Heart Internet Presents

DESIGN SYSTEMS

Build better products that scale

Heart Internet

An ebook version of this book is available at:

https://www.heartinternet.uk/blog/design-systems-free-ebook-download/

Contents

The editor

Oliver Lindberg

Oliver Lindberg is an independent editor, content consultant and founder of new web events series Pixel Pioneers (https://pixelpioneers.co), based in Bath, England. Formerly the editor of .net magazine, he's been involved with the web design and development industry for more than a decade and helps businesses across the world create content that connects with their customers.

FOREWORD

Design systems are one of the hottest topics in web design and development right now, and for a very good reason: they can speed up the design process, improve collaboration between teams, create a cohesive user experience and ultimately help develop better digital products.

Brands all around the world are currently figuring out how to build design systems, but the first steps can be challenging, especially if you are not starting from scratch but have to deal with the realities of multiple websites, multiple platforms and legacy code.

In this book, we explore the language of design systems and how naming the system itself as well as its components is fundamental to its success. We then move onto case studies and discover how two very different companies – Amnesty International and Auto Trader – created and implemented their design systems. Finally, we learn how to maintain a design system and encourage usage and participation to make it flourish.

All of our authors have been instrumental in the creation of design systems themselves and thus they're able to provide inclusive insights and practical, actionable advice. Follow their tips and techniques and you're well on your way to building a successful design system for your own organisation.

The author

Micah Godbolt

Design systems advocate, speaker, trainer, and writer Micah Godbolt (twitter.com/micahgodbolt) is the program manager for the Fabric design system at Microsoft. He is passionate about seeing design systems empower designers, developers and managers to take ownership in their sites and applications. He blogs at micahgodbolt.com, where you can find links to his book, *Frontend Architecture for Design Systems* (fea.pub), workshops and conference talks.

THE LANGUAGE OF DESIGN SYSTEMS

Design systems have been around almost as long as the web. Twenty years ago Frog Design **[1]** helped launch an incredibly successful design system that powered ecommerce website Dell.com. Since then, design systems have been embraced by nearly every single major player on the web, from Google's Material Design Language **[2]**, to Salesforce's Lightning Design System **[3]** to Microsoft's UI Fabric **[4]**, which I currently have the pleasure to work on. Yet despite this widespread adoption of design system-powered websites and applications, it can still be a difficult task to explain to your boss why your company should build one. I believe that in order to make an appropriate appeal for a design system, it's important to understand, in no uncertain terms, what a design system is.

When I was preparing for my first talk about design systems, I took an early stab at trying to write an appropriate definition... in 140 characters nonetheless. I said:

A design system is a set of rules and assets that define how to express everything a visual language needs to say. **[5]**

To this day, I stand by that definition, even as my understanding of design systems has evolved, so let me tell you why. It starts by diving into the way in which we study language.

The scientific study of language

If we understand that the purpose of a website or web application is rooted in communication, and that a visual language is defined as "a system of communication using visual elements" **[6]**, then, in order to properly communicate with a visual language, we first need to study it!

The study of language, or linguistics, is a several hundred year old scientific practice that evaluates the form, meaning and context of a language. Through this evaluation process, we can break down a sentence into its smallest possible pieces, while gaining an understanding of how to put it back together. Let's analyse the following sentence and use a few areas of linguistics to identify its form, meaning and content.

"This website loads quickly".

Syntax

The syntax gives us an insight into how a sentence is formed and how one section in a sentence conveys meaning onto another. Our sentence above can be broken up into two clauses. "This website" is the noun clause, or subject. By understanding the syntax of the English language we can determine that the following "loads quickly" is a predicate, or verb clause, that describes what the website is doing.

Morphology

To better understand the meaning of words, especially those composed of many smaller words, or morphemes, we can turn to morphology **[7]**. This study of words, and how they are formed helps us to evaluate the meaning of words, and the rules we can use to create new ones. Looking at the word "website" we can break it

down into two distinct words, each with their own meaning, that, when combined, create a brand new word. "Web" can be used as an adjective to describe things related to the internet like "world wide web", "webpack" or "web browser". "Site" is a noun that usually refers to some location, or area, like a "dig site" or "launch site". Morphology is the area of study that helps us understand how these different types of words can be combined together to create new parts of the language.

Phonology

In our journey to break the sentence "this website loads quickly" into the smallest pieces possible, the last part of our deconstruction introduces us to the study of phonology, that is, the study of the individual sounds that make up a language. The reason we break words down past their message and into the sound that produces them is that a spoken language must be transferred from one person to another, and if sounds are lost or modified during that transmission, it can dramatically affect the message.

Phonology breaks each syllable down into two parts, the onset and the rhyme. Looking at the syllable "web" we can determine that it's made up of an onset "w" sound, followed by a rhyme of "eb". With an understanding of these phonemes we can describe how our mouths produce each sound, how our ears hear them, and how we can communicate our message with the greatest possible impact.

The scientific study of visual language

Using the same three techniques I described above, we can perform a similar study of a visual language. So let's see how we can break down a typical webpage.

Syntax

Entire pages can be broken down into smaller pieces, and the syntax describes how those various pieces relate. Just like in our verbal language example we employ a syntactical analysis to understand how a modal dialog conveys meaning on form, or a sidebar has a meaningful connection with the selected item in a list.

Morphology

Looking at individual compound words, or in a website's case, compound UI, we can see that the combination of a form field with a button creates a piece of a user interface greater than the sum of the two parts. By defining these various patterns, we increase the capacity of our language to communicate complex topics in concise ways.

Phonology

We study the phonology of a verbal language, so that we can understand how the language is conveyed to ensure that we can communicate in the most efficient way possible. In the same way, we work to understand the smallest units of our visual language, so that we can be confident that our message is properly reaching its audience.

When conveying messages with our voice, phonology breaks each syllable into two parts, the onset and the rhyme, but with a visual language there are many parts of the language to think about:

1. **Space**: The various margins, padding and white space that impact usability and information density.

2. **Typography**: The number, classification and weights of the fonts used on the page. Combinations of different types can solicit different impressions of the content being conveyed.

3. **Iconography**: Our visual languages include more than just text on backgrounds. We use icons to supplement, or even replace textual information, which allows us to increase information density without increasing clutter.

4. **Colour**: The colour of a visual language is the heart and soul of the language. It conveys a great deal of information about the personality of the message. It can be warm and friendly, cold and nothing but business, or hot and aggressive. It's amazing what a new coat of paint can do to a visual language.

5. **Animation:** A website is not a poster. It's not something to sit back and read, but something to interact with. Animation is the key to creating a cohesive interactive experience with your visual language.

The conclusion of our study

Once we have studied a visual language and evaluated its form, meaning and context, we can then start to create meaningful guidance and documentation for the language, and how to use it. With spoken languages these artifacts are "grammar" and "vocabulary".

Grammar: "the set of structural rules governing the composition of clauses, phrases, and words in any given natural language"

Vocabulary: "a set of familiar words within a person's language"

You might not be using the terms grammar and vocabulary in your

own design system, but you'll certainly be familiar with two other popular artifacts, the **style guide** and **pattern library**.

A style guide is the grammar of a visual language. It defines the structural rules that govern the composition of various pieces of your visual language, including everything from fonts, to colours, to common patterns and usage guidance. A style guide is your visual language distilled down into a set of rules that describe how your language communicates with its intended audience.

A world of grammatical rules is little use without the vocabulary that was created based on those rules. Our vocabulary helps us to communicate our thoughts, ideas and instructions in an efficient way. If you've ever learned a new language and are having trouble communicating clearly because you can't remember the right word, that's a lack of vocabulary. In the same way our websites struggle when we don't have a "word" to properly communicate what we're trying to say. These "assets" are also essential in allowing our language to communicate effectively.

