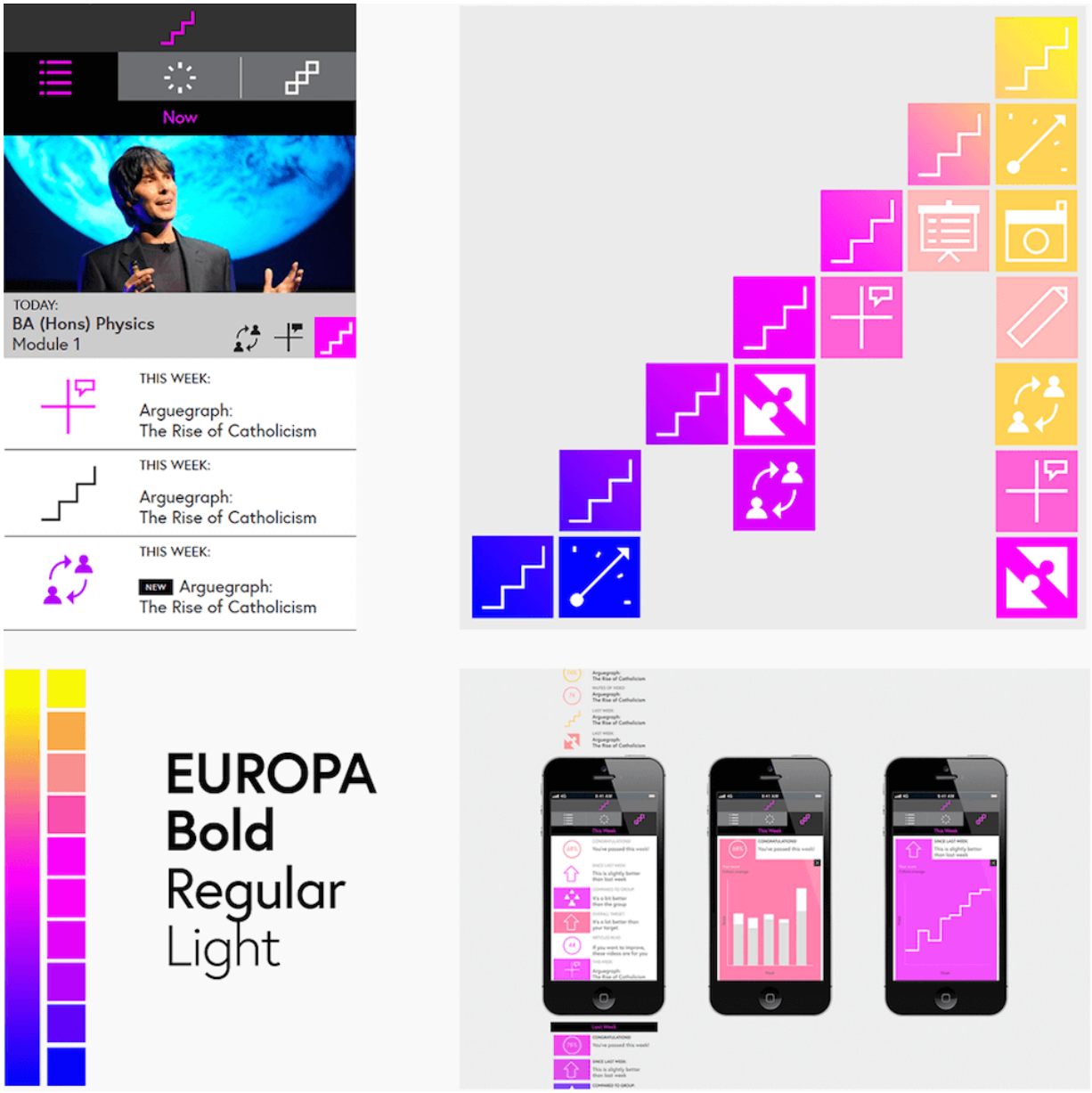
The differences between these techniques aren't always obvious, and people use them interchangeably. For me, the distinction lies in their relative fidelity. Mood boards are looser and more open; they combine existing materials from various sources to create a certain

visual feel. Style tiles and element collages are more focused on a specific product and the practical application of patterns in the interface. Element collages provide the highest level of fidelity and can be used as a starting point to transition into full comps.

## Iteration And Refinement

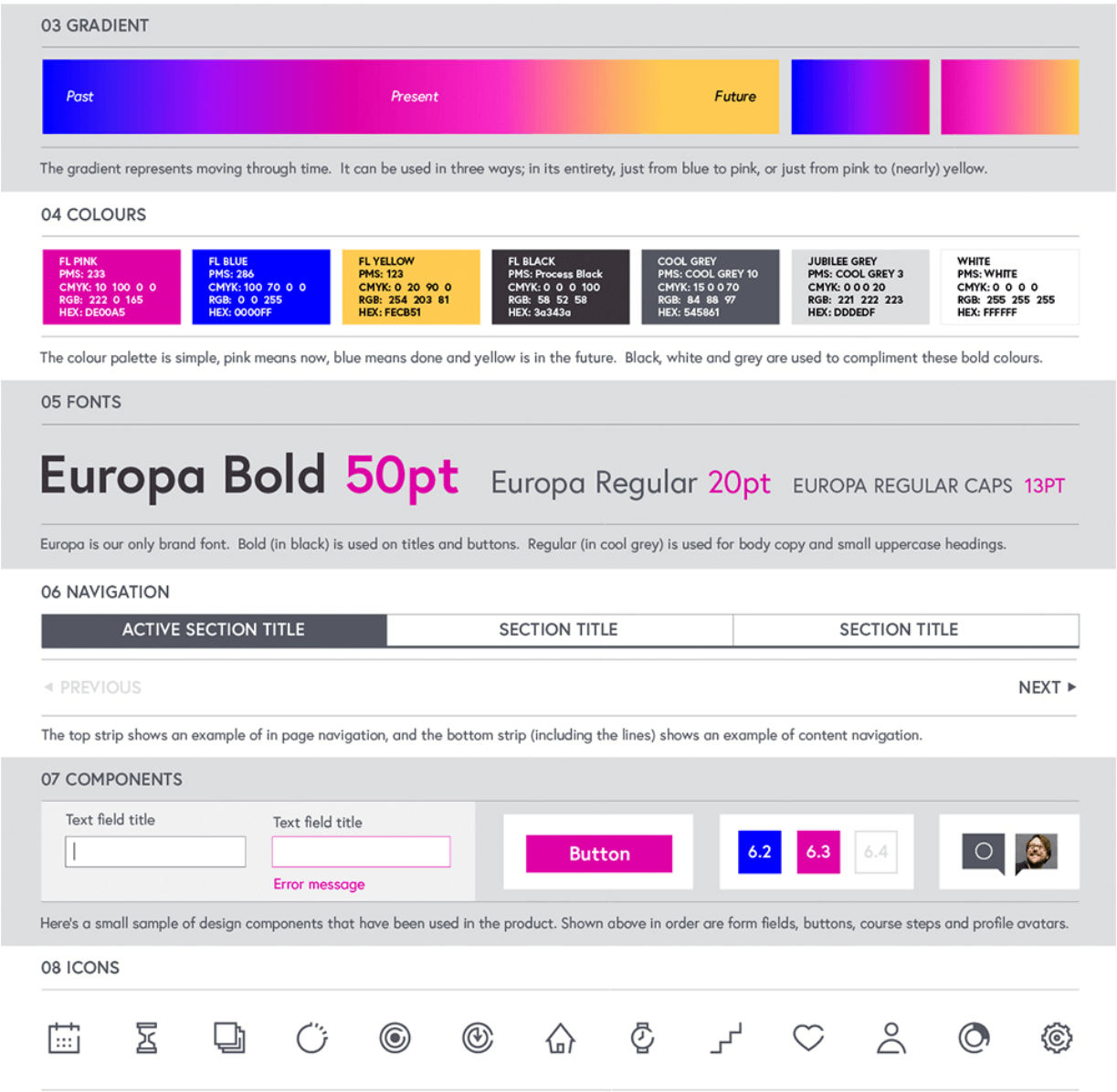
The process of evolving the styles continues when they're integrated into the product. Trying out the brand in a more realistic setting of the interface, with modules and interactions, often results in refinement of both perceptual and functional patterns. This is typically an iterative process, where the patterns influence one another, until a design language starts taking shape.

Let's take a look at how FutureLearn's styles were developed. The image below shows the initial explorations by Wolff Olins[11] for the FutureLearn brand. While they capture some of the personality FutureLearn was trying to project (minimal, bold, bright, positive, celebratory), there's a difference between the initial direction and how it evolved over time.



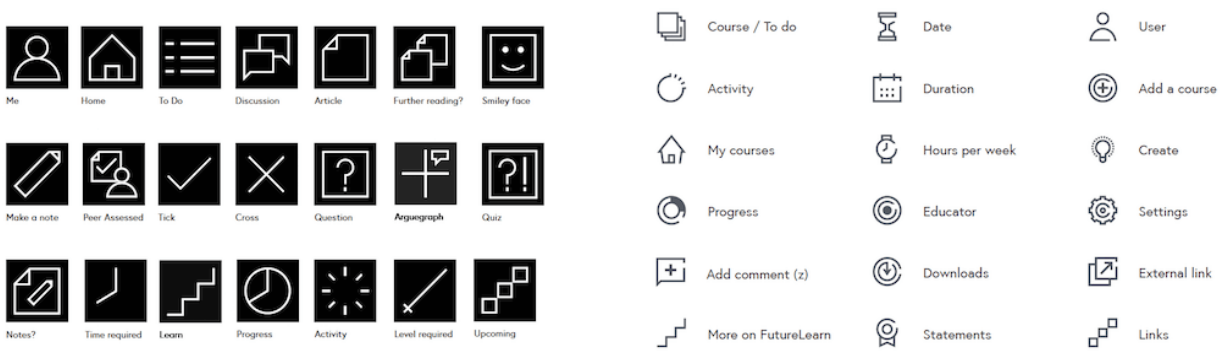

*Initial brand explorations for FutureLearn.*

Here is how the core perceptual patterns looked a few months later, after the visuals were passed on to the internal design team at FutureLearn:




*Element collage for FutureLearn.*

By applying them to the actual context they were going to live in, the patterns had to become more grounded, more specific and more practical. Here's how FutureLearn's iconography evolved from initial concepts to the designs you can see on the site today:




*Initial icon designs by Wolff Olins (left) and how they were evolved by FutureLearn's design team (right). The gaps in the icons signify that a learning process is never complete.*

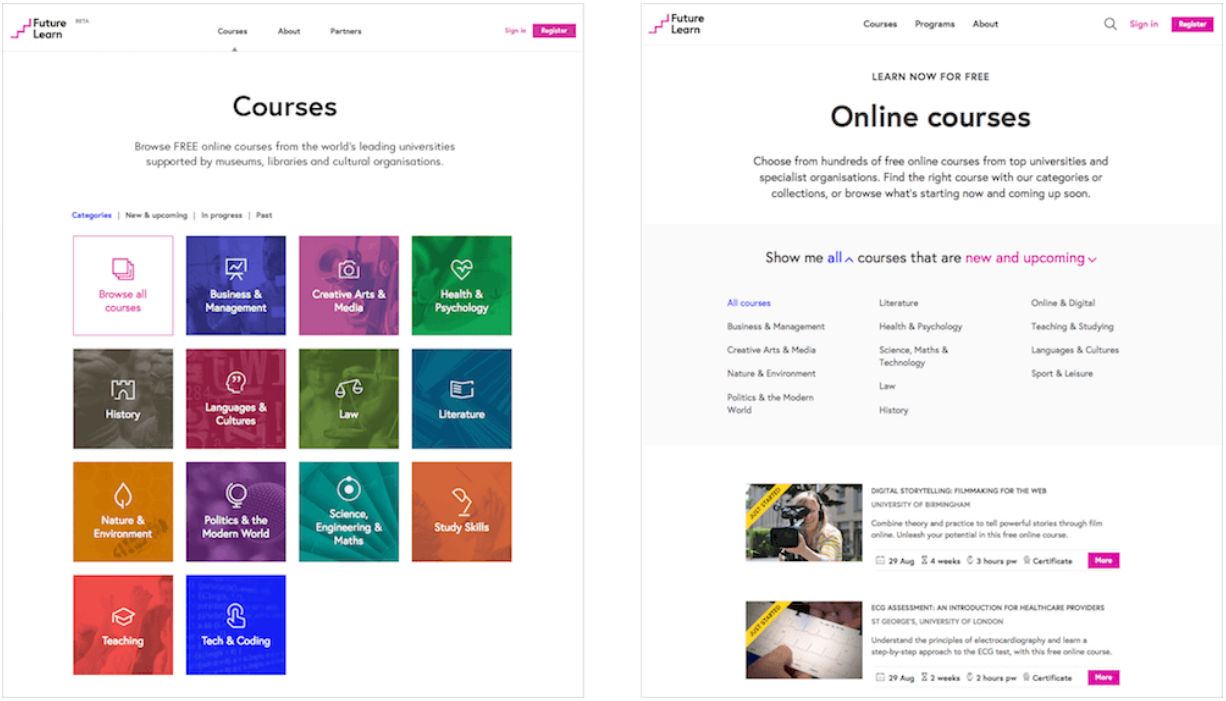
In the conceptual phase of brand development, it was important to go broad and big, and not worry about every detail. But when we started applying the concepts, they had to be refined, so they felt appropriate for the new environment they occupied. The focus shifted from an open exploration to refinement and consistency.

At this stage the challenge is to continue evolving the brand, while keeping the patterns consistent. As Lucy Blackwell, creative director at FutureLearn noted: "When you're fully focused on consistency, some of the important subtleties of what makes a product feel a certain way can be lost."

## BALANCING BRAND WITH CONSISTENCY

Just as introducing too many exceptions can weaken a brand, too much focus on consistency can also stifle it. Paradoxically, making design perfectly consistent doesn't guarantee it's going to be "on brand." Sometimes it can have the opposite effect — there's a fine line between consistency and uniformity.

With seven designers working in several streams, at FutureLearn we had to set up processes that allowed us to focus on reusability and utility. But in some areas of the site we found ourselves following them too closely, sometimes at the cost of weakening the brand. Here's how the courses page has changed over time:



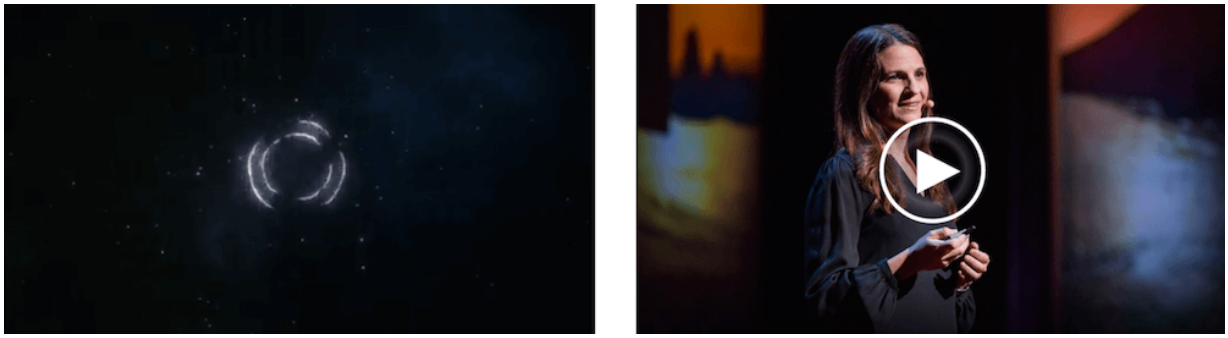

*FutureLearn courses page in 2015 and late 2016.*

It made sense to make the design more practical, clean and organized. It was also a positive change for SEO and metrics, and was more consistent with other pages on the site. But at the same time we felt that in the process some of the visual distinctiveness present at the beginning was lost. While we accepted this change in some areas of the site, in others — particularly in branded marketing campaigns — we started experimenting more, in the attempt to make a stronger brand statement.

If a design system only prioritizes perfect consistency, it can become generic and rigid. Evolving perceptual patterns requires room outside of the boundaries of the system, and designers should be actively encouraged to explore. A good design system strikes a balance between consistency and creative expression of the brand.

## A NOTE ON SIGNATURE MOMENTS

Perceptual patterns can be concentrated in the smallest details. In his book *Microinteractions*[12], Dan Saffer coined the term "signature moments" — small interactions that become product differentiators, such as "an elegant 'loading' animation or a catchy sound ('You've got mail!')." Signature moments are especially powerful when they have meaning or a story behind them. For example, the subtle ripple effect on TED's play button was inspired by the iconic drop animation of their videos' intros.



*TED's drop animation of their videos' intros mirrored in the ripple effect on the video play button.*

In digital products, signature moments are not always seen as a requirement, and some teams struggle to prioritize them.[13] But it's the small details that can add an additional layer of depth and meaning to the experience. In our efforts to systemize and structure

design it's important to be conscious of the details that make something feel distinct. In a design system, there always needs to be space to nurture and evolve those moments.

## SMALL-SCALE EXPERIMENTS

How can we make a space for creative explorations? And how do we then fold the new styles into the system? At FutureLearn, we found that it was most effective to experiment on a small scale first. If some of the elements worked well, we'd gradually integrate them into other parts of the interface.

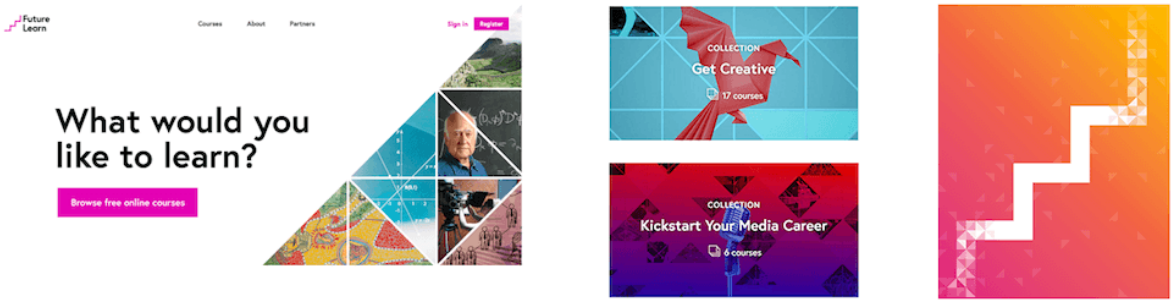
For example, we felt that the purely functional toggle button lacked the feel of celebration and accomplishment when learners completed a step. It was replaced with a circular style, a bouncy animation and a tick icon popping up.


*Original progress button (left) and redesigned button (right) on FutureLearn.*

While the new button received positive feedback from our learners, it didn't feel like part of the system until we started echoing the circular pattern, the bouncy animation and the ticks in other areas of the site. Without these additions, the element felt disconnected.

On occasion we tried out new patterns on promotional areas, such as the home page or a campaign site. FutureLearn's brand used to

employ primarily square shapes. During a home page redesign, we introduced a triangle pattern. It was strengthened when other designers started applying it in other areas, such as image styles and certificate designs.



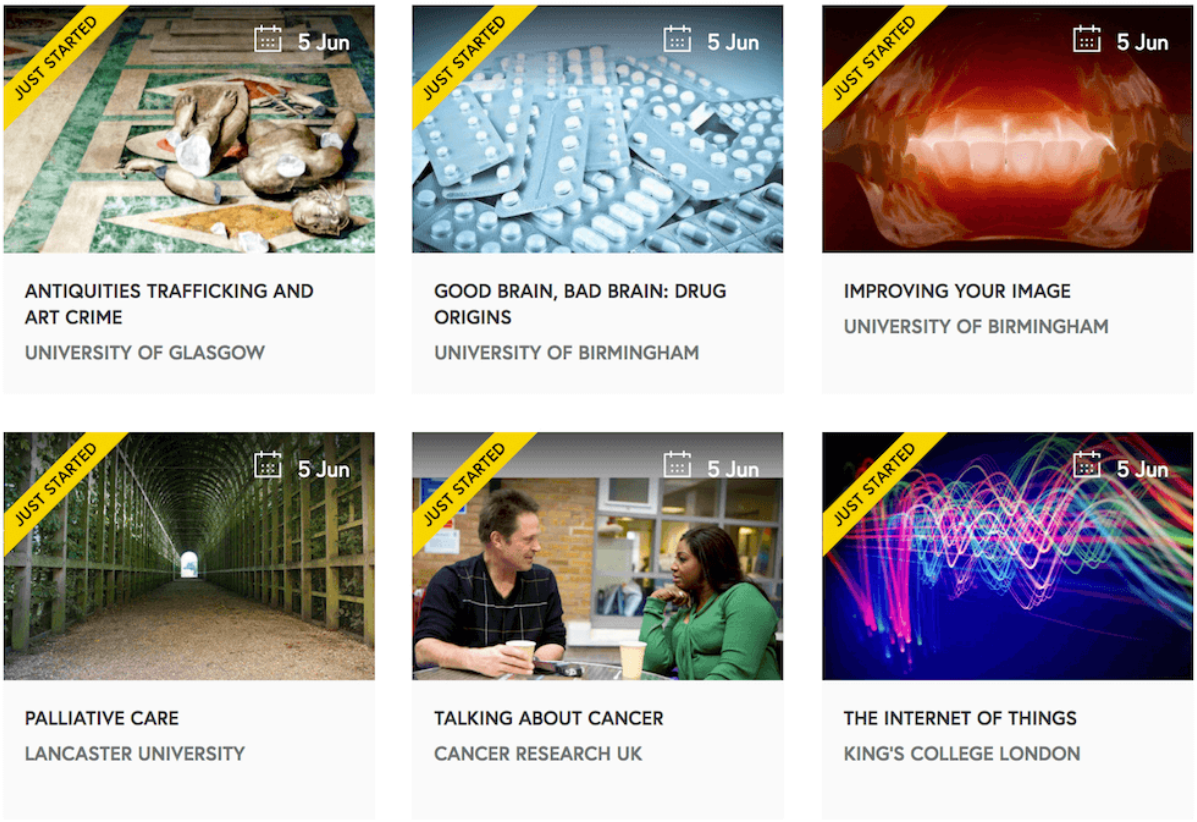

*Initial experiments with triangles on the home page started off quite flat (left), but were given a new twist by other designers who took the pattern and applied it to their projects.*

While experimenting with the triangle patterns, we were aware that they were outside of FutureLearn's typical square aesthetic, but wanted to give them a try to see what would happen. We learned later that while triangles worked with the brand, they had to be used sparingly and only as a visual enhancement in the discovery and marketing areas of the site, not on the in-course learning experience pages.

When exploring new styles, try them out on a small area of the site. Be aware of what you're doing differently, the patterns that are outside of the system, and the reasons for trying them. If they work, gradually fold them into the system by applying them in other areas of the site. Be conscious of the role they play. For FutureLearn, triangles are used to create a dynamic effect; circles are used as a positive reassurance of progress, typically in combination with a tick.

## BALANCING BRAND WITH BUSINESS REQUIREMENTS

Because perceptual patterns are sometimes seen merely as styling or decoration, changing them can be less contentious than, say, updating the flow of an interaction. As a result, ad hoc business requirements can lead to introducing of elements that don't sit comfortably with the brand. For example, we wanted to let our learners know when new courses were starting, so we added yellow banners to the course images.




*“Just started” banner on FutureLearn for highlighting recently started courses.*

Even though the banner wasn’t perfectly on brand, it wasn’t a problem since only a few courses were running at the time. What we didn’t realize was how many more courses would start within a few

months. When it happened, the balance of the course module shifted from feeling like a highlight, to appearing a bit garish and sales-oriented, which is a feeling we try to avoid in the FutureLearn brand. Again, because this was seen as part of styling, it was hard to prioritize and took a long time to address.

No matter how much we guard the brand, these things will happen — new requirements demand custom patterns and one-off tweaks. If we're not conscious of them, such exceptions can dilute or weaken the brand.

## Signature Patterns: A Team Exercise

Sometimes even a small change can alter perception. At FutureLearn, we once almost replaced the square shapes in the course progress module with circles, before we noticed that doing so would completely change the feel of the interface. To control how something feels, you need to understand the exact patterns that influence it.

In your team, try running a quick exercise. Ask each person to write down the most distinct perceptual patterns for your product. Encourage people to look beyond the building blocks such as color palette and typeface, and instead think of high-level principles, combinations and treatments that are specific to your brand. Think not only about the elements, but the *meaning* behind them — the images they portray and the stories they tell. In FutureLearn's interface, some of those patterns are:

- Positive and encouraging tone of voice
- Primarily white color palette with pink accents
- Generous white space
- Generally large type size with a focus on readability
- High-contrast typography with proportionally large headings
- Vibrant pink to blue gradient
- Pink to blue subtle hover interactions
- 1px light-gray strokes
- 1px square icons with a “break”
- Mostly square and occasionally circular shapes, triangles in promotional areas

Similarly to the exercise with finding shared themes in you design principles described in the second chapter, comparing your team’s thoughts can surface different ways in which you perceive your brand. The qualities might be vague and indistinct at first but they’re a great basis for discussions. Reaching a shared understanding of the most distinct perceptual patterns is a useful starting point for your future work on systemizing them.[14]

Patterns and principles are an important part of a design system. But, of course, if you work in a team, they’re not enough. A selection of words and rules doesn’t make a language. It only starts becoming a language when you attribute meaning to those words and other people start sharing this meaning.

—

1. This way of thinking can make visiting places a whole different experience. Be it a coffee shop, a new city, or a picnic spot, I like to think about how a place feels and then try to determine some of the patterns that combine to create that atmosphere.
2. It's Ulysses, in case you wanted to know.
3. Spotify's vision "The right music for every moment" and their design principle "emotive" are inline with the feel created through perceptual patterns.
4. These screenshots were taken in March 2017. Interestingly, a couple of weeks later Vox changed their design to a denser newspaper-like feel, to come across as more "credible and smart". See "Behind Vox.com's Homepage Refresh" for more details.
5. http://smashed.by/designingemotion
6. *Designing for Emotion* by Aarron Walter (see also "Personality in Design")
7. https://uk.pinterest.com/
8. http://samanthatoy.com/
9. http://styletil.es/
10. http://danmall.me/
11. http://www.wolffolins.com/
12. http://microinteractions.com/
13. For great practical advice on designing microinteractions and integrating them into a product, see *Microinteractions: Designing with Details* by Dan Saffer.
14. In chapter 9 we'll talk about conducting an interface inventory on perceptual patterns and integrating them into the system.

CHAPTER 5

# Shared Language

*This chapter describes how to establish a shared language, which allows a group of people to create and use patterns cohesively for a particular product.*

Digital products are built by teams. Everyone will have their own specific goals to accomplish and personal deadlines to meet. Inevitably, this means that sloppy patterns will be added, or that modules will be duplicated or misused.

Can we make sure a product still feels cohesive and whole, even when many people work on it? Yes, if we have a shared understanding in the team of what our design system is and how it works. This means that we follow the same guiding principles, that our vision of the brand lines up, that we have a shared approach to design and front-end architecture, and that we know our most effective patterns and what makes them work. In other words, creating cohesive design systems requires a *shared language*.

In *The Timeless Way of Building*, Christopher Alexander introduced the idea of a “pattern language” as a way to create buildings that feel alive and a pleasure to be in. At the core of his book is the observation that many great buildings (Chartres, the Alhambra, Brunelleschi’s Dome) weren’t created by one master architect working laboriously at the drawing board. They were built by groups of people who all had a deep, shared knowledge of the design patterns that were capable of bringing those buildings to life.

> *“…groups of people can conceive their larger public buildings, on the ground, by following a common pattern language, almost as if they had a single mind.”*
> *— Christopher Alexander,* The Timeless Way of Building

A similar idea can be applied to building digital products. The language can empower people to create products that feel whole, even when multiple contributors work on different parts. Naturally, some people will be steeped in it more deeply than others, but the idea is that everyone — designers, developers, researchers, product managers, content producers — should have some degree of fluency, and that the fluency improves over time, as the team continues to learn, use and evolve the language.

But what Alexander doesn’t mention in his book, is exactly how much work the pattern language approach takes to achieve. Medieval cathedrals took decades to build, and stonemasons went through years of apprenticeship to learn the pattern language. Similarly, it can take a lot of effort to establish a shared language in a product team and make it work.

But it can be done, even in larger teams. We can start by paying attention to the language decisions we make.

## Naming Patterns

James Britton, an influential British educator, explains in *Language and Learning*[1] that by conferring names on objects, we start "bringing them into existence," just like children use language "to draw out of nothingness" the world around them. Similarly, if an interface object doesn't have a proper name — a name that is known and makes sense to people in your team — then it doesn't really exist in your system as an actionable unit to work with. Once you name an object, you shape its future; if you give it a presentational name, its future will be limited because it will be confined by its style: a "pink" button can only be pink.

A well-chosen name can become a powerful tool for shaping your design system, since names affect how patterns will be used. And, of course, it is not only about the names themselves, but a *shared approach to naming* within your team.

## LEARNING WHAT MAKES A GOOD NAME FOR YOUR TEAM

A "good name" means different things for different teams. Sometimes it takes experimentation to find an approach that works. The team at Sipgate[2], a German telecommunications company, used presentational names initially. But they found that names such as "Prominent tile" or "Circle with a dot" weren't effective and ultimately contributed to the fragmentation of their system.

---

*"The main problem with presentation based naming is that you can't find what you are looking for when the number of*

*patterns in your library increases. It also gives you no guidance or inspiration for where to use a specific pattern. People start to build more and more new patterns instead of reusing and enhancing the existing ones, which makes the problem worse and worse over time."*
*— Tobias Ritterbach, experience owner, Sipgate*

---

At Atlassian[3], components are named from a user's perspective. For instance, "Lozenges" and "Inline Edit" got their names because that's how their users referred to them. At first it was seen as controversial and an overhead for developers. But the team felt that naming modules this way enabled engineers to think from a user's perspective and always have users in mind.

---

*"It's a bit of an overhead for engineers, but to a certain extent we're driving user empathy."*
*— Jürgen Spangl, head of design, Atlassian*

---

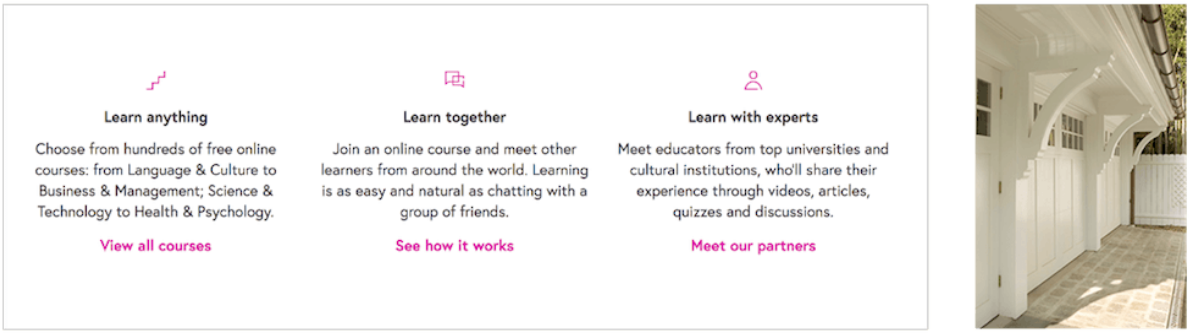
At FutureLearn[4], we agreed that a good name means that it is *focused, memorable and embodies the purpose of the module it represents*. It is the name people relate to and *want to use*. Like other teams, we tried different ways to name things — from more precise and descriptive such as "Progress toggle button," to fun ones, such as "Whisperbox." Having looked how the team used the

modules, we noticed that the most effective names in our system had one or more of these qualities: they were metaphorical, they had a personality, and they communicated the purpose of the pattern.

## Good Names Are Based on Metaphors

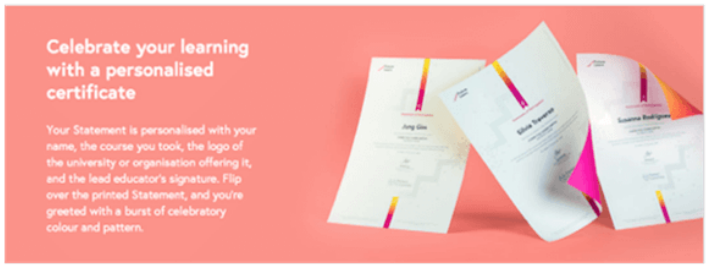
A metaphor from other industries, such as architecture or publishing, can inspire a good name. It can also make the name more memorable for the team, as they'll have an association — something to imagine — when they think of that module.

Brackets in architecture are elements that support and strengthen a structure; for example, helping to hold up a roof. Similarly, in FutureLearn's interface "Brackets" also support the main content by presenting small chunks of additional information.




*"Brackets" support the main content of the page by providing additional information.*

Another example is "Spotlight": a promotional element designed to draw attention to a specific piece of content.

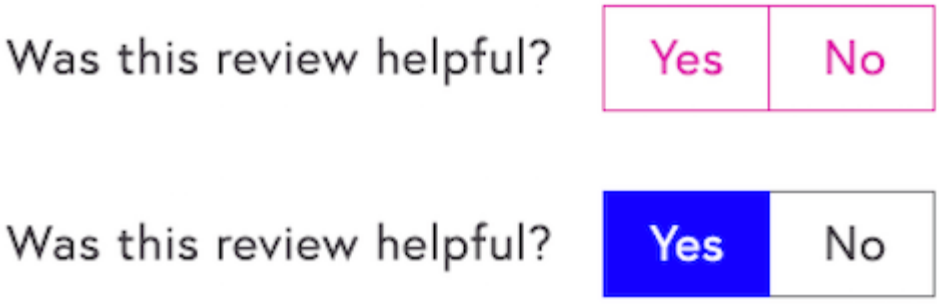



*“Spotlight” is a promotional element designed to draw attention to a specific piece of content.*

“Bracket” and “Spotlight” are examples of effective names — people on the team know and use them. Conversely, names that don’t have a visual metaphor associated with them turned out to be less effective.

For example, hardly anyone can remember what a “Progress toggle button” is, or what a “Binary radio button” looks like.


*Progress toggle button.*




*Binary radio button.*

The problem is that neither “Progress toggle button” nor “Binary radio button” creates an image in your mind that makes it easier to remember those patterns. Even if some people preferred a more precise and descriptive name like “Binary radio button,” the fact is no one remembered it. And if no one remembers it, there’s a high chance that people will recreate the pattern instead of reusing it.

## Good Names Have a Personality

Some names for buttons worked much better. There are small, secondary function buttons that are used all over FutureLearn’s interface. They are called “Minions.”


*Minion buttons.*

And, of course, where there are minions there should also be a boss. FutureLearn’s “Boss” is a large button, typically the main call to action on a page.


*Boss button.*

It's also fun to see `.minion` and `.boss` class names used in CSS.[5]

```scss
// Tiny button
// ------------
.a-button--minion {
  @include button-sizing("minion");
}


// Large button
// ------------
.a-button--boss {
  @include button-sizing("boss");
}
```

*`.minion` and `.boss` class names in CSS.*

Names with personality are easier to remember. Not only do they stick around, they also inspire other names, and can even establish a family of names. A “Whisperbox” module on FutureLearn was designed to be promotional without being too distracting. When another team needed something more prominent, they created a “Boombox.” “Whisperbox” and “Boombox” are a pair, which helps to make the vocabulary more connected and memorable.

## Good Names Communicate the Purpose

The best names offer guidance or inspiration for where to use a specific pattern. It’s easy to remember there can be many minions (on a page) but only one boss. People enjoy using them and we don’t need to enforce guidelines because they come with the name. Even a few names like this can help make your vocabulary more compelling, and the team will be more likely to use it and contribute to it.

Names let us not only identify and distinguish between patterns, but also describe what they’re for. Naming patterns is especially hard when we haven’t fully understood their purpose. If you find yourself struggling to come up with a name, chances are something isn’t quite right. Maybe a module’s purpose is unclear, or it overlaps with another’s; either way, it’s a red flag you should notice.

One of the modules in Eurostar’s new interface was introduced specifically to improve SEO. During a pattern library workshop, the team struggled to come up with a name for it: “It’s an SEO module! It doesn’t have a function. There’s no meaning to its existence!”


*A module introduced to the Eurostar interface specifically to improve SEO.*

The name they gave it in the end was “le blurb.” And the purpose of le blurb was something along the lines of: to provide potentially interesting information with a potential for SEO.

Approaching difficult naming decisions with humor can help. And yet it’s almost never difficult for no reason. At some point we have

to ask ourselves: what's wrong here? Why can't we come up with a name?

## NAMING COLLABORATIVELY

We can understand the purpose of a pattern better if the naming process in our team is collaborative. That doesn't mean involving the whole company: it is not about the number of people, but a diversity of perspectives. Too often the responsibility of naming modules falls on developers when they write CSS, but it can be made easier (and more fun) if the naming process is broadened to include more people from the team.

