Why Content Design Is Becoming More Technical

And what to do about it

Patrick Stafford · UX Content Collective

Patrick Stafford

CEO & Co-founder
UX Content Collective

Writers of Silicon Valley
Podcast

Almost Famous
Best film ever made

The expectation is already in the job description

Spotify — Content Designer, Personalisation

"Partner with engineers to design and run human-in-the-loop evaluations. Define what 'good' looks like for AI-generated text."

Meta — Content Designer

"Use your familiarity with how code works in software — reviewing code to ensure in-product language is as intended, or creating the logic to personalise the experience."

What companies actually want from content designers

Source: ademolaadepoju.substack.com

INTRO

Content design has always been technical

- There's always been a wide spectrum of technical knowledge in content design — HTML, JSON, GitHub, APIs.
- But our work has always sat inside technical systems — we just chose not to look.
- Technical concepts were a barrier — steep learning curves, unfamiliar syntax, a culture that didn't always welcome non-engineers.
- So we built our identity around craft. Which is right. But it became a way of not looking over the wall.

WHY NOW

Two things changed

- **Reason 1:** Large language models are built on language. The thing powering AI-driven products is what we work with every day. Content decisions are now engineering problems.
- **Reason 2:** Natural language tools mean you can now touch the code. Claude Code, Cursor, Copilot — you describe what you want in plain English. The code changes.

"

The hottest new programming language is English

Andrej Karpathy — former Director of AI, Tesla

The way into technical work is now the same as the way into writing. There's no longer a structural reason not to engage with it.

More teams are expecting content designers to understand where their content lives, how changes ship, and how to participate in technical conversations.

THE OPPORTUNITY

What technical skills actually unlock

- Edit strings directly in code — without waiting for a sprint
- Run content audits at a scale that used to take a week
- Catch issues before QA instead of after
- Wield influence in technical conversations you were previously left out of

SECTION ONE

How products are built

Top-down vs bottom-up thinking — and why mixing them causes friction

01

When you think of a product, what do you picture?

TOP-DOWN

Designers often think top-down

- We start with what the user sees: the screen, the interface, the words.
- We go deeper into the technical stack only when something forces us to.
- The user experience is the main event. Technical infrastructure is context at best, obstacle at worst.

BOTTOM-UP

Engineers often think bottom-up

- They start with the system: what data exists, where it's stored, how it moves.
- The UI is the end result — not the main event.
- Many engineers rarely look at the interface at all. They're thinking in logic, dependencies, and infrastructure.
- Neither view is wrong. They're different entry points into the same product.

Mixing these types of thinking gets us into trouble.

When a content designer asks for a "simple" change and an engineer pushes back — it's usually not obstruction. It's a collision between two different mental models of what the product is.

The same button. Two completely different conversations.

You see

- Is 'Submit' the right word?
- Maybe 'Save' is clearer
- What does it feel like to use?
- Does it match the pattern?

They see

- Where is that string defined?
- Is it in a Localization file?
- Is the same key used elsewhere?
- If I change it here, what breaks?

Why "simple" changes aren't

- Maya notices a settings button says "Submit" when it should say "Save."
- The key `common.button.submit` is used in 14 places.
- The right fix is a new key — plus translation, design review, code review, and QA.
- A two-minute fix becomes a multi-day coordination exercise — and nobody is wrong. The system is just more connected than it looks.

You don't need to see the Matrix. You need to know it's there.

You're good at your job because you think user-first. Engineers are good at theirs because they think systems-first. The gap between those two orientations is fine — not knowing it exists is the problem.

SECTION TWO

What "technical" means

A different frame for a loaded word

02

Wrong frame vs right frame

Wrong frame

- "Technical" means developer
- Learn everything or nothing
- Full competence required
- All-or-nothing skill set

Right frame

- Technical fluency = vocabulary
- Know what to ask, not the answer
- Participation, not mastery
- A map, not a skill set

The triage framework: how to decide what's worth learning

1

Do it yourself

Worth learning to execute. Direct capability, low time investment.

2

Understand conceptually

Follow along and collaborate — you don't need to execute.

3

Know it exists

Awareness is enough. You know the word. You know to ask.

