DesignBetter
by InVision

Design Engineering Handbook

By Natalya Shelburne, Adekunle Oduye, Kim Williams, Eddie Lou, and Caren Litherland (editor)

This book is brought to you by InVision, the digital product design platform used to make the world's best customer experiences. Discover more best practices for better design and get free podcasts, articles, books, and reports at designbetter.com and insidedesign.com from InVision.

Check out the rest of the Design Better library

Business Thinking for Designers

Remote Work for Design Teams

Animation Handbook

Enterprise Design Sprints

DesignOps Handbook

Design Systems Handbook

Design Leadership Handbook

Design Thinking Handbook

Principles of Product Design

Design Engineering Handbook

Learn how *design engineering*, an essential discipline to creating great products, brings together form and function while accelerating innovation. This book was written by industry leaders from the ***New York Times*, Mailchimp, Minted,** and **Indeed,** and will help you connect design and engineering to work more efficiently as a team.

Also available in epub, pdf, and audiobook formats.

The book you are about to read dives into the discipline of design engineering. Learn more about InVision and how you can bring design engineering best practices to your team's operations.

Discover more from the InVision platform

Design System Manager connects design and development with a shared language and workflow so teams can work smarter, faster, and more in sync.

Prototype for designers and developers to iterate on clickable experiences to establish confidence in designs before it's time to build.

Freehand for real-time collaboration on a digital whiteboard. Ideate, wireframe, and map out customer journeys as a team across design, product, and engineering.

Inspect makes it easy to move from design to code with specifications automatically surfaced in designs. Get the details you need to build directly from your prototype faster.

About the authors

Natalya Shelburne is a designer, developer, writer, educator, speaker, and artist. She is a senior software engineer at the ***New York Times*** and an occasional instructor at **Harvard Extension School.** Previously, she taught design at a nonprofit. Natalya holds bachelor's degrees in studio art and psychology, and a master's in creativity and talent development. Bridging mental models and building collaborative tools and workflows for design and engineering is at the foundation of much of her work.

Adekunle Oduye (Add-eh-koon-lay Oh-due-yay) is a UX engineer born and raised in the great city of New York. Currently, he helps build a design system that serves millions of users at **Mailchimp**. He has previously built products for companies like **Memorial Sloan Kettering, Justworks,** and **NASDAQ**. Outside of work, he is a board member for the Code Cooperative, an organization that teaches digital literacy and programming skills to individuals impacted by incarceration. He is also a mentor and coach for next-generation designers and front-end engineers.

Kim Williams is head of product design at **Minted.** Kim's an expert at threading together and synthesizing the vision of brand, product, and technology. She leads design to craft experiences that are human, hopeful, trusted, and performant. Prior to Minted, Kim led UX for the job-seeker organization at

Indeed, working with design technologists and UI engineers on search, native apps, and job seeker journeys. At **eBay,** Kim was head of design systems and collaborated with design technologists. Kim knows firsthand that design engineering is a critical discipline for accelerating product innovation.

Eddie Lou is the senior UX director and head of the design system team at Indeed, where he leads designers, engineers, and design technologists. Eddie initially started Indeed's Design Technology team with the focus of providing critical technical capabilities to the UX organization. The team later evolved into design engineering organizations which include UI engineers who focus on creating quality user experience to production. Previously he held various UX and engineering leadership roles at **BigCommerce, Visa, Apple, PayPal,** and **Cisco**.

Acknowledgements

Thank you to everyone who contributed to *Design Engineering Handbook.*

Editors: Caren Litherland, Eli Woolery, Aarron Walter, Rob Goodman, Jessica Dawson

Video production: Daniel Cowen, Eli Woolery, Andy Orsow

Illustration: Ranganath Krishnamani

Creative Direction: Aaron Stump

These people were instrumental in the production of this book: Tim Shelburne, Jina Anne, The Oduye Family, Chantel Renee Williams, Marshé Hutchinson, Trevor Williams, and Yvonne Williams

Contents

Chapter 1

Introducing Design Engineering

The unicorn myth

By Natalya Shelburne

Are unicorns real?

Odds are you've heard the myth of the *tech unicorn*: The hybrid designer slash developer who is as comfortable with kerning type as they are with sorting functions. This prized collaborator can magically leap across the boundary between design and engineering to deliver business-critical solutions.

You have probably worked with a unicorn. Maybe you've even been called one yourself. If so, have you ever thought to question it? Names are important because they convey meaning and shape attitudes. Why has an entire industry embraced referring to people as mythical beings when the work they do is not only very real, but actually quite common? Is it because we recognize the existence of the role, but can't quite articulate its definition? Or do we misunderstand the very nature of the role—as somehow being magical and innate rather than a skill set that can be learned and performed by anyone with enough curiosity and drive? Whatever the reason, the fact is that a proper name for the discipline and a definition of the roles it encompasses are long overdue.

The name of the discipline is *design engineering*, and a *design engineer* is someone who specializes in the intersection of the disciplines of design and engineering. Formal job titles, which can vary from company to company and even from individual to individual, are starting to emerge and become standardized. Titles for the people working at the intersection of design and

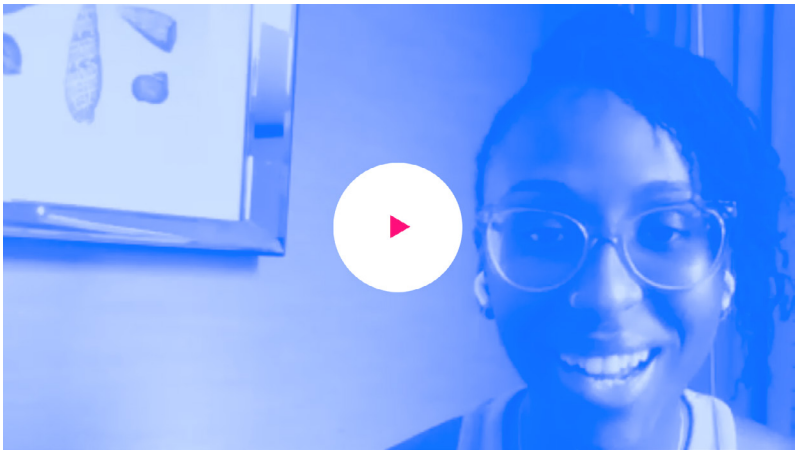
engineering include *design engineer, design technologist, front-end designer,* and *user interface (UI) engineer*. For the purposes of this book, and for the sake of simplicity, we will refer to this role under the umbrella term of design engineer. Words matter. Calling someone who serves to connect two disciplines a rare and mysterious mythical creature has a profound impact. If the work is named, it becomes both visible and valued. After all, we don't call full-stack engineers unicorns, do we?

Unfortunately, the notion of a "gap between design and engineering" is almost as common as the myth of the tech unicorn. Although a gap in understanding may exist between specific design and engineering teams, no gap exists where the two disciplines intersect. Instead there is an overlap, and design engineers are experts in the complexities that arise in that space. Design engineers are skilled at both design and front-end development, and they are able to contribute wireframes and mockups as well as front-end code. Prototyping at all levels of fidelity, whether via pen-and-paper sketch or live code, lets design engineers quickly grow their idea and shepherd it through the development process.

The work of design engineering

As the tech industry matures, and as organizations scale to unprecedented levels of impact, new specializations are emerging across all disciplines. The work of design engineering is quickly joining the ranks of functions demonstrating real business value.

But what is this work, exactly? Well, it involves setting up individual workflows and organizational structures that facilitate collaboration and communication across the intersection of design and engineering, as well as across product, marketing, and stakeholders. The work also entails understanding a team's product and process, and bringing it up to speed with front-end and design best practices. A design engineer might focus on setting up a design system, documenting patterns, performing workflow audits and updates, building UI components, writing usage documentation, or working with stakeholders spread across an organization. Additional tasks could include establishing and maintaining a local development environment, setting up testing, and release management.



Ire Aderinokun, design engineer and cofounder of BuyCoins, talks about the efficiencies gained from speaking the languages of design and engineering.

Because design engineers are bilingual, they can facilitate better collaboration between highly specialized designers and engineers. They build and refine connections between form and function. They help designers see new technological solutions to design problems, and they help engineers better understand the power of design. Whatever the specifics may be for your organization, one thing is clear: Design engineering is an important cultural shift that allows organizations to scale their efforts through collaboration.

Whether you're an individual contributor playing the part of the unicorn who is struggling to define your path or someone in a leadership role who realizes you need more than luck to

attract, retain, and empower multidisciplinary collaborators, this book is for you.

Don't get left behind

Organizations that recognize the opportunity in the gap between design and engineering will outpace those that fail to invest in it.

The ability to quickly innovate, experiment, and iterate can mean the difference between success and failure. Simply sitting a designer and an engineer next to each other and leaving it up to them to figure out how to collaborate is a recipe for disaster. Sure, you might get lucky and see some success at first, but the passive approach neither lasts nor scales. If you don't invest in the people doing design engineering at your organization or support them by creating the organizational structures to facilitate their work, you will be left in the dust by the companies that do.

Investing in design engineering

In this book, we'll look at prototyping techniques and collaborative workflows, weigh the balance between soft and

hard skills in design engineers, and hear practical advice from industry leaders.

What are the right questions to ask when creating a focus on design engineering? What are the first steps to take? How do you scale this effort? Everyone can agree that improving speed to market and having the ability to validate product ideas early is important, but what does investing in those capabilities really entail?

The chapters to come contain practical, actionable guidance from industry experts on:

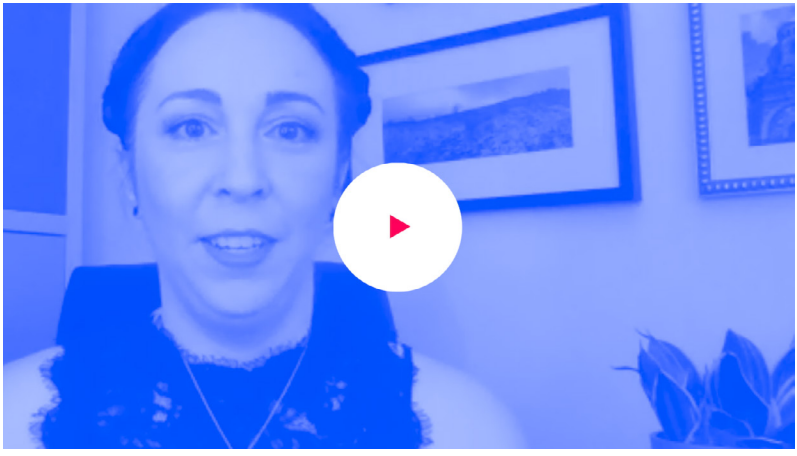
- The impact of design engineering on the product-development process
- Methods for examining user needs and connecting them to business goals
- Validation of ideas through prototyping
- Practical processes to integrate into existing design and developer workflows
- Building systems to facilitate collaboration
- Collaborative failures: rookie mistakes, skill siloing, and gatekeeping

- Rethinking organizational structures to support cross-functional collaboration
- Attracting, hiring, supporting, and growing design engineers
- Considerations for building a design engineering discipline from the ground up

Prerequisites

As you read the following chapters, I recommend you take a good hard look at your own beliefs and assumptions about what the work of design engineering can be. Keep in mind that people love to think in binaries: right brain versus left brain. Creative versus logical. Emotional versus rational. Design versus engineering. Generalizations aren't necessarily bad in and of themselves; simplifications and abstractions enable us to make sense of the world around us. Both designers and engineers excel at identifying patterns, assigning meaning, and categorizing. If generalizations and stereotypes are left unexamined, though, the superpower of pattern-matching can get us in trouble and hold us back. For example, the stereotype that logic is the domain of the engineer and creativity that of the designer is harmful. Both disciplines require logic and creativity. Does anyone really believe that designers create without any logic? Or that engineers are incapable of creativity

as they invent systems and architect solutions? Of course not. Yet these stereotypes persist and continue to shape the way we interact; some people really do believe that design and engineering are opposites. Unfortunately, believing that design and engineering are at odds represents a failure to grasp the profound opportunity of collaboration between disciplines in pursuit of a common goal.



Brenda Storer, an independent design engineer, talks about the hybrid nature of her profession and how it helps bridge gaps in cross-functional teams.

In fact, concentrating exclusively on the perceived friction between design and engineering excludes the many other disciplines that collaborate in an organization. Design engineering not only unblocks engineers and designers, but serves to facilitate collaboration and communication between

design, product, engineering, and marketing. Investing in design engineering results in improvements for everyone contributing their skills and ideas towards achieving a business goal.

In other words, as you continue to learn about design engineering, reject simplified binary thinking. Actively embrace the collaborative and multidisciplinary nature of the work we do. It's a feature, not a bug.

Self-fulfilling prophecy

Whether you are an individual contributor or an industry leader, your first priority is to establish your own belief in the work ahead. It's easy to feel cynical about the plethora of terms and different roles that are emerging in the industry. I've seen talented people grow discouraged by failed attempts or false starts in the race to set up design systems, establish collaborative workflows, or unblock lines of communication. It is critical that you ask yourself if you really believe in the work and the people doing it.

