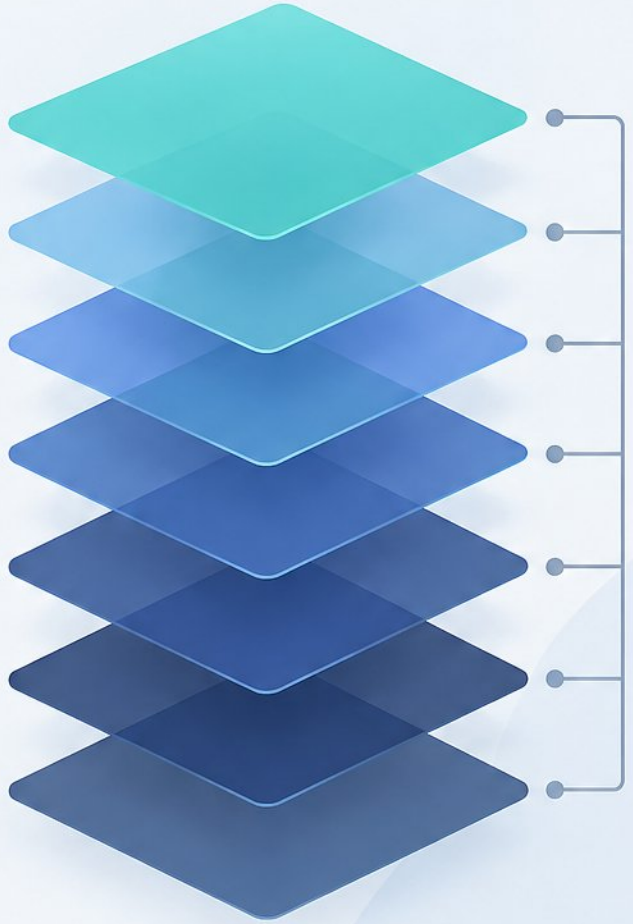
How to Create Design Systems: A Practical Workflow

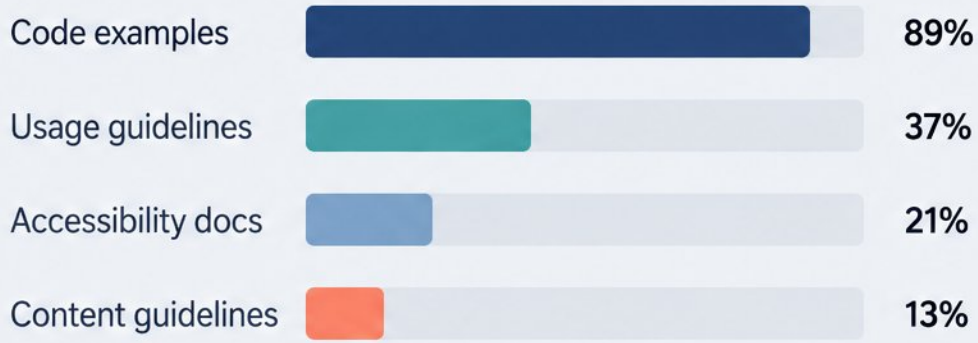
- A design system is governed decisions, coded components, patterns, docs, and working agreements — not just a Figma library
- Seven connected layers: foundations, tokens, components, patterns, guidelines, governance, tools
- Course roadmap: discovery, tokens, specs, build, docs, governance, adoption, 90-day plan
- Designers own intent and guidance; developers own implementatiom and tests; the system team owns the contract and releases
- A workflow turns scattered UI work into a versioned product with owners and support



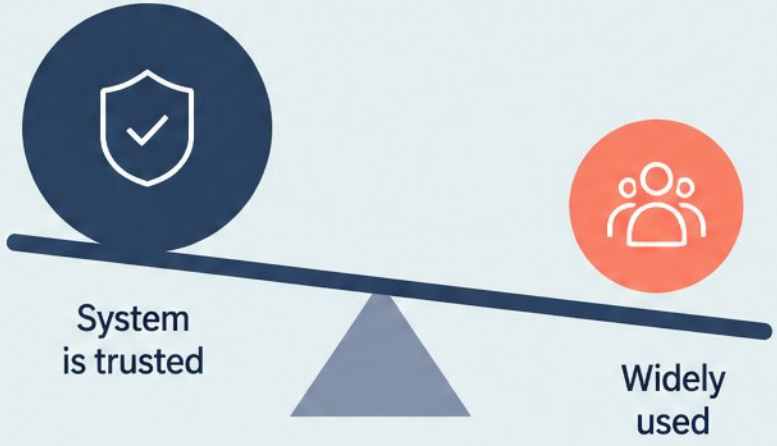
Where Teams Actually Get Stuck: Evidence and Failure Modes

