# Dialogues on Natural Code

Lu Wilson
TodePond
London, UK
todepond@gmail.com

David H. Ackley
Living Computation Foundation
Placitas, New Mexico, USA
ackley@livingcomputation.org

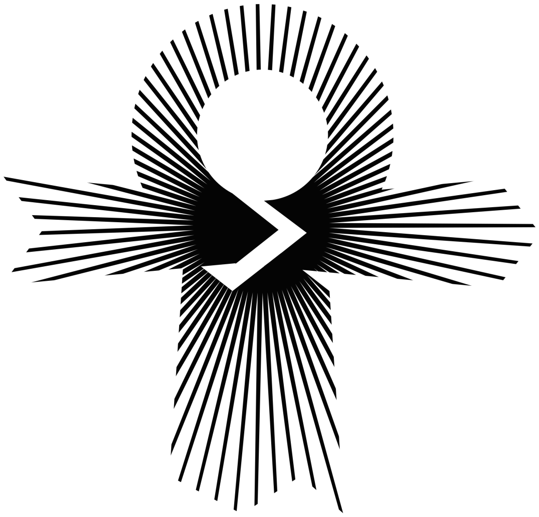


**Figure 1.** The **SelfImage** starburst.

## Abstract

This essay, based on a series of discussions between the authors, is a loosely edited collage in which we work to flesh out our shared interests in non-traditional machines and coding mechanisms. We primarily focused on the idea that all human language can usefully be viewed in programming language terms — as “natural code”. Programming languages and natural languages differ in many ways, such as having relatively formal definitions versus not, emphasizing strong syntax versus large dictionaries, and demanding rigid implementations versus building on the vagaries of living systems. Still, we saw deep unities as well, much more than mere metaphor, and we glimpsed the possibility of applying humanity’s decades of programming language design and software engineering experience to the task of debugging and refactoring the natural codebase that we all share. These fragmentary and overlapping dialogues represent both a description and an example of natural code, and we offer them here, with a simple “natural API” illustration, in hopes of *programming* people to join in natural code development.

***CCS Concepts:*** • **Software and its engineering** → *Very high level languages*; • **Computing methodologies** → *Distributed computing methodologies*.

***Keywords:*** Natural Code, Human Computation, Robust API Design, Implementability

**ACM Reference Format:**
Lu Wilson and David H. Ackley. 2024. Dialogues on Natural Code. In *Proceedings of XXXXXXX (SPLASH ’24)*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Being machinery

**DAVE:** I think living organisms can be meaningfully viewed as machines.

**LU:** Sorry, what?

**DAVE:** They’re physical arrangements of matter that move and do work. They have power supplies. Living systems are machines.

**LU:** Including us?

**DAVE:** Including us. We’re machines.

**LU:** Really? I don’t feel like a machine.

*SPLASH ’24, October 20–25, 2024, Pasadena, California, United States*

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

**DAVE:** I mean, people usually think of machinery as metal and screws and batteries, and I have *very few* of those in my actual living body.

**LU:** A non-zero amount?

**DAVE:** I want to take machines way beyond metal and screws, and say: Any time matter is arranged in space, and an energy supply is incorporated so that the arrangement of matter and energy can *do something* — that's what we're talking about as a machine. And that description is as true for screws and metal as it is for people and amoebas.

**LU:** I don't know if I *want* to think of myself as a machine though.

**DAVE:** It can be uncomfortable, but when we go to the doctor, say, we *want* them to be talking about us in mechanistic ways, like "`the heart machine is not working as well as it could`" or whatever. This framing of a living system as a machine can be useful when we're trying to understand how it works, and how to make it work better.

### 1.1 Building machinery

**LU:** As well as *being* machinery, living things are also capable of *building* machinery. That's what you're saying, right?

**DAVE:** That's right. Machines that somehow work to preserve their structures, their *patterns*, are what we call "`life`". Persistence involves maintenance and repair, but also building copies.

**LU:** I guess so! Though I was thinking more about traditional ideas of "`building machinery`", like a beaver building a dam, or a wasp building a nest.

**DAVE:** That happens too. And humans build bridges, rockets, and programmable computers. I think about "`building machinery`" writ large. It can be something like lighting a fire, or folding a paper airplane, or moving a rock off a path.

**LU:** You're using the phrase "`building machinery`" extremely loosely here, right? Because to me, "`building machinery`" sounds like *creating* something, or *making* an artifact of some sort. But you're using it to refer to what seems like just an action, or a process.

"`Lighting a fire`" doesn't sound like building anything at all. It just sounds like *enacting a change*.

**DAVE:** Yeah I screwed that up. Collecting wood and stuff is building the machine. Lighting the fire is flipping its switch.

But, say you're working at a hamburger joint, where all you have to do is slap a burger on a bun and put on ketchup or mayo, and it's done. You're "`building a machine`" out of other complex arrangements of matter.

**LU:** You're changing the arrangement of the burger's ingredients, and that's what you're calling "`building machinery`". It's not that you've "`created`" these ingredients, but you've built them into a particular pattern.

**DAVE:** Yes, you're arranging matter to get certain properties.

**LU:** Okay so you said that a burger is a machine and —

**DAVE:** The reviewers had some troubles with that.

**LU:** And I can understand their troubles. You said that a machine can *do something*, but a burger just sits there.

**DAVE:** I — Fair enough. I understand. I mean, there are many power sources for machines. You could have a battery, or gasoline, or gunpowder. But you could also have a human.

**LU:** A human?

**DAVE:** Like, an old-fashioned well pump is a hand-powered machine. You pump the handle, and water comes up out of the spout and helps you live. It's a human-powered machine.

And maybe a hamburger isn't the cleanest example —

**LU:** It really isn't the cleanest example.

**DAVE:** — but the hamburger machine runs on muscle power too. You pick it up and chomp it on down, and it absolutely *does something*: It feeds you and helps you live.

**LU:** You're stretching the use of the language quite a bit, but what you're saying is — when you're building machinery, you're building a pattern.

I could have some LEGO bricks on my table, and they're all scattered around. I could build something new just by moving them around. I could build a pattern, or a house. Either way, I'm building machinery just by rearranging. Is that how you see it?

**DAVE:** Right. Arranging matter. A house is a pattern too.

### 1.2 Contracting machinery

**LU:** And you're saying there are two ways of building machinery? One way is to do it yourself, to build it *directly*.

**DAVE:** Wood, hammer, nail. Yeah.

**LU:** And the other way is by getting *another machine* to do the work for you. You can instruct it to do the building on your behalf. In this case, you're building *indirectly*.

**DAVE:** Yes, you find a programmable machine that's out there in the world already. You don't have to build it yourself. You ship some code, and have that machine do the work for you. When you don't have to send the wood or the tools, code is incredibly cheap to ship. That's its superpower.

**LU:** And that programmable machine could be anything we can transmit code to, like a mechanical arm in a factory, or a rocket, or a computer.

**DAVE:** Or we flip the switch on the wall. We want light.

**LU:** Okay, I see where this is going.

## 1.3 Human hardware

**LU:** You're saying that "the programmable machine could be a person".

**DAVE:** Right. As humans, we can transmit code to another person and get them to do something for us. We can say, "Hey, can you help me build this shelter?" or "Can you build a fire while I gather food?".

**LU:** I'd argue that animals do that too, right? Living things often communicate with each other in some sort of way.

**DAVE:** It's certainly a spectrum. Maybe an animal sends a signal that means "run" or "danger" or "food".

**LU:** Either way, you're saying that we can code one another. Asking someone to do something is coding them, in a way?

**DAVE:** Yes, we transmit "natural code" all the time — when we talk with each other, or teach stuff to our kids. If I was trying to wrap it all up in a box, I'd say

> **I think we should use our knowledge of programming languages, of software and computing, to examine our own natural code. To understand it and debug it. To make society better, and to improve our shared codebase.**

This is why I want to push for a view of computation broad enough that we can see humans as *programmable* machines — that are programmed by "natural code".

## 1.4 Coldness and evil

**LU:** This idea that people "program" other people. To me it seems —

**DAVE:** It seems really obvious, right? It helps us to —

**LU:** No. Actually, I was going to say that it seems really cold.

**DAVE:** Oh. Well.

**LU:** It almost seems psychopathic, because it sounds like it's all about trying to manipulate other people.

**DAVE:** Well, I —

**LU:** But communication isn't only for influencing people. We also talk to share our feelings, and connect with others. Or we just want to be heard, or rant, or share a joke.

**DAVE:** Right! And I think that's a good —

**LU:** So we can't boil down communication to just "getting someone to do stuff" because that's cold, and it's not true!

**LU:** Reviewer C is worried about "the ideological, technocratic undertones" of the essay, and "it's a pervasive fallacy in the tech world to see all our problems as technological" and "Every human interaction is reduced to a kind of programming".

**DAVE:** Yeah. And how do you react to that?

**LU:** I was genuinely worried about this when we submitted, because it's something I agree with. There *is* this pervasive fallacy to see all our problems as technological. I hate it, and I see it time and time again.

Like recently, I've been hearing more and more people around me saying that "all we need is better technology" and all our computer accessibility issues will disappear.

**DAVE:** I just can't imagine somebody saying that seriously.

**LU:** For example, I read a recent essay [17] saying that "AI will soon come to the rescue" for accessibility.

Or take the climate crisis. There's this fallacy that we don't need to worry about reducing our energy usage, or replacing our energy sources [10] because —

**DAVE:** "We will technology our way out of it". Carbon capture, seeding the clouds, or whatever we can tell ourselves to delay dealing with the real problems.

**LU:** Exactly. In these cases, the actual solution is to *not* see the problem as mostly technological. Instead, the solution is to try to change our behavior, both as individuals and as a society. I think this is where natural code can help. It can give us a new perspective and understanding of our communications and how to improve them.

**DAVE:** One answer to such criticisms is that we are reading the concept of "technology" broadly enough to include stuff that's not traditional technology. People can hear us say "technology" and think it means traditional programming languages and computers and "tradtech" generally.

**LU:** Right, we say "natural code can help us" but sometimes people hear "traditional technology can help us".

**DAVE:** But really we're saying "technology writ large is much bigger than tradtech" and part of that is understanding ourselves better — that we can be viewed meaningfully as machines, and our communications can be viewed as code, and we build more machines to help keep ourselves alive.

**LU:** And we exchange code with each other.

**DAVE:** For sure. We are coders. We ship code.

**LU:** I mean, it's a tricky idea to sell. And it does sound quite "technological".

**DAVE:** And I think we just have to own that. But we also have to stress that judgment goes beyond just the tech. Shipping code "to make money" is different than "to help society", no matter how tech hypocrites may try to conflate them.

**LU:** If anything, I think we are calling for *fewer* problems to be seen as solvable by tradtech. For example, at work, we wanted to make it easier to hear each other on our video calls. We got new tradtech — software, microphones — but still had problems.

**DAVE:** And the real solution was like "talk slower"?

**LU:** It's mainly "avoid cross-talk" and "be sure to set up everything properly". In that situation, deploying "natural `code`" is what improved things. We actually wrote up a document — guidelines for behaving in meetings. And for me this is a form of "`natural code`".

**DAVE:** Like how modern programming projects often have an explicit "Code of Conduct." That's "`natural code`"!

## 2 Beyond determinism

**LU:** Another obvious objection to these ideas is that humans seem really different to computer hardware, because computers are absolutely rigid and repeatable. They're *deterministic*, and humans are not.

**DAVE:** Deterministic execution of code has always been an illusion. There's always the possibility of cosmic rays coming in and flipping a bit, say, and that does happen sometimes [25]. But we know that we can engineer traditional computer hardware so that the chance of that is small enough that we can usually ignore it.

**LU:** But someone could still come and turn off your computer's power, right?

**DAVE:** Right, or overheat it.

**LU:** Or smash it with a hammer.

**LU:** In web development, when you do a "`fetch`" request to an endpoint, you usually use your own special kind of "`fetch`" function that automatically retries a few times [18].

**DAVE:** Right, because in the network world —

**LU:** In the network world, things can go wrong, and in fact, they often do go wrong [16, 19]. So you run the same code again and again, to increase the chances that it will work.

**DAVE:** People certainly don't do the same thing every time.

**LU:** So when we transmit code to a person, we can't know for sure what the effects will be. They might ignore us, or say no, or do something completely different.

The essay might make no sense to them, or they might get it but disagree. But even if the chance of convincing them is low, we might still think that it's worth a try.

**DAVE:** Yeah, maybe we'll succeed. Maybe we won't. The machines executing the code of this essay are going to be way non-deterministic.

**DAVE:** I've been trying to get ideas like `natural code` across for a long time [2], and it's been hard. People bring all of their traditional computing misconceptions to it. And the idea of `natural code` just looks crazy to them.

**LU:** Has non-determinism been a blocker for some people?

**DAVE:** Some people would outright say "`without deterministic execution, it's not computation.`"

**DAVE:** There's this idea that "`if you can't predict exactly what the code will do, it'll be chaos`". My claim is no, we can still talk in terms of computation and code, even if the "`computer`" is not fully deterministic.

Even if we only have a 51% chance that some code will work versus a 49% chance that it won't, say, we might still want to run the code, again and again, for that 2% edge.

**LU:** I've been thinking about how we can get across this "`non-determinism idea`", and I wonder if we can use the format of the essay itself to help us dripfeed it throughout.

**DAVE:** Oh I see, bits of conversation out of order, and so on.

**LU:** Yes, we don't need to be strictly chronological. We can jump around and revisit things. When we transmit natural code, we don't know exactly how that code will be executed. We don't know what the exact order of execution will be either, but we can still talk about it in terms of code and computation. It's still possible to do that.