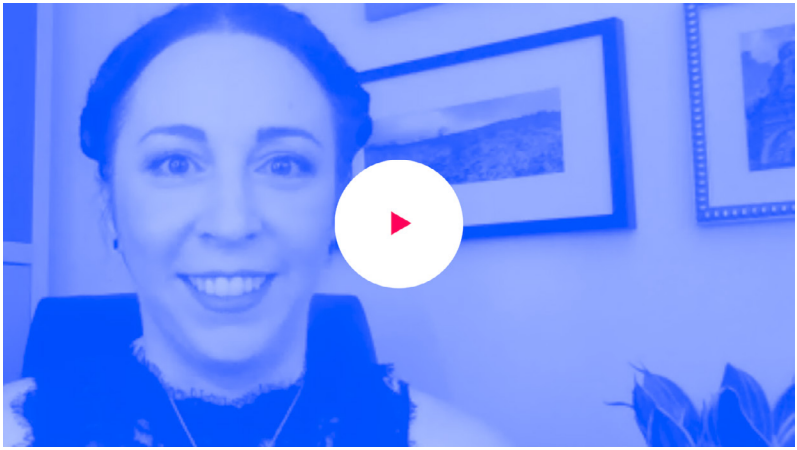
Most educators can tell you that the *Pygmalion effect*, named after the Greek myth of a sculptor who fell in love with a statue he had carved, is a self-fulfilling prophecy. Psychologist Robert Rosenthal and educator Lenore Jacobson created an

assessment that could spot academic potential, identifying which students would intellectually blossom during a given school year. Rosenthal and Jacobson had classrooms of children take a battery of tests and measures. They then gave their teachers a list of the students whom the tests identified as "having potential," even if some of that potential was considered latent. The researchers assured the teachers that the identified students would make the biggest improvements throughout the school year. The teachers agreed not to reveal the students' status or treat them any differently. At the end of the year, the high-potential children identified by Rosenthal and Jacobson did indeed show the greatest academic improvements. Amazing, right? It would be, if Rosenthal hadn't selected the children at random. There was no real assessment, no measure of potential; the real experiment was to observe the effect of teacher expectations on student achievement. As the researchers suspected, teacher expectations informed student outcomes. Without realizing it, teachers had adjusted their own behaviors in small ways, and the students benefited. Belief can shape reality.

This study has since been recreated and verified many times in the intervening decades. The Pygmalion effect has been observed not only in the classroom, but also in the workplace. What does this mean for design engineering? Simply put, it means that your belief about people and their work can impact their potential for success. This is especially true

for those in positions of leadership. People can tell when someone understands the value of their work and believes in their potential. We also recognize when we are not valued. Design engineering creates a space and a structure where the multidisciplinary talent that connects design and engineering is valued.

The work ahead



Brenda Storer shares some of the pros and cons of working in-house versus at an agency or consultancy.

Product design has reached maturity. As UX organizations scale, companies are adding functions that lead core capabilities. Design engineering is the name for the discipline

that finesses the overlap between design and engineering to speed delivery and idea validation. From prototyping to production-ready code, this function fast-tracks design decisions, mitigates risk, and establishes UI code quality. The design engineer's work encapsulates the systems, workflows, and technology that empower designers and engineers to collaborate most effectively to optimize product development and innovation.

Chapter 2

The Design Engineering Process

Hybrid method in practice

By Adekunle Oduye

Hybrid method in practice

Coming up with innovative solutions without room to explore, experiment, and iterate can be a challenge. That's why many companies are starting to create hybrid roles to bridge the gap between design and engineering. Enter the *design engineer*. This new "hybrid" thrives on quickly exploring different solutions and validating with users.

This chapter will explore the world of design engineering and its impact on the product-development process. Organizations that practice design engineering observe the following principles:

- **Value.** Does our product add value and solve problems for our users?
- **Usability.** Can users figure out how to use it?
- **Feasibility.** Will engineers be able to build the features within the time, resources, and technology available?
- **Business Viability.** Will this solution benefit the business?

But before we dive into these principles, let's describe what design engineering is and how it's being practiced today.

Defining design engineering

Design engineering may not be well known in the product world yet, but we've been putting it into practice without really knowing it for a while now. It all started with a seemingly simple question: "Should designers code?" This has stirred up heated debates on social media and conferences for at least the past ten years. I believe there is a place for designers who code, and a place for designers who don't. Those who don't are able to push the boundaries of the design without feeling limited by potential constraints imposed by the code. Those who do are able to understand what's feasible, and can envision challenges that might arise from the design. When these two types of designers join forces, they are able to prototype ideas quickly and come up with innovative solutions.

Simply put, design engineering is where solutions are created at the intersection of design and development. It's not only about solving problems for customers; it's also about improving the design and engineering process, making a space for communication and collaboration, and building a great experience. Or to phrase it another way: It's a question of building the right thing versus building the thing right.

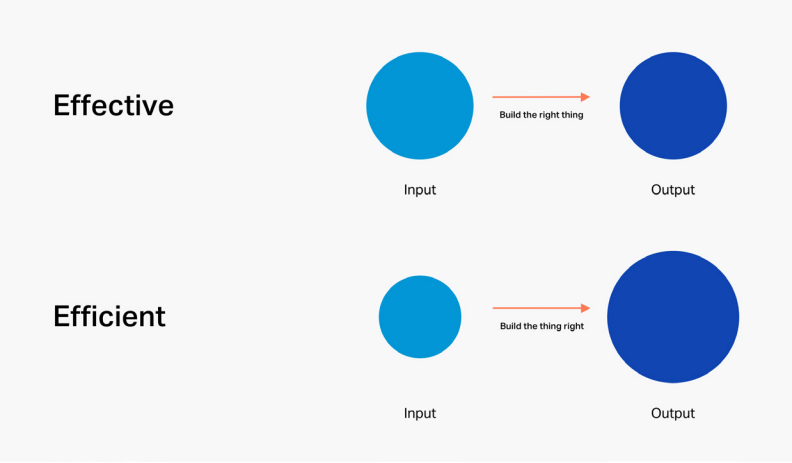


Figure 2-1: There can be tension between building the right thing (effectiveness) and building the thing right (efficiency). Ideally, design engineers accomplish both.

On one end of the spectrum, designers strive for pixel-perfect mockups and beautiful interfaces. On the other, engineers endeavor to architect systems and optimize for patterns. In the middle are designers and engineers concerned with how those two approaches intersect. We reach this happy medium with rapid prototyping and creating experiments to test usability with real data. The idea is to learn fast and apply our findings to the solutions.

But it's hard to go fast without having a solid foundation for building solutions. That's why the other important part of design engineering is *tooling*. Choosing the right tools for

the job depends on the people and processes within your organization. It's important to adopt tools that have a clear purpose and are intuitive to use. Spending tons of time figuring out how to operate a tool while developing a product defeats the purpose. Chosen wisely, the best tools allow teams to prototype ideas fast and come up with innovative solutions.

Value

In life and especially in business, we need to communicate the value an initiative will bring to get buy-in for it. In product design, we measure this by the customer's willingness to switch to and/or pay for a product. A business can improve these chances by creating a product that solves the customer's problem. From the customer's point of view, they have to believe the product is valuable enough to pay for and to spend time learning to use it. Products fail when they don't deliver valuable solutions to customers.

Design engineering brings product teams, stakeholders, and customers closer together. It achieves this, first, by understanding not only the business, but also its customers' problems, goals, motivations, and behaviors. Truly grasping all of these facets of the business allows product teams to create features that solve customers' problems, thereby making the product valuable to them. Research and data form the

foundation for creating a valuable product, and it is important for the product teams and stakeholders to understand and be aligned on them.

It is also crucial that product teams serve customers, not stakeholders. Stakeholders should provide the vision that product teams can use as a touchstone when building features. But if research conducted by the product teams diverges from the direction the stakeholders have laid out, the project usually follows the research.

Rather than a top-down approach, make room for a bottom-up approach. This accomplishes two things: First, it empowers product teams by giving them ownership over how to approach the problem. Second, it allows for a more user-centric design process. You can achieve this by listening to your customers' needs and building empathy. Here are some questions you might ask:

- Who are my typical customers?
- What are my customers' pain points?
- How does the product/feature I am making fit into their life?
- What are some of their habits or behaviors?
- What are their needs and goals?

- How does this product solve their problems?
- What are the most important features for the user?

Next, focus on the business goals. Consider these questions:

- How can these pain points be solved?
- Based on my user, how will the business make a profit?
- Who are my top three competitors?

The purpose of this exercise is to connect user needs with business goals. The closer those two areas are, the more likely the product will be valuable to customers and successful for the business. Once this is completed, research needs to be carried out to identify unknown challenges the user faces. There are plenty of methods for this, but the simplest approach is to interview existing customers. Ask them what they love about the product, what needs to be improved, and what other features might be useful. Once the interviews have been conducted, the next step is to identify trends. If enough customers are asking for a specific feature, this is something product teams might want to explore. These findings should be presented to stakeholders to discuss and prioritize future features. Once you have buy-in from them, the fun begins.

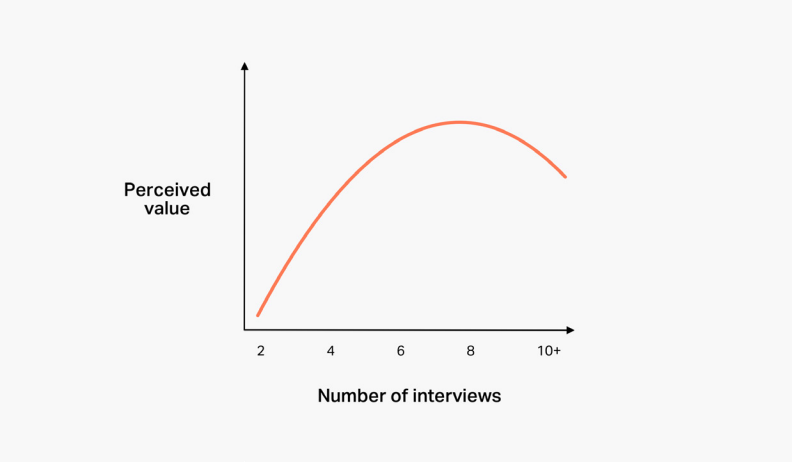


Figure 2-2: This graph shows the correlation between the number of interviews and perceived value.

Design process

Although it's a good idea to pair a design engineer with a product designer so that the design engineer can focus on interaction design and prototyping, it's important for design engineers to have a comprehensive, hands-on understanding of the design process. My own design process has evolved significantly over the past couple of years. I started out as a print designer, which meant that I often kicked off a project with a design brief and then gathered inspiration before sketching out solutions. As I made the shift to product design,

my process started to emulate the scientific method. It begins with a question about the challenges users face. I follow up with research to gain a better understanding of the problem. Next, I come up with a hypothesis and make an educated guess to answer the question. This is followed by creating experiments to execute and test the hypothesis. While conducting the experiments, I collect data on my observations and results. Creating experiments can take the form of *prototypes*, which are simulated concepts used to test and validate ideas.

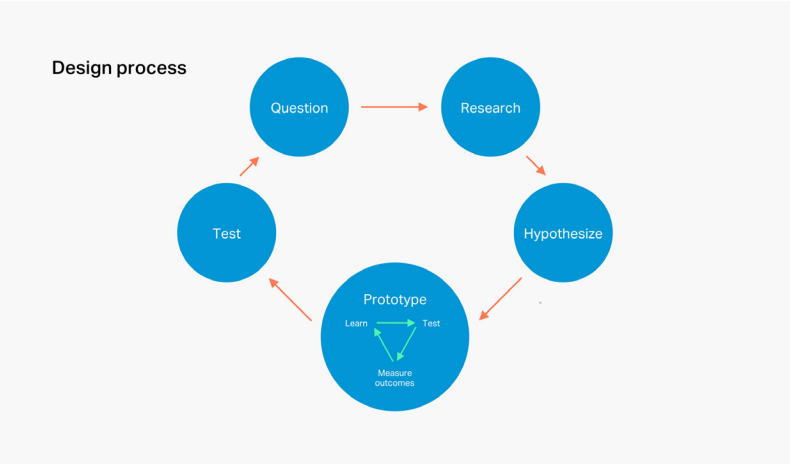


Figure 2-3: The iterative cycle of the design process.

In the next section, we'll dive more deeply into prototypes—their purpose, the different kinds, and when you should use one method over another. After the experimentation phrase,

you want to analyze the data to identify trends and come up with a conclusion. The last step is creating a *findings report* to communicate to the rest of the team what you've learned and explain whether or not your hypothesis was correct. This is an iterative process that can be done in a week or can span several months. If this is a new feature or product, one of the main goals is to create a *proof of concept* (POC) to test the hypothesis and demonstrate the feasibility of the idea to colleagues, stakeholders, and customers. The benefit of doing this is to make sure you are building the right thing before you invest resources to bring it to market.

Now that we understand the value principle in design technology and how we can use the scientific method to build empathy for customers and solve their problems, let's look at how prototyping and validation can improve the usability of features.

Usability

The next important principle is *usability*. It can be summed up with the following questions: How easy is it to use the product? How quickly can new customers learn it in order to complete an objective?

Usability has three main facets:

- **Effectiveness.** Does the product achieve users' intended goals?
- **Efficiency.** Can users accomplish a task with minimum effort?
- **Satisfaction.** Does the product make people feel good?

The more satisfied a customer is with a product, the more likely they are to recommend it. We all have one or two products we use that make us feel good every time we use them. When we have this emotional feeling about products, we become ambassadors for them by recommending them to others. Happiness increases the chances of having loyal and long-term customers. Because one of our main goals is to validate our hypotheses, usability plays a big role in design technology. The most common way to validate hypotheses is by building prototypes, which we touched on briefly in the last section. For me, prototyping is both an art and a science. We use a scientific approach to create products that not only bring value but also leave room for innovative ideas. We prototype to figure out which solutions work without investing a lot of time, resources, and money.

Before you start prototyping, it's helpful to nail down the purpose and goals. I usually write out a statement like this:

"The purpose of this prototype is to [explore || validate] the [problem] that [user || persona] faces. I believe we can solve this by [hypothesis]."

This sort of statement allows us to scope down the work and really focus on the objective and the outcomes we want. It's especially important for new features or products; sometimes starting from scratch can be very intimidating.

The prototype process

A prototype is an early concept that lets you quickly explore and validate ideas. There are many ways to prototype, but there are two strategies I always use: the *fuzzy front-end* (a.k.a. the design squiggle) and *branching exploration*.