- Consistency at scale is a governance problem before a technology problem
- 158 systems audited: 89% ship code examples, only 37% ship usage guidelines
- Accessibility docs drop to 21%, content guidelines to just 13%
- Trust outruns adoption: most say the system is trusted, few say it's widely used
- Weak mandate and governance outrank component completeness as adoption blockers
- Maturity is a profile, not a stage — fix the weakest link

Documentation Coverage Drops Off

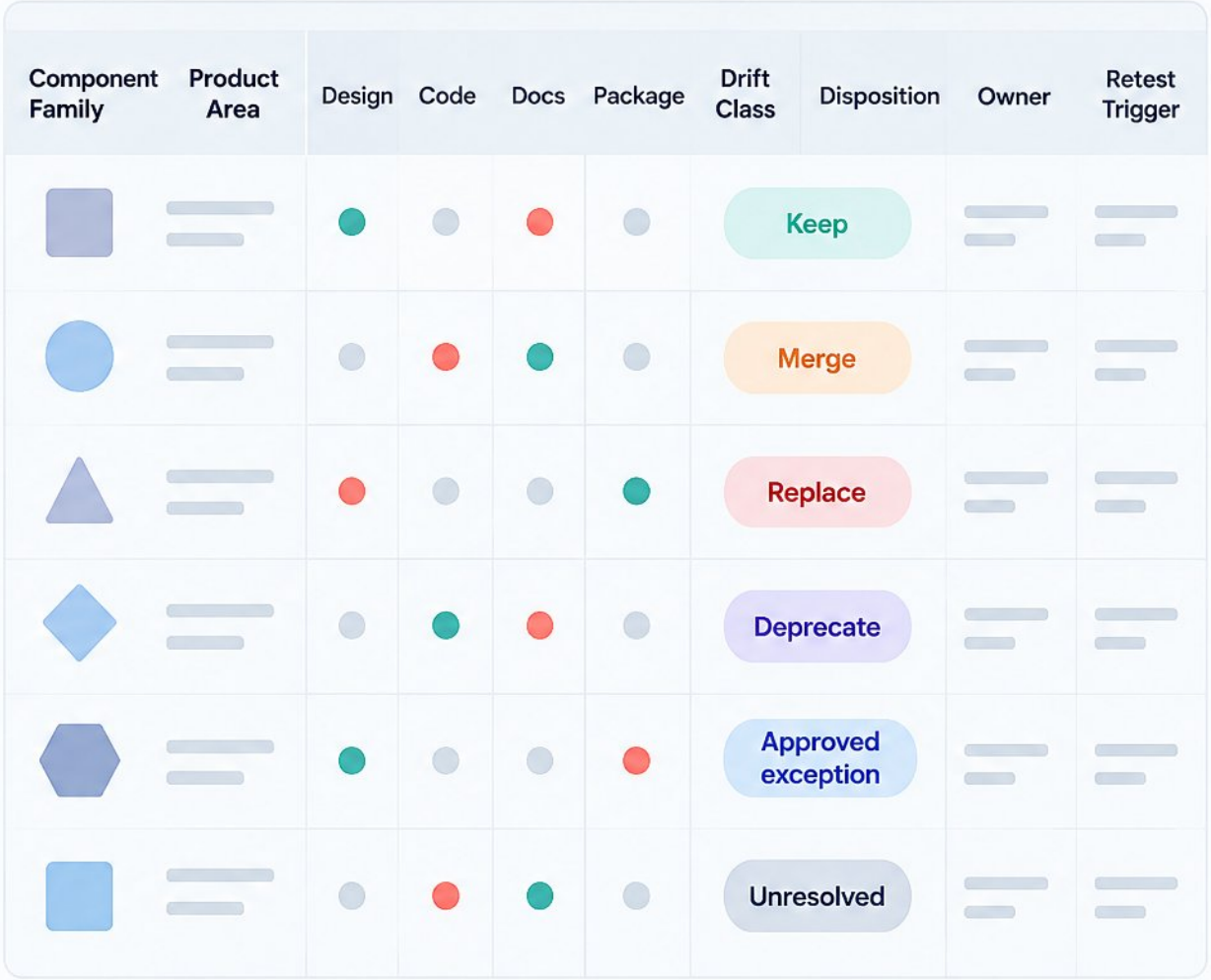


Trust Outruns Adoption



Discovery: Bound the Audit, Then Prioritize

- Start from a repeated cost, not a full inventory
- Sample one or two component families across at least two product areas
- Record design, code, docs, and package identities separately before reconciling
