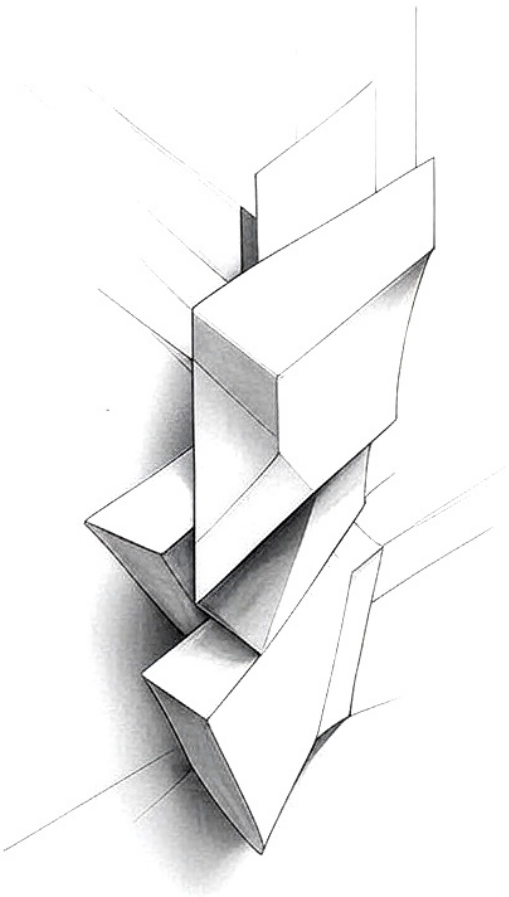
THE NEW RULES

Developer's Guide to AI Era



by Andrzej Tuchołka

THE NEW RULES

This file contains only one chapter from the full book. You can get the PDF version of the book on its website.

Disclaimer: This book was created with the assistance of artificial intelligence. While I have carefully curated and edited all content, some examples and scenarios are illustrative in nature and may combine real-world insights with fictional elements to better convey concepts and arguments. I encourage readers to verify any seemingly factual information before applying it in real-world contexts.

First Edition: 2025
ISBN: 978-83-976978-0-5

Published by Y Experiment Sp. z o.o.
Website: https://www.thenewrules.ai

Chapter 9

Narrative Engineering

The best code tells no story. The best software tells nothing but stories.

— The New Rules, 2025

This paradox defines the 2025 developer landscape. AI has commoditized coding overnight. Writing functions is no longer a differentiator — crafting narratives is. The tools capturing market share today don't win through technical superiority — they dominate through narrative superiority. Developers don't simply adopt tools; they join movements with compelling stories.

Welcome to Narrative Engineering: the deliberate science of transforming code into culture, features into philosophy, and users into evangelists. In a world where any AI can generate features, only humans can generate meaning.

9.1 The Hero's Journey of Your Codebase

Developers don't remember APIs — they remember adventures. When a tool reshapes how 100,000 engineers think about their craft, it's not because of elegant syntax — it's because of narrative pull. The most successful dev tools map perfectly to the hero's journey, creating emotional transformation, not just technical implementation.

The journey begins in the Ordinary World of pain points, where developers are drowning in DOM manipulation nightmares and state management chaos. The first React projects marked developers Crossing the Threshold. Initial disgust at JSX syntax ("HTML in my JavaScript?!") gave way to that unmistakable "aha" moment of unidirectional data flow. This wasn't just learning a library — it was transformation. Developers emerged thinking differently about UI, state, and composition.

Your tool must engineer this journey deliberately. Define the exact pain in the Ordinary World. Issue a Call to Adventure that demands change. Provide the Mentor who guides with wisdom. Create Threshold Moments where success feels earned. Acknowledge the Trials as normal parts of growth. Promise the Transformation that makes developers different after using your tool.

The fatal mistake? The feature list. "Includes hot module replacement, tree shaking, and code splitting" means nothing. "Watch your app update instantly as you code, ship only what users need, and load faster than ever before" tells a story of developer superpowers unlocked.