**DAVE:** Perhaps also showing how we can bend the familiar overall "`syntax`" of a paper, but still transmit legible code.

**LU:** Someone could skip ahead to the end of the essay, or miss out a whole section, or just look at the diagrams.

## 3 Prior "`art`"

**LU:** But, Dave: Why put this essay forward as a submission to a programming language conference? Why not go to a philosophy conference, or "`art`"? Why enter through programming languages as a lens?

**DAVE:** I mean sure, if we had more time and more collaborators, we'd go to all those conferences — a full court press — and then FOMO would descend, and the world would change.

**LU:** "`FOMO`" as in "`Fear Of Missing Out`"?

**DAVE:** Yes, if we could figure out how to —

**LU:** If we could market this "`natural code`" idea in all those conferences, lots of people might get "`FOMO`" and get involved.

**DAVE:** And that would be great. But we can only do what we can figure out how to do — can only do what's "`implementable`" for us at the time.

I do want to poke the bear a bit, and it seems appropriate for a venue like *Onward! Essays* that's explicitly aimed at computation and programming languages writ large.

**LU:** Yeah, I see that. I think it's helpful for you to share why you're coming through programming languages, because people reading this might think there's a particular reason behind that. But it sounds like it's partly just because that's where you're starting from.

**DAVE:** Right that's my history. Code's what I know best.

### 3.1 Historical traditions

**DAVE:** It's like philosophy, psychology, and all those things, are trying to describe *what we are* — what our touchstones and key concepts are, how we see what we see, and so on. I think, despite their great successes, such fields have deep assumptions that limit how clear and effective they can be.

I think we should start again with notions of programming languages and software engineering, but move beyond deterministic execution. Then we can start talking about our human collective computation in terms of APIs, programming languages and structures, compositionality and modularity, and so on.

The goal is: Whenever we speak, we can always know, or plausibly believe, that what we are saying is *implementable*. We could always, at least in principle, build a machine — using ordinary silicon chips or exotic biological bricks or whatever — that could *run the code* we're shipping. Then we point at the machine and say "`I mean like that!`" And that's what we cannot do with philosophy or psychology or religion or anything, that we maybe could do if we say "`Let's pretend natural language is code`".

### 3.2 Implementability

**LU:** I would challenge the idea that `natural code` is the only route to implementability. I think that neuroscience, say, or even physics, offers implementability in some way.

I know there are studies out there where they've taken an organism, a *hydra vulgaris*, and they've mapped out its entire neural networks, and they've used that to get closer to determining how the creature is implemented [13].

**DAVE:** I certainly do not want to say that natural code is the *only* route to implementability. I would argue that it looks like the most *direct* route to implementability.

Driving around a cockroach by putting wires into its spine [20] is clearly building a piece of living machinery, working at a pretty low level. But in the computation world, instead of writing assembly code, we glue together giant stacks of software and plug one abstracted part into another.

I would argue that, if neuroscientists build more machines out of more neurons, displaying more complex behaviors, they'll stop talking about that overall machine in terms of neurons. They're going to start talking about it in terms of inputs and outputs, and parallel and sequential processing — in terms of computation and code.

**LU:** So you think that it all comes back to computation in the end?

**DAVE:** Back to *implementation*. I find neuroscience and biology results inspirational for seeing how nature does things. Many perspectives help! I argue that `natural code` is yet another point of view that can be a useful framing for understanding our world, and making it better.

### 3.3 Related work

**LU:** Okay, okay. But I don't think that this "`Prior Art`" section actually covers any prior art so far. It feels like a rejection of everything existing. `Natural code` can't be *that* new, right?

**DAVE:** Of course, lots of things are connected. Dan Dennett's ideas had a big impact on me personally, for one.

**LU:** I saw you tooted a little remembrance about him. [5]

**DAVE:** Yeah, he was so clear. With his notions of descriptive "stances" [12], I see `natural code` as a way of connecting the `intentional stance` with the `physical` and `design` stances.

**LU:** I'm reminded of Alexander's pattern language stuff too [8]. His "patterns" are like code, describing how to solve various problems through architecture and design. And there's an emphasis on the patterns being "`tentative`" and unpredictable. There is a non-deterministic aspect to it.

**DAVE:** Right, and of course design patterns [14] have similar flavors. Language not quite executable on a computer, but very "code like" and absolutely executable on *developers*.

**LU:** For me, these examples demonstrate that we can spot aspects of `natural code` within existing works, perhaps implicitly, and what we're trying to do is—

**DAVE:** We're trying to *explicitly* frame things as code.

### 3.4 Blending fields

**LU:** Personally, I seek out the projects that aim to blend numerous fields, like those that combine science and art in some way, or those that try to bring together different categories of research. It's not always easy to do, but I think it's often where the most impactful work can be done — you get to pick and choose the strengths of various fields, and get the "`best of both worlds`" in many cases.

**DAVE:** Let me be completely honest. My problem combining art with science is that the results often feel a bit like the *worst of both*. You know, not great science, not great art, no impact at all. And so I feel that art is too —

**LU:** You make a few art pieces though.

**DAVE:** Well —

**LU:** Yeah, it's funny hearing you criticize using art, because from my perspective, you seem to do a lot of art.

**DAVE:** What? What!?

**LU:** Yes, I mean, I would —

**DAVE:** Name one!

**LU:** The **`SelfImage`**. That's art! (See Fig. 2.)

**DAVE:** Okay, I see that as computation, I guess.

**LU:** This is how I see it. I think you're in this world of trying to get different fields to put their heads together, and learn from each other.

**DAVE:** Yeah.

**LU:** And maybe you see a divide between the "`art world`" and the "`non-art world`." But for me, it isn't helpful to draw these lines when trying to bring the different fields together.

I accept that you don't need to *open* with art. You can open with something else and then sucker-punch with art, right?

**DAVE:** Yes, yes, yes, it's like "`just kidding, it was all a dream`".

**LU:** "`It was art the whole time`".

**DAVE:** For the **`SelfImage`** in that sense, you are 100% right. There *is* an art component to it, and a marketing component — an attempt to be viral, which I have completely failed at.

**LU:** Except —

**DAVE:** Well I mean, everybody wants the next zero on their views, on their citations, on their patreon, whatever it happens to be. But I'm still only down at the sort of two to three zeroes range, so, you know, I can legitimately claim lack of virality, and — well, anyway, that's another topic.

**LU:** Yeah okay, I just think it's good I got you to admit that the **`SelfImage`** is art.

## 4 The nature of natural code

**DAVE:** The canonical Chomsky hierarchy stuff [11] is all about languages having compositional, recursive, syntactic structures, allowing language users to create open-ended complexity. And I think that's great, but it doesn't go nearly far enough. On their own, syntactic properties are almost a detail. There's other ways to get modularity, complex representations, and so on. For example, you could just list chosen words in a random order — "`wood, hammer, nail`" — and it could create a notion in the listener's head that could be quite rich, with hardly any syntax.

**LU:** Splinters.

**DAVE:** Right. Sore thumb. So I'm hesitant to embrace the idea that it's all about *language* and which structural properties of language are important. I think that's wrong. Instead, I want to talk about "`code`", and not "`programming language`". And by saying `code`, I want to rope in signals, gestures, grunts — stuff that seems below the level of programming languages.

### 4.1 Starting from signals

**LU:** Okay, "`code`" "`code`" "`code`". Not just language. I think that's right. You can get too focused on the structure and syntax of language. I think it's more important to think about the *purpose* of language — the purpose of code, I mean.

When I was a teacher, I worked with very young children who struggled to communicate with other people, for various reasons. It wasn't that these children necessarily struggled with *language*. In fact, some of them were hugely competent with language and its syntax. They struggled with *communication* in a more general sense, which can sometimes involve no syntax or language at all. It can mean "`prodding someone`", "`looking at someone`", or simply "`tugging on their hand`" to pull them along.

The first step that we always tried to get across to these young children was, "`look at all the good things you can get from interacting with someone`", and we used a lot of *biscuits*.

Most children love biscuits, right?

**DAVE:** Cookies.

**LU:** And if you can tell them, "`look, you can prod me, point at a biscuit, and I will give you a biscuit`", then you can show them the purpose of communication. And in some way there's very little syntax or structure to learn there.

For the next step, we did this thing called PECS with some of the children. It's a Picture Exchange Communication System [9] where they can give me a little bit of card that has a picture of a biscuit on, and I give them a biscuit in return. So the key thing here is the code. This card is this executable program. It says "`give me a biscuit`".

The funny thing is, once a child realizes, "`oh I can get what I want from this`" and "`I can make people do things`" then they quickly become very motivated to learn how to communicate more complicated things.

**DAVE:** That's great. I do think you're right. That example gets to the heart of what bugs me about abstract language discussions versus all-in `natural code`.

What matters is that a communication occurs, and that it causes something to happen. It causes the world to become better for the transmitter. If the act of transmitting code, by holding up that picture card, actually leads to "`yum yum`" then all the syntax and stuff can come later. I think it could really help if we thought of programming languages starting from no syntax, starting from just signals.

### 4.2 From spatial computing to symbols

**DAVE:** A key aspect of what you said is that it relies on spatial computing [e.g., 1, 24]. You said "`point at the biscuit and I will give you a biscuit`". That depends on being physically close to the thing that you're indexing because you cannot say "`biscuit`" yet. You don't know how to do that, but when it's close enough, you can just indicate *that thing right there*. And that's how semantics *begins*.

Then going to the cards is great as a next step because that is an example of a *pointer dereference*. You have a symbol that, physically, is just some ink on paper, and yet it can *refer* to a biscuit, and program someone to bring it to you, even if it's in another room, out of sight.

**LU:** We talked about it as "`symbols`". That's the terminology we used in that field of education, and it's the terminology I use now when I talk about coding. That symbol could be the child `pulling on your coat`, or a particular made-up sound, as long as you know that it means "`biscuit`".

**DAVE:** Right right, it could be anything. All that matters is that there's a shared understanding. It's a little specific API.

### 4.3 "`Natural code`" as a symbol

**LU:** When we saw children make a jump to verbal language, it was often when those first symbols just became more inconvenient. Getting out the biscuit card from your little pack of cards becomes a chore. Then you realize that it's much quicker and more effective to just say the word, "`biscuit`".

And now I see that happening with me and you too. Sometimes, I want to refer to a concept that we've previously discussed, but in a much more concise way, and we don't have a word or symbol for that concept yet, so we keep having to go through it in its entirety again and again. I mean, we can edit that out in the essay, but it's very time-consuming for us right here, right now.

So the solution, of course, is to make a symbol that can serve as an abstraction. We need a word that we can *dereference* to get a whole concept. And that's what the term "`natural code`" can be. It can refer to this shared understanding that we're building.

**DAVE:** I see. So now, now you're at a meta level.

**LU:** "`Natural code`" is a symbol. It's a namespace. It's an API that we can use to make our communication more effective. But it only works if we both understand what it means, so that it's a compatible format for us both to use. That's exactly what we're doing in these dialogues — we're developing a shared language — we're developing our shared codebase.

## 5 The `SelfImage` API

**LU:** So, Dave: What *is* the **`SelfImage`** API? I know from your video [4] that it has four processes, but what does it mean?

**DAVE:** Fields like philosophy and religion and science offer us *language* to talk about what kind of machines we all are. Like, "`I think therefore I am`", or "`I am a collection of neurons`".

**LU:** Or "`We are made up of needs and wants and motivations`", or whatever.

**DAVE:** Right. All of these languages contain some germ of truth, but none of them are going to be wholly sufficient to answer all of the variety of questions that we might want to ask. So what we need to do is choose multiple approaches — multiple languages. I think of them as "`APIs`". They're clearly not perfect, and don't cover everything, but they emphasize certain parts, and make it easier to express some concepts versus others.

So the **`SelfImage`** (see Fig. 2) is such an API. It depicts us as arrangements of four computational processes:

1. **`Input`**: Handling influences from our surroundings,
2. **`Output`**: Performing work on our surroundings,
3. **`Sequence`**: Changing internal states over time, and
4. **`Judge`**: Assessing situational desirability.

If we're interested in how we understand the world around us, we'll focus on the **`input`** process. If we want a deeper understanding of how we actually create and do things in the world, we'll unpack the **`output`** process, and so on.

The **`SelfImage`** is a really basic framework to see ourselves through a computational lens. It's a starting point.

### 5.1 API design

**LU:** To me, the **`SelfImage`** API seems no different than a psychological model that aims to describe how people behave. It reminds me of something like Maslow's hierarchy of needs [15], or operant conditioning [21], even.

**DAVE:** Ah, okay. What I'm suggesting is that, by taking the computational metaphor, the **`SelfImage`** API can simultaneously describe both people *and* other programmable machinery. That's one difference.

And secondly, I'm claiming that the **`SelfImage`** API leads more directly to implementability than a psychological description, because it uses the language of computation.

**LU:** So it's not solely a *descriptive* model?

**DAVE:** Right. It can be a blueprint. It can be a recipe for how to build machinery.

**LU:** Okay, it seems more like a *design* challenge — you want to make an API that's useful, regardless of how truthful it is as a description.

**DAVE:** A scientific theory succeeds when it gives us an *unexpected truth.* But that's not the goal of an API in software design. We want an API to be as unsurprising as possible. We want to adhere to the law of least astonishment. [22]

Ideally, an API should not teach us anything new. The goal of an API is to be *obvious*, and that's what we can judge it on — how universally obvious it is.

**LU:** I think I get it. It's more like user experience design, in a way. It's a communication tool that lets us talk about the world in a certain way — under a computational lens.