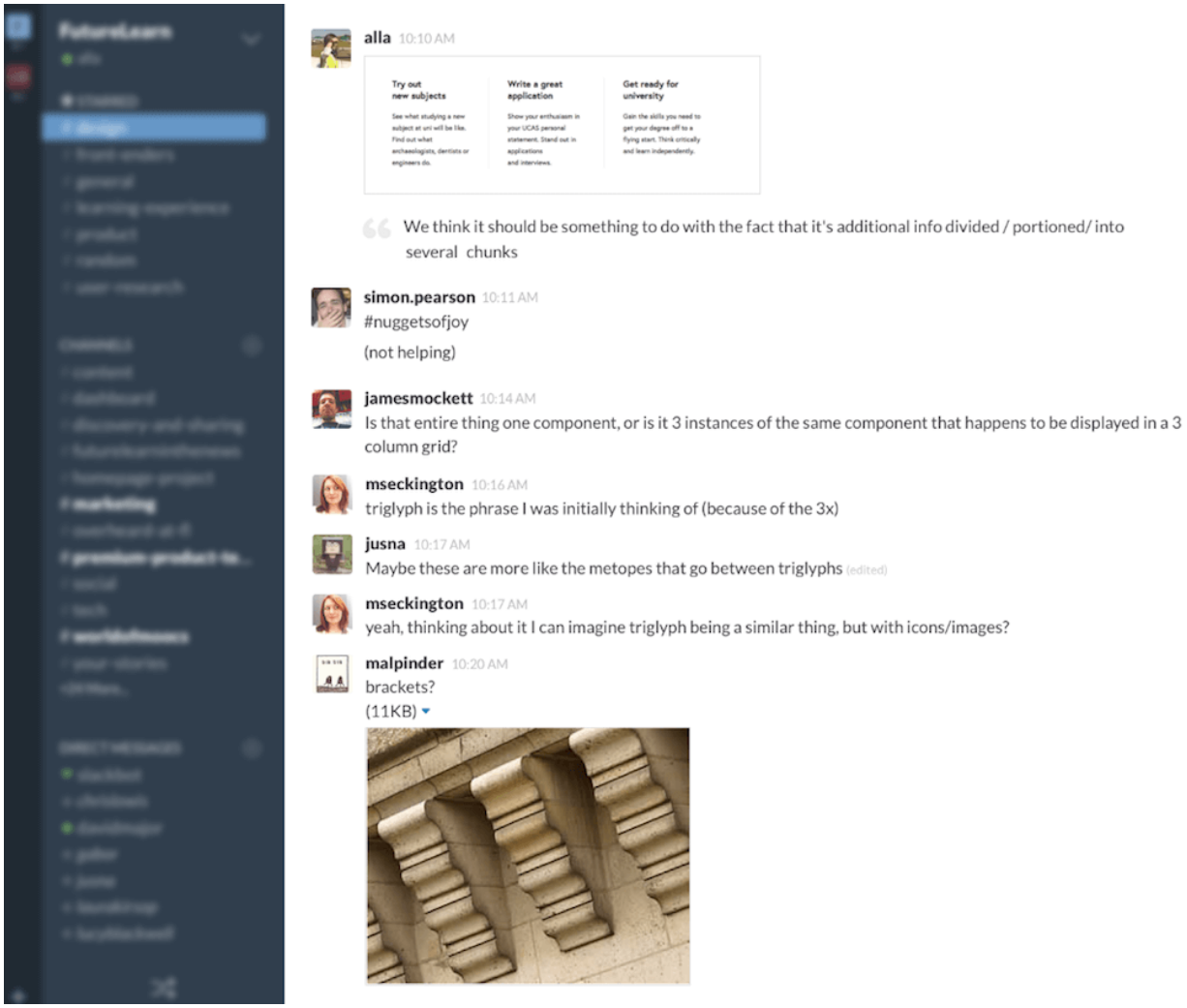
People from different disciplines will each view a module slightly differently. Someone from a content background might see it in a generic way because they'd need it to be flexible. Developers might see technical specifics, because they're aware of how the module is built — they'll know that something is a radio button even though it might not look like a radio button. Designers and user researchers might be more familiar with the original behavior a pattern is meant to support. Involving people with different points of view can help make a more informed and objective decision about the purpose of a module. And once you know the purpose, coming up with a name is easier.

Collaborative naming will also help team members who did not design or build the patterns understand their use. Involving the content team, for example, can help engage them as participants in

this process, rather than simply being handed a stack of boxes to fill in.

## Set Up a Dedicated Channel

One of the easiest ways to collaborate on naming is to set up a dedicated space on your favorite everyday communication app (like a “Design system” Slack channel). Share a design mock-up or an existing module with the team, and describe briefly what it’s for and what distinguishes it. You might say: “This typically represents additional or supporting information portioned into smaller chunks.” It can help to share the names you’ve come up with so far.




*“Nuggets of joy” could be a great name for a module, but we decided to settle on “Brackets” in this case.*

A few people will join the discussion, ask questions, and suggest their ideas. Some suggestions won’t be quite right, others will be humorous or unexpected. That’s OK — the point is to generate discussion. Talking through design decisions and the purpose of patterns helps to strengthen and evolve the shared language.[6] If you come up with a good name, remember to give people kudos and celebrate it. It’s those moments that help unite the team and make collaborative naming part of your culture.

## Test Your Language with Users

You might want to go even further and involve your users in the language decisions. Try testing modules on paper cards. Testing on cards differs from other user research techniques, which use linear tasks and scenarios for users to work through. Here, participants can pick up the cards, move them around, discuss and scribble on them, actively becoming part of the design process. This can give you a chance to test your language choices and make sure that the modules you've defined are aligned with your users' potential behaviors and mental models. On some occasions you might discover that your "Prominent tabs" are completely ignored, or that your "Wizard control" is interpreted as a set of optional tabs.

*User research with cut-out modules.*

As useful as it is to involve team members and users in the naming process, it's important to stay focused and not get stuck in a loop of endless discussion. Sometimes too much input can lead to diluted or confusing names. To avoid this, at FutureLearn we would take on board the suggestions, but always leave the final decision up to the designer-developer pair who worked on a module.

## Immersing Your Team In The Design Language

Naming things together, however, is not enough to establish a shared language. The entire team should be immersed in it. Make it omnipresent. If you create the right conditions, even people who aren't initially interested will learn passively, through exposure. Here are some tips for creating such conditions.

### MAKE DESIGN PATTERNS VISIBLE

Set up a dedicated space on a wall to represent your design system visually: a pattern wall.

*A pattern wall in the FutureLearn office.*

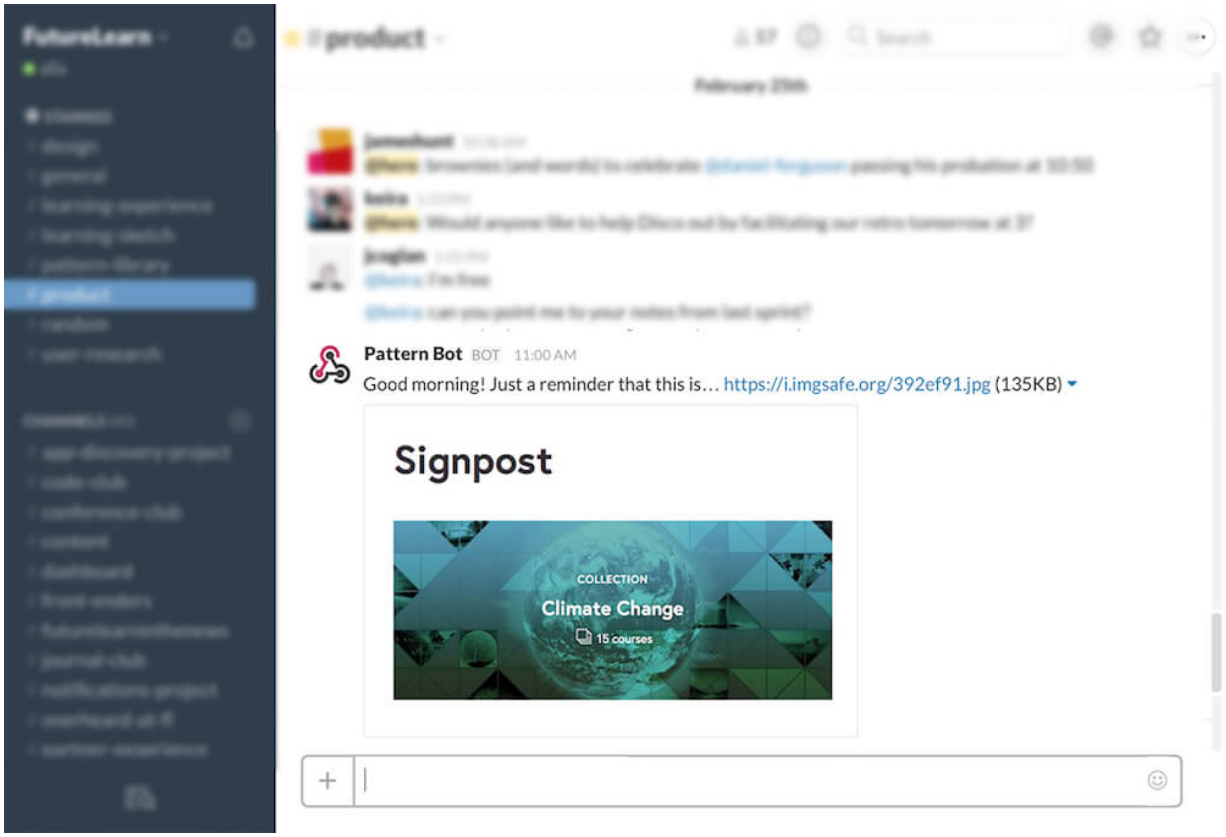
A pattern wall gives you a bird's-eye view of your system. This makes it a great space for naming conversations to happen, as you can refer to things right there in front of you — no need to search all over the site or remember what they look like. Having a dedicated space also makes your system more open: people feel like they are welcome to join in, ask questions and contribute.

You don't need a lot of space to create a pattern wall, and not all the screens of your product have to be there. Start with the ones that are the most crucial or used most frequently. Print them out on A3 sheets, put them up on the wall, and label the most prominent patterns. It can be helpful to position the printouts in the order that follows your most common user journeys; for example: discovery screens, sign-in journey, comparing products, making a purchase.

You can be creative when making your design patterns visible. The team at MOO, a digital print and design company, printed some of the pages from their style guide onto MOO postcards as handy sheets the employees can grab for reference.


*Some of MOO's style guide postcards.*

Automated reminders can also be helpful. It only takes a few minutes to set up a Slack bot that will occasionally remind your team what different elements are called.[7]

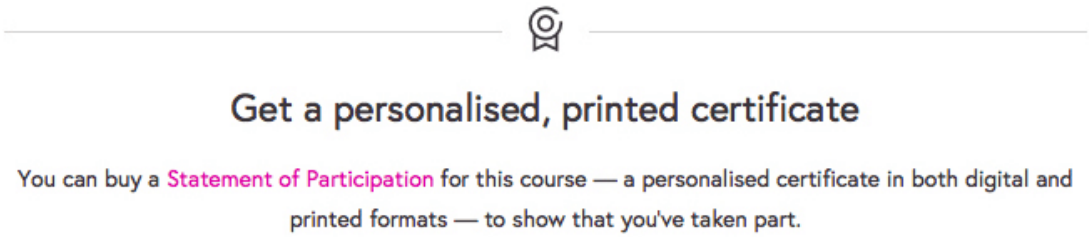



*A Slack bot reminding the team what “Signpost” looks like.*

## REFER TO THE OBJECTS BY THEIR NAMES

Like with any language, you need to use it to keep it alive. It needs to be part of day-to-day conversations. That’s why it’s important to make a conscious effort to keep referring to the patterns by the names you agreed on — no matter how bizarre this might sound.

“Whisperbox” is a promotional module on FutureLearn. As the name suggests, it is meant to be subtle and not draw too much attention to itself.



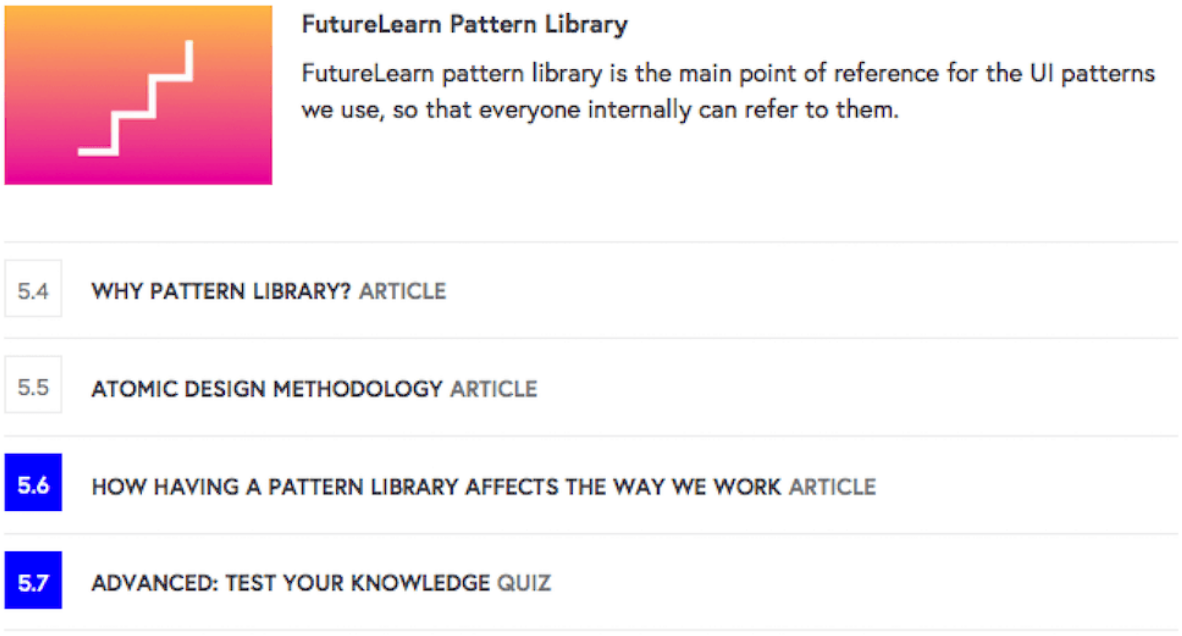

*“Whisperbox”: a subtle promotional module on FutureLearn.*

Until we gave it a proper name, we kept referring to it as “that thing with the lines and an icon in the middle.” It was easy to call it that. It took more effort to start calling it “Whisperbox.” But until you start calling a pattern by its actual name, it doesn’t exist in your system as a solid actionable block to work with. And every time you do use the name, you strengthen the element you call on, and evolve your design language.

Doing so requires a certain self-discipline in the team. It can be hard, especially if you’re not used to it (imagine you joined a team where everyone is talking about minions, bosses and whisperboxes!). But very soon those names become part of a normal conversation and people get used to it. The goal is to get to the point where everyone knows exactly what you’re talking about just by calling a name. Everyone knows the purpose of sequence navigation and refers to it as “Sequence navigation,” and not “Fancy bubbles” or “Wizard control.” Naturally, it also means that *names in the design file and code match*.

## MAKE IT PART OF THE INDUCTION PROCESS

It’s easier to introduce new employees to your design system if it’s part of your induction process. New team members at Atlassian are taken through the story of how the guidelines were created so they can understand why and how the decisions had been made. At FutureLearn, we created an internal induction online course, which includes a dedicated chapter about the pattern library, with quizzes and bite-size lessons.




*A chapter on the pattern library in the FutureLearn’s online induction course.*

## ORGANIZE REGULAR DESIGN SYSTEM CATCH-UPS

Everyone hates meetings. But design system catch-ups are worthwhile if you want to keep everyone on the same page when evolving your system. It’s the time when all designers and developers can fully focus on the system together.

The catch-ups can work with a group of around 16–20 people, and for larger groups people can take turns to attend. They don't have to take long — half an hour is usually enough, if you have a well-structured agenda. Initially, you can run them weekly, and then fortnightly when the team finds its rhythm. Teams can use this time to agree on the overarching patterns, such as icons or typography across the product. It is also a good opportunity to share new modules and discuss their purpose, usage, and any problems and questions people might have.

## ENCOURAGE DIVERSE COLLABORATION

Try pairing on designing and building patterns as much as possible. All the exercises described in the previous two chapters can help collaboration and establish shared language between disciplines:

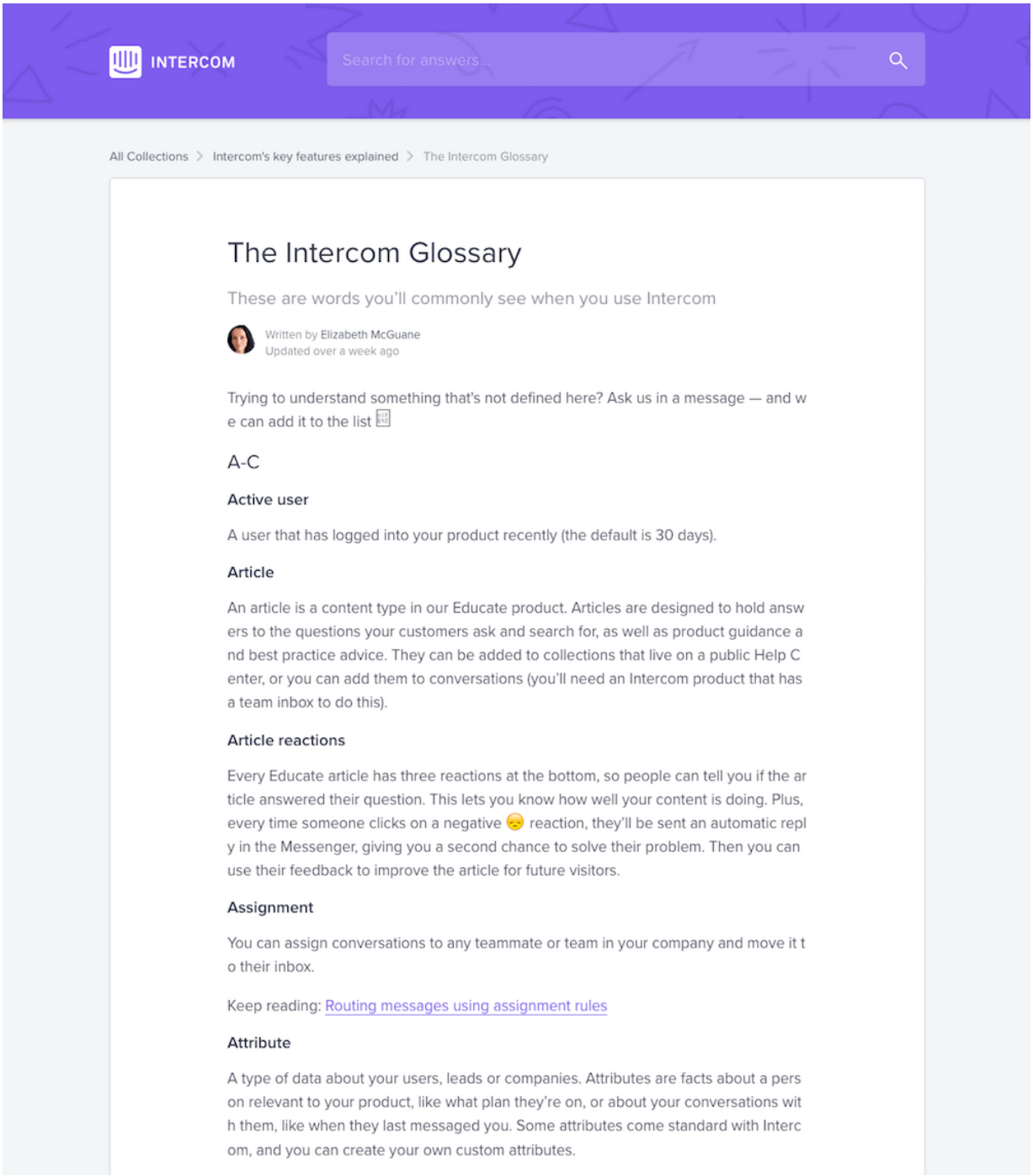
- Creating a pattern map
- Conducting an interface inventory focused on a specific pattern
- Drawing pattern structure
- Agreeing on the purpose of a pattern and coming up with a name
- Articulating the patterns that make your product feel a certain way
- Running small scale experiments with new patterns

In any team there will be people who are more fluent in your pattern language and more enthusiastic to work on the system, and they

might naturally gravitate towards working with each other. But try to encourage them to work with everyone, so that they have an opportunity to share their knowledge and enthusiasm with people who are less immersed in the system. By spreading out the knowledge across an organization, a design system becomes more resilient.

## KEEP A GLOSSARY

One of the most effective practices for sharing and evolving a design language is to keep a glossary of the terms you use. Creating and keeping a glossary allows you to be consciously aware of the words you use, as you have to articulate things to write them down. For example, at Intercom, a customer messaging platform, the team keep a glossary of terms to make sure they use “the same language from code to customer.”




*The Intercom Glossary*[8]

And, of course, an up-to-date, easily accessible pattern library can also be a reliable glossary of design patterns and a reference point

for the entire team (as well as being the actual *toolkit of patterns* for designing and building an interface).[9]

The value of a glossary is not only in the tool it provides: it is also in the *language practices* it cultivates. By establishing and keeping a glossary, you get into a habit of vetting, discussing and articulating your language decisions as a team — you acknowledge that words matter.

Not all teams are equally collaborative and open to discussing design principles and patterns every day. Establishing a shared language requires a certain kind of team culture. But it can also work the other way around — integrating language-focused processes can lead to better collaboration. Three years ago at FutureLearn we didn't have a working shared design language and we didn't collaborate nearly as much as today. Designers were documenting patterns in a PDF brand document and developers built their front-end styleguide. Although both documents were useful for each of the disciplines, they didn't provide a shared language foundation to work with. But by putting practices and processes in place, we gradually transformed how we work.

Establishing a shared language is always a gradual, piecemeal process. Sometimes it will be messy and slow, but if you just keep going, you'll see your language evolving and strengthening. Eventually, the effects will ripple out within your team, with other teams, and with stakeholders, as they start to understand what you're trying to achieve and join you in this process.

—

1. http://smashed.by/languagelearning
2. https://www.sipgate.de/
3. https://www.atlassian.com/
4. https://www.futurelearn.com/
5. In the FutureLearn interface there are more button styles, the examples shown here are some of the most effective ones.
6. What's more, when sharing a new design, someone might spot that a similar module already exists and you can prevent a duplication.
7. Some tools, such as Brand.ai and Frontify, also have integrations with Slack to ping channels when a pattern library is updated. More about tools in chapter 10.
8. http://smashed.by/intercom
9. We'll discuss pattern libraries in detail in chapter 10.

# Summary

In this first part of the book we've talked about establishing foundations for a design system. Here's a summary of the key points.

## PURPOSE

The purpose of a design system is to help achieve the purpose of the product: "Cook a healthy meal in ten minutes"; "Spread the talks as far and as wide as possible"; "The right music for every moment." Everything in a system — from how a team operates, down to the smallest pattern — should be optimized toward that purpose.

## PRINCIPLES

Teams choose how to achieve the product's purpose through design. Their design approach and priorities can be captured in a set of principles: "Design for first impressions"; "Appropriate over consistent"; "Timeless, not cutting edge." The more aligned the team are on their principles, the more cohesive the patterns they create.

## PATTERNS

Through the interface we aim to help people accomplish certain goals and feel a certain way: learn a new recipe; focus on writing; feel productive; feel inspired. Our design intent is rendered through

design patterns. Functional patterns support user behaviors and actions: “select ingredients,” “choose a recipe,” “follow recipe steps,” “rate a recipe.” Perceptual patterns focus on how a product should feel intuitively: “utilitarian,” “newspaper-like,” “openly enthusiastic.” The purpose of the patterns needs to be thoroughly understood by the team. Only then can we make sure it is interpreted as intended by our users.

### SHARED LANGUAGE

The patterns should be connected through a shared language — a deeply rooted knowledge in the team about how to create and use design patterns for a particular product to create effective and coherent user experiences. This knowledge is propagated through a shared design approach, front-end architecture, brand vision, and daily practices such as collaborative naming, cross-discipline pairing, making patterns visible, conducting regular interface inventories, and maintaining a pattern library. The language should be evolved, strengthened, iterated and continuously tested.

## Understanding How Your System Works

A design system is not built overnight, but shaped gradually, through our daily practices. If you are working on a digital product, the foundations of your system probably already exist. One way or another, interfaces are designed and built, and end up in front of users. Unless this process is entirely random, you have a system. The question is, what kind of system is it? Is it flexible and

adaptable, or is it designed for one specific purpose? Is it cohesive or fragmented? Is it easy to work with, or is it time-consuming? Does your system thrive on freedom and autonomy, or is it strictly hierarchical?

To make a design system more effective, we need to know how it works, what it consists of, what makes it function well and what doesn't. If we don't have this knowledge, the same problems keep occurring (systematically!). We tidy up all the buttons and six months later end up with too many buttons again. We can solve a problem but if the mechanism that created it remains, the same problem will keep coming back.

Different design systems work in different ways. Your organization, team culture, design approach, the project, and even the physical space you're in, will shape your system. In the next part of the book we'll start by looking at the underlying structures that influence design systems. We'll then focus on techniques for establishing and maintaining a design system, including: planning; conducting a functional interface inventory; setting up a pattern library; and creating, documenting, evolving and maintaining design patterns.

# Part 2: Process

CHAPTER 6

# Parameters Of Your System

*This chapter introduces some of the qualities a design system can have, and the ways in which risks and downsides can be managed.*

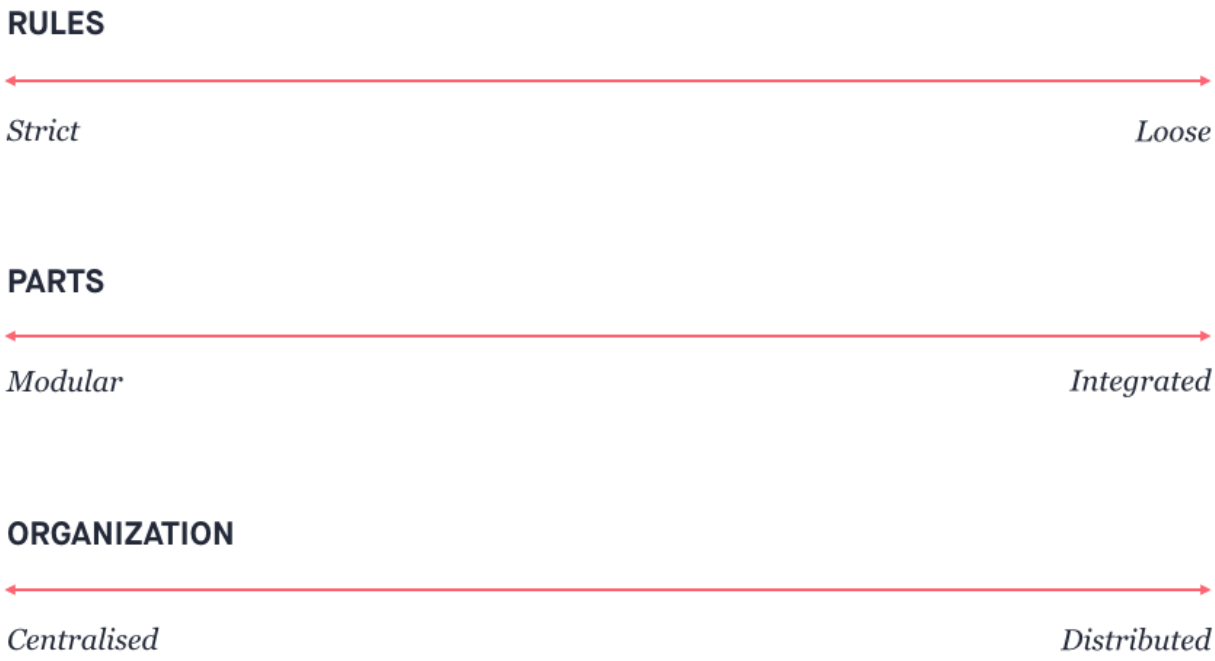
The team at Sipgate, a German telecommunications company, ran into a problem. Duplications and inconsistencies across their product websites were diluting the brand and creating needless extra work for the whole team. They decided to address the issue by establishing a pattern library. After weeks of workshops and interface inventories, patterns across the product sites were standardized. A few months later, the team rolled out a brand new pattern library.

In some companies you might struggle to excite people about your pattern library. They won't see its value, they won't contribute. This was not the case at Sipgate. All the product teams documented their patterns diligently and efficiently. There was no shortage of enthusiasm. But a year later, there were so many modules that it became extremely difficult to find the one they were looking for, and to understand which one to use. It became easier to just add a new one. After a year of maintaining a pattern library, their product websites were still full of duplicated patterns, albeit thoroughly documented ones.

A design system doesn't start or end with building a pattern library. There are many factors that shape a system: the structure of your

organization, your team culture, the type of product you're working on, and your design approach, among other things.

To see how these factors play out, I find it useful to characterize design systems using three attributes: strictness of the rules, modularity of the parts, and distribution of the organization.




These parameters are not binary and all companies lie somewhere on each continuum. As we take a closer look at them, we'll draw on examples from different companies to see the benefits and drawbacks of each direction.

## Rules: Strict Or Loose

Some systems are strict, others benefit from being more loosely set up. Let's take two teams as examples: Airbnb and TED.

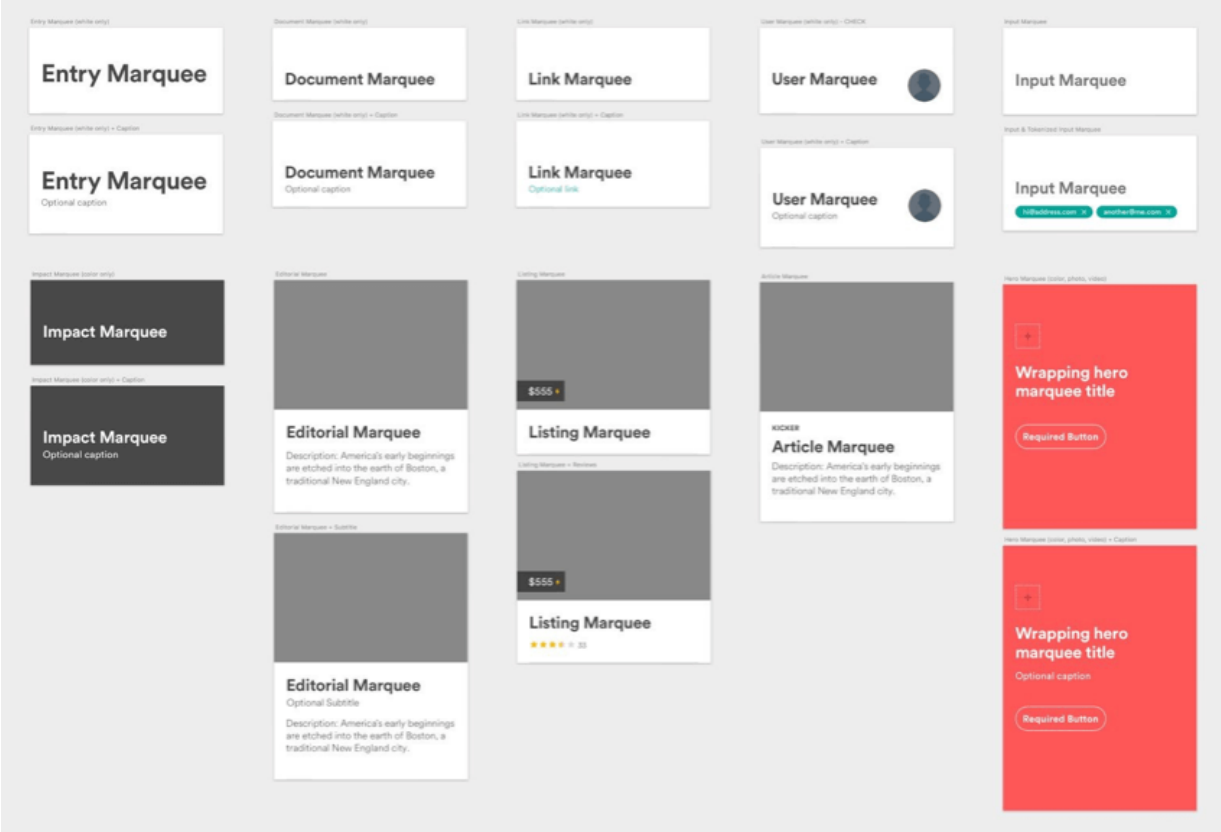
## AIRBNB

Airbnb has over 2,000 employees worldwide and around 60 product designers working across multiple work streams. The system is managed entirely by their Design Language System (DLS) team which consists of six designers and their engineering partners for web, native mobile, and React Native platforms.

The design system at Airbnb is strict: there are precise rules and processes in place that are rigorously followed.

### Standardized Specifications

To minimize deviations, modules in Airbnb's DLS are specified precisely. For example, typography treatments are strictly defined; there is an eight-pixel grid used for spacing; interactions are documented; naming conventions are consistent. Here's how a module called "Marquee" is specified in the master Sketch file. Notice that there are two examples of each one, to show some options available to the designers.



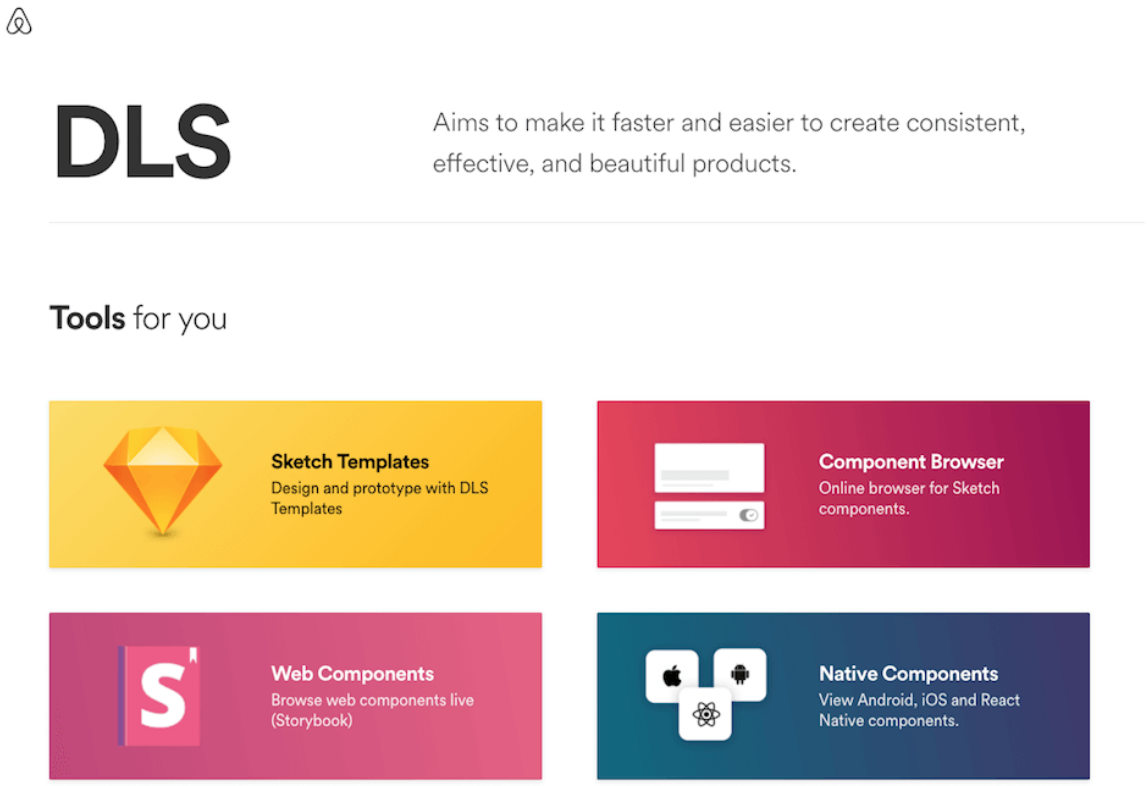

*“Marquee” module in Airbnb’s master Sketch file.*

## Design Is Fully Synchronized with Engineering

In Airbnb’s system, design and engineering are fully synchronized. Specifically this means three things:

- Design modules in the master Sketch file have an exact equivalent in the code, and they are in sync.
- Names in the Sketch file and the code match.
- All modules are cross-platform: every component drawn in the Sketch file has a matching component that is as similar as possible in iOS, Android, React Native, and responsive web libraries.

The synchronization is seen as a priority. The team creates custom tools, such as Sketch plug-ins, to achieve that.




*A gateway to all DLS resources.*

## Strict Process for Introducing New Patterns

The DLS team aims to provide all the patterns required by the product designers across the company. Their goal is to reuse around 90% of the existing modules, so creation of new patterns is relatively infrequent. There's a strict process for introducing new components:

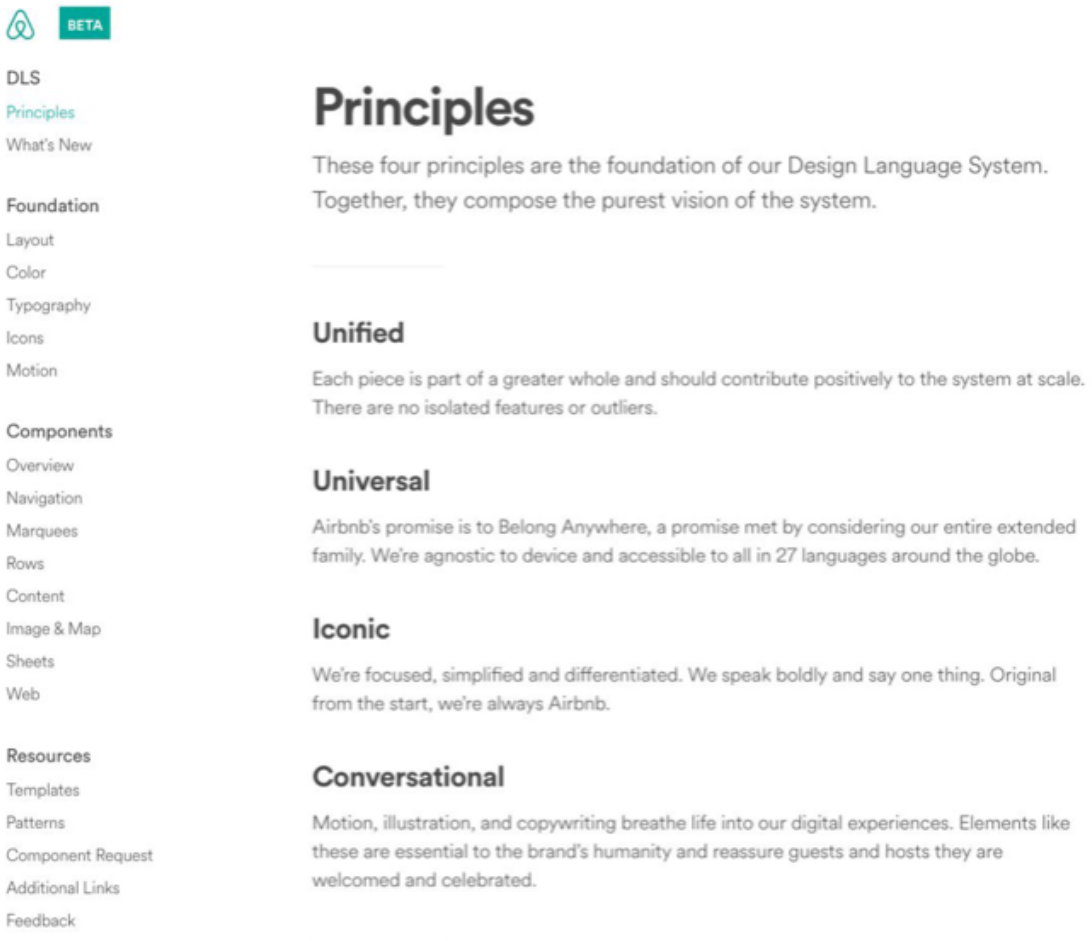
1. A designer submits a proposal using a Sketch template with instructions about related behavior and rules. They suggest a

suitable name and provide an example of how the proposed component can be used in context.