Looking back at my earlier definition of a design system, you can see why I've stood by its accuracy for so long. Grammar, or the style guide, is the set of rules that defines how our vocabulary, or pattern library, is created. It's those assets, and the rules that created them, that allow us to "express everything our visual language needs to say".

Resources

[1] Frog Design
Dell.com Redesign Project
https://www.frogdesign.de/work/dellcom.html

[2] Google Material
https://material.io/

[3] Salesforce Lightning Design System
https://www.lightningdesignsystem.com/

[4] Microsoft UI Fabric
https://developer.microsoft.com/en-us/fabric

[5] Micah Godbolt, Twitter
https://twitter.com/micahgodbolt/status/717565115919237120

[6] Visual Language, Wikipedia
https://en.wikipedia.org/wiki/Visual_language

[7] Morphology (Linguistics), Wikipedia
https://en.wikipedia.org/wiki/Morphology_(linguistics)

The author

Charlotte Jackson

Charlotte (www.lottejackson.com) is a Brightonian expat living in Sydney, Australia. She is currently the product manager on the design ops team at Ansarada, and was previously a front-end developer at Clearleft in Brighton. For several years her work has focused on building component libraries and helping teams build their own. She co-organises a meetup in Sydney called codebar (www.codebar.io) to help improve diversity in the tech industry.

THE RESPONSIBILITY OF NAMING

Many terms have been given to collections of UI elements: component libraries, pattern libraries, pattern portfolios, front-end style guides. They all mean the same thing, and they all describe libraries of tangible outputs, which can be used in many different combinations. How these items are used and combined to form screens is defined by a system. The main benefit of building a component library isn't the components and patterns themselves – of course they are the reason we do it – but it's the language used to name and organise them. That's what turns a library into a system.

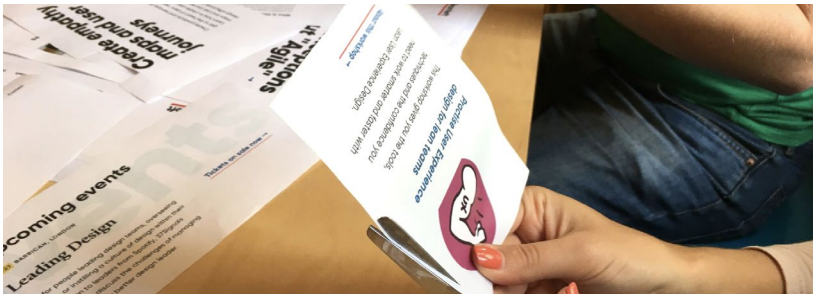
Whether we intend it or not, design systems come with a language – from the name given to the system itself down to the most granular components. Everyone in the organisation talks about the elements encompassed by the system. Designers create and define components and patterns. Developers build them. Product managers prioritise design and development. Client services representatives provide support to customers who interact with components on screens. Marketing share the components and design language. Stakeholders review designs. Without strong documentation for a system, users find another way to navigate and infer a language themselves. Then what happens when one teams refers to a component with a completely different name to another team? Time can be wasted on confusion and making changes to components that were not intended.

Creating a common language

A successful design system is understood by the whole team. Everyone follows the same approach to design and front-end architecture, and the same principles. They know the components and patterns, and how they work. There is a common language between everyone.

Creating a common language requires collaboration and stepping away from the screen. When people are involved in the creation of something, they are more likely to remember and use it. An exercise that I have found really useful at the start of design system projects is the Pages to Patterns exercise **[1]**. The key people to involve in the exercise are the users and creators of the design system. However, everybody who is involved, or has an interest in the design system, can take part; not just developers and designers. The more perspectives and diversity, the better. There's no coding involved, just good old paper and scissors.

In the exercise we print out screen designs, cut them up into components and group them together. Once we have identified components, we start naming them.



The Pages to Patterns exercise for the redesign of Clearleft's website

Here's how it works:

- As a group, we choose a component to work with.

- Everyone in the group writes a suggested name on a post-it note and keeps it covered until everybody is ready. Only spend a couple of minutes thinking about the name because often it's the obvious choice that makes the most sense and is easiest to remember.

- Once everyone has written a name, reveal them. In turn explain the name and why it was chosen. Names that appear multiple times are usually good candidates because they indicate a shared understanding of the component.

- Repeat for each component.

The post-it notes for components

What is the right name?

As many of us know too well, naming is difficult, but choosing the right names is key to building a robust system. A component's name defines its reusability and length of life, so it's important to get it right. For example a "product card", can only be used in a product context. What if it's needed in another context, like a contact form? Then it no longer works. Yet, the name "card" can be reused anywhere, and it's far more likely to survive the test of time. Components may change their look, feel and placements, but their purpose and function are likely to stay the same. Asking a designer why they made a component look the way it does can help identify its purpose.

The right names can vary between teams and organisations. Names that mean something to the team can be great candidates. At Clearleft, I worked on a component library for an organisation called BikeRegister. We followed the cycling theme for classnames. They included names like: frame, tandem, pump, ride, ride-race. It was fun to create, and the client was happy with it.

Here's another example: for an online clothing store, you might want to use a clothes sizing range to represent the sizing of components on the website. Dresses have values such as mini, midi and maxi, which represent small, medium and large lengths like this:

The three types of dress lengths

Using a language that means something to the team can encourage adoption. It's memorable and enjoyable to work with.

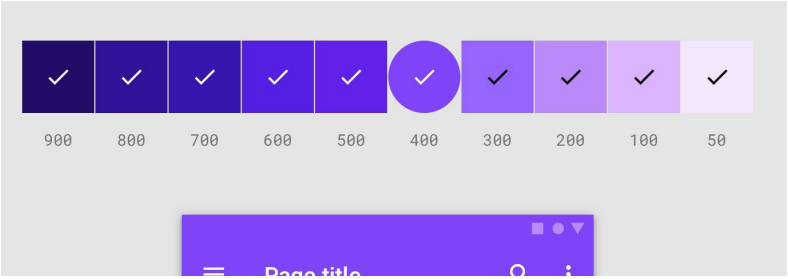
At Ansarada, we chose to name components after their functions because it's easy for people in the organisation to understand what they do. It helps developers quickly understand what they mean to users. For example: Expander, Date Picker, Message.

Naming scales are useful when you have many variations of an element, such as colours, type and spacing. Mandy Michael, front-end development manager at Seven West Media, shared a naming scale for the range of grey colours in their design system during her talk at Web Directions in 2016:

Mandy Michael's grey naming scale

This is a fun way to document colours. It offers flexibility to grow because the names are not connected to each other. So it'd be easy to add another grey and call it 'rabbit', for example. On the other hand, the lack of connection between names makes it more difficult to remember which name stands for which colour.

Here's another example by Seven West Media, an approach that we use at Ansarada as well:



Another Naming scale by Seven West Media, Perth

This is more limiting on scalability, but it's easier to understand which colour goes with which number without looking at a chart. I find this approach quicker to work with.

Whatever the choice made by the organisation, it's worth creating a set of guiding principles to facilitate naming decisions in future. People in the team may change, so it's important that the language is learnable and that consistency is maintained.

Once you have fleshed out some component names, share them with others in the organisation. See if names make sense to others. Run the naming exercise again with different people in the organisation. If you have access to customers, why not ask them to validate your name choices, too?