My interpretation of the fuzzy front-end is that when we start a project, it's in an ambiguous state. There's some uncertainty about what the problem is, which direction to go in, which process to take to solve the problem. The reason I love the fuzzy front-end approach is that it lets me embrace the chaos in the beginning. I have a background in art, and for me one of the most stressful experiences is starting a new project. Since art is subjective and at times lacks constraints and boundaries, beginning can be paralyzing. It's the same with product design. At the start of my career, I used to have a lot of anxiety and

stress at the beginning of a project. Now, though, I tend to have more of a calm, excited feeling. There are a couple of reasons for this. One is that I now have a bunch of techniques and processes to rely when solving a problem, which gives me confidence; the other is that I've grown more comfortable with failing. The fuzzy front-end forges a path from uncertainty and fogginess to focus and clarity. The path itself is made up of research, prototyping, and design.

While the fuzzy front end is a great approach for thinking about prototyping at a macro level, *branching exploration* is a great practical process to rely on at a micro level. When using this approach, keep in mind that the first solution is almost always the worst idea. Why? Because designers often get very attached to the first solution due to the time and energy they've invested in it.

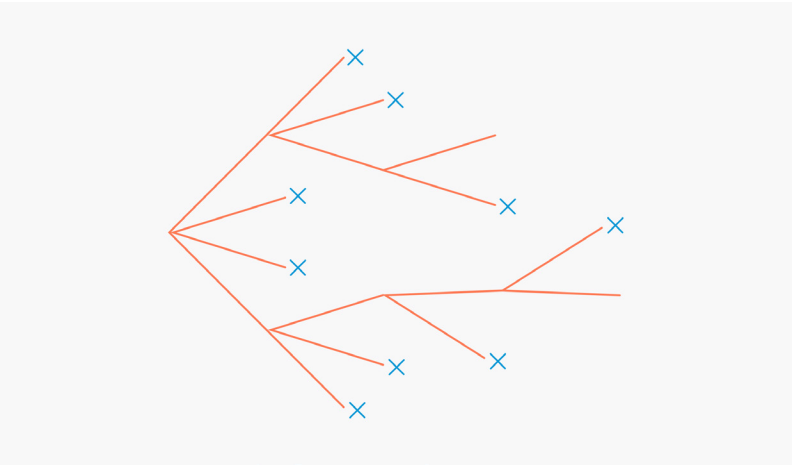


Figure 2-4: Branching exploration.

One technique that can allow designers to detach themselves from their work is to share early and often. Have a rough sketch you did on the back of a napkin and want to see if it's a good solution? Share it with your team and get their feedback. Some ideas may work; others may miss the mark. You want to be aware of those edge cases early in the process to be sure your design is going in the right direction.

The next technique consists in building solutions off one another. Say we create a feature and it doesn't work. Should we just trash it? Maybe, but a better approach would be to understand why the solution didn't work and then adjust it. This makes for a fast and iterative approach.

Now that we understand the micro and macro levels of the prototype process, we can move on to the three most common kinds of prototype and when to use one over the other.

The simplest way is a low-fidelity *paper prototype*. The materials you need for this method are pen, paper, and tape. Once you have the materials, people from the product team (designers, engineers, researchers) should get together and focus on solving a specific user story. The team members can use paper prototypes to communicate their ideas by cutting out pieces of paper to represent the UI of the feature.

Figure 2-5: Roll up your sleeves and get out some pens, paper, and tape. Lo-fi paper prototypes are a simple but effective way to communicate design ideas.

The next method is a *static prototype*, built with tools like InVision. Static prototypes are clickable, stitched-together static mockups that demonstrate the experience of features or workflow. They come in many variations, and their purpose is to solidify the workflow, visual design, and layout of the features.

Finally, there is the *interactive prototype*, which simulates the end experience. Interactive prototypes are built in code and incorporate mock or production data. Their purpose is to solidify the interaction design, validate solutions, and uncover any edge cases that the other methods might not catch.

So when would you choose one kind of prototype over the other? It all depends on where in the process you are, what you want to prototype, and what you want to learn from it. Imagine you've started on a new feature and the team is still unsure about possible solutions. In that case, the best prototyping method would be the one that improves your flexibility and speeds up ideation.

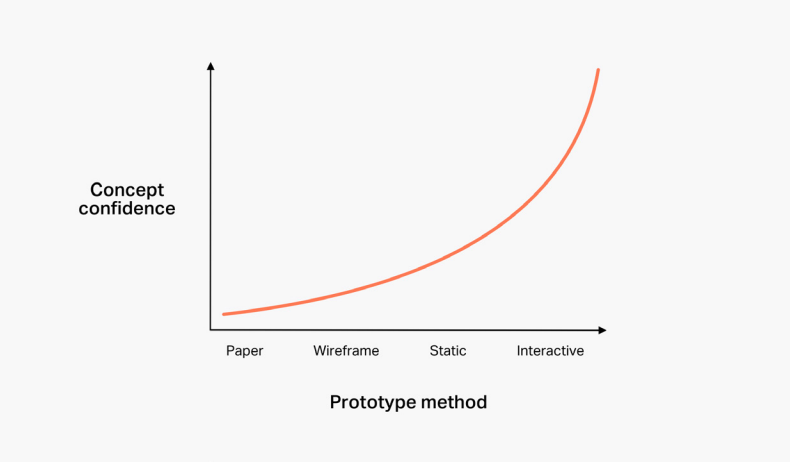


Figure 2-6: This graph shows the correlation between concept confidence and the prototype method.

Paper prototypes are a way for product managers, designers, stakeholders, and engineers to share and present ideas. You don't need design skills to participate; you just need to know how to draw lines, squares, and circles. Once the solutions start to weed themselves out and become more concrete, you can up the fidelity of the prototype.

While doing this, there might be a scenario where you have a couple of ideas you want to test with users. In this situation, you have to decide what you want to test. If you want to test the information architecture (IA) or workflow, go with a *static prototype*.

There are three varieties of static prototype. The first is a *low-fidelity static prototype*, which basically looks like a supercharged paper prototype. This method is great for iterating on the layout without getting into the details of the content. If you have an idea of the content or type of content you want, place it in the middle of the box. With this, you have enough to do some guerrilla testing to validate the IA and workflow.

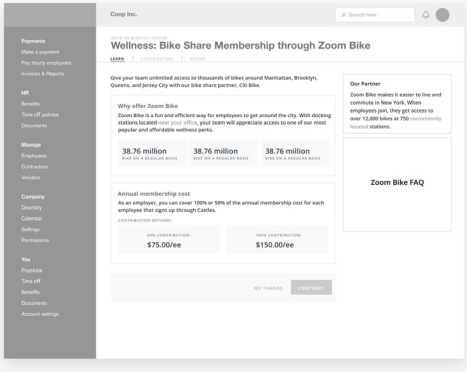


Figure 2-7: A low-fidelity clickable prototype.

Once that is in a good place, you can graduate to the *high-level static prototype*. With this technique, you add real content. I love this kind of prototype because it compartmentalizes the design into content and form. When using this method, it's a good idea to enlist a content strategist to help craft the

content. With this prototype, you can validate content and also the IA and workflow.

The next step isn't as clear as the previous ones. You can create a *static prototype* that looks like the end experience, or you can create an *interactive prototype* that demonstrates what the end experience will feel like. Ask yourself the following questions to help you decide which avenue to take:

- How much time do I have?
- Who is on my team?
- Is there anything else I would like to test and validate before the features get released?

Building interactive prototypes requires time and resources. If you've never done an interactive prototype before, the first one will at least take a sprint or two to set up. But once you get the hang of it, your setup time will go from a couple of days to a couple of hours. If you have four or more weeks before launch, then you definitely have enough time to create an interactive prototype—but whether or not you'll actually be able to do it hinges on your answer to the second question: Who is on your team?

Why is this so important? Let me tell you about a past job experience. I was working on a team with other designers;

we all were comfortable writing code and building interactive prototypes. There were many reasons why interactive prototypes were our main deliverables, but one was that our engineering team was remote (and in a different time zone). They needed everything about the design to be spelled out in a spec document. So rather than creating and maintaining mockups and spec documents, we consolidated them into an interactive prototype. This gave us the ability to design the whole experience from visual design to animation to interaction to responsiveness.

In another role, I was part of a cross-functional team that worked and sat together. I didn't have much time to create interactive prototypes since I was juggling multiple projects, but that was okay because I was working closely with the engineers. I was able to show them my designs pretty much every day, which allowed them to provide regular feedback and let me iterate quickly. Strengthening the relationship between designers and engineers will reduce the need for comprehensive documentation in favor of frequent interactions and working software, true to the agile method.

With a waterfall approach, design happens first, followed by development. The problem with this is that some edge cases can only be identified in the development stage—and it's always best to find those sooner rather than later. Working in parallel allows for a more collaborative process that includes

engineers, designers, and product working together from concept to completion.

Miriam Suzanne, cofounder of OddBird, shares how different levels of prototype fidelity are appropriate for different phases of the design process.

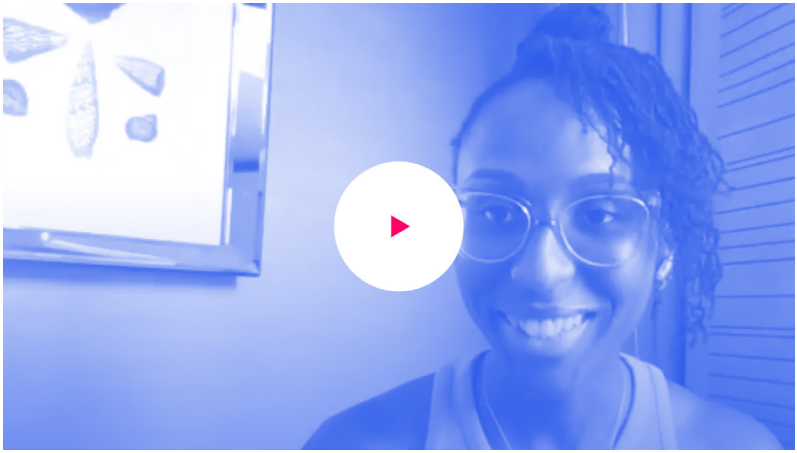
I prefer interactive prototypes because they let me explicitly design the whole experience. The best way to do this for web apps is to create *interactive prototypes*, which are built with HTML, CSS, and JavaScript. There are various ways to create them; it can be done either with vanilla HTML, CSS, and JavaScript or with a static site generator. I like static generators because they make it possible to create app-like websites that can output to static files, which can be uploaded for anyone to see. I recommend trying out a couple of them to see which one fits your team's style the best.

Front-end prototypes not only let you design a better experience; they're also an effective communication tool. In meetings, interactive prototypes give rise to more focused feedback sessions and allow anyone to test-drive the features. Another benefit is that they're a valuable tool for usability testing. In my experience, the closer the prototype is to the end experience, the better and more detailed feedback you'll get. You can also personalize the prototype with the user's data so they get the full experience. I'll discuss more about how to incorporate personalized data into your prototypes shortly. But first let's discuss the three different ways to build a front-end prototype.

The first variation consists of a mix of code and static assets like images. One scenario where I used this was when I was starting a project that included a section with data visualization. I didn't know what visualization I wanted to present there, so I created a mockup of one and added it to a front-end prototype. This allowed me to paint the picture without having to implement the details. During feedback sessions with stakeholders, we had a discussion about what sort of visualization might be valuable for the customers.

The second type uses mock data. With this front-end prototype, you will either incorporate static mock data or data that comes from an API. I love this kind of prototype because it's flexible and simplifies things by removing any concerns

around privacy breaches. Mock data can be used in prototypes without leaking real user data; this is something the QA team can provide for you.



Ire Aderinokun talks about pushing the fidelity of prototypes to better test products with users.

Finally, the third variation lets you prototype with real data. Here's another story for you: I was once working on a team that designed some line charts for a dashboard without exploring the data first. When we incorporated real data, the graph was flat for almost every beta user. This was a good lesson for me and it remains a good lesson for anyone building a data-heavy product. If you have some sort of visualization that tells a good story in static format, make sure you test it with real data. More often than not, adjustments will have to be made—and you

want to make those adjustments before the feature gets into users' hands!

Let's walk through how you can create an interactive prototype using a mock API:

1. **Pick a framework.** There are many front-end frameworks out there. I recommend choosing the one you and your team are most familiar with. I like Gatsby, which is an open-source static site generator built on React and GraphQL.

2. **Incorporate a pattern library.** Speed is essential when prototyping. This is especially true when building interactive prototypes. In the beginning, you want to spend less time focusing on the patterns and more time testing and building solutions. If you have one from your company, you can use that. If not, there are open-source pattern libraries like Semantic UI you can utilize.

3. **Import mock data.** If you don't have a mock API at your disposal, you can create your own using Google Sheets. The goal is to make a fetch request from the Google Sheets API to the interactive prototype so you can utilize the data in real time. For comprehensive directions on how to do this, check out the Node.js Quickstart guide.

4. **Start Prototyping.** You're finally ready to prototype. This is where we can use the branch-exploration technique to iterate on ideas. We are going to use Git to help us branch from one idea to another. A branch naming convention I use is [project]{feature}-{version}, which in the real world would look like this: [ads]portal-0.1.2.

5. **Deploy.** There are many ways to deploy a Gatsby site. My preferred method is using Netlify, which is a platform for hosting web projects. When using Netlify, every time you push to the dedicated git repo for your project or create a pull request, it provides a preview link to the prototype. This is what you can use when demoing to your team or stakeholders and for user testing with users.