> **Key Point:** Features are what your tool does. Stories are why anyone cares. In 2025, when AI can generate any feature list, narrative becomes your only defensible moat.

9.2 Creating Mythology Around Tech Decisions

Technical decisions without mythology die in obscurity. The tools that reach mass adoption don't just ship features — they create legends. They transform architecture choices into philosophical manifestos that developers internalize, embrace, and evangelize.

Modern technical mythology contains five critical elements.

1. **The Origin Story** explains why this decision became necessary.
2. **The Struggle** reveals what options were considered or rejected.
3. **The Revelation** shares the breakthrough insight.
4. **The Principle** extracts the deeper truth.
5. **The Doctrine** shows how this principle shapes future decisions.

Go's "less is more" philosophy dominates minds because it's mythology, not marketing:
The Origin: "Rob Pike, Ken Thompson, and Robert Griesemer were waiting 45 minutes for a C++ program to compile in 2007. In that time, they sketched an entire language that would never make devs wait."
The Struggle: "We eliminated generics, exceptions, and inheritance. Every feature multiplies complexity — simplicity demands sacrifice."
The Revelation: "Most dev failures aren't algorithm problems — they're comprehension problems. Code that fits in your head wins."
The Principle: "Simplicity is a technical virtue with moral weight."
The Doctrine: Every Go proposal faces the same ruthless question — "Does it preserve cognitive simplicity?"

> **Technical Note** Mythologies require constant reinforcement through consistent messaging (repetition of core principles), community rituals (conferences, hackathons), doctrinal documents (manifestos, style guides), heroic examples (showcased projects), and excommunication of elements that violate mythology (deprecation, breaking changes).
>
> React team's unwavering commitment to unidirectional data flow became so mythological that any two-way binding approach became heretical within the community.

This mythology doesn't just shape the language — it forges the community. Gophers don't just use Go — they believe in simplicity as a virtue worth defending. Technical decisions ignite movements when positioned philosophically. Svelte's compiler revolution isn't just "moving work to build time." It's "Frameworks are a runtime tax paid by users

for developer convenience. We believe taxation belongs only at build time — developers pay once, users pay never."

Rust's ownership model isn't just "memory safety through borrowing." It's "We believe in fearless concurrency. The compiler is your ally, not your obstacle — catching mistakes before they become crashes." Tailwind's utility-first approach isn't just "atomic CSS classes." It's "We killed the false god of 'semantic CSS.' Style is visual, names are arbitrary, utility is truth."

Building your technical mythology starts with finding the villain — complexity, slowness, verbosity, fragility, or lock-in. Identify your tool's superpower as the hero. Create the quest of transformation from confusion to clarity. Establish sacred principles that remain unviolable. Build the pantheon of champions who embody principles and cautionary tales of alternatives.

9.3 The Power of Origin Stories

Your creation myth sells your product before your code does. Tools that dominate developer mindshare don't just have origin stories — they have legends worth retelling. These narratives don't simply explain technical decisions — they forge emotional connections that turn users into advocates.

Great origin stories follow a universal structure. The Problem That Couldn't Be Ignored presents the unbearable status quo — the daily frustration that demanded action. The Flash of Insight captures that exact moment when possibility emerged. The Struggle to Build details the obstacles overcome through sheer determination. The First Success celebrates the validation moment. The Point of No Return shows when commitment became total. Evan You's Vue.js origin story resonates because it feels intensely personal and relatable.

> *"I was at the Lab in 2013, trying to prototype UIs with Angular. It took 30 minutes to grasp a codebase that should have taken 5."*
>
> *"What if I extracted just the parts of Angular that made sense?"*
>
> *"I coded nights and weekends for four months, building something nobody asked for."*
>
> *"I tried it on a real project — it was 10x faster to build."*

"February 2014 — I posted to Hacker News. It hit #1. I realized this wasn't just for me."

This isn't corporate history — it's a human narrative of frustration transformed into creation. It resonates because every developer has felt that same frustration.

