6.5.1. Why Open Source Standardizes Things Fastest and Best

The Selection Pressure Mechanism

Open source projects face brutal, continuous selection pressure that proprietary software avoids. Here's why this drives toward standardization:

Proprietary Software	Open Source
Users stuck with bad decisions (switching costs are high)	Users vote with their feet immediately; projects die or are forked
Manager decides architecture; no external feedback	Community feedback forces iterative improvement
Mistakes buried in legacy code forever	Bad ideas are exposed, alternatives emerge, best rises to top
Optimization for vendor lock-in	Optimization for genuine utility and elegance

The key insight: open source projects that become standard are survivors of intense natural selection. A mediocre closed-source product can persist for years through inertia and switching costs. A mediocre open-source project dies because the friction to abandoning it is zero.

The Compound Learning Problem

When thousands of developers use and contribute to a single project (like Docusaurus), the aggregate learning and iteration is exponentially faster than any isolated team could achieve.

Consider the journey of Docusaurus:

Year 1-2: Meta engineers build a static site generator for docs. It has basic features, some rough edges.

Year 3-5: Thousands of projects use it. Contributors report bugs, request features, propose optimizations. Each fix and feature represents one less thing someone else has to invent.

Year 6+: The project has crystallized into a form that solves 80% of documentation needs *extremely well*. This wasn't planned from the top—it emerged from collective problem-solving.

A solo AI system or small team rebuilding documentation tools from scratch would need decades to accumulate this knowledge. They'd reinvent solved problems.

The Specialization Advantage

Open source standardization enables extreme specialization and division of labor that isolated teams can't match:

- Search: Algolia doesn't need to rebuild search; they integrate with existing Docusaurus plugins
- Deployment: Vercel doesn't need to optimize for every static site generator; they specialize on Docusaurus, Next.js, etc.
- Theming: Designers can create premium Docusaurus themes because there's a large enough market
- Accessibility: The community contributes accessibility fixes once, and everyone benefits

When a single standard emerges, specialists can build around it. When everyone vibe-codes their own solution, nobody can specialize deeply because the surface keeps changing.

The Network Effects of Standards

Standards create network effects that are invisible at first but become compounding:

- Knowledge transfer is instant – A new developer learns Docusaurus once, can work on any Docusaurus site
- Plugin ecosystems thrive – Developers invest in Docusaurus plugins because they serve thousands of projects
- Hiring becomes easier – Companies need "Docusaurus expert," not "person who understands our custom docs site"
- Migrations become easier – If you're on the standard, switching tools is cheap; if you're on a custom solution, you're locked in
- Onboarding new contributors is faster – Documentation for how to contribute is written once, shared everywhere

Each of these compounds. More developers → more plugins → more attractive to new projects → more developers. Vibe-coded solutions break this cycle at every step.

Why Docusaurus Specifically Is Optimal (Not AI Vibe Coding)

Let me be concrete about why letting AI generate a custom documentation site from scratch would be a disaster compared to using Docusaurus.

The Problem With AI Vibe-Coding Docs Sites

An AI system told "build a documentation site" would:

1. Reinvent solved problems inefficiently

The AI might generate:

- A custom search implementation (slow, no linguistic understanding)
- A hand-rolled navigation system (no accessibility, no breadcrumbs, no hierarchy)
- A custom syntax for code highlighting (buggy, unmaintained)
- A custom dark mode toggle (loses state on refresh, flashes on page load)

Each of these are solved problems with proven solutions. The AI rebuilding them means:

- Starting from zero rather than standing on giants
- No community bug fixes or improvements flowing back
- No ecosystem of specialists maintaining these components

2. Creates technical debt that compounds

A bespoke docs site is a *liability*. When:

- Security vulnerabilities discovered in syntax highlighting library → you patch it
- Dark mode CSS rules conflict with new framework version → you debug it
- Search breaks on new device types → you investigate it

With Docusaurus, the community bears these costs collectively. One person fixes a search bug; everyone benefits.

3. Guarantees knowledge loss

The developer who built the custom site eventually leaves. New developers inherit code with:

- No documentation of why certain patterns were chosen
- Idiosyncratic naming and structure
- Fragile coupling between components ("don't refactor this, it breaks search mysteriously")

Docusaurus is well-documented because *thousands of developers* maintain it and write about it.

4. Locks you into architectural choices

A custom site might have:

- Tight coupling between routing and styling
- Data structures not designed for internationalization
- Search index built into page HTML (can't scale beyond ~10K pages)

Docusaurus was designed by people who've maintained docs for products with millions of pages and dozens of languages. The architecture embeds hard-won knowledge.

