The Interface Is the Curriculum

September 2026

13 min read • 2,888 words

In a study published in 2019, researchers put adults who had played Pokémon extensively as children into an fMRI scanner. When those adults looked at Pokémon, a particular part of their visual cortex responded selectively, in a consistent location across the experienced players. The novices didn't show the same pattern.

> Gomez, Barnett, and Grill-Spector's study compared eleven experienced players with eleven novices. The selectivity appeared in the occipitotemporal sulcus. The authors connected its location to the shared visual conditions of childhood play: small sprites, viewed centrally, on a Game Boy.

This wasn't evidence that Pokémon players have identical brains. The finding concerned functional responses to familiar images, not shared anatomy or personality.

I keep thinking about the screen.

Those children encountered the same little images through the same constrained piece of hardware, over and over, while learning which distinctions mattered. That makes me wonder about all the less memorable interfaces we grow up with and continue using for decades. The desktop. The file browser. The search box. The arrangement of a web page. The grid of an instrument. The vocabulary of a Python library.

What are we learning through all of that repetition?

The Pokémon result doesn't answer the whole question. In particular, it doesn't establish that every interface produces a distinctive brain region, or that childhood findings transfer straightforwardly to adult habits. What it gives me is a concrete place to begin an argument: an interface participates in the formation of thought. It helps establish what we notice, how we categorize it, what we remember, and which actions occur to us as possible.

We usually evaluate an interface by what someone can accomplish with it today. I'm interested in what using it repeatedly teaches them to think tomorrow.

The Abstraction You Learn

An interface is an abstraction. In software, that is quite literal: the interface is implemented in code. It presents selected parts of a system as objects you can recognize and operations you can perform, while keeping other details out of the way.

The folder on a desktop isn't a little paper container inside the machine. It's a way of making storage understandable and actionable. A calendar turns time into cells you can put things in. A waveform makes sound available to your eyes and hands. Each gives you a useful model to work with, and each leaves something out.

When you learn the interface, you learn that abstraction. Its categories become distinctions you can make fluently. Its operations become things you know how to try. There is a feedback loop here: we design interfaces around how people think, then people develop habits of thinking around the interfaces available to them.

Consider the irritation of a familiar button moving after an update. The feature still exists. It may even be fewer clicks away. Someone can explain, quite reasonably, that the new location is more logical.

Your hand goes to the old one anyway.

You had learned a place. Some of the work of finding that command had become a relationship between your memory and the arrangement of the screen. Now you have to spend attention on something you previously knew how to do without stopping. The redesign has a migration cost inside the person using it.

> This is more than a metaphor about muscle memory. In CommandMaps research, a spatially stable overview of commands helped experienced users select commands faster than menus or the Ribbon. That's evidence about learned interface performance, not proof that a toolbar changes your personality.

The learning goes beyond location. A file browser offers a model of knowledge: things have names, things live inside other things, and you can recover something by remembering where you put it. Search offers a different route: recall a distinguishing feature, formulate a query, recognize the result. Neither is universally better. They make different demands on memory and reward different kinds of organization.

Use either often enough and its questions become familiar. Where does this belong? What would I search for? A tag allows something to belong to several categories without requiring several copies. A folder asks you to choose a location. These are decisions about software, but they are also opportunities to practice different ways of relating ideas.

Undo has a lesson in it, too. When reversal is reliable, trying something becomes less expensive. You can move a paragraph, change a color, hear a different chord, and learn from the result. An interface with no trustworthy way back makes caution sensible. The person hasn't necessarily become less curious. The environment has made curiosity more costly.

We are learning the tool, certainly. We are also learning how to be a person operating in that particular environment: where to look, what to attempt, when to hesitate, what kinds of mistakes are recoverable.

That is cognitive activity. It doesn't need a chat box or an implanted electrode to count.

The Shape of the Page Is Part of the Thought

Take the same information and arrange it three ways.

Put it in a table, and a reader can compare one attribute across several entries. Put it in a sequence of full-screen cards, and comparison requires remembering what has disappeared. Put it in an outline, and the nesting offers a claim about which ideas belong inside which others.

The words haven't changed. The thinking available around them has.

A heading gives a section a name you can return to. A paragraph gathers a few sentences into a unit. A link makes a relationship traversable. These are modest decisions, familiar enough to disappear into the background, but they help determine how an argument can be held in mind. Even the possibility of looking back matters. You can inspect an earlier premise without having to reconstruct it from memory.