2. The proposal then goes to the product support team via JIRA, along with the Sketch file. In many cases, the support team finds that a similar module exists already, or that an existing module should be updated.
3. If a need for a new module is justified, the proposal goes through to the DLS team, who consider the requirement and decide if the proposed design will work. Sometimes they use the proposed solution, sometimes they adapt or redesign it, to make sure it fits with the system.

## Comprehensive Detailed Documentation

The design language is documented on the internal website, DLS Guidelines, generated from the master Sketch file. Airbnb's tools team built an automated process that generates screenshots and metadata about the components, and publishes them to the guidelines site. Needless to say, documentation is fully in sync with the Sketch file and the code.




*The internal DLS Guidelines website.*

These are some of the practices that make DLS function well as a strict system. On the opposite end of the strictness scale we have companies with looser structures. Such systems are optimized more towards experimentation and sensitivity to context. They can be effective too.

## TED

The team at TED is small, with only five or six key people responsible for the design system decisions: two UX practitioners

and four front-end developers. TED's system is loosely set up. Brand feel and the utility of the page take priority over perfect visual consistency. For example, introducing an additional color or diverging from the standard layout as an exception is not a major concern, as long as it helps to achieve the right effect:

---

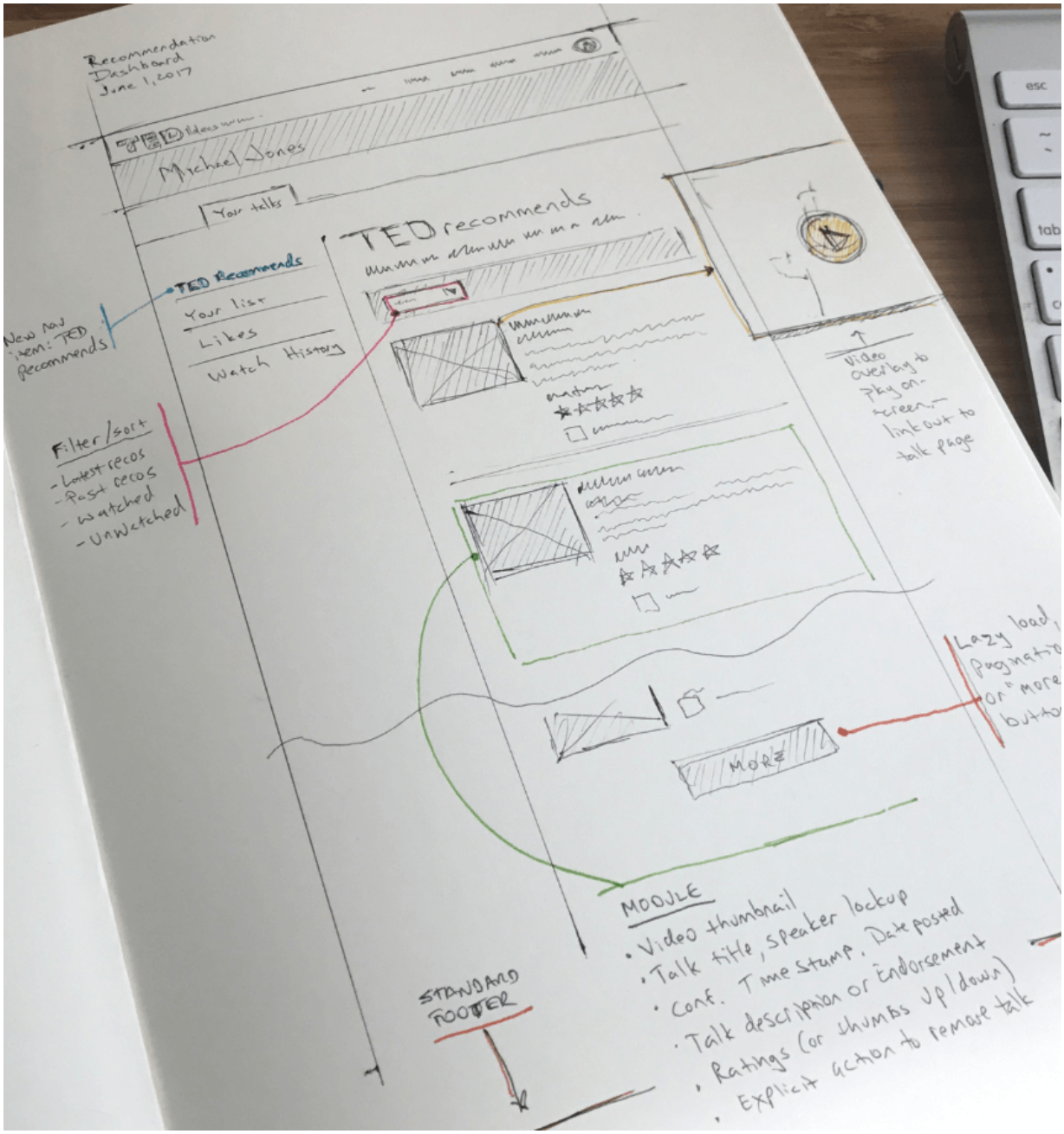
*"Design what's right, not what's most consistent. The best utility of the page is a priority. We'll modify the page to make it work. Dogmatic consistency and established patterns are not what should drive design decisions."*
*— Michael McWatters, UX architect, TED*

---

There's a lot of scope for creative experimentation with this kind of system. Because each page can be fine-tuned, it can adapt to specific contexts and use cases. The designs such a system generates can be coherent, but they're not necessarily perfectly consistent. In contrast to Airbnb, TED's processes are also more relaxed and informal.

## Simple Sketch over Detailed Specs

Instead of detailed specs, TED's team can often use a whiteboard or low-fi paper sketch with rudimentary notes. It is then shared in person or posted in Dropbox or InVision, where the team exchanges comments and feedback. Designers and developers then work collaboratively as they bring it to a higher fidelity.

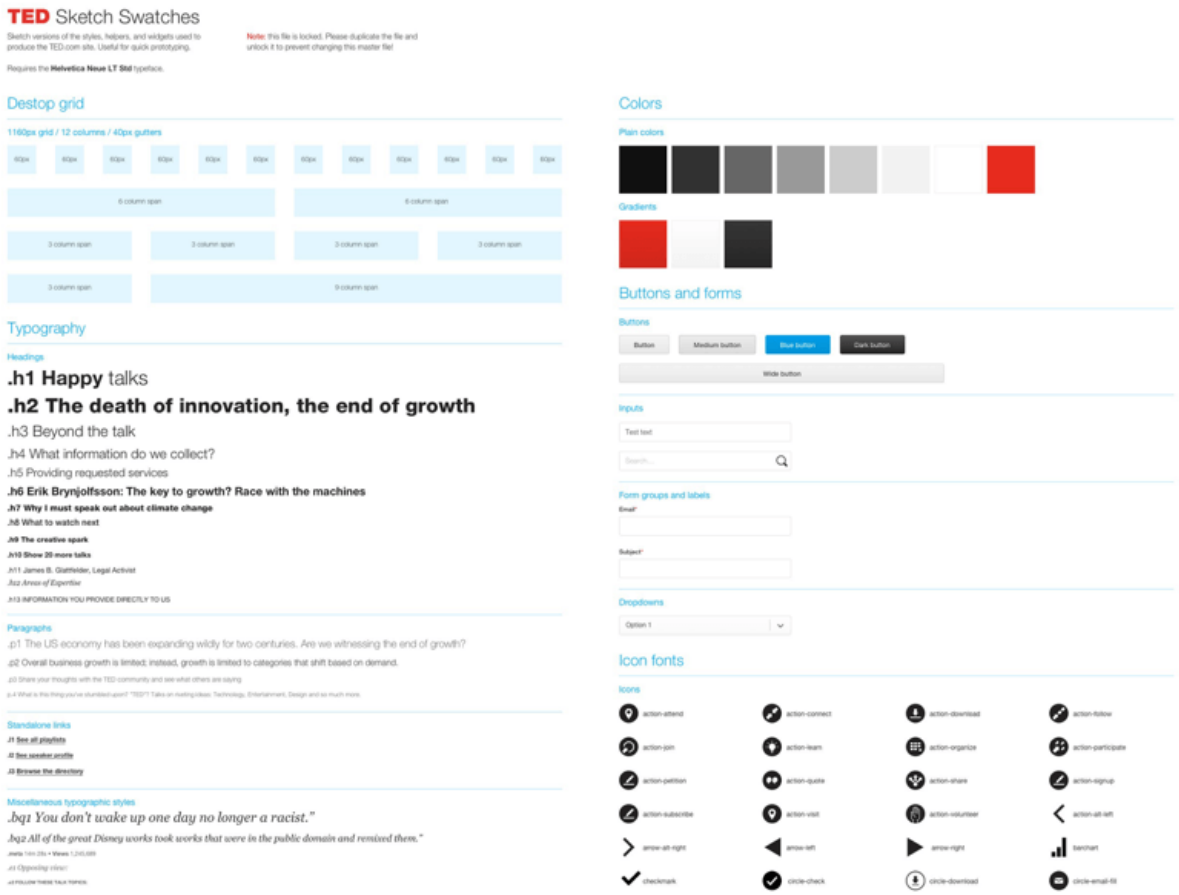



*At TED a simple sketch with notes is often used instead of detailed design spec.*

## Simple Documentation

The documentation is also simple. The team doesn't have a comprehensive pattern library. Instead there's a simple collection of what they call swatches on a web page and in a Sketch file, which

contains some of their core design patterns — but by no means everything.




*Some of TED's swatches, saved in a Sketch file.*

Perhaps you're thinking that this is because TED is small and it simply doesn't have the time and resources to set up a comprehensive pattern library. But that's not exactly the case. So far, there just hasn't been a need to document everything in detail. If the team starts growing, that might change, but they emphasize that even with a pattern library in place, patterns are not going to drive the design. "Design acumen and sensitivity to context will always come first, even if it means that in some cases patterns will be ignored or modified," says Michael McWatters.

Even though TED's interface designs adapt to unique contexts, sometimes breaking from established contexts, the team still considers their system to be effective at generating designs that work for their users. There's a lot of continuity and coherence in their brand and the user experience.

What makes such system work is not strict rules and processes — it's the shared design knowledge, deeply rooted in the culture. The team is fully in sync on their product vision ("Spread the ideas as far and as wide as possible") and their design approach. The design of patterns is guided by the shared principles (such as "Timeless, not cutting edge"), and there's a deep mutual understanding of the purpose of patterns and their usage. This shared knowledge is *the foundation* that makes this system effective, even though it is loosely set up.

## MANAGING DOWNSIDES AND CHOOSING DIRECTION

These were two contrasting examples, but, of course, these parameters are not binary: all teams sit somewhere along this scale. It can seem that strictness is related to company size — younger smaller systems tend to be (and can get away with) being looser, to allow more freedom and experimentation. As a system grows, it becomes stricter. But maybe it's not as simple as that. I once worked in a small team with a brilliant but authoritarian creative director, who closely monitored all design output. It was a small but very strict system. On the other hand, you can imagine a larger company having a loosely set up system, to encourage each team

to experiment and make their own decisions. Perhaps it's not so much to do with the size, but a team's approach and their priorities.

In general, stricter systems provide precise and predictable outcomes and visual consistency. But at the same time, a strict system can become rigid, to the point that you start making UX compromises for the sake of consistency.

To make sure this doesn't happen, there should be opportunities outside the boundaries of the system, such as creative experiments and side projects. People need to understand the rules and be able to challenge them. If there's no understanding, rules will be ignored or overridden. That's why clear, comprehensive and compelling documentation is fundamental to this type of system.

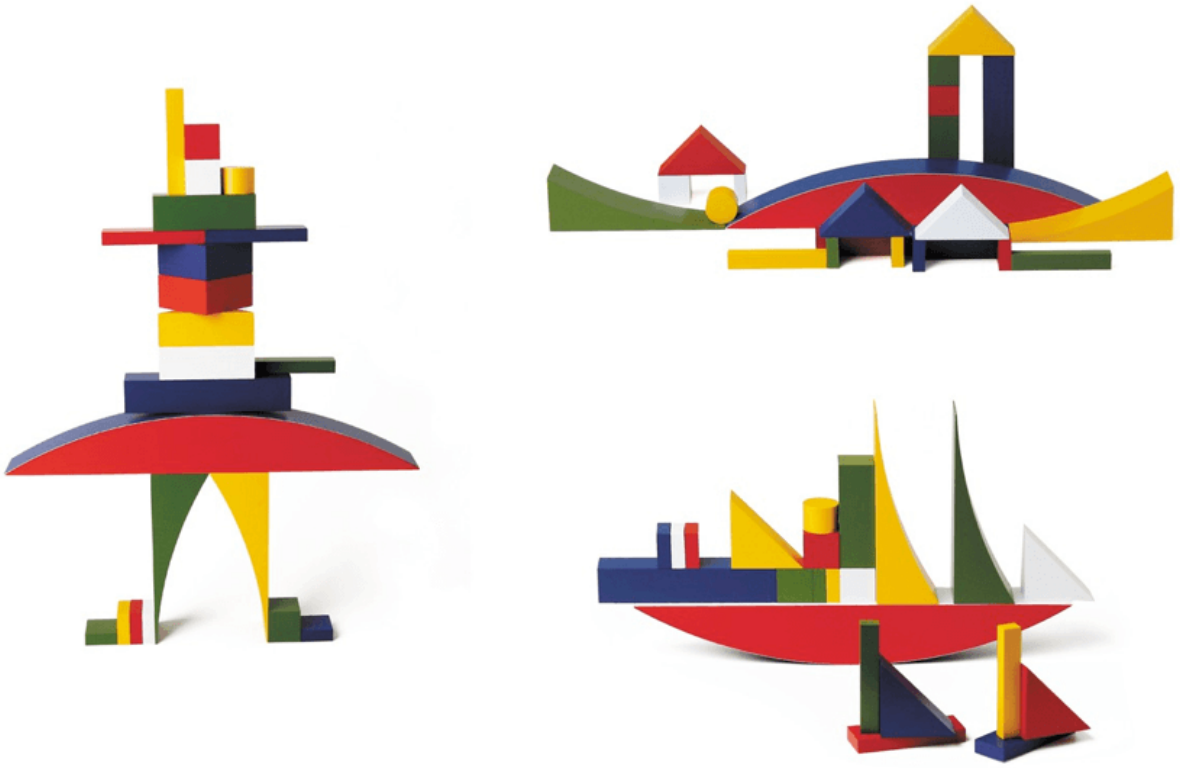
Loosely set up systems are well suited to products that prioritize sensitivity to context and experimentation. But a loose system, which works so well for TED, can quickly become too messy and fragmented in other companies.

To have a simple, flexible system like TED's, everyone on the team needs to be fully aligned on a product's purpose and the design approach, which both need to be ingrained deeply into their culture. Even a loosely set up system needs a solid foundation.

## Parts: Modular Or Integrated

All systems are made of parts. But in the context of design systems, modularity means not just that a system is made of parts, but that

the parts are *interchangeable* and can be assembled together in *various* ways to meet different or changing user goals.



*A modular design, such as the iconic Bauhaus Bauspiel construction set, can adapt to different requirements.*

A modular approach has many known advantages:

- It's agile, because multiple teams can design and build modules in parallel.
- It's cost-effective, because those modules can be reused.
- It's relatively easy to maintain, since you can fix a problem in one module without affecting the others.
- It's adaptable, because modules can be assembled in the ways that meet different user needs.

- It can have a generative quality, which means that you can create entirely new outcomes by introducing new patterns or combining them in new ways.

An opposite of a modular structure is an integrated design approach. An integrated design can also be made of parts, but those parts are not interchangeable because *connections between them* are not designed to fit in different ways.

*Integrated design is optimized towards a specific goal.*

Integrated designs also have a number of benefits:

- They can be specific to a particular content, context, story or art direction.

- They tend to be more coherent and connected (unlike modular structures which can feel disjointed).
- They can be built quicker as one-offs, because there's no need to spend time on making the parts reusable.
- They are easier to coordinate, since everyone on the team works towards one purpose.

Our conversations about design systems on the web have been in favor of modularity and standardization of components. We talk about how patterns should be modular and reusable, how everything should be just like Lego. But the extent of modularity should depend on your team and your product.

## MODULARITY AND THE USER EXPERIENCE

As well as efficiency and cost savings, consider how a modular approach can benefit your users and enhance the experience of your product. In architecture, there are examples where modularity doesn't only add to the experience but becomes its core characteristic.

Puma City is a retail store made of 24 cargo containers that can be dismantled and reassembled. What allows this building to travel around the world is its modularity. It is also key to its design: the containers are put together in a way that shapes the personality of the building. Shifting the containers in different ways can create outdoor spaces and terraces, as well as the interior. Even the way

the logo is fragmented as a result of moving the containers is part of the composition, part of the unique feel of this building.

*Puma City, LOT-EK.*

Now let's look an an example of integrated architecture. The Greendo apartment complex in Takamatsu, Japan is built into the side of a mountain and has five levels; each unit's roof serves as another's garden. Not only is the building embedded in the landscape, it also lives and breathes with the land, taking advantage of natural insulation and heat from the earth to maintain a stable temperature inside.

*Greendo, Keita Nagata.*

Sometimes modularity makes sense at the implementation level but not in the design, or the other way round. This student accommodation in Paris is made to look modular, as though it's constructed of shifting baskets rotated at different angles.

*Basket Apartments, OFIS architects.*

But in fact it's the placement of the balconies and the way they protrude from the building that gives it this feel. In this case it didn't make sense to make the building fully modular, but the *modular aesthetic* still suited that particular building. The opposite could be the case too.

In short, more modular is not always better. The extent of modularity should depend on what you're trying to achieve.

## EXTENT OF MODULARITY AND PROJECT NEEDS

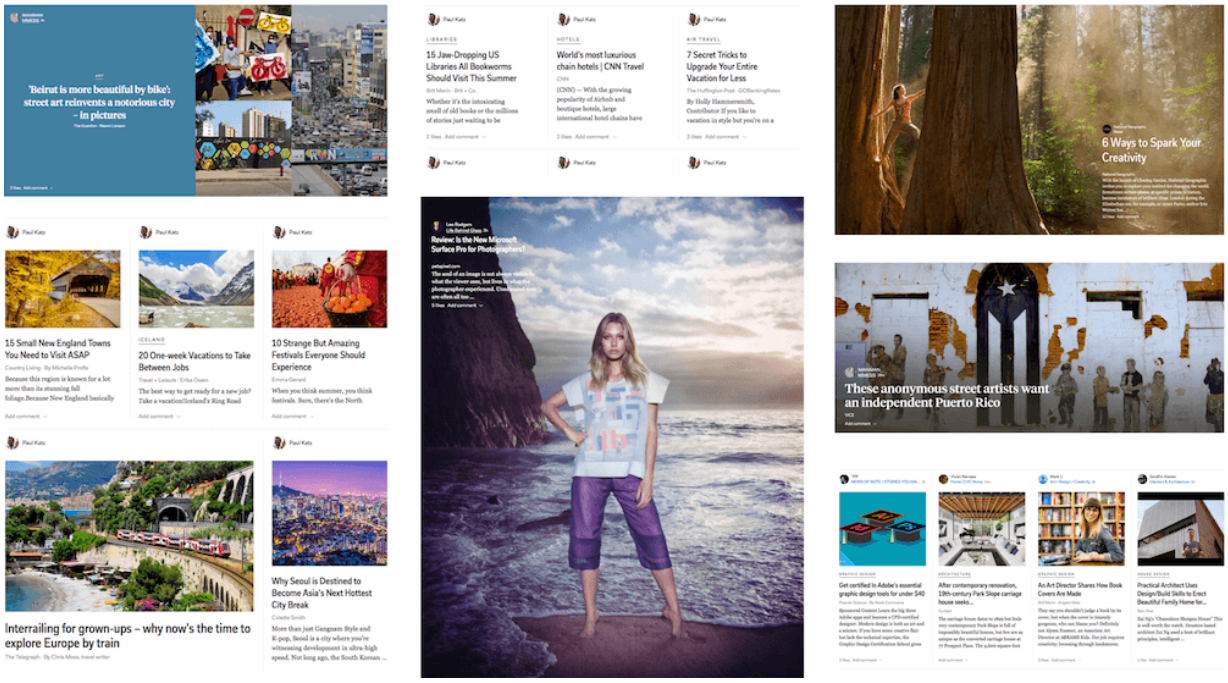
In general, the modular approach is suited for products that:

- need to scale and evolve
- need to adapt to different user needs
- have a large number of repeating parts
- have multiple teams working on them concurrently and independently

Examples can include large-scale sites for e-commerce, news, e-learning, finance, government — anything that needs to scale, evolve and cope with different user needs.

I find it especially interesting when modularity becomes part of the brand and the experience of the product. In Flipboard, for example, modular layouts are at the heart of the design and the brand. This is what helped make the brand distinct: "Each magazine page layout

feels hand-crafted and beautiful — as if editors and designers created it just for you."[1]




*Flipboard's modular layouts are at the core of its user experience.*

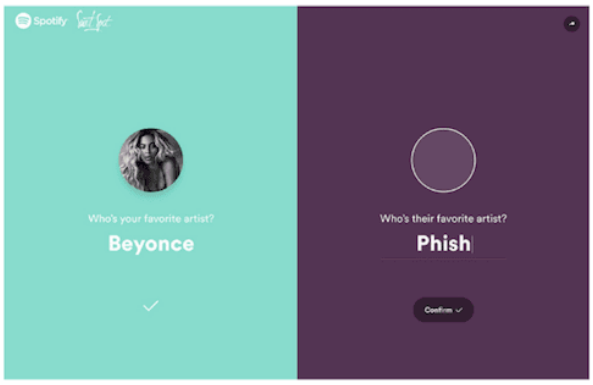
On the other hand, integrated systems are suited for products that:

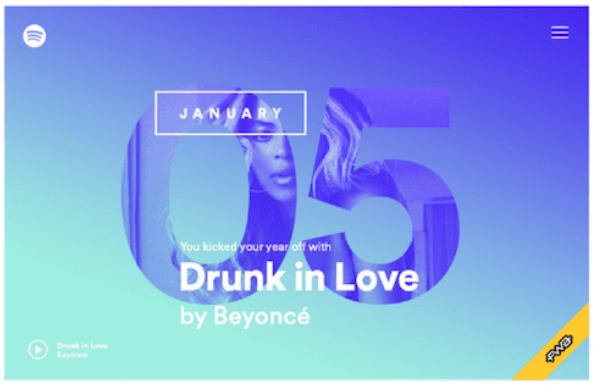
- are designed for one specific purpose
- don't need to scale or change
- require art direction outside the boundaries of the system
- have few shared repeating parts
- are one-offs that are unlikely to be reused

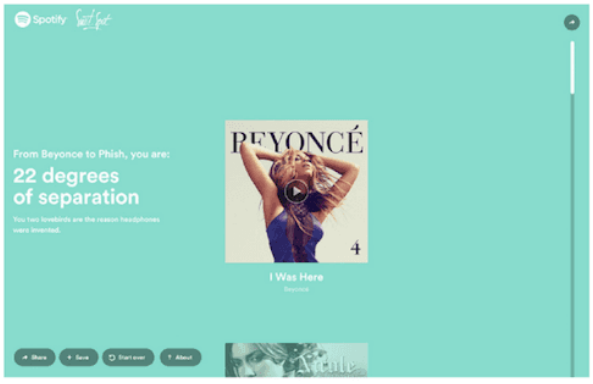
Examples of an integrated approach can include creative conference websites, one-off marketing campaigns, creative portfolios and showcases.

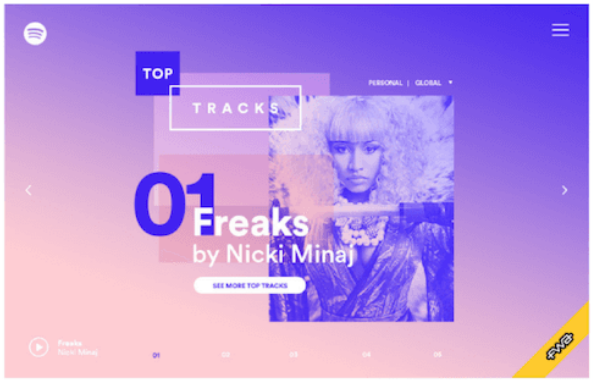

*Circles Conference is a conference for designers and other creatives. Its site features bold design with a number of unique modules that make full modularization not worthwhile. (KIKK.be, 2016)*

Spotify's campaigns feature entirely different designs when promoting specific music events.



*Examples of Spotify's music event campaigns.*

While you can spot some reuse of a few of the brand styles (such as shapes, color and typography), it is less common to see larger building blocks from the main consumer products in these campaigns. To allow more flexibility and creative expression, it made more sense to create these designs outside their modular system for consumer products.[2]

## MANAGING DOWNSIDES AND CHOOSING A DIRECTION

A modular approach is adaptable, scalable and cost-effective in the long run. But modularity comes with some drawbacks.

First, building reusable modules is typically more time-consuming than creating a one-off solution designed to accomplish a specific goal. You have to think through different use cases and plan how they will work across the system. To become cost-effective, modules need to be reused, which can take time. Some teams struggle for a while to see a return on investment in a modular system, which makes it hard to justify the investment in the first place.

Second, modules typically have to be fairly generic to accommodate a variety of cases. The result can be predictable generic designs where power of composition and story are lost in favor of efficiency. When teams choose a modular approach they need to find other ways to innovate — distinct content or service, a strong voice, or effective use of perceptual patterns.

Third, to make modularity worthwhile, teams sometimes force the reuse of the modules. At FutureLearn we ended up occasionally sacrificing the potential impact a page could have for the sake of reusability. To prevent this from happening, technical efficiency should always be balanced with the benefits modularity brings to the user experience.

Finally, one of the main challenges is making modules connect seamlessly. With modular design the expectation is that you can mix and match the parts, and they should fit perfectly together. But sometimes people combine modules in ways that don't work as a whole. And paradoxically, even though there is a lot of consistency across the modules, there is little coherence in the overall experience. To prevent that, we should focus not only on the modules, but also on the *connections* between them: the rules of how they relate to one another, their relative importance (such as visual loudness), their role in the overall user journey, their hierarchy in the overall composition.

Integrated designs can be more specific because they're optimized for one purpose. They also tend to be more coherent and work better as a whole. But they don't scale well. Integrated designs aren't adaptable or reusable, which is often exactly what we need on the web.

The degree of modularization can change over time. Your system can start with a few shared styles and principles. As it grows, you might notice more and more repeating patterns, and increased modularity becomes a logical progression for your product.

Equally, it can develop in the opposite direction. At FutureLearn we have a largely modular system. But we noticed that having to work with reusable patterns on high-impact sales pages can feel limiting. Since FutureLearn started to do more branded marketing campaigns, we work with more custom sections, to make a stronger brand statement.

## Organization: Centralized Or Distributed

Another important aspect of a design system is how it is organized.

### CENTRALIZED MODEL

In a centralized model, rules and patterns are managed primarily by one group of people. Typically this means that:

- they define the patterns and rules
- they have veto power over the system
- they manage the pattern library or other resources where patterns are stored

The most obvious advantage of this structure is ownership. If someone is responsible for it, it is more likely that the system will be curated, maintained and evolved. It also helps the creative direction to be focused and opinionated, because it comes from one source. This is probably why sometimes we find this model in design-led companies, such as Apple or Airbnb.

## DISTRIBUTED MODEL

An alternative approach is a distributed model, where *everyone* who uses the system is also responsible for maintaining and evolving it.

This type of structure provides autonomy for individuals and empowers them to make decisions. It tends to be more agile and resilient — if one team misses something, another one can pick it up. Design knowledge and creative direction are distributed, rather than concentrated in the minds of a few people.[3]

This is the approach used at TED, and it's also the structure we tried to establish at FutureLearn. For a small company like FutureLearn, it was impractical to have a dedicated design systems team. All the designers and front-end developers who work on the product also actively contribute to the system. And because everyone contributes a little, running a system in this manner doesn't take that long. That's the only way we have been able to maintain the pattern library for the last three years.

## ORGANIZATIONAL CHALLENGES

While a distributed approach works for FutureLearn, it's not for everyone. Many companies I spoke to had a different experience when trying it. People were enthusiastic at first but without someone in charge, the work could slow down or stop completely.

The team at Eurostar is not large: it consists of four designers, four product managers and ten front-end developers. A distributed

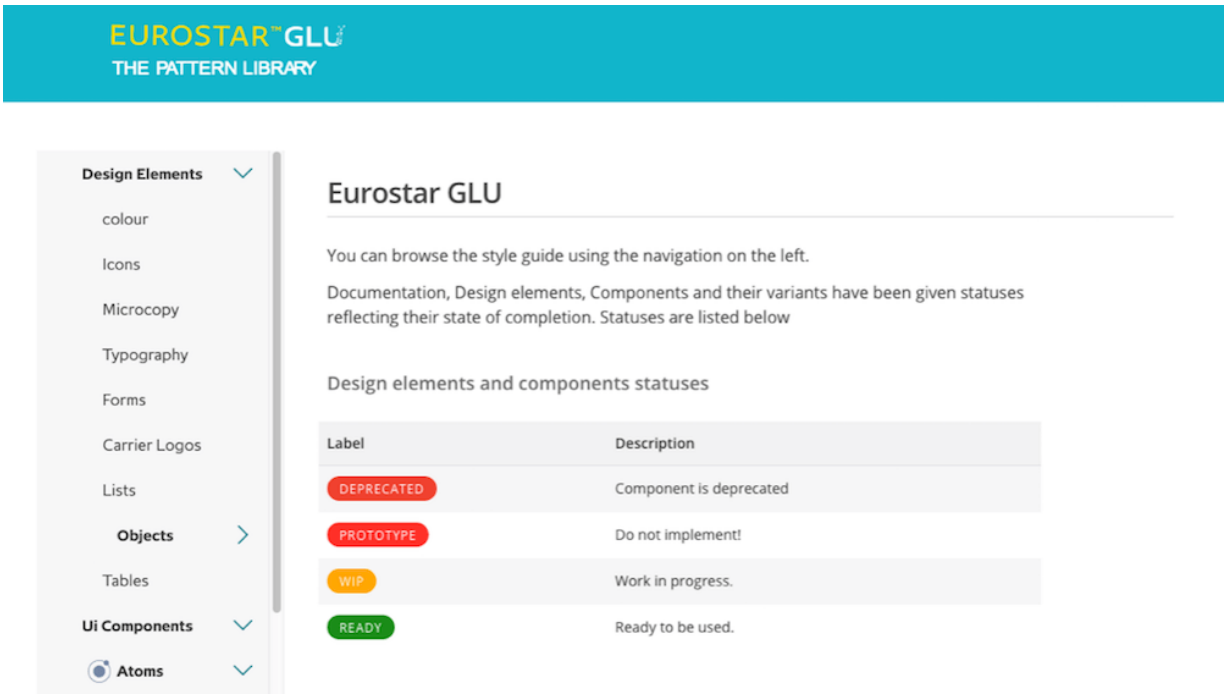
approach initially made practical sense to a team of this size. But after a year, they still struggled to make everyone contribute evenly.

---

*“We wanted to see everyone contributing a little. But instead we saw a couple of people contributing a lot.”*
*— Dan Jackson, solutions architect, Eurostar*

---

After switching to centralized approach, the team made a much better progress.




*Eurostar’s pattern library*[4]*, 2017.*

A fully distributed approach seems to require a certain type of team culture to work. Equally, a strict system also fails without a culture to

back it up. At the BBC, a centralized approach didn't quite work with GEL. Ben Scott, a technical lead at the BBC, shared in an interview how despite investigation into centralizing their system, it was not worthwhile because each product team always had their own strong views on what their design should be. So the distributed approach worked much better.[5]

## MANAGING DOWNSIDES AND CHOOSING DIRECTION

A centralized approach provides ownership and reliability. At the same time, it can work against the team. Because the responsibility falls on one group of people, they can become a bottleneck, slowing down the entire product's life cycle.[6] For smaller teams, like FutureLearn or TED, it may also be impractical taking someone away from the product to allow them to dedicate most of their time to focusing on the system.

A distributed approach promotes more autonomy and empowers the whole team to create. It's more agile, more resilient, because it doesn't depend on a small group of people. But it's challenging to make it work in a way that's sustainable and that doesn't dilute creative direction.

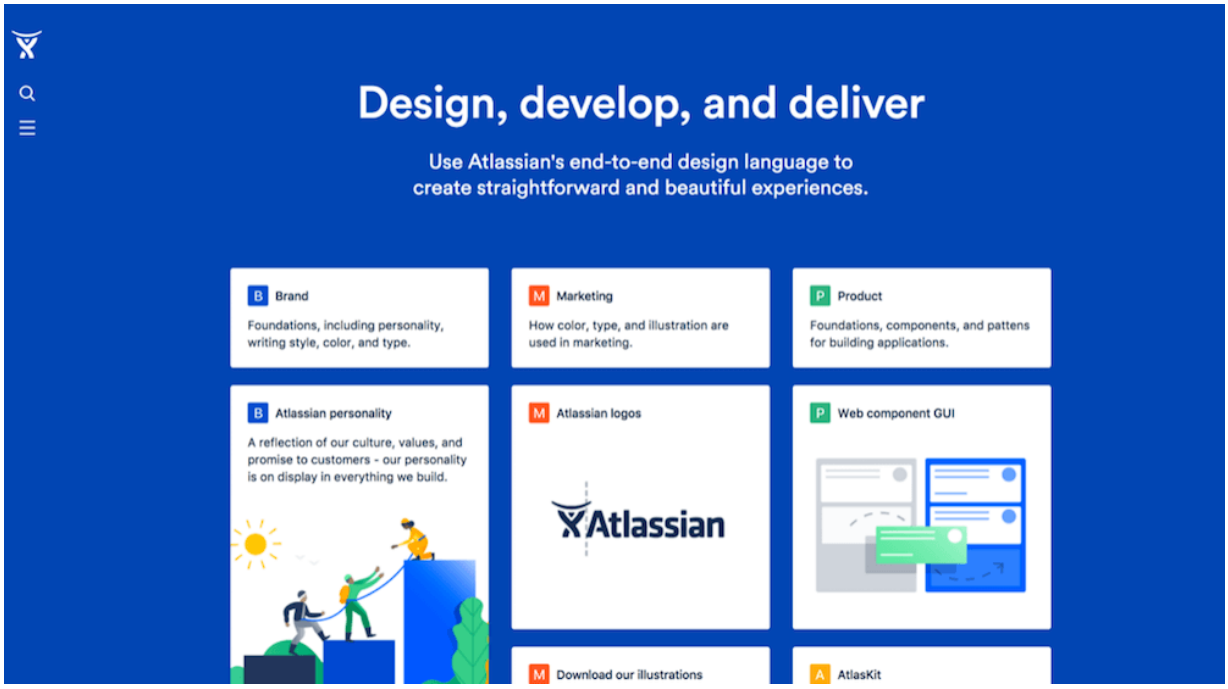
Like the relative strictness of a system's rules and its degree of modularity, ways of organizing a system don't depend on the size of the team. In a small company, creative direction can come from a single source, such as CEO or a creative director. In larger companies, a distributed system can work also.

For example, Atlassian is a relatively large organization with over 2,000 employees. There's a dedicated team that curates the patterns, but there is also an open source model for contributions. Everyone in the company is not only allowed but also actively encouraged to contribute.

---

*"We want people not just to consent to the design language but to embrace it and move it forward."*
*— Jürgen Spangl, head of design, Atlassian*

---



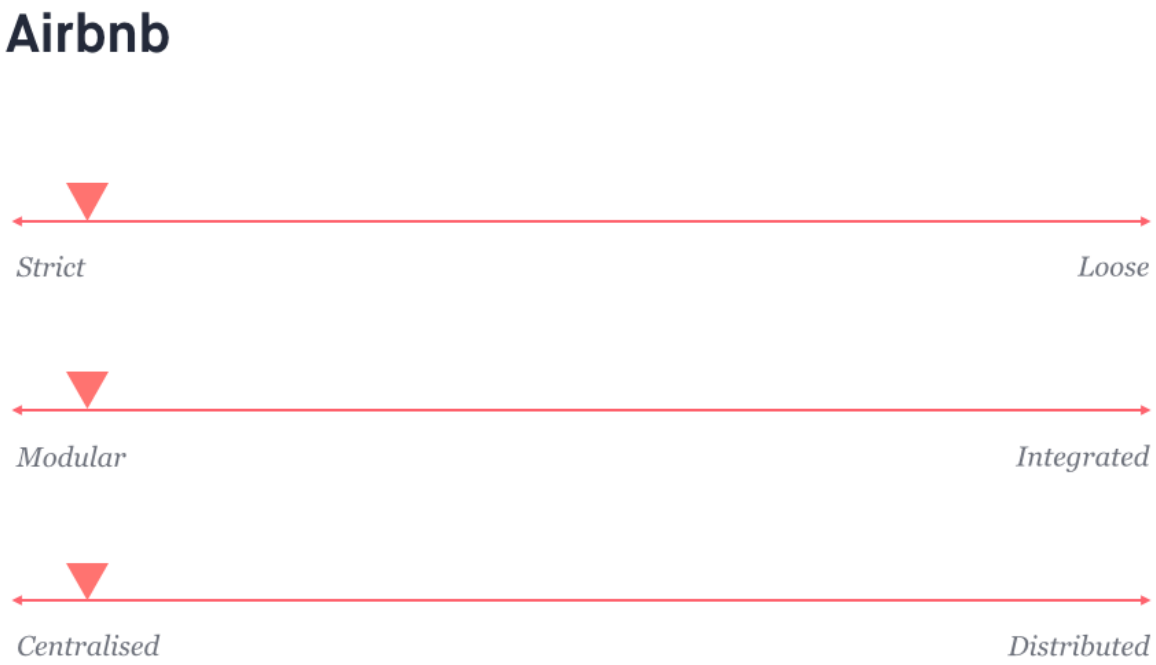

*Atlassian's design guidelines have an open source model where everyone is allowed to contribute, and contributions are managed and curated.*

At the BBC, GEL provides "platonic ideal" versions of patterns, and then each department has its own implementation.

