The Art of Naming Things in Code

SEPTEMBER 2025

6 min read • 1,399 words

Themes: Consciousness Technology Programming Human Centered Recursive Spiritual Contemplative

---

There are only two hard things in Computer Science: cache invalidation and naming things.

Phil Karlton said that, and everyone nods because it's painfully true. But here's what keeps me thinking: naming isn't hard because we're bad with words. It's hard because every name is an act of creation—you're literally speaking something into existence.

Every time I stare at a blank function definition, cursor blinking after `def `, I feel this weight. What do you call something that doesn't exist yet? It's like being asked to name a color nobody's ever seen. The name will shape how people think about it, what they believe it can do.

From Frustration to Philosophy

So here's something embarrassing: the original tagline for what became Requests was "HTTP for Python that doesn't suck."

I was so frustrated with urllib2. Just furious at how something so fundamental could be so hostile to human thought.

That name captured my anger perfectly—but it was defining something by what I hated rather than what I loved. It's like naming your child "Not-Brad" because you had a bad roommate once. Accurate? Maybe. Helpful? Not really.

It took a few tries to arrive at "HTTP for Humans". That shift wasn't just better branding—it was a different way of seeing the problem entirely.

"For humans" became a north star. Every API decision, every documentation choice, every community interaction bent toward that gravitational center. The name wasn't describing the code; it was prescribing a philosophy. And that philosophy infected everything it touched, in the best possible way.

How Names Shape Thinking

Here's what I've realized: naming in code is consciousness trying to understand itself. It's this recursive loop where the names we choose shape how we think, how we think shapes what we build, and what we build shapes how millions of developers think about problems. It's consciousness all the way down.

Consider the difference between these function names:

```python
def get_url_for_profile():
    # Verbose but clear about what it does
    pass

def profile_url():
    # Clean, direct, trusts the reader
    pass
```

Same functionality, completely different philosophies. The first is verbose and procedural—it tells you exactly what steps it's taking. The second is clean and declarative—it trusts you to understand that `profile_url()` will give you what you need. One assumes you need hand-holding; the other assumes you're intelligent.

When thousands of developers import your module, something profound happens: they start thinking with your vocabulary. Your function names become their mental models. You're not just writing code; you're architecting thought itself

> This connects to the broader theme of The Recursive Loop—programmer consciousness shapes collective consciousness through the interfaces and abstractions we create.

.

The responsibility is staggering. Every poorly named function is confusion multiplied by every developer who encounters it. Every clarity is a gift that keeps giving.

No wonder naming is hard. You're not labeling things—you're building the conceptual infrastructure of someone else's mind.

Why Names Matter More Than We Think

Douglas Adams got something important about language: naming creates reality, it doesn't just describe it. When you name something in code, you're defining what kind of thing it is, how it relates to everything else, what operations make sense on it.

Take the simple act of choosing between `user`, `person`, `human`, or `individual`:

- `user` implies consumption and interaction with a system.
- `person` implies social and legal recognition.
- `human` implies biological and conscious reality.
- `individual` implies distinctness and autonomy.

Watch how this propagates: `user.authenticate()` feels natural, but `human.authenticate()` feels dystopian. Users have passwords; humans have names. The noun you choose determines every verb that follows.

The name becomes a strange attractor—all future development curves toward it

> Good naming constraints actually force clarity of thought. When you can't figure out what to call something, it's usually because you don't understand what it actually is yet.

.

Naming as an Act of Care

I think good naming is fundamentally an act of love. You're being kind to the developer debugging at 2 AM, eyes bloodshot, seventh coffee cold. That developer is probably you, six months from now, when all the context has evaporated and only the names remain like ancient ruins, hopefully still legible.

This connects to the broader "for humans" philosophy—technology should serve human mental models, not the other way around. Names are really the primary interface between how humans think and how code is structured.

