Building a Design System to Increase UI Consistency and Dev Velocity

Context

When I joined the team, UI development was slow and inconsistent. Not because of bad engineers — because of entropy.

Every team solved the same UI problems independently: form fields, loading states, empty states, modals, tables. Each solution made sense locally. Collectively, they made the system harder to reason about.

The symptoms were everywhere:

- Buttons that looked almost the same but weren't
- Spacing that varied by a few pixels across apps
- Typography implemented slightly differently in each application
- Designers producing specs that couldn't be implemented consistently because there was no shared foundation

New features took longer than they should have. UI changes required touching multiple files across multiple apps. Fixes in one place introduced regressions in another. Nothing was technically broken. Everything was fragile.

Engineering velocity was suffering, and the inconsistency was becoming visible to customers. Leadership was asking why features took so long. Designers were frustrated that their work didn't translate faithfully to production.

Action

I proposed building OUI — a shared component library and design system. But rather than starting with theory (tokens before components, exhaustive variants before real usage), I took a pragmatic approach.

Building alignment across the organization:

First, I partnered with our UX designer to audit the existing UI landscape — cataloging duplicate patterns, inconsistencies, and the cost of each. We created a visual inventory that made the problem concrete enough for non-technical stakeholders to understand.

I then brought this to management and leadership, framing the investment not as a "design system project" but as an infrastructure play that would directly accelerate feature delivery. I quantified the time engineers spent recreating the same components and showed how that compounding cost would only grow.

Defining the architecture with engineers:

With buy-in secured, I worked with senior engineers across teams to determine the architecture. We made deliberate choices:

- **Headless first, opinionated where it matters.** Behavior and structure are shared. Styling is constrained but extensible. Too headless and every consumer reinvents the same patterns. Too opinionated and teams fight the library instead of using it.
- **Tailwind for composition.** Instead of fighting CSS abstractions, OUI embraces utility composition while enforcing consistency where it actually matters.
- **Figma as visual source of truth, code as enforcement.** Tokens bridge that gap — they don't introduce a second system teams have to keep in sync manually.

We established three constraints for inclusion: every component had to already be duplicated across apps, have real (not hypothetical) usage, and reduce code rather than just moving it.

Driving implementation and adoption:

I worked directly with developers to implement components, review PRs, and ensure consistent patterns. We avoided "wrapper hell" — when a primitive already did the job well, OUI composed it instead of replacing it.

Versioning became a social contract. Breaking changes were explicit. Migrations were documented. Consumers were never surprised. If upgrading the UI library felt risky, teams would stop upgrading — and once that happens, the library is dead.

I ran workshops with engineering teams to drive adoption, created migration guides, and paired with developers during the transition period.

Result

After OUI was adopted:

- **New features shipped measurably faster** — engineers spent time on business logic instead of rebuilding UI primitives
- **UI regressions decreased significantly** — shared components meant fixes propagated automatically
- **PR reviews became easier** — reviewers could focus on logic rather than debating UI implementation details
- **Designers and engineers developed a shared language** — the gap between Figma specs and production implementation narrowed dramatically
- **Engineering velocity improved by over 200%** through standardized workflows and shared components

None of this showed up as a single metric. It showed up as less friction everywhere. That's the hallmark of good infrastructure.

Learning

Start from production, not from theory. The strongest design systems are extracted from real code, not designed in abstraction. Every component in OUI earned its place by solving an existing problem.

Alignment is as important as architecture. The technical decisions were the easy part. Getting UX, engineering, management, and leadership aligned on why this mattered — and keeping them aligned through the messy middle — was what made adoption possible.

Stability is a feature. The moment teams can't trust the library, they stop upgrading. Treating versioning as a social contract, not a technical detail, kept adoption growing.

Restraint compounds. OUI didn't try to own everything. It made the right things boring. UI libraries aren't about control — they're about leverage. And like most leverage in engineering, their value compounds quietly over time, long after the excitement of building them has passed.

That's when you know you built the right thing.