## Summary

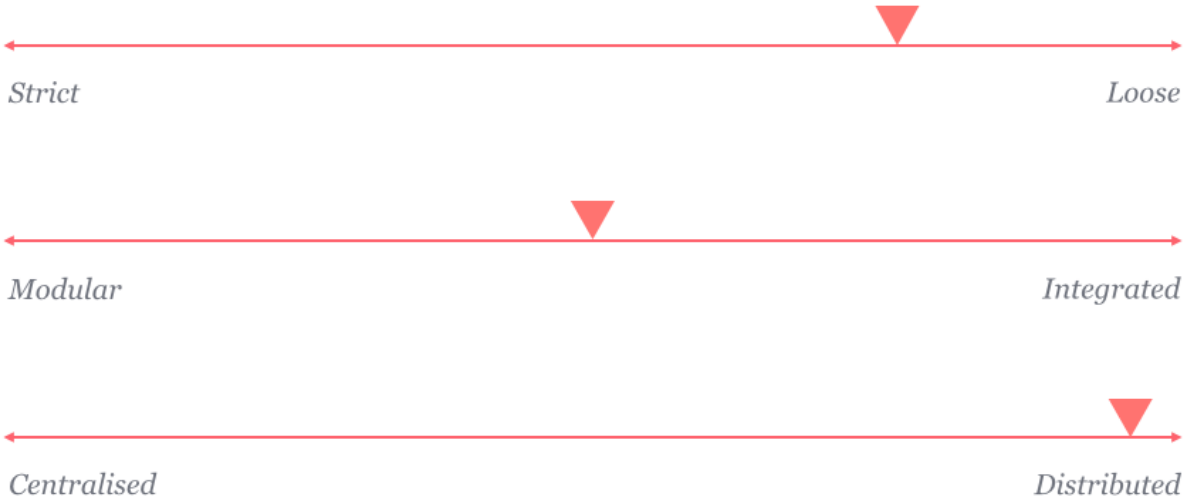
In this chapter we've looked at several teams, with different approaches to design systems. Here's how I would place them on the three scales introduced earlier.

The Airbnb system is strict, modular and centrally organized. It operates through rules and processes which are followed strictly. There's a lot of certainty and consistency with this kind of system.



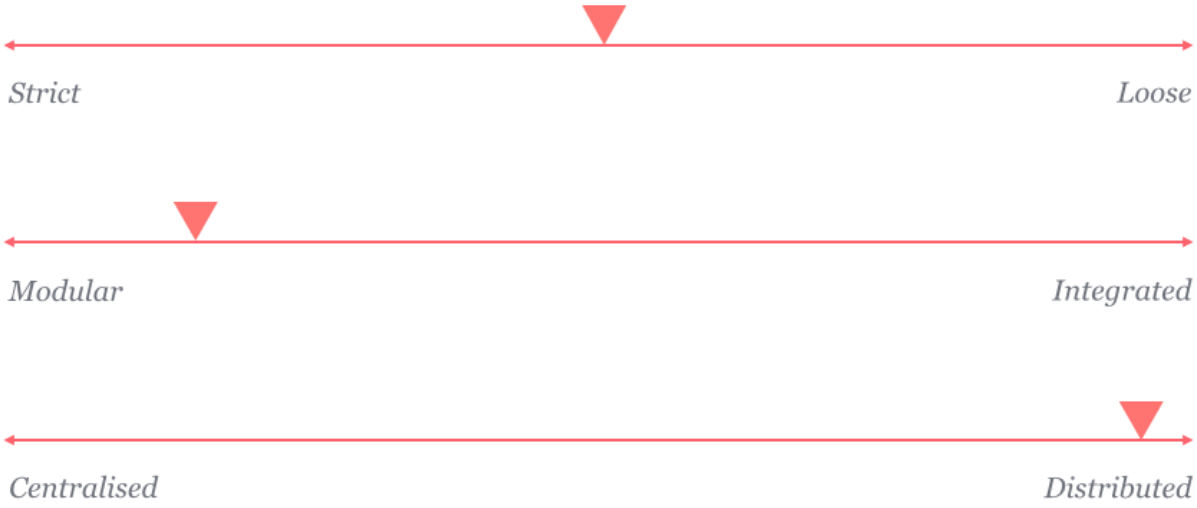

On almost the opposite end of the scales we have systems like TED's, which are much more loosely set up and where creative direction is distributed throughout the team. These systems typically allow more experimentation, fine-tuning and sensitivity to context.

## TED




Somewhere in the middle between the two, we have teams like FutureLearn. As a young and growing company, the design system at FutureLearn has moved along the scales. It started as more centralized and integrated, gradually becoming more distributed and more modular. The rules of the system have also become more strict over time, once we started focusing on consistency.

FutureLearn




## THE RIGHT SYSTEM FOR YOU

Another important aspect is, of course, a *team culture*, which is inevitably reflected in the design system they will produce. As Conway's law states:

---

*"Organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations."*[7]

---

The team at Sipgate are now rebuilding their pattern library. What's interesting is not that they switched their model from distributed to centralized, but that in order to make the centralized model work

they had to first demonstrate to the company what full autonomy would look like. At first, the free-spirited and enthusiastic teams at Sipgate couldn't imagine their designs being "controlled" by someone else. But with an autonomous culture like theirs, they realized that that's exactly what they needed to keep their system coherent and unified.

To make it work, they had to make a few changes in their design process and how they collaborate. The team now put more emphasis on having a shared methodology and the design approach, rather than efficient documentation of the patterns. For them it wasn't only a change in direction — it was also a cultural shift.

The right system for you is not some else's system. Whatever works for one team might not work for another. Sometimes we think other teams have got it right and aspire to build a system just like Airbnb. But every approach has its downsides. The right system for you is the one where you're able to manage the downsides.

At the heart of every effective design system aren't the tools, but the shared design knowledge about what makes good design and UX for *your* particular team and *your* particular product. If that knowledge is clear, everything else will follow.

—

1. http://smashed.by/flipboard
2. The brand and creative team at Spotify, which manages the campaigns, has a simple system that allows for a very flexible combination of brand styles. This is

how they can create a whole range of campaigns which all fit within the Spotify brand.

3. As we discussed in chapter 5, this approach works through close collaboration and shared knowledge. Christopher Alexander refers to a similar concept as a “pattern language” approach in *The Timeless Way of Building*.
4. https://style.eurostar.com/
5. Informal interview with Ben Scott, May 2017
6. At Airbnb the process of adding a new module can take up to two weeks. This is one of the things the team is trying to improve.
7. Conway’s law emerged out of an observation made in 1967 by Melvin Conway, a computer scientist and programmer.

CHAPTER 7

# Planning And Practicalities

*In this chapter we'll look at finding support for establishing a design system in your organization, and planning the work.*

How does a team start thinking about design more systematically? Typically, when they notice issues with their current workflow. Designers become frustrated always solving the same problems, or not being able to implement their designs properly. Developers are tired of custom styling every component and dealing with a messy codebase. Both struggle to meet the deadlines and demands of a growing product. Without a shared design language and practices, collaboration is difficult.

Some people start making improvements — standardizing buttons, setting up a master Sketch file, creating a simple library of existing components. Others notice the benefits and join their efforts. The team might standardize more patterns, align the design vocabulary, set up new tools and processes. After some trial and error, they will see improvements in how their system functions. These first initiatives contain valuable lessons. But to make a real difference, working on a design system as a side project is not enough. You need widespread support — not only from your peers but the senior stakeholders in the business.

## Getting Support From Senior Stakeholders

Getting buy-in is not always easy. Some teams start by collecting examples of the visual inconsistencies across the product. An image of inconsistent buttons might be a compelling graphic but it's not always enough to make a CEO or your manager see the value of the changes you're proposing.

To get support from the business, you need to demonstrate that an effective design system will help *to meet business goals faster and at lower cost*. In other words, you need to make a business case for it.

Sometimes it helps to use familiar language in your pitch. If the idea of a "comprehensive design system" sounds too broad and abstract to your audience, try to present it as "a modular interface" instead. Modularity has many proven practical benefits, which we discussed in the previous chapter. As long as it is the right direction for you, those benefits can be demonstrated in relation to your team and your product. Here are a few examples.

## 1. TIME SAVED ON DESIGNING AND BUILDING MODULES

Naturally, reusing an existing element, rather than building one from scratch, is quicker. You might even be able to work out roughly how much time it can save. At FutureLearn, building a relatively simple custom component for the first time can take about three hours. Building the same component in a modular way (getting the structure right, making it flexible for various use cases, coming up with a good name, adding it to the pattern library) can take twice as

long. But when the same component is used again, it's almost free. In the long run this can save a lot of time, if you work in a way that emphasizes repeatable choices.

Even seemingly simple elements like buttons take time and effort to design and build. To get senior stakeholders to understand the value of reusing components, design system consultant and author of *Modular Web Design* Nathan Curtis uses a story about button complexity. In a few slides he demonstrates how buttons can cost hundreds of thousands of dollars to design and build.

---

*"If your enterprise has 25 teams each making buttons, then it costs your enterprise $1,000,000 to have good buttons."*[1]
*— Nathan Curtis*

---

Quantifying and demonstrating the cost of inefficiency is the often the most effective way to get executive buy-in.

## 2. TIME SAVED ON MAKING SITE-WIDE CHANGES

A bloated and inefficient system means that even the smallest changes are time-consuming and fiddly to make. In his article "Designed for Growth,"[2] Etsy's Marco Suarez described how technical and design debt slow their team down. He shared an example of a diff of Jessica Harllee's work for updating the styling of buttons on etsy.com. The deleted code is shown in red, and the

code written in green. Evidently, far too much code was touched in order to make a simple visual change.

*A diff file for updating the styling of buttons on etsy.com where the code deleted is shown in red and the code written in green.*

Not only are changes time-consuming to make, sometimes the same change has to be applied in different places. Conversely, a reusable pattern will be updated automatically everywhere that pattern is used. This makes site-wide updates quicker to make.

In the long run, modules should get better the more they are used. Different teams come up with different use cases and solutions to meet them. By improving individual components, the whole system becomes more robust and easier to maintain. And the less time a team spends fixing bugs and untangling messy code, the more time they have to work on things that bring value to their users and the business.

## 3. FASTER PRODUCT LAUNCH

If you visit a patisserie and order a selection of ready-made pastries you will get an entirely different quote from ordering a custom-made

cake. At FutureLearn, product managers understand the different timescales for developing a feature made from existing modules versus new ones. Building a page using the patterns from the library is typically a matter of days; a new design can take weeks. So if we need to test a concept or pilot a new feature, sometimes existing modules are used to release a feature quickly. It might not be perfect, but it gives the team time to test, collect feedback, and work out a better solution.

Tobias Ritterbach noted how the teams at Sipgate that use their new pattern library are many times faster when delivering new features than the ones that don't:

---

*"Having a pattern library for sipgate.de allows us to build pages 10–20 times faster than for other product sites which are not connected to the library."*
*— Tobias Ritterbach, experience owner, Sipgate*

---

These examples show that a modular system helps to meet the demands of a growing product by enabling teams to prototype and ship features faster.

## OTHER BENEFITS

When making a case for a design system, the winning arguments tend to focus on *demonstrating and quantifying the cost of*

*inefficiency*. But, of course, there are other important benefits, which may be valued in some organizations.

## Brand Unity at Scale

It's not uncommon for companies to end up in a situation where their different products (or sometimes even parts of the same product) all look like they belong to different brands. Unifying a growing product or line of products under the same brand requires an effective design system.

## Visual Consistency

Design is a form of language — through design we communicate a mental model of the product. A consistent visual representation helps people learn the interface quicker and reduce cognitive load by making things familiar and predictable. It helps to make an interface feel *intuitive*.

Creating consistency is like making small promises throughout the interface (when you see a pink button it is always an action; the "Cancel" button always comes before "Submit"). When people can be confident of what will happen, they can rely on the product. Consistency helps to build trust.

## Teamwork and Collaboration

A shared language is fundamental to collaboration, and that's exactly what an effective design system provides. It gives people the tools and processes to create things together. They can build on one another's work, rather than recreate the same things from

scratch. Even after simply collecting components in one place, designers at Airbnb began to see improvements in productivity.

---

*“We collected components in a master Sketch file. After a week or two we began to see huge leaps in productivity by using the library when iterating on designs.”*[3]
*— Karri Saarinen, Design Lead, Airbnb*

---

Presenting these cases in ways specific to their situations has helped teams get the support they needed. Using a test project to demonstrate the benefits can also be effective, as designer Laura Elizabeth advised in one of her talks.[4]

---

*“Trying a design system on a small test project allows you to see the effect it can have on your work and to demonstrate how much time you’re saving.”*

---

Not everyone can establish a dedicated design system team or formalize their plans right away, but even spending one sprint with a couple of dedicated people could be a helpful starting point to demonstrate what could be achieved. The success of these first steps can be used to get more support later.

# Where To Start

As we saw in the previous chapter, every team will have different requirements, and different strategies will work for them. But here are a few tips that have been helpful for most people, regardless of their situation.

## AGREE ON YOUR GOALS AND OBJECTIVES

What are the main outcomes you're hoping to accomplish with this work? Having clear goals gives a team direction and motivation. It helps them organize their time and balance priorities. A goal should include a few objectives — specific measurable steps to achieve it.

If you, like me, see design systems in relation to *patterns* and shared *practices*, your goals can reflect that. For instance, one of them can be focused on the interface, and the other on the way team operates.

### 1. Systematize the Interface(s)

- Define guiding design principles.
- Define and standardize reusable design patterns.
- Establish a pattern library.
- Set up master design files with up-to-date patterns.
- Refactor code and front-end architecture to support the modular approach.

## 2. Establish Shared Processes and Governance

- Set up knowledge-sharing processes (through conversations, collaboration, pairing, training, and so on).
- Promote the pattern library and encourage its use across the company.
- Promote shared design language across teams and disciplines.
- Make introduction to the design system part of the induction process.

Objectives can be broken down into smaller tasks and stories. To plan how your goals will be achieved over time, at this stage it is also helpful to create a simple roadmap for your design system. This can be done in software such as Trello, or even on a whiteboard with sticky notes. What matters is that the roadmap gives the team a collective understanding of their priorities and how the system should evolve over time.[5] At FutureLearn we also integrated design system stories into the main product's roadmap. This helped to make the rest of the team aware of the work we were doing and balance it with other priorities.

Having clear goals and milestones also helps manage expectations in the rest of the company. A design system is a long-term investment — its value increases gradually over time. It's important that people expect to see gradual and steady improvements rather than quick dramatic ones.

## MAKE YOUR PROGRESS TRANSPARENT

In my observations, teams who have made public some of their work on the design systems tend to progress faster than those who kept everything under wraps. Dan Jackson at Eurostar explained in an interview the importance of making their style guide public early on, even when it wasn't perfect. Knowing that others might have been learning from their work provided additional motivation:

---

*"I wanted the style guide to be a public-facing product that we're proud of. Other people may be looking at it and using it as a resource. It makes us feel that we have to keep up."*
*— Dan Jackson, solutions architect, Eurostar*

---

Some teams write short blog posts about their system as it evolves. This is especially useful if you describe not only your successes, but also your mistakes, stumbling blocks and things you'd do differently next time. Documenting your progress in an open and honest way is a powerful way to help your team learn and stay motivated.

Doing that also makes transparent the work done behind the scenes. Writing and talking regularly about our design system at FutureLearn was invaluable in getting more people to understand the value of the work we were doing.

## CREATE A CULTURE OF KNOWLEDGE SHARING

As we saw in the example of Sipgate in the previous chapter, a team can have an up-to-date pattern library, but it won't provide as much as value without effective cross-team collaboration. Getting your team to think in a more systematic way requires powerful knowledge-sharing practices. We discussed some of them in chapter 5, "Shared Language":

- Set up a dedicated Slack channel to collaborate on defining and naming design patterns and to discuss system related questions.
- Create a pattern wall to make the process open and transparent to the rest of the company and encourage more people to join in.
- Make introduction to the design system part of the induction process.
- Organize regular catch-ups to keep everyone on the same page as your system evolves.
- Encourage collaboration not only within individual teams *but across teams and disciplines*. In particular, encourage people who are more knowledgeable about the design system to work with everyone, so they have an opportunity to share their knowledge and enthusiasm with people who are less immersed in the system.
- Organize workshops and tutorials to introduce the team to the changes as the system evolves. At FutureLearn, the most effective presentations had a "problem—solution" format. First, we talked people through current problems then explained how

the changes we were proposing would help to solve the problem. For example: “Current typography means that text is too small on large screens and too large on small screens; reading experience is affected; it’s not clear which headings to use, and there are too many styles, which creates inconsistencies, and so on. Here’s how the new typography system solves these problems.”

- One of the guerilla tactics Vitaly Friedman and his team have been applying is dedicating each day to a component in the interface. They’d have a carousel day, a lightbox day, an accordion day, and so on. Everybody would receive a print-out highlighting the component and its variants, including front-end code and related styles.

---

*“We put it next to the kitchen sink and in the bathroom. A month later, everybody remembers the naming of all the components, including the cleaning personnel!”*
*— Vitaly Friedman, editor-in-chief, Smashing Magazine*

---

## KEEP UP THE TEAM’S MORALE

Working on a design system is a long-term process. Your team might not see the rewards of what you’re doing for some time.

*“You don’t always get the personal satisfaction right away — the reward comes when you see other people using the module you created in their work, or when someone comments on how helpful the information was for them.”*
*— Jusna Begum, front-end developer, FutureLearn*

---

There are a few things you can do to help keep up the team’s morale during the process.

Rather than chipping away at an endless list of tasks, aim to *complete the bulk of the work in one go*, and then continue the rest as part of ongoing work. At Atlassian, initial progress was made through design spikes by two or three people. Matt Bond, a product designer who led the initial work on the Atlassian Design Guidelines (ADG), explained in one of his blog posts that having a two-phase approach allowed the team to get through the initial stages quickly and to then maintain momentum:

---

*“It was high output, getting many new patterns to 80% completion in a short amount of time. We’d then spend the next week or so dedicating small amounts of time to refine a pattern and get the guidelines and code up to scratch to include in the ADG.”*[6]

---

For some of the work, such as conducting an interface audit or setting up a pattern library, it's useful to get the whole team (or representatives from multiple teams) involved, at least in the initial stages. Doing so provides a shared sense of ownership. If it's not possible owing to other priorities, let a smaller group do the groundwork and involve others as needed. At FutureLearn, two of us (a designer and a front-end developer) spent a sprint fully focused on on the system, roping others in as required once we'd figured out how it should work.

It also helps to plan the tasks in a way that affords *the most benefit for the least effort*. At FutureLearn, our goal was to make all the components living. This meant that the code for the modules on the website, and in the pattern library, would need to be the same. But achieving that involved refactoring every module. As we refactored them, we added them to the pattern library, one by one. It was a painfully slow process and the team started to lose motivation.

We then realized that we could provide value quicker by adding all the patterns in one go and displaying them as screenshots instead of code. This allowed the team to start using the pattern library for reference right away. In the following months we gradually replaced the screenshots with living modules, as we continued refactoring them. Had we not done that, it would probably have taken another year before all the patterns were documented.

## Practice Thinking In Systems

One of our first experiments with modularity at FutureLearn was an attempt to redesign the home page. A visual designer created modular slices, and we then held a workshop where we tried to organize them into full comps. That's what we thought (perhaps naively) a modular design process looked like.


*One of our first experiments with modularity.*

We ended up with three designs that eventually became fully functioning prototypes. But even though it was a useful exercise, the result wasn't truly modular:

- Modules didn't have a clear purpose. The differences between them were mostly presentational.
- We didn't define and name them.
- We didn't put a lot of thought into how they would be reused.

- Their role in the overall system was unclear.

The prototypes never made it into production. But it's these type of experiments that helped make our design process more systematic. By trying different things we realized that modular design is much more than cutting up the interface and putting the pieces back together. If your team is new to this way thinking, it's useful to first explore what modular means by experimenting on a side project or on a small area of your product first.

After trying out a few different approaches, we arrived at a more structured team exercise for systemizing an interface. The next two chapters describe this exercise in detail. In a nutshell, it will follow three steps:

1. Identify key behaviors or aesthetic qualities
2. Audit existing elements
3. Define patterns

The steps are slightly different for functional and perceptual patterns. With functional patterns, the focus will be on the *user behaviors*, on defining individual modules and naming them. We will look at perceptual patterns more as a whole, focusing on the *feel and aesthetics*, and on the general principles of how they work together. The order you do it is not critical. Some teams find it helpful to look first at the foundational styles, such as typography; others start with core functional modules. It is also possible to look at both simultaneously in parallel.

In both cases, we consider the big picture first, and then deconstruct the interface into smaller parts. Approaching it this way helps us think not only of individual modules but also how they work *together*, and how they help to achieve the *purpose of the product*.

—

1. From “And You Thought Buttons Were Easy?” by Nathan Curtis.
2. http://smashed.by/designforgrowth
3. See “Creating the Airbnb Design System” by Karri Saarinen.
4. See “Selling Design Systems” by Laura Elizabeth.
5. For practical advice on creating design system roadmaps, see “Roadmaps for Design Systems” by Nathan Curtis.
6. See “How we made the Atlassian Design Guidelines” by Matt Bond.

CHAPTER 8

# Systemizing Functional Patterns

*The exercise in this chapter describes an approach to systemizing functional patterns, starting with a product's purpose.*

In the town where I live there's a small bookstore. As you walk in, you see a few shelves of book covers. Some have small handwritten notes attached to them: reviews from the people who read them. Even if you don't know what you'd like to read, there's a good chance you'll stumble upon something intriguing. Once you do, there's a quiet area with sofas to look through the books over coffee. You might decide to buy something or you might not, there's no pressure. The ethos of the store is discovery and reading; sales appear secondary. Its patterns — the notes, quiet areas, sofas and coffee table — reflect that.

Similarly, digital products encourage or enable certain behaviors. Consider how Slack supports ways of working which are more collaborative compared with email or other chat apps. Or think how Tinder promotes casual, commitment-free relationships with its swiping interaction. Products can be designed around similar user goals and needs, while encouraging entirely different behaviors. That's why thinking of behaviors can be helpful when connecting patterns with the design intent and ethos of the product.[1]

Design intent can be rendered in countless ways — patterns don't have to be visual. They can be represented in physical objects (like the interior of a book store), or they can be read out by a voice.

Articulating the behaviors helps to define patterns in a way that is more futureproof, because behaviors are platform-neutral.

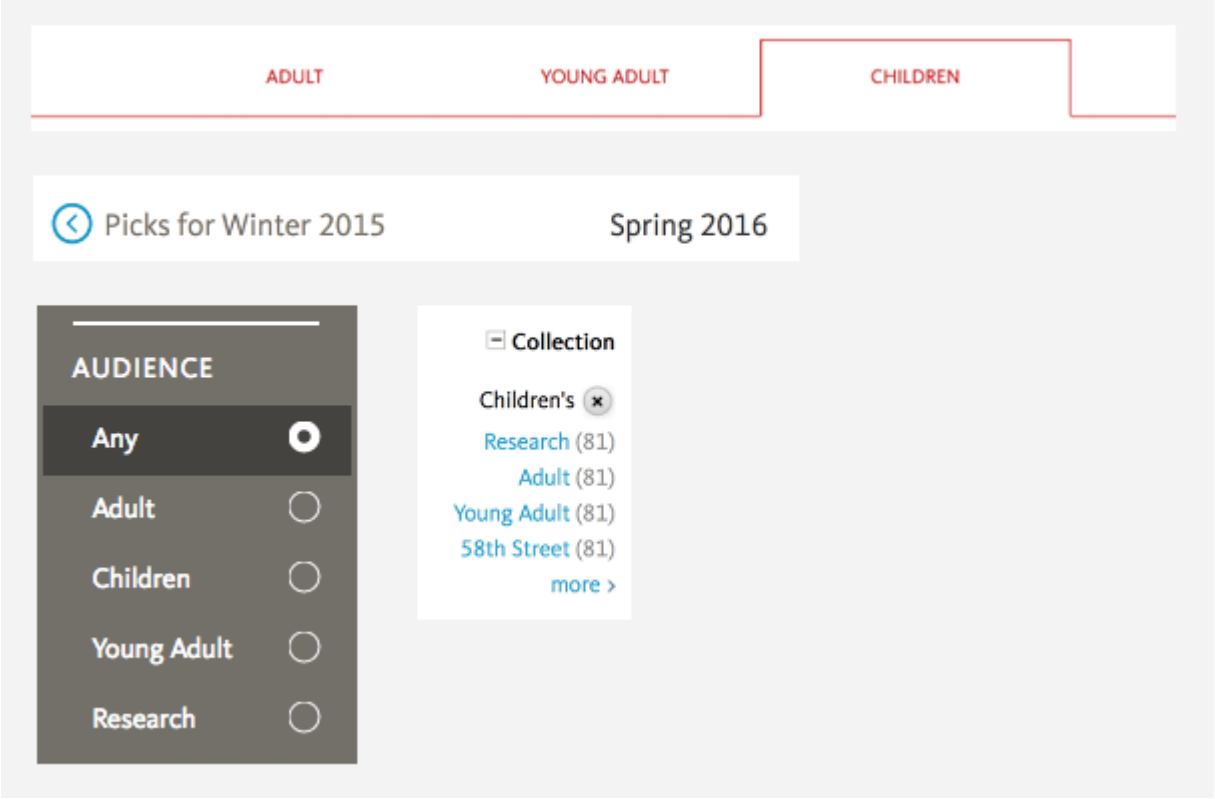
## A Purpose-Directed Inventory

The interface inventory[2] is a popular exercise to start systemizing an interface. It involves taking screenshots of various UI elements, and then grouping similar-looking things together.

But while the idea is straightforward, it can be done in a variety of ways. Sometimes inventories focus on the visual consistency of the interface; for example, making sure all buttons look the same, all menus are consistent, and so on.

The main goal of the process described in this chapter is not to account for all the visual inconsistencies; it's to *define the most essential design patterns* and get mutual understanding in the team on how they should work across the system. Going through this process will give your team an idea of which areas need more attention. A typical outcome would be a list of elements that need to be standardized, along with some sketches and ideas of how patterns should be defined.

While a visual inventory typically groups things by appearance and type (buttons, tab controls, and so on), in the following exercise you might end up with things in the same group that look different, because you're grouping them by *purpose* (*the behaviors* they're designed to encourage or enable).




*In a purpose-directed inventory, things in the same category might look different because they're grouped by purpose rather than visually.*

This means rather than focusing on making all buttons look consistent, we will first try to understand when to use a certain type of button, when to use a link instead of a button, when not to use a button at all and instead click directly on the object. Of course, in the process of doing that we will improve visual consistency, but it won't be the focus.

## PREPARATION

### Timing

To be most effective, this process should be run after the foundational UX work — user research, content strategy,

information architecture, design direction — has been worked out. If the design has fundamental flaws and usability issues, they would be distracting and counterproductive to deal with. For similar reasons, if your interface is about to go through a major redesign, it's best to get clarity on the new design direction first.

## People

Having different perspectives can help you to be more objective and account for more use cases. It's important that designers and front-end developers take part, but ideally involve a back-end developer, someone with a content background, and a product manager. The ideal group size is around 4–8 people. If a larger group needs to be involved, consider running the initial exercise with a few representatives from different disciplines, and then hold follow-up sessions to debrief more people.

## Interface Printouts

Identify the key screens and user flows that are *absolutely fundamental* for your product, those without which the product couldn't exist. Typically, about 10–12 screens is enough, sometimes fewer. They can be design mock-ups, or screenshots of an existing interface.

Let's say you're working on a public library website. The purpose of the website is to extend the experience of the physical library, perhaps by making sure that readers can reserve books in advance and so avoid queueing up and waiting for materials once they are at the library. The key screens help to achieve that purpose by allowing

you to find specific books, discover new books, reserve materials for collection, and download the materials. Of course, there's a lot more to a library website: events and exhibitions, memberships, online collections. While we should take note of the other areas, usually every single view is not needed to get started.

Print out *two copies* of each screen. Put the first set on the wall, in the order of a typical user journey. The second set will be used for cutting out patterns and grouping them. You will be constantly shifting from the system as whole to individual patterns throughout the exercise. Having two sets of printouts will help you focus on both the details and the bigger picture, without losing the context of where the patterns come from once you've cut them out.

You will also need scissors, markers, sticky notes, and plenty of wall and desk space to work on.

## 1. Identify Key Behaviors

Start by identifying the key user needs and behaviors you want to support at each segment of the user journey. For a small app with only a few screens, you'll be looking at the individual screens or different states on the same screen. For larger products it helps to group pages into segments of the user journey.

To return to the public library website example, you might group some of your pages based on these behaviors:

- **Discovery.** Encourage people to discover books they might be interested in. To draw an analogy with a bookstore, this area is like the staff picks or new books showcase shelves. If someone doesn't know what they're looking for, they might be inspired by the selection on display.
- **Catalog.** Find specific books. Searching through a catalog is like approaching a member of staff and making a request.
- **Wish list.** Allow people to view and manage their shortlisted books. In a physical store, you would put some of the books aside so you can decide later which ones to keep.[3]


*Some of the core screens grouped by the behaviors they support throughout the user journey.*[4]

Take note of the pages with conflicting behaviors: situations where we encourage people to look at new books, download something, sign up for a newsletter, and check the latest events all at the same

time. Even if a screen supports several behaviors, the most important actions should be clear and not in conflict with one another. When dealing with multiple behaviors, focus on the core user journeys and most important behaviors first. In this example: discovering, finding and reserving books.

## WORDING IS FUNDAMENTAL

The words we choose matter. They influence how we think. For a few months the team I worked in at FutureLearn had “retention” as our metric. It focused on getting more people to continue learning on a course after it began. Designing for retention was hard. It also wasn’t clear how exactly retention benefited our users. Had the metric been called “engagement,” it might have led to different design outcomes. And perhaps even more, had the metric been centered on quality and satisfaction of learning, rather than the time spent on the site. (Someone could have spent half an hour at FutureLearn and learned what they needed, but it wouldn’t have counted as success).

Behaviors should be meaningful and work from the user’s perspective, as well as the business’s.[5] “Promotion” of the books benefits only the library, but “Discovering” new books also has value for the reader. This makes it a better language choice and can influence the selection of the books being displayed, as well as how they’re displayed.

## BREAK DOWN BEHAVIORS INTO ACTIONS

After you define the high-level behaviors, break them down into more specific *actions* that feed into those behaviors. Write them down next to each screen. For instance, the actions that support “Book discovery” are:

- Scan for any inspiring or interesting books.
- Refine list of recommended books.
- Control how list is presented.
- View and learn about a book.
- Make a selection of books you might like.
- Shortlist and reserve books.

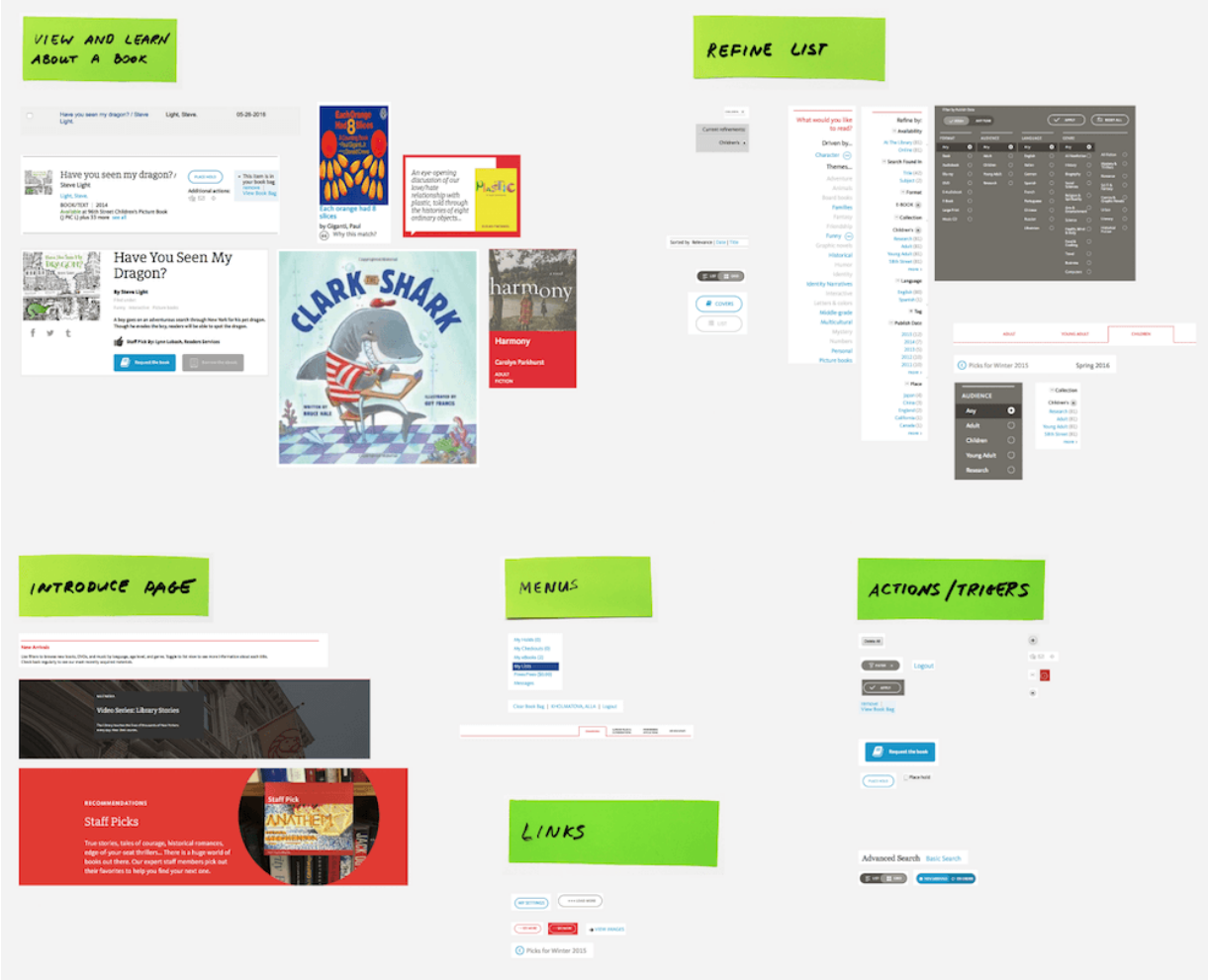
*Actions feed into higher-level behaviors.*

You might notice that some of the actions are repeated throughout the interface. But the elements that represent them aren't always the same. On some occasions we refine a list of books by clicking through tabs; on others, by selecting an item from a menu. To reveal these inconsistencies, we can audit the existing elements.

## 2. Group Existing Elements By Purpose

Taking one behavior at a time, look *across all the pages* to find the elements that support it. For example, to "View a book" we might be using different items on the promo pages, in the catalog search results, and in the wish list.

Cut the related elements out using the second set of printouts. Arrange them into groups and label each group: "View a book," "Refine a list," and so on. They are the *candidates* to be defined as patterns. The elements should be grouped at the *same level of granularity*, so you won't have a "Book list" module and a "Reserve" button in the same group.




*Groups of items: candidates to define as patterns.*

## 3. Define Patterns

Now that you have groups of elements, decide how to deal with the items in each group. Should they be merged into one pattern or kept separate? Typically, this is worked out on a case-by-case basis. But there are two techniques I find particularly helpful: placing a pattern on a specificity scale, and mapping out its content structure.

## SPECIFICITY SCALE

The same pattern can be defined as more specific or more generic. Say we need to display events and exhibitions on the library site. If we define them as two separate patterns we can make each one more specific. On the other hand, unifying them into something like a “content block” would make the pattern more generic.


*Specificity scale.*

While it seems like a simple concept, deciding the level of specificity is one of the trickiest things about modular design. The more specific something is, the less reusable it is. And conversely, to make something more reusable, you also need to make it more generic. With more specific parts, the system becomes harder to maintain and to keep consistent. But too many generic modules lead to generic designs. Like with many things, there’s no right way to define patterns and it all depends on what we’re trying to achieve.

Do we want visitors of the site to perceive exhibitions differently to events? Is there anything about events that might be in conflict with the design of exhibitions? If so, we should consider splitting them up. For example:

- The design of an exhibition module can be centered around an image of the art. Since exhibitions are unique, they can feature custom titles that complement the art to give it a poster-like feel. The date could be set in smaller type and positioned in the corner, so that it doesn't distract from the poster.
- Events are simpler. We could center design around a prominent date and an icon of the event.

If there's no reason to differentiate between the two types, we should unify them into one pattern: things to do in the library. Doing that will make the pattern more generic because it would have to work for both cases. But it would also mean that every change we make to events will apply to exhibitions. Consistency will be easier to achieve but at the expense of flexibility.