Encouraging adoption

Once the language is forming, it needs to be visible to everyone to encourage adoption. It takes time to embed a design system into a culture, so it needs to be evangelised from an early stage. Talk to people about the components that are being worked on, even if they aren't finished. Provide regular presentations to the wider organisation. Write blog posts internally to share the language.

Display components and their names on a wall to remind people as they walk past. This encourages discussions and feedback. If teams can't co-locate, then a shared document online works well.

Invite people to give feedback, and test them regularly. Surveys and face-to-face sessions to allow people to scribble on components are useful.

At Ansarada, we have a weekly catch-up to discuss our design system. Anyone is invited, so it's a great time to present components and get feedback. We also use this time to discuss names for new components or variations.

Names might change, so it's important to make sure that this is communicated to those who directly work with them as soon as possible. Ideally those people will have been involved in that decision.

Developing a language that everybody understands takes time. Building a living system takes time. Laying the foundations should begin long before coding starts, whilst exploration and change is cheap. Naming is no longer something that can be left to developers alone at the time of implementation. It might seem like a lot of work, but it's a necessary investment for the system to survive the test of time. The more time goes by, the more expensive it becomes to change.

Resources

[1] Charlotte Jackson, A List Apart
From Pages to Patterns: An Exercise for Everyone
http://alistapart.com/article/from-pages-to-patterns-an-exercise-for-everyone

The author

Daniel Schutzsmith

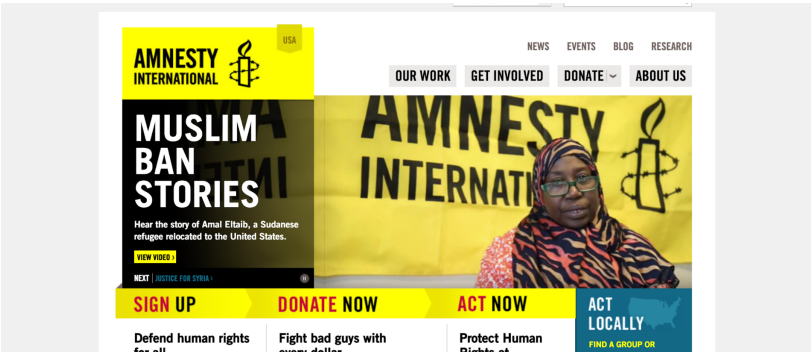
Daniel Schutzsmith (daniel.schutzsmith.com) is a rare breed – a hybrid of equal parts design, code, and strategy. He's devoted his career to making positive change to protect our world for generations to come. Currently, Daniel is the digital technology director of Natural Resources Defense Council (NRDC), leading an amazing team to create websites, apps and interactive experiences to help safeguard the earth – its people, its plants and animals, and the natural systems on which all life depends. Previously Daniel was the digital technology manager / senior web developer for Amnesty International USA.

CASE STUDY 1: HOW AMNESTY INTERNATIONAL USES DESIGN SYSTEMS TO WIN ON THE WEB

Amnesty International USA (AIUSA) has adopted a design system that combines the visual elements needed for all possible types of online content and included it in their content management system, so any site administrator can create or edit a page with any design elements they might need.

In the summer of 2016 Amnesty International USA **[1]** wanted to embark on a redesign and restructuring of their main website along with its content and CMS.

The previous AIUSA website was built on Drupal 6 in 2010. Since then, the website hadn't evolved very much.



The old website design of Amnesty International USA

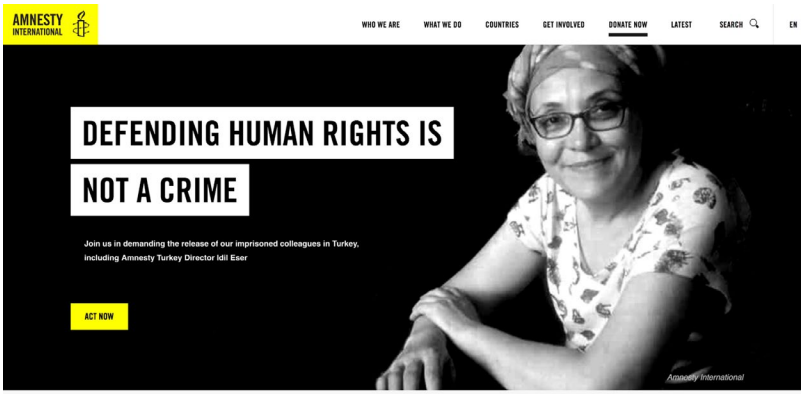
The main issues were:

- **No clear focus or call to action**. Each page had several calls to action and overused the dominant branding colour, yellow, so the user had no clue what they should be focusing on or why.

- **Poor mobile view.** The website was created before responsive design really took off, and therefore the user experience on smartphones and tablets was terrible.

- **Confined design.** The design was too constricting for website administrators to work with, and any type of change in content display became a massive endeavour that was time consuming and costly.

- **No longer looked like sibling websites.** Amnesty has several sections around the world, each located in a different country. These sections run independently from one another but answer to a common parent organisation called the International Secretariat (IS). This parent org had redesigned its website in 2014 **[2]** and several other Amnesty sections followed suit in recent years **[3]**, **[4]** making the AIUSA website stick out like a sore thumb.



The website of Amnesty's International Secretariat

Redesign goals

In order to bring the website up to the standards of their sibling sites, AIUSA knew they'd need to define some key goals to work towards to help measure success:

- **Responsive / mobile first**. The website should load on all devices quickly and in the best layout for the device at hand. Images and content should be optimised for that device and load fast.

- **Define a ladder of engagement.** The website should simplify and guide the user experience along a path of online engagement (make a donation, become a member, sign a petition, sign up for the newsletter, etc).

- **Create a useful visual language.** Design and develop a common visual language that can be used to create new microsites, campaign sites, and other subdomains easily on

the front- and back-end. It needed to be reusable and easy to implement by anyone skilled in front-end coding as well as without any coding in some situations. A design system felt like the right thing to focus on moving forward.

Reasons for a design system

For AIUSA, it was a no-brainer to create a design system to accomplish the goals they wanted to achieve.

They saw three key benefits of using a design system:

- **Support of the brand.** Because we're a global brand we needed to fit in with other countries' branding but still differentiate ourselves. Also great for microsites.

- **Naturally agile.** All of the pieces of the building blocks are available to use to make new solutions without sacrificing speed. Fits nicely into an agile workflow.

- **Ease of use.** Gives familiarity to the visual and interactive objects provided for a user. Reinforces a recognisable pattern the user can rely on for minimal decision making.

Another major factor for AIUSA to use a design system was the fact that most front-end frameworks were, essentially, already design systems! AIUSA was planning to use the Bootstrap 4 framework as the basis and then to theme only the elements that would be necessary to reduce page speed bloat.

Lastly, a visual identity toolkit from the parent organization, lovingly called the "Big Yellow Book", already existed and provided the foundation for typography, colours, imagery and more.

The visual identity toolkit specified how Amnesty's logo should be used, for example

The IS website and the Amnesty Australia websites both used the visual identity toolkit as the guideline for their redesigns, so it was a natural fit for AIUSA to follow suit.

If the toolkit hadn't already existed, it's exactly where the AIUSA team would have started anyways on their design journey: establishing the colours, typography, and imagery style first, then making the design system second.

Choosing the right design partner

AIUSA started by finding the right design vendor to work with that could compliment their in-house web development team. They found the perfect partnership in the team at Hyperakt **[5]**, a social design agency based in Brooklyn, NYC. The company was founded by two refugees and has grown to become a large team of talented designers and strategists that understand the needs of causes like Amnesty.

