

# 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

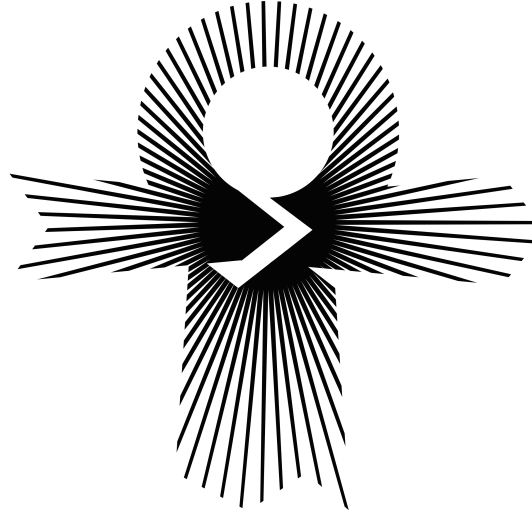


Figure 1. The SelfImage starburst.

## 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.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org). *SPLASH '24, October 20–25, 2024, Pasadena, California, United States*

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/18/06

<https://doi.org/XXXXXXX.XXXXXXX>

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

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

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

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

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

### 130 1.1 Building machinery

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

134 **DAVE:** That’s right. Machines that somehow work to pre-  
135 serve their structures, their *patterns*, are what we call “life”.  
136 Persistence involves maintenance and repair, but also build-  
137 ing copies.

138 **LU:** I guess so! Though I was thinking more about traditional  
139 ideas of “building machinery”, like a beaver building a dam,  
140 or a wasp building a nest.

142 **DAVE:** That happens too. And humans build bridges, rock-  
143 ets, and programmable computers. I think about “building  
144 machinery” writ large. It can be something like lighting a fire,  
145 or folding a paper airplane, or moving a rock off a path.

146 **LU:** You’re using the phrase “building machinery” extremely  
147 loosely here, right? Because to me, “building machinery”  
148 sounds like *creating* something, or *making* an artifact of  
149 some sort. But you’re using it to refer to what seems like  
150 just an action, or a process.

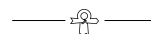
151 “Lighting a fire” doesn’t sound like building anything  
152 at all. It just sounds like *enacting a change*.

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

155 But, say you’re working at a hamburger joint, where all  
156 you have to do is slap a burger on a bun and put on ketchup  
157 or mayo, and it’s done. You’re “building a machine” out of  
158 other complex arrangements of matter.

159 **LU:** You’re changing the arrangement of the burger’s ingre-  
160 dients, and that’s what you’re calling “building machinery”.  
161 It’s not that you’ve “created” these ingredients, but you’ve  
162 built them into a particular pattern.

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



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

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

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

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

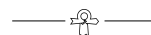
174 **LU:** A human?

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

178 And maybe a hamburger isn’t the cleanest example —

179 **LU:** It really isn’t the cleanest example.

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



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

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

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

### 192 1.2 Contracting machinery

193 **LU:** And you’re saying there are two ways of building ma-  
194 chinery? One way is to do it yourself, to build it *directly*.

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

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

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

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

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

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

### 221 1.3 Human hardware

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

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

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

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

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

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

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

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

### 245 1.4 Coldness and evil

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

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

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

250 **DAVE:** Oh. Well.

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

253 **DAVE:** Well, I —

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

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

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

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

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

266 **LU:** I was genuinely worried about this when we submitted,  
267 because it's something I agree with. There is this pervasive

268 fallacy to see all our problems as technological. I hate it,  
269 and I see it time and time again.

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

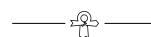
273 **DAVE:** I just can't imagine somebody saying that seriously.

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

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

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

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



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

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

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

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

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

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

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

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

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

315 **LU:** It's mainly "avoid cross-talk" and "be sure to set up  
316 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.

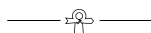
LU: Or smash it with a hammer.



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.

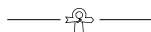


DAVE: People certainly don’t do the same thing every time.

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.

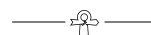
LU: Has non-determinism been a blocker for some people?

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.

551 **LU:** And maybe you see a divide between the “art world”  
552 and the “non-art world.” But for me, it isn’t helpful to draw  
553 these lines when trying to bring the different fields together.

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

556 **DAVE:** Yes, yes, yes, it’s like “just kidding, it was all a  
557 dream”.

558 **LU:** “It was art the whole time”.

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

562 **LU:** Except —

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

568 **LU:** Yeah okay, I just think it’s good I got you to admit that  
569 the **SelfImage** is art.  
570

## 571 4 The nature of natural code

572 **DAVE:** The canonical Chomsky hierarchy stuff [11] is all  
573 about languages having compositional, recursive, syntactic  
574 structures, allowing language users to create open-ended