## CONTENT STRUCTURE

Another tool that I find helpful is mapping out a pattern's content structure. We covered it briefly in chapter 4 on functional patterns. Here's a reminder of what it involves:

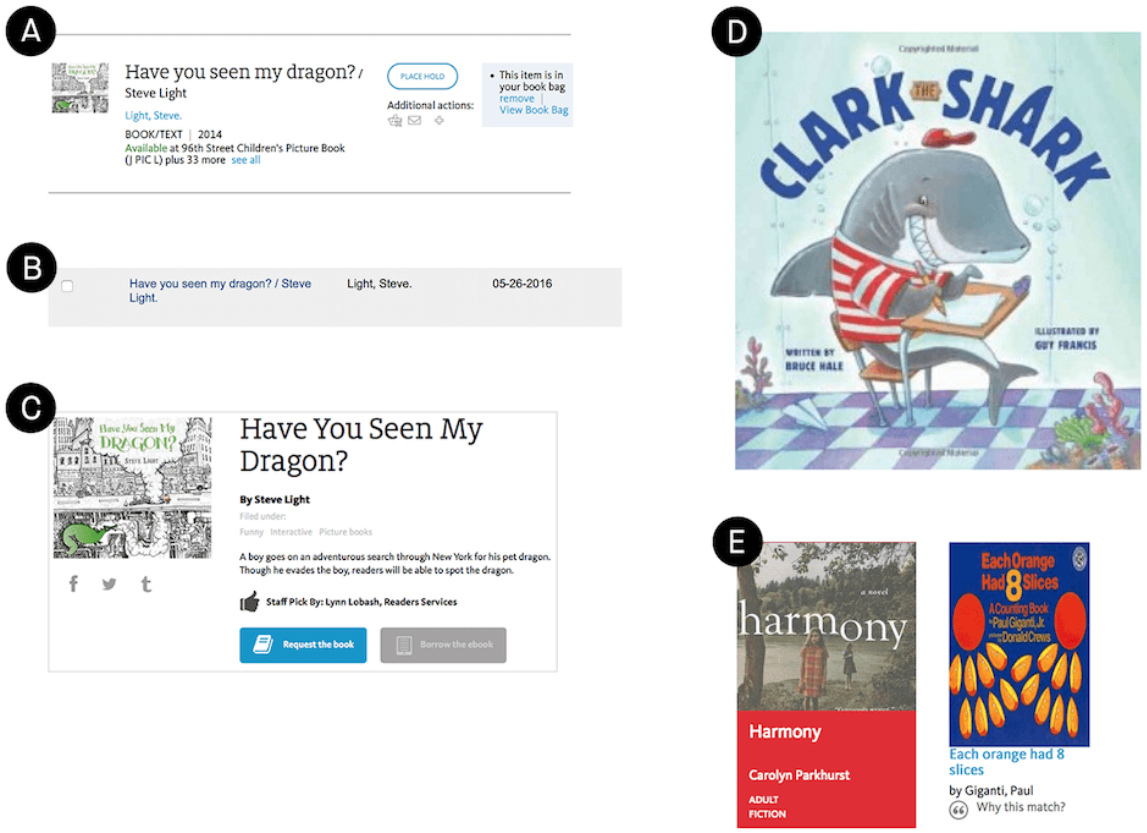
1. List the core content slots a module needs to be effective. Can this module function without the image or is the image essential to its purpose? Is a label always necessary? Mark optional elements.
2. Determine the hierarchy of elements and decide how they should be grouped: is the icon part of the key info or is it part of the image?

3. Make sketches to visualize the structure. The same pattern can be presented in countless ways and sketching helps find the optimal design.

Typically, elements that can be merged into one pattern share the same underlying structure. On the other hand, if you struggle to unify the structures of multiple elements without compromising their purpose, it's an indication that they shouldn't be merged.

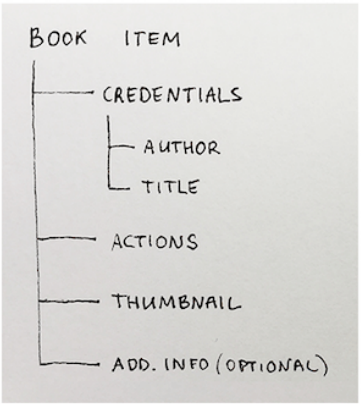
Sometimes elements have a similar structure, but owing to context or our design intent, they need to look or behave differently. In this case we can create *variants*. A variant is a modified version of the same pattern.

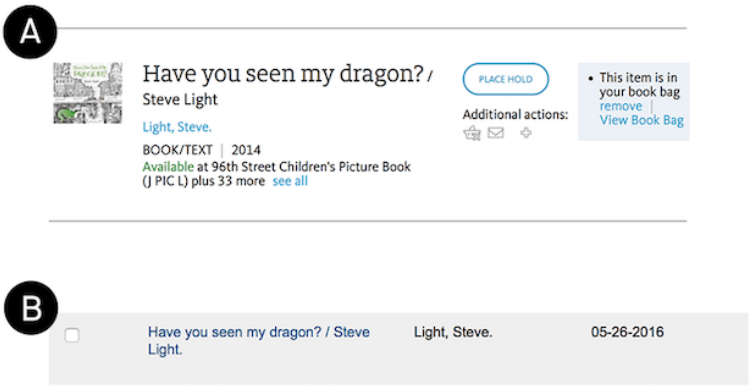
Take the library website again. Say you've ended up with these items in your "View book" group.




*Grouped book items: candidates to define as patterns.*

You might decide that items A and B share the same purpose: they both appear in lists and allow people to view a book and learn about it. They also share the same content structure:



*Content structure for book item.*

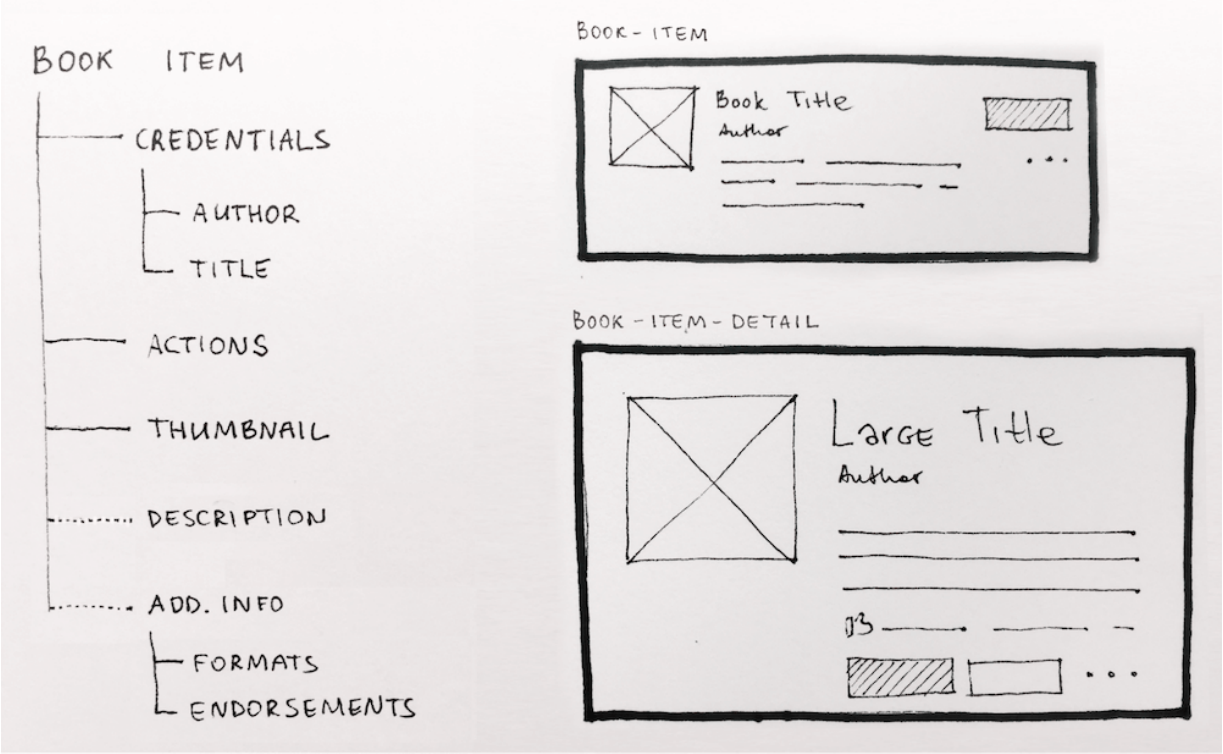
Although *actions* and *thumbnail* are missing from item B, there's no obvious reason for that. Thumbnails are useful for scanning the books and you should be able to reserve one without having to leave your wish list.

On the other hand, items D and E are different. Their main purpose is focused on providing inspiration and *showcasing* new and noteworthy items. If we draw their structure, it might look like this:


*Content structure for book showcase.*

You can check this by thinking about how you expect changes to happen. Ask yourself: if I change this module, *do I want the others to change in the same way*? For instance, even though "cover" and "thumbnail" look similar we might decide to treat them as entirely different things. Perhaps the design of the discovery pages involves some specific interactions and animation to draw attention to the showcased books. We don't want the same effect to apply to a standard book item in a list.

Now let's take a look at item C. It is similar to A and B and shares their content structure. But it is more prominent because of its context: the discovery and showcase parts of the site. It is also more detailed and provides more information than a book item in a list. In this situation it would make sense to make this element a *variant* of the book item.




With variants, you would normally have a *default* pattern with the core styles. Variants would have additional styles. It's important to know which features are core to the pattern, and which are specific to the variants. Then you can predict how a change in one of them will affect the others.

In the example above, some of the elements in the core default pattern vary in scale, making the pattern feel quite different:

- large title
- large thumbnail
- more spacious layout

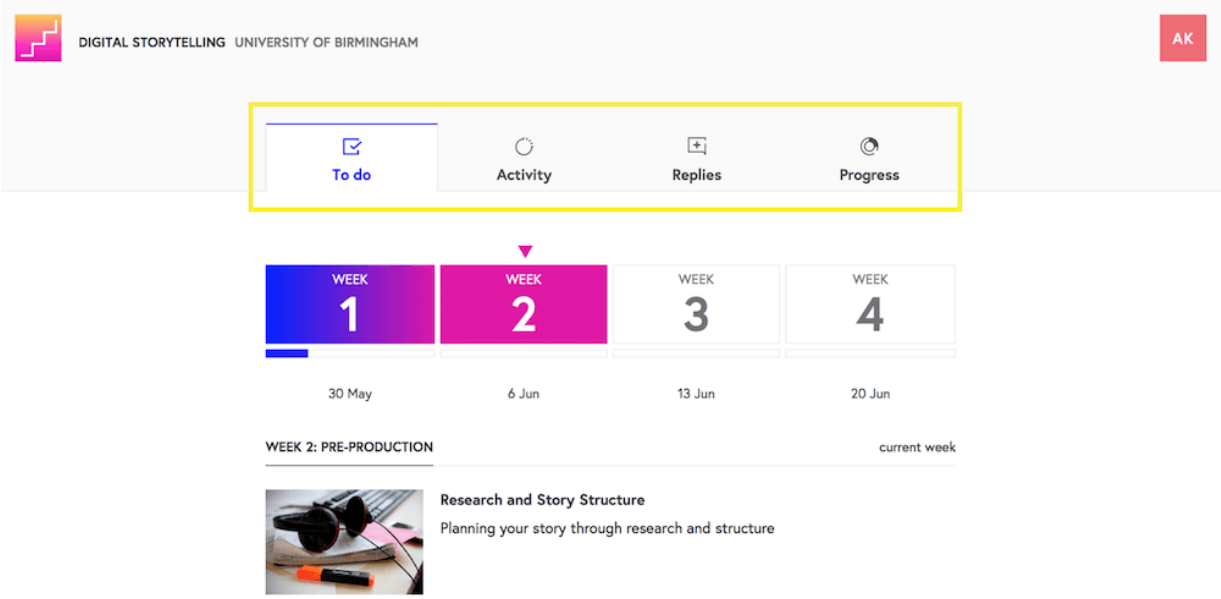
We know that we can change the title without affecting the book item, but if we change the author style, for instance, the change will apply in both places.

Looking at the relationship between the content structure and styles can increase the reuse of more patterns. Try going through all your patterns and match the underlying content (“book title”) with style names (“large title,” “small title,” “small metadata”). Similarly, this would be a good place to start looking at character counts or image sizing variations. A pattern will work with more content types if different sizes are standardized.

## NAMING

As we discussed in chapter 5 on shared language, names affect how patterns are used. A well-chosen name is a powerful tool for shaping your design system.

Try to find a name that reflects a pattern’s position on the specificity scale. If in doubt, start with a more specific name. For example, we used to see the process of learning on a course on FutureLearn as a special kind of experience, which came with its own set of modules, specific to the course area.




*Course tabs on the course overview area on FutureLearn.*

In this case it made sense to use the name “Course tabs” — we didn’t want to reuse them anywhere else. This name also signalled to the rest of the team that they weren’t just any generic tabs — they were specific to the course area. Later we decided that this module could also be useful in other places and changed the name to “Page tabs.” The new name was more generic, and again signaled to the team that it was now available to be used in other areas.

Sometimes modules are named in the front-end but naming is also a UX decision and should be made collaboratively at the design stage. Names need to take the content type into consideration but shouldn’t be based solely on the content. Effective names guide usage and reduce the chances of duplicate patterns.

## Repeat The Process At A Smaller Scale

Once you group the self-contained parts, repeat the process with other elements. Typically, this would involve several sessions: one to discuss big-picture user behaviors, and separate ones to look at more granular patterns such as:

- buttons and links
- headings
- lists
- tabs and menus
- radio buttons, toggle buttons and checkboxes
- feedback messages
- navigation
- images
- icons

If you have elements with similar purposes, think of them in relation to each other rather than independently. How are buttons different from links? How is tabbed navigation different to a list menu? How is a dropdown different from a set of buttons? How is a checkbox different from a toggle button?[6]

Here are some points to consider when auditing *links* and *buttons*.

## CONSISTENCY OF ACTIONS

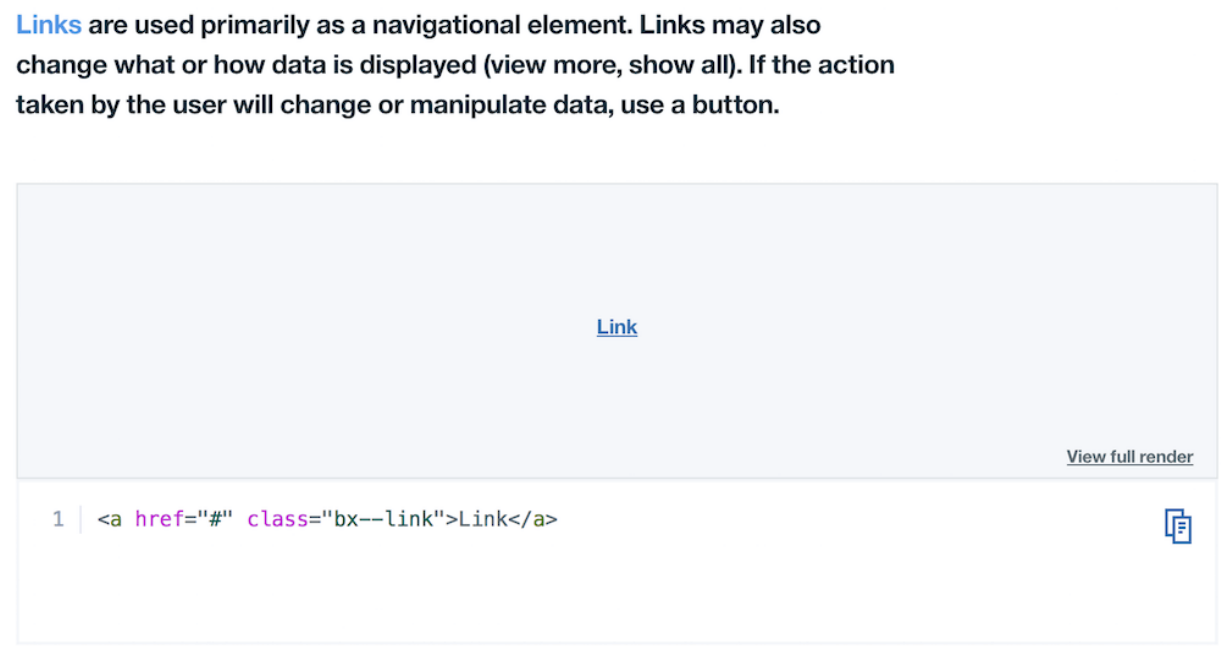
### Buttons and Links

Traditionally in web development, links and buttons are different. A *link* navigates the user away from the current page. A button submits an action and toggles something in the interface.[7] But in practice, it's not easy to make design decisions based on this criterion alone.

Suppose we have a book item with a "View book" button. Clicking the button expands the module, revealing more information about the book. Now imagine that the same information opens on a different page instead. Does this mean that the action should be presented as a link?

As with many other things, the confusion often lies in the language. Some people (developers, often) define a button as a trigger that submits data. So a link marked up as a button wouldn't be considered a true button by them. Others (often designers) view a button as a distinct, standalone call to action. They would refer to a standalone element "View book" as a button, even if it's marked up as link.

Different systems also approach this differently. In IBM's Carbon[8], links are a navigational element. Buttons are only used if the user's action will change or manipulate data. In Shopify Polaris[9], on the other hand, buttons can represent any type of action, including navigation. Links are used both for embedded actions and for navigation.



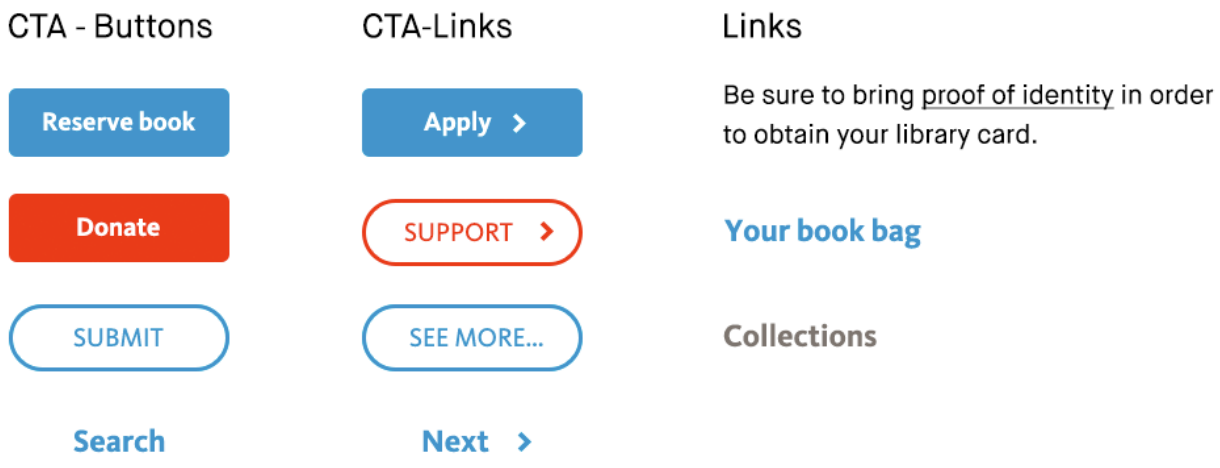

*Use of links in IBM's Carbon.*

To me, the most important aspect is a *consistent expression of purpose*. Users (both those who access the interface visually and via screen readers) need to know what to expect. If buttons are always used only for submitting data, then it would be confusing to have one situation where they behave as links. But if there's a consistent use of links styled as buttons (such as for standalone calls to action) throughout the interface, then it would be appropriate.

To avoid confusion and misuse of these essential elements, it's important to agree on their definitions. What are the shared meanings of "button" and "link" in your team? What are the basic guidelines for their usage?

One of the simplest and most effective distinctions I've come across was suggested by Heydon Pickering in *Inclusive Design Patterns*[10].

The idea is to differentiate between links and calls to action (CTAs), rather than buttons and links. An important standalone action can be presented as a button, but be marked up either as a link or a button, depending on the interaction. The question of whether it's a link or button is a matter of variants — first and foremost it's a CTA.




*An example of a classification of buttons and links. Additionally, it helps to make a subtle difference in the style of CTAs, to indicate a difference in the interaction.*

If the action occurs on the current page, use a CTA button. If the action takes the user away from the current context, use a CTA link. Calls to action are different from standard links, which represent pathways to optional information and are typically embedded in the content: body text, titles, images.

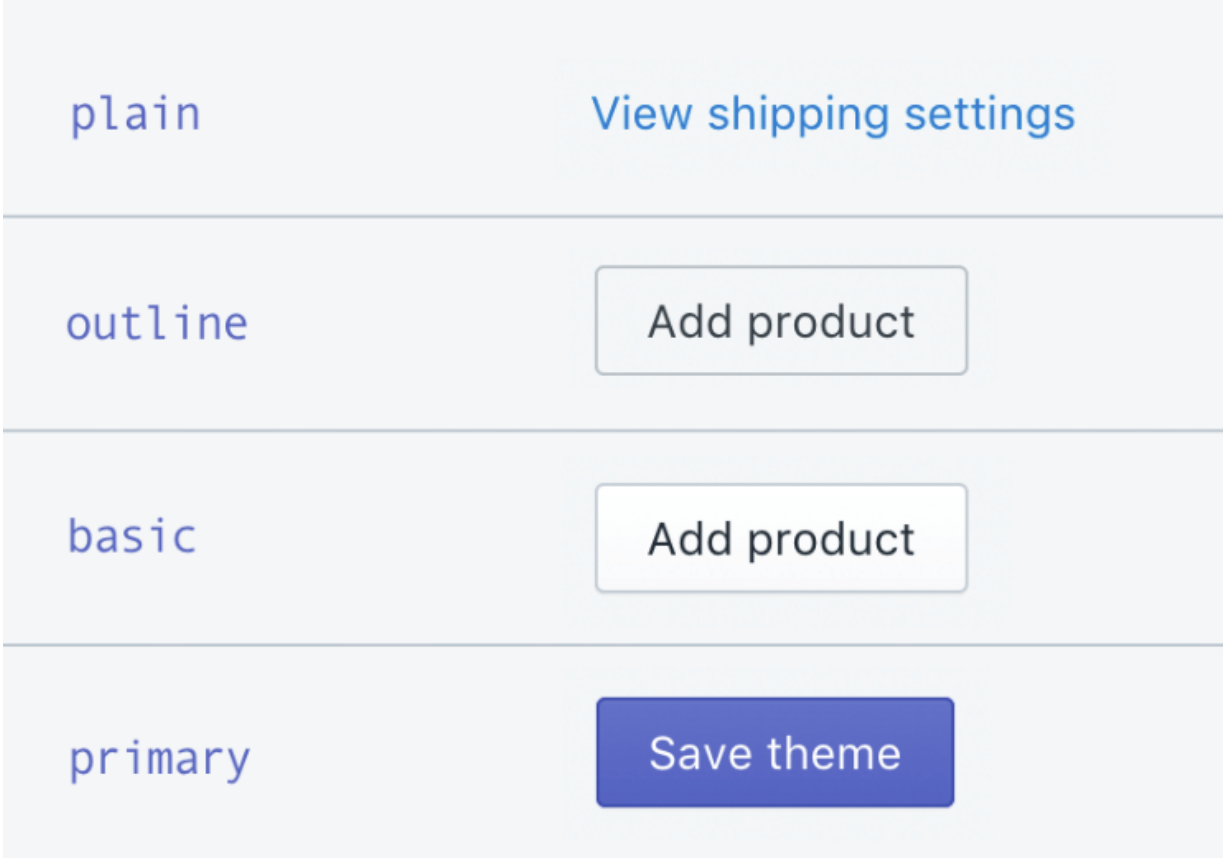
Making a distinction in this manner allows you to meet the design need for keeping the important calls to action prominent, while at the same time keeping the code simple and accessible.

## VISUAL HIERARCHY

Most interfaces have equivalents of primary and secondary buttons. But what does “primary” mean exactly? Does it signify the most important action in the context of the whole interface, or a specific screen or section? For example, should a “Reserve book” button always be a certain style because of the importance of the action on a library website?

In Marvel’s design system[11], “flat” buttons are used to signify “necessary or mandatory actions”; “ghost” buttons are used to signify “optional, infrequent or subtle actions.” Flat buttons can be used alongside each other, when actions are equally important. I like this distinction because it’s simple, clear, and specific to the button’s purpose.

But for more complex interfaces with a larger number of buttons, it’s hard to keep their functions so specific. You may also need to see how buttons relate to each other when used together. In Atlassian’s system[12] and Shopify’s Polaris[13], primary buttons represent “the most important actions in any experience” and therefore should only appear once per screen.




*Some of the button types in Polaris, arranged by the level of prominence.*

They have a “basic” button, used by default. Other styles are used only “if a button requires more or less importance.” Think of it this way: if the interface were read out by a voice, which action would be read out first? Which actions would be announced more loudly or with a different intonation?

## SPECIAL CASES

There will always be special cases. In the library website example, the “Reserve” button could be treated differently. It could include states specific to the action; for instance, its label could change to “Cancel reservation” if the book hasn’t been collected yet.

FutureLearn's "Progress toggle" button can also be seen as a special case. It is only used on learning steps, to indicate if a step is complete. A bouncy animation and a tick icon popping up are designed to give it a celebratory feel. It's not meant for anything else.


*Progress toggle button.*

Perhaps this specificity is why we struggled so much to name it — we tried to come up with a generic name ("Progress toggle") when in fact it could have had a name specific to its function — even "Mark complete" could have been a more appropriate and memorable name in this case.

Both the "Progress toggle" button and the library "Reserve" button are things we might want to make more memorable. They are key functions of the brand, and perhaps opportunities for signature moments. Equally special cases like this should be occasional, both so they appear distinct but also so the general pattern rules are maintained.

## Summary

In this chapter we looked at systemizing a small section of an interface. After following this process in your team, you'll have a better knowledge of your system and the areas that need attention.

For the next steps, teams can dive into code and Sketch to work on finalizing designs for the patterns — making sure it works for all required use cases, defining states and behaviors, refactoring code.

The first time you do this exercise you might be overwhelmed by the number of elements and patterns. You don't have to do it all in one go. Start with core patterns, fundamental to the experience, then move to another area. Most importantly, this exercise needs to be done regularly, as your system evolves. It's a bit like gardening — the longer you leave it, the harder it is to get it into a good shape.

Now, let's look at perceptual patterns.

—

1. Equally, design patterns can be used to create behaviors that hijack user attention or manipulate people to spend time and money on something they'll regret later (https://darkpatterns.org/). Remaining conscious of behaviors can help make sure that user interests are always at the heart of the design.
2. http://smashed.by/interfaceinventory
3. As an aside, if you have too many pages in the same segments supporting similar behaviors, it's an indication that your information architecture might need work.
4. The example pages are taken from the New York Public Library website, for illustrative purposes.
5. Successful businesses join user goals with business goals. If you really struggle to join the two together, your product might have deeper issues a design system won't be able to solve.

6. There are general guidelines and best practices (for an excellent resource see *Designing Web Interfaces* by Bill Scott and Theresa Neil; and *Designing Interfaces: Patterns for Effective Interaction Design* by Jenifer Tidwell), but some things might be specific to your situation. Even if they're common knowledge for some people, it is worth verbalizing them so the rest of the team can learn.

7. See "Links vs. Buttons in Modern Web Applications" by Marcy Sutton and "Proper Use of Buttons and Links" by Dennis Lembrée.

8. http://smashed.by/ibmcarbon

9. http://smashed.by/polaris

10. http://smashed.by/inclusivedesignpatterns

11. http://smashed.by/marvel

12. http://smashed.by/atlassian

13. http://smashed.by/polaris

CHAPTER 9

# Systemizing Perceptual Patterns

*The exercise in this chapter describes how to define perceptual patterns and integrate them into the system.*

Something grabbed my attention recently in the two products I was using — the design of accordion controls. In both interfaces the accordions looked similar and had identical (standard) functionality: expanding and collapsing sections of content. Both would be considered “aesthetically pleasing” by most people. But somehow one of them didn’t feel as robust as the other. The hover state was too subtle, the transitions were a little slow, the selected state didn’t stand out, and there didn’t seem to be enough contrast between headings and body copy.

The design of the other accordion seemed to get all the details just right. Not only that, the same patterns — color, transitions, contrast, typography — were applied throughout the interface which helped to make it feel sturdy and well built. Even though the two products had similar utility, one of them gave a perception of quality and reliability, while the other came across flimsy and fragile.

Sometimes we think that if beauty is not what we’re after, we don’t have to place any importance on aesthetics: “It’s just a functional tool. It doesn’t have to feel like anything in particular.” But then we miss a key opportunity to influence how a product is perceived. What’s important is, of course, not the styles themselves but the

*effect* they have. A functional tool should be useful and usable, but it should also *feel* simple, safe and robust.

To influence perception in a reliable and scalable way, we need to be aware of the patterns that create it.

## Looking Beyond Style Properties

The most obvious way to think of color, typography, spacing and other styles, is in terms of their properties: color values, text sizes, measurement units.

Take color, for instance. In many pattern libraries it's represented by a set of values.


*Pivotal's*[1] *color variables demonstrate a common approach to represent color information.*

But even with a standardized color palette there's still plenty of scope for interpretation. What do the values represent? Which variation of green should you use? How do the colors work together?

Here's a counterintuitive thought, for a design systems enthusiast: slight diversions in color aren't problematic. In fact, having twenty shades of blue isn't an issue, if blue has a consistent *meaning* throughout the interface. But if blue represents links in some parts of the site, and in others there are blue headings users can't click, it can cause usability concerns. A set of shared colors is not enough — you also need a shared *use of color* in the context of the product.

Likewise, a well-defined typographic scale alone won't make your typography more cohesive. Even after we standardized all the text sizes on FutureLearn and introduced a unified type scale, the headings still weren't always used consistently — designers and developers weren't sure which size to pick from the scale. A shared use of typography required clear guidelines and patterns of usage which everyone could understand.

| | viewport: < 679<br>content: 576 | | viewport: 680 - 1694<br>content: 648.0 | | viewport: 1695+<br>content: 728.96 | |
|---|---|---|---|---|---|---|
| pico | 11.24 | 16.05 | 12.64 | 19.78 | 14.22 | 22.00 |
| nano | 12.64 | 17.80 | 14.22 | 22.00 | 16.00 | 24.50 |
| micro | 14.22 | 19.78 | 16.00 | 24.50 | 18.00 | 27.31 |
| milli | 16.00 | 22.00 | 18.00 | 27.31 | 20.25 | 30.48 |
| uno | 18.00 | 24.50 | 20.25 | 30.48 | 22.78 | 34.04 |
| kilo | 20.25 | 27.31 | 22.78 | 34.04 | 25.63 | 38.04 |
| mega | 22.78 | 30.48 | 25.63 | 38.04 | 28.83 | 42.55 |
| giga | 25.63 | 34.04 | 28.83 | 42.55 | 32.44 | 47.61 |
| tera | 28.83 | 38.04 | 32.44 | 47.61 | 36.49 | 53.32 |
| peta | 32.44 | 42.55 | 36.49 | 53.32 | 41.05 | 59.73 |
| exa | 36.49 | 47.61 | 41.05 | 59.73 | 46.18 | 66.95 |
| zetta | 41.05 | 53.32 | 46.18 | 66.95 | 51.96 | 75.06 |
| yotta | 46.18 | 59.73 | 51.96 | 75.06 | 58.45 | 84.20 |
| bronto | 51.96 | 66.95 | 58.45 | 84.20 | 65.76 | 94.47 |

*The typographic scale on FutureLearn provided a foundation for all typography.*

How do we define the styles in a way that communicates their purpose and encourages their consistent use? As before, we'll start at a higher level, and then go down into the details. With functional patterns we looked first at user behaviors. With perceptual patterns, we will start with aesthetic qualities.

## Aesthetic Qualities And Signature Patterns

Every interface has a certain feel, even if it's only represented through text or voice. The most effective styles aren't applied on the surface but evolve with the product — they will link to its ethos and core design principles ("Timeless, not cutting edge," "Direction over choice"). Think how those qualities are embodied. What makes your product feel timeless, or utilitarian, or traditional, or cutting edge, or warm, or robust?

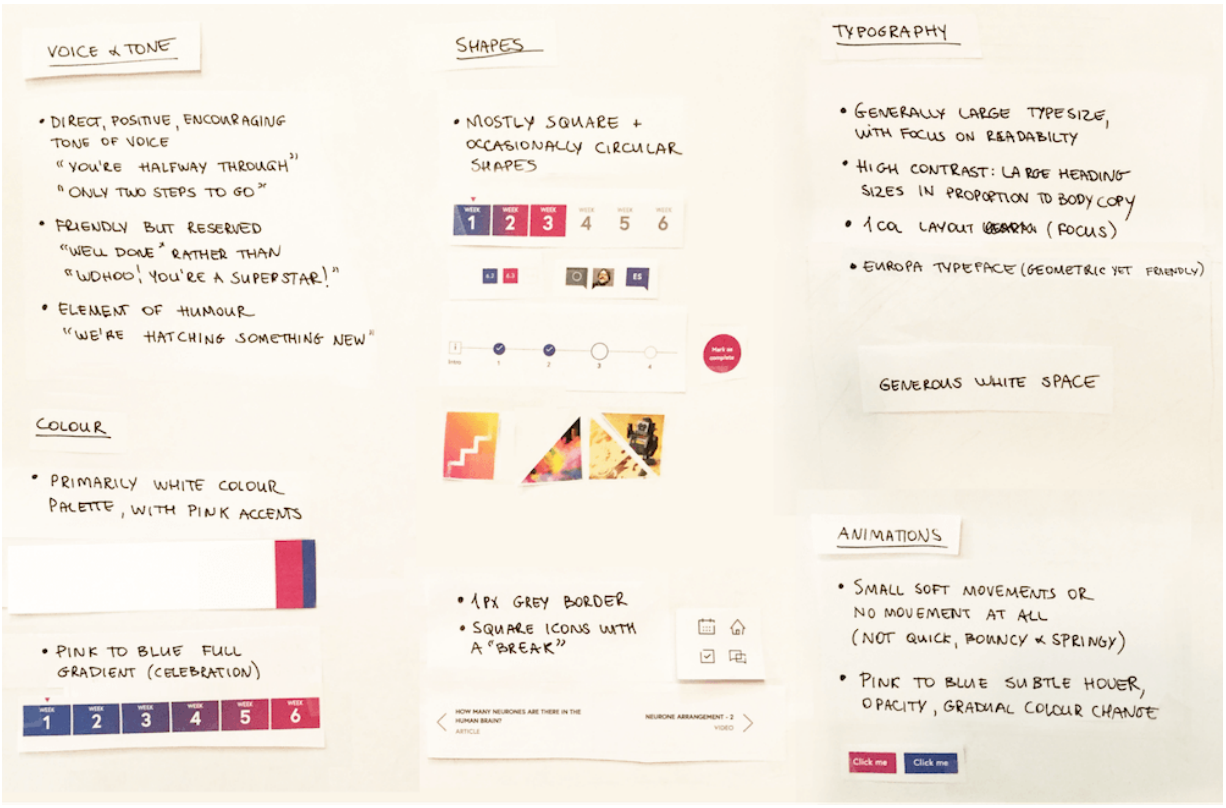
If the design has been around for a while, pinning down those patterns isn't always easy. We've seen (with the FutureLearn examples in chapter 4) that as your product grows, its core aesthetic can change. By the time you look at the styles, you might notice that some of them are more effective than others and have stronger associations with your brand.

Signature patterns[2] is a useful kick-off exercise to get the whole team (not only designers) on the same page, especially if they're not used to thinking about perceptual patterns. Ask each person to write down the most memorable and distinct elements that make your product feel a certain way.

- What are the styles and tones that first come to mind when people think about your product?
- How do your users describe the aesthetic?
- Are there any moments frequently recalled in user feedback (“That pink tick always makes me smile”)?

It’s also helpful to identify places or examples where the design is off-brand; for example, “Small subtle animations, not quick bouncy ones.”

Think not only about the properties but high-level principles, combinations of different elements, and the relationships between them. For instance, instead of simply listing the colors, describe the proportions between them: “Primarily white with pink and blue accents.” Include examples of a typical representation of the pattern. Here’s what your list might look like:




*Notes from signature patterns exercise for FutureLearn.*

I find that capturing signature patterns as a team can provide guidance and inspiration for the whole process. When looking at shapes, say, if circles are a distinguishing feature in your interface, you might want to question why you use squares in some cases.

## Identifying Perceptual Patterns

Drawing on the signature patterns exercise, you should then make your own list of individual styles to work through. Here are some of the types of things that may appear in your list:

- color

- interactive states
- animations
- typography
- spacing
- iconography styles
- shapes and borders
- illustrations
- photography
- voice and tone
- sounds and audio cues

With each individual style you choose to focus on, there's a simple process to systemize them.

1. Start with the purpose.
2. Collect and group existing elements.
3. Define patterns and building blocks.
4. Agree on the guiding principles.

You won't be able to go through all of the styles in one go. Each one will need its own inventory (and possibly a sprint or longer to integrate the changes afterwards).

As you go through them, there will be overlaps: between typography and space, color and text, shapes and borders, borders and

iconography. This is good, because you can build on the properties you've defined previously and you can see how they relate to each other. For example, the colors you define will be used in interactive states; interactive states will be used in animations. When looking at typography and spacing, you can see how size of text affects spacing around it.

## Color

The goal of the first step is to make the use of color more consistent. To do that we will start by listing *the roles* color plays in your interface.

### START WITH THE PURPOSE

Wording is important. How we phrase a purpose shouldn't be vague and obvious. A statement like this, from the government of Canada's Web Experience Toolkit[3], wouldn't be particularly helpful: "Use color as a presentation element for either decorative purposes or to convey information."

Be specific. For example, color can be used to:

- Display different types and hierarchy of text (body, headings, blockquotes).
- Highlight links and actions (main CTAs, supporting CTAs, links).
- Draw attention to messages and differentiate between them (affirmation, warning).

- Separate content or create emphasis (backgrounds, borders).
- Differentiate between types of data (in charts, graphs).

The roles you define will determine the categories for your inventory.

## COLLECT AND GROUP EXISTING COLORS

Even though I prefer paper inventories because of their tangibility, it's tricky to audit styles by cutting them out on paper. A Google doc[4] can work better, and so can Keynote, Powerpoint or Sketch — whatever suits you.

Set up a blank document with the categories. As you go through the audit, you might adjust the headings or add new ones, but it helps not to start with a blank page.

Highlight links and actions

| Type | Example | Value | Feel |
|---|---|---|---|
| Main CTA | | | |
| Supporting CTA | | | |
| Links | | | |

Display different types/hierarchy of text

| Type | Example | Value | Feel |
|---|---|---|---|
| Body | | | |
| Headings | | | |
| Blockquotes | | | |

Separate content/Create emphasis

| Type | Example | Value | Feel |
|---|---|---|---|
| Backgrounds | | | |
| Borders | | | |
| | | | |

*Example of initial categories for auditing colors in a Google doc.*

For each category add:

- **Type**
  Specify the type of object the color is applied to, such as a call to action, a heading, a feedback message, and so on.

- **Example**
  Add a screenshot of the element, to show where color is used.
- **Value**
  Specify the hex value.
- **Feel**
  If the purpose of the color is to create a certain mood or feel, specify that.

You'll end up with a list of color instances grouped by purpose. Here's an example of an audit for links and buttons in the public library's interface. In the same manner, I would collect and group text colors, feedback messages, backgrounds, borders, and so on.