Feasibility

Having a great user experience is cool, but the difference between ideas and products lies in the execution. I believe the role of designers is to create ideas that solve users' problems and push the boundaries toward innovation. Engineers take those ideas and bring them to life so customers can use them. I call this *building the right thing versus building the thing right*, which I talked about earlier in the chapter. First, you want to build the right features to solve the problem. The next challenge is to build those solutions right, so that they are

flexible and scalable. But how do you know if the engineers can build these features within the time and resources available to them?

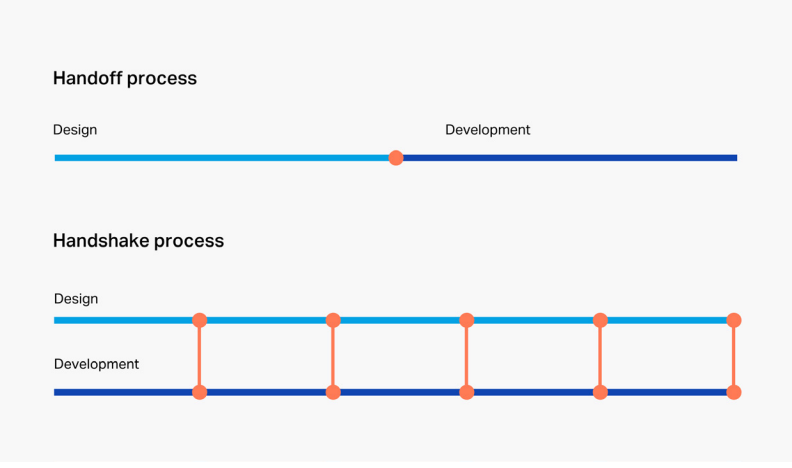


Figure 2-8: More flexible and pragmatic than a traditional handoff process, a handshake approach facilitates transparency and cross-functional collaboration.

Earlier, I talked about how designers and engineers should work in parallel (handshake) rather than sequentially (handoff).

The major benefit of a handshake approach is cross-functional collaboration and transparency.

During my time as a Product Designer at NASDAQ, the product design team would deliver an interactive prototype to the developers at the end of every sprint so that they could utilize it when building the application. This allowed us to communicate our design decisions and showcase the experience in a detailed format. And it allowed the developers to reuse our code and have only one artifact to reference while building the product. Another benefit is uncovering edge cases early and often. This is important because the earlier edge cases can be brought to light, the more options you have to address them. Inevitably, some edge cases won't be discovered until the feature is launched, and that's okay. But the number of edge cases that crop up after launch will be far fewer.

Sometimes it's hard to gauge what can be built until you start. This is where a *proof of concept* (POC) can be valuable. Creating POCs can be complicated, which means that this is an ideal place for design engineers, whose role is to stress-test design ideas, uncover edge cases, and provide realistic prototypes.

Now that we've talked about the feasibility principle and the role it plays in the product-development process, let's look at how business viability helps tie everything together.

Business viability

Business viability is about measuring the long-term profitability and sustainability of a product or feature. It plays an important part in design engineering because it's an outcome of one of the other principles: value and usability. If a feature is truly valuable to customers and is easy to use, then its viability will increase. Other factors to consider are *uniqueness* (to what extent the feature stands out in the market), *customers* (who the target audience is) and *competition* (who your competitors are and how you stand out from them). The goal is to create features that align with the business and increase market share. Ideally, you want to validate the business viability of a feature or product before it launches—but, as you may know, it doesn't always work like that.

It's hard to know whether something is successful or not unless you are keeping track of metrics. For products, we do that with *success metrics*. When you define success metrics for a feature or product, you start with the outcomes. To do that, you outline the pain points and problems customers face and list the outcomes that will occur when implementing the feature or product.

As an example, I was once working on a financial tool that was going to replace a fourteen-year-old product. A major pain point was that it was hard to find where certain pieces

of information and features lived. One of our metrics was to improve the IA so customers could perform these key tasks more efficiently than they could with the legacy workflow. Since we had different types of users, we had different outcomes for each of them. That's why if you find yourself in this situation, you want to conduct research that tells the story of the current pain points customers face. When the product launches, conclude your story with how it improves the customer workflow.

When creating success metrics, you'll probably want to start with a statement like this: "Increase signups of new customers." Then, once the product is launched, some sort of number should be included: "Increased signups of new customers by 30 percent." The best way to do this is by measuring your customers' behavior. Taking a snapshot of the usage and sentiments before and after the feature or product launches is a great way to measure success. These stories are what stakeholders want to hear from product teams. They've made an investment with people, money, and time, and they want to know what the return on their investment is. Being able to communicate this effectively will allow product teams to take bigger risks in the future.

In this chapter, we discussed the meaning of design engineering—how its core ideas provide guidance and key principles for product-development teams. The next chapter

will look more deeply at the challenges and opportunities afforded by a collaboration between design and engineering.

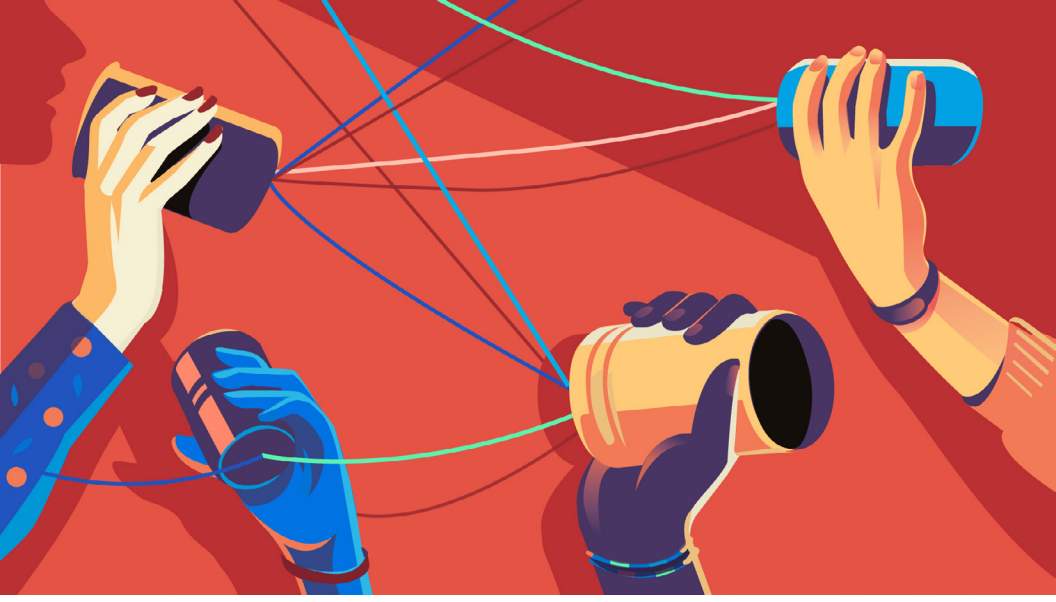
Further reading

Why Designers Should Never Go to a Meeting Without a Prototype

Prototyping Playbook by Matt Rothenberg

The Master Prototype by Atif Azam

The Design Squiggle



Chapter 3

Engineering Collaboration

A cognitive shift

By Natalya Shelburne

A cognitive shift

If collaboration between design and engineering were easy, everyone would be doing it and you wouldn't be reading this chapter. The hard truth is that the effort to facilitate effective workflows between disciplines is as real and challenging as the rest of our work. Fortunately, there are steps anyone—from an individual contributor to a team lead—can take to help create the conditions for the efficient exchange of ideas and empower people to do their best work. Those steps include shifting expectations, surfacing patterns, and building systems to facilitate collaboration.

Shifting expectations

Establishing collaborative workflows between design and engineering shouldn't just be an afterthought—it's an entirely new way of working. Good intentions aside, telling people they should simply "talk more" or "sit next to each other" is not a strategy. What's worse, it renders the effort of creating effective workflows completely invisible. Unfortunately, the consequences of this approach are usually only felt when it's too late—when the collaboration grinds to a halt because the person everyone was relying on to create channels for communication moves on. As an industry, we can and should

do better than leaving people to figure out collaboration on their own.

If things seem to run on your steam and fall apart as soon as you step away, then you have not built a system and are unsustainably performing invisible work.

Ire Aderinokun speaks to the evolving roles of designers and developers.

Full disclosure: I did that invisible work myself for years. My passion for both design and engineering meant I always volunteered to be the one to bridge the infamous gap between the two. How did I do it? The same way a lot of folks in tech do it: with their time and energy. Scope creep? Work late. Surprise feasibility issue? Close some tickets on the weekend. Notifications always on and mind always occupied. Like many,

I felt the guilt of stepping away, so I didn't—until I needed to go on parental leave. That's when I realized that my working methods needed to be formalized. Shifting my efforts to make my work visible was the first step to creating systems, tools, and workflows that set up collaboration between design and engineering for success. Let's take a closer look at the insight I gained and the steps I followed.

The translator

Whenever I hear people talk about "bridging the gap" between design and engineering, I like to say that the work really consists of translating between the two disciplines. Talking more won't do anything to improve the dynamic between a designer and an engineer if they are effectively speaking different languages. I don't mean to say that designers and engineers are so different that they can't communicate, or to resort to reductive right-brain/left-brain, creative-versus-logical stereotypes. A larger pattern of human behavior plays out between design and engineering.

I spent my graduate education learning how learning happens, how to encourage creativity, and how to develop potential into talent. In my classroom, I amassed a collection of interventions for when learning and productivity veer off course. If we look at it through the lens of human development, we see that

the "gap" between design and engineering is the result of a struggle to communicate effectively at the intersection of different mental models.

What are mental models?

People aren't computers. They don't just acquire knowledge line by line as they read it; instead, they organize patterns of thought in their minds and construct meaning in the form of mental models. When confronted with more information, they reinforce or adjust their mental model of how things work. That means prior knowledge is key. Every new bit of information has to interact with our existing mental models of how stuff works. What you learn first matters: It serves as the foundation upon which you build understanding. Our starting points inform our perspectives, which means that two individuals can have two very different experiences of the same event.

For example, someone who learns Haskell (a mathy and rigid language) first may find JavaScript (which is much more permissive) chaotic. Someone who starts out with JavaScript may find Haskell too limiting and opinionated. Designer learning CSS? The cascade is freedom and thinking declaratively works great! Programmer learning CSS? Scope all the things, get me some control flow in here, and oh my god the global cascade is unpredictable. Prior knowledge

may interfere with how we accept new information. An entire technology exists at the intersection of two different mental models, of design and engineering: CSS. The opinionated debates about it are a giveaway: There seems to be an even split of tech folks arguing fervently when any of the following questions are asked:

- Is CSS programming?
- Who should write/own CSS? Designers or software engineers?
- Is CSS broken or awesome?

CSS is over twenty years old. There is still no agreement on these questions precisely because of the clash of prior knowledge and differing mental models of the *people*, not of the technology itself. To an educator, these arguments are predictable.

Continuing to insist on your perspective will not serve you nearly as well as developing the ability to see from someone else's. Managing the complexity that emerges at the intersection of mental models—in this case, transforming visual communication into computer- and human-readable code—is exactly what makes CSS so incredible. CSS is made for programming design. It's a tool that serves both designers and engineers, and that's precisely why it is often the subject

of such debate. Intersections where people with different mental models need to work together tend to be loud and sometimes even hostile—but that's where learning happens.

The beginner trap we all fall into

In your work, the feeling of confidence and competence is wonderful. You're getting good at something and you've built a useful mental model of how it works. Being good at things feels good! Then you try something new, expecting it to work a certain way . . . and then it doesn't . . . and you don't understand why someone would make something so frustrating. That feels bad. We naturally seek to avoid that bad feeling. That bad feeling is *cognitive dissonance*—the frustration we feel when confronted with a different way of thinking. Cognitive dissonance is both critical to learning and, predictably, one of the biggest barriers to learning.

How so? Well, I can tell you that an educator expects an early learner to become upset at the first sign of their mental model being challenged. And odds are they will blame the tool/method/process—anything but their own understanding. That frustration, if reinforced, may cause them to retreat into their comfort zone and stay there. I'm not judging—all of us have felt cognitive dissonance at some point. It becomes a problem,

though, when the many platforms on the internet amplify these private frustrations and enable them to do harm publicly.

Poor reactions to cognitive dissonance can undermine both the newcomers in the community and the more experienced people venting their frustrations. Falling into this beginner trap limits people and creates arbitrary barriers. And that can lead to scenarios like this:

- The communication gap between design and development can feel unbreachably wide.
- Technologies at intersections of mental models, like CSS, keep being loudly debated.
- People in tech often get derailed and eventually siloed into narrow job roles.
- Too many people get trapped in patterns of below-average learning or gatekeeping.

Grim. So how do we move forward?

Un-silo yourself and others

The good news is that because these barriers are arbitrary and largely self-imposed, a lot can be done to overcome

them. But once again, if it were easy, everyone would already be doing it. The first step lies in increasing the flexibility of your own mind. How often have you been locked in a broken workflow where the only thing being traded back and forth was blame? To break toxic patterns of interaction, we need to un-silo ourselves and deliberately practice acquiring different mental models. If we don't practice the flexibility of other ways of thinking, we run the risk of becoming rigid and unable to collaborate.

If you've ever written a line of code, odds are you've heard this famous quote from Rear Admiral Grace Hopper: "The most dangerous phrase in the language is: We've always done it this way." The quote is often used to motivate people to embrace change. Unfortunately, shifting course, especially at scale, can feel even more dangerous. Simply put, because the cognitive dissonance of stepping outside of our comfort zones feels bad, we avoid it. Learning is uncomfortable, so creating the right circumstances to experience this discomfort safely is a way to make progress. Teachers do this every day in their classrooms. I did it in mine.

I designed a semester-long course called Modular Design Patterns with React for Harvard Extension School, with a focus on the intersection of design and development. Predictably, the curriculum attracted multidisciplinary professionals, from graphic designers to backend developers. On the first

day, I told them that collaboration would be a learning goal for us, but I didn't warn them about how, exactly, we would be approaching it.

