THREE

Why Words Shape Systems

"The word is not a label. The word is a pair of glasses. You put them on, and the world looks different."

The Question You Never Asked

Here is a question that sounds academic but is not.

Why do words have this power? Why does calling something Transaction instead of Settlement change the architecture? Why does Customer lead to different services than User? Why do the fintech, logistics, and healthcare companies all die the same death?

The easy answer is: because words mean things. And different words mean different things. That is true. It is also useless. It explains nothing(sorry, really, really sorry)!

The harder answer is more interesting. Words shape how we think. How we think shapes what we see. What we see shapes what we build. What we build shapes what everyone else sees. The loop closes. The loop tightens. The loop becomes the architecture.

//TODO image

This chapter is about the first link in that chain. Word to thought. Language to cognition. Not because it is the most technical link. Because it is the most invisible link. And because if you do not understand it, you will keep freezing broken words into broken architectures without ever knowing why.

The Glasses You Did Not Know You Were Wearing

Let me start with a confession.

I used to think that words were merely labels. You have a thing. You give it a name. The name does not change the thing. A rose by any other name would smell as sweet. Shakespeare said that. I believed it.

I was wrong. Not about roses. About software.

In software, the name changes the thing. Not because of magic. Because of people. People read the name. The name shapes their thinking. Their thinking shapes their decisions. Their decisions shape the code. The code becomes the thing.

Call it Customer. You think about relationships. Loyalty. Lifetime value. The person who pays. You build features for retention and upsells.

Call it User. You think about sessions. Permissions. Authentication. The person who clicks. You build features for login flows and role management.

Same data, same business and for the same requirements. But different words lead to different thinking that eventually led to different software.

The word is not a label. The word is a pair of glasses. You put them on, and the world looks different. You do not notice you are wearing them. You think you are just seeing reality. You are not.

The Linguists Who Knew This

There is a famous idea in linguistics called the *Sapir-Whorf* hypothesis. Named after Edward Sapir and his student Benjamin Lee Whorf. It comes in two versions. A strong version. A weak version.

The strong version says language determines thought. If your language has no word for a concept, you cannot think that concept. If your language carves up reality in a certain

way, you perceive reality that way. There is no escape. Your language is your mental prison.

Most linguists today think this is too strong. You can think concepts your language does not have words for. You can learn new words. You are not a prisoner.

The weak version is more modest. Language does not determine thought. But it influences it. It directs attention. It makes some distinctions easier to notice and others harder. It shapes habits of thinking that then shape further thinking.

This weak version is widely accepted. And it is the version that matters for software.

A team that calls something Transaction notices different things than a team that calls it Settlement. The first team notices atomicity, completion and database consistency. The second team notices clearing. Legal settlement. Both teams can see the other perspective. But one is easier. One is automatic. One becomes habit. The habit becomes the architecture.

A Simple Experiment

Try this.

Think of the word Order. Write down everything that comes to your mind. A customer. A list of items. A total price. A status. A delivery address. A payment.

Now think of the word Reservation. What comes to mind? A table at a restaurant. A time slot. A deposit. A cancellation policy. A hold on inventory.

Same business. A restaurant could call a customer's request an Order or a Reservation. Different words. Different mental models. Different software.

An Order system has payment processing, fulfillment tracking, shipping costs. A Reservation system has time slots, table management, no-shows, deposit refunds.

The word chose the architecture. Not the architect.

You can try this with your own system. Pick the most important word. Customer or User. Patient or Client. Claim or Request. Write down what each word brings to mind. The differences are not trivial. They are the blueprint for your system.

The Child Who Knew Before Language

Before we get too carried away with language, let me tell you about the other side of the argument.

There is a Swiss psychologist named Jean Piaget. He studied children. He watched them grow from babies who could not talk to toddlers who would not stop talking.

Piaget noticed something interesting. Children develop certain cognitive abilities before they develop language. A baby who cannot yet speak can still search for a hidden toy. That requires object permanence. The ability to know that something exists even when you cannot see it. The baby has this ability before the word *object* or *hidden* or *search*.