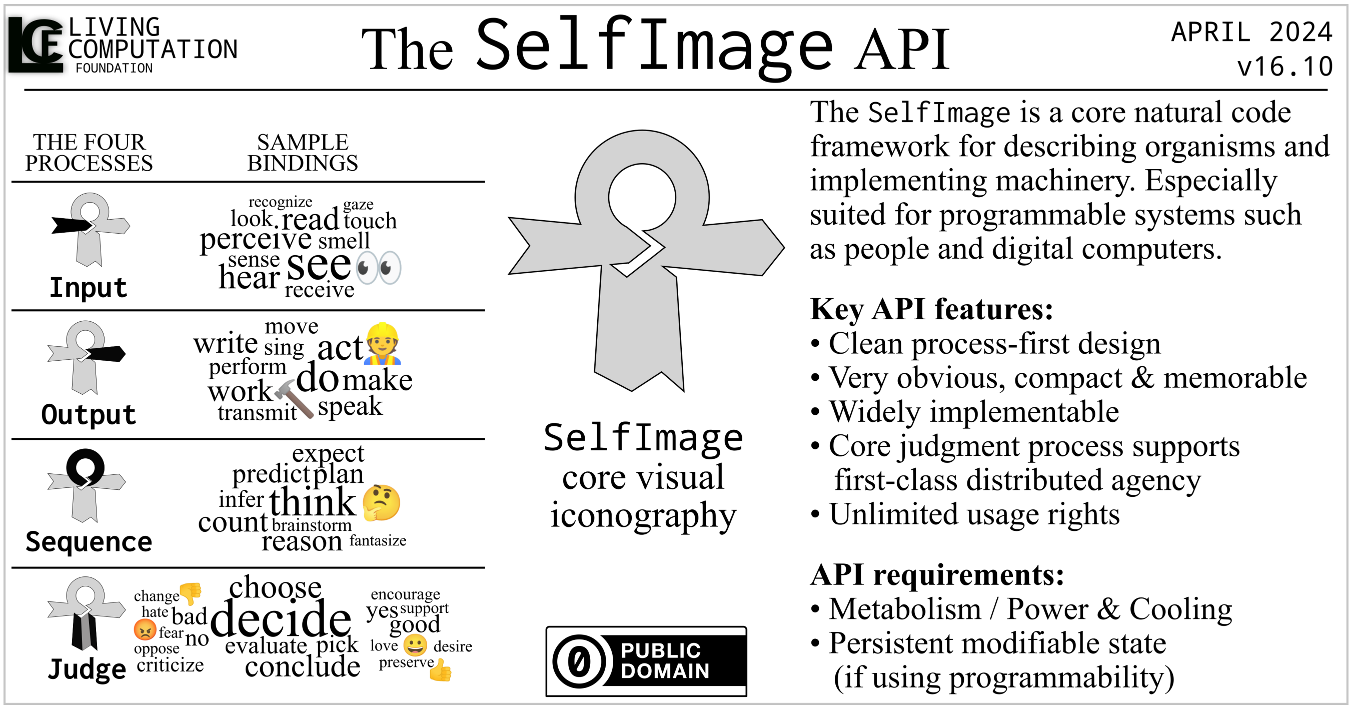
It should be as easy and straightforward to use as possible.

### 5.2 Shared code

**LU:** Sometimes, when I'm developing computer code, I use some tooling to help me, like Google Chrome's DevTools, to see what code is being executed, where it crashes, and so on.

But sometimes the tooling doesn't show me enough helpful information, so I draw my own visualizations of my code's execution — on a piece of paper, or a whiteboard, or a virtual whiteboard like tldraw [23]. It could be a drawing of a state machine, or a flowchart, or a memory layout. Regardless, my drawing is a highly simplified version of what's actually happening in execution.

On top of that, my drawings become a shared language that I can use to communicate with my colleagues. They can look at my visualization and understand what I'm trying to achieve. And if they have a suggestion for how to improve it,


LIVING
COMPUTATION
FOUNDATION
APRIL 2024
v16.10
THE FOUR PROCESSES
SAMPLE BINDINGS
Input
recognize gaze look read touch perceive smell sense see hear receive
Output
move write sing act perform do make work transmit speak
Sequence
expect predict plan infer think count brainstorm reason fantasize
Judge
change hate bad fear no oppose criticize choose decide evaluate pick conclude encourage yes support good love desire preserve
SelfImage
core visual
iconography
PUBLIC DOMAIN

The SelfImage API

The SelfImage is a core natural code framework for describing organisms and implementing machinery. Especially suited for programmable systems such as people and digital computers.

**Key API features:**

- Clean process-first design
- Very obvious, compact & memorable
- Widely implementable
- Core judgment process supports first-class distributed agency
- Unlimited usage rights

**API requirements:**

- Metabolism / Power & Cooling
- Persistent modifiable state (if using programmability)

**Figure 2.** The `SelfImage API` datasheet cover. To propagate successfully, even the most complex and subtle ideas must also have small and memorable representations. If the idea creators fail to provide them, the idea consumers — if there are any — must and will. Here, as an example, the **`SelfImage`** API begins with four simple words and a single shape.

they can communicate with me via the shared model. They can draw on it, or edit it, or make their own version. It's a shared API we have between us.

To me, the **`SelfImage`** API feels like a similar kind of visualization. It's not necessarily an accurate representation of what's going on inside my machine, but it's a helpful abstraction that allows me to think through how my code is executing, and how it could be improved.

**DAVE:** Yes, absolutely. The diagram is still much simpler than the code and the machine it's depicting, but it has value in the moment. All we really need is to be confident that the diagram is implementable.

When we derive a diagram from running code, we know the diagram is implementable, because "`here's an implementation`". But if we add another arrow, say, the diagram may no longer be implementable in the existing code. And that tension, between simplified abstractions and actual implementations, is what code development is all about.

If there's a small set of abstract but widely implementable processes with a lot of descriptive power, we should give them a name to go by. That's all the **`SelfImage`** API is.

## 6 Developing natural code

**LU:** Okay, imagine I've bought into the "`natural code`" idea, and now I want to put it into practice — I want to start developing "`natural code`". I want to improve the shared codebase! Well, that feels really hard to do, because the concept is so unsatisfyingly vague. How do I actually develop "`natural code`"? Can you spell it out for me?

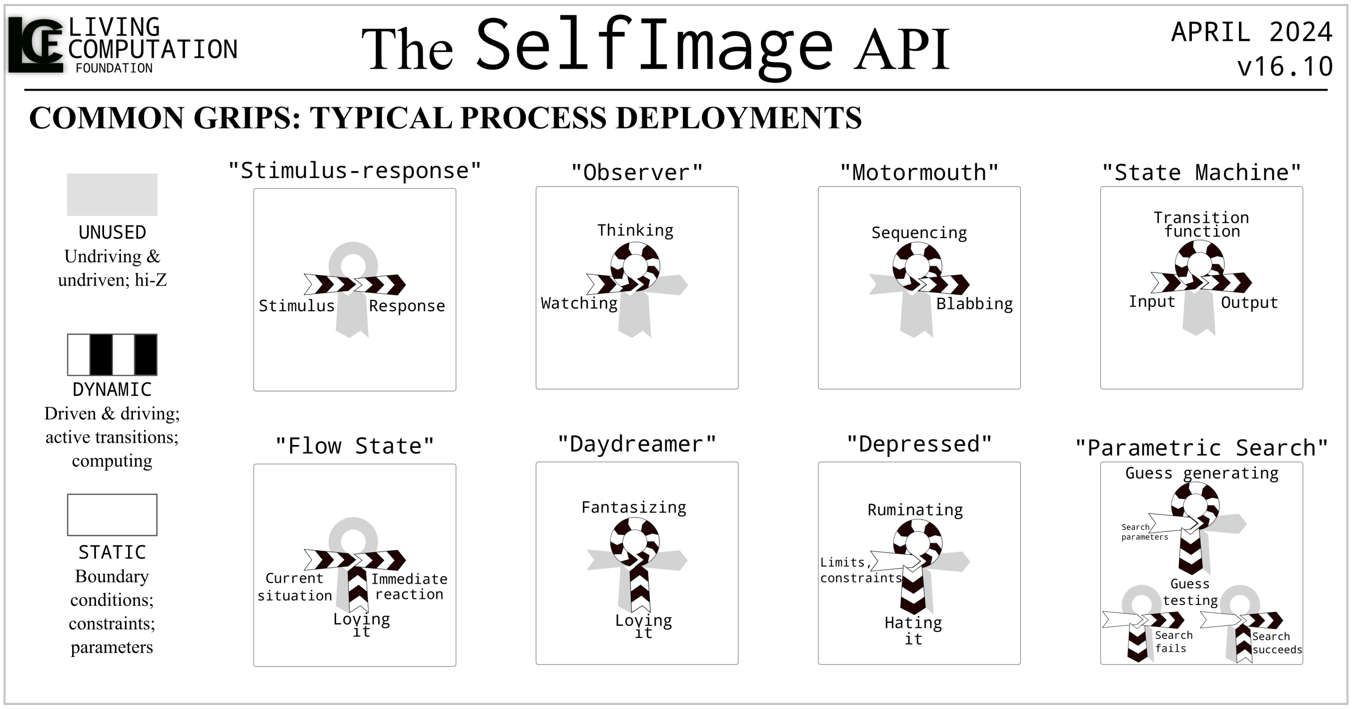
**DAVE:** I've been accused of being too vague before, and to some degree I will plead guilty to that. But also, that's just the nature of APIs. The whole idea is that they're abstract. I mean, like a linked list is utterly vague about what's inside it. It's utterly vague about exactly how many items you're going to need in the list, and so on. That's by design. That's the point. It's compatible with a wide range of uses, and the **`SelfImage`** API is the same.

**LU:** Right, I see. And I saw in your video [4] how you're using the **`SelfImage`** API as a model for some example computations, like "The Daydreamer" (see Fig. 3). But, in all honesty, it feels like you could put anything in there.

**DAVE:** I *hope* that you could model anything — at least, any implementable machine — with the **`SelfImage`** API, because it's deliberately trying to be as general as possible. Like, if either of us think some example is *not* implementable, then we should focus on that until we reach some shared notion of an implementation strategy. Or maybe we discover there's some deeper bug with the API, and we need to back up.

**LU:** Okay so perhaps the vagueness of natural code is actually a *feature*?

**DAVE:** Yeah it's `Vagueness As A Service`.



LIVING COMPUTATION FOUNDATION

The SelfImage API

APRIL 2024
v16.10

COMMON GRIPS: TYPICAL PROCESS DEPLOYMENTS

**Figure 3.** Sample applications page from the `SelfImage API` datasheet. Though informal, rough, and categorical, such simple visual representations of **`SelfImage`** configurations —"`grips`" — may offer insights. For example, highlighting the similarities between "`Parametric Search`" and "`Depressed`" might possibly be useful to an organism stuck in the latter grip.

### 6.1 Traditional programming

**LU:** And what about this? One reviewer felt that "`natural code`" doesn't help with traditional programming — so it's maybe off-topic for *Onward! Essays*.

**DAVE:** It's true we didn't stress implications for traditional programming, but I think there are some basic connections.

**LU:** And what are they?

**DAVE:** One way natural code informs traditional programming is by shouting "`Snap out of it! It's time to get over hardware determinism!`" And abandoning hardware determinism drives a focus on `robust-first programming` [6].

**LU:** Yes. I guess, with the MFM architecture [7], and T2 Tile Project [3], you've made a case for a new, non-deterministic kind of computer architecture. But that involves switching to a whole new hardware stack. Does robust-first speak at all to people programming on traditional hardware?

**DAVE:** Well, yeah, if the computing model is big CPU and big flat RAM and hardware determinism, serious robustness is scarcely an option. But still, natural code can at least offer support for some programming concepts over others.

**LU:** Like what?

**DAVE:** Well, here's three:

1. *Event-driven programming*: Prefer dialogue over monologue — shorter code sequences interacting.
2. *Self-stabilizing code*: First be robust, then as correct as possible, then as efficient as necessary.
3. *Minimize state*: Prefer recomputing over caching where possible; let the world be its own representation.

And maybe overall, natural code says be wary of people advocating correctness and efficiency only. I think traditional programming needs to hear that!

### 6.2 Debugging natural code

**LU:** I'm thinking back to when I said that "`programming other people`" seems cold and —

**DAVE:** And how do you feel now?

**LU:** Well, I still think it seems cold. And I can see that "`coldness`" blocking some people.

But I see you're not saying it for a cold-hearted reason. Instead, it's a way of thinking deeply about our communications, that will allow us to try to figure out how to become more compatible with each other, right?

My natural code is going out and yours is coming back. And maybe we're not hearing each other. Maybe we're not on the same page. Maybe we're struggling on the same thing. Maybe we're both trying to improve the world in the same way, but we're not able to work together. We're not able to *understand* each other in some way.

And you have this idea of "`Right, let's look at this in natural code terms`". "`Let's try to look at where our` `code is incompatible.`” “`Let's try to find a shared code that we both understand.`” “`Let's try to transpile the code between us.`”

**DAVE:** In the secret fortress of solitude in our heads, we are all trying to get what we want, but there's this huge veil of silence over that fact. We don't quite admit it, because it doesn't sound good. It sounds selfish, and so people ask, “`Do you do good because you're actually trying to do good or just because you're selfishly trying to make people give you the results of being good?`” Well, so that is an example of something that can be cleared up by taking this point of view of code transmissions.

We are coders. We're all trying to get what we want. And because we're alive, what we want tends to be stuff we think will help us persist and survive in the world. And cookies are a proxy for survival because we need energy to persist and sweets are a proxy for energy. So we think we're helping ourselves persist, and it's “`yes, yes, cookie, yes`” from the hardware. Then we end up looking like me.