- Score drift in four classes: token, variant, pattern, documentation
- Headline metric: variant sprawl — rendered variants vs. library variants
- End every row with a disposition, owner, and retest trigger
- Dispositions: keep, merge, replace, deprecate, approved exception, unresolved



Scoping the First Slice and Aligning Stakeholders

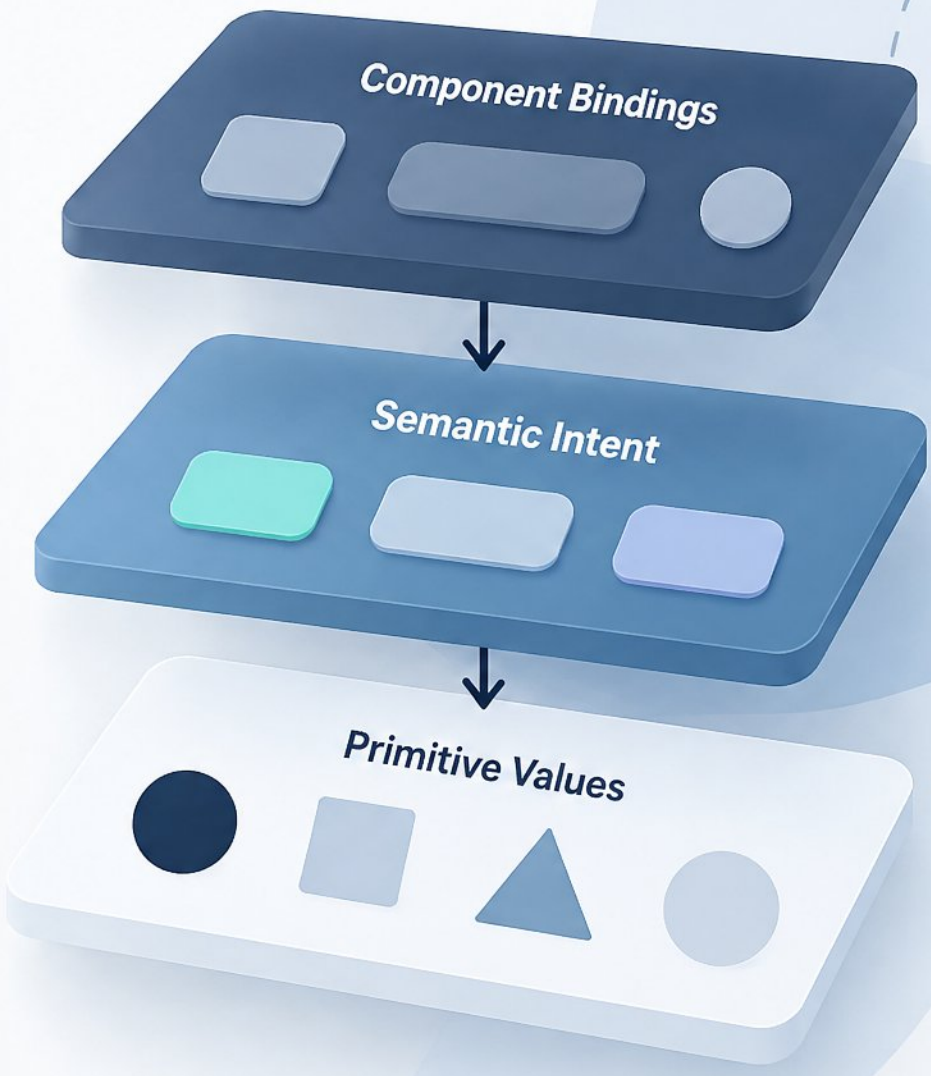
1. Triangulate demand, supply, and usage — include shadow components teams rebuild
2. Prioritize by usage frequency versus build cost; cap P1 at a small number
3. Write one success line tied to a business outcome, e.g. 30% fewer duplicates
4. Answer the scoping questions: what, who, maintenance, growth, lifespan, budget
5. Prove the direction with one real refactor before widening adoption