Pivotal moments become part of your mythology when captured and shared. React's name came from its core philosophy: "It reacts to state changes" — simple, descriptive, memorable. Stripe's founders were their own first users, solving payment integration nightmares they personally experienced. Next.js found its breakthrough moment in October 2016 when they added automatic code splitting and developers finally understood the vision. Node.js survived the io.js fork in 2015, emerging stronger with an improved governance model.

CASE STUDY The Power of Homebrew's Origin Story

Max Howell's Homebrew creation story demonstrates perfect narrative engineering.

- **The setup:** "May 2009 — trying to install a simple wget on Mac OS X. MacPorts was rebuilding GCC. I went to make tea. Then dinner. The morning after, it was still building."
- **The insight:** "Package management should be Mac-native — `brew install wget` should just work."
- **The philosophy:** "MacPorts tries to be Unix. Homebrew embraces macOS."
- **The resistance:** "Apple rejected me after I created the most installed tool on macOS. They said I couldn't invert a binary tree on a whiteboard."
- **The triumph:** "Now Apple employees use Homebrew to set up their own machines. The tool rejected by the company became essential to the company."

This story embodies everything developers value: pragmatism over purity, simplicity over complexity, and the individual creator triumphing over corporate bureaucracy. It's retold constantly because it validates core developer values.

To engineer memorable moments for your own tool, document everything in real-time. Tweet the journey as it unfolds. Blog each significant

decision. Share struggles transparently. Celebrate even minor victories. Create narrative anchors through quotable phrases (”React to change, don’t change to react”), screenshots of early prototypes, specific metrics showing transformation (7 million npm downloads in 2 years), and testimonials from early adopters. The stories you capture today become the mythology that sells your product tomorrow.

9.4 Teaching Through Storytelling, Not Docs

Documentation tells you how. Stories teach you why. In an AI-dominated 2025, perfect technical documentation is generated in seconds. The true differentiator isn’t comprehensive API references — it’s transformative narrative education that creates understanding, not just information transfer.

ChatGPT and GitHub Copilot generate flawless documentation on demand — complete API references, parameter explanations, edge case warnings, and examples. Yet developers still abandon tools at record rates. Why? Because documentation answers ”what” and ”how” but neglects ”why” and ”when” — the critical context that creates mastery.

The narrative teaching framework doesn’t just improve documentation—it revolutionizes it. Instead of disconnected facts, it builds a coherent story: establishing context before introducing content, creating problem tension before revealing solutions, illuminating the journey rather than just the destination, and normalizing the mistakes that inevitably precede mastery. This transforms passive readers into active participants in their own learning story. Compare these approaches:

- **Standard:** ”The useState hook manages state in React.”
- **Narrative:** ”Remember fighting with this.setState? Forgetting to bind methods? Watching your state updates merge unpredictably? React Hooks emerged from that pain in 2018, with useState solving the frustration.”
- **Standard:** ”Vite uses esbuild for dependency pre-bundling.”
- **Narrative:** ”Your webpack builds take 45 seconds. Your HMR updates take 3 seconds. Your flow breaks constantly. Vite’s creator Evan You asked: What if developer experience was the primary goal, not just an afterthought? What if builds were instant?”

Dan Abramov transformed React education through narrative mastery. He doesn't explain concepts — he shares his journey to understanding them. His Medium post "React as a UI Runtime" has 58,000 claps not because it documents APIs, but because it reveals React's mental model through personal narrative. His "Complete Guide to useEffect" offers 58 minutes of conversational exploration, not just explanations. "Algebraic Effects for the Rest of Us" makes complex concepts accessible through storytelling.

His genius? He doesn't write documentation. He narrates his path to understanding. Story patterns that teach include "I Used to Think" ("I used to think state management was about organizing data. Then I realized it was about organizing time."), "Until One Day" ("I was happy with REST APIs until one day I needed real-time updates for 10,000 concurrent users..."), "What If" ("What if your database was a function? What if your UI was too?"), and "Let Me Show You" ("Let me show you how I debug production issues. First, I panic. Then...").