**LU:** And I think that most of us, as adults, we pick that up implicitly, right? We learn that we can influence other people by deploying code, verbally or otherwise. Like saying “`Hey, duck!`” to someone and they duck.

But some of these children I worked with — for one reason or another, they struggled to pick this lesson up implicitly, so they had to explicitly learn it. And they often ended up understanding it better than many of their peers, who did learn it implicitly. These children gained mastery over communication by debugging it when it wasn't serving their interests as well as it could have. Perhaps more people could benefit from this kind of explicit debugging of their communication — of their code transmission.

**DAVE:** Right! We can often see implementations most clearly when they break down. The children's code wasn't executing the way they wanted, and that's frustrating, so you worked together to debug that. You made super-accessible communication channels, so step by step the kids could start choosing to transmit code that makes their world better.

**DAVE:** Once we admit, or once we just decide, that language is code, then the `natural code` framework says it's all about acts of code transmission. Some transmission through space from `A` to `B` at time `C`: What code shipped? Did that transmission happen for a good reason? Would we rather widen that channel, or maybe block it? All such questions are fair discussion topics among “`natural coders`.”

The overall goal is to debug the great machine and improve its codebase. Close up, between us, the purpose is to find a win-win, so I understand what your language means in my terms and vice versa — so we can share code effectively and our collective distributed machine works better. And I think, if we choose to be resolutely explicit about that — that we are coders, we are developers, and we're trying to debug the machine — we might all be happier and more productive, and our world more robust and sustainable.

### 6.3 Buggy code

**DAVE:** But unfortunately there are also grifters, who deliberately and knowingly *ship buggy code*, where the transmitted narrative is a trick to cover theft, or corruption, or other evil.

**LU:** People sowing division, spreading misinformation —

**DAVE:** Even good people can ship bad code in moments of weakness. They know in their hearts that the code isn't *exactly* right, and that its bugs benefit the transmitter. In tiny ways at least, it's like nobody is completely without sin, so typically all remain silent. And the result is that good people's petty hypocrisies enable other's great crimes.

**LU:** Some bugs are bigger than others.

**LU:** One of the reviewers expressed concern that `natural code` can be misused.

**DAVE:** For sure. `Natural code` gets misused a *lot*.

**LU:** Yes, it's happening already, all around us, whether we explicitly acknowledge it as `natural code` or not, harmful `natural code` is being shipped and —

**DAVE:** And we'd be better off acknowledging that —

**LU:** Because then we can be more explicit about naming it as such, and calling it out, and then —

**DAVE:** And then we can start talking like developers, and get down to debugging our shared natural codebase.

## 7 Owning our natural codebase

**DAVE:** Okay another run at a summary: There are many many ways to describe things. On the one hand, they are not all equally good for all purposes, but on the other hand, there's no one language that's “`uniquely most true`” either. You talk differently to your grandma than to a colleague or friend, because different code receivers understand differently, and have different shared dictionaries between you.

So the claim has two parts. First: We have to *make choices* about how to describe and understand ourselves and the world. We cannot delegate those choices, even if we really want to — not to other people, not to the universe itself. And second: One choice should always be that *we are coders.*

It's about all our code transmissions, natural and artificial. Is it all a metaphor? Sure, if you need it to be, but I'll still claim it's a simple and powerful basis for understanding and improving our shared computation.

So `natural code` will be one of many ways of describing and building things. It won't erase art, or philosophy, or any of those things. But it will always be available in addition. “`Let's consider this in terms of natural code.`”

**LU:** Over the last months we have attempted to own the ideas of natural code — struggling towards shared understanding where previously there was none. My hope is that other people will see our example and become inspired to do the same, though we cannot know for sure if that will happen.

**DAVE:** Indeed. We can only do what we can, and it won't all be easy. I hope that, once they see themselves as *natural coders*, people of good faith everywhere will work for a better shared codebase. I do have hope.

**LU:** To me, natural code is about building bridges, and getting people to work together — to name and call out the bad code, while celebrating the shipping of better code.

To do this, we *may as well* talk in terms of natural code. We *may as well* talk about developing our APIs, and debugging our difficulties, and improving our codebase. And I do believe that more and more people will join us on this, and become more deliberate about being natural coders.

**DAVE:** And this is just a beginning.

**LU:** "`Step by step`"!

**DAVE:** Step by step.

## References

[1] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. 2000. Amorphous Computing. *Commun. ACM* 43, 5 (May 2000), 74–82. https://doi.org/10.1145/332833.332842

[2] Dave Ackley. 2004. *Machines*. Retrieved April 22, 2024 from https://livingcomputation.com/lc/d/ae/machine.html

[3] Dave Ackley. 2015. Living Computation: Robust-first programming in ULAM. Video submission (accepted) to the *Future Programming Workshop* at SPLASH 2015, Pittsburgh. Video retrieved July 2024 from https://www.youtube.com/watch?v=I4flQ8XdvJM.

[4] Dave Ackley. 2021. We Are Coders - HSA101.2: Hypersubspaces. Video. Retrieved April 22, 2024 from https://www.youtube.com/watch?v=ScYgBxLupAs

[5] Dave Ackley. 2024. *Thank you Dan*. Retrieved July 17, 2024 from https://livingcomputation.com/lc/morning/202404201125-dan-dennett.html Also appeared in the 16th *The Artificial Life Newsletter*, at https://alife-newsletter.github.io/Newsletter/edition_016.html.

[6] David H Ackley. 2013. Beyond efficiency. *Commun. ACM* 56, 10 (2013), 38–40.

[7] David H. Ackley, Daniel C. Cannon, and Lance R. Williams. 2013. A Movable Architecture for Robust Spatial Computing. *Comput. J.* 56, 12 (2013), 1450–1468. http://dx.doi.org/10.1093/comjnl/bxs129

[8] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press.

[9] Andrew S. Bondy and Lori A. Frost. 1994. The Picture Exchange Communication System. *Focus on Autistic Behavior* 9, 3 (1994), 1–19.

[10] Tristan Bove. 2021. Techno-Optimism: Why Money and Technology Won't Save Us. (June 2021). Retrieved July 10, 2024 from https://earth.org/techno-optimism/

[11] Noam Chomsky. 1956. Three models for the description of language. *IRE Trans. Inf. Theory* 2, 3 (1956), 113–124. http://dblp.uni-trier.de/db/journals/tit/tit2n.html#Chomsky56

[12] Daniel C. Dennett. 1987. *The Intentional Stance.* The MIT Press, Cambridge, MA.

[13] Christophe Dupre and Rafael Yuste. 2017. Non-overlapping Neural Networks in Hydra vulgaris. *Current Biology* 27, 8 (24 Apr 2017), 1085–1097. https://doi.org/10.1016/j.cub.2017.02.049

[14] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software* (1 ed.). Addison-Wesley Professional. http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1

[15] A. H. Maslow. 1943. A theory of human motivation. *Psychological Review* 50, 4 (1943), 370–396. https://doi.org/10.1037/h0054346

[16] Steve Muir. 2004. The Seven Deadly Sins of Distributed Systems. In *First Workshop on Real, Large Distributed Systems (WORLDS 04).* USENIX Association, San Francisco, CA. https://www.usenix.org/conference/worlds-04/seven-deadly-sins-distributed-systems

[17] Jakob Nielsen. 2024. Accessibility Has Failed: Try Generative UI = Individualized UX. (February 2024). Retrieved July 10, 2024 from https://jakobnielsenphd.substack.com/p/accessibility-generative-ui

[18] Markus Oberlehner. 2020. *Retry Failed API Requests with JavaScript.* Retrieved April 22, 2024 from https://markus.oberlehner.net/blog/retry-failed-api-requests-with-javascript/

[19] Arnon Rotem-Gal-Oz. 2008. Fallacies of Distributed Computing Explained. *Doctor Dobbs Journal* (01 2008).

[20] Carlos J Sanchez, Chen-Wei Chiu, Yan Zhou, Jorge M González, S Bradleigh Vinson, and Hong Liang. 2015. Locomotion control of hybrid cockroach robots. *J R Soc Interface* 12, 105 (April 2015).

[21] B. F. Skinner. 1938. *The behavior of organisms: an experimental analysis.* Appleton-Century, Oxford, England.

[22] R. N. Southworth. 1967. PL/I bulletin no. 5. *SIGPLAN Not.* 2, 12 (Dec 1967), 1–71. https://doi.org/10.1145/1139502.1139504

[23] tldraw Inc. 2022. *Virtual whiteboard.* Retrieved April 22, 2024 from https://tldraw.com/

[24] Franco Zambonelli and Marco Mamei. 2005. Spatial Computing: An Emerging Paradigm for Autonomic Computing and Communication. In *Autonomic Communication*, Michael Smirnov (Ed.). Lecture Notes in Computer Science, Vol. 3457. Springer Berlin / Heidelberg, 227–228. http://dx.doi.org/10.1007/11520184_4

[25] J. F. Ziegler and W. A. Lanford. 1979. Effect of Cosmic Rays on Computer Memories. *Science* 206, 4420 (1979), 776–788. https://doi.org/10.1126/science.206.4420.776

Revised 18 July 2024

July 19, 2024

*Onward Essays!* Program Committee
via Internet

To whom it may concern,

This letter documents the changes made to our *Onward Essays!* submission #4 — entitled *Dialogues on Natural Code* — since it was conditionally accepted on June 6, 2024.

We thank the reviewers, and especially our paper shepherd, for their extensive comments and assistance. The essay is much better for their contributions.

We have made extensive revisions, ranging from small wording changes to new sections and an overall paper reorganization. All these changes are visible in the attached diff, and we have outlined them all below.

### Reviewer A concerns

- **Clarity of experimental style**: Reviewer A stated that they enjoyed the experimental style of the essay, but found it difficult to orient themselves at times because of it. As suggested, we have renamed some headings to help signpost the reader better. We have also added an additional top-level section, "The SelfImage API", to help create a more coherent flow. And we have added box formatting to a key early sentence, to emphasise the main goal of the essay.

### Reviewer B concerns

- **Purposeful dialogue form**: Reviewer B suggested that the "dialogue form" of the essay was not always effective. It was best when it served a purpose in expressing the essay's points, but could be distracting when it did not. Therefore, we removed some moments of meta-dialogue that did not add to the essay's message.
- **Jarring shift of dialogue style**: Reviewer B noted that there were some jarring shifts in the essay's dialogue style. It suddenly changed from quick-fire back-and-forth to longer monological paragraphs, which broke the flow of reading. We have now smoothed out these transitions with small interruptions in some of the longer monologues.
- **Undefined terms**: Reviewer B noted that some terms, such as "pattern", were used before being defined, due to the non-deterministic nature of the essay. In this case, we reworded the sentence in question to give the reader more context about what we mean by the term.
- **Prior art section**: Reviewer B stated that the "Prior art" section did not

feel appropriately named, as it did not include related work — it included other content instead, such as the "SelfImage API". As suggested, we have now pulled out the "SelfImage API" parts of the essay into their own top-level section, and we have added examples of related work within the "Prior art" section.

- **FOMO**: Reviewer B noted that the term "FOMO" might not be familiar to all readers. We have now added a brief explanation of the term within the dialogue.
- **SelfImage API detail**: Reviewer B commented that we should provide more detail about the "SelfImage API" as it seems to be an important part of the essay to understand. As mentioned already, it now has its own top-level section, and more focus has been placed onto it.
  We have also moved the SelfImage API section to a later position in the essay, based on followup feedback from our shepherd. The purpose of this move was to let the reader become more familiar with the concept of "natural code" before introducing them to the SelfImage API. We hope that this will increase the chance of the reader understanding the model. This new section also includes the "Shared code" subsection in order to provide more immediate context around the purposes of the API.
  And we have added more detail to the captions of the SelfImage's figures, giving more context on terms like "grip", which Reviewer B found unclear.
- **Delivering promises**: Reviewer B commented that it felt like the essay did not deliver what had been promised by the "Developing natural code" section heading. We have now padded out this section with further examples and detail, such as the new "Traditional programming" and "Buggy code" subsections, to try to better deliver on this promise.
- **Backfilling conclusion**: Reviewer B commented that the conclusion section did well at grounding the essay, and we should consider bringing some of this into earlier parts. We did this through a new "Buggy code" subsection, and we also brought some of the conclusion's themes to the very start of the essay, within "Coldness and evil".
- **Ackley references**: Reviewer B shared that they were able to understand the essay better after exploring some of Dave Ackley's previous work. We added more references to Dave's work to address this.

### Reviewer C concerns

- **Technocratic undertones**: Reviewer C shared their concerns about the essay's "ideological, technocratic undertones". To address this, we added a substantial new subsection that discusses this concern directly — at an early point in the essay. We also brought forward a dialogue around "coldness" into this early section, to try to establish a more humane tone from the beginning.
- **Helping understanding**: Reviewer C noted that it was unclear how the

model helps to understand anything. To address this, we have now added further examples of domains in which a natural code approach can be helpful, such as accessibility and climate. We also added a new subsection about how natural cocde can impact our understanding of "traditional programming". And we have tried to clarify the role of the SelfImage API by giving it its own dedicated section, as mentioned previously.