Creating the system

With Hyperakt's help, AIUSA started with a robust discovery phase that lasted about four months. Because there were so many stakeholders involved in the project, it was important to discuss the needs, concerns, and expectations with as many people across the organisation as possible. This included board members, member leaders, group leaders, volunteers, donors, interns, and staff.

Staff and members of Amnesty International USA participate in a content mapping exercise facilitated by the Hyperakt design team

The redesign team then...

- ...attended Amnesty regional conferences across the US to talk directly with members and hear what their issues had been with the old site and what they needed to accomplish with a new site.

- ...held three stakeholder workshops made up of staff and member leaders to walk through content mapping, site mapping, and user flow brainstorming.

- ...conducted an online survey that included three fundamental questions:

  - What do you like about the current website?

  - What do you dislike about the current website?

  - What do you wish you could do on a new website?

- ...met with member leaders and volunteers during their regular monthly phone calls. This also doubled as a way to keep AIUSA's influential members always informed along the way.

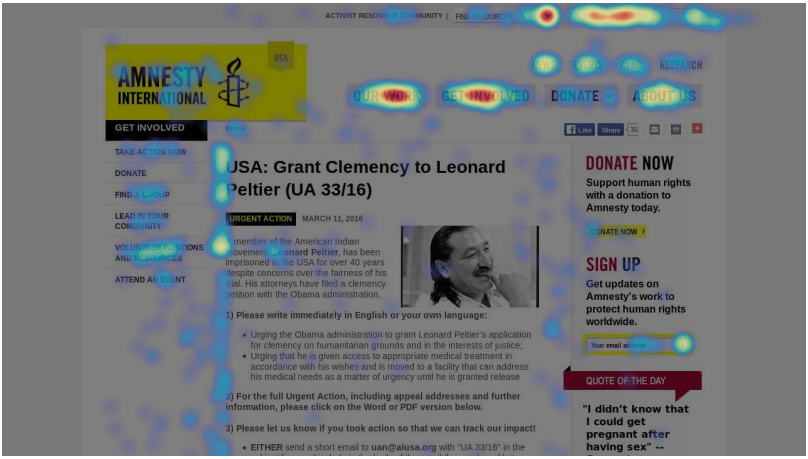
- ...conducted one-on-one interviews with board members and some key member leaders to ensure AIUSA knew their needs and concerns.

We also used the following tools:

- **Analytics**, such as Google Analytics, played an integral part in the discovery process. For example, the average bounce rate for the year before the redesign was 15 percent. Drilling down further we were able to identify older, evergreen content that was responsible for this figure.

- **Heat mapping**, using Hotjar **[6]**, provided a terrific way to see how users were interacting with the website.

- **Feedback forms** were placed on the old website to get insight from users, and similar forms are still being used now to keep listening. This service was also supplied by Hotjar.



Heatmap of the old AIUSA website showing popular areas clicked on in red, yellow and green

Designing for content

At the suggestion of Hyperakt, AIUSA started the design system by designing wireframes for blocks of content that they knew they would need – evergreen content.

This included the 'Issues & Campaign' landing pages, which are part of 'Our Work' on the website.

They also focused on creating generic blocks of content that users expect, such as page teasers that entice the user to click through to a new page.

Once wireframes were complete, the design team focused on applying the guidelines from the visual identity toolkit to the wireframes.

The AIUSA development team asked Hyperakt to deliver the design

as Sketch files **[7]**, so that developers could use Sketch's "copy as CSS" feature and override as necessary in Bootstrap 4 in a matter of seconds, which saved valuable time.

Implementing the design system in a CMS

AIUSA decided on WordPress as their content management system of choice.

"The previous CMS was Drupal 6, and we didn't see a bright future to keep following the Drupal path. WordPress, on the other hand, had a robust community and an ease of use for both user and developer that we hadn't seen in any other CMS," said Gabriel Dekoladenu, junior web developer at AIUSA.

The development team also decided that a WordPress starter theme made the most sense. They chose Sage **[8]**, formerly Roots, because it comes with many modern tools such as Gulp, Browsersync, and Bower.

Reasons Amnesty moved to WordPress

There were several factors that convinced AIUSA to move to WordPress:

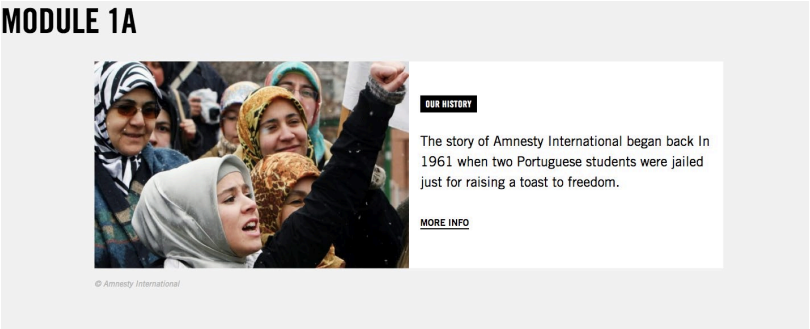
- **Faith in maintenance.** Plugins and core are regularly updated. Security updates are flawless. Drupal upgrades were cumbersome and often have breaking changes, even in Drupal 7 and 8.

- **Easier development.** Their experience was that smaller teams were able to get bigger problems solved faster with WordPress. Dev environments were also much easier to work with using plugins like MigrateDBPro **[9]**.

- **Robust community.** An abundance of experts and vendors to work with as well as a support community always willing to help answer questions and explain solutions.

Building the design system into the theme consisted of going through the wireframes that Hyperakt made, creating the code, and experimenting with different layouts.

The development team started by building a giant style guide page that consisted of Bootstrap 4, combined with custom HTML, CSS, and JavaScript of all possible content blocks:



The team then focused on using Advanced Custom Fields **[10]** and its amazing Repeater Fields and Flexible Content Fields, so any block of content from the design system could be added to any page.

The main advantage of using this combination of custom fields was to allow website administrators to move content around on a page and display it in different ways without having code at all.

The implementation of the design system in the PHP files of the theme was surprisingly easy.

"It was basically a series of 'if' statements inside of Flexible Content Fields. This made it extremely simple for any web developer present or future to be able to evolve this system over time", Gabriel said.

Here's an example:

```
if (  have_rows( 'module content' )  ):
          //loop through the rows of data
          while (  have_rows( 'module content' )  ) : the_row( );
                    //show module 1
                    if( get_row_layout( ) == 'module_content_1'  ):
                              //put module 1 content here
                              echo "Do something cool for Module 1 here";
                    //show module 2
                    elseif(  get_row_layout( ) == 'module_content_2'  ):
                              //put module 2 content here
                              echo "Another cool thing, but now for Module 2!";
                    endif;
          endwhile;
endif;
```

Website administrators now have an easy-to-use interface right in their content management system that enables them to use the design system without any concern of being off brand or running into any inconsistencies. Best of all, as the design system evolves, the code can be updated and the changes automatically show up across the website without the need to edit individual pages.

The future of the design system

AIUSA took the time to plan their design system with plenty of input

directly from the people who would be using it, both internally and externally. They then took that system and implemented it into a front-end framework and CMS to make it usable rather than theoretical.

The final website design

Thus, they were able to create a system that can evolve over time and even be used on other platforms as the digital tools change.

For example, AIUSA is planning a mobile application to roll out later this year along with a possible desktop app in 2019.

They consider this only the beginning of making design systems come alive in usable ways for both users, administrators, and developers.