## Highlight links and actions

| Type | Example | Value |
|---|---|---|
| Main CTAs |  | #1B7FA7 |
| |  | #0095C8 |
| Supporting CTAs |  | #E32B31 |
| |  | #E32B31 |
| |  | #BB1D12 |
| |  | #2799C5 |

| | | |
|---|---|---|
| | Search | #1B7FA7 |
| Links | Collections<br>Locations | #5F5B54 |
| | International Visitors | #36322D |
| | ‹ HOME | #7B756F |
| | Atmospheric | #0095C8 |
| | My Book Bag | #1DA1D4 |
| | YOUNG ADULT | #CC1A16 |

*Audit of links and buttons, conducted in a Google doc.*

Some colors will have a specific feel associated with them. In TED's interface, black headers are used to create a more cinematic (and less informational) feel. On FutureLearn, a full blue to yellow gradient helps to create a celebratory mood when a learner completes a milestone.

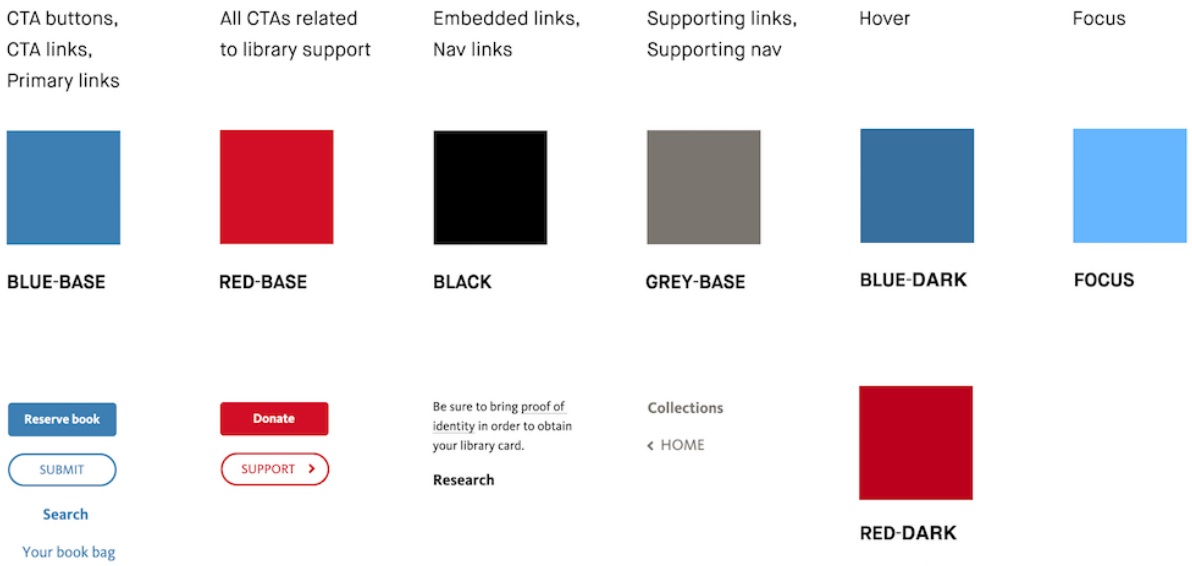

*Use of full gradient on FutureLearn for a milestone celebration.*

If there are specific emotional qualities that color is meant to bring into your product, it's important to capture that. Misuse can weaken the purpose colors were intended for. Using a full gradient in promotional modules on FutureLearn, for instance, would weaken its association with celebration.

## DEFINE PATTERNS

Next, you can define patterns of usage based on the purpose of color (and feel, if appropriate). When do you use blue links and when gray? What is the meaning of red calls to action? Why are some backgrounds gray and others brightly colored? What is the difference between black and red headings?

Don’t worry about the exact hex values just yet. What matters is that you agree on the *use of color* across the interface. Here’s an example of how patterns could be defined for links and buttons.




*How color patterns for links and buttons could be defined for the library site.*

Considering purpose first means that you’re not only adjusting variations of the same color, but sometimes will change the way color is used. On the library’s site, because some interactive elements are red, we’d expect all red elements to be interactive. But in the example below, you can’t click the “Recommendations” heading to view recommendations. In this case we can consider changing the color of the heading to black or making the heading interactive.


*A red heading on the library site that is not interactive.*

It's important to note that these decisions can alter the overall aesthetic of the site. We might decide that links and calls to action should be red instead of blue, but that could result in a more noticeable overall change to the aesthetic of the interface — suddenly there would be many more red elements in proportion to blue.

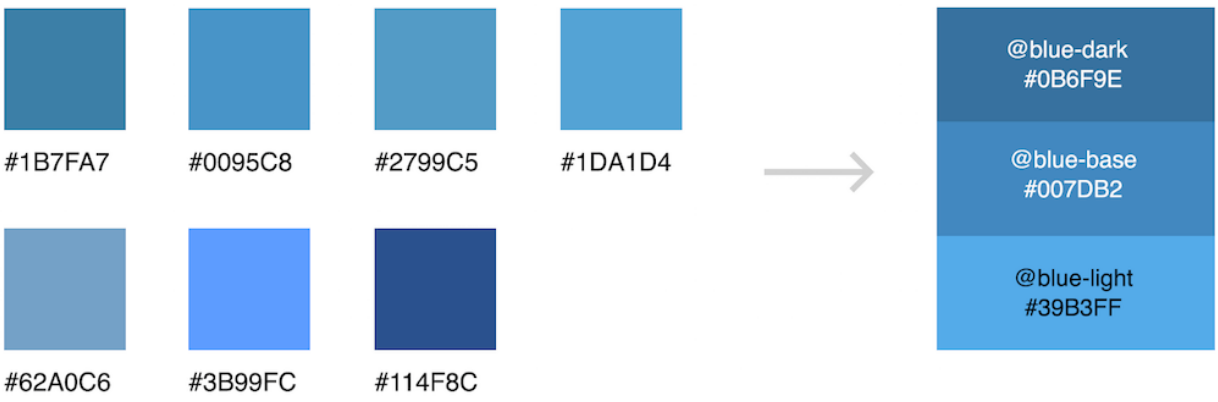
In FutureLearn's interface, we considered changing the square shapes used in the course progress module to circles, before we realized that the course navigation was a signature pattern and replacing the shapes would alter the brand's feel.

Understanding signature patterns can help you find the right balance between making improvements and making sure you don't weaken or dilute the existing aesthetic. If your goal is to change the current design, it should be done prior to the systemizing exercise.[5]

## SPECIFY BUILDING BLOCKS

During a color inventory it's not uncommon to discover dozens of variations of the same color (Marcin Treder discovered 62 shades of gray while doing the color inventory for UXPin[6]). Most of them are unnecessary and create needless complexities in design and code.

The goal of this step is to make the color palette more focused, precise and accessible. Typically this involves reducing the number of variables for each color.




Here are a few tips that can be helpful in this process.

## Start Only with What You Need

The advantage of a purpose-directed inventory is that it helps you guide and limit color choices. When you start with the role and meaning of color, you get an idea of how many options you really need. By considering where and how they'll be used, you will know, for instance, that you only need three variations of blue.

The number of shades and tints will vary, depending on your circumstances. In FutureLearn's interface, we purposefully avoided darker and lighter varieties of the same color to keep the palette crisp. It helped us make the color system simple and focused.

On the other hand, UXPin, a prototyping tool, has light and dark modes, which means the color palette needs shades and tints to provide sufficient contrast in both settings.


*FutureLearn's primary and secondary colors (left) and some of the UXPin colors (right) show how the need for color variation is different in different interfaces.*

Sometimes you need to have more options, particularly if there are multiple themes, or if you're dealing with data visualization. But it's important to avoid including the colors just to add more variety to your palette. The more choices there are, the more complex the system is, then the harder it is to achieve consistent use of color. Start with only what you need and build on it.

If you have more than two variations of the same color, it helps to specify a base value and then add additional shades and tints: 20% lighter than base, 20% darker than base, and so on. Base color values provide consistent defaults. When there are many options to consider, defaults and meaningful increments are easier to remember and work with. Specifying the base color and increments also works for other perceptual patterns, such as typography (base font size), spacing (base measurement unit), and animations, as we will see later.

## Make Sure Color Contrast Is Accessible

Test the color contrast between text and background. Adjusting or removing the values as needed will limit your palette. For example, among several variations of light gray used for supporting links on the library site, one of the frequently used values passes WCAG 2.0 standards. This would make it an obvious choice for the default value of supporting links.

There are plenty of tools to check color contrast, such as Lea Verou's Contrast Ratio[7], which is quick and straightforward to use.


*Lea Verou's Contrast Ratio checker.*

It's worth mentioning that adjusting color values needs careful balancing within the overall aesthetic. Change the blue to a darker shade, for instance, and the whole interface can suddenly feel different, perhaps less vibrant. If your color palette was created without considering color accessibility in the first place, getting the balance right will require some finessing.[8]

You can introduce different accent colors for light and dark backgrounds, or change text on colored backgrounds from light to dark, or vice versa. There are also plenty of tools for generating contrast-compliant color combinations, or for finding accessible alternatives to the original color, such as Color Safe[9] and Tanaguru Contrast Finder[10] [11].

## AGREE ON THE GUIDING PRINCIPLES

Finally, agree on a few basic principles for color usage. Guiding principles help you approach color holistically, and they can be referred to when something doesn't quite work. Some principles can be general (such as "always use accessible color contrast"); others will be more more specific to your brand (and can be defined during the signature patterns exercise).

For example, in Sky's toolkit[12] the team explains the reasons for a minimal color palette:

---

*"We allow our great content to be the color that brings the page to life. We do not color code our sites, or sections within our sites."*

---

The University of Oxford[13] explains clearly how and why to use its colors:

---

*"The (dark) Oxford blue is used primarily in general page furniture such as the backgrounds on the header and footer. This makes for a strong brand presence throughout the site. Because it features so strongly in these areas, it is not recommended to use it in large areas elsewhere. However it is used more sparingly in smaller elements such as in event date icons and search/filtering bars."*

## Animations

Even with more complex patterns, such as animations, we can follow the same process: start with the purpose, collect and group existing styles, define patterns and building blocks. Let's take FutureLearn as an example this time.

### PURPOSE AND FEEL

Specify the roles animation plays. For example:

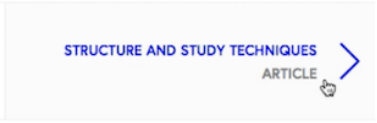
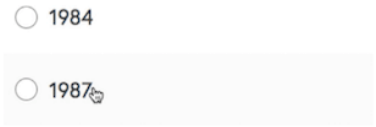
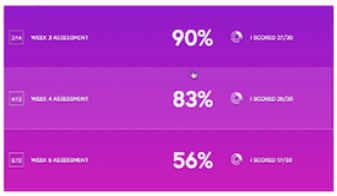
- *Soften transitions* between interactive states, such as hover states.
- *Add emphasis* to specific information or an action; for example, a nudge to encourage users to progress to the next step.
- *Hide and reveal extra information*, such as a menu being hidden to the side, a dropdown, or a popover.

The *feel* of the animation is another important aspect to consider. In *Designing Interface Animation*[14], Val Head explains how adjectives describing brand qualities can be used for defining motion. A quick, soft, bouncy motion can be perceived as lively and energetic, whereas steady ease-in-outs feel certain and decisive. To be meaningful and effective, animations should have a purposeful feel across the interface.

## AUDIT EXISTING ANIMATIONS

Once you have an idea of the role animation plays in your interface, and how it should feel, the next step is to audit existing animations. Collect the animations and group them into categories, as we did with color earlier. The examples can be captured with QuickTime or another screen recording application.

Soften state change transitions

| Effect | Example | Timing and easing | Properties | Feel |
|---|---|---|---|---|
| Colour change | Button and action link   | 2s ease | Pink → Blue | Calm and soft |
| Colour change | Step nav   Quiz question   | 0.4s ease-in-out<br><br>0.4s ease-in-out | White → Grey | Calm and soft |
| Fade in white background | Progress   | In - 0.3s,<br>Out - 1.1s ease | Opacity<br>0 → 0.15 | Calm and soft |
| Fade in colour overlay | Category signpost   | 0.4s ease | Opacity<br>0.57 → 0.65 | Calm and soft |
| Scale up | Quiz nav   | 0.3s ease | Scale<br>1 → 1.2 | Calm and soft |

*The "State Change" page from FutureLearn's animation audit, conducted in a Google doc*[15].

# DEFINE PATTERNS

Define *patterns of usage* based on the purpose and feel. In FutureLearn's interface we noticed that emphasis animations typically feel more playful, and that state change transitions are more subtle and calm.

If these are the tones you want to strike throughout the system, try aligning all the animations to them. Like the signature patterns exercise, take the examples that work well (that is, achieve the purpose effectively and have the right feel) and try out their properties with other animations from the same category. You'll end up with a handful of patterns:

| **Purpose** | **Animation effects** | **Feel** |
|---|---|---|
| Interactive state change | **Color**, 2s ease<br>**Opacity**, in: 0.3s; out: 1.1s ease<br>**Scale**, 0.4 ease | Calm, soft |
| Info reveal | **Slide down**, 0.4 swing<br>**Slide up**, 0.7s ease<br>**FadeInUp**, 0.3 ease<br>**Rotate**, 0.3 ease | Calm, soft |
| Emphasis | **Energetic pulse**, 0.3s ease-in<br>**Subtle pulse**<br>**Wiggle**, 0.5s ease-in-out | Energetic, playful |

*Animation patterns on FutureLearn, grouped by purpose and feel.*

## SPECIFY BUILDING BLOCKS

There are two important concepts in animation, which go hand in hand: timing and easing. Timing is how long an animation takes. In combination with distance, it determines speed. Easing defines how something is animated: does it start out slow and build up speed (ease-in), or does it start out fast and gradually slow down (ease-out)? Additionally, we would define the properties that are being animated, such as color, opacity, scale, and so on.

Timing, especially, is crucial in animation. Getting the timing right is not so much about perfect technical consistency as making sure that the timing *feels* consistent. Two elements animated at the same speed can feel completely different if they are different sizes or travel different distances.

I like Sarah Drasner's[16] idea to deal with animation timing like we deal with headings in typography.[17] Instead of just a single value, start with a base and provide several incremental steps. For example, if the base time is 0.5 seconds, smaller items that travel a shorter distance (such as an icon scaling up) would take less time. Items that travel longer distances (such as a menu popping up) would require more time. A full-screen transition would be one or two increments above the base value.

## AGREE ON THE GUIDING PRINCIPLES

If your team is not yet confident with animation, it may be worth defining general principles, such as “Reserve animation for the most important moments of the interaction,” and “Don’t let animation get in the way of completing a task.” But the most helpful principles are usually specific to how your team approaches animation. For example, the Salesforce Lightning Design System[18] principles advise keeping the timing short and the motion subtle.

Guiding principles can also include spatial metaphors, which can provide a helpful mental model to animators. Google’s Material Design[19] is a great example of how viewing an interface as physical materials can provide a common reference for designers and developers when thinking about motion in their applications.[20]

## Voice And Tone

Voice and tone in UI copy play a fundamental role in how a product is perceived. This is particularly the case in voice-based interfaces but also for people who experience digital products through senses other than sight. In a recent conversation with Léonie Watson[21], an accessibility expert who is also a screen reader user owing to blindness, she noted that her experience of digital products “often comes through in the form of style of writing.”[22]

However, team members who define the interactions and patterns are often not the same people who will be working on voice and tone. This can lead to a patchy and thoughtless style of writing across the experience. To make sure voice and tone are expressed

consistently and purposefully, design, brand and marketing teams need to coordinate their efforts when defining patterns.

## AUDIT VOICE AND TONE PATTERNS

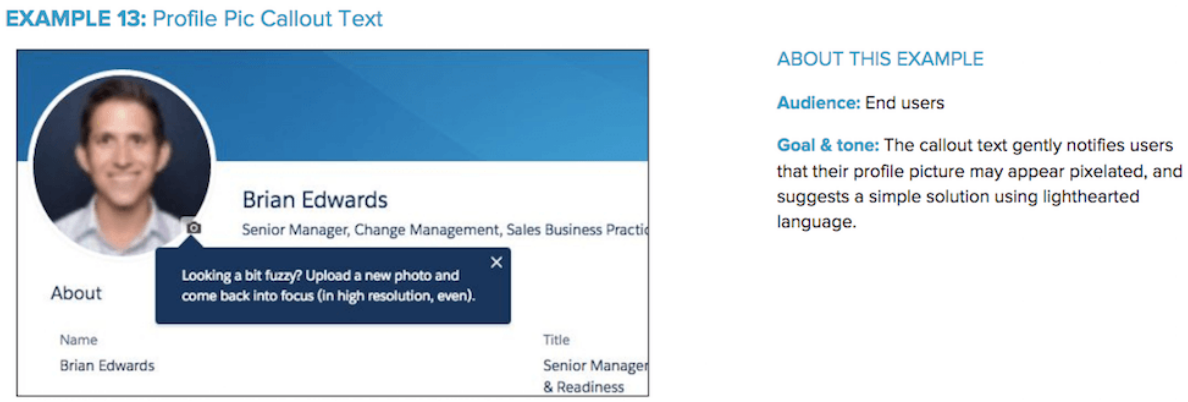
In addition to collecting all the UI copy in a Google doc, there are more creative ways to audit voice and tone. In her blog post[23], content strategist Ellen de Vries, shared how she "harvested" the language patterns during Clearleft's[24] voice and tone refresh: from phrases people use in meetings and pitch presentations, to informal conversations with the founders. They even made a mood board to see how language and imagery work together across Clearleft's website.


*Inventory of voice and tone patterns for Clearleft.*

## DEFINE PATTERNS

Once the copy and other materials have been gathered, define the patterns and formulate guidelines for how they can be applied in the interface. MailChimp's Voice & Tone[25] is one of the most effective examples of how language patterns can be defined. The tone shifts to respond to the emotional condition of the user: when it's appropriate to be humorous and lighthearted ("Fine piece of work"), and when the copy needs to take a serious practical tone ("We're expecting a problem at one of our data centers").

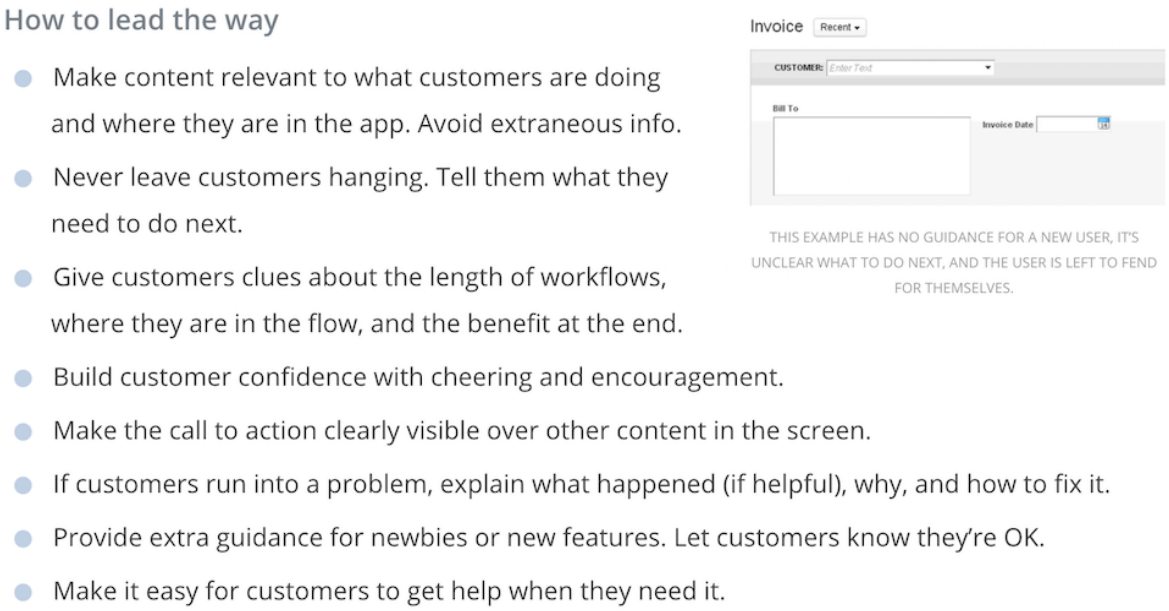

Similarly, Salesforce gives a breakdown of common use cases and suggests patterns of copy to use with each one. The goal of the message affects the emotional tone, such as "suggest a solution using lighthearted language".




*Example of voice and tone patterns in Salesforce voice and tone guidelines.*

## AGREE ON THE GUIDING PRINCIPLES

Like the overarching design principles (see chapter 2), guiding principles for individual styles shouldn't be vague and general. Not only Intuit's Harmony[26] lists the voice and tone principles ("Lead the way," "Keep it simple," "Have fun"), they also explain how to do all those things.



*Intuit's voice and tone guidelines explain how to apply the principles.*

## Summary

Each style should be approached as a system in its own right — typography system, layout system, color system, and so on. They should be interconnected and directed towards achieving a larger purpose — to help shape how a product is perceived.

To do that, look at the big picture first. Capture the aesthetic qualities as a whole and identify the patterns that are particularly effective at expressing it. Then you can follow a similar process for all the styles: start with the key roles a style has in the context of your product, audit existing instances, and then define patterns and building blocks. The guiding principles help to connect everything together and link it back to the purpose.

Let's now look at pattern libraries as a tool for documenting and sharing the patterns.

—

1. http://smashed.by/pivotal
2. See chapter 4.
3. http://smashed.by/wetoolkit
4. http://smashed.by/coloraudit
5. As explained in chapter 8.
6. http://smashed.by/colorinventory
7. http://smashed.by/contrastratio

8. And conversely, in a project where color accessibility was a factor right from the start, you wouldn't end up with such widely different palettes.
9. http://colorsafe.co/
10. http://smashed.by/contrastfinder
11. For further reading on color accessibility and balance with aesthetics, I highly recommend *Color Accessibility Workflows* by Geri Coady.
12. http://smashed.by/skytoolkit
13. http://smashed.by/oxfordstyle
14. http://smashed.by/designingia
15. http://smashed.by/animationaudit
16. https://sarahdrasnerdesign.com/
17. http://smashed.by/animationdesignsys
18. http://smashed.by/salesforcemotion
19. http://smashed.by/materialmotion
20. For a more detailed overview of the process, see my article "Integrating Animation into a Design System".
21. https://tink.uk/
22. Interview with Léonie Watson, August 2017
23. http://smashed.by/voicetoneinventory
24. https://clearleft.com/
25. http://voiceandtone.com/
26. http://smashed.by/intuit

CHAPTER 10

# Pattern Libraries

*In this chapter we'll look at some practical techniques to set up foundations for a long-lasting and multidisciplinary pattern library.*

For some teams, a systematic approach to designing and building digital products is almost unthinkable without a pattern library. But as we've discussed throughout the book, a pattern library is not the system itself, it is a *tool* for documenting and sharing design patterns. To be effective, it needs a system foundation at its root. In chapter 7 we looked at some of the general strategies for establishing such foundation:

- Agree on the main goals and objectives, related to both the interfaces and how the teams operate, such as "Define and standardize reusable design patterns," "Define guiding design principles," "Establish a pattern library."
- Break up objectives into manageable stories and create a simple roadmap for your system.
- Make your progress transparent by documenting and sharing it. For many teams, making their pattern library public made a huge difference to their progress and the team's confidence in the work they were doing.
- Create a culture of knowledge sharing by making the design language visible and accessible to the whole team.
- Practice thinking in systems through experiments, workshops and group exercises.

In the experience of every team I spoke to, multidisciplinary pattern libraries are more resilient and enduring. They facilitate a shared language across the organization and bring value to everyone. Conversely, a pattern library built to serve the needs of one discipline is more fragile.[1]

The technical complexity of Sipgate's first pattern library prevented designers being fully involved. Without the knowledge of what patterns existed in the system, they would sometimes create a page from scratch in Photoshop, where existing patterns could be used.

*"It was often left to developers to fit the design with the existing patterns, who had to tweak them until they fit. This led to numerous if-statements, exceptions and duplicate patterns."*
*— Mathias Wegener, front-end developer, Sipgate*

---

Even though developers were determined to make a pattern library comprehensive and up to date, it was impossible without designers' active involvement.

Similarly, patterns designed without the content discipline's perspective can fall apart in everyday use. We end up designing patterns that are too closely tied to specific content, such as a module where an extra line of copy would push an important call to action below the visible area. Or we force content into patterns that aren't designed for it, compromising both content and design.

In this chapter we'll focus on establishing foundations for a pattern library that can support the goals of multiple disciplines.

## Content

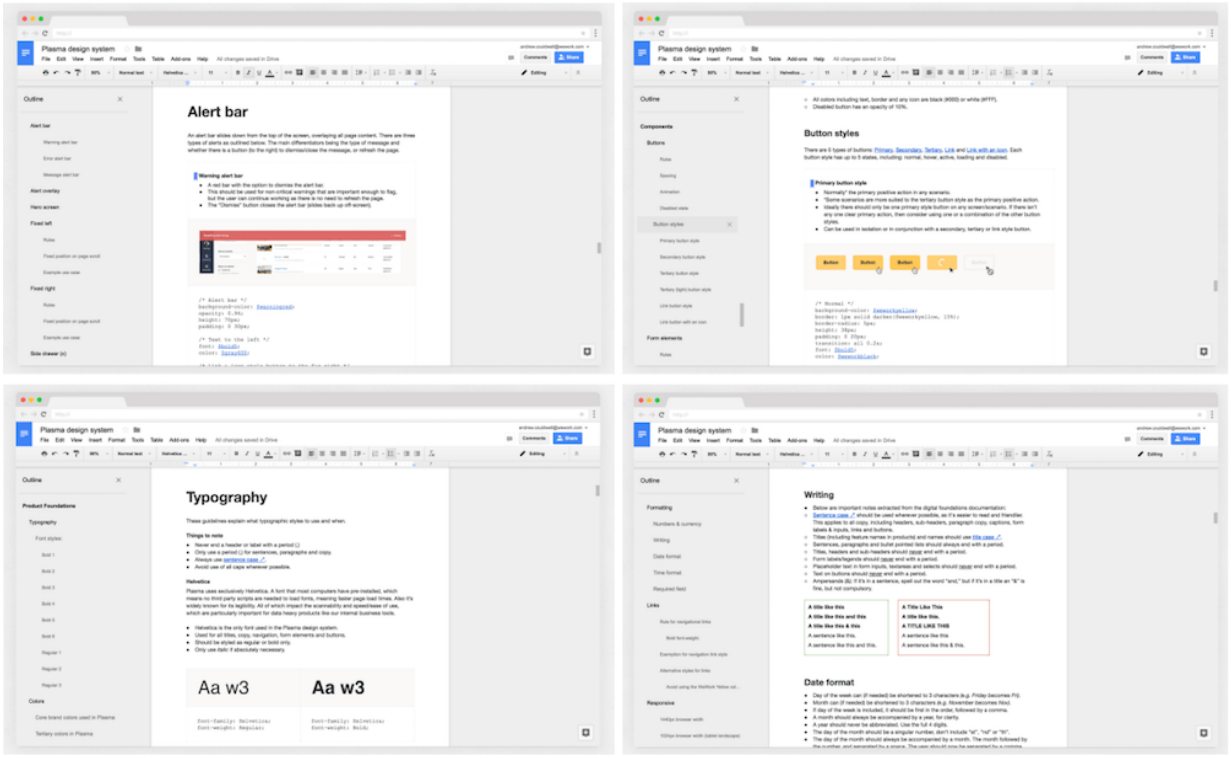
Looking back, at FutureLearn we spent far too much time researching tools and working out what the pattern library should look like. There wasn't full agreement on how it should be designed and built, and the work progressed slowly. Switching our focus to the *content* of the library made a big difference — both to our progress, and the team's morale.

Going through the process described in the previous two chapters will give you a good grasp of what can go into your pattern library. It can be documented simply, using Google Docs[2] or another collaborative writing app. There are two main benefits of this:

- First, everyone on the team can access the content to review, make edits and provide their feedback. By using a tool that's familiar and easily accessible, you give more people an opportunity to be involved.
- Second, a folder in Google Docs is like an MVP pattern library — the team can start using at as a reference right away. Once you have the content, it will be

easier to figure out how the website for it should be designed and built.

Here's how Andrew Couldwell at WeWork[3] captured some of the patterns for the Plasma design system[4] using Google Docs:




*Documenting patterns in Google Docs for the Plasma design system.*

The team was able to get down all the core patterns and their definitions quickly, instead of being held back by build and design constraints.

## Organization Of Patterns

When documenting the content, the question will likely arise of how the patterns should be organized. The navigational structure is one of the things teams tend to struggle to agree on. Should buttons be separate or grouped with form elements? Where does the footer go? Should pagination be part of the navigation section?

The structure doesn't have to be perfect at the start — you can (and probably will) change it later. What's important is that the team are on the same page. Having a common methodology to organizing patterns will make it easier to add and find things

as your pattern library grows. The same thinking can apply not only to the pattern library, but front-end architecture and the design files.

Let's take a look at some of the common approaches.

## ABSTRACTING PERCEPTUAL PATTERNS

The simplest way to think about the structure is in terms of components and styles (functional and perceptual patterns). As we saw in the previous chapter, perceptual patterns are connected and work together. Abstracting them makes it easier to be aware of their role in the system. Here are a few examples of how perceptual and functional patterns are referred to.

| **Pattern library** | **Functional patterns** | **Perceptual patterns** |
|---|---|---|
| Airbnb DLS | Components | Foundation |
| Atlassian[5] | Components | Foundations |
| BBC GEL[6] | Design Patterns | Foundations |
| IBM Carbon[7] | Components | Style |
| Lonely Planet Rizzo[8] | UI components | Design Elements |
| Marvel[9] | Components | Design |
| Office Fabric[10] | Components | Styles |
| Salesforce Lightning Design System[11] | Components | Design tokens |
| Shopify Polaris[12] | Components | Visuals |

*There seems to be a general consensus to refer to functional patterns as "components," but more diversity in terminology used for perceptual patterns.*

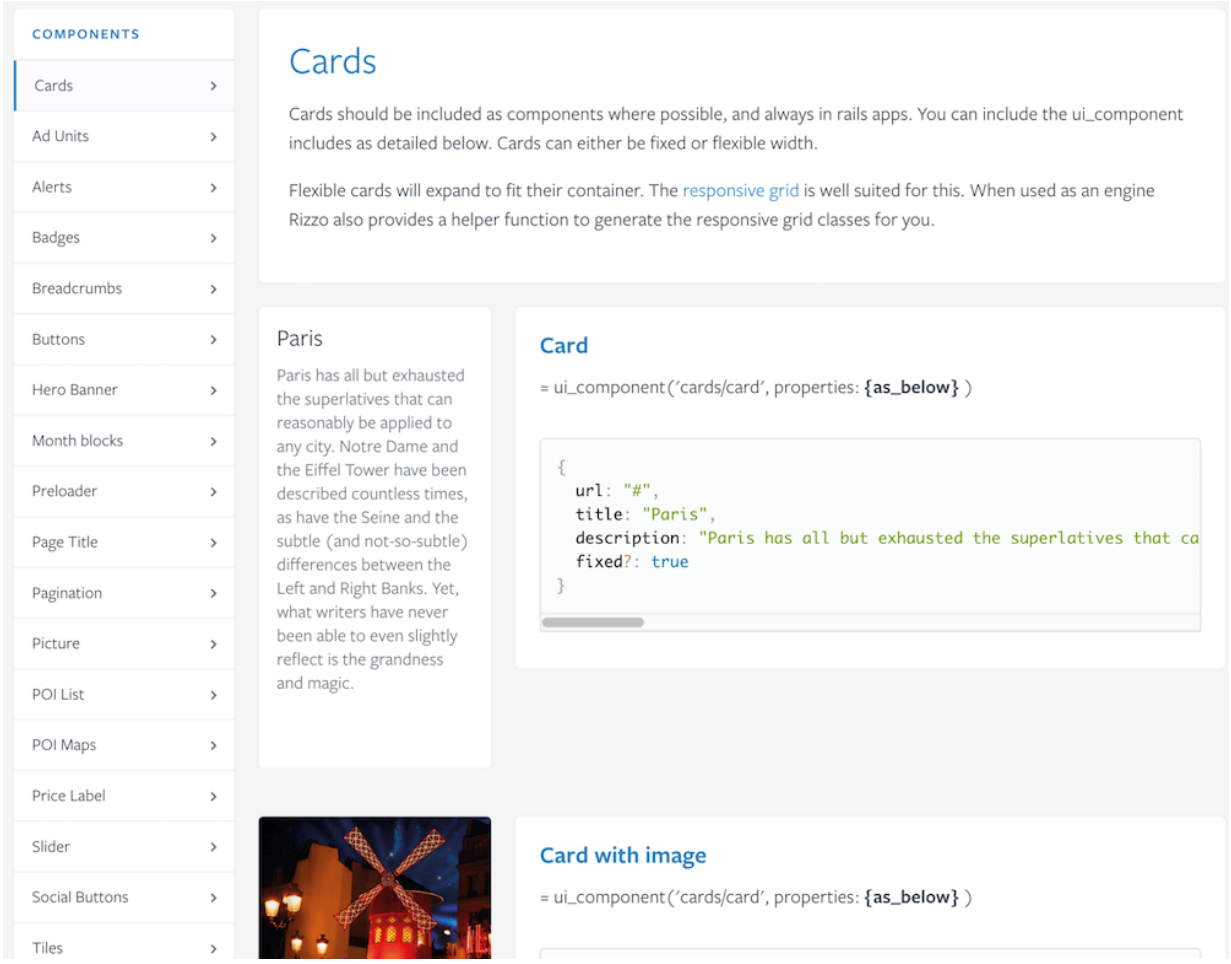
## ORGANIZING FUNCTIONAL PATTERNS

While the number of styles is limited, the list of functional patterns can keep growing. The findability of modules is one of the greatest barriers to pattern library adoption. If team members don't know that a pattern exists or can't find what they need, they are likely to create a new one or go outside the pattern system.

Teams organize modules alphabetically, hierarchically, by type (navigation, form elements, and so on), by purpose, or in entirely different ways.

## Alphabetical

In IBM's Carbon design system, Sky Toolkit[13] and Lonely Planet's Rizzo (among others) components are kept in one list and arranged alphabetically.




*Most components are arranged alphabetically in Lonely Planet's Rizzo (although navigation and form elements are in separate groups).*

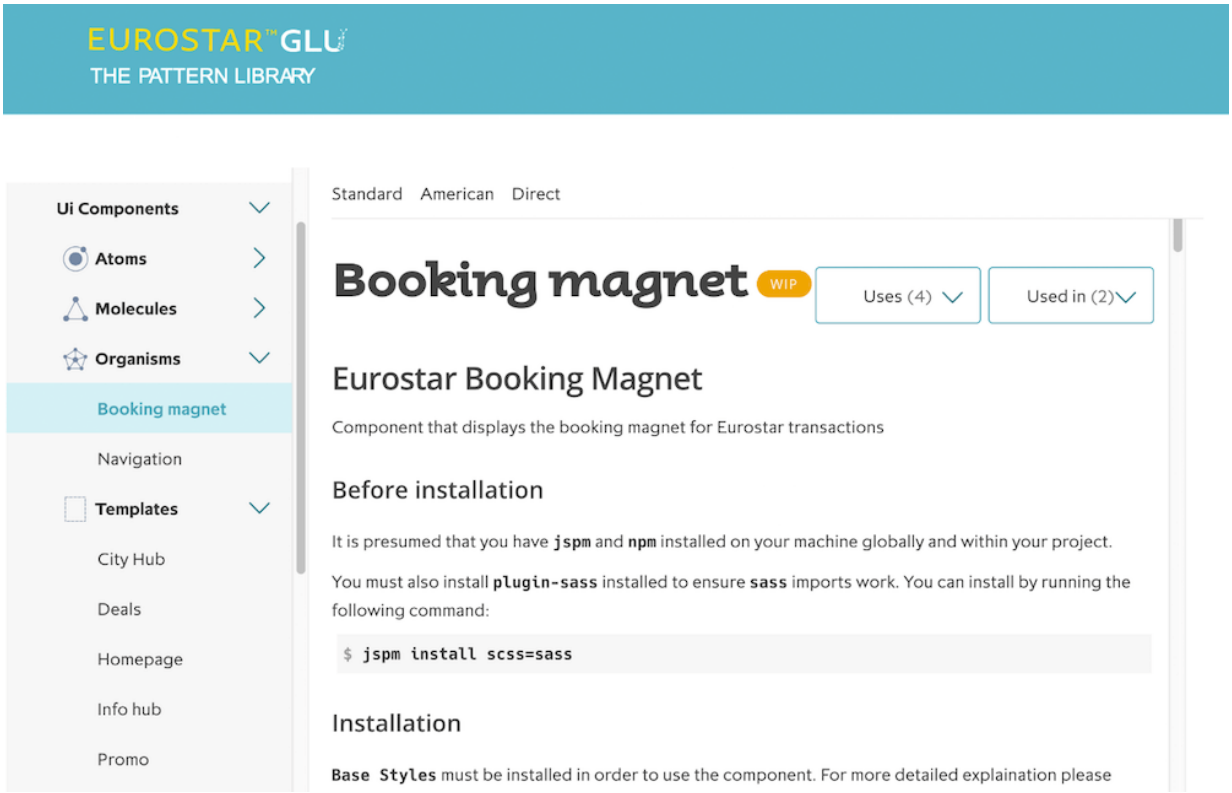
A single list makes decision-making easier — it avoids the debates about how things should be categorized. If the list grows and becomes unmanageable, teams start

experimenting with other options to make components more discoverable.

### Hierarchical

Another way to classify functional patterns is in terms of their complexity. Some teams separate granular elements from more complex ones. The levels of granularity vary in number and perceived complexity.

Atomic design, pioneered by Brad Frost, is a popular example of hierarchical categorization. Atoms are the basic building blocks, which combine to create more complex standalone elements: *molecules*. For example, a form label, input and button combine into a search form. Molecules join together into *organisms* (such as a site header), and organisms into *templates* and *pages*.




*Components arranged hierarchically in Eurostar's GLU*[14].

As a methodology, atomic design can bring many benefits. Thinking of patterns as nested matryoshka dolls can help to reuse the elements, since you're conscious of how the levels build on each other. Defining the rules for combining and encapsulating the patterns can promote consistency across the system. For the team at FutureLearn,

the analogy with chemistry provided a shared point of reference when we were new to modular design thinking.