What Docusaurus Got Right (Why It Won)

Docusaurus didn't become standard by accident. It won because it solves the *actual* problems documentation needs:

Problem	Docusaurus Solution	Why This Matters
Writing docs should be easy	Markdown + MDX (embed React components in Markdown)	Lowers barrier to contribution; no special syntax language
Search must work out of box	Algolia integration (free tier available)	Users expect search; Docusaurus doesn't burden you with implementing it
Docs should look professional without effort	Beautiful default theme	First-time users don't have to hire a designer
Internationalization must not be an afterthought	Built-in i18n support	Not a hack; baked into routing and data structures
Versioning is essential	Documentation per software version	Critical for libraries; Docusaurus designed for this from the start
Sidebar navigation must scale	Declarative JSON sidebar structure	Works at 10 pages or 1000; doesn't degrade
Performance matters for docs	Static site generation; fast page loads	Docs are often accessed on slow connections; Docusaurus optimizes aggressively

Every single one of these represents a decision made by someone who'd shipped docs at scale. An AI vibe-coding a docs site would either:

- Miss these problems entirely (results in bad docs site)
- Reinvent them poorly (results in slow, unmaintainable docs site)
- Copy Docusaurus (results in using Docusaurus)

The Economics of Standards vs. Custom

Let me quantify why Docusaurus wins over custom solutions:

Custom AI-generated docs site:

- AI coding time: 40 hours (to vibe-code something functional)
- Maintenance per year: 80 hours (bugs, framework updates, feature requests)
- Knowledge cost: High (only the original author understands it)
- Hiring cost: Medium (need to hire someone who can maintain *this specific* codebase)
- Total 5-year cost: ~$50K-100K

Docusaurus:

- Setup time: 2 hours (install, configure, write docs)
- Maintenance per year: 5 hours (most issues handled upstream by community)
- Knowledge cost: Low (any competent developer can work on it)
- Hiring cost: Low (any developer knows Docusaurus or can learn it in a day)
- Total 5-year cost: ~$5K-10K

The custom solution is 5-10x more expensive over time, and it's *worse* because:

- The custom solution has bugs the community already fixed
- The custom solution can't take advantage of new Docusaurus features
- The custom solution is brittle in ways Docusaurus explicitly handles

---

Why This Pattern Generalizes (The Real Lesson)

Your original insight about standardization applies here:

Once a solution becomes dominant in open source, it's almost always better to use it than to rebuild it, because:

1. It's survived selection pressure – Bad ideas die; good ones compound
2. The community has invested more than you could solo – Thousands of hours of collective labor
3. The architecture embeds institutional knowledge – Decisions made by domain experts at scale
4. The ecosystem has formed around it – Plugins, integrations, expertise, documentation

This is why Docusaurus will remain the standard for docs. Not because it's perfect, but because:

- Any attempt to improve it is better made *as a Docusaurus contribution* than as a fork
- Any attempt to replace it has to overcome network effects (hiring, plugins, knowledge)
- The marginal benefit of starting over is negative; you give up 80% of working features to maybe get 20% improvements

---

When AI-Generated Custom Code Makes Sense (and When It Doesn't)

This is the critical boundary:

Use Case	Better Approach	Why
Docs/blog site	Use Docusaurus	Standard exists, solved problems, ecosystem
Admin dashboard	Use Refine, AdminJS	Standard exists, complex CRUD patterns
Authentication	Use Auth0, Supabase	Standard exists, security implications too high
Database ORM	Use SQLAlchemy, TypeORM	Standard exists, query optimization critical
Custom business logic	AI-code with human review	No standard exists; differentiation is here
Domain-specific algorithms	AI-code, human architect	No standard exists; this is where you compete

Use Case	Better Approach	Why
Integrations/glue code	AI-code freely	Standard is "whatever works"; customization is cheap

The pattern: use standards for solved problems. Only vibe-code (with understanding) where you actually differentiate.

Docusaurus exemplifies this perfectly: it's the standard for documentation, so any team using a custom docs site is wasting engineering resources on a solved problem instead of investing in their actual product.

The Architect's Role in This World

Here's why becoming an architect is harder but also why it matters:

An architect's job is to:

1. Know what's been standardized – Which problems have winning solutions?
2. Know when to deviate – Where does differentiation actually matter?
3. Design constraints and specs – How do the standard pieces fit together?
4. Guide implementation – Are we using standards correctly, or are we reinventing?

An architect using Docusaurus doesn't say "build a docs site." They say:

- "We're using Docusaurus because it solves N problems we don't want to solve"
- "Here's how we're customizing the theme to match our brand"
- "Here's the plugin we're building because Docusaurus doesn't do X"
- "Here's the migration path to Docusaurus v3 and why it's worth the upgrade cost"

This requires understanding *both* what Docusaurus does and *why* your use case fits it. You can't do that without understanding the problem domain deeply.

The cost of becoming an architect (years of learning) is exactly the cost of building taste and judgment about which standards to adopt and how to use them effectively.

Revision #3
Created 2026-05-30 16:04:39 UTC by dpasn
Updated 2026-05-30 21:02:00 UTC by dpasn