30% fewer duplicates

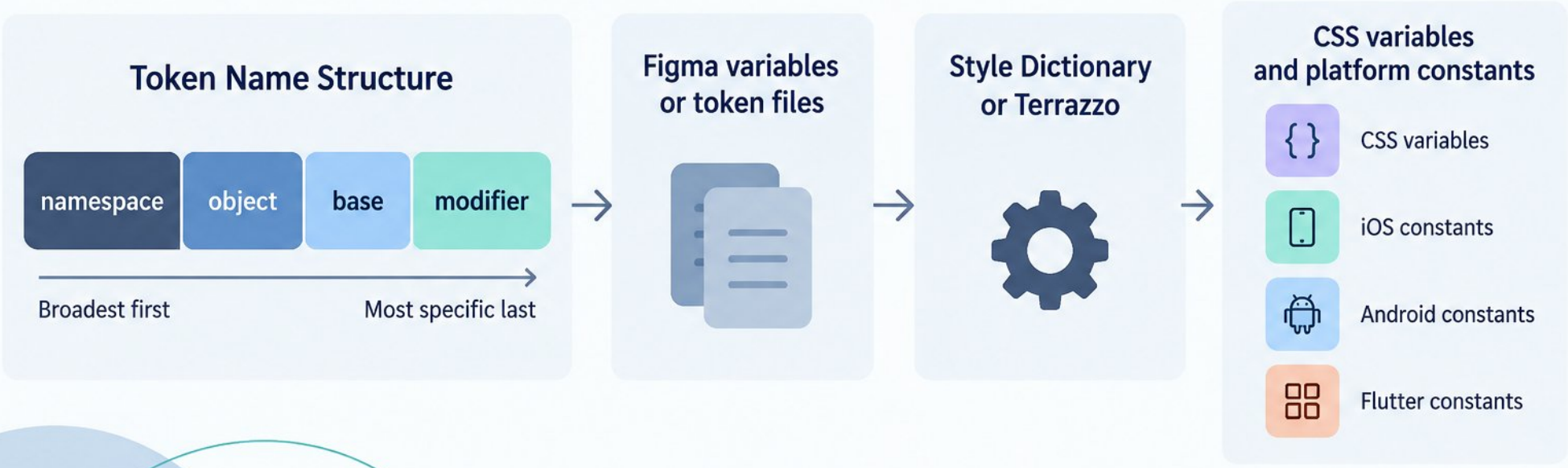
Foundations: Tokens as the Contract

- Tokens: named, typed, described values covering color, spacing, type, radius, elevation, motion, and states
- Three tiers with downward references only: primitive values → semantic intent → component bindings
- Theme by swapping values, never by naming modes: light, dark, and high-contrast share one key set
- Semantic layer is the public API: closed grammar, one meaning per name, no literals in code
- Contrast enforced at the token layer: 4.5:1 text, 3:1 boundaries, focus rings, meaningful icons



Token Naming and Interoperability Across Tools

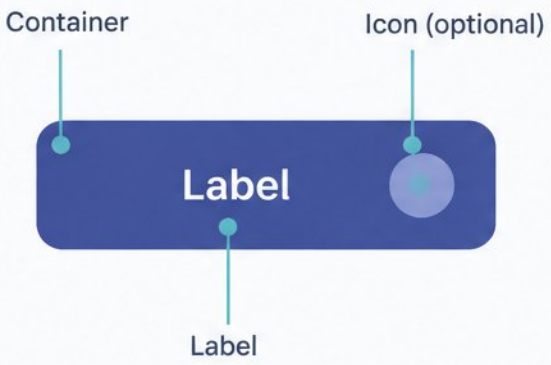
- Use ordered segments: namespace, object, base, modifier — broadest first, most specific last
- Name the decision, not the value: danger and surface survive a rebrand
- DTCG 2025.10 hit first stable release in October 2025; core format is production-ready
- Theming, multi-brand, Display P3 and Oklch color, and aliases in one spec
- Pipeline: Figma variables or token files → Style Dictionary or Terrazzo → CSS variables and platform constants



Component Design: From Anatomy to API

A clear spec turns components into consistent, accessible, and scalable experiences.