I wrote The Interface Is the Subconscious with a distinction between visual interfaces shaping feelings and linguistic interfaces shaping thinking. I would loosen that distinction now. A visual arrangement can support reasoning directly. You can see a missing step, a contradiction between columns, or a pattern in a sequence before you have put the observation into a sentence.

> Visual Hierarchy and the Shape of Attention approaches this through the page itself. The further question here is what repeated use of those arrangements teaches, beyond the immediate act of reading.

This is why the layout deserves scrutiny alongside the recommendation algorithm. One chooses what arrives; the other helps organize what you can do with it once it's there.

In a one-item-at-a-time feed, the next available action may be to react or move on. In a document with a visible structure, it may be to compare, reread, or follow a reference. Either format can contain thoughtful material. Neither guarantees a particular state of mind. But they offer different sequences of practice, and I think those sequences matter when repeated for years.

I've written a lot about what the algorithm eats. The interface is where some of that consumption becomes a routine you know how to perform. The thumb learns its part even when the subject changes.

A Shared Education

Scale this across people using the same conventions, and interface design starts to look like a form of mass education with no enrollment process.

There are real benefits to that. A familiar search field lets you arrive somewhere new with a skill you already possess. Shared conventions mean every application doesn't have to teach you everything from scratch. Some of the most humane design work consists of leaving a useful convention alone.

But conventions also carry assumptions. Suppose every contribution in a community receives a visible count. Participants can now compare contributions by that number with almost no effort. Other judgments, such as whether something was generous or changed someone's mind, may require context the interface doesn't display. The count hasn't made the deeper judgments impossible. It has made one particular comparison unusually convenient.

My concern is the cumulative effect of that convenience. Which distinctions become automatic because we are constantly invited to make them? Which capacities go underused because there is rarely an occasion to exercise them? What starts to feel like an inherent property of a conversation when it is actually a property of the software hosting it?

When I talk about the cognitive architecture of a population, this is what I mean: shared habits of attention, shared categories, familiar ways of finding and judging things. I don't mean identical brains. People resist, reinterpret, and repurpose their tools. They bring different histories and abilities to the same screen, and they can learn new habits.

> Learning an interface, changing a habit within it, and carrying that habit into unrelated situations are different claims. The last needs its own evidence. The broader cultural argument here is a proposal about accumulated practice, not a conclusion established by the Pokémon study.

Still, we shouldn't require proof that a feed rewrites someone's entire personality before asking what it repeatedly asks them to do. The practice itself is observable. We can examine it, and we can choose a different one.

An Instrument Teaches Possibilities

I don't want this to become an argument that being shaped by tools is inherently bad. Some of the best things tools have done for me involve exactly that.

In 2013, I wrote about understanding Ableton Push. I had spent more than a decade playing drums. My musical thinking gravitated toward rhythm and geometry, and the layout of a piano keyboard could be frustrating. Push gave me a grid through which I could approach harmony differently. Within hours, I was playing a melody in a harmonic minor scale that would have been challenging for me on a conventional keyboard.

> The old post is an enthusiastic first encounter, complete with instructions to order one immediately. Beneath the excitement was a fairly durable discovery: a different arrangement of controls changed my access to an existing musical capacity.

Same person. Same underlying relationships between notes. A different surface through which to discover them.

The grid gave my hands patterns to learn. Those patterns made certain experiments inviting, and the experiments produced sounds I could respond to. The interface was involved in forming the musical idea, not merely recording an idea I had already finished elsewhere.

There is a lovely, very literal version of this in research on Tetris. Players sometimes rotate pieces to make it easier to recognize how they fit. The action helps them work out the answer. It doesn't simply execute an answer they already know.

> Kirsh and Maglio called these epistemic actions: actions that make a problem easier to think about, distinguished from actions whose purpose is directly advancing the task. Their 1994 paper studied this distinction in Tetris.

Sometimes the thinking happens through moving the thing.

A good creative tool makes room for that loop. You try something, perceive a result, adjust, discover a possibility you couldn't quite imagine in advance. Repetition can build fluency, and fluency can make more ambitious thought available. This is the part I want more of. Tools that help people develop capacities they are glad to have.

HTTP Is an Interface, Too

Requests belongs in this argument, which means I don't get to stand outside it and complain exclusively about other people's software.

An API has no obligation to draw a button. It offers the same basic bargain as a GUI: here is a manageable representation of the system, and here are the operations available through it. Its names are categories. Its objects group things you are invited to consider together. Its defaults decide which questions you can postpone. Its errors determine what information you have when your model of the situation turns out to be wrong.