A few weeks into the semester, once I get to know my students and assess their skills, I assigned a group project with defined roles similar to ones you would find in a small tech startup. The roles were clearly defined: Each person was directly responsible for a different aspect of building a prototype. However, every person in the group had to sign up for the role they were the "worst" at. After the initial panic subsided, they took on the group role that required them to shift into a different mental model of "how stuff should work" far outside of their comfort zone.

- The programmer with a decade of experience writing JavaScript became the designer responsible for the user experience.
- The SEO expert was now the team's Git expert. They suddenly had to make sure everyone was able to contribute code.
- The graphic designer morphed into the front-end developer responsible for the code quality and accessibility.

I created a situation where feeling cognitive dissonance and shifting mental models was not only safe, but encouraged.

The result? Students stretched their minds and shared a lot of laughs as they figured things out. Most importantly, at the end, many of them said that stepping into another role enabled them to better understand the impacts of their own work. They said they could see new ways to be better collaborators. They got better at their own work by doing someone else's. The word "empathy" found its way into most of the feedback.

Of course, it's likely not feasible to try something this extreme in the workplace. Pressing deadlines, social and professional reputation, and job security all raise the stakes tremendously. If your work environment doesn't feel like a safe place to make mistakes, learn from others, ask questions, or be a beginner, this sort of experiment will fall flat. Your design system implementation starts with making your work environment a safe space for experimentation.

Matt Rothenberg, principal front-end engineer at Clearbit, talks about the importance of setting aside time to experiment.

Tech is notorious for siloing people into narrow lanes in order to maximize speed and output. That kind of pressure is a creativity and collaboration killer, and yet we are still constantly sprinting. Collaboration requires a two-way flow of information between disciplines, and that means creating and tending to those channels of communication. If you are tasked with either opening or reopening channels of communication between design and engineering, I recommend starting with understanding. That means asking questions.

Surfacing patterns

Lead with questions

Too often, we make the mistake of assuming that opening the channels of communication means we have to speak louder and more if we want to be heard. Unfortunately, simply talking more won't do anything if we don't speak the same language as our interlocutors, or about the same things. Whether you've been part of a cohesive team collaborating and communicating effectively or a team struggling to sync up, asking more questions will help determine your systemic pain points and guide your next steps. Here are some questions you might ask:

- Where are your team's collaborative pain points? Is someone constantly having to pick up the slack? Is a single person a bottleneck for a workflow or process?
- Are unhealthy team dynamics emerging? Are designers and engineers increasingly feeling unheard or undervalued for their contributions?
- Do we frequently experience feasibility surprises and amass technical debt as a result? Do we find ourselves reinventing the wheel? Are we losing time solving solved problems and duplicating code and effort?

- Who owns the work? Who is accountable? Who reviews the code? How does a component get shared? How do we collaborate on ownership?
- Is our approach too waterfall-like? Are engineers unable to start working (prototyping, validating, scoping work) until designs are finalized? Are designers having to make decisions without having requirements set?
- Are design and UI in alignment? When design brand guidelines are updated, who makes sure those changes are correctly updated at the appropriate level without creating overrides or inconsistent implementation? How do we test that something is not missing?
- Are we having a lot of unnecessary, unproductive meetings? Are we throwing time at the problem of communication gaps?

If any of these questions hit home, it's time to take a deeper look at the patterns of dysfunction that have emerged. Every workplace and team is unique. In my case, I humbly had to realize that my default was to become a bottleneck. I was the expert, after all, and my prize was late nights, stressful deadlines, and the occasional frustration at anyone who disagreed. Fortunately, going on parental leave forced me to break that cycle before it repeated itself at my latest job. What felt like a huge risk at the time—making myself entirely

redundant and replaceable—turned out to be the best thing I've ever done. But it was far from easy.

Change starts within

First, I had to believe. We are told we can change the world through technology, but we somehow fail to translate that optimism to our own workplace situations. Instead, we keep lobbing the same old ineffective solutions back and forth: rearranging desks in a quixotic attempt to force collaboration through proximity, or piling meeting after meeting on the calendar to try to remediate ineffective communication. Shift your expectations, ask more questions, and believe a better way to work together is possible. We can change the way we work and interact to facilitate collaboration without relying on the interpersonal skills of individual contributors or managers to open and maintain effective channels of communication. We can introspect openly and honestly and identify our patterns of failure. We can create systems to support collaboration and engineer the conditions for an effective exchange of communication across disciplines.

Building systems to facilitate collaboration

Establishing conditions for growing effective design and front-end workflows means working with the people you have, not the people you wish you had—Starting with yourself. That's right, the old adage "do as I say, not as I do" is just as ineffective in the workplace as it is in other parts of life. Meaningful organizational change starts with you.

When pondering how to approach parental leave, my first impulse was to write down exactly how everyone should do things while I was gone. That would have been a mistake. Instead of writing overly specific documentation of "the right way to do things" that no individual who values their time would ever read, I decided to shift my efforts toward building systems to facilitate the transition between my mental model of how things worked and the mental models of my coworkers. I needed to find ways to enable my team to succeed at doing things their best ways.

For example, in my role, I translated between design assets and engineering implementation. I was a skill silo and I realized all too late that I didn't really teach anyone my working mental model. Oops. So I set up a tool to enable others to do the translation between design and engineering without having to be inside my head. A style guide and component library made our design patterns visible and created a structure to relay it to the application code: the components, the layout rules,

the colors, the functionality available. It felt wonderful to stop being a bottleneck and see a system help enable others.

Matt Rothenberg speaks about how pairing facilitates collaboration.

Since then, I have been obsessed with creating systems that help lower the barriers to contribution. When I'm not working solo, I shift my focus to thinking about what would enable my teammates to work best instead of what would be most comfortable for me. On the front end, for example, I embraced CSS modules despite my own preferences so that other programmers on the team would feel more confident about writing CSS. The module approach more closely matched their existing mental models of programming, and I no longer had to review every pull request that contained style changes because now any error would be scoped by default instead of remaining precariously global.

If you want to change how your team works together, changing how you work is mandatory. Fortunately, getting out of one's comfort zone is the best way to learn and grow; in our fast-paced world, that's a very good thing. Take a moment to reflect on how you work. How can you make the way you think, decide, and create more visible?

Avoid wrong turns by introducing prototyping tools or preview environments in your application to allow your ideas to be visible and validated earlier. Before your team spirals into an argument, agree on what makes a pattern by documenting a framework for how to decide whether to design a new button or reuse an existing one. Save yourself hours of clearing up misunderstandings by aligning on vocabulary, so that, say, design and engineering are talking about the same things when they say "component" or "layout." Such strategies are not just a "nice to have" but a better way to work. Teams that prioritize this work will outperform teams that don't.

How do you say no?

I don't.

When tackling a business problem, designers and engineers should be equals on the same team. Collaboration means we speak the same language and use our respective strengths to

tackle an issue, instead of getting locked in a back-and-forth cycle of shutting each other down. If I find myself starting to argue, defend, or justify my work, I check myself and ask what is really going on. Am I threatening someone's status or comfort zone? Do I feel like they're threatening mine? Am I frustrated because I don't feel understood? What am I missing? And so on. Usually, I reach the same conclusion: I need to put in the work to surface patterns of misunderstanding, adjust my workflows, make my mental model more visible, and create systems to facilitate collaboration. In my case, a style guide, a design system, and clear agreement on conventions free me from constantly feeling like I have to course-correct by saying "No." Constant vigilance is not a strategy; building a system to facilitate collaboration is.

If you're the one consistently hearing "No," however, let me be clear: There's no cure for a gatekeeper. No amount of effort will enable collaboration with someone determined to preserve their status quo as a bottleneck. You can't change someone who doesn't want to change, and you can't collaborate effectively with someone who doesn't value your opinion. Fortunately, in my experience, more people want to find a better way to work together than to champion the status quo. Find allies. Go around the gatekeepers together. What does success look like?

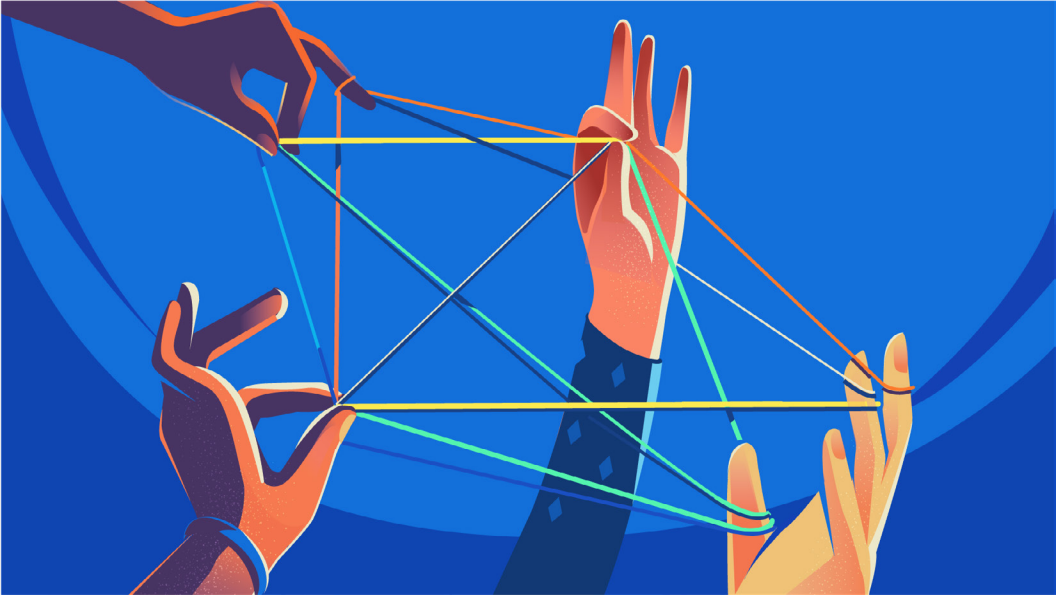
It looks something like this: Information flows effectively between the different disciplines on your team. People can take leaves or vacations without worrying that everything will fall apart without them. A culture of learning and flexibility prevails as people adjust their workflows. Your product ships, the company succeeds, and there are good vibes all around.

What if you do nothing?

Enjoy having a million meetings placed on your calendar in an attempt to help unblock collaboration and get people talking. But understand that simply throwing your and other people's time at the problem is not an effective strategy. If simply "getting people in the same room" or "sitting next to each other" worked, there would be no such thing as a gap between design and engineering, and design engineering would not be emerging as a line of work. Collaboration does not happen by accident. Like the products we work on, it takes design and engineering.

Castles in the sand

Nothing lasts. The work we do is like a castle in the sand: The tide will always come in and wash it away. And that's okay. The work is worthwhile. Remember: Engineering collaboration and facilitating the translation between disciplines is all about the people.



Chapter 4

Design Engineering Organizational Models

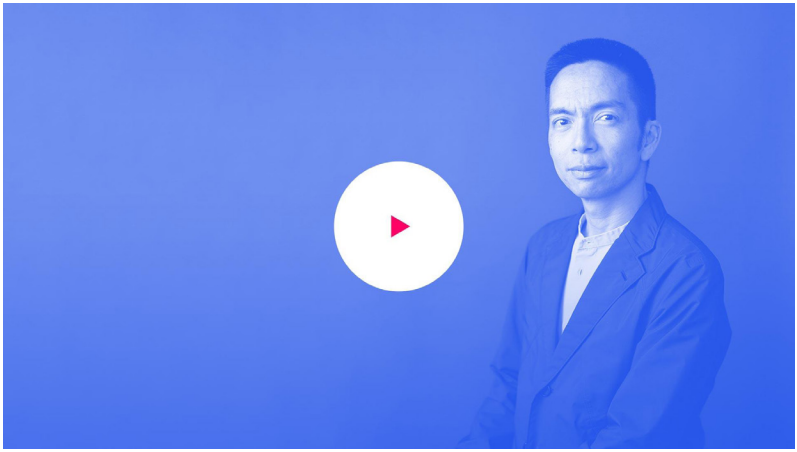
Finding the right fit

By Kim Williams

The success of a product design function hinges on how it is set up organizationally to align and partner with product and engineering. Reorgs and shifts in business priorities mean that change is constant in any company. Understanding how a function is able to adapt to any change while retaining its culture is key to resilience.

There is no "right" answer to organizational structure. Rather, as John Maeda put it in an interview for the Design Leadership Handbook, there are two fundamental questions you need to ask:

- Do we have the right organizational structure now, for our stage of growth and the current market conditions?
- If we know where we want to be in the future, will we be ready in time with the right structure?



As John Maeda describes in this video, organizational structures change as teams grow and priorities shift.

I've had a chance to lead some teams through organizational change, and I've also learned a lot from some of my Indeed colleagues, like Eddie Lou (Senior Director of Design Engineering) and Anna Vu (UX Manager, previously Manager of Design Engineering). I'll share some of these perspectives here, and we'll explore topics and questions like these:

- *Centralized model:* What are the pros and cons of every team member of the function rolling up to one leader?
- *Decentralized model:* What are the pros and cons of team members being distributed across the organization and reporting to different functions and multiple leaders?

- *Hybrid model:* What does a blended approach between centralized and decentralized structures look like?
- *Reporting structure:* What are the advantages and disadvantages of reporting to different functional leaders?

Although these topics have been covered by Peter Merholz, Kristin Skinner, and others in the past for design teams, org design for design engineering teams has some specific nuances that we'll discuss in this chapter. We place a focus on the dimensions of resourcing, product knowledge, core competencies, career paths, and leadership.

Product, design, and engineering leaders often focus on technical debt within a company as a key driving force for improving speed and strategic capability. However, entrepreneur Steve Blank asserts organizational debt is like technical debt—but worse.

