Deborah Foley

Design Systems · Platform Strategy · Design Leadership

Seventeen years building design systems, developer platforms, and the governance that keeps them alive at scale — at Amazon, JPMorgan Chase, and Goldman Sachs. Five case studies, oldest to newest. They are the same problem: encode what a system means precisely enough that people you will never meet can build on it correctly. Only the consumer has changed.

Seattle, WA / Remote · foley-design.com

CONTENTS

01 · GOLDMAN SACHS · WEALTH MANAGEMENT

Re-architecting a design system already in production

Hired specifically to re-architect the system and build the component library.

ROLE

Design System Lead Consultant, Technology Manager, CSS Expert (via Sapient). Partnered with the Lead Architect, Product Owner, and Application Tech Lead. Trained and managed two design teams — in-house and vendor — and 20 front-end developers, in-house and offshore.

THE CONSTRAINT

The Wealth Management application was two years into production and struggling: poor performance, long load times, and a persistent tail of visual UI bugs. The cause was organizational, not technical. As the design and

engineering teams grew, each new design team was designing its own "improved" version of components that already existed, and each development team was solutioning those same components in custom variations. The result was bloat — in the CSS, in the file structure, and in everyone's head.

Rewriting the application was not on the table. The system had to be re-architected underneath work that was already shipping.

THE DECISION

Fix the architecture, not the symptoms — and deliberately under-engineer it. Atomic Design applied to both folder and file structure. Sass with explicitly defined design tokens. Object-Oriented CSS and BEM for encapsulation and reusability, replacing generic HTML5 styling that was driving specificity and processing cost. Utility classes to create consistency through constraint rather than through review.

The judgment call that mattered was where to stop. Say yes to Sass and tokens, but do not over-engineer the code to the point where it cannot be read: it needs to be understood three years from now by a team that was not there when it was written. A system that requires its authors to be present has not solved the problem it was built for.

RESULTS

- **50% reduction in CSS**
- **85% fewer UI bugs**
- **45% improvement in delivery velocity**

ADOPTION

Re-architecture only holds if people consume it. I designed and built a component library website, sent weekly new-component release communications, and ran trainings and workshops teaching developers to consume and extend the library rather than fork it. Weekly sessions with designers worked through new component rationale, inconsistencies, and alternative responsive designs — the standing forum where exceptions got argued and either absorbed or refused.

WHAT IT PROVED

The constraint is the deliverable. The measurable wins did not come from better components; they came from removing the ability to invent new ones casually.

02 · JPMORGAN CHASE · CROSS CHANNELS

Governing a design system across a 450-person org

Manhattan Design System — scaling a system when the organization outgrows every model for running it.

ROLE

Product Design Manager, Cross Channels. Partnered with the Head of Digital Customer Experience and the Tech Lead of Design Systems. Team: one system designer, two UX designers, an illustrator, a motion designer, a content strategist, and a developer.

THE CONSTRAINT

In my five years at Chase Digital Technology, the design org grew from 75 people to 450 as a centralized team. Every model for running a design system has a size at which it breaks, and we crossed several of those thresholds inside one tenure.

I have worked all three of the standard models, and each fails differently:

- **Design system leads per team, meeting in a forum.** Good for a handful of teams. Breaks when there are too many cooks — sessions turn brutal and decisions stop landing on time.
- **Centralized Center of Excellence.** Good for auditing and for getting started. Becomes the bottleneck on every decision, and suppresses innovation because only a small group decides.
- **Embedded consultants where everyone contributes.** Scales to many teams — the central team consults, supports, runs office hours, and maintains versioning and documentation. Requires strong system thinkers with genuine diplomacy, and keeping that team motivated is the ongoing cost.

DISCOVERY

The Foundation team ran generative testing across every product team — designers and developers both — to find what was actually working. The pain points clustered into four areas: design process (accessibility, interactions and developer code missing from patterns; documentation scattered across several systems; no clear approvals path), development (no open dialog with lines of business, no understanding of handoff, no quick-fix route), tribe support (no place to showcase work or archive past projects; duplicated work across tribes), and communication (designers needing support but not knowing how to reach the Foundation team).

THE DECISION

At 450 people, governance is the product. We moved the system from a library to an institution: design tokens, voice and tone, and masterbrand guidelines defined as a single source; a component library carrying live components and consumable code alongside the design language; content management support so documentation could be maintained without an engineer; and a centralized communication hub integrating the system with the other applications teams already used.

Two mechanisms did the real work. **Design system briefs** gave anyone a structured way to propose a change — problem and solution stated, alternatives explored and explained — so contribution was open but disciplined. A **governance board**, owned collectively by the product teams consuming the system rather than by the design org, decided what entered it. Ownership sat with the consumers, which is what stopped the central team becoming the bottleneck.

WHAT IT PROVED

Past a certain scale you are not designing components, you are designing the decision process that produces them. The system that survives is the one whose users own it.

03 · JPMORGAN CHASE · DEVELOPER ECOSYSTEMS

Three developer portals, built from nothing

2018–2021. Started with one other designer; ended as the firm's API storefront.