Anatomy



Variants and States



Accessibility

- Role
- Keyboard map
- Focus
- Announced changes

Content Rules

- Min/max length
- Truncation
- Error text
- Ship edge cases resolved

Spec template: purpose, anatomy, variants, states, behavior, responsive rules, token mappings

Separate interaction states (hover, focus, active) from behavioral modifiers (destructive, loading)

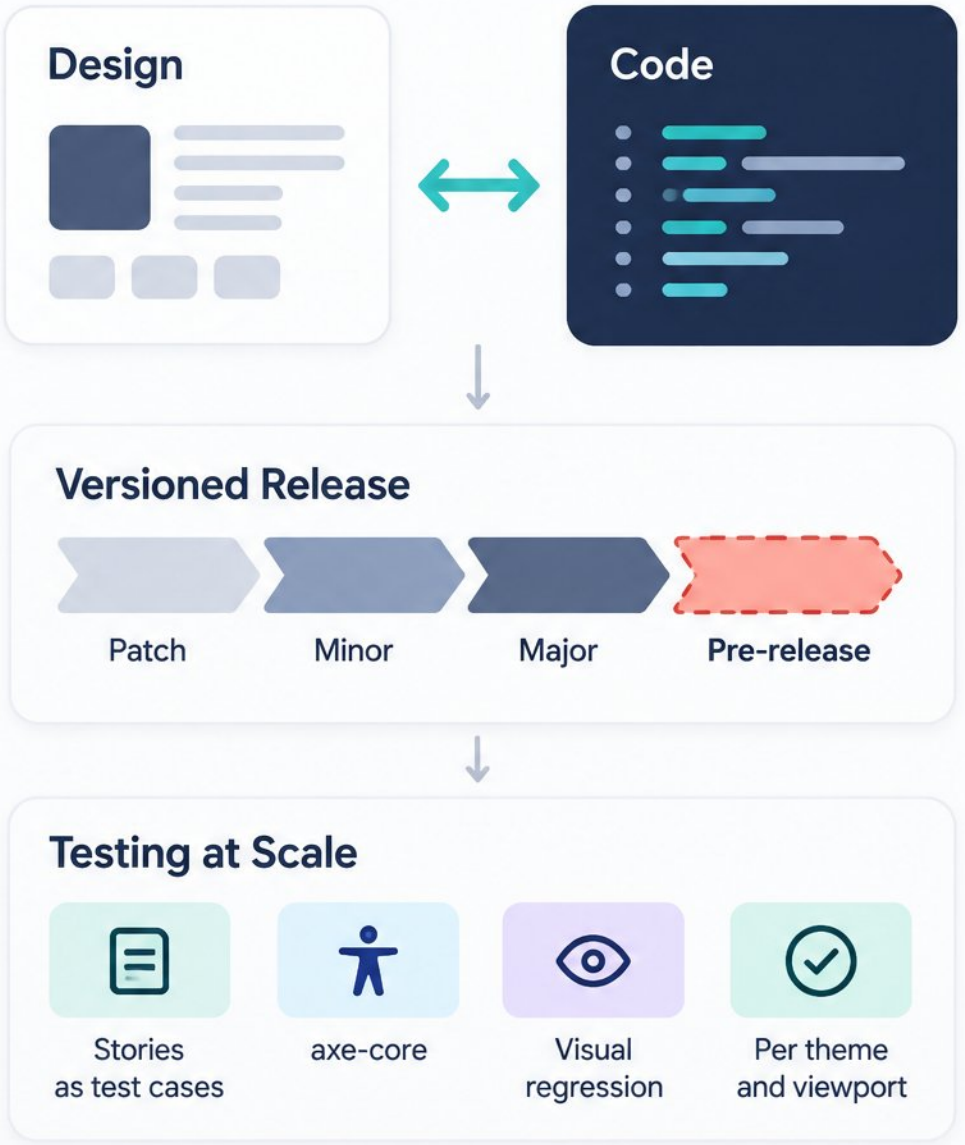
Cap variants deliberately—more than a handful? Split into a pattern