Steve Blank coined the term organizational debt in 2015 and defined it as: all the people/culture compromises made to "just get it done" in the early stages of a startup.

This thinking can easily be applied to early stages of a new discipline, and/or refactoring an existing discipline. Blank suggests that nothing kills a company faster than organizational debt.

A discipline with a solid culture defined with a vision—and supported by a team that understands their greater purpose and receives investment in their growth—makes for an organization with a competitive advantage.

Centralized model

If your design engineering team is still nascent, or recently formed, you are likely using a centralized model of organization. A centralized structure keeps all design engineers on the same team in a shared space (whether physical or virtual), with everyone reporting up to one leader. Some refer to this as a "center of excellence," since there's a central leader that implements standards for the discipline and other teams come to the centralized group for their services—much in the same way a client would approach an agency.

Like any organizational structure, a centralized model has its pros and cons. We'll evaluate these in the following sections, before we move on to the decentralized model.

Resourcing

One advantage of having a centralized structure is that the team's resources are more fluid and adaptable to change. If a

pressing business problem arises, the team can be deployed to meet the needs of the challenge. Because of the top-down structure, leaders have access to their team's talent and skill set, and can make quick assessments on where to deploy the right skills to solve the right problems. Having one large team under central leadership also produces some economies of scale.

On the other hand, this model can create a certain amount of tension between the core teams and the centralized one, who may be viewed as "outsiders." This can lead to political conflict, and requires strong leaders who are proactive about building relationships with other parts of the organization to counteract it.

Product knowledge

A centralized team will likely spend its time with many different verticals within an organization, so they will have the advantage of a broad knowledge base of the different products in the company, and can work across areas that might previously have been siloed. Not surprisingly, the downside of this arrangement is that there isn't usually an opportunity to develop deep product knowledge in any one area. Onboarding is not in-depth, as product engagements may vary from short- to long-term.

Core competencies

Because centralized teams are constantly being drawn into new products with different teams, they are typically more comfortable with ambiguity and are adept at asking the right questions to get to a decision quickly when evaluating the possible paths a product or strategic initiative might take. These teams are often inherently curious and able to ramp up quickly into new situations. They're also able to pivot readily based on the learnings they get from prototypes and user tests.

But centralized teams don't often stay in any one product area for a great length of time, so they won't develop as rigorous an understanding for the inner workings of a given product team, including their technical challenges and operational constraints. Additionally, as centralized teams can be perceived as "outsiders," there's a lot more effort required to gain permission to deliver on strategic initiatives.

Career paths

Because centralized teams report up to a single leader, there is a clear owner and decision-maker for career growth. Team members have an opportunity to build a strong foundation around a skills matrix and calibrations, which can evolve over time as the team scales. The natural hierarchy of a centralized

team also opens up opportunities for different types of career growth, for both manager and individual contributor tracks.

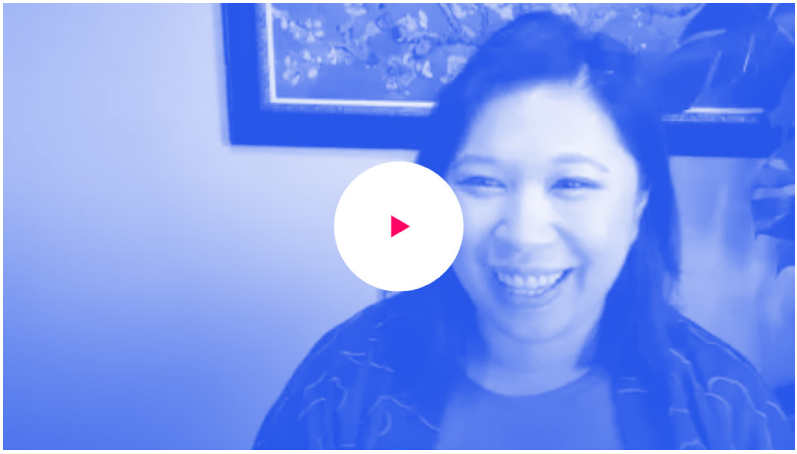
The strength imparted by having a single leader can also be a potential weakness, though. Team members will have limited exposure to a variety of leadership styles from cross-functional partners, as the functional leader is a dominant influence on their career.

Leadership

As discussed above, having a single leader for the centralized team has the potential to confer strong benefits within this model. In addition to career paths, leaders can set consistent discipline standards through a unified vision that holistically meets business needs. Leaders have an entrepreneurial spirit, as they consistently champion their team's discipline and show a return on investment. The foundations of the team will evolve over time as a company scales, so leaders can keep a running health check to see where things are going well and where more investment in resources (tools, training, personnel) need to be made.

One of the challenges for this model is that product leaders may have varying points of view on discipline standards, particularly as it pertains to design engineering team growth

and development. For example, a product team may want to engage design engineering for back-end software engineering work that may not align with the design engineers' core competencies. Of course, the team is always eager to help, but the role of the central leader is one that ensures standards for career paths.



Anna Vu, UX manager and former manager of design engineering at Indeed, speaks about the evolving organizational structure of her team.

Decentralized model

Decentralized teams insert design engineers within cross-functional teams. These teams tend to have more autonomy

to make key decisions, and communication is more immediate across engineering, product, and design. This type of model is common with companies that have a general manager (GM) structure.

Because the relative autonomy of a decentralized model is attractive, it might be tempting to jump right into this structure from the start. But a decentralized model won't work for many teams unless the right foundations are in place. Let's take a closer look at those foundations, along with the other pros and cons of this model.

Resourcing

Unlike within a centralized model, when design engineers are inserted in cross-functional teams, they will naturally feel a connection to that team, and accordingly are less likely to be treated as "outsiders"—at least over time, as relationships and trust develop. But that can come at a cost to flexibility; once assigned to their dedicated teams, it is harder for an organization to realign resources if business priorities change or unexpected challenges arise.

Product knowledge

Once they are placed in their dedicated teams, design engineers in a decentralized model will gain a depth of knowledge about their assigned product area. They'll have access to in-depth documentation of the product requirements, or will likely be part of the team driving the development of that documentation if it doesn't already exist. Onboarding happens once within the product team upon hire. The downside of this model is that team members can become siloed, as they are generally experts in their own product area(s). Additionally, to gain deep expertise, onboarding takes a longer time.

Core competencies

Decentralized teams tend to follow a more structured approach to product development. Because of their in-depth product knowledge and proximity to their engineering and product counterparts, they don't need to make as many assumptions about technical and operational constraints.

The difficulty with this scenario is that the team might be more challenged by ambiguous problems and strategic initiatives, especially if they are locked into an Agile workflow that doesn't leave room for research, exploration, and experimentation.

They also may be operating without the clear vision that a central discipline leader can provide.

Career paths

Because of the cross-functional partnership in a decentralized model, design engineers have a chance to be evaluated by their engineering and product peers, which may also prompt them to learn more about these disciplines and contribute to the unity within the product team. Team members benefit from more exposure to a variety of leaders and career advice.

But career paths may be the most critical foundational element that needs to be in place before a team transitions to a decentralized model. Design engineers in this scenario report to another functional partner that has different priorities and standards, which may or may not be relevant for a member of the design engineering team's career trajectory. If clear career path foundations aren't established, each team might try to solve the same problems and come up with career development plans that are inconsistent across teams.

Leadership

Leaders within decentralized teams benefit most from the autonomy to make key decisions. Demonstrating return on investment happens cross-functionally with this model. Leaders are closer to the conversation with their counterparts and can advocate for investment. Without a centralized leader at the top of the organization, however, leaders in these teams can suffer from a lack of a unified vision for their function, and fragmentation can happen when top leaders don't see a need to align.

Hybrid model

A third model of organization is becoming more common. This is a hybrid model, which blends multiple organizational models and runs different models in parallel. Let's take a look at a hybrid model to consider the intersection of centralized and decentralized models:

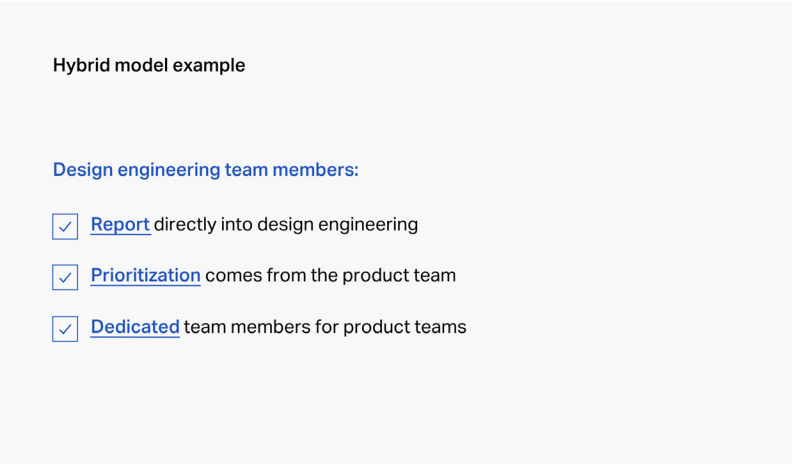


Figure 4-1: Would a hybrid model work for your org? What might that look like for you?

One hybrid structure includes a centralized team with dedicated team members within a cross-functional product team. The centralized leader is a functional lead elevating the craft of the discipline and the health of the organization. Dedicated team members of design engineering are getting their day-to-day priorities from UX leads within a product team.

This hybrid model gets all the advantages of both centralized and decentralized models, and all the disadvantages of a centralized model.

The reason that this model can be adopted by a wide range of organizations from small to large is because it is more

flexible, practical, and scales to different business needs quite easily. There's also a great deal of trust that is imbued from the product teams knowing that they get dedicated team members.

This is one expression of a hybrid model we explored early on at Indeed. Peter Merholz and Kristin Skinner have been documenting organizational approaches through years of work in their book, *Org Design for Design Orgs*, as well as their respective consultancies, talks, and workshops. They penned the hybrid model "Centralized Partnership" and provided additional considerations for why a blended approach is most effective. You can think through how a hybrid model could work for your design engineering discipline and what might be the right approach for the functional org that design engineering will be reporting into. Let's take a look at reporting structures in more detail.

Reporting structure

Just as it's important to understand the pros and cons of your organizational structure, it can be helpful to know the advantages and disadvantages of your reporting structure, even though you or your leaders may have less control over it. There are essentially three ways that a design organization will report up into leadership, with some variability in how

the design engineering team(s) will report in a product-led organization:

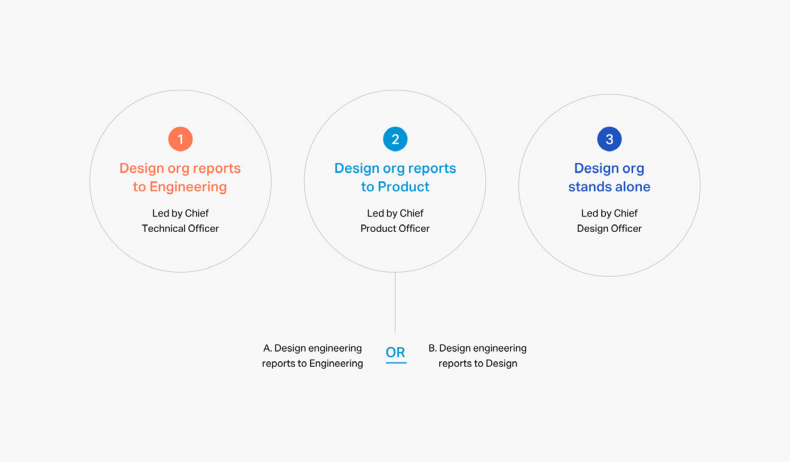


Figure 4-2: Three ways a design organization may report into leadership and options for how design engineering fits in.

I chatted with Eddie Lou to get his perspective on how these different reporting structures affect the design engineering team specifically, with a focus on CPO and CDO-led design organizations.

When design engineering reports to engineering in a product-led organization, there tends to be a bias toward full-stack engineers, and front-end engineers that enjoy doing design engineering work can get lost in the shuffle. If they don't have the skills to build databases or work on security, they will often

be left with few options for career growth or promotion. Unless the engineering leader is adept at building a practice around design engineers, it will be challenging for the individuals to feel supported. The result may be that the business does not get to leverage the best from the talent.

The situation can be more advantageous if design engineering reports up to design, in a product-led organization. In this scenario, designers often look to design engineers to unblock the design team and drive the execution of their ideas forward. The natural collaboration between the two functions, and the fact that design leaders are more likely to invest in a robust career ladder for their design engineers, makes talent retention more likely.

Figure 4-3: The EPD structure is like that of a three-legged stool—engineering, product, and design need to be equal to achieve balance in product design. Art adapted from Don Norman.

Finally, there is a scenario where the entire design org, including design engineering, reports up to a Chief Design Officer. In many ways, this structure is ideal, because it gives engineering, product, and design an equal voice at the executive level, and can result in more balanced priorities for the entire product team. Organizations like Airbnb use this structure to, in the words of Alex Schleifer (CDO at Airbnb), make sure that "each function can grow in parallel at a proper ratio as the broader organization scales."

Review pros and cons of team structures in this spreadsheet.

Conclusion

Circling back to John Maeda's advice at the start of this chapter, the organizational structure that your design engineering team adopts will be dependent on your stage of growth and the current market conditions. The structure you have now might not be the structure you need a year from now, and you should start planning the evolution of your teams early to give yourself the time needed for big organizational changes. This is especially true if you're growing fast.

As Lynsey Thornton, VP of User Experience at Shopify, said in the Design Better Podcast, "We find that every org structure we create is out of date quickly as we've grown so fast. We have to plan for 18 months out to try to predict where we'll be and what we'll need."

Additionally, at a company-wide level with rapid growth, there is often a shift to a GM model, which can take on decentralized organizational models. If you take the time to plan for these changes, and understand the potential advantages and shortcomings of each scenario, your teams will feel more supported and will be more likely to stick with you through the changes ahead.