ROLE

Head of Developer Ecosystems. Led product strategy for three portals — JPM Developer, Chase Developer, and the Merchant Services Developer Center — and managed a cross-functional team of UX designers, researchers, content strategists, and technical writers. I started with one other designer and no existing product. I was the design function until I built one.

THE CONSTRAINT

Five distinct audiences had to be served by one system: external developers, internal API developers, project managers, customer service, and technical writers. Underneath that sat three different access models — private APIs for internal employees, partner APIs gated behind a trusted-client relationship and a manual approval process, and public APIs for third-party developers on full self-service. Each tier had its own trust boundary, and the same interface had to hold all three.

DISCOVERY

Dozens of developer interviews to establish what practitioners considered best-in-class, plus a UX audit of banking and fintech competitors and of the reference portals outside finance. Nine developers walked through their own API selection and implementation process, then through the prototype, before and after registration and test-case verification.

The finding that shaped everything: developers are smart, busy, and savvy. They are not browsing. They arrive to find one API that solves one problem, and every step between arrival and a working key is attrition.

THE DECISION

Build one API store — a single application serving two brands, Chase Developer and JPM Developer, rather than separate products per line of business. And sequence it deliberately: the API publishing application first, the developer-facing experience second. The publishing tool was the unglamorous half and the reason the catalog could grow without the design team as a dependency.

The 2018 MVP was a landing page, a catalog, and documentation. Competitive analysis and the interviews fed a recommended roadmap; I facilitated the ideation workshop that aligned every stakeholder on what a developer portal actually needed to be. From 2019 to 2021 it grew into a large catalog of internal and external APIs with enterprise tooling, a support system, and a publishing system behind it.

RESULTS

- **668 internal APIs** published to the platform — firm-wide adoption of infrastructure that began with two people
- **31 partners** and 20 external APIs onboarded through the governed access tier
- **DevPortal Awards special mention** for visual design and accessibility
- Fully responsive mobile experience delivered as part of the GA release

WHAT IT PROVED

A developer portal is not a website with documentation on it. It is a distribution channel with a trust model, and the design problem is the tiering, not the landing page.

04 · JPMORGAN CHASE · CONSUMER

An architecture that outlived its brief

Security Center became the COVID-19 crisis hub, then Help & Support, then a platform serving marketing.

ROLE

Led product strategy. Managed a cross-functional team of designers, content strategists, product management, and research. Defined the opportunities, facilitated the workshops that aligned stakeholders across the bank, and recommended the roadmap. I was still running the developer portals when Chase moved me into the Consumer space to broaden the portfolio.

THE CONSTRAINT

In 2018, security was becoming a differentiator for brand trust in banking. Chase's security features were buried in Profile & Settings, and the analytics were unambiguous: customers were not finding them. The harder problem surfaced in testing — users did not expect security features from a bank at all, so discovery alone would not fix it. The design had to educate before it could engage.

DISCOVERY AND VALIDATION

A ten-part design sprint workshop took the group from jobs-to-be-done and competitive best-in-class through design studio sketching and voting, sitemap, poker planning of technology estimates, and a roadmap built on a customer value matrix. Card UI concepts were built around education and the customer's own data, then A/B tested. Eye tracking decided between the two finalists — Right Rail and Split Screen — on where attention actually landed rather than on which the room preferred.

THE DECISION

The decision that mattered was not the layout. It was building the hub as a modular, template-driven architecture backed by content management, rather than as a designed page. Feature cards, a value proposition banner, a knowledge center, personalization slots, and legal callouts were all defined as components a content owner could recompose without a design or engineering cycle. At the time this was a harder sell than a fixed layout and slower to build.

ACT TWO: COVID-19

In March 2020 branches and call centers were overwhelmed and the team was asked to stand up a communication hub. We did not design one. We repointed the Security Center architecture — **16 days from ideation to delivery, with more than 20 content updates over the following two months**, at a time when the guidance changed weekly. I proposed the decoupled content-management platform that made that update cadence possible without a release.

ACT THREE: THE PLATFORM

The same architecture then became the Help & Support hub, and eventually a platform serving marketing as well. That pushed Chase to think in modular systems, templates, and content management for reuse and searchability across the estate — an argument the crisis had proven rather than one I had to make in a deck. I later took on the aggregation area, replacing fintech screen-scraping of Chase properties with governed security APIs that gave customers explicit control over who could reach their data.

RESULTS

— **JD Power 2020: #1 in Security and #1 in Web Channels** for U.S. online banking

- **3M+ hits** on Security Center, with some applications reaching 90% engagement
- **1.5M web unique users and 210K mobile** on the COVID-19 hub; 398K customers reached payment assistance
- **CSAT improved 2%** and held steady through the pandemic
- #2 in Business Insider's Security & Reliability ranking; called best-in-class by Global Banking Magazine

WHAT IT PROVED

A system specified well enough gets consumed by uses nobody scoped it for. That is the whole argument for treating design systems as infrastructure rather than as deliverables — and it is the same argument I am making now with different tools.