But it's important to remember that atomic design (or any other methodology) might not be right for you right out of the box. At FutureLearn we struggled to find a use for "templates" and "pages." The team preferred to work with smaller elements, so we could have greater flexibility over how they were combined.

What's more, we spent far too much time debating whether something was a molecule or an organism. Since the team didn't see enough distinction between the two types, they were merged together. We ended up with two levels of hierarchy: atoms and molecules.

More than two types of functional patterns can get confusing but separating granular elements from more complex ones makes sense — both in the pattern library and in the code. Teams do it to varying degrees, with or without following atomic design nomenclature.

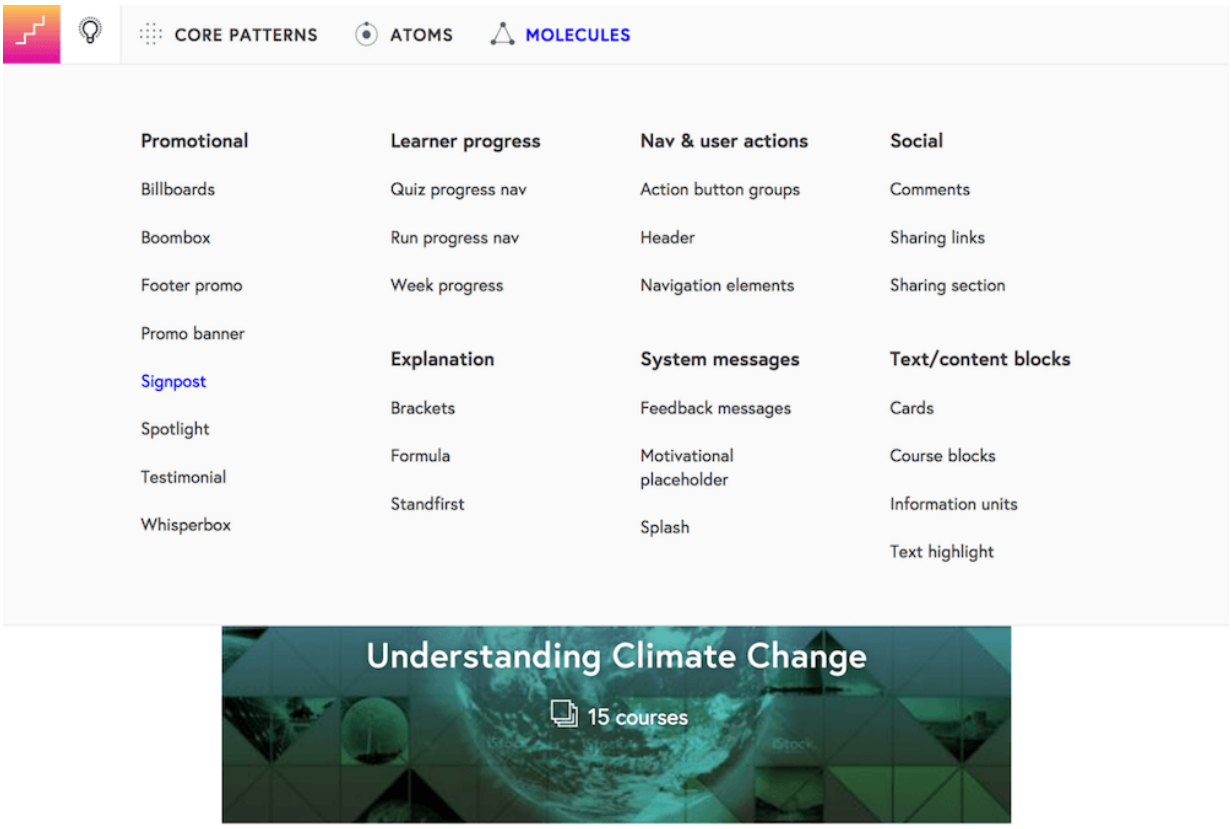
| | | | | |
|---|---|---|---|---|
| Atomic Design[15] | Atoms | Molecules | Organisms | Templates |
| Ceasefire Oregon[16] | Elements | Components | - | - |
| ClearFractal[17] | Units | Groups | - | - |
| GE Predix[18] | Basics | Components | Templates | Features |
| Lewis+Humphreys[19] | Elements | Components | Compositions | - |
| WeWork's Plasma[20] | Components | Patterns | - | - |

What I find interesting is that the strictness of a system (discussed in chapter 6) may be reflected in how the pattern library is structured. The more granular the patterns are, the more loose and flexible is the system. In a strict system, like Airbnb's or GE's Predix, the larger patterns are documented: user flows, templates and pages. In a system like TED's or FutureLearn's, you would document smaller parts and leave it up to the individual designers to combine them as they see fit.

### By Purpose or Structure

At FutureLearn we never stopped experimenting with ways to organize modules: in one long list, hierarchically (following atomic design methodology), by compositional role on the page (“intros,” “outros,” “heroes,” and “bridges”). But everything was either too restricting, or too complex to work with.

After two years of trial and error we settled on classifying elements by purpose — promotional modules, modules focused on encouraging learner progress, modules around communication with the user, social modules, and so on.



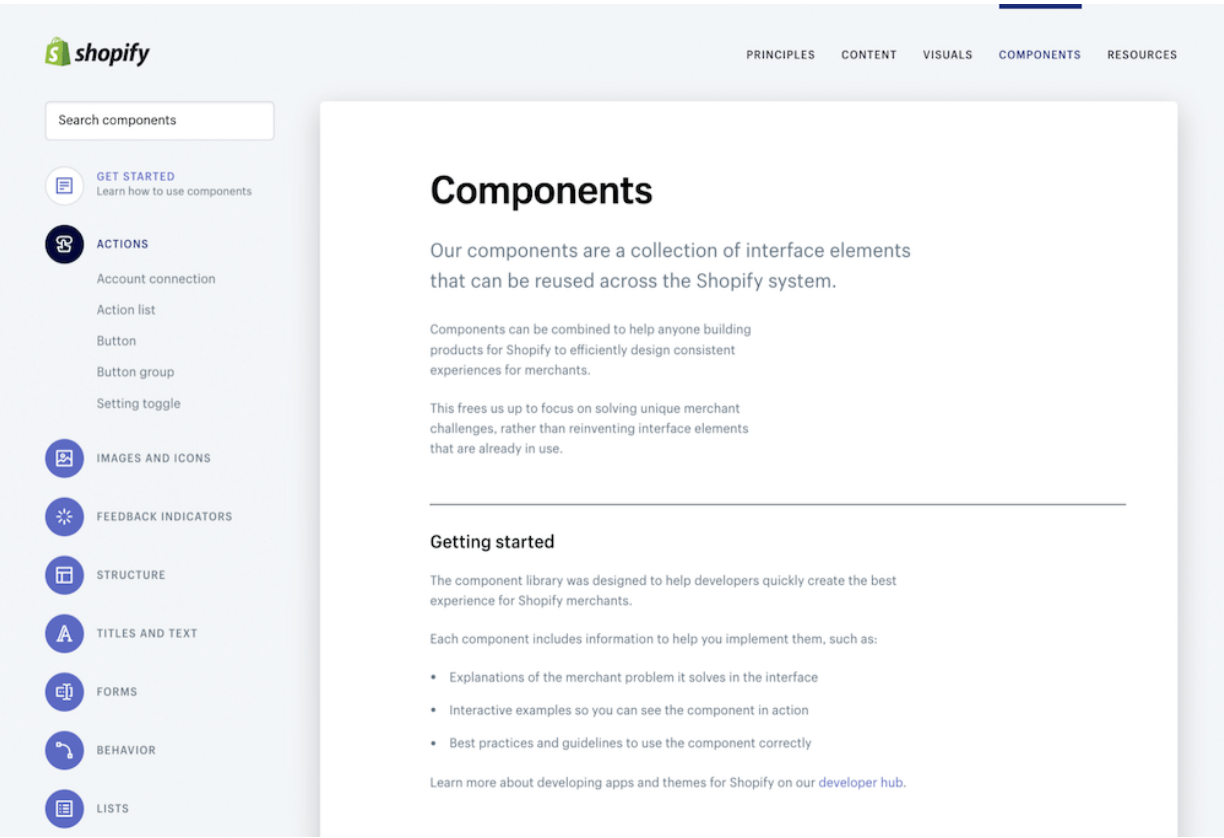

*Modules arranged by purpose in FutureLearn’s pattern library*[21].

Organizing patterns by purpose gives the team some guidance and inspiration for where to use a specific module. This structure also fit with our purpose-directed approach for defining patterns.

In Shopify Polaris, components are also categorized based on the team’s mental models. The initial grouping was the outcome of an open card sort and usability testing. Even though there isn’t perfect alignment among different disciplines, the internal user research is continuously shaping how patterns are organized:

*“Designers tended to think in terms of structure. Developers tended to default to functionality. Content strategists tended to combine both. We’re conducting a range of usability studies to understand how well the grouping of components is working for people.”*[22]

*— Selene Hinkley, content strategist, Shopify Polaris*




*Components in Shopify Polaris are arranged by structure and function.*

These are just some of the ways teams organize patterns. What’s important is that the choice is grounded in how people who use it think. Find a structure that’s right for your team. If it doesn’t work, if people struggle to find what they’re looking for, continue experimenting with different approaches. This can take time. The phrase I hear the most from all the teams with effective patterns libraries is that their “work is never done.”

# Pattern Documentation

Although many things can be documented alongside each pattern, trying to cover everything right away is not feasible, especially for smaller teams.

To see tangible benefits sooner, start with a lightweight overview of the main patterns. Once you have a simple foundation, you can improve the pattern library over time, by adding features and information the team needs. Here are some of the points to consider when documenting functional and perceptual patterns.

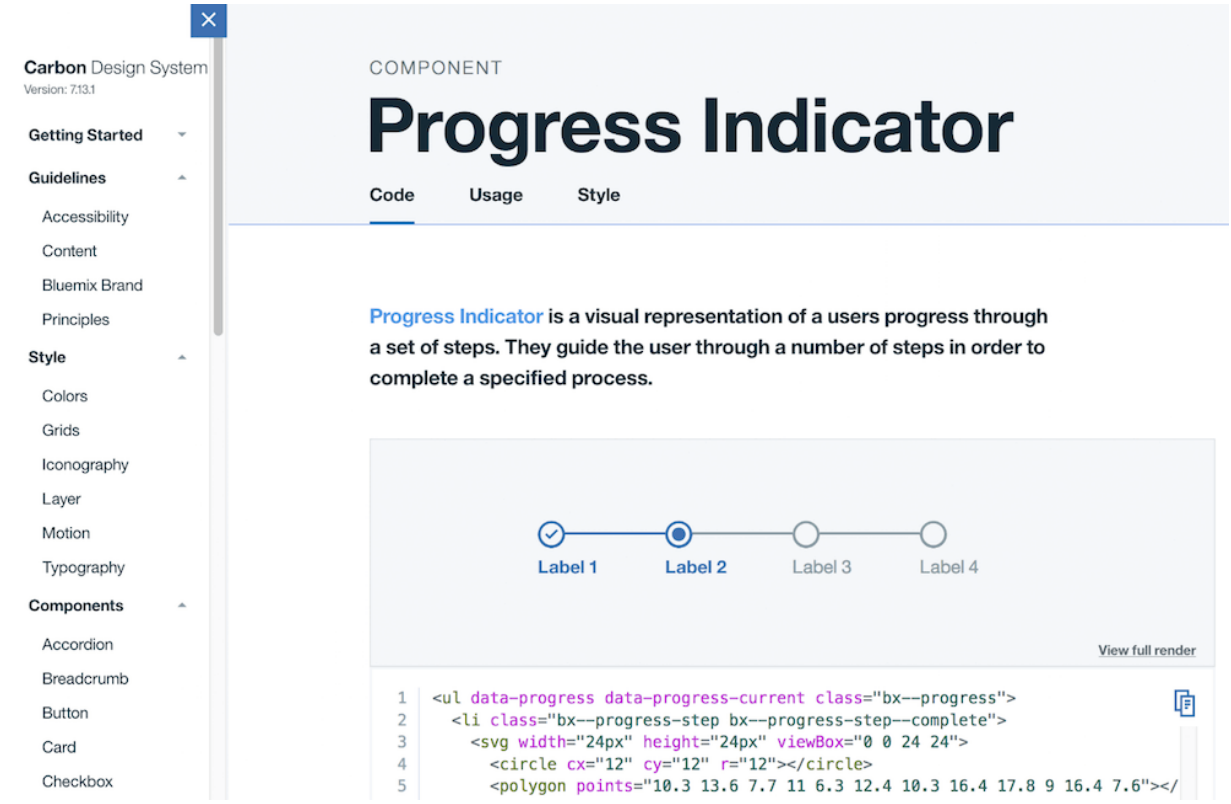
## DOCUMENTING FUNCTIONAL PATTERNS

To make documentation focused and easily scannable, start with the basics:

- name
- purpose
- example (visual and code)
- variants

### Name

Throughout the book I've tried to emphasize the importance of a well-chosen name. A good name is focused and memorable, and embodies the purpose of the pattern. Ideally, someone should be able to glean the purpose from the name, without needing to read the description. To help make the page more scannable, names should be prominent and stand out from the rest of the content.




*Each pattern's name is prominently displayed in IBM's Carbon.*

## Purpose

When browsing a pattern library, most people skip descriptions, especially long ones. That's why they should be focused and to the point: typically, one or two sentences explaining what a pattern is and what it's for is enough. Although this seems like a simple task, in practice capturing the purpose of a pattern concisely and accurately is not easy. Too often we come up with vague descriptions that don't have a lot of practical value.

Take a look at how the team at Sipgate initially described a component called "Showcase":

---

*"Use Showcase to present multiple types of information with a media file."*

---

Even though factually correct, it doesn't communicate the purpose of "Showcase", which makes it more likely to be misused or duplicated. Later, the team adopted a new practice for defining a pattern's purpose, and writing descriptions. Here's how it was applied to another example:

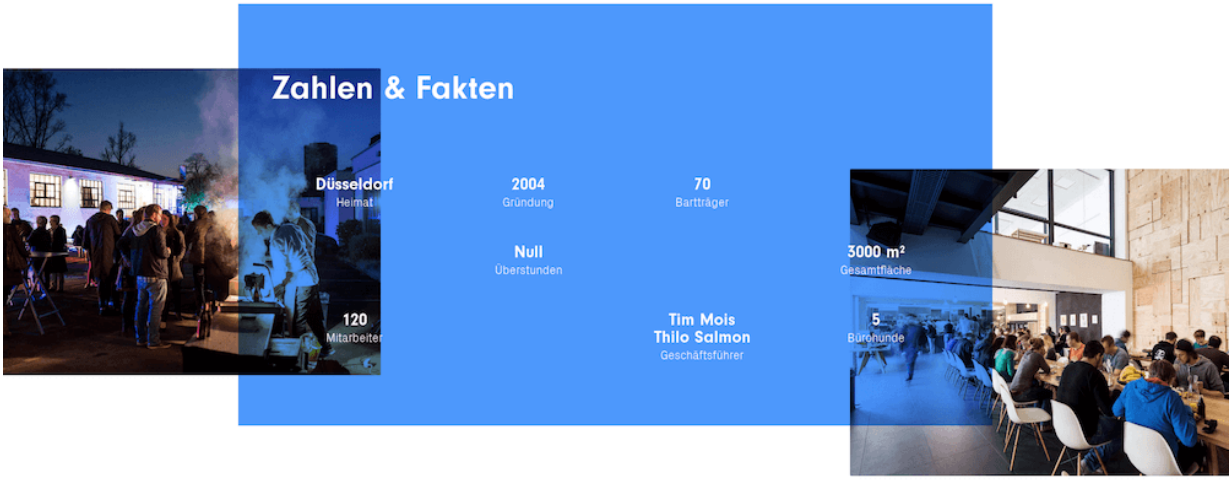
---

*"Fact Grid is a shortlist of facts or bits of interesting information. Use Fact Grid to give the reader an immediate impression about the upcoming content."*

---

This second description is much more effective at communicating what the pattern is for. You might even be able visualize the "Fact Grid" just by reading these two sentences.

Additionally, there are design and content recommendations for making the pattern achieve its purpose in the most effective ways, such as "Maximum of 3 lines per fact" or "Maximum of 12 facts." Collaborating with a content discipline can be invaluable for defining these rules.

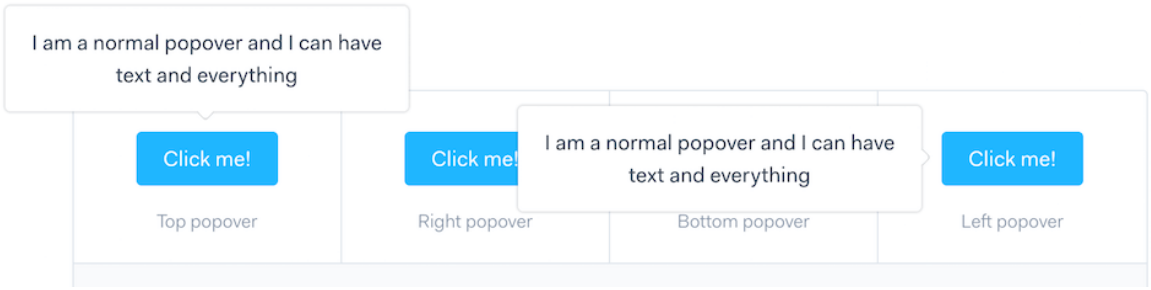



*Fact Grid in Sipgate's pattern library*[23].

## Example

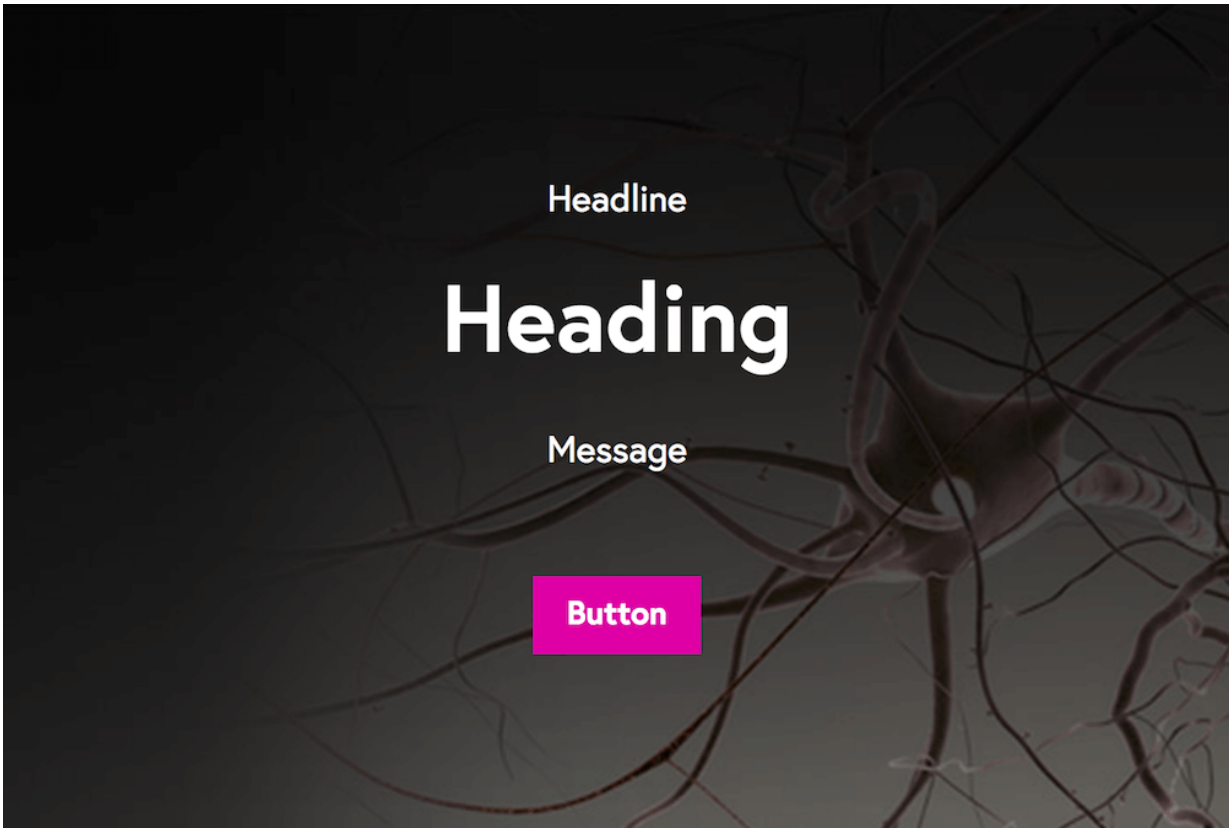
A good example helps to enhance the understanding of the pattern's purpose. In Marvel's style guide, examples are self-documenting and show multiple variants and

use cases. The UI copy in the pattern helps guide usage further.




*In Marvel's style guide, examples make it easy to see how different popovers behave.*

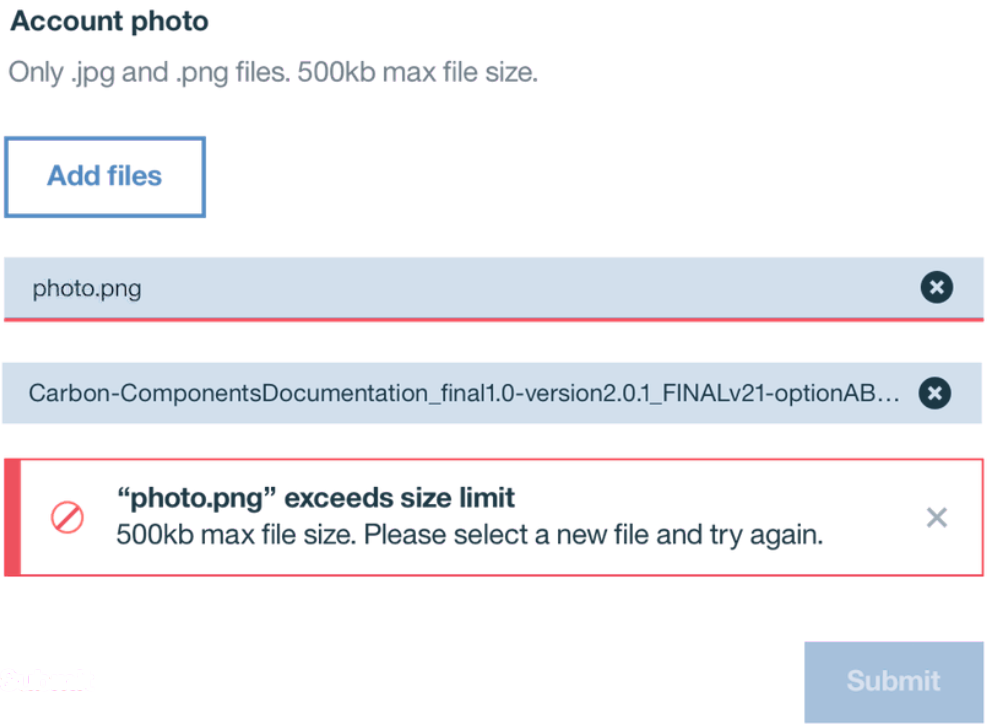
A lesser example doesn't help to communicate the purpose. Nothing in the way "Billboard" is presented in FutureLearn's library suggests that it's a "prominent promotional element." Making small adjustments, such as changing the default copy and the background image could help express its purpose more clearly.




*"Billboard" in FutureLearn's pattern library is less than inspiring.*

A living instance of a pattern, with component code alongside it, is usually preferred — it can show responsiveness, interactions and animation. But in some cases a static

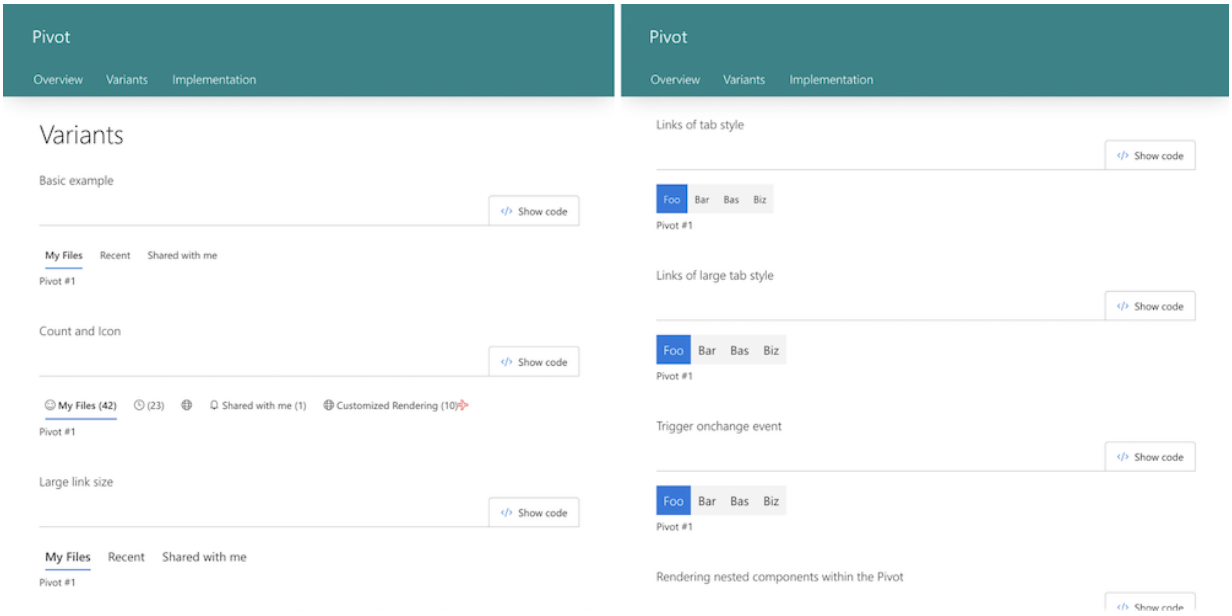
image or a GIF is more helpful, particularly when you need to show a specific behavior or a state that can't be recreated in a living example.




*Carbon uses a combination of live and static examples to illustrate specific behaviors.*

## Variants

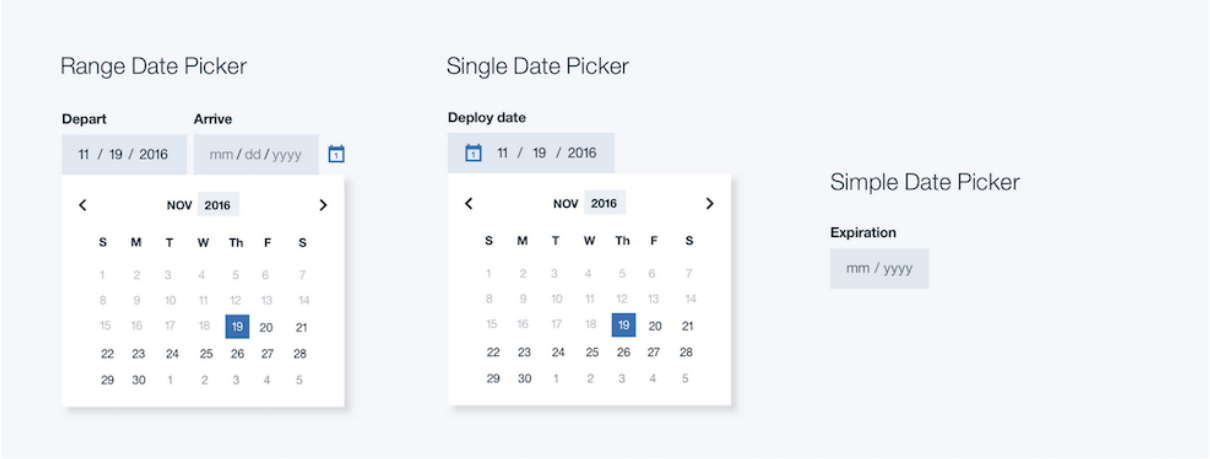
Presenting variants alongside one another, as a suite, makes it easier to see at a glance what is available within the purpose. Not only that, we need to know how the options are *different* from one another. Although Office Fabric helpfully presents all the variants, it doesn't explain the differences between them.




*Variants in Office Fabric.*

Compare it with Carbon's presentation, which clearly states the purpose of each variant.

| TYPE | PURPOSE |
| --- | --- |
| **Range Date Picker** | To select a range of dates, accompanied by a calendar widget. |
| **Single Date Picker** | When a user needs to select one date, accompanied by a calendar widget. |
| **Simple Date Picker** | When the date is known by the user, and they don't need a calendar to anticipate the dates. |




*Types of Date Pickers*

*Types of date pickers and the differences between them clearly explained in Carbon.*

Similarly, Atlassian's design guidelines describe when to use each type of button (although, from my point of view, some of the copy, such as "Use when you have nothing to prove," could benefit from being more precise).

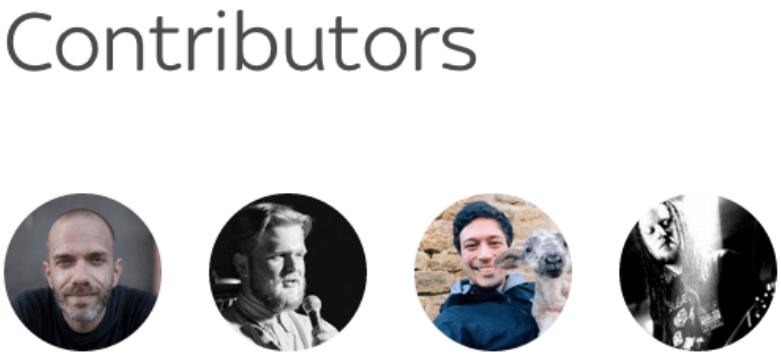
| Button | Variation |
| --- | --- |
| Label only | The most general use case. Use when you have nothing to prove. |
| Icon and label | Use when you want to draw more attention to the button, or when an icon helps to convey more meaning. |
| | Icon only - Use when space is constrained and the function of the button is obvious. A button with a label is preferred over this approach. |
| Subtle | Use when a toolbar or standard buttons would draw attention away from more important content. It is often used at the top of a page or section and must use an icon and label. |
| Disabled button | Use when another action has to be completed before the button is usable, like changing a field value or waiting for a system response. Use only with primary and standard button types. |
| Split | A button with an attached drop menu of related actions. The labelled section shows the 80% use case action for easy access, while the attached menu shows a list of related actions (including the one shown on the button). |

*Button variations explained in Atlassian's design guidelines.*

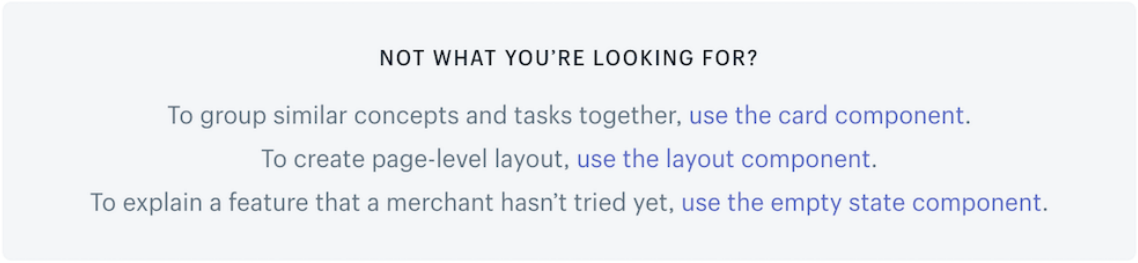
There are many other aspects that can be important to document, such as:

- **Versioning of components.** If products supported by a pattern library get major upgrades, some components can benefit from documenting changes in the API or UI elements, relative to previous versions. The same goes for obsolete elements and their replacements.
- **Team members.** Listing people involved in the creation of the pattern, like the Sky Toolkit[24] example below, can give people a sense of ownership, and also

helps with future development.




- **Related patterns.** Shopify Polaris[25] helpfully shows alternatives if the pattern is not quite what you're looking for. This can reduce the chance of patterns being duplicated.




Depending on your team's needs, there are many other bits of information that can be included. In his article "Taking The Pattern Library To The Next Level" Vitaly Friedman shared two checklists: one for the patterns to document, and another for the things to include alongside each pattern.[26]

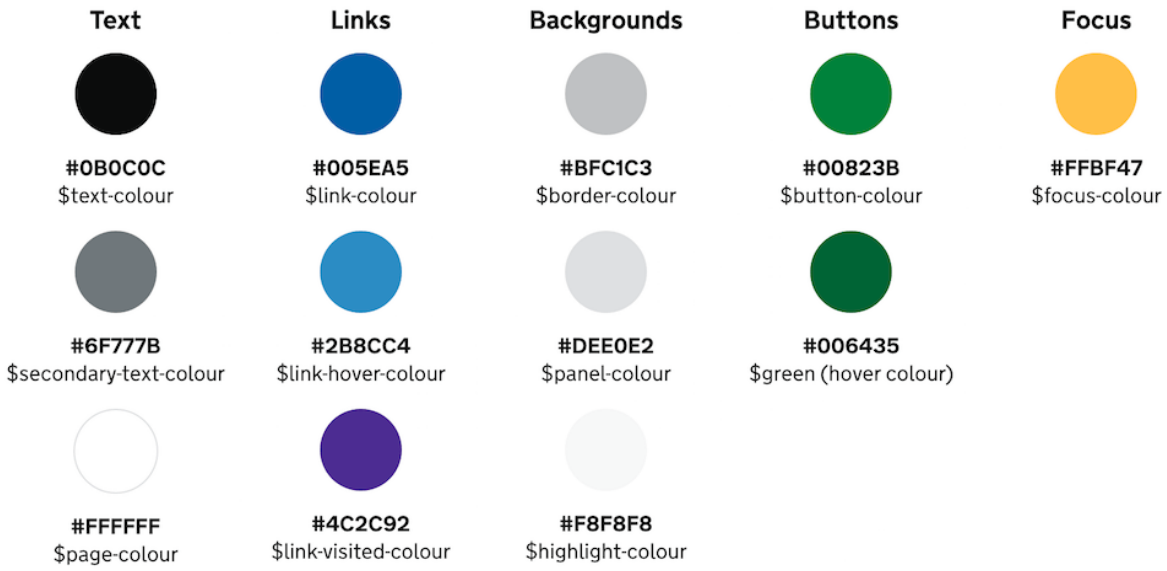
## DOCUMENTING PERCEPTUAL PATTERNS

When documenting perceptual patterns, the focus tends to be on the buildings blocks — color palette, typographic scale, and so on. But as we saw in the previous chapter, it's also important to know how those properties are used and how they work together. Here are a few tips and examples.

### Specify Usage, Not Only the Building Blocks

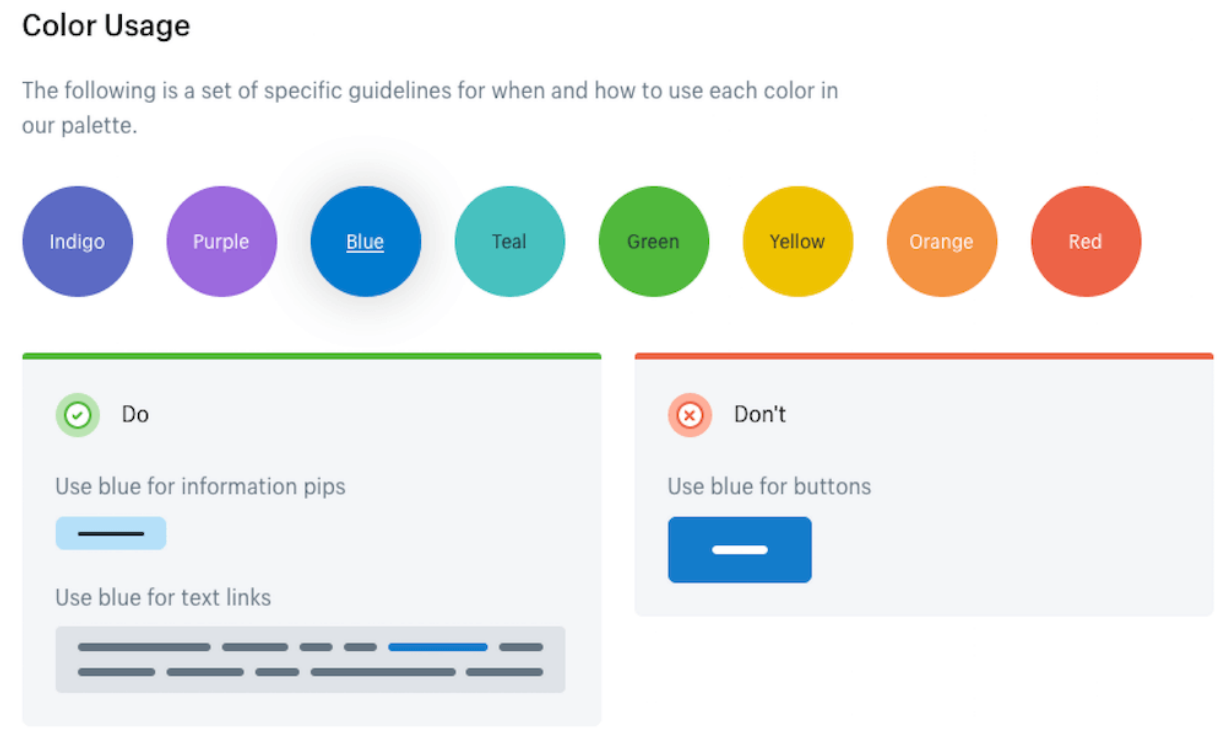
Representation of color doesn't have to be limited to a list of variables. In its color palette, the GOV.UK style guide[27] helpfully specifies the usage of color for text, links,

backgrounds, and other roles.