In the next chapter, we'll review all the foundations of building and scaling a design engineering practice. These foundations will help you transition from one organizational model to the next.

Further reading

Org Design for Design Orgs by Peter Merholz and Kristin Skinner

Organizational Debt is Like Technical Debt – But Worse by Steve Blank

Your Guide to Resourcing Discussions as a Design Manager by Jehad Affoneh

Chapter 5

Design Engineering Leadership

Sage advice for scaling a practice

By Kim Williams

Good leadership is integral to starting new or scaling existing disciplines. In addition to learning from my own successes and failures, I've had the good fortune to learn from some of the best leaders in the industry. I qualify "best" as highly effective leaders who deliver excellent business outcomes by fostering a healthy and inclusive work environment. Eddie Lou, a senior director of design engineering at Indeed, is one such leader. Throughout this chapter, I'll share answers to questions I've asked Eddie over the years, as well as what I've been able to observe of his leadership style up close. Other leaders and colleagues you'll hear from include Anna Vu, Indeed's UX manager, and Mica Mercé, the UX director. We'll pass along what we have learned for building a design engineering practice from the ground up, as well as sage advice for scaling a practice.

Stephanie Rewis, director of engineering, design systems framework at Salesforce, tells the story of her journey from individual contributor to leader of her team and some of the challenges she faced.

Over the past decade, there has been an increased focus on the full-stack engineer. At some companies, this has caused a shift in career-growth opportunities for front-end engineers. Eddie noticed that most of his design engineering positions were being filled by front-end engineers who felt they were being forced into a role without a career path at their previous companies. (Of course, some companies do foster a vibrant community for front-end engineers.) Design engineering creates a space for talent at the intersection of design and code to feel empowered, regardless of the reporting structure.

Leadership for design engineering is all about bridging the gap between design and engineering, creating a sense of belonging by building and fostering a team and creating strategic relationships throughout the company.

Minding the gap

To echo something Natalya said in Chapter 3, sometimes the distance between design and engineering can feel insurmountable. I'd like to highlight some of the ways this distance manifests itself at an organizational level. For design engineering leadership, bridging the gap between the disciplines of design and engineering gets to the heart of real cross-functional pain points in service of efficiency and collaboration.

Misaligned priorities and incentive structures

How design and engineering disciplines evaluate each team member through levels, expectations of roles, and calibrations (performance review cycles) has a significant impact on how the disciplines work together. Because design is often

held accountable for quality, designers may feel frustrated about the quality of what ships. And since engineering is often held accountable for speed or velocity, engineers may feel frustrated about proposals that cause ship dates to slip. A natural tension arises from the fact that design and engineering approach quality and speed through different lenses. These priorities and values can be amplified when calibrations are connected to incentives. How can we embrace the values of quality and speed that are both required to meet the expectations of our users?

Long-term versus short-term delivery

Long-term strategic work focuses on new products, features, and functionality that are critical to innovation. That's what's required for transforming products and creating competitive advantages for businesses. Designers and engineers alike may not always be comfortable with the varying degrees of ambiguity involved in long-term strategic work.

Short-term strategic work focuses on innovation within the context of iteration or refinement of existing features for products while creating immediate business impact. Not surprisingly, therefore, short-term strategy is often what shows up on the product roadmap that controls ship dates. How can we craft a culture of innovation that delivers on the

best ideas from engineers, designers, product managers, researchers, and content strategists?

Design engineering's remit

Design engineering addresses cross-functional pain points through a remit that focuses on increasing both quality and speed. Design engineering spans a broad scope ranging from validation to implementation. For many organizations, these scopes together fit into one role. Throughout this book, we've referenced one discipline (design engineering) and one role (design engineer) for the sake of consistency and simplicity. In this chapter, we'll take a closer look at Indeed, which has two dedicated roles within design engineering: design technologists and UI engineers. Indeed started out focusing primarily on validation (the realm of design technologists) and over time evolved to include implementation (the province of UI engineers). Here's a quick snapshot of these nuanced scopes:

- **Design technology (validation)** engages in rapid prototyping to accelerate long-term and short-term strategies. This does the heavy lifting of storytelling, documenting, and jump-starting the future state of new products or features. Design technologists contribute to

all aspects of the design to validate the work through user research or A/B testing.

- **UI engineering (implementation)** delivers production-level code, as defined by the engineering team, that is both performant and beautiful. UI engineers also focus on building custom frameworks and tooling.

Design engineering is a discipline that has the ability to see into an organization's code—its strengths and weaknesses—to create sophisticated prototypes and tools to accelerate innovation. Crafting your team to directly meet the needs of your organization sets your team up for success. Before you build your team, consider some of the pain points you're experiencing cross-functionally. What are the strategic goals you're trying to accomplish? What tooling does your company need to deliver on quality and speed successfully? What scope within the realm of design engineering would be most helpful for the maturity of the org—design technology, UI engineering, or a blend of both?

Anna Vu, UX manager and former manager of design engineering at Indeed, talks about how caring for her team led her to take on a leadership role.

Building your team

There are two core aspects to building a team. The first is attracting talent (recruiting): hiring engineering talent that's passionate about solving UX challenges. The second is keeping talent (career growth): proactively advocating for skill development in both design and engineering, and crafting individual-contributor and people-management leadership opportunities.

Let's take a look at each of these areas in more detail.

Attracting talent

What makes design engineering unique is that the pool of talent broadens as you search among two distinct areas of expertise: designers wanting to focus on interactivity and engineers wanting to focus on user experience. Recruitment for these areas of expertise typically focuses on portfolios and, from a technical standpoint, evaluating whether code is performant to the millisecond.

Eddie's principal ethos for hiring is this: "If you're passionate, you can learn anything." Whether hiring for a design technologist or a UI engineer, Eddie's team places a singular focus on a passion for solving problems. The success of the team hinges on this simple and singular focus, which allows them to build an organization of doers.

Technologies come and go, but the willingness to be curious is a lasting key trait for all members of a thriving design engineering organization.

Soft and hard skills

The team uses an evaluation criterion that blends soft and hard skills (Fig 5.1). This is how they've unearthed the problem-

solvers—the self-starters with the skills, knowledge, and perspective to know there is always more than one answer to any given problem. The soft skills are evaluated via manager feedback, which considers how an individual will contribute to company culture, behaviors observed throughout the interview process, points of view around ethics, learning style, organizational style, capacity to work autonomously to deliver on their role, and, most importantly, an eagerness to solve problems.

These softer skills become apparent throughout the evaluation of hard skills demonstrated in technical, design, design technology, and UI engineering assessments. The assessments are made through the lens of design exercises or existing portfolio pieces. During an onsite, an interviewer can choose a whiteboard exercise, a previous code evaluation, or a review of an exercise provided.

The goal of the assessments for design technologists is not necessarily to see performance or final code, but rather to examine whether everything was built for rapid iteration. This is important when we consider the importance of speed for user-research sessions and product launches. Additionally, the team is eager to see how candidates bring their designs to life. For UI engineering, the goal is to see the code quality and how complete solutions are, including whether or not the code is scalable and easy to read and follow. Additionally, without

prompting, do candidates talk about how they made their code more accessible or discuss internationalization needs?

While the design technologists have an additional assessment to evaluate prototyping techniques, what remains true for both design technologists and UI engineers is the emphasis on the user. It's important for the team to understand how candidates demonstrate their understanding of user needs and how they evaluate the success of their work from the user's perspective.

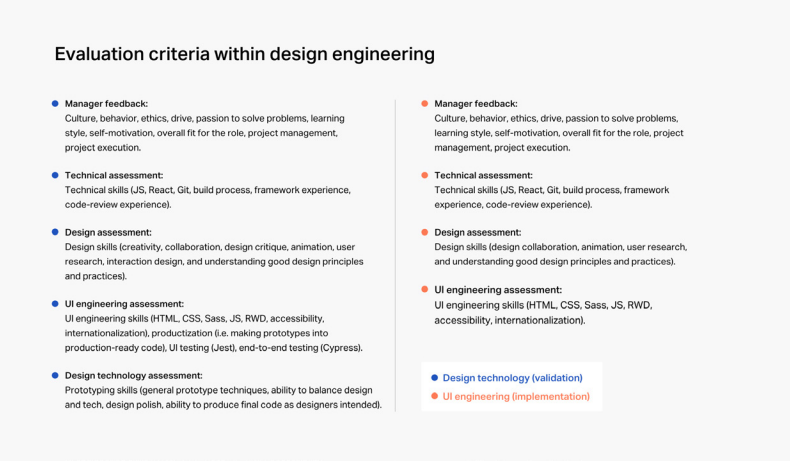


Figure 5-1: These evaluation criteria blend soft and hard skills to surface problem-solvers.

Hiring for passion and curiosity

Most interview processes for design technologists are quite standard and what you would expect for any role within a company. They include a résumé screen, a phone screen, a take-home exercise, a 1:1 onsite, and a debrief. Admittedly, there has been some controversy about whether or not design exercises are a good practice. But Eddie's team has crafted an approach that is agnostic to their work and allows candidates to share their strengths while also having a bit of fun.

The exercise is a game (Fig 5.2). It's pretty bare-bones—just simple circles. When you create a game, you generate problems and solutions in a world that doesn't exist. A game is a great way to unleash creativity, passion, and curiosity. Eddie, who identifies as an introvert, admits that "people like me may not interview well." He continues: "Sometimes nerves get the best of us; however, how a candidate lights up around solutions is undeniable." Design exercises like this help level the playing field by reducing bias: Someone's personality and the way they show up aren't a factor in a design exercise.

The game is the same for both design technologists and UI engineers, but the evaluation differs according to the aforementioned evaluation criteria:

- **Design technologists (validation):** How does the candidate address the game dynamic? How does the candidate approach the game from a prototyping perspective? How easy is it to change or update any aspect of the game, making it faster or slower and updating the design? How does the candidate express themselves and add their own flavor to the game, with an eye to user experience? When she was interviewing, Christine Ma, a UI engineer at Indeed who started out as a design technologist, submitted a game about a meteorite hitting the earth. Instead of dots falling from top to bottom, which is the standard wireframe of the game, she made the dots fall diagonally. There was a race against the clock to prevent the meteorite from hitting the earth. Christine's willingness to add a narrative and introduce a better user experience was sublime.
- **UI engineers (implementation):** How does the game demonstrate candidates' excellence with regard to code quality and coding practice? Is their solution scalable? How complete is the solution? Has the candidate successfully resisted the urge to overengineer the game by not adding any unnecessary technologies? Have they tested the game to make sure everything is useful and performant?

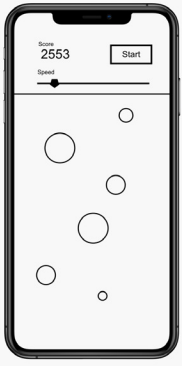


Figure 5-2: The Design Exercise: a simple game to showcase creativity, passion, and curiosity.

Eddie's team gives everyone a minimum of two weekends to complete the exercise, for an estimated five-hour duration. They want to be considerate of everyone's time and effort.

However you define your evaluation criteria, complement your search with exercises that reach both spectrums of talent within design engineering and allow you to easily gauge where a candidate's focus and passions lie.

Keeping talent

Okay, so you've invested in getting the best design and engineering talent. Now, here are some ways to be proactive around helping each team member grow while building a community for the practice.

Career growth in design and engineering

Design engineering is still a new path. It's Eddie's vision that all team members feel confident they are growing and learning in their core interest areas and are able to stay competitive within our industry with whatever they want to do next—whether that's accepting a design or engineering role, or perhaps stepping into a senior design engineering role.

The design engineering skills matrix

The team developed a tool called the Design Engineering Skills Matrix to track training, career paths, and growth across soft and hard skills. The tool is *qualitative*: Managers build relationships with each team member to identify both explicit, demonstrated interests and implicit interests, and apply ratings to discern trends over time (Fig 5.3).

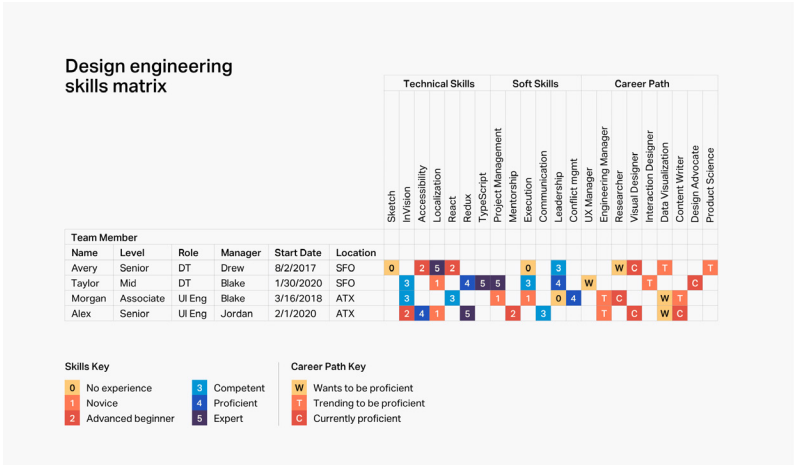


Figure 5-3: The Design Engineering Skills Matrix for investing in team growth and development.

This tracker is not meant for ranking and is never used that way. The ratings aren't strict and are considered pretty fluid. The tool offers a means for managers to work with each team member on their path to mastering their skills and interests. Managers are able to note opportunities to establish mentorship within the community by pairing those who are interested in growing in various areas with those who have the relevant expertise.

What makes the design engineering community at Indeed unique is that most of the management and leadership came from varying backgrounds and tech companies and grew with the organization; they were promoted from within through a

diligent effort to actively tap into the best each team member has to offer. When team members demonstrate or are trending toward mastery, they step into leadership as individual contributors or people managers.