- **Misuse**: Reviewer C noted that the essay only acknowledges the potential misuse of natural code in the very final section. We have now brought this dialogue forward to an earlier section, and we have given it some more room and discussion.
- **Relevance to programming**: Reviewer C suggested that the essay might not relate closely enough to "programming" to be suitable for *Onward! Essays.* To address this, we added a new subsection highlighting ways in which natural code can inform traditional programming practice. We also added a subsection on related work to help ground the essay in works that are well-known to the programming world.
- **Machine example**: Reviewer C noted that one of our examples of a "machine" seems to contradict our previously stated definition. We have expanded on that example to clarify our thinking around it, while also acknowledging the example's flaws. We have offered an alternative example within the same dialogue — one that more easily fits our definition.
- **Monospace font size**: Reviewer C commented that the monospace font of the quoted fragments was too large, and it wasn't clear what they were being used for within the visual language of the essay. We have now reduced the size of that font, and we have made use of it more consistently throughout the dialogues.

Best regards,

Lu Wilson & Dave Ackley

# Dialogues on Natural Code

Lu Wilson
TodePond
London, UK
todepond@gmail.com

David H. Ackley
Living Computation Foundation
Placitas, New Mexico, USA
ackley@livingcomputation.org

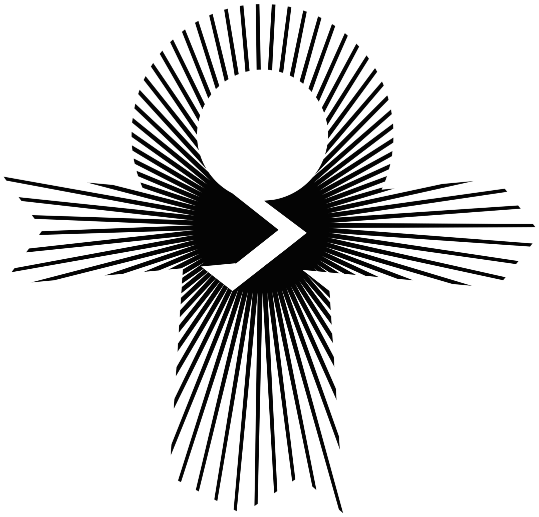


**Figure 1.** The **SelfImage** starburst.

## Abstract

This essay~~is a loosely edited collage ~~, based on a series of discussions between the authors~~that took place in early 2024, as we worked ~~, is a loosely edited collage in which we work to flesh out our shared interests in non-traditional machines and coding mechanisms. We primarily focused on the idea that all human language can usefully be viewed in programming language terms — as "natural code". Programming languages and natural languages differ in many ways, such as having relatively formal definitions versus not, emphasizing strong syntax versus large dictionaries, and demanding rigid implementations versus building on the vagaries of living systems. Still, we saw deep unities as well, much more than mere metaphor, and we glimpsed the possibility of applying humanity's decades of programming language design and software engineering experience to the task of debugging and refactoring the natural codebase that we all share. These fragmentary and overlapping dialogues represent both a description and an example of natural code, and we offer them here, with a simple "natural API" illustration, in hopes of *programming* people to join in natural code development.

***CCS Concepts:*** • **Software and its engineering** → *Very high level languages*; • **Computing methodologies** → *Distributed computing methodologies.*

***Keywords:*** Natural Code, Human Computation, Robust API Design, Implementability

**ACM Reference Format:**
Lu Wilson and David H. Ackley. 2024. Dialogues on Natural Code. In *Proceedings of XXXXXXX (SPLASH '24).* ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Being machinery

**DAVE:** I think living organisms can be meaningfully viewed as machines.

**LU:** Sorry, what?

**DAVE:** They're physical arrangements of matter that move and do work. They have power supplies. Living systems are machines.

**LU:** Including us?

**DAVE:** Including us. We're machines.

*SPLASH '24, October 20–25, 2024, Pasadena, California, United States*

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

**LU:** Really? I don't feel like a machine.

**DAVE:** I mean, people usually think of machinery as metal and screws and batteries, and I have *very few* of those in my actual living body.

**LU:** A non-zero amount?

**DAVE:** I want to take machines way beyond metal and screws, and say: Any time matter is arranged in space, and an energy supply is incorporated so that the arrangement of matter and energy can *do something* — that's what we're talking about as a machine. And that description is as true for screws and metal as it is for people and amoebas.

**LU:** I don't know if I *want* to think of myself as a machine though.

**DAVE:** It can be uncomfortable, but when we go to the doctor, say, we *want* them to be talking about us in mechanistic ways, like "~~the heart machine is not working as well as it could~~`the heart machine is not working as well as it could`" or whatever. This framing of a living system as a machine can be useful when we're trying to understand how it works, and how to make it work better.

## 1.1 Building machinery

**LU:** As well as *being* machinery, living things are also capable of *building* machinery. That's what you're saying, right?

**DAVE:** That's right. Machines that somehow work to preserve their ~~patterns~~ structures, their *patterns*, are what we call "~~life~~`life`". Persistence involves maintenance and repair, but also building copies.

**LU:** I guess so! Though I was thinking more about traditional ideas of "~~building machinery~~`building machinery`", like a beaver building a dam, or a wasp building a nest.

**DAVE:** That happens too. And humans build bridges, rockets, and programmable computers. I think about "~~building machinery~~`building machinery`" writ large. It can be something like lighting a fire, or folding a paper airplane, or moving a rock off a path.

**LU:** You're using the phrase "~~building machinery" very~~`building machinery`" extremely loosely here, right? Because to me, "~~building machinery~~`building machinery`" sounds like ~~"creating something"~~*creating* something, or *making* an artifact of some sort. But you're using it to refer to what seems like just an action, or a process.

"~~Lighting a fire~~`Lighting a fire`" doesn't sound like building anything at all. It just sounds like ~~enacting a change~~*enacting a change*.

**DAVE:** Yeah I screwed that up. Collecting wood and stuff is building the machine. Lighting the fire is flipping its switch.

But, say you're working at a hamburger joint, where all you have to do is slap a burger on a bun and put on ketchup or mayo, and it's done. You're "~~building a machine~~`building a machine`" out of other complex arrangements of matter.

**LU:** You're changing the arrangement of the burger's ingredients, and that's what you're calling "~~building machinery~~`building machinery`". It's not that you've "~~created~~`created`" these ingredients, but you've built them into a particular pattern.

**DAVE:** Yes, you're arranging matter to get certain properties.

**LU:** Okay ~~that makes sense to me. You'~~so you said that a burger is a machine and —

**DAVE:** The reviewers had some troubles with that.

**LU:** And I can understand their troubles. You said that a machine can *do something*, but a burger just sits there.

**DAVE:** I — Fair enough. I understand. I mean, there are many power sources for machines. You could have a battery, or gasoline, or gunpowder. But you could also have a human.

**LU:** A human?

**DAVE:** Like, an old-fashioned well pump is a hand-powered machine. You pump the handle, and water comes up out of the spout and helps you live. It's a human-powered machine.

And maybe a hamburger isn't the cleanest example —

**LU:** It really isn't the cleanest example.

**DAVE:** — but the hamburger machine runs on muscle power too. You pick it up and chomp it on down, and it absolutely *does something*: It feeds you and helps you live.

**LU:** You're stretching the use of the language quite a bit, but what you're saying is — when you're building machinery, you're building a pattern.

I could have some LEGO bricks on my table, and they're all scattered around. I could build something new just by moving them around. I could build a pattern, or a house. Either way, I'm building machinery just by rearranging. Is that how you see it?

**DAVE:** Right. Arranging matter. A house is a pattern too.

## 1.2 ~~How to build~~ Contracting machinery

**LU:** And you're saying there are two ways of building machinery? One way is to do it yourself, to build it *directly*.

**DAVE:** Wood, hammer, nail. Yeah.

**LU:** And the other way is by getting *another machine* to do the work for you. You can instruct it to do the building on your behalf. In this case, you're building *indirectly*.

**DAVE:** Yes, you find a programmable machine that's out there in the world already. You don't have to build it yourself. You ship some code, and have that machine do the work for you. When you don't have to send the wood or the tools, code is incredibly cheap to ship. That's its superpower.

**LU:** And that programmable machine could be anything we can transmit code to, like a mechanical arm in a factory, or a rocket, or a computer.

**DAVE:** Or we flip the switch on the wall. We want light.

**LU:** Okay, I see where this is going.

### 1.3 Human hardware

**LU:** You're saying that "~~the programmable machine could be a person~~the programmable machine could be a person".

**DAVE:** Right. As humans, we can transmit code to another person and get them to do something for us. We can say, "~~Hey, can you help me build this shelter?~~Hey, can you help me build this shelter?" or "~~Can you build a fire while I gather food?~~Can you build a fire while I gather food?".

**LU:** I'd argue that animals do that too, right? Living things often communicate with each other in some sort of way.

**DAVE:** It's certainly a spectrum. Maybe an animal sends a signal that means "~~run~~run" or "~~danger~~danger" or "~~food".~~food".

**LU:** Either way, you're saying that we can code one another. Asking someone to do something is coding them, in a way?

**DAVE:** Yes, we transmit "~~natural code~~natural code" all the time — when we talk with each other, or teach stuff to our kids.

~~I think we should use our knowledge of programming languages, of software and computing, to examine our own natural code. To understand it and debug it. To make society better, and to improve our shared codebase.~~ If I was trying to wrap it all up in a box, I'd say

> **I think we should use our knowledge of programming languages, of software and computing, to examine our own natural code. To understand it and debug it. To make society better, and to improve our shared codebase.**

This is why I want to push for a view of computation broad enough that we can see humans as *programmable* machines — that are programmed by ~~natural code~~"natural code".

### 1.4 Coldness and evil

**LU:** ~~I mean~~ This idea that people "program" other people. To me it seems — ~~before we get to that~~

**DAVE:** It seems really obvious, right? It helps us to — ~~one~~

**LU:** No. Actually, I was going to say that it seems really cold.

**DAVE:** Oh. Well.

**LU:** It almost seems psychopathic, because it sounds like it's all about trying to manipulate other people.

**DAVE:** Well, I —

**LU:** But communication isn't only for influencing people. We also talk to share our feelings, and connect with others. Or we just want to be heard, or rant, or share a joke.

**DAVE:** Right! And I think that's a good —

**LU:** So we can't boil down communication to just "getting someone to do stuff" because that's cold, and it's not true!

**LU:** Reviewer C is worried about "the ideological, technocratic undertones" of the essay, and "it's a pervasive fallacy in the tech world to see all our problems as technological" and "Every human interaction is reduced to a kind of programming".

**DAVE:** Yeah. And how do you react to that?

**LU:** I was genuinely worried about this when we submitted, because it's something I agree with. There *is* this pervasive fallacy to see all our problems as technological. I hate it, and I see it time and time again.

Like recently, I've been hearing more and more people around me saying that "all we need is better technology" and all our computer accessibility issues will disappear.

**DAVE:** I just can't imagine somebody saying that seriously.

**LU:** For example, I read a recent essay [?] saying that "AI will soon come to the rescue" for accessibility.

Or take the climate crisis. There's this fallacy that we don't need to worry about reducing our energy usage, or replacing our energy sources [?] because —

**DAVE:** "We will technology our way out of it". Carbon capture, seeding the clouds, or whatever we can tell ourselves to delay dealing with the real problems.

**LU:** Exactly. In these cases, the actual solution is to *not* see the problem as mostly technological. Instead, the solution is to try to change our behavior, both as individuals and as a society. I think this is where natural code can help. It can give us a new perspective and understanding of our communications and how to improve them.

**DAVE:** One answer to such criticisms is that we are reading the concept of "technology" broadly enough to include stuff that's not traditional technology. People can hear us say "technology" and think it means traditional programming languages and computers and "tradtech" generally.

**LU:** Right, we say "natural code can help us" but sometimes people hear "traditional technology can help us".

**DAVE:** But really we're saying "technology writ large is much bigger than tradtech" and part of that is understanding ourselves better — that we can be viewed meaningfully as machines, and our communications can be viewed as code, and we build more machines to help keep ourselves alive.

**LU:** And we exchange code with each other.

**DAVE:** For sure. We are coders. We ship code.

**LU:** I mean, it's a tricky idea to sell. And it does sound quite "technological".

**DAVE:** And I think we just have to own that. But we also have to stress that judgment goes beyond just the tech. Shipping code "to make money" is different than "to help society", no matter how tech hypocrites may try to conflate them.

**LU:** If anything, I think we are calling for *fewer* problems to be seen as solvable by tradtech. For example, at work, we wanted to make it easier to hear each other on our video calls. We got new tradtech — software, microphones — but still had problems.

**DAVE:** And the real solution was like "talk slower"?

**LU:** It's mainly "avoid cross-talk" and "be sure to set up everything properly". In that situation, deploying "natural code" is what improved things. We actually wrote up a document — guidelines for behaving in meetings. And for me this is a form of "natural code".

**DAVE:** Like how modern programming projects often have an explicit "Code of Conduct." That's "natural code"!

## 2 Beyond determinism

**LU:** Another obvious objection to ~~this~~ these ideas is that humans seem really different to computer hardware, because computers are absolutely rigid and repeatable. They're *deterministic*, and humans are not.

**DAVE:** Deterministic execution of code has always been an illusion. There's always the possibility of cosmic rays coming in and flipping a bit, say, and that does happen sometimes [? ]. But we know that we can engineer traditional computer hardware so that the chance of that is small enough that we can usually ignore it.

**LU:** But someone could still come and turn off your computer's power, right?

**DAVE:** Right, or overheat it.

**LU:** Or smash it with a hammer.

**LU:** In web development, when you do a "~~fetch~~fetch" request to an endpoint, you usually use your own special kind of "~~fetch~~fetch" function that automatically retries a few times [? ].

**DAVE:** Right, because in the network world —

**LU:** In the network world, things can go wrong, and in fact, they often do go wrong [? ? ]. So you run the same code again and again, to increase the chances that it will work.