05 · AMAZON · CORE UX, DEVICES AND SERVICES

Making a design system machine-legible

Current work. AI-directed design systems tooling — architecture, validation, and the round trip back to design.

ROLE

I lead design systems and platform strategy for first-, second-, and third-party enablement within CORE UX. The role runs across two disciplines by design: I manage designers and I manage programs — the processes, the pipeline, and the partnerships across several partner organizations. In practice that means owning the Amazon Developer B2B experience for app subscription submissions on Alexa devices, running the Fire TV component pipeline and the toolkit our teams build in, and building the operational tooling those teams depend on.

It is a two-sided problem in the same shape as the Appstore marketplace work I owned previously: first-party teams, second-party partners, and third-party manufacturers all need the same system to hold, while each needs to bend it differently. When third parties customize your components, you cannot review every downstream implementation. The system has to encode its own rules, or it does not hold.

THE PROBLEM

AI did not remove the bottleneck in design production. It moved it. Generating a first version — a screen, a component, a page of markup — is now nearly free. Reconciling that version back into a real system is not. Teams add prototyping capacity and end up with more artifacts requiring refinement, in code and in design files, than they had before. The visible symptom is velocity that does not convert: more output, the same shipping rate, and a rework queue that grows.

The common response is to buy more generation, which makes it worse, because generation was never the constraint. The actual constraint is that design systems are not legible to a model. A design file carries geometry, style, and layer structure. It does not carry ontology — what a thing is, what it is for, what it may and may not do. Most tooling in this space treats the design file as the source of truth and tries to extract semantics that were never encoded there. That is why it plateaus in the same place every time: it can produce something plausible, and it cannot produce something correct.

THE INVERSION

The design file supplies appearance. A separate specification layer supplies meaning. Once you accept that, the work stops being prompt engineering and becomes systems architecture. The specification layer I have been building has five parts:

- **Intent encoded in the token, not the value.** A token named for its appearance tells a model nothing about where it is allowed to appear. A token named for its role tells it exactly that.
- **A composition grammar.** Which components may nest inside which, what a valid slot is, which combinations are forbidden. This lives almost entirely as tacit knowledge in two or three maintainers; externalizing it is most of the work and nearly all of the value.
- **A closed vocabulary instead of negative instruction.** Telling a model not to embellish competes against everything it has ever seen. Enumerating the allowed components, props, token names, and composition rules — and rejecting anything outside the enumeration — changes the behavior without persuasion.

— **Executable definitions of failure.** Canonical accepted and rejected examples, each rejection carrying its reason, and a validator that runs. This turns a reviewer's judgment into infrastructure, and it is the precondition for a system that can correct its own output.

— **Stable identity across the round trip.** Without an ID that survives design file to code and back, "bidirectional" is regeneration wearing a costume. The test is whether something returns as a bound component instance with its layout structure and constraints intact — something a designer can keep working in.

WHAT I BUILT

Working proofs, built through prompt- and IDE-driven development, currently cover: ingesting design pages and frames and identifying components within them; pulling in and validating design tokens against the specification; translating design files into code or into structured markdown for further ideation in either direction; exporting code, interface guideline documentation, and component libraries; validators and instructional layers that check output against the rules and explain what failed; and a round trip that returns generated components to a layered, editable design file — the piece most tooling in this category does not attempt.

WHAT IT PROVED

The model was never the limiting factor. Quality tracked almost entirely with the precision of the specification. Every failure I chased turned out to be an ambiguity in the system I was describing, not a deficiency in the tool describing it. Which means the work is design leadership — deciding what the system means — wearing a technical costume.

The intelligence belongs in the specification layer, not in a tuned model. A rules corpus is portable across model generations, versionable, and auditable, and it improves every time the underlying models improve. A fine-tuned model is locked to a vendor and a version, and it goes stale.

WHAT I WOULD BUILD NEXT

With a team rather than the margins of a role:

— **Take one component family fully to production standard** — spec, golden set, validator, round trip — rather than broad partial coverage. A complete narrow slice is demonstrable; a broad partial one is not.

— **Formalize the golden set as a regression suite**, so quality is enforced continuously rather than reviewed periodically.

— **Close the self-correction loop** — reject, return the reason, retry against it — which is only possible once failure is executable.

— **Instrument it.** Rework rate, time from generated to merged, percentage of output passing validation on first attempt. If the argument is productivity, the argument needs numbers.

WHY THIS IS A LEADERSHIP PROBLEM

Wiring a design tool to a model is a weekend. Specifying a system precisely enough that a model cannot produce a wrong result requires knowing what the system means, what the teams consuming it need, where the exceptions are legitimate, and where the quality bar is real rather than habitual.

That knowledge does not come from the tooling. It comes from having built design systems from scratch, from having managed the teams that live with the output, and from having sat between design and engineering long enough to know which arguments are worth encoding and which are worth dropping. Sixteen years, design

teams of up to twenty, systems built from nothing at Goldman Sachs and net-new developer portals at JPMorgan Chase. This is the same work. The consumer changed.

Written from my own method and architecture. No proprietary system content, code, or interface material is included.