Accessibility belongs in the spec: role, keyboard map, focus, announced changes

Content rules— min/max length, truncation, error text—ship edge cases resolved

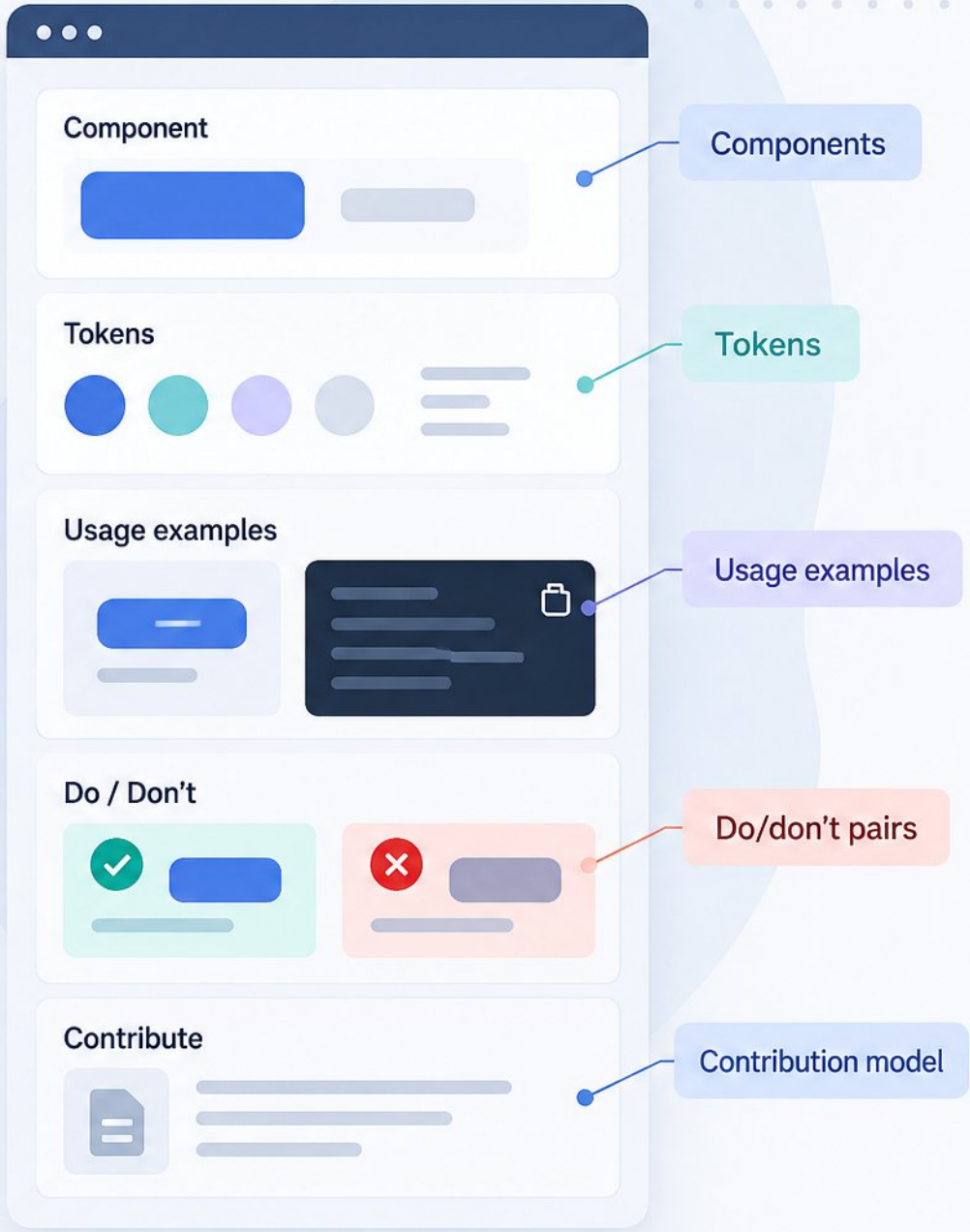
Building and Distributing the Component Library