**DAVE:** People certainly don't do the same thing every time.

**LU:** So when we transmit code to a person, we can't know for sure what the effects will be. They might ignore us, or say no, or do something completely different.

The essay might make no sense to them, or they might get it but disagree. But even if the chance of convincing them is low, we might still think that it's worth a try.

**DAVE:** Yeah, maybe we'll succeed. Maybe we won't. The machines executing the code of this essay are going to be way non-deterministic.

**DAVE:** I've been trying to get ~~this natural code message~~ ideas like natural code across for a long time [? ], and it's been hard. People bring all of their traditional computing misconceptions to it. And the idea of ~~natural code~~ natural code just looks crazy to them.

**LU:** Has non-determinism been a blocker for some people?

**DAVE:** Some people would outright say "~~without deterministic execution, it's not computation.~~without deterministic execution, it's not computation."

**DAVE:** There's this idea that "~~if you can't predict exactly what the code will do, it'll be chaos~~if you can't predict exactly what the code will do, it'll be chaos". My claim is no, ~~no, not really.~~

~~My claim is that~~ we can still talk in terms of computation and code, even if the "~~computer~~computer" is not fully deterministic.

Even if we only have a 51% chance that some code will work versus a 49% chance that it won't, say, we might still want to run the code, again and again, for that 2% edge.

**LU:** I've been thinking about how we can get across this "~~non-determinism idea~~non-determinism idea", and I wonder if we can use the format of the essay itself to help us dripfeed it throughout.

**DAVE:** Oh I see, bits of conversation out of order, and so on.

**LU:** Yes, we don't need to be strictly chronological. We can jump around and revisit things. When we transmit natural code, we don't know exactly how that code will be executed. We don't know what the exact order of execution will be either, but we can still talk about it in terms of code and computation. It's still possible to do that.

**DAVE:** Perhaps also showing how we can bend the familiar overall "~~syntax~~syntax" of a paper, but still transmit legible code.

**LU:** Someone could skip ahead to the end of the essay, or miss out a whole section, or just look at the diagrams.

## 3 Prior “art”

**LU:** But, Dave: Why put this essay forward as a submission to a programming language conference? Why not go to a philosophy conference, or ~~art~~“art”? Why enter through programming languages as a lens?

**DAVE:** I mean ~~ideally, if there was a lot~~ sure, if we had more time and ~~a lot~~ more collaborators, we~~would~~’d go to all those conferences — a full court press — and then ~~the~~ FOMO would descend, and ~~then~~ the world would change.

**LU:** “FOMO” as in “Fear Of Missing Out”?

**DAVE:** Yes, if we could figure out how to —

**LU:** If we could market this “natural code” idea in all those conferences, lots of people might get “FOMO” and get involved.

**DAVE:** And that would be great. But we can only do what we can figure out how to do — can only do what~~is “implementable~~’s “implementable” for us at the time.

I do want to poke the bear a bit, and it seems appropriate for a venue like *Onward! Essays* that’s explicitly aimed at computation and programming languages writ large.

**LU:** Yeah, I see that. I think it’s helpful for you to share why you’re coming through programming languages, because people reading this might think there’s a particular reason behind that. But it sounds like it’s partly just because that’s where ~~where~~ you’re starting from.

**DAVE:** Right that’s my history. Code’s what I know best.

### 3.1 Historical traditions

**DAVE:** It’s like philosophy, psychology, and all those things, are trying to describe *what we are* — what our touchstones and key concepts are, how we see what we see, and so on. I think, despite their great successes, such fields have deep assumptions that limit how clear and effective they can be.

I think we should start again with notions of programming languages and software engineering, but move beyond deterministic execution. Then we can start talking about our human collective computation in terms of APIs, programming languages and structures, compositionality and modularity, and so on.

The goal is: Whenever we speak, we can always know, or plausibly believe, that what we are saying is *implementable*. We could always, at least in principle, build a machine — using ordinary silicon chips or exotic biological bricks or whatever — that could *run the code* we’re shipping. Then we point at the machine and say “~~I mean like that!~~I mean like that!” And that’s what we cannot do with philosophy or psychology or religion or anything, that we maybe could do if we say “~~Let’s pretend natural language is code~~Let’s pretend natural language is code”.

### 3.2 Implementability

**LU:** I would challenge the idea that ~~natural code~~ natural code is the only route to implementability. I think that neuroscience, say, or even physics, offers implementability in some way.

I know there are studies out there where they’ve taken an organism, a *hydra vulgaris*, and they’ve mapped out its entire neural networks, and they’ve used that to get closer to determining how the creature is implemented [? ].

**DAVE:** I certainly do not want to say that natural code is the *only* route to implementability. I would argue that it looks like the most *direct* route to implementability.

Driving around a cockroach by putting wires into its spine [? ] is clearly building a piece of living machinery, working at a pretty low level. But in the computation world, instead of writing assembly code, we glue together giant stacks of software and plug one abstracted part into another.

I would argue that, if neuroscientists build more machines out of more neurons, displaying more complex behaviors, they’ll stop talking about that overall machine in terms of neurons. They’re going to start talking about it in terms of inputs and outputs, and parallel and sequential processing — in terms of computation and code.

**LU:** So you think that it all comes back to computation in the end?

**DAVE:** Back to ~~implementation~~*implementation*. I find neuroscience and biology results inspirational for seeing how nature does things. Many perspectives help! I argue that ~~natural code~~ natural code is yet another point of view that can be a useful framing for understanding our world, and making it better.

~~The SelfImage API datasheet cover.~~

### 3.3 Related work

**LU:** Okay, okay. But I don’t think that this “Prior Art” section actually covers any prior art so far. It feels like a rejection of everything existing. Natural code can’t be *that* new, right?

**DAVE:** Of course, lots of things are connected. Dan Dennett’s ideas had a big impact on me personally, for one.

**LU:** I saw you tooted a little remembrance about him. [? ]

**DAVE:** Yeah, he was so clear. With his notions of descriptive “stances” [? ], I see natural code as a way of connecting the intentional stance with the physical and design stances.

**LU:** I’m reminded of Alexander’s pattern language stuff too [? ] . His “patterns” are like code, describing how to solve various problems through architecture and design. And there’s an emphasis on the patterns being “tentative” and unpredictable. There is a non-deterministic aspect to it.

**DAVE:** Right, and of course design patterns [? ] have similar flavors. Language not quite executable on a computer, but very “code like” and absolutely executable on *developers*.

**LU:** For me, these examples demonstrate that we can spot aspects of `natural code` within existing works, perhaps implicitly, and what we're trying to do is—

**DAVE:** We're trying to *explicitly* frame things as code.

### 3.4 Blending fields

**LU:** Personally, I seek out the projects that aim to blend numerous fields~~together~~, like those that combine science and art in some way, or those that try to bring together different categories of research. It's not always easy to do, but I think it's often where the most impactful work can be done — you get to pick and choose the strengths of various fields, and get the "~~best of both worlds~~`best of both worlds`" in many cases.

**DAVE:** Let me be completely honest. My problem combining art with science is that the results often feel a bit like the ~~worst of both~~*worst of both*. You know, not great science, not great art, no impact at all. And so I feel that art is too —

**LU:** You make a few art pieces though.

**DAVE:** Well —

**LU:** Yeah, it's funny hearing you criticize using art, because from my perspective, you seem to do a lot of art.

**DAVE:** What? What!?

**LU:** Yes, I mean, I would —

**DAVE:** Name one!

**LU:** The ~~**SelfImage**~~**`SelfImage`**. That's art! (See Fig. 2.)

**DAVE:** Okay, I see that as computation, I guess.

**LU:** This is how I see it. I think you're in this world of trying to get different fields to put their heads together, and learn from each other.

**DAVE:** Yeah.

**LU:** And maybe you see a divide between the "~~art world~~`art world`" and the "~~non-art world~~`non-art world`." But for me, it isn't helpful to draw these lines when trying to bring the different fields together.

I accept that you don't need to *open* with art. You can open with something else and then sucker-punch with art, right?

**DAVE:** Yes, yes, yes, it's like "~~just kidding, it was all a dream~~`just kidding, it was all a dream`".

**LU:** "~~It was art the whole time~~`It was art the whole time`".

**DAVE:** For the ~~**SelfImage**~~**`SelfImage`** in that sense, you are 100% right. There *is* an art component to it, and a marketing component — an attempt to be viral, which I have completely failed at.

**LU:** Except —

**DAVE:** Well I mean, everybody wants the next zero on their views, on their citations, on their patreon, whatever it happens to be. But I'm still only down at the sort of two to three zeroes range, so, you know, I can legitimately claim lack of virality, and — well, anyway, that's another topic.

**LU:** Yeah okay, I just think it's good I got you to admit that the ~~**SelfImage**~~**`SelfImage`** is art.

~~Okay I made up a couple "datasheet pages" to present some / API stuff.~~

~~Oh good! But, they didn't exist for most of these discussions. How will we incorporate them into the essay?~~

~~Maybe we can just sneak in some "(see Fig. 2)"s like we're doing with citations?~~

### 3.5 ~~The / API~~

## 4 The nature of natural code

~~So, Dave: What *is* the / API? I know from your video [? ] that it has four processes, but what does it mean?~~

**DAVE:** ~~Fields like philosophy and religion and science offer us *language* to talk about what kind of machines we all are. Like, "I think therefore I am", or "I am a collection of neurons".~~

~~Or "We are made up of needs and wants and motivations", or whatever.~~

~~Right. All of these languages contain some germ of truth, but none of them are going to be wholly sufficient to answer all of the variety of questions that we might want to ask. So what we need to do is choose multiple approaches — multiple languages. I think of them as "APIs". They're clearly not perfect, and don't cover everything, but they emphasize certain parts, and make it easier to express some concepts versus others.~~

~~So the / (see Fig. 2) is such an API. It depicts us as arrangements of four computational processes:~~

1. ~~**Input**`Input`: Handling influences from our surroundings,~~
2. ~~**Output**`Output`: Performing work on our surroundings,~~
3. ~~**Sequence**`Sequence`: Changing internal states over time, and~~
4. ~~**Judge**`Judge`: Assessing situational desirability.~~

~~If we're interested in how we understand the world around us, we'll focus on the **input**`input` process. If we want a deeper understanding of how we actually create and do things in the world, we'll unpack the **output**`output` process, and so on.~~

~~The / is a really basic framework to see ourselves through a computational lens. It's a starting point.~~

### 4.1 ~~API design~~

~~To me, the / API seems no different than a psychological model that aims to describe how people behave. It reminds me of something like Maslow's hierarchy of needs [? ], or operant conditioning [? ], even.~~

~~Ah, okay. What I'm suggesting is that, by taking the computational metaphor, the / API can simultaneously describe both people *and* other programmable machinery. That's one difference.~~

~~And secondly, I'm claiming that the / API leads more directly to implementability than a psychological description, because it uses the language of computation.~~

~~So it's not solely a *descriptive* model?~~

~~Right. It can be a blueprint. It can be a recipe for how to build machinery.~~

~~Okay, it seems more like a *design* challenge — you want to make an API that's useful, regardless of how truthful it is as a description.~~

~~A scientific theory succeeds when it gives us an *unexpected truth*. But that's not the goal of an API in software design. We want an API to be as unsurprising as possible. We want to adhere to the principle of least astonishment.~~

~~Ideally, an API should not teach us anything new. The goal of an API is to be *obvious*, and that's what we can judge it on — how universally obvious it is.~~

~~I think I get it. It's more like user experience design, in a way. It's a communication tool that lets us talk about the world in a certain way — under a computational lens.~~

~~It should be as easy and straightforward to use as possible.~~

The canonical Chomsky hierarchy stuff [? ] is all about languages having compositional, recursive, syntactic structures, allowing language users to create open-ended complexity. And I think that's great, but it doesn't go nearly far enough. ~~Syntactic language properties, on~~ On their own, syntactic properties are almost a detail. There~~ are~~'s other ways to get modularity, ~~and~~ complex representations, and so on. For example, you could just list chosen words in a random order — "`wood, hammer, nail`" — and it could create a notion in the listener's head ~~— "`Wood, hammer, nail`" — and~~ that could be quite rich, with hardly any syntax.

**LU:** Splinters.

**DAVE:** Right. Sore thumb. So I'm hesitant to embrace the idea that it's all about *language* and which structural properties of language are important. I think that's wrong. Instead, I want to talk about "~~`code`~~`code`", and not "~~`programming language`~~`programming language`". And by saying ~~`code`~~`code`, I want to rope in signals, gestures, grunts — stuff that seems below the level of programming languages.

### 4.1 Starting from signals

**LU:** Okay, "~~`code" "code" "code`~~`code`" "`code`" "`code`". Not just language. I think that's right. You can get too focused on the structure and syntax of language. I think it's more important to think about the *purpose* of language — the purpose of code, I mean.

When I was a teacher, I worked with very young children who struggled to communicate with other people, for various reasons. It wasn't that these children necessarily struggled with *language*. In fact, some of them were hugely competent with language and its syntax. They struggled with *communication* in a more general sense, which can sometimes involve no syntax or language at all. It can mean "~~`prodding someone`~~`prodding someone`", "~~`looking at someone`~~`looking at someone`", or simply "~~`tugging on their hand`~~`tugging on their hand`" to pull them along.

The first step that we always tried to get across to these young children was, "~~`look at all the good things you can get from interacting with someone`~~`look at all the good things you can get from interacting with someone`", and we used a lot of *biscuits*.

Most children love biscuits, right?

**DAVE:** Cookies.

**LU:** And if you can tell them, "~~`look, you can prod me, point at a biscuit, and I will give you a biscuit`~~`look, you can prod me, point at a biscuit, and I will give you a biscuit`", then you can show them the purpose of communication. And in some way there's very little syntax or structure to learn there.