SECTION THREE

The skills

Five things worth learning — and why

03

SKILL 1

Know where your content lives

WHY IT MATTERS

- Hardcoded strings need a code change and a release.
- Localization files are readable — but still need a release.
- A CMS can often be changed without engineering at all.
- Knowing which one tells you who's involved, how long it takes, and where you have leverage.

```
Hardcoded
 → code change
 → release

JSON / YAML
 → readable
 → release

CMS
 → no engineer

Database
 → needs dev
```

SKILL 1 · WHY IT UNLOCKS

Knowing where content lives means you can…

- Scope work accurately — stop promising "that'll take five minutes" when it won't
- Know who needs to be involved before you start — no more late-sprint surprises
- Find your own content during QA — go look for it instead of waiting for someone to show you

SKILL 2

Understand what a repo is

WHY IT MATTERS

- A repo is a folder containing all the code for a product — with a full history of every change ever made.
- Branches are where changes happen in isolation before they're merged in.
- Pull requests are proposals to merge. Once you can navigate one, it stops being a black box.

SKILL 2 · WHY IT UNLOCKS

Knowing what a repo is means you can…

- Find your content without asking a developer — navigate to the file, find the key, read the value
- Review a pull request — see what changed, leave a comment on a specific line, catch issues before they ship
- Understand why "it's on a branch" means it's not live yet — no more confusion about what's in production

SKILL 3

Some basic terminal commands

WHY IT MATTERS

- You don't need to memorise commands.
- You need to know what the terminal is, how to open it, and how to follow a README.
- The goal is removing the blocker of "I don't know how to run this" — and it makes using coding agents dramatically more effective.

```
$ cd project
$ ls
$ cat en.json
$ grep "Submit"
   src/strings/
$ npm run audit
```

SKILL 4

Know how your product is built

WHY IT MATTERS

- You don't need to write React or Python — you need to know what they mean for content.
- Knowing the stack tells you what's possible, what requires a release, and what you can change on the fly.
- Understanding "what's our tech stack?" and "where does content typically live?" is worth the ten-minute conversation.

```
Frontend
 React / Vue / Angular
 → where strings render

Backend
 Node / Python / Ruby
 → logic, APIs, data

CMS / i18n layer
 → where you can edit
   without a dev
```

SKILL 5

Use AI to make technical changes

WHY IT MATTERS

- You describe what you want in plain English. The agent reads the codebase, makes the change, shows you the diff.
- You are still the content designer — your judgment is the product.
- The guardrail: always read every line that changed before merging. Agents hallucinate keys, miss context, and change things you didn't intend.

You type:

"Find all Submit strings in en.json that should say Save instead"

Agent reads:

→ scans the codebase
→ finds 3 candidates
→ shows the diff

You decide:

→ review → approve
→ or push back

CLOSE

Where to go from here

Where to go from here

04

Three questions. Yes to any of them is your entry point.

- **Question 1.** Have you ever been blocked by "I don't know how to run this"?
- **Question 2.** Have you ever made a content decision without knowing where that content lives?
- **Question 3.** Have you ever been cut out of a technical discussion because you lacked the vocabulary to participate?

Find one piece of content in your product. One string. Figure out where it lives and how it gets to the screen.

That single exercise will teach you more about the gap between top-down and bottom-up thinking than anything else.

CLOSING THOUGHT

Craft is not being replaced

- If anyone can generate text, craft is the differentiator. Clear, precise, human-centred writing matters more, not less.
- What's changing is that craft alone isn't enough to be effective in the environments where most of us work.

Questions?

Patrick Stafford · · uxcontent.com

SELF-PACED COURSE

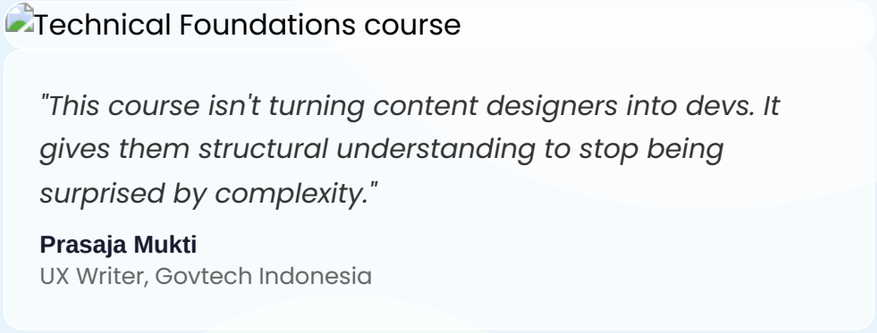
Technical Foundations for Content Designers

Read and manage strings in code, navigate GitHub, use the terminal, and work with engineering teams. Not to become a dev — to become a more equipped content designer.

- Understand where content lives in the architecture
- Navigate GitHub and read pull requests
- Use AI tools for technical tasks and verify the output
- Make a real change to a GitHub repository as your final project

Use code at checkout
MEETUP20
20% off

uxcontent.com/course-technical-foundations-for-content-designers



"This course isn't turning content designers into devs. It gives them structural understanding to stop being surprised by complexity."

Prasaja Mukti
UX Writer, Govtech Indonesia