- Code is the source of truth for behavior; the design file mirrors it
- Code Connect-style mappings link design components to real code — no reinvention
- Semantic versioning via Changesets or semantic-release; pre-release tags for breaking work
- Publish what counts as breaking: visual and interaction changes that force consumer edits
- Test at scale: stories as test cases, axe-core, visual regression per theme and viewport



Documentation That People Actually Use

- Five core blocks: components, tokens, usage examples, do/don't pairs, contribution model
- Answer "how do I use this right now" in under a minute, with copyable code
- Prefer live rendered components over static screenshots; match pattern to page
- Keep the why and resolve edge cases: 80-character titles, mobile errors
- Generate docs from component source so they cannot drift; make everything searchable



Governance, Contribution, and Decision Rights

- Choose a defensible model: centralized, federated, or hybrid
- Most teams start centralized, then shift to hybrid as they scale
- Sort decisions by blast radius: brand tokens and shared components go to core
- Contribution lifecycle: proposal and RFC, prototype, beta, stable, deprecated
- Record decisions and releases — one accountable owner beats a committee
- A MAJOR bump should never be discovered in a broken build

Decision Rights



Contribution Lifecycle



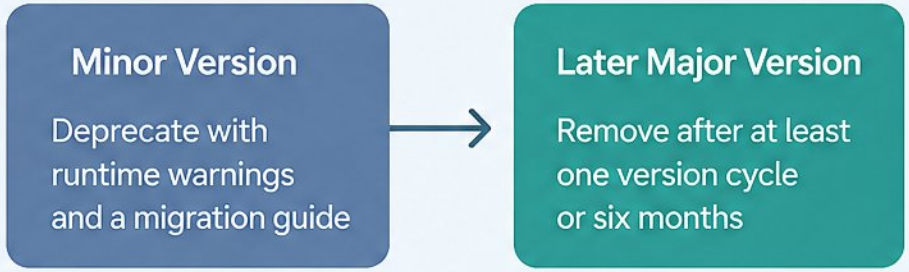
Audit / Disposition

keep | merge | replace | deprecate | exception

Deprecation and Communicating Breaking Changes

- Publish what counts as breaking — that definition is your contract with consumers
- Deprecate in a minor with runtime warnings and a migration guide
- Remove in a later major: at least one version cycle or six months
- Three communications: informal heads-up, pre-deprecation review, then formal notice
- Use a Next-prefixed replacement when a drastic redesign can't wait
- Raise lint rules to error level late; disable comments track remaining work

Deprecation Timeline



Communications



Adoption: Metrics, Migration, and Enablement

Coverage = design-system usage ÷ all eligible usage; track per product, not one aggregate

Instrument early: CI scans, token lint, Figma detach rates, version distribution

Expect the 70%→100% plateau: opt-out enforcement and named conversations close it

Migrate through normal product change: high-traffic, accessibility-problem areas first

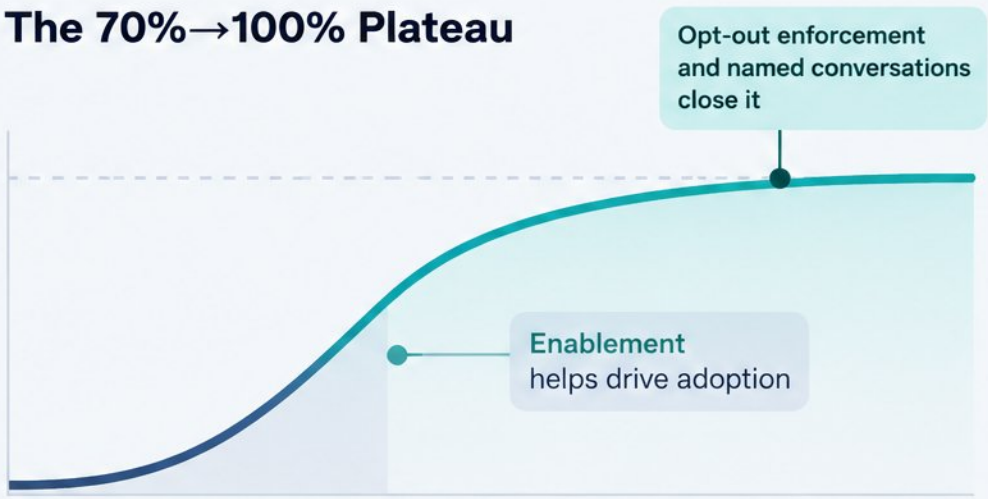
Enablement: rotating office hours, onboarding kits, champions, logged support backlog