For the next step, we did this thing called PECS with some of the children. It's a Picture Exchange Communication System [? ] where they can give me a little bit of card that has a picture of a biscuit on, and I give them a biscuit in return. So the key thing here is the code. This card is this executable program. It says "~~`give me a biscuit`~~`give me a biscuit`".

The funny thing is, once a child realizes, "~~`oh I can get what I want from this`~~`oh I can get what I want from this`" and "~~`I can make people do things`~~`I can make people do things`" then they quickly become very motivated to learn how to communicate more complicated things.

**DAVE:** That's great. I do think you're right. That example gets to the heart of what bugs me about abstract language discussions versus all-in ~~`natural code`~~`natural code`.

What matters is that a communication occurs, and that it causes something to happen. It causes the world to become better for the transmitter. If the act of transmitting code, by holding up that picture card, actually leads to "~~`yum yum`~~`yum yum`" then all the syntax and stuff can come later. I think it could really help if we thought of programming languages starting from no syntax, starting from just signals.

### 4.2 From spatial computing to symbols

**DAVE:** A key aspect of what you said is that it relies on spatial computing [e.g., ? ? ]. You said "~~`point at the biscuit and I will give you a biscuit`~~`point at the biscuit and I will give you a biscuit`". That depends on being physically close to the thing that you're indexing because you cannot say "~~`biscuit`~~`biscuit`" yet. You don't know how to do that, but when it's close enough, you can just indicate *that thing right there*. And that's how semantics *begins*.

Then going to the cards is great as a next step because that is an example of a *pointer dereference*. You have a symbol that, physically, is just some ink on paper, and yet it can *refer* to a biscuit, and program someone to bring it to you, even if it's in another room, out of sight.

**LU:** We talked about it as "~~symbols~~symbols". That's the terminology we used in that field of education, and it's the terminology I use now when I talk about coding. That symbol could be the child ~~pulling on your coat~~pulling on your coat, or a particular made-up sound, as long as you know that it means "~~biscuit~~biscuit".

**DAVE:** Right right, it could be anything. All that matters is that there's a shared understanding. It's a little specific API.

~~Sample applications page from the SelfImage API datasheet.~~

### 4.3 "Natural code" as a symbol

### 4.4 ~~Natural code as a symbol~~

**LU:** When we saw children make a jump to verbal language, it was often when those first symbols just became more inconvenient. Getting out the biscuit card from your little pack of cards becomes a chore. Then you realize that it's much quicker and more effective to just say the word, "~~biscuit~~biscuit".

And now I see that happening with me and you too. Sometimes, I want to refer to a concept that we've previously discussed, but in a much more concise way, and we don't have a word or symbol for that concept yet, so we keep having to go through it in its entirety again and again. I mean, we can edit that out in the essay, but it's very time-consuming for us right here, right now.

So the solution, of course, is to make a symbol that can serve as an abstraction. We need a word that we can *dereference* to get a whole concept. And that's what the term "~~natural code~~natural code" can be. It can refer to this shared understanding that we're building.

**DAVE:** I see. So now, now you're at a meta level.

**LU:** ~~Natural code~~ "Natural code" is a symbol. It's a namespace. It's an API that we can use to make our communication more effective. But it only works if we both understand what it means, so that it's a compatible format for us both to use. That's exactly what we're doing in these dialogues — we're developing a shared language — we're developing our shared codebase.

## 5 The ~~SelfImage~~SelfImage API

**LU:** So, Dave: What *is* the ~~SelfImage~~SelfImage API? I know from your video [? ] that it has four processes, but what does it mean?

**DAVE:** Fields like philosophy and religion and science offer us *language* to talk about what kind of machines we all are. Like, "I think therefore I am", or "I am a collection of neurons".

**LU:** ~~Okay, imagine I've bought into the natural code idea, and now I want to put it into practice~~ Or "We are made up of needs and wants and motivations", or whatever.

**DAVE:** Right. All of these languages contain some germ of truth, but none of them are going to be wholly sufficient to answer all of the variety of questions that we might want to ask. So what we need to do is choose multiple approaches — ~~I want to start developing natural code. I want to improve the shared codebase! Well, unfortunately, that feels really hard to do, because the concept is so unsatisfyingly vague. How do I actually develop natural code? Can you spell it out for me?~~ multiple languages. I think of them as "APIs". They're clearly not perfect, and don't cover everything, but they emphasize certain parts, and make it easier to express some concepts versus others.

So the ~~SelfImage~~SelfImage (see Fig. 2) is such an API. It depicts us as arrangements of four computational processes: ~~I've been accused of being too vague before, and to some degree I will plead guilty to that. But also, that's just the nature of APIs. The whole idea is that they're abstract. I mean, like a linked list is utterly vague about what's inside it . It's utterly vague about exactly how many items you're going to need in the list,~~

1. ~~Input~~Input: Handling influences from our surroundings,
2. ~~Output~~Output: Performing work on our surroundings,
3. ~~Sequence~~Sequence: Changing internal states over time, and
4. ~~Judge~~Judge: Assessing situational desirability.

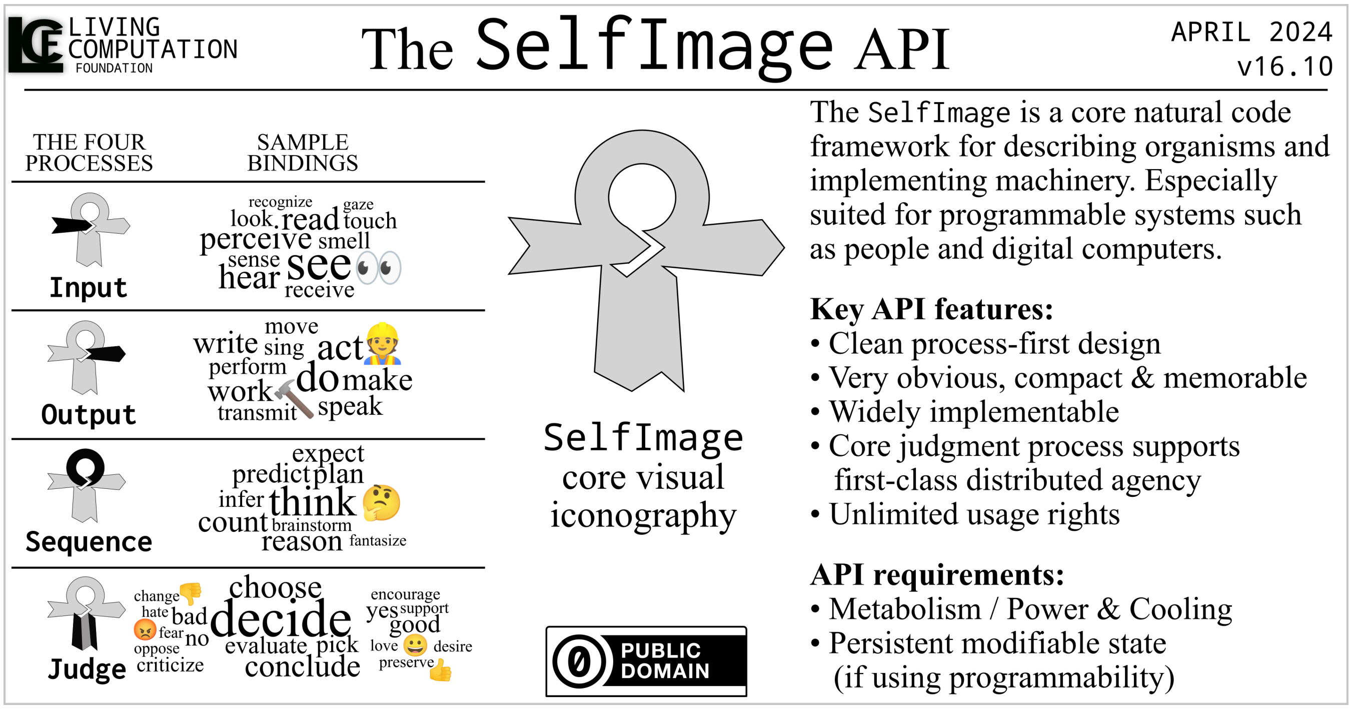
If we're interested in how we understand the world around us, we'll focus on the ~~input~~input process. If we want a deeper understanding of how we actually create and do things in the world, we'll unpack the ~~output~~output process, and so on. ~~That's by design. That's the point. It's compatible with a wide range of uses, and the~~

The ~~SelfImage~~SelfImage ~~API is the same.~~ is a really basic framework to see ourselves through a computational lens. It's a starting point.

### 5.1 API design

**LU:** ~~Right, I see. And I saw in your video [? ] how you're using the~~ To me, the ~~SelfImage~~SelfImage ~~API as a model for some example computations, like "The Daydreamer" (see Fig. 3). But, in all honesty, it feels like you could put anything in there.~~ API seems no different than a psychological model that aims to describe how people behave. It reminds me of something like Maslow's hierarchy of needs [? ], or operant conditioning [? ], even.

**DAVE:** ~~I *hope* that you could model anything — at least, any implementable machine — with the~~ Ah, okay. What I'm suggesting is that, by taking the computational metaphor,


LIVING COMPUTATION FOUNDATION
THE FOUR PROCESSES
SAMPLE BINDINGS
Input
recognize gaze look read touch perceive smell sense see hear receive
Output
move write sing act perform do make work transmit speak
Sequence
expect predict plan infer think count brainstorm reason fantasize
Judge
change hate bad fear oppose no criticize choose decide evaluate pick conclude encourage yes support good love desire preserve
PUBLIC DOMAIN

The SelfImage API

APRIL 2024
v16.10

The SelfImage is a core natural code framework for describing organisms and implementing machinery. Especially suited for programmable systems such as people and digital computers.

SelfImage
core visual iconography

**Key API features:**

- Clean process-first design
- Very obvious, compact & memorable
- Widely implementable
- Core judgment process supports first-class distributed agency
- Unlimited usage rights

**API requirements:**

- Metabolism / Power & Cooling
- Persistent modifiable state (if using programmability)

**Figure 2.** The SelfImage API datasheet cover. To propagate successfully, even the most complex and subtle ideas must also have small and memorable representations. If the idea creators fail to provide them, the idea consumers — if there are any — must and will. Here, as an example, the ~~SelfImage~~SelfImage API begins with four simple words and a single shape.

the ~~SelfImage~~SelfImage ~~API, because it~~ API can simultaneously describe both people *and* other programmable machinery. That's one difference.

And secondly, I'~~s deliberately trying~~ m claiming that the ~~SelfImage~~SelfImage API leads more directly to implementability than a psychological description, because it uses the language of computation.

**LU:** So it's not solely a *descriptive* model?

**DAVE:** Right. It can be a blueprint. It can be a recipe for how to build machinery.

**LU:** Okay, it seems more like a *design* challenge — you want to make an API that's useful, regardless of how truthful it is as a description.

**DAVE:** A scientific theory succeeds when it gives us an *unexpected truth*. But that's not the goal of an API in software design. We want an API to be as ~~general~~ unsurprising as possible. ~~But if you think any of those examples are *not* implementable, then we should focus on thatuntil we reach a shared understanding between the way you're thinking about it and the way I 'm thinking about it, and we can reach an agreement. Or maybe we discover there's some deeper bug with the API, and we need to back up.~~ We want to adhere to the law of least astonishment. [? ]

~~Okay so perhaps the vagueness of natural code is actually a *feature*?~~ Ideally, an API should not teach us anything new. The goal of an API is to be *obvious*, and that's what we can judge it on — how universally obvious it is.

~~Yeah it's Vagueness As A Service~~**LU:** I think I get it. It's more like user experience design, in a way. It's a communication tool that lets us talk about the world in a certain way — under a computational lens.

It should be as easy and straightforward to use as possible.

### 5.2 Shared code

**LU:** Sometimes, when I'm developing computer code, I ~~will~~ use some tooling to help me, like Google Chrome's DevTools~~.~~ ~~It shows me what 's going on inside my machine — what~~ , to see what code is being executed, where it crashes, and so on.

But sometimes the tooling doesn't show me enough helpful information~~. In these circumstances, I often construct my own visualization~~ , so I draw my own visualizations of my code's execution ~~. I often draw it —~~ on a piece of paper, or a whiteboard, or a virtual whiteboard like tldraw [? ]. It could be a drawing of a state machine, or a flowchart, or ~~an arrangement of how my program's memory is laid out~~a memory layout. Regardless, my drawing is a highly simplified version of what's actually happening in execution.

~~To me, the / API feels like a similar kind of visualization. It's not necessarily an accurate representation of what's going~~

~~on inside my machine, but it's a helpful abstraction that allows me to think through how my code is executing, and how it could be improved.~~

On top of that, my drawings become a shared language that I can use to communicate with my colleagues. They can look at my visualization and understand what I'm trying to achieve. And if they have a suggestion for how to improve it, they can communicate with me via the shared model. They can draw on it, or edit it, or make their own version. It's a shared API we have between us.

To me, the ~~SelfImage~~SelfImage API feels like a similar kind of visualization. It's not necessarily an accurate representation of what's going on inside my machine, but it's a helpful abstraction that allows me to think through how my code is executing, and how it could be improved.

**DAVE:** Yes, absolutely. The diagram is still much simpler than the code and the machine it's depicting, but it has value in the moment. All we really need is to be confident that the diagram is implementable.

When we derive a diagram from running code, we know the diagram is implementable, because "~~here's an implementation~~here's an implementation". But if we add another arrow, say, the diagram may no longer be implementable in the existing code. And that tension, between simplified abstractions and actual implementations, is what code development is all about.