Rubric

If you've been through the process of creating ladders and rubrics within an organization, you know how complex it can be. Anna Vu helms this effort at Indeed and has used the following approach. For a design engineering organization, be inclusive of UX and engineering requirements and consider these factors:

- *Alignment:* Use the same rubric structure that reflects your reporting to ensure both the definitions of success and ways the team are incentivized are directly connected to the parent org. Strategically add components for alignment with a partner org. If your org reports to UX, use that rubric as a foundation and modify it to be inclusive of engineering requirements. If your org reports to Engineering, use that rubric as a foundation and modify it to be inclusive of UX requirements.
- *Skill:* Create two related and distinct versions of the rubric for design technologists and UI engineers. Give design

technologists more UX requirements and UI engineers more engineering requirements.

- *Leadership:* Create a clear path and journey to leadership for individual contributors and people managers.

It is important for the design engineering community to feel valued and have a clear career path forward within an organization. Be thoughtful and creative with the different ways you invest in how a team member can achieve mastery in their craft; the flexibility to transition from one role to the next (from design technologist to UI engineer and vice versa); and the ability to transition to leadership (individual contributor and people manager).

Building relationships

To build and foster a team, and establish strategic relationships throughout a company, you need to be able to rely on each team member to show up consistently. Whether starting a new discipline or looking for ways to scale your existing discipline, thoughtful approaches to building relationships foster openness and trust.

Stephanie Rewis talks about how finding support with your colleagues can help overcome imposter syndrome.

A cornerstone for building a new discipline is culture. From the beginning, Eddie's team has operated as a unit. Eddie puts it this way: "Because design engineering is a new discipline, it was important to us to band together to deliver value and demonstrate our capabilities." A team-first mentality means democratically coming together to figure out the best way to get things done. As a unified body, the team understands the importance of one another's roles and appreciates that no individual is greater than the whole; they operate as one. Creating a sense of oneness and pride as a team helps create lasting bonds both within the team and with strategic partners.

Let's take a look at some of the ways to build relationships within a culture.

Rhythms and rituals

Manager's weekly meeting

Weekly manager meetings are key for growing leadership within a new practice. Eddie's team uses a simple structure of *topics, top-of-mind*, and *team health* for this meeting. "Topics" covers any product- or project-related work a manager would like to share status on and/or request support on. "Top-of-mind" is open-ended and doesn't include status; rather, it focuses on what matters most for the managers themselves or for their team members that week. No subject is off limits. Lastly, "team health" provides status on the health of the team. Red is critical; yellow is moderate; green is stable or good. Managers provide context for team health and requests, if any.

Worldwide meeting

Every two months, a rotating manager at Indeed hosts "Design Engineering Worldwide," a time for the entire global community to come together through video hangouts. The host schedules

the event and coordinates with other managers and team members to share demos of recent project work, welcome new members, and celebrate big milestones. Three or four of the demos typically span multiple locations featuring new prototypes or tools built. There's a sense of pride in seeing all of the work and ideas. It's an opportunity to recognize the broader impact of the practice across the company and proactively seek efficiencies and collaboration. Additionally, opening up the meeting to product and engineering leaders yields trust and transparency for the work.

Awards

Twice a year, two Indeed team members host the Design Engineering Awards ceremony. This is an organic event that was spearheaded by the culture and community. The community self-organizes to vote and nominate winners across many categories, from MVP to Best Invention, with handmade trophies and a red carpet to boot.

Figure 5-4: Images from Indeed's Design Engineering Awards ceremony.

Communicating results

Newsletters

Mica Mercé emphasizes how important it is for teams to have a consistent pulse around their value and contributions. At Indeed, she started UX newsletters showcasing user research, direct quotes from users, the latest tests and experiments, and upcoming milestones for the team. What began as a UX newsletter morphed into a collaboration between project management and UX, in which they crafted the content for

a holistic and user-centered narrative around the health of products.

Inspired by this approach, Eddie's team put together a monthly design engineering newsletter for transparency and to keep UX, product, and engineering teams up to date with each department's newest members, accomplishments, and latest prototypes and builds. Having this consistent touchstone increases confidence with stakeholders and investors in the organization.

Library

Setting up a design engineering library was a pivotal moment for Eddie's team. This library is accessible to everyone in the company and contains all of the prototypes and builds the team has created for every product area. It is a living showcase of the capabilities of the team to drive innovation, dream up new frameworks, and show what's possible. Putting in the work to add to this library requires discipline, but it's well worth the effort.

Conclusion

Stephanie Rewis speaks about how her varied life experiences have affected her perspectives as a manager.

There are no shortcuts to building strategic relationships within your company. Everything we've covered represents a long-term investment in a culture that sets you up for success through trust and openness. Flourishing organizations require leaders who won't shy away from the disciplined effort of investing in the health of the relationships within their team and with their cross-functional partners.

Coda

A Design Engineering Manifesto

A blueprint for your organization

By Eddie Lou

A new world

If you've made it this far, you know that design engineers merge two disciplines that are often separated. But we hope that if this book has taught you anything, it's that design engineers aren't "unicorns"—jack-of-all-trades-master-of-none generalists. No, they specialize in solving the very particular problems that emerge at the locus of the intersection of design and development. Prototyping, tooling, design systems—each of these realms present precise challenges that require unique skills, passions, and priorities. Design engineers address these challenges with speed and flexibility, an innovative spirit, and a commitment to serving their users and colleagues.

We'd like to leave you with a blueprint that will guide you as you go back out into the world and apply this book's lessons to your own teams and organizations. A manifesto, if you will. Take it, hack it, and make it your own.

It's possible that you've been putting many of these principles to use without knowing it, or without being able to pin a precise name on this new discipline of creative crossover. It's called design engineering, and this is what it does.

A Design Engineering Manifesto

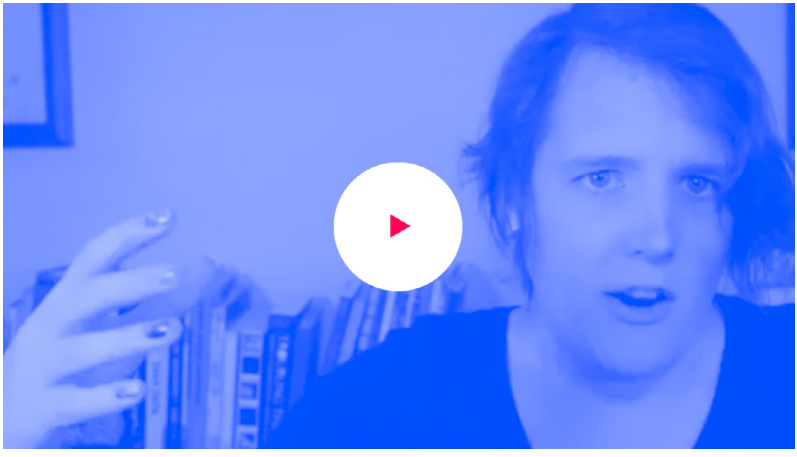
- **Design engineering delivers solutions** to UX problems, to design implementation problems, to development inefficiencies.
- **Design engineering bridges gaps** between UX and engineering, design and development, creative vision and technical implementation. It fosters better collaboration and builds a more cohesive user experience.
- **Design engineering innovates.** It explores, tests, and validates new concepts early. It takes risks. It makes leaps. It runs experiments. Design engineers deliver experiences that enable teams to learn faster and make better design decisions. They make wild ideas clickable, testable, and tangible.
- **Design engineering creates tools** to find solutions that work right now, and will work the next time and the time after that.
- **Design engineering builds fast and tests fast.** Design engineers embed with teams to enhance their capacity. They maintain flexibility in process and priorities to go where urgent needs lie.

- **Design engineering supports both design and development.** We are allies and advocates, moving things forward, together.
- **Design engineering is about flexibility.** It prioritizes meeting the need over enforcing a rigid process.
- **Design engineering champions craft.** And dreaming and doing. Refining. Shaping. Shipping. Serving real needs. Raising the bar. Innovating, implementing, and improving.

Designers who code, developers who design

Big tech companies often separate the people who design things from the people who build them. Designers determine exactly how everything should look and flow, and then they hand their work over to engineers, who figure out how to make the design functional and performant.

But things are changing. Expertise is evolving. Needs are shifting. New tools make front-end development a more accessible skill. Designers are learning to find the beauty in code; engineers are coming to appreciate the technical challenges of impeccable UX.



Miriam Suzanne, cofounder of Oddbird, talks about their process, how they often pair designers with developers, and how they work through edge cases together.

Interactive experiences now require increasingly sophisticated prototypes earlier in the design process. Consumers expect digital products to be simpler to use, even as they perform more complex tasks. Design systems require designers and engineers to work together to ship usable components, not static guidelines. So forward-thinking organizations are finding ways to foster collaboration across the old boundaries.

That's where design engineering comes in.

As users come to expect more polished, intuitive experiences, and as design becomes increasingly central to business success, tech companies can't prioritize speed and scale at the expense of quality and human relationships. They have to

operate with an appreciation of the whole experience, and to build teams that are incentivized to deliver on that goal.

Hybrid thinking

As we've reiterated throughout this book, design engineers bring both engineering skills and design skills to the table. They understand front-end programming, UX, and design. They care about the integrity of their code and the quality of the product's look and feel. Through testing and research, they grasp how it all works together.

Design engineers live and breathe accessibility, responsive web design, performance best practices, internationalization/localization techniques, native app development, and modern front-end frameworks. Although they focus on core UI development (HTML, CSS, Javascript), they never hesitate to learn new concepts in order to build better experiences.

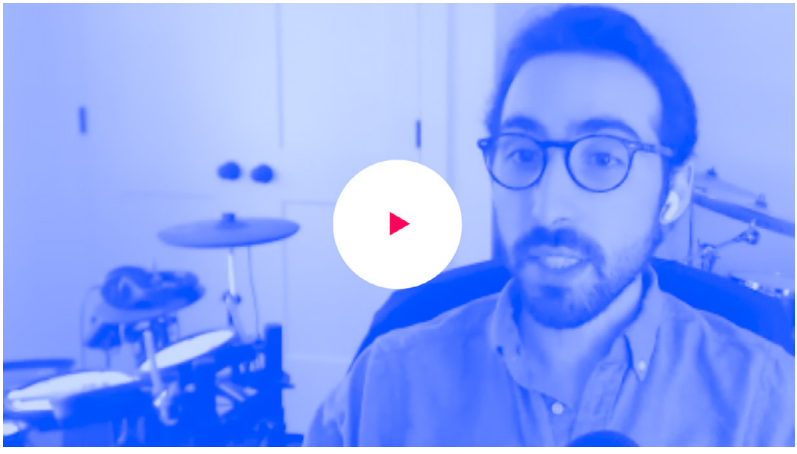
They have the ability to guide and contribute to all phases of the design process: wireframes, clickthroughs, mocks, and prototypes at all levels of fidelity. They have a UX sensibility and a solid knowledge of user experience and interaction design.

But perhaps the most important element design engineers contribute is mindset.

Design engineers are tinkerers who are naturally curious. They want to build things collaboratively and learn new skills along the way. They not only solve the problem at hand, but also create tools to help solve similar problems in the future. They care as much about users as they do about code. They're confident enough to take risks and try new things, but humble enough to ask questions and learn from others. They don't shy away from uncertainty or ambiguity, and they're not afraid to learn from being wrong and trying things that don't work.

Fluid, flexible, tactical

As a discipline, design engineering is unusually flexible. It isn't constrained by a rigid process or a single way of working. A design engineering team's time and resources aren't tied to narrowly defined product goals the way most engineering teams' are. Design engineers can move around quickly, coming in to address a problem as it arises, and proceed to the next problem once the first one is solved. They can change direction as the need evolves without upsetting a quarterly master plan. They don't have to tend to a deep backlog or ongoing maintenance; they put their attention where the immediate needs are.



Matt Rothenberg, principal front-end engineer at Clearbit, talks about prototype fidelity and how to use it to assess the feasibility of a new product or feature.

As long as they're making new things, solving new problems, simplifying processes for product teams, and advancing design, design engineers are on track. Their primary goal is to be helpful. They measure their success by the success of the teams they support. "Is the design system advancing? Are tools improving? Are we staying nimble? Are we moving fast? Are we fostering innovation? Are we introducing new ways of working, collaborating, testing? Are we offering a better experience to our users?" These are the questions design engineers ask themselves when gauging success and impact.

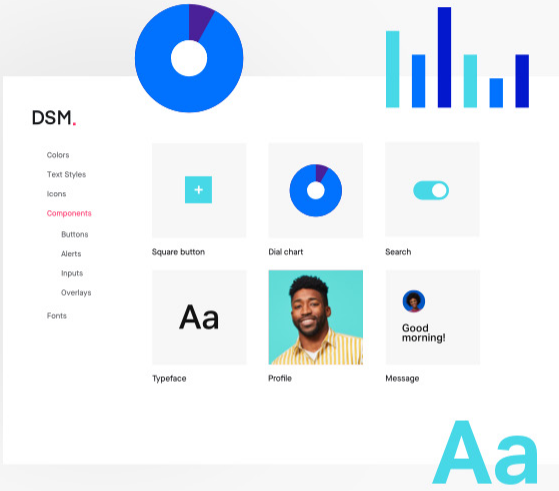
Design engineers aren't just consultants, swooping in with opinions and then disappearing. And they don't just close

tickets other teams toss their way. They're in the trenches with those teams, functioning as extended members of them. Design engineers innovate and get things done. They impact the bottom line. They're full-spectrum, carrying out bold experiments today and shipping tomorrow.

Venture bravely into the creative, nimble space where design and engineering intersect, and start producing your best work.

InVision

Design System Manager.



Connect design and code so teams can work smarter, faster, and more in sync.

SEE DSM IN ACTION