When I was developing Requests, I started by writing the usage I wanted to exist. The README came before the implementation. I wanted the act of making an HTTP request to be something a person could express directly, without assembling a small monument to incidental complexity first.

> I described that process in How I Develop Things and Why. Looking back, writing the example first was also choosing the concepts I wanted someone else to be able to work with.

```
import requests


response = requests.get("https://httpbin.org/get", timeout=10)
response.raise_for_status()

data = response.json()
```

There is a little model of the world in those lines. Ask for something. Receive a response. Check its HTTP status. Interpret its contents. Those are distinct operations, with names you can reason about. The abstraction makes a complicated exchange manageable without pretending the exchange cannot fail.

And the choices are not beyond criticism because the library says "for humans." Requests has no default timeout. A person who learns from an example that omits one can carry that omission into real software. Examples are interfaces, too, and the habits they teach deserve the same scrutiny as the function signatures.

> The Requests documentation explains this explicitly. A timeout value also isn't a deadline for the entire download. Even a simple-looking parameter needs a model behind it.

That's the responsibility I keep coming back to. Making a task easier is valuable. Making it easier in a way that leaves someone with a useful understanding is better still. A humane abstraction gives you a place to begin and a way to investigate when beginning is no longer enough.

Computational Thinking Is for People

One kind of thinking I very much want our interfaces to cultivate is computational thinking: breaking a problem into parts, choosing useful representations, following a sequence of operations, checking assumptions against results, and recognizing when an abstraction has hidden something you now need to understand.

> Jeannette Wing's 2006 essay on computational thinking argued for its relevance beyond computer science. Her distinction matters: these are ways for humans to approach problems, not an instruction to make human thought imitate a machine.

These habits are useful in ordinary life. They become particularly important when ordinary life is so thoroughly mediated by software. Employment, money, communication, access to information: we encounter systems that classify, rank, route, and decide things about us. Being able to operate their interfaces doesn't necessarily mean being able to question what they do.

Computational thinking gives a person questions to ask. What were the inputs? Which rule produced this result? What is being measured? What happens when a value is missing? Who chose what counts as success?

Imagine a form that insists your situation cannot exist. A little experience with data models can help you recognize a possibility: perhaps your life isn't the error. Perhaps someone made a field required because their model didn't account for

you. That understanding does not magically get the form submitted, but it changes where the problem can be located and what you might ask someone to fix.

A default becomes something somebody selected. A ranking becomes the output of a process with assumptions. An error becomes available for investigation. Those are small openings for agency in systems that often present their decisions as finished facts.

Our interfaces can help cultivate that agency. A spreadsheet can expose the formula behind a total. A search tool can show which filters excluded a result. A workflow can distinguish what happened from what was expected, then let someone revise an input and try again. None of that requires turning every user into a professional programmer. It requires treating understanding as a worthwhile outcome of using software.

There is a limit I want to keep visible, particularly as someone inclined to think in systems. A person is not an invalid record because they don't fit a schema. A relationship is not a process that will become painless once its inefficiencies are removed. Some of the most important things we know are bodily, emotional, or difficult to articulate at all.

> The Un-Englishable is about that remainder. Being able to reason with a model should include being able to notice what the model cannot hold.

Computational thinking belongs alongside those capacities. It can help us understand the machinery well enough to challenge its account of the human being standing in front of it.

What Are We Practicing?

A designer can't determine what another person will become. But a designer does help determine what that person is repeatedly invited to practice.

Does the interface make comparison possible? Can you recover from a mistake? Can you inspect why something happened? Does it leave room to pause, or is every available gesture another way to continue? When you become proficient, are you more able to pursue your intentions, or mainly faster at responding to someone else's prompts?

Those questions belong in ordinary design work, alongside whether the text fits and the button is discoverable. They also belong in accessibility work. Familiarity, understandable state, and recoverable actions matter especially when someone has less attention or energy to spare. Designing for the worst day includes respecting the knowledge a person has already invested in your tool.

I want software that makes people more capable of noticing, investigating, composing, and deciding. Sometimes that means exposing a useful distinction. Sometimes it means handling a tedious detail so someone has room to think about something that matters. Sometimes it means keeping a button exactly where it was.

The curriculum is already there, in the repeated encounters. We can be more deliberate about what it contains.

I used to ask whether an API was pleasant to use. I still care about that. Now I also want to ask: what is someone practicing when they use my software, and what are they becoming better at?