Coverage (per product)

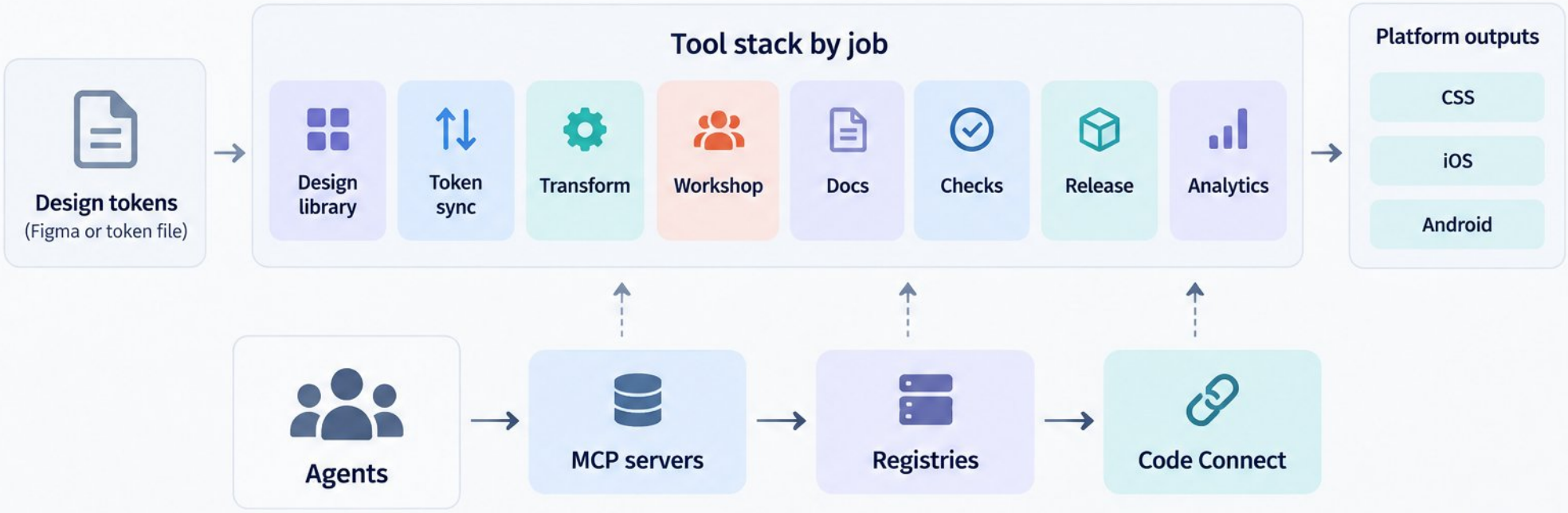


Coverage = design-system usage ÷ all eligible usage; track per product, not one aggregate.

The 70%→100% Plateau



Tooling and AI-Assisted Workflows in 2026



- Stack by job: design library, token sync and transform, workshop, docs, checks, release, analytics
- Agents are now consumers — MCP servers, registries, and Code Connect decide if output uses your system
- Agents only use what's documented; missing guidance gets guessed or invented
- AI is strong at drift detection and first-draft docs, weak at institutional memory — keep a human owner
- Avoid over-automation and lock-in: keep DTCG canonical, everything downstream regenerable

Putting It Together: A 90-Day Action Plan

From foundation to measurable impact — a focused plan for the next 90 days.

Days 1–15

- Days 1–15: bound the audit, name the owner, align on the success line

Days 16–45

- Days 16–45: ship tokens and one stable component family with specs, tests, docs

Days 46–70

- Days 46–70: pilot in two products, fixed release cadence, changelog, office hours

Days 71–90

- Days 71–90: measure coverage, publish health dashboard, open RFC path

Name roles explicitly and plan for scope creep, version spread, governance bottlenecks.



Plan for these risks

- Scope creep
- Version spread
- Governance bottlenecks

PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/3a9e58aa/public