If there's a small set of abstract but widely implementable processes with a lot of descriptive power, we should give them a name to go by. That's all the ~~SelfImage~~SelfImage API is.

## 6 Developing natural code

**LU:** ~~Looking back,~~ Okay, imagine I've bought into the "natural code" idea, and now I want to put it into practice — I want to start developing "natural code". I want to improve the shared codebase! Well, that feels really hard to do, because the concept is so unsatisfyingly vague. How do I actually develop "natural code"? Can you spell it out for me?

**DAVE:** I've been accused of being too vague before, and to some degree I will plead guilty to that. But also, that's just the nature of APIs. The whole idea is that they're abstract. I mean, like a linked list is utterly vague about what's inside it. It's utterly vague about exactly how many items you're going to need in the list, and so on. That's by design. That's the point. It's compatible with a wide range of uses, and the ~~SelfImage~~SelfImage API is the same.

**LU:** Right, I see. And I saw in your video [?] how you're using the ~~SelfImage~~SelfImage API as a model for some example computations, like "The Daydreamer" (see Fig. 3). But, in all honesty, it feels like you could put anything in there.

**DAVE:** I ~~can see people getting stuck on the "programming other people" idea. A gut reaction I get from it is that it seems really cold, you know? It almost seems psychopathic, because it sounds like~~ *hope* that you could model anything — at least, any implementable machine — with the ~~SelfImage~~SelfImage API, because it's ~~all about trying to manipulate other people.~~ deliberately trying to be as general as possible. Like, if either of us think some example is *not* implementable, then we should focus on that until we reach some shared notion of an implementation strategy. Or maybe we discover there's some deeper bug with the API, and we need to back up.

~~But I~~**LU:** Okay so perhaps the vagueness of natural code is actually a *feature*?

**DAVE:** Yeah it's Vagueness As A Service.

### 6.1 Traditional programming

**LU:** And what about this? One reviewer felt that "natural code" doesn't help with traditional programming — so it's maybe off-topic for *Onward! Essays*.

**DAVE:** It's true we didn't stress implications for traditional programming, but I think there are some basic connections.

**LU:** And what are they?

**DAVE:** One way natural code informs traditional programming is by shouting "Snap out of it! It's time to get over hardware determinism!" And abandoning hardware determinism drives a focus on robust-first programming [?].

**LU:** Yes, I guess, with the MFM architecture [?], and T2 Tile Project [?], you've made a case for a new, non-deterministic kind of computer architecture. But that involves switching to a whole new hardware stack. Does robust-first speak at all to people programming on traditional hardware?

**DAVE:** Well, yeah, if the computing model is big CPU and big flat RAM and hardware determinism, serious robustness is scarcely an option. But still, natural code can at least offer support for some programming concepts over others.

**LU:** Like what?

**DAVE:** Well, here's three:

1. *Event-driven programming*: Prefer dialogue over monologue — shorter code sequences interacting.
2. *Self-stabilizing code*: First be robust, then as correct as possible, then as efficient as necessary.
3. *Minimize state*: Prefer recomputing over caching where possible; let the world be its own representation.

And maybe overall, natural code says be wary of people advocating correctness and efficiency only. I think traditional programming needs to hear that!

### 6.2 Debugging natural code

**LU:** I'm thinking back to when I said that "programming other people" seems cold and —

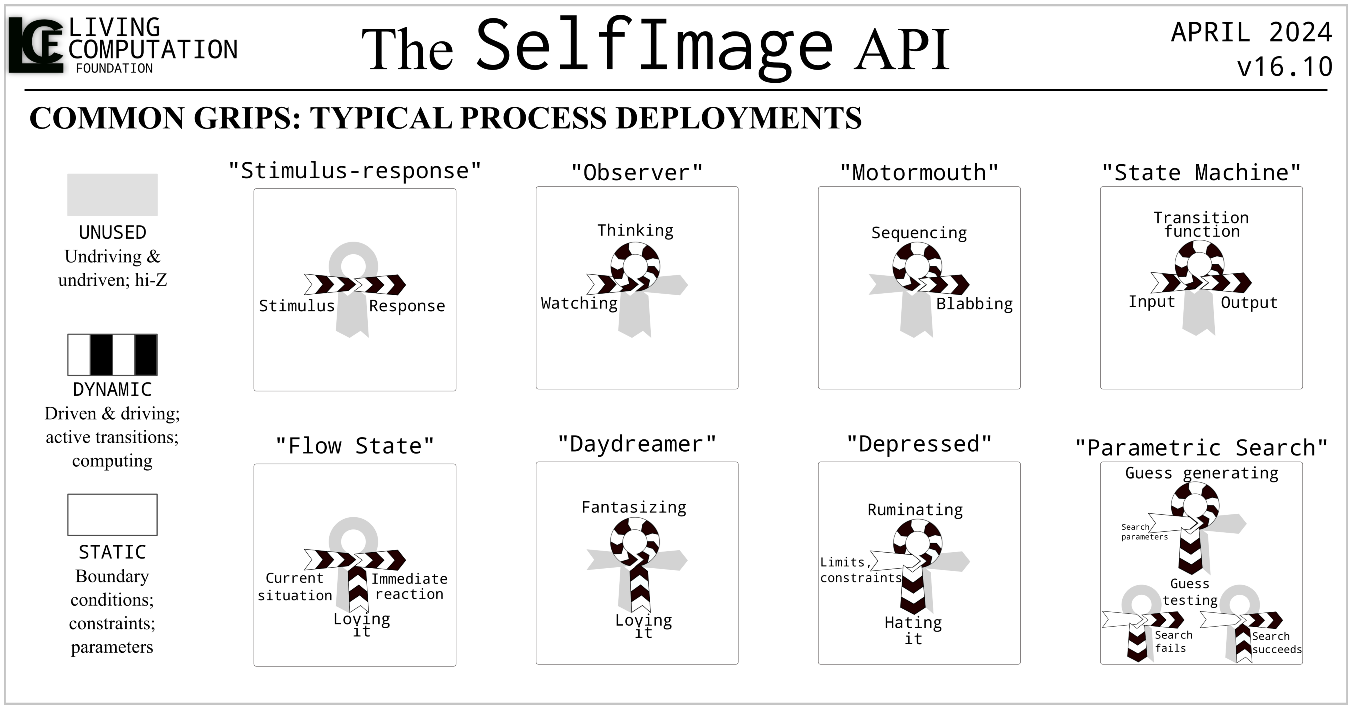


**Figure 3.** Sample applications page from the SelfImage API datasheet. Though informal, rough, and categorical, such simple visual representations of ~~SelfImage~~**SelfImage** configurations —"grips" — may offer insights. For example, highlighting the similarities between "Parametric Search" and "Depressed" might possibly be useful to an organism stuck in the latter grip.

**DAVE:** And how do you feel now?

**LU:** Well, I still think it seems cold. And I can see that "coldness" blocking some people.

But I see you'~~m realizing that you'~~re not saying it for a cold-hearted reason. Instead, it's a way of thinking deeply about our communications, that will allow us to try to figure out how to become more compatible with each other, right?

My natural code is going out and yours is coming back. And maybe we're not hearing each other. Maybe we're not on the same page. Maybe we're struggling on the same thing. Maybe we're both trying to improve the world in the same way, but we're not able to work together. We're not able to *understand* each other in some way.

And you have this idea of "~~Right, let's look at this in natural code terms~~Right, let's look at this in natural code terms". "~~Let's try to look at where our code is incompatible." "Let's try to find a shared code that we both understand." "Let's try to transpile the code between us.~~Let's try to look at where our code is incompatible." "Let's try to find a shared code that we both understand." "Let's try to transpile the code between us."

**DAVE:** In the secret fortress of solitude in our heads, we are all trying to get what we want, but there's this huge veil of silence over that fact. We don't quite admit it, because it doesn't sound good. It sounds selfish, and so people ask, "~~Do you do good because you're actually trying to do good or just because you're selfishly trying to make people give you the results of being good?~~Do you do good because you're actually trying to do good or just because you're selfishly trying to make people give you the results of being good?" Well, so that is an example of something that can be cleared up by taking this point of view of code transmissions.

We are coders. We're all trying to get what we want. And because we're alive, what we want tends to be stuff we think will help us persist and survive in the world. And cookies are a proxy for survival because we need energy to persist and sweets are a proxy for energy. So we think we're helping ourselves persist, and it's "~~yes, yes, cookie, yes~~yes, yes, cookie, yes" from the hardware. Then we end up looking like me.

**LU:** And I think that most of us, as adults, we pick that up implicitly, right? We learn that we can influence other people by deploying code, verbally or otherwise. Like saying "~~Hey, duck!~~Hey, duck!" to someone and they duck.

But some of these children I worked with — for one reason or another, they struggled to pick this lesson up implicitly, so they had to explicitly learn it. And they often ended up understanding it better than many of their peers, who did learn it implicitly. These children gained mastery over communication by debugging it when it wasn't serving their interests as well as it could have. Perhaps more people could benefit from this kind of explicit debugging of their communication — of their code transmission.

**DAVE:** Right! We can often see implementations most clearly when they break down. The ~~code the childrenwere transmitting~~ children's code wasn't executing the way they wanted, and that's ~~certainly very~~ frustrating, so you worked together to debug that. You made super-accessible communication channels, so step by step the kids could start choosing to transmit code that makes their world better.

**DAVE:** Once we admit, or once we just ~~choose~~decide, that language is code, then the ~~`natural code`~~ `natural code` framework says it's all about acts of code transmission. Some transmission through space from ~~A to B at time C~~`A` to `B` at time `C`: What code shipped? Did that transmission happen for a good reason? Would we rather widen that channel, or maybe block it? All such questions are fair discussion topics among “~~`natural coders`~~`natural coders`.”

The overall goal is to debug the great machine and improve its codebase. Close up, between us, the purpose is to find a win-win, so I understand what your language means in my terms and vice versa — so we can share code effectively and our collective distributed machine works better. And I think, if we choose to be resolutely explicit about that — that we are coders, we are developers, and we're trying to debug the machine — we might all be happier and more productive, and our world more robust and sustainable.

### 6.3 Buggy code

**DAVE:** But unfortunately there are also grifters, who deliberately and knowingly *ship buggy code*, where the transmitted narrative is a trick to cover theft, or corruption, or other evil.

**LU:** People sowing division, spreading misinformation —

**DAVE:** Even good people can ship bad code in moments of weakness. They know in their hearts that the code isn't *exactly* right, and that its bugs benefit the transmitter. In tiny ways at least, it's like nobody is completely without sin, so typically all remain silent. And the result is that good people's petty hypocrisies enable other's great crimes.

**LU:** Some bugs are bigger than others.

**LU:** One of the reviewers expressed concern that `natural code` can be misused.

**DAVE:** For sure. `Natural code` gets misused a *lot.*

**LU:** Yes, it's happening already, all around us, whether we explicitly acknowledge it as `natural code` or not, harmful `natural code` is being shipped and —

**DAVE:** And we'd be better off acknowledging that —

**LU:** Because then we can be more explicit about naming it as such, and calling it out, and then —

**DAVE:** And then we can start talking like developers, and get down to debugging our shared natural codebase.

## 7 Owning our natural codebase

**DAVE:** Okay another run at a summary: There are many many ways to describe things. On the one hand, they are not all equally good for all purposes, but on the other hand, there's no one language that's “~~`uniquely most true`~~`uniquely most true`” either. You talk differently to your grandma than to a colleague or friend, because different code receivers understand differently, and have different shared dictionaries ~~with you.~~between you.

So the claim has two parts. First: We have to *make choices* about how to describe and understand ourselves and the world. We cannot delegate those choices, even if we really want to — not to other people, not to the universe itself. And second: One choice should always be that *we are coders.*

It's about all our code transmissions, natural and artificial. Is it all a metaphor? Sure, if you need it to be, but I'll still claim it's a simple and powerful basis for understanding and improving our shared computation.

So ~~`natural code`~~ `natural code` will be one of many ways of describing and building things. It won't erase art, or philosophy, or any of those things. But it will always be available in addition. “~~`Let's consider this in terms of natural code.`~~`Let's consider this in terms of natural code.`”

**LU:** Over the last months we have attempted to own the ideas of natural code — struggling towards shared understanding where previously there was none. My hope is that other people will see our example and become inspired to do the same, though we cannot know for sure if that will happen.

**DAVE:** Indeed. We can only do what we can, and it won't all be easy. I hope that, once they see themselves as *natural coders*, people of good faith everywhere will work for a better shared codebase. I do have hope. ~~But unfortunately there are also grifters, who deliberately and knowingly *ship buggy code*, where the transmitted narrative is a trick to cover theft, or corruption, or other bad behavior.~~

~~People sowing division, spreading misinformation —~~

~~Even good people can ship bad code in moments of weakness. They know in their hearts that the code isn't *exactly* right, and that its bugs benefit the transmitter. In tiny ways at least, it's like nobody is completely without sin, so typically all remain silent. And the result is that good people's petty hypocrisies enable other's great crimes.~~

~~Some bugs are bigger than others.~~

**LU:** To me, natural code is about building bridges, and getting people to work together — to name and call out the bad code, while celebrating the shipping of better code.

To do this, we *may as well* talk in terms of natural code. We *may as well* talk about developing our APIs, and debugging our difficulties, and improving our codebase. And I do believe that more and more people will join us on this, and become more deliberate about being natural coders.

**DAVE:** And this is just a beginning.

**LU:** ~~Step by step!~~ “Step by step”!

**DAVE:** Step by step.

Revised 18 July 2024