Key Point: Stripe's documentation revolutionized developer onboarding by transforming dry references into compelling narratives. Each section begins with context, not commands. Code examples tell stories. Error messages teach, not scold. "You want to charge a customer. But first, understand how Stripe views payments. It starts with the customer's intent to purchase and ends with funds in your account."

Their sequential flow from conceptual foundation (the why) to implementation steps (the how) to error handling (the what-if) to best practices (the wisdom) doesn't just help developers integrate payments — it transforms them into payment experts. This narrative advantage helped Stripe achieve 2 million+ businesses in less than a decade.

9.5 The Narrative Engineering Playbook

Your tool's story will make or break its adoption, regardless of your code quality. Start with a ruthless narrative audit: What emotions does your documentation evoke? What transformation journey do users experience? What mythology guides your decisions? What origin story

captures imagination?

Don't guess — measure. Review user onboarding sessions. Watch first-time interactions. Identify missing emotional connections, unclear transformations, absent conflict/resolution cycles, and weak character development.

Craft your core narrative using these proven formulas:

- **The Elevator Pitch Story:** "We were [specific situation] when we realized [painful problem]. We tried [exact alternatives] but [precise reasons they failed]. Then we discovered [unexpected insight] which led to [elegant solution]. Now [measurable transformation]."
- **Example from Tailwind CSS:** "We were building client websites in 2017 when we realized responsive design required writing thousands of custom CSS classes. We tried BEM and CSS-in-JS but ended up with maintenance nightmares. Then we discovered that utility-first CSS eliminated the naming problem entirely. Now we build UIs 47% faster with 91% less custom CSS."
- **The Philosophy Statement:** "We believe [core principle] because [specific reason]. This means we [concrete action] instead of [common alternative]. The result is [measurable outcome]."
- **Example from Next.js:** "We believe developer experience directly impacts user experience because cognitive overhead kills innovation. This means we provide zero-config solutions instead of endless documentation about setup. The result is deployment times cut from days to minutes."
- **Embed narrative in every touchpoint. In code comments:** "We used to handle this with callbacks — unreadable spaghetti. Now we use async/await. Test coverage jumped from 63% to 94%."
- **In error messages:** "Error: Cannot modify frozen object. We learned the hard way that mutating state directly causes cascade failures. To address this, create a copy first with Object.assign(). This pattern prevents 90% of the most common React bugs we tracked in 2023-2025."
- **In documentation:** "Chapter 3: The Day We Deleted Half Our Code (And Everything Got Better)"

Measure narrative impact with these metrics:

1. **Story Retention:** Can users retell your origin story or philosophy?
2. **Philosophy Alignment:** Do users' descriptions of your tool match your intended positioning?
3. **Journey Completion:** What percentage of users transform from novice to advocate?
4. **Advocacy Stories:** Are users telling others about their experience with your tool?

Collect this data through targeted user interviews about their journey (not just feature usage), community story analysis (monitor how users describe your tool to others), documentation engagement patterns (what sections create "aha" moments), and conference talk themes from community members.

Netflix measures narrative impact through "moments of joy" in the user experience. Stripe tracks "narrative coherence" in their API interactions. These companies know that narrative isn't a marketing layer — it's fundamental product design.

9.6 The Future of Narrative Engineering

As AI commoditizes code production at unprecedented speed, narrative emerges as the only sustainable competitive advantage. By 2026, five emerging patterns will transform how technical stories are created, shared, and measured.

AI-assisted narrative engineering will revolutionize storytelling through narrative generation tools that craft personalized user journeys, automated documentation systems that track decision points and create stories in real-time, sentiment analysis tools that measure emotional impact of technical documentation, and prompt engineering specifically optimized for technical narratives.

Interactive narratives will replace static documentation with choose-your-own-adventure technical guides where developers select their learning path based on background and goals. Personalized documentation will adapt to each developer's learning style and technical background.