Resources

[1] Amnesty International US
https://www.amnestyusa.org/

[2] Amnesty International Secretariat
https://www.amnesty.org/en/

[3] Amnesty International Australia
https://www.amnesty.org.au/

[4] Amnesty International UK
https://www.amnesty.org.uk/

[5] Hyperakt
http://www.hyperakt.com/

[6] Hotjar
https://www.hotjar.com/

[7] Sketch
https://www.sketchapp.com/

[8] Sage WordPress Theme
https://github.com/roots/sage

[9] Migrate DB Pro
https://deliciousbrains.com/wp-migrate-db-pro/

[10] Advanced Custom Fields
https://www.advancedcustomfields.com

The author

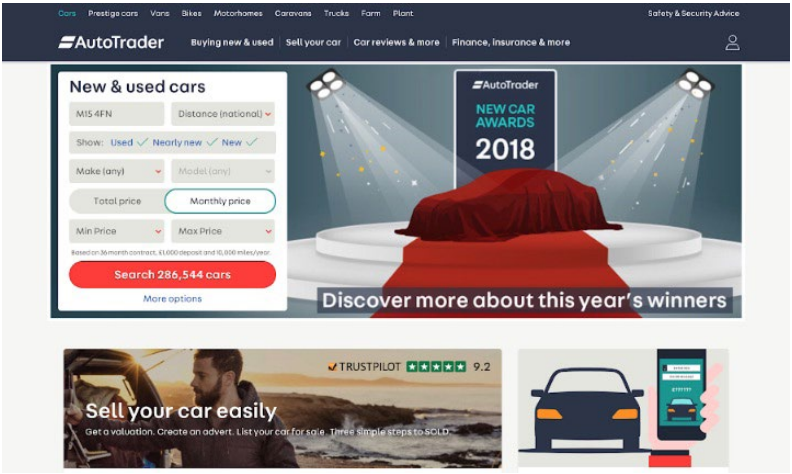
Dan Donald

Dan (twitter.com/hereinthehive) is the front-end lead at Auto Trader, previously at McCann Manchester and the BBC, and aims to help designers and front-end developers collaborate better and improve the quality of their output. He's the co-organiser of the UpFront conference (upfrontconf.com) in Manchester and previously organised gig-style web event Speak the Web. When not doing web stuff, he makes music in alt-rock outfit Mark of 1000 Evils (markof1000evils.bandcamp.com) and dabbles in writing projects he'll never finish.

CASE STUDY 2: HOW AUTO TRADER JOINED UP THE PIECES OF THEIR DESIGN SYSTEM PUZZLE

Design systems can help bridge the gap between design and live production code and facilitate conversations across many teams. Our community has shared so many different approaches companies have taken, and there are so many open source tools for the creation of style guides and component libraries, that you'd be forgiven for thinking that implementing a design system is always simple and straightforward. When you're working on a large site, which forms part of an ecosystem of other smaller sites and back-end tools, it can be a very different experience, though.

Most people will be familiar with the main consumer site of Auto Trader **[1]**, the UK's largest car marketplace, but we also have niche sites for farm machinery and campervans and a suite of tools called Portal, which enables people within the industry – such as car dealers and franchise groups – to manage their stock and make the most of the Auto Trader platform.



The Auto Trader homepage

The reality of different tech stacks

Like any organisation that offers a group of technology products, we had to deal with differing tech stacks, and some legacy code alongside some that's really contemporary. On top of our Java back-end you'll find vanilla JavaScript, older frameworks and libraries such as Backbone and jQuery as well as versions of Angular and React. That's the reality of working on a large real estate of sites and apps. We also have design staff split between Manchester and London and teams focused on various, specific business priorities. So there was a lot we needed to take into account when we considered implementing a design system.

Our designers had already all migrated to Sketch as their tool of choice, worked with Craft **[2]** and Zeplin **[3]**, and had established a decent workflow for collating design assets as reusable patterns to speed up design and achieve greater consistency but this hadn't yet

translated into code. Across the business we were discussing how we could improve things, so we established a working group – initially called 'Core' – at the end of last year that brought together design, front-end, QA and web platforms.

We had duplicated effort across many sites for essentially the same thing – buttons is just one example. There was a lack of control over how design changes were rolled out, as any changes were dispersed through various teams and deployed at different times. Inevitably, this lead to an overhead on any development work and gave room for inconsistencies to creep in.

People often have different ideas when they talk about design systems, what they are and how they should be used. We wanted to find a way to better communicate the workflow of design patterns, which would result in a higher quality output. Because of different tech stacks and products, a few component libraries that used Storybook for React **[4]** and a custom tool for Portal were already starting to grow, but we felt that there was a piece missing.

The goals of our design system

We set ourselves some goals. The design system had to...

- ...work across tech stacks
- ...embrace accessibility and inclusive design
- ...use OOCSS, so we could easily skin objects

We also needed to...

- ...make it easy for people to use it in all kinds of projects

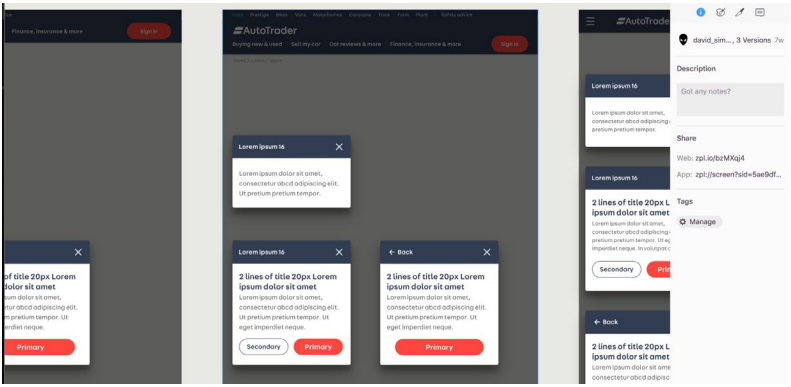
- ...ensure everyone felt a sense of ownership and could contribute

- ...maintain quality and communicate changes effectively

We decided that the missing piece of our design system puzzle should be something that transfers Craft and Zeplin's output to some kind of a style guide focusing on output that the component libraries could use. That way, we would be able to translate the original design into Sass with suggestions of possible markup, which the component libraries used in parts of the business could pull in through npm [5] or Yarn [6] as developers build active components. Having identified the problem area, it certainly gave us plenty to think about with a set of constraints. It's not like a greenfield project: it didn't have a consistent base to build from as every project had its own codebases with their own baselines.

Around the same time, two colleagues were championing inclusive design within Auto Trader. They helped raise awareness of accessibility issues as we worked on our features and user stories. So when we started looking at reusable patterns, accessibility was at the forefront of everyone's minds – from designers and developers to QAs and testers.

Whatever method we'd use to communicate the code we created, we needed to show examples of what 'good' looked like and reference any issues a developer should be aware of with any given pattern. This meant, for example, illustrating that there may be different tag structures based on context of use, using notes to explain common issues with accessibility as well as how this code should be used. While collating examples and documentation around a design pattern is no doubt useful, we were conscious that these are the parts which can also lag behind and become obsolete if not kept up to date.