Piaget concluded that language is not the foundation of thought. Thought is the foundation of language. Language is built on top of pre-existing cognitive structures. Language is a cloak draped over the body of thought. The body comes first. The cloak comes later.

This is cognitive determinism. Cognition determines language. Not the other way around.

If Piaget is right, then my entire argument about words shaping architecture is backwards. The architecture shapes the words. Not the words the architecture.

So who is right? Sapir-Whorf or Piaget? Language shapes thought? Or thought shapes language?

The answer, as with most interesting questions, is both.

The Dialectic of Language and Thought

Here is where the Russian psychologist Lev Vygotsky enters the scene. He died young. He left behind a body of work that is still provoking arguments decades later.

Vygotsky argued that both Sapir-Whorf and Piaget were missing something. The relationship between language and thought is not one-way. It is not even two-way. It is

dialectical. Language and thought emerge together. They shape each other. They cannot be separated.

He introduced the concept of inner speech. The silent monologue that runs through your head as you think. This inner speech is not the same as external speech. It is condensed. It is abbreviated. It is full of shortcuts and implied meanings.

Inner speech, Vygotsky argued, is the bridge between language and thought. It is language that has been internalized. And it is thought that has been shaped by language. You cannot pull them apart. They are two sides of the same coin.

This is messy. It is bidirectional. It is dialectical. It is also, I think, the most accurate description of what actually happens.

Your language shapes your thinking. Your thinking shapes your language. The loop feeds itself. The loop becomes stronger over time. The loop becomes your architecture.

The Metaphors We Code By

Now we come to George Lakoff. Linguist at UC Berkeley. He has spent decades studying how metaphor shapes thought.

Lakoff's central claim is that metaphor is not just a literary device. It is not just a figure of speech. Metaphor is a fundamental mechanism of cognition. You cannot think without metaphors. You can try. You will fail.

Here is an example. Think about the concept of love. How do you think about it? Lakoff and his colleagues discovered that people overwhelmingly understand love in terms of a journey.

- *"We have come a long way."*
- *"We are at a crossroads."*
- *"Our relationship is going nowhere."*
- *"We have hit a dead end."*

These are not just poetic expressions. They are evidence of a conceptual metaphor running in the background. Love is a journey. Lovers are travelers. The relationship is a vehicle. Difficulties are obstacles.

This metaphor is not arbitrary. It arises from embodied experience. When you are in a relationship with someone, you travel through time together. You make progress. Or not. You encounter obstacles. Or not. You decide whether to continue or turn back. The experience of moving through space provides the structure for understanding the experience of moving through a relationship.

Lakoff calls this embodied cognition. The structure of abstract thought comes from the structure of bodily experience. And metaphor is the mechanism that maps bodily experience onto abstract concepts.

Now apply this to software.

When you call something a Service, you are invoking a metaphor. Services are requested. Services are delivered. Services have clients. Services have providers. This metaphor brings assumptions about how your software components should interact.

When you call something a Repository, you are invoking a different metaphor. Repositories store things. You put things in. You take things out. You search for things. Repositories are passive. They do not initiate. They wait for commands.

When you call something an Event, you are invoking yet another metaphor. Events happen at a specific time. Events are observed by listeners. Events trigger reactions. Events have causes and effects.

These metaphors are not arbitrary. They shape how you think about your system. They shape how you design your components. They shape how you name your classes and methods and APIs.

And most of the time, you are not even aware of them. They run in the background. They are the water you swim in.

Lakoff's work shows that you cannot escape this. You will always think in metaphors. The question is not whether you will use metaphors. The question is whether you will choose them intentionally or inherit them by accident.

The Framing Effect in Software

There is another piece of this puzzle. It comes from behavioral economics. Researchers have shown that the way a choice is framed dramatically affects decision-making.