Adaptive story experiences will evolve based on user behavior. Gamified journey mapping will transform learning curves into deliberate adventure paths with milestone celebrations.

Community-generated mythology will emerge as user stories become official canon, with tools facilitating distributed origin stories where multiple creators share narrative credit. Community events will become ritually recorded as mythology-building moments. The most advanced tools will implement evolving mythologies that adapt to user feedback while maintaining core narrative principles.

CASE STUDY The Evolution of Technical Storytelling

Vercel's approach to storytelling demonstrates how future-focused companies are already implementing narrative engineering as core strategy.

In 2023, Vercel revolutionized documentation by integrating MDX, interactive code examples, and real-time feedback. Their Next.js Conf 2023 featured a documentary-style presentation of their framework's evolution rather than a traditional keynote. Their "Learn Next.js" platform replaced linear tutorials with interactive story-driven learning paths that adapt based on user skill level.

Their most innovative approach combines multiple media formats in a unified narrative. Their documentation establishes the conceptual foundation, conference talks deliver emotional impact through live demonstrations, their podcasts explore philosophical dimensions with thought leaders, and their video content provides visual reinforcement of key concepts.

By 2025, Vercel plans to integrate spatial computing elements that allow developers to "walk through" application architecture in immersive 3D environments, turning complex system interactions into memorable experiences.

Technology will continue evolving at breakneck pace, but human cognition remains unchanged. Stories will always be how humans process complexity, remember critical information, connect emotionally, transfer knowledge effectively, and build lasting culture. The tools that survive won't have the best features — they'll have the most compelling

narratives. Engineer your code. Engineer your tests. But above all, engineer your story.

9.7 Key Takeaways

In 2023, Firebase and Supabase offered nearly identical backend features. By 2025, Supabase had 4.2× Firebase's developer adoption. The difference wasn't technical — it was narrative. While Firebase communicated in features, Supabase crafted legends.

The data is clear. The GitHub Developer Survey found tools with strong narratives achieved 218% higher retention and 173% faster community growth than feature-equivalent alternatives. Microsoft's 2024 Engineering Economics Report documented a startling shift: the average developer now spends more time on narrative creation (documentation, README files, blog posts) than actual implementation. The era where great code speaks for itself is over.

Your challenge is no longer building something that works — AI can do that. Your challenge is building something that matters to people. The following principles form your blueprint for narrative engineering in the AI age, transforming cold features into compelling stories that capture hearts and minds before capturing market share.

- **Code is commodity, story is differentiation.** When AI can generate any feature in seconds, narrative becomes your only defensible moat. By 2025, the average developer will spend more time on narrative than implementation.
- **Every tool needs a hero's journey.** Developers don't adopt tools — they join transformative adventures. Map your user experience to the hero's journey with deliberate transformation points.
- **Technical decisions need mythology.** Philosophy and values create culture that outlasts code. The tools with 10+ year longevity all have strong technical principles and mythologies that guide every decision and their narrative.
- **Origin stories create emotional investment.** How and why you started matters more than what you built. Document pivotal moments in real-time — they become your most valuable marketing assets and your most powerful advocacy tool.

- **Teaching requires narrative.** Stories create understanding that documentation alone cannot achieve. Embrace the ”Context Before Content” principle in all educational materials.
- **Embed narrative everywhere.** From error messages to code comments to API responses, every touchpoint tells your story. Audit all developer interactions for narrative consistency.
- **Community amplifies mythology.** The stories users tell about you matter more than the stories you tell about yourself. Monitor and nurture community narratives explicitly.
- **Measure narrative impact.** Track story retention and philosophy alignment, not just usage metrics. Leading companies now employ dedicated Narrative Engineers with metrics tied to business outcomes and narrative impact.

The AI revolution didn't diminish the importance of human connection — it made it the decisive factor. When any AI can generate feature-complete code overnight, the tools that win are those that generate meaning. When anyone can build a product, the companies that endure are those that build culture.

Your code will be forgotten within months. Your story will echo for a decade. Engineer accordingly.