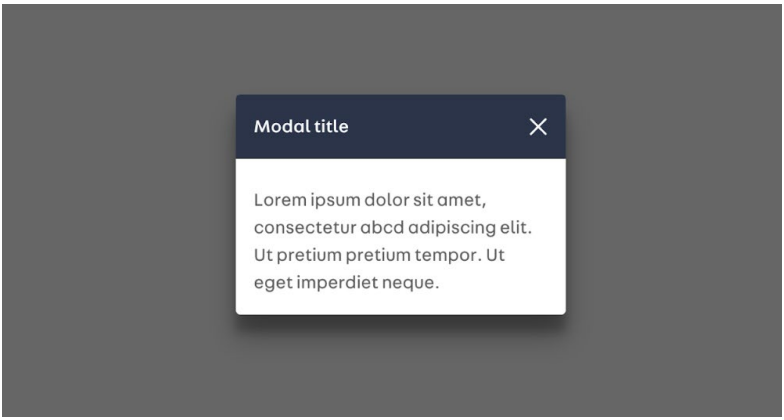
We may move designs over to Zeplin, which uses the same symbols as Craft and shows us the expected variants of a design pattern

Building the style guide

We explored the many existing tools for style guide generation, which all use different approaches. I had used KSS-based solutions before (Knyle Style Sheets, a methodology for documenting CSS and generating style guides **[7]**), but it felt like whichever system we'd use needed to be something that posed a low risk in case we chose something else further down the line.

We needed something that didn't entwine with our production code and discovered Fractal **[8]** by Clearleft, which mostly fit our needs. Its documentation uses Markdown, which implied a low barrier of entry for non-developers, it was easy to integrate into npm tasks and gave us a style guide we could host almost anywhere. Each pattern needed some specific configuration and template-based work but it wasn't significant. It gave us a view of the pattern using production CSS and the capability to add example markup or CSS with the option of adding some documentation specific to each pattern.

With a little trial and error, we found processes that worked for us: the simple feature-branch model in Git **[9]** for specific new patterns would be peer reviewed through pull requests and sent through QA. Once a pattern was fit to be consumed by other developers, its status could be changed from 'Experimental' to 'In Development', then 'Production'. We'd manually up the version of the code, which would trigger a build process in Artifactory **[10]**, making it available for developers to pull into their projects. Automated notices would go to the channel we have on Slack and we'd send out messages explaining what non-developers would need to be aware of.



Example of the first pass of the modal pattern. There are a few variants, each with their own responsive behaviour. We use the 'Notes' tab in Fractal to collate information around accessibility, use of language and soon-to-add findings from user research

Communication and collaboration

As is often the case with projects like this one, it wasn't really about the technology but the communication between people. The project progressed, and the small initial team encouraged more developers

and designers to be a part of it. We'd established a solid method of how code would be added to the project, reviewed, tested, deployed and consumed across the business. SemVer **[11]** was in use to clearly mark whether code would be a breaking change (a major release) or something less impactful (a minor release). We used Slack channels for general project-related conversations but separated out the developer conversations to keep the noise low. Release notes had their own channel too. We also gave talks in-house and sent out emails to ensure anyone that needed to know about the project heard about it first hand.

This resulted in some production-ready code that translated design assets into a style guide, which anyone could view internally and developers could pull into their projects through npm or Yarn, but that's not the end of the story!

CSS strategies

We analysed the CSS strategies we used across teams and agreed on how we'd work on this project (which we decided to name Sparkplug – it had to be car-related, didn't it?). This also helped developers to harmonise workflows more than in the past. Multiple parts of various strategies wove together nicely:

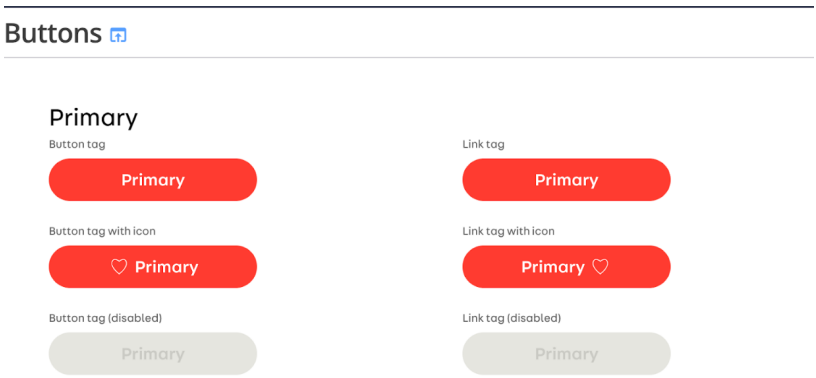
- BEM as a naming convention **[12]**
- OOCSS as a structural model **[13]**
- ITCSS as a means of splitting code into meaningful layers, which also allowed projects to import only what was needed above the common settings, tools and base layers **[14]**

When used together in the right way, these strategies can be really powerful and again allowed us to communicate between developers

more easily, as there was a clear rationale behind what we were using and why. Writing in objects gave us opportunities for skinning and enhancing, which – as some products diverged from the core design assets – just made perfect sense. We also considered how this code would need to be used in the real world. Sass in a style guide can be very different from where it'll actually be used, especially when the intent is to use it in many different places, each with a different baseline to work from.

Occasionally, we coded defensively and assumed a pattern could be used on more than one markup structure for certain use cases. A pattern would be coded to anticipate this and ensure it would work as planned. When it came to the implementation, there were of course teething problems. Our solution was to aim for an archetype, an ideal solution, and deal with problems pragmatically. This project only works if people actually use it and doesn't add unrealistic overheads or demands to their work.

A great example of dealing with change pragmatically was a fairly large design change in one area of the business that was about to be introduced. We had to ask ourselves if it still made sense to use Sparkplug to supply the CSS for reusable patterns if the patterns were to change in one of the products? A button still very much shared properties with the button pattern in Sparkplug. There were some aspects of variability that weren't there before like a version which needed a gradient background and a different border radius value, for example. This was a real-world challenge. Change happens.



Example of the button pattern, starting with a simple object, then the layering state (disabled, focus and hover) and variant (such as primary, secondary, etc). The reference shows that if you need to use a link tag for your use case, it'd look and behave the same as when a button tag is used

We explored where variability might be needed, abstracted local variables from each pattern and moved them to a central config map (an array in Sass terms **[15]**). We then ended up adding a small config layer to our ITCSS implementation as this needed both the settings and tools layers and allowed people to merge a local config map with the defaults to get the output they needed. This allowed for output that was necessary and didn't have to be overwritten within a project's context, potentially adding risk to future changes and otherwise generating bloated output. It was a great exercise both of communication with design and developers as well as attempting to be pragmatic with our output.

Constant evolution

It was a sign of the success of the design system that our head of content was able to skill-up quickly in using Git and Markdown to add her angle both to the general documentation in the form of a glossary but she was also able to add content against certain design patterns where considering use of language was appropriate. We're now also encouraging our UX researchers to contribute. Sparkplug should continue to be as inclusive as we can make it, so it should be a living project, which adds real value.

Sparkplug is constantly evolving. The design system has now grown to include many patterns, which are slowly being integrated into various tech-specific component libraries. It's not finished: we have production code out there which gives us greater control and consistency than we had before. Going forward, we always need to be adaptable to change and focus on the people and communication as much as the code.