Consider the difference between:

```
# Machine-optimized naming
def proc_usr_auth_req(uid, pwd_hash):
    """Process user authentication request."""
    if validate_creds(uid, pwd_hash):
        return gen_sess_token(uid)
    return auth_err_resp()

# Human-optimized naming
def authenticate_user(email, password_hash):
    """Verify someone's identity and create a session."""
    if credentials_are_valid(email, password_hash):
        return create_session_for(email)
    return authentication_failed()
```

The machine doesn't care either way. But humans reading the second version can actually breathe easier—the names carry the mental model along with them.

Finding Names That Last

The best names age well. `requests.get()` started simple—get something from a URL. But the name scaled: get JSON, get with authentication, get with custom headers. The verb "get" carried the right weight for all these variations.

Compare that with `XMLHttpRequest` in JavaScript, which handles JSON more often than XML these days. That name carries historical baggage that misleads more than it helps.

There's an art to choosing names that are specific enough to be meaningful but general enough to evolve. Names that point toward what something is rather than how it's implemented.

Creating Shared Understanding

In Genesis, before Adam builds anything, before he plants anything, before he even gets lonely, he names the animals. First job: bring order to chaos through language.

Every programmer gets their own Genesis moment. When you type `class PaymentProcessor`, you're not just naming—you're creating a gravitational center. Future developers won't ask "Where do we handle refunds?" They'll ask "Where does the PaymentProcessor handle refunds?" The name becomes the territory.

You're not just labeling—you're creating the conceptual framework through which others will navigate complexity.

When Names Feel Just Right

Sometimes you stumble into perfection by accident. `requests.Session` just appeared one day, fully formed, obvious. Of course that's what you call a persistent HTTP context. The name was already there in the marble, waiting to be uncovered.

When you're working in uncharted territory, sometimes you borrow metaphors: "Orchestrator" from music, "Factory" from manufacturing

> The danger is when metaphors become prisons—when you can't see possibilities outside the metaphor's constraints. Good metaphors illuminate; bad metaphors obscure.

. These give people conceptual scaffolding.

And yeah, sometimes you write `ProcessorThingy` at 11 PM and promise yourself you'll fix it tomorrow. Tomorrow becomes next week becomes production becomes "why is our core abstraction called ProcessorThingy?"

Names are hypotheses about reality. Good code lets them evolve. Bad code calcifies them into monuments to our past ignorance.

Small Acts of Kindness

Whether names come easily or through struggle, they're all acts of hope. Hope that someone else will understand. Hope that the abstraction holds. Hope that we're building something that makes the world a little more comprehensible.

When you choose `calculate_compound_interest` over `calc_ci`, you're being kind to whoever has to maintain that function at 3 AM. When you choose `requests.Session` over `HTTPStatefulManager`, you're choosing clarity over cleverness.

These small acts of care add up. The names you choose today shape the thoughts that get thought tomorrow.

Why This Matters

So here's what I actually think is happening: we're not just writing code—we're writing the cognitive infrastructure of the future. The names we choose today become the thoughts people think tomorrow.

Better names create clearer thinking. Clearer thinking creates better abstractions. Better abstractions create more space for human consciousness to flourish instead of struggle. Round and round it goes.

Douglas Adams was right—we really are capable of building bridges across the incomprehensible spaces between minds. And sometimes those bridges are made of nothing more than a perfectly chosen name.

A name that says: I see you, future human. I recognize that you're going to be confused and tired. Here's a small gift of clarity.

That's all naming really is—consciousness being kind to itself across time.

This essay examines naming in code as a philosophical act that shapes collective consciousness through cognitive infrastructure. It connects to The Recursive Loop: How Code Shapes Minds on programmer consciousness, The "For Humans" Philosophy on human-centered design, Programming as Spiritual Practice on contemplative technical work, and Building Rapport with Your AI on conscious communication.

For practical guidance, see The Design of Everyday Things by Donald Norman, Code Complete by Steve McConnell, Domain-Driven Design by Eric Evans, and The Hitchhiker's Guide to the Galaxy by Douglas Adams.