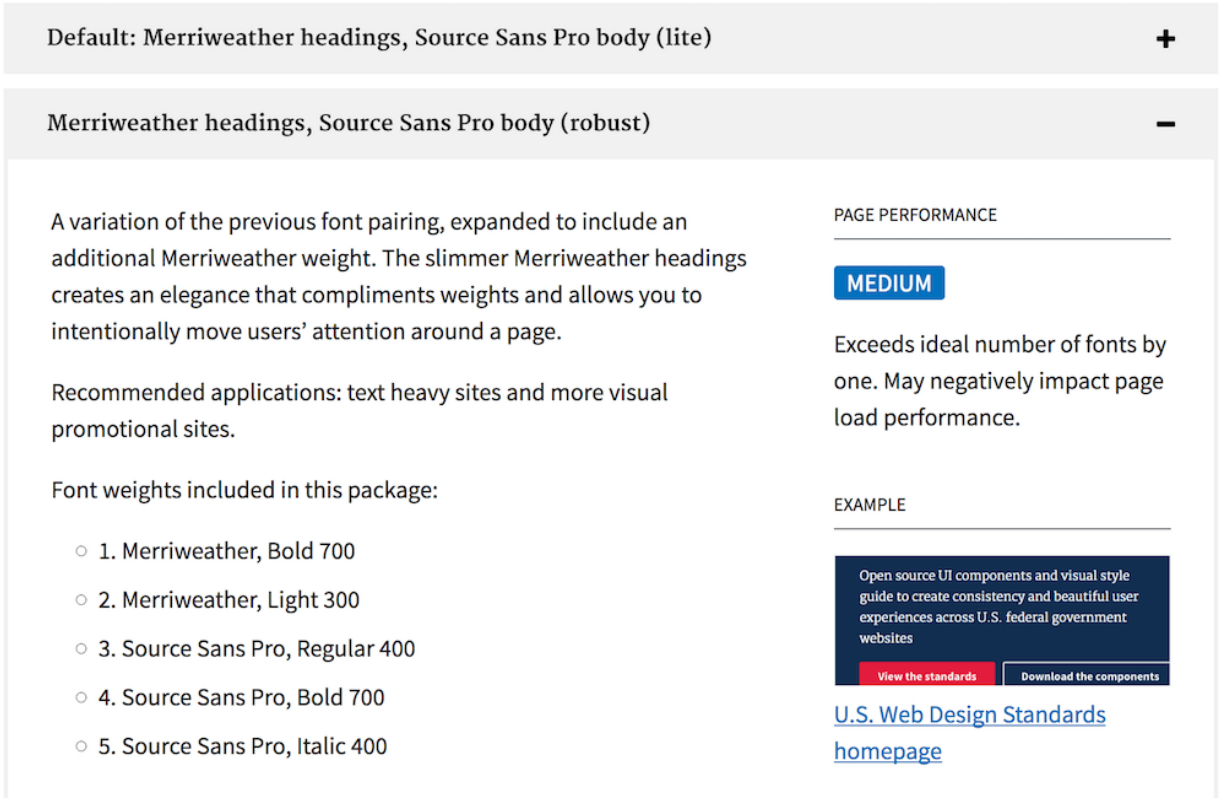



*Color palette shows patterns of usage in the GOV.UK style guide.*

The dos and don’ts format can also be useful, particularly when there’s an expected misuse. In Shopify Polaris[28], both indigo and blue are primary colors used for interactive elements. Stating explicitly that blue shouldn’t be used for buttons is helpful in this case, as it would be reasonable to assume otherwise.




The typography section of the US government's web standards[29] shows type pairings and their recommended usage. Expandable examples demonstrate typographic treatments in context.

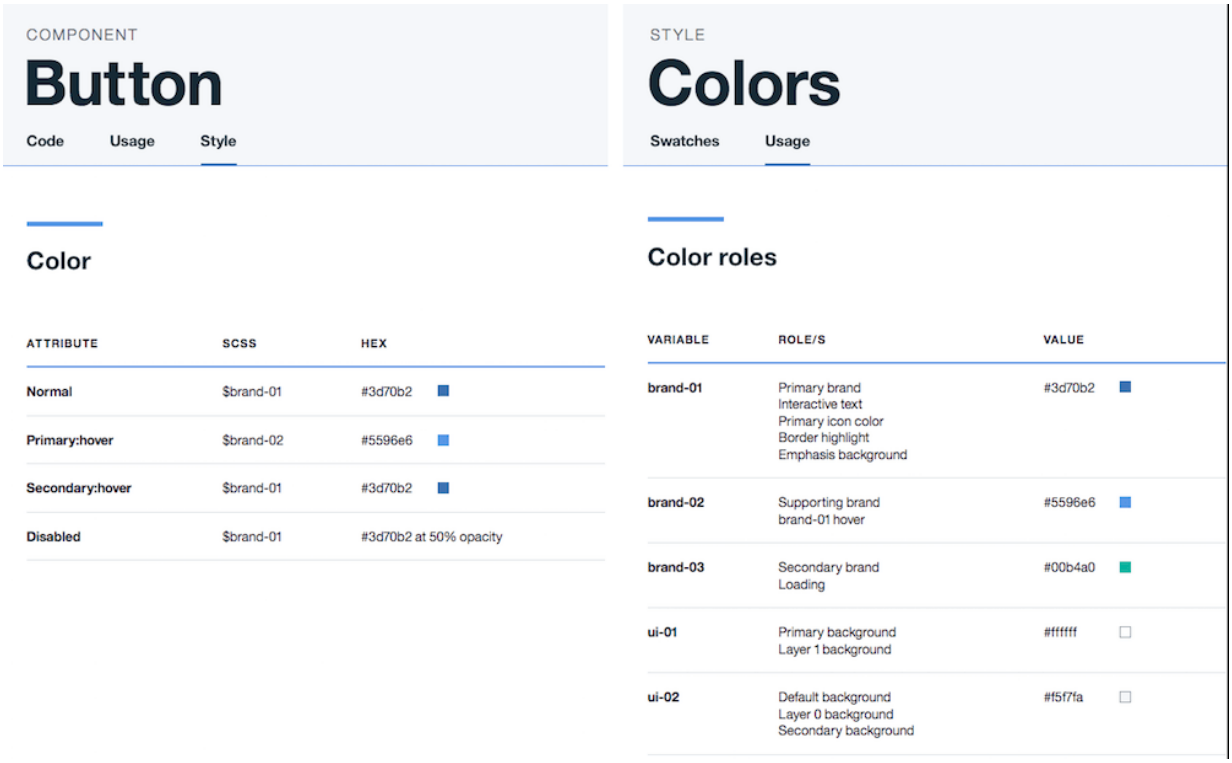



*US government web standards include pairings and their recommended usage.*

## Cross-Reference Styles

Although we separate styles and components to make them easier to work with, in practice they're closely interlinked. Even if there are duplications, referencing styles at the module level, as well as separately, is useful. A button has many styles that define what kind of button it is (color, shape, style of label, transitions, spacing, and so on). At the same time, some of those styles can be applied to other objects — menus, links, toggle controls. Sharing styles is what makes those objects feel like they belong to the same system.

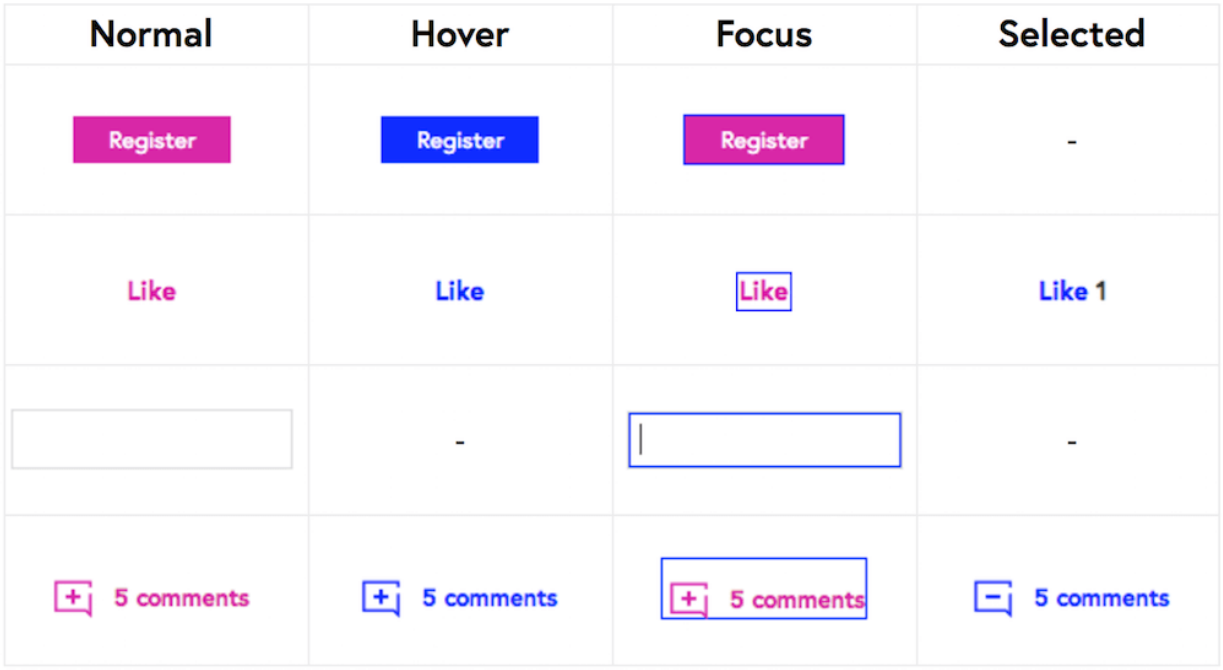
In Carbon, the styles of a specific module, such as color, are shown on a separate tab. The usage of colors is also documented separately.




*In Carbon, colors are referenced at both the module level and all together.*

Take another example: interactive states. Typically, we see them documented only at the module level: here's a button and its hover state. But it is also useful to see all the states together, at a glance. How does the hover state apply to secondary links? To icon buttons? To ghost buttons? To tab controls? Why in some cases does the outline change to solid, and in others the color value changes?

Showing interactive states for FutureLearn in one grid allowed us to define the overarching rules for interactive states and apply them consistently as more interactive elements were added.




*Some of the interactive states in FutureLearn's pattern library[30] are shown together.*

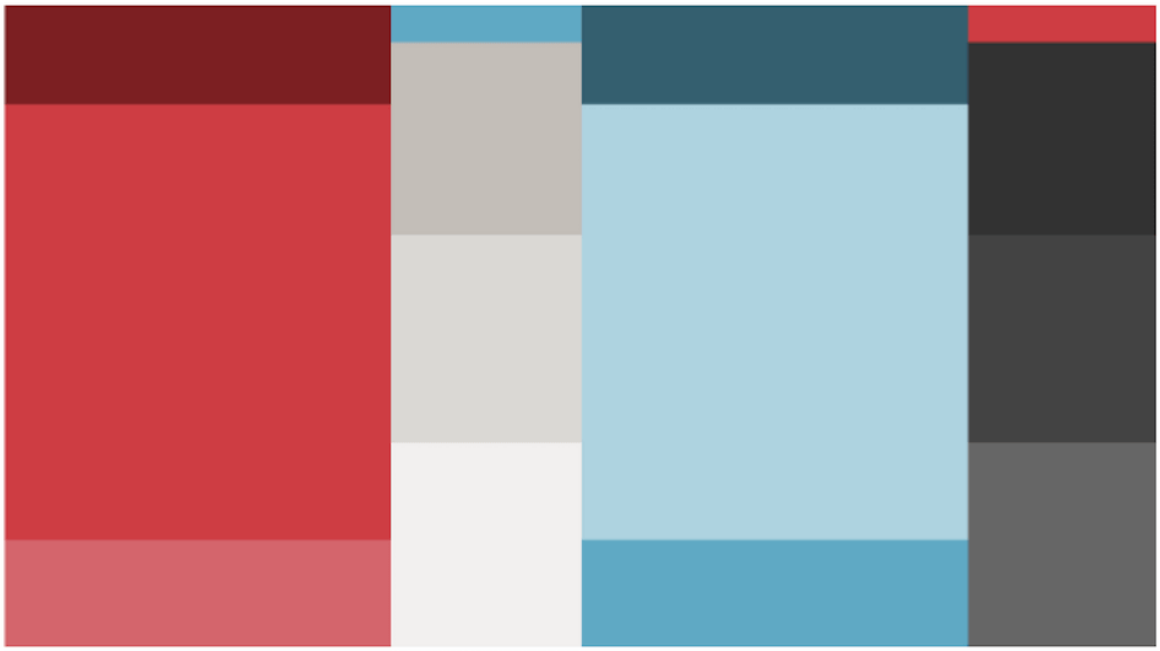
## Show Relationships between the Elements

To be effective, perceptual patterns should interconnect and work together. By showing the relationships (between colors, between typography and spacing, between voice and tone and the visuals), you help make the whole system more connected.

The same color values can have entirely different effects when applied in different proportions. As Michael McWatters noted, with too much or too little red, TED can feel like a different brand. The color chips in the Open Table style guide[31] make the hierarchy of colors clear.

# Color

**DINER** **RESTAURANT**



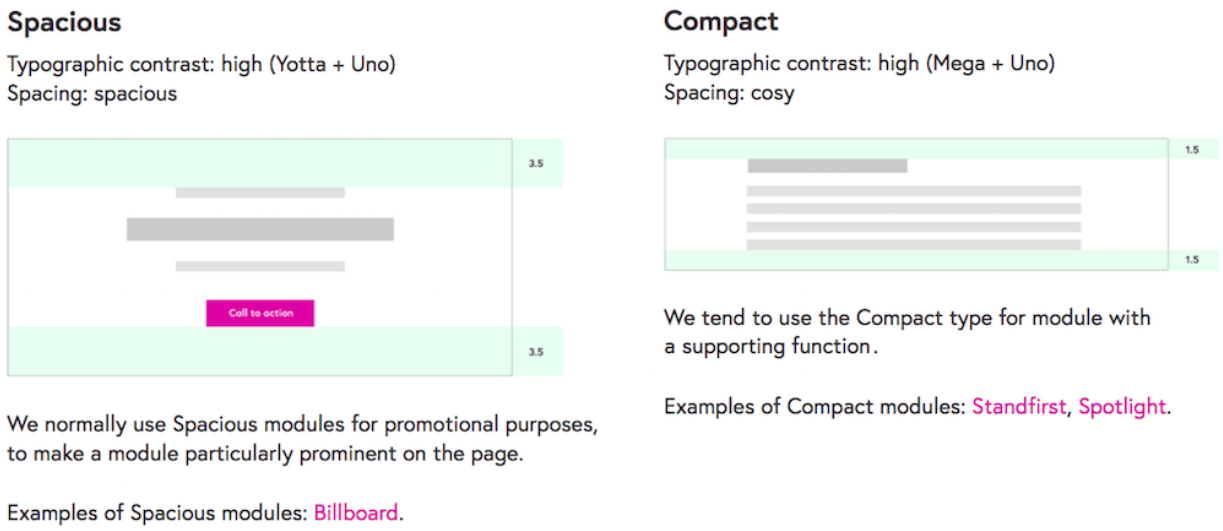
Our brand color palette conveys our passion for delightful dining experiences. The OpenTable color palette is refined and focused to provide consistency for both the diner and restaurant sides of the business.

Typography and spacing are also closely interlinked. Large, higher-contrast typography requires more white space. Smaller text can get lost in the same amount of space if you don't compensate by reducing the padding. Even if you have a limited selection of predefined spacing options (such 8px or 16px units), different designers might have different preferences — some prefer more generous white space, others like it cosier. The values might be consistent, but that doesn't mean that the visual density will be.

To help guide the density and contrast across the product, in FutureLearn's interface we tried to show the relationship between typography and spacing.

- Spacious modules have high typographic contrast (large heading in proportion to the body font size) and generous spacing to balance out the high-contrast typography.

- Regular modules form the majority of sections on FutureLearn. They have the default heading and spacing.
- Compact modules have headings only slightly larger than the body copy.




*Some of the section types for FutureLearn.*

Those settings also reflect the purpose of the modules. High-impact promotional sections benefit from high-contrast typography. On the other hand, modules with a supporting function tend to be more compact.

Finally, in the vast majority of today's pattern libraries, styles are displayed on separate pages. I see this as a limitation. Perhaps the next generation of pattern libraries can show them in more connected ways. Like mood boards or element collages, styles could be presented in a way that shows how they work together, and that highlights signature patterns and the relationships between various elements.

## Workflow

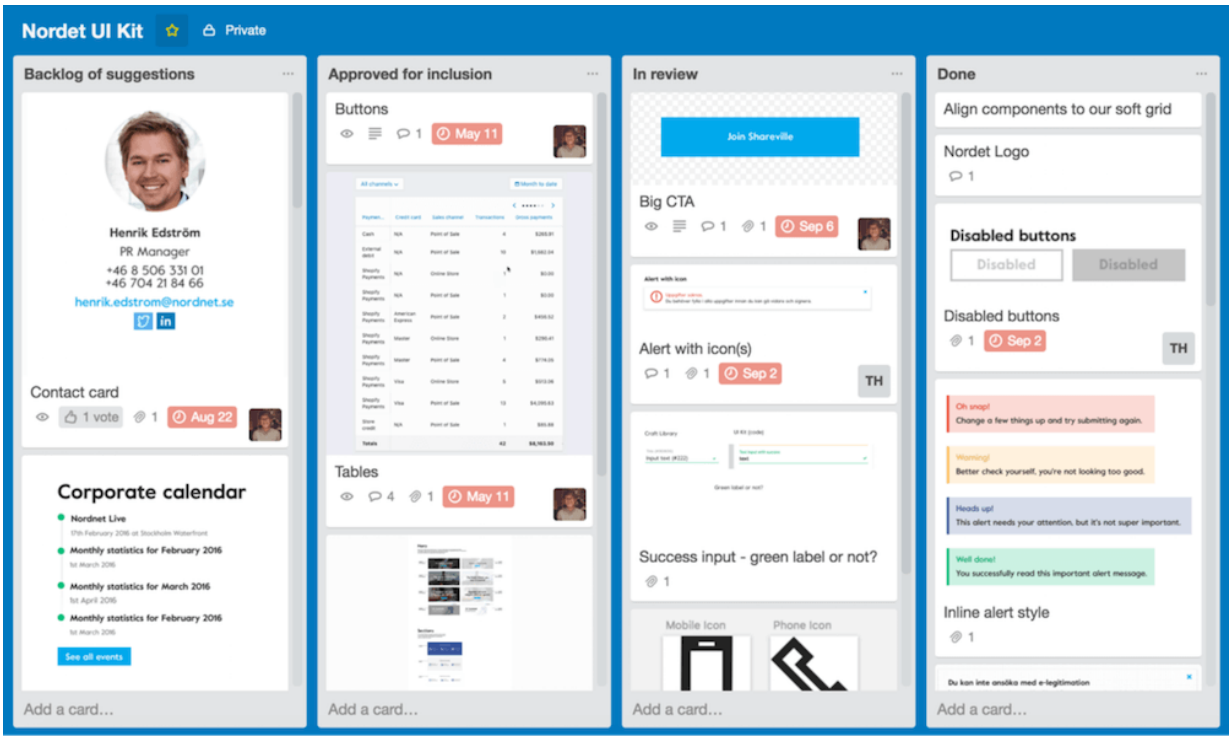
Teams with effective pattern libraries have systematic approaches ingrained in their workflow. How, exactly, varies across companies. Some teams, like Airbnb, have strict and precisely specified processes with powerful tooling. Others are much more informal.

### PROCESS FOR ADDING NEW PATTERNS

One of the foundational aspects to agree on is how the new patterns will be added to the system. The team at Nordnet follows a simple three step process:[32]

1. Submit a design to the UI kit folder on Dropbox.
2. Discuss the inclusion of the pattern as a team.
3. Document any included designs inside the UI kit. Add the new design to the Craft Library which will automatically roll out to the entire team.

The team meet every fortnight to discuss new submissions. They go through a Trello backlog and decide if a module should be approved for inclusion or archived.




A similar workflow is adopted by the teams at Shyp[33] (using GitHub for adding and reviewing the patterns), FutureLearn, and many others. The process doesn't have to be strict but it's important to have something in place that enforces, in some way, a regular review of patterns.

To make sure the format of submissions is consistent, some teams find it useful to have a standard template with simple guidelines, such as name, description, author, and date. At FutureLearn, submissions come directly to the pattern library rather than the master design file, and there is an informal guide for writing a description for a

pattern. It consists of three questions: What is it? What is it for? How does it achieve its purpose?

## CRITERIA FOR ADDING NEW PATTERNS

A common problem teams have is a lack of agreement on what constitutes a design pattern. A way to manage this is to have shared criteria for adding (and also updating and removing) patterns.

The two most common approaches are:

- Every new element on the site is also automatically added to the pattern library. This works if you're strict with accepting patterns to the system. There should be a process which checks if a similar pattern exists already, or if an existing one can be modified (such as regular review of new patterns as a team). Without those processes, the risk is ending up with duplicated patterns.
- Elements are added only when they're *reused*. Some teams add modules only on the second, or even third use. The idea is that an element has to prove itself as a pattern before being added to the system, which helps to keep the pattern library lean. With this approach it's important to have visibility of everything being created and effective communication across teams. A log of undocumented patterns should also be kept, so the team has full visibility of what's available, even if it's not in the pattern library.

It's also possible to base your decision on *potential* reuse. At FutureLearn, the specificity of a component's purpose is used as a criterion. If an element is defined in a generic way, it is more likely to be reused in the future. In this case, it is added to the pattern library. If a new component has a specific purpose (such as a seasonal promo, a module related to a specific event, and so on), it can be treated as a one-off.

When following this rule, the whole team should take care in how they define components and not make something specific unless absolutely necessary. If someone introduces a one-off, they should share it and explain why it is specific. Occasionally someone else will find the module useful for their needs. In this situation, we'd redefine the pattern as more generic and add it to the library.

## PEOPLE AND RESPONSIBILITIES

Another aspect to consider is the practicalities of updating documentation, particularly if there's no dedicated team.

If contributions come from everyone, you have to be strict at making sure they're added to the library. For instance, adding a component can be a part of the story for creating it. The designer and developer who create a pattern are responsible for adding it to the pattern library. As we saw in chapter 6, this model doesn't work for every team. Sometimes you need a person or group of people responsible for curating and maintaining the pattern library, even if everyone contributes to it.

If there's a dedicated design systems team, it's important to agree on their role, as well as the process for managing contributions. A systems team can have the role of a curator or a producer, and many companies use a combination of both.

- **Curator.** Contributions for new patterns come from all over the organization. The systems team defines the ways in which internal teams contribute, including setting requirements and the review process. If a submitted pattern doesn't meet the standards, the team encourages the designers and developers who created it to change it, rather than making the change themselves. The team at Atlassian follow this model.
- **Producer.** With this approach the design systems team creates the majority of patterns. They'd typically work closely with the product designers in different teams and hold open meetings where others can ask questions, give feedback, or propose missing patterns.The systems team accept submissions from across the company, but they have the final say over what is included, adjusted or removed. Airbnb uses this approach.

When choosing a direction, consider your organizational structure, team culture, and specific product needs. The curator role is usually suited to distributed teams with looser system structures, whereas producers are more common in stricter and more centralized systems.

In both cases it's important that the systems team are seen as *partners*, rather than police.

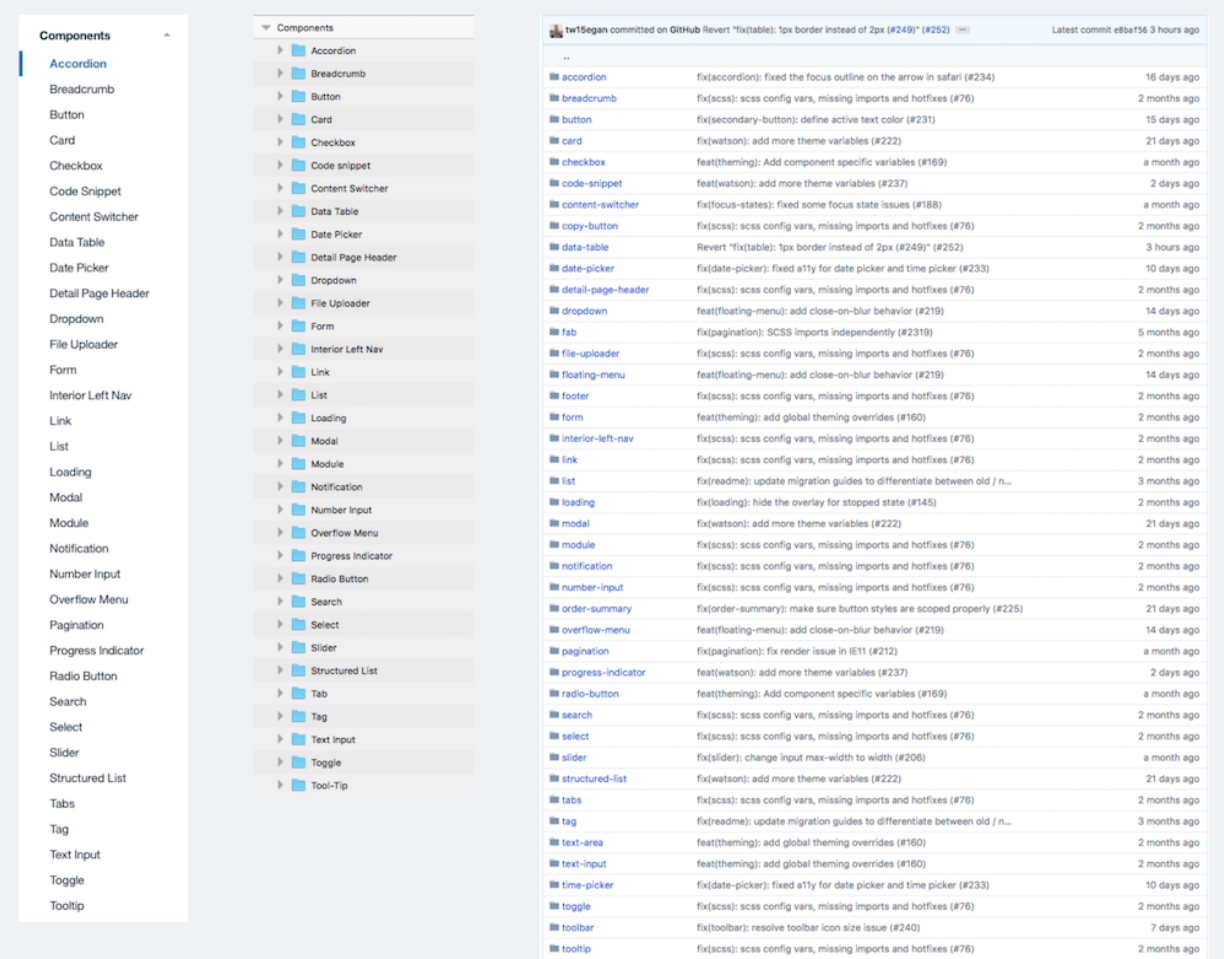
*“We want to collaborate with teams as early as possible when they’re thinking about developing new patterns and components. Our relationship with product teams should be a partnership, rather than a situation where someone goes away and does a bunch of work and then we either approve or veto it. If that happens, we’re not doing our jobs very well.”*
*— Amy Thibodeau, UX lead, Shopify*

---

## ALIGNING FACETS OF THE SYSTEM

Code, design and the pattern library are facets of the same system. Treating it that way makes it more robust, because the system is supported from multiple angles. This doesn’t necessarily mean that the patterns must be fully synchronized. What’s important is that the team practises the same *approach* across the facets — to naming, structure and understanding of the purpose.

The Carbon design team tries to be as consistent as possible across their Sketch design kit, component library and the code.




*In the Carbon design system, names and folder structure are consistent across the three facets of the system.*

Designers at Nordnet use atomic design to organize folders in their Sketch kit. They even follow BEM naming conventions for their design files, to help developers and designers talk the same language.[34]

When design and code are aligned conceptually, synchronization between them is easier to achieve. It's also easier to find the tools that fit with your workflow.

## Tools

Keeping the pattern library in sync with production code is one of the main challenges. Teams use different approaches — from manual copy-and-paste, to making a pattern library part of the production environment (Lonely Planet's Rizzo is an example of the latter). Many tools help to achieve it. Here are some of the most popular ones.

## KEEPING THE PATTERN LIBRARY UP TO DATE

Some of the easiest to implement are CSS documentation parsing tools, such as KSS[35]. They all work in a similar way: comments in the CSS provide a description (which is pulled into the documentation programatically); running a script generates the markup for the pattern library. Parsing tools are relatively simple but limited in functionality. They can also lead to duplicate markup, which can make maintenance more time-consuming.

Among more powerful tools are style guide generators, such as Pattern Lab[36] by Brad Frost, Dave Olsen and Brian Muenzenmeyer. Pattern Lab comes with many useful features, such as responsive preview and support for multiple languages. It's predominantly suited to larger sites with multiple templates, particularly those that practice atomic design methodology.

Fractal[37] by Mark Perkins is one of the more lightweight and flexible tools which is gaining popularity. Fractal helps build and document a pattern library, and integrate it into your project. One of its main advantages is that it's flexible and unopinionated — Fractal can work with any templating language and organizational structure.

Full synchronization between the pattern library and the code is extremely difficult to achieve, and companies manage it with varying degrees of success. The ways teams prioritize synchronization also varies:

---

*"It's always slightly off-sync. If it's too perfect, it's not going to work. Our design language, as any language, is constantly evolving. We change details and patterns and we add patterns. We constantly build products. So at any given time there are many versions of the design language. We embrace this fact and design a system which can deal with these imperfections."*
*— Jürgen Spangl, head of design, Atlassian*

---

In stricter systems and centralized organization it is more important, whereas companies with looser structures are more tolerant to having it out of sync.

## KEEPING MASTER DESIGN FILES UP TO DATE

Designers practicing a systematic approach currently tend to use Sketch[38] as their main tool (largely thanks to the features such as text styles, symbols and artboards, which seem to be well suited for a design system workflow). Teams typically have a master file which contains a UI kit with some or all of the core components and styles. Product designers tend to work from their own files, pulling elements from the master as needed.

The challenge is making sure that the master kit always has the latest patterns. There are many tools to help achieve that — from the lightweight to more comprehensive solutions.

Abstract[39] is a version-controlled hub for your design files. You can create branches, commit explorations, and merge changes. Abstract makes it easier to keep one single source of truth for your design files, including a master UI kit. Another popular tool is Invision's Craft[40]. Craft is a set of plug-ins for Sketch which syncs the UI kit to anyone who has the plug-in installed. A Craft library can be saved on Dropbox.

More comprehensive options include UXPin[41], Brand.ai[42] and Lingo[43]. These tools allow you to create and manage a pattern library without having to use code. Naturally, they don't provide as much flexibility as a custom-built pattern library, but many of them have useful features, such as interactivity of components, Sketch plug-ins[44] for keeping files up to date, integration with Slack that pings a channel when the library is updated, and more.

## PATTERN LIBRARY AS THE SOURCE OF TRUTH

With pattern libraries gaining the "source of truth" status, in some companies it has become somewhat less important to keep master UI kits perfectly up to date. At FutureLearn, the master Sketch file (updated and shared via GitHub) only contains the core granular elements that don't tend to change (typography, buttons, navigation, and so on). Designers use the pattern library as the main reference for up-to-date patterns, Sketch or Photoshop are used mainly for exploratory work. Because the majority of the components are defined and named, more and more often the team can get by with paper sketches, without needing detailed design specs.

Thanks to design systems and pattern libraries, design and engineering workflows are moving toward each other. There are a lot of experiments in this area[45], such as tools for generating Sketch files directly from a web page, and importing real data. In the near future we may not have to worry about keeping UI kits in sync, as they could be generated any time from the pattern library.

## The Future Of Pattern Libraries

Tools should accommodate the natural workflow of the whole team. Only then will everyone take ownership, and contributions to the pattern library will be evenly distributed. FutureLearn's library didn't have the capability for designers to update descriptions of modules, which in some way reduced their responsibility. Front-end developers were under more pressure to keep the documentation updated, which at times felt like a burden. In the future I hope to see pattern libraries accommodate multidisciplinary workflow. They could become environments where all disciplines could contribute to discussions around design patterns and help define their purpose.

With tools getting better, pattern libraries and a systematic approach to design will continue affecting designers and developers deeply. Many teams are seeing the changes already. Something that used to take days of manual work can be done in minutes — no more detailed design specs, no more reinventing the same solutions again and again. At first, this might seem threatening (Will we have jobs in years to come? Does it take away from the creativity and craftsmanship on the web?). But perhaps the opposite is the case. Design systems free our time and energy to solve bigger and more meaningful problems, like understanding our users better and making design languages more inclusive.

—

1. There are many other types of design documentation, such as brand identity documents, front-end style guides for clients, and so on. In this chapter we're only talking about pattern libraries created by in-house teams to support a design system.
2. https://docs.google.com/
3. https://www.wework.com/
4. http://smashed.by/plasmads
5. http://smashed.by/atlassianpl

6. http://smashed.by/bbcgel
7. http://carbondesignsystem.com/
8. http://smashed.by/rizzopl
9. http://smashed.by/marvelpl
10. http://smashed.by/officefabricpl
11. https://www.lightningdesignsystem.com/
12. https://polaris.shopify.com/
13. http://smashed.by/skycomponents
14. https://style.eurostar.com/
15. http://smashed.by/atomicdesign
16. http://smashed.by/ceasefire
17. http://smashed.by/clearfractal
18. http://smashed.by/predix
19. https://medium.com/@lewisplushumphreys
20. http://smashed.by/plasmads
21. https://www.futurelearn.com/pattern-library
22. From email correspondence with Amy Thibodeau, UX lead at Shopify, August 2017
23. https://design.sipgateteam.de/
24. http://smashed.by/skytoolkit
25. http://smashed.by/polarisnav
26. http://smashed.by/pattern2doc
27. http://smashed.by/govuk
28. http://smashed.by/polariscolor
29. http://smashed.by/usgov
30. http://smashed.by/flstates
31. http://smashed.by/opentable
32. See “Super easy Atomic Design documentation with Sketch app” by Ross Malpass.
33. http://smashed.by/shyp
34. See “An Atomic workflow for design & development at Nordnet” by Ross Malpass.
35. http://warpspire.com/kss/

36. http://patternlab.io/
37. http://fractal.build/
38. https://www.sketchapp.com/
39. https://www.goabstract.com/
40. http://smashed.by/craft
41. http://smashed.by/uxpin
42. https://brand.ai/
43. https://www.lingoapp.com/
44. https://www.lingoapp.com/sketch/
45. http://smashed.by/airbnbds

# Conclusion

In programming and design, Christopher Alexander's pattern discipline has become one of the most widely applied and important ideas. It is now influencing how many of us think about design systems. But there's an essential feature we may still be missing from Alexander's original idea: the moral imperative to create systems that make a positive difference to human lives.

In his keynote speech for OOPSLA[1] in 1996, Alexander emphasized that at the root of the architectural pattern language there's a fundamental question: will these patterns make human life better as a result of their injection into the system? Even if not all of them will have that capacity, there should be at least a *constant effort* to achieve it.[2]

Many aspects of our lives can now be managed online, from buying groceries and paying bills, to finding a date or completing a degree. As architects of design systems, we play an important part in shaping the digital world. Pattern language gave us a format for thinking about design — and it also gave us a challenge: do the patterns we create have a positive impact on human life? How do we know if they do? How do we continuously test that?

It is hard to carefully consider these questions, when you are given a task to optimize the number of clicks, or encourage people to spend more time on a site. Even with the best intentions, a lot of what we create on the web is designed for short-term commercial benefit, rather than bringing real value to everyday lives: patterns designed

to get users hooked[3], patterns biased towards some population groups, patterns that encourage people to spend time and money in ways they might regret later.

On the other hand, we don't always consider, for instance, what happens to all our digital accounts and information when someone passes away, how the designs we create improve someone's quality of life, or how inclusive and empathetic our systems really are.

The pattern language for the web we're creating is powerful. It has the capacity to influence not only the digital world, but the physical one. We owe it to ourselves, and the people who use our products, to constantly consider and challenge the shape this language takes, and be thoughtful in what we contribute to it.

## Further Reading And Resources

These three books are a great source of knowledge and inspiration for anyone interested in design systems. I kept referring to them again and again while writing this book.

- *The Timeless Way of Building*[4] by Christopher Alexander
- *Thinking in Systems: A Primer*[5] by Donella Meadows
- *How Buildings Learn: What Happens After They're Built*[6] by Stewart Brand

See also:

- *How to Make Sense of Any Mess*[7] by Abby Covert
- *Front-end Style Guides*[8] by Anna Debenham
- *Atomic Design*[9] by Brad Frost
- *Responsive Design: Patterns and Principles*[10] by Ethan Marcotte
- *Inclusive Design Patterns*[11] by Heydon Pickering

## OTHER RESOURCES

- Design Systems Slack Team[12], created by Jina Anne
- Design system articles[13] by Nathan Curtis
- Style Guide Podcast[14], hosted by Anna Debenham and Brad Frost
- Design Systems Newsletter[15], curated by Stuart Robson
- Responsive Web Design Podcast[16], hosted by Karen McGrane and Ethan Marcotte
- Website Style Guide Resources[17]

Thank you for reading. This book is really just the beginning of a conversation about design systems. I'm keen to continue it beyond the book, and would be happy if you'd email me at alla@craftui.com with your thoughts and stories. ❧

—

1. ACM Conference on Object-Oriented Programs, Systems, Languages and Applications.
2. See "Patterns in Architecture" by Christopher Alexander.

3. See “How Technology is Hijacking Your Mind — from a Magician and Google Design Ethicist” by Tristan Harris.
4. http://smashed.by/timeless
5. http://smashed.by/thinksys
6. http://smashed.by/howlearn
7. http://smashed.by/makesense
8. http://smashed.by/fesg
9. http://smashed.by/atomic
10. http://smashed.by/rwdpatterns
11. http://smashed.by/inclusivedesignpatterns
12. http://smashed.by/dslack
13. http://smashed.by/nathan
14. http://styleguides.io/podcast/
15. http://smashed.by/dsnl
16. http://smashed.by/rwdpc
17. http://styleguides.io/

# Copyright Information

## Image Credits

Chapter 1, Visual Loudness Guide:
http://smashed.by/visualloudness

Chapter 1, NASA's Graphics Standards Manual:
http://smashed.by/nasastandards

Chapter 1, Whitney Museum of American Arts:
https://www.experimentaljetset.nl/

Chapter 4, The Washington Examiner 2012 Campaign Site:
Samantha Warren, http://styletil.es/

Chapter 4, Element collage for RIF: Dan Mall, http://danmall.me/

Chapter 5, Minions and Gru: Despicable Me, ©Universal Pictures

Chapter 6, Puma City, LOT-EK: "The Box" by Sibyl Kramer

Chapter 6, Greendo, Keita Nagata: http://www.designboom.com

Chapter 6, Basket Apartments, OFIS architects:
http://www.archdaily.com and http://www.ofis-a.si/