Resources

[1] Auto Trader website
https://www.autotrader.co.uk/

[2] Craft
https://www.invisionapp.com/craft

[3] Zeplin
https://zeplin.io/

[4] Storybook
https://storybook.js.org/

[5] npm
https://www.npmjs.com/

[6] Yarn
https://yarnpkg.com/en/

[7] KSS methodology
https://github.com/kneath/kss

[8] Fractal
https://fractal.build/

[9] Atlassian, Git Feature Branch Workflow
https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

[10] JFrog, Artifactory Universal Repository Manager
https://jfrog.com/artifactory/

[11] Tom Preston-Werner, Semantic Versioning
https://semver.org/

[12] CSS Guidelines, BEM-like Naming
https://cssguidelin.es/#bem-like-naming

[13] Louis Lazaris, Smashing Magazine
An Introduction to Object Oriented CSS (OOCSS)
https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/

[14] Harry Roberts, Creative Bloq
Manage large CSS projects with ITCSS
https://www.creativebloq.com/web-design/manage-large-css-projects-itcss-101517528

[15] Map array, Sass (Syntactically Awesome StyleSheets)
http://sass-lang.com/documentation/file.SASS_REFERENCE.html#maps

The author

Inayaili de León Persson

Inayaili de León Persson is principal design consultant at Make Us Proud (www.makeusproud.com) / YLD (www.yld.io) — a London-based digital design and engineering agency — where she leads the focus on design systems and design ops for a roster of high profile clients. Previously, she led the team creating Canonical's Vanilla design system. She writes frequently for well-known online publications and on her own web design blog and speaks at local and international web conferences and meetups. She's the co-author of *Pro CSS for High Traffic Websites*, and author of *Moving to Responsive Web Design*. She's Panamanian Portuguese, born in the USSR, and lives in London — her favourite city in the world.

KEEPING YOUR DESIGN SYSTEM ALIVE

It can be difficult to persuade your company to create your first design system — but it tends to be even more difficult to convince people that the system, once released, has to be maintained.

A common misconception about design systems is to think that once the first release of the initial set of UI components and guidelines is done and implemented, the work is finished — which is quite far from the truth. In fact, once the first release is out of the door, the hardest part of the work is about to get started: getting people to use, contribute to and improve the system.

These are my top seven tips to maintain a healthy living design system, and encourage usage and participation from your organisation.

Treat the design system as a product

The concept of design systems can be tricky to explain to people who haven't used or created one before and not seen the benefits for themselves.

I find it easier to talk about a design system as you would of any other product. If you think about it, it includes all the same constituents:

1. Users
2. Roadmap
3. Releases
4. Features and feature requests

5. Maintenance
6. Support and help documentation
7. Bugs (whether you like it or not)
8. Cross-functional teams working on them

Also, just like a product, you don't get it right the first time — you iterate and make it better.

Explaining the concepts and framing conversations about design systems by drawing parallels to product design and development makes it easier for those less familiar to understand that it isn't a simple project which, once completed, can be forgotten.

Put someone in charge

Before anything else, make sure someone is responsible for and leading the project. Regardless of their title (for example, project lead or product owner), it must be clear that they're responsible for keeping the design system alive and healthy.

Trust me: if you don't assign responsibility, the design system won't lead itself — never mind how enthusiastic your team is about it. Everyone is always busy and, even though it can save teams loads of time in the long run, it's easy for design system work to not be seen as a priority.

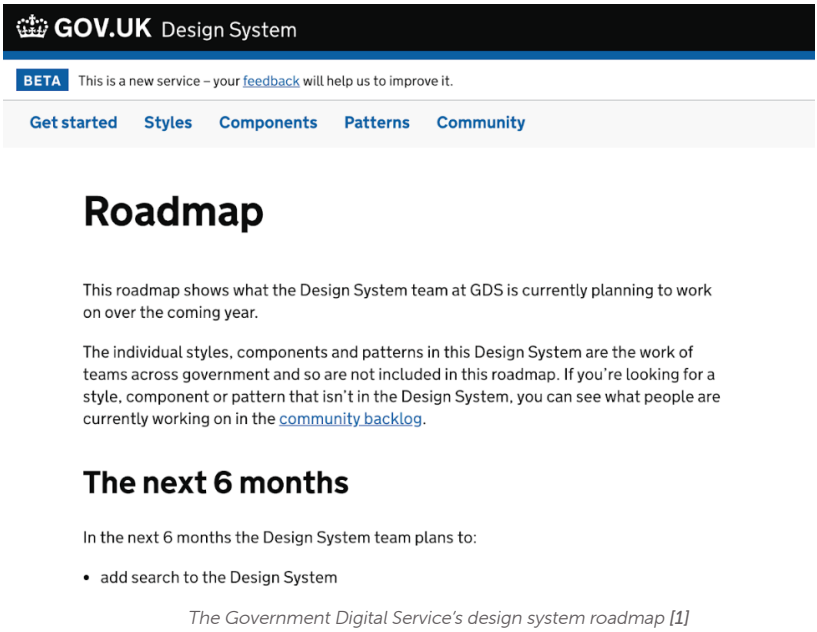
Your design system's lead will be there to make sure that the project is prioritised, and that you're working towards the goals that you've defined for it. And also importantly, they will be there if people have questions or need guidance. They can have the final word on disagreements and have the ability to gather key people if needed.

Have a roadmap

It's important that everyone working on a design system shares a common vision: where you would like to be in three months, six months, one year.

Just like any other product roadmap, your roadmap should not be a view of your daily tasks, but a high level view of the vision and what you're focusing on next.

You can refer to the roadmap regularly when you're planning design system work. It will make it easier to decide what tasks need to be done next, what is the most important thing that you need to work on, and what is less important.



The Government Digital Service's design system roadmap [1]

How to start

You can get a roadmap started by gathering a few key people from different disciplines in a room and have them write down what they see as key goals for your design system on sticky notes.

These can be things like "having a website", "having a social media presence", "being fully accessible", "getting 50 percent of the teams to use it", "writing better docs", and so on.

Then prioritise them together: divide them into sprints, months, quarters, releases — whichever way fits your working patterns best.

You may have to rephrase some things to be less solution-focused, but this will get you started and will get people discussing what the priorities are.

Remember: roadmaps change, so you need to make sure you're checking and updating your goals regularly.

Be open

When working on a design system, your ultimate goal is to get people to use it. This works better if you include people in your vision, and share your work and your process.

It's important to work hard at becoming a good listener, be open to different ideas and not disregard people's comments and suggestions. This can be particularly tricky as your most likely users (designers and developers) tend to be opinionated people!

There are a few things you can try to be more open:

- Use tools that are more inclusive. In one of my recent

projects the team decided to use Figma, instead of Sketch, so anyone on Linux, Mac or Windows can see and comment on design files as it's a web-based app without the need to install any software.

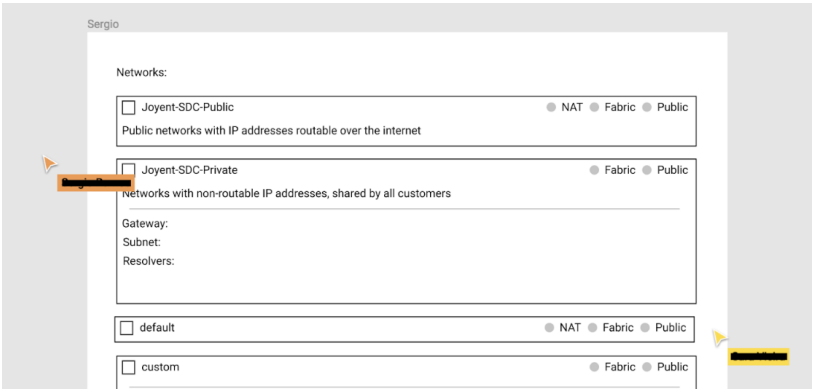
- Have discussions that others can keep track of asynchronously, for example on GitHub.

- Make time for discussion, like having regular show and tells and critiques on new components.

- Organise workshops to define and name new components.

- Organise mini-sprints (instead of two-week sprints), which include people not usually part of the team to focus on fixing as many bugs as possible on a specific theme, like typography or performance.

- Write about your work, either internally on mailing lists, or externally on a blog.