A medical treatment described as having a "*90 percent survival rate*" is chosen more often than the same treatment described as having a "*10 percent mortality rate*." Same facts. Different words. Different choices.

This is the framing effect. It works because language does not just describe options. Language creates options. It highlights some features and hides others. It makes some outcomes salient and others invisible.

Now apply this to software

When a bug is called a defect, the team treats it one way. A defect is a mistake. Someone's fault. Something to fix and assign blame.

When the same bug is called a gap, the team treats it differently. A gap is a missing piece. A difference between expectation and reality. No fault. No blame. Just work to do.

Different words. Different emotions. Different outcomes.

When a task is called a refactoring, the team treats it as technical work, with low that can be postponed.

When the same task is called a semantic cleanup, the team might treat it differently. But they do not have that word. So they postpone. The debt compounds.

The words are not neutral. They are interventions. They are design decisions. They are architectural constraints.

The Loop That Designs Your System

Let me bring this back to software architecture.

You start with a word. Order. The word brings a metaphor. Orders are placed. Orders are fulfilled. Orders have statuses.

The metaphor shapes your thinking. You think about a linear flow. Created. Paid. Shipped. Delivered. You do not think about branching. You do not think about cancellation. You do not think about returns.

Your thinking shapes your model. You create an Order class with a status field. The status has four values. Pending. Paid. Shipped. Delivered.

Your model shapes your architecture. You build an OrderService that handles the status transitions. You draw a state machine. The state machine is linear.

Your architecture shapes your code. The code implements the linear state machine. It works so chances are that you will ship it.

Six months later, the business needs returns. Returns do not fit the linear model. An order cannot go from Delivered to Returned. The linear model breaks. The team adds a hack. A return_status field. The model becomes messy. The architecture cracks.

The word Order chose the linear model. The linear model chose the state machine. The state machine chose the code. The code could not handle returns.

Not because the code was bad. Because the word was too simple. Order did not capture the full complexity of the domain. But the word was frozen. The architecture was frozen. The code was frozen. The business had to adapt to the software.

This is the loop. Word to thought to model to architecture to code. The code then reinforces the word. The loop closes. The loop tightens.

And most teams never notice.

What This Means for Architects

Here is what I take from all of this.

Words are not (just)labels. They are lenses. They shape what you see and what you cannot see.

Metaphors are not decorative. They are cognitive primitives. They run in the background, shaping every design decision.

The Sapir-Whorf hypothesis, in its weak form, is true. Language influences thought. Not completely, not deterministically, but (of course) significantly.

Piaget was right that thought exists before language. But Vygotsky was also right that language feeds back into thought. The relationship is dialectical. Bidirectional. Messy.

And Lakoff was right that metaphor is the mechanism. We think in metaphors. We design in metaphors. We code in metaphors.

We cannot escape this. We will always have metaphors. We will always have lenses. The question is not whether we use them or not. The question here is whether we choose them intentionally or inherit them by accident.

The fintech company inherited Transaction. The logistics company inherited Order. The healthcare company inherited Patient. They did not choose these words. They inherited them. And they inherited all the assumptions, limitations, and fractures that came with them.

This is why **Language-Driven Design** exists. Not to eliminate language, instead to choose it intentionally. To see the lenses. To pick the ones that clarify rather than distort. To change them when they break.

A Warning Before You Continue

This chapter has been abstract. Cognitive science. Linguistics. Philosophy. You might be wondering when we will get back to code.

We will. The next chapter returns to software. To the hidden layer beneath every architecture diagram. To the concepts that bridge language and models. To the chain that connects words to code.

But I needed you to understand why language matters. Not because someone said so. Because the science says so. Because the evidence says so. Because every failed system says so.

The fintech company did not fail because they had the wrong programming language. They failed because they had the wrong word.

Transaction instead of *Settlement*. *Order* instead of *Request*. *Patient* instead of *Person*. Different words. Different architectures. Different outcomes.

The word is the first line of code you write. Before any language. Before any framework. Before any database.

Choose carefully, and I can sleep easy tonight.