Follow an open process by using tools everyone can access, such as Figma ***[2]***

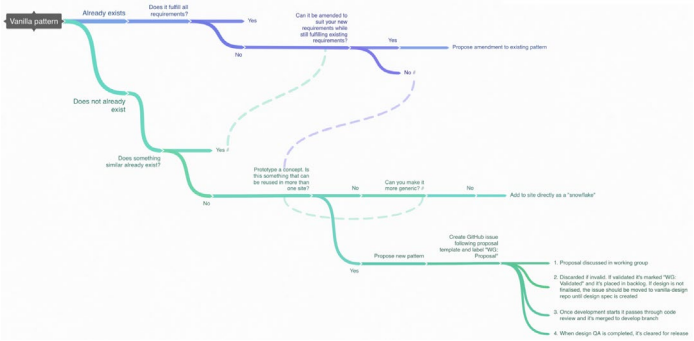
Become comfortable sharing work in progress. Even people that aren't working directly on the project will be affected by it, and you want the design system to be useful and have a positive impact on their daily work.

Explain how to contribute

As any other product, a design system is never finished. Styles change, new components are added, old components are removed, improvements are made to the codebase, bugs are fixed.

Even if you don't accept contributions from a wider community, you should still define the contribution process for people within your team.

A good way of doing this is by creating a diagram of the process. You could use a whiteboard and iterate on it, together with your team, as you find better ways to expand and maintain your design system.



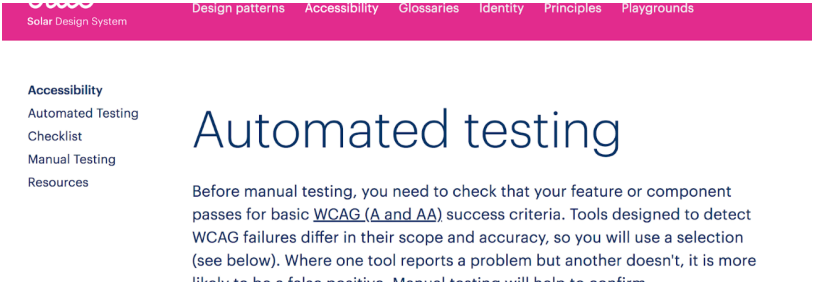
Decision process before you submit a proposal for a new component in Canonical's Vanilla Framework ***[3]***

These diagrams may look simple and even obvious, but going through the process of thinking how you want the system to expand usually brings up important conversations at every step about the principles, values and standards you want to maintain, and also things that you still need to work on that you hadn't thought of.

Love your docs

Even if only a small number of people are working with your design system, you want to define some parameters on how things should be used, how to put everything together, and how to propose new code or changes.

The documentation can include brand guidelines, style guides, browser support, accessibility guidelines, contribution guidelines, getting-started docs, coding standards, tone of voice, etc. In a way, your docs *are* your design system!



Bulb's design system includes not only guidance on pattern usage, but also accessibility guidelines, design principles, and checklists among other documentation [4]

Maintaining a robust set of UI components can be hard, but maintaining good docs is much harder. Yet they are crucial to maintain the consistency and the high standards that you want from your design system.

There are a few things you might want to try to keep your docs well-looked after:

- Have a column in your task board (kanban, scrum, etc) specifically for documentation, so every ticket needs to visibly go past it to be completed.

- Consider adding the need for well-written documentation as a requirement for pull requests.

- Test your documentation frequently, by having different teams implement the design system while your team (if you have one) serves only as support. The feedback you will get on how to imprdove your docs will be invaluable.

- Keep your docs clear and specific enough, but don't let them be authoritarian or dogmatic, or you'll scare people out of contributing and improving your design system.

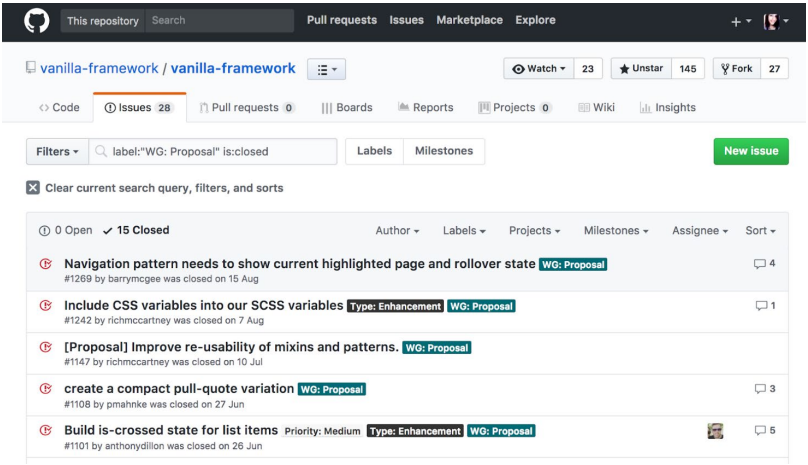
- Make all your docs easy to find by linking to all of them from one single central location (even if it's just the README file of your project).

Assume change (and plan for maintenance)

A good design system is never done. New components and guidelines are needed; new technologies impact it; maybe you want to add things like animation, or improve performance or accessibility. Or you may need to deprecate things that are not useful anymore. It never ends.

To know that maintenance and improvements are needed, you need to make sure communication between the design systems team and other teams happens frequently and often.

Consider creating something similar to what we did at Canonical: a fortnightly working group meeting to which the entire design team was invited. The agenda was defined beforehand by GitHub issues submitted to the system's project with a specific label.



An open-sourced design systems meeting agenda, created by the participants in GitHub before the meeting (the agenda items are marked WG: Proposal) ***[5]***

Discussions at this meeting could go from "I think we need more utility classes for spacing" to "we designed a new accordion pattern for product X, should we add it to the library?" and "the contribution diagram isn't clear enough". Notes were taken and shared for everyone who couldn't be there.

Having this fixed time in everyone's calendar worked well and created a good cadence of discussion and information-sharing.

You can also create some freeform space in your sprint planning for fixing minor bugs and let your team prioritise that time. This will ensure the small cracks in your design system are being repaired regularly.

Remember that even teams at large companies that you might assume always do everything right learn new things and change their minds midway through their projects — always expect change, whether it's big or small. Whether you like it or not, your design system will need maintenance.

Conclusion (and bonus tip!)

As you can see, design systems are a lot of work — but they are also a lot of fun and really rewarding to work on.

My bonus tip is to learn about what other people outside of your team are doing as much as possible, and share your experience with others. Our industry is still young and we're inventing and trying new things all the time. We all have new ideas and problems that others find interesting — including you!

Resources

[1] Gov.UK Design System Road Map
https://design-system.service.gov.uk/roadmap/

[2] Figma
https://www.figma.com/

[3] Inayaili de León Persson, Vanilla Framework v1
https://yaili.com/work/vanilla

[4] Bulb, Solar Design System
http://design.bulb.co.uk/

[5] GitHub, Vanilla Framework
https://github.com/vanilla-framework/vanilla-framework/issues

Your business deserves
powerful virtual private servers

FULL CONTROL AND REAL-TIME PROVISIONING

Run your server your way with full root access and have it ready to go when you are with super-fast set up and upgrading.

CHOICE OF OS,CONTROL PANEL AND BACKUPS

Choose the operating system, use cPanel or Plesk, and select a backup strategy that works for you.

UP TO 400GB DISK SPACE AND 16GB MEMORY

We only use the latest hardware and software on our platform to give you a fast and responsive website.

FULLY MANAGED AVAILABLE

In-house Customer Services team is here to help you with any questions - 24/7/365.

Find your perfect hosting package at

www.heartinternet.uk or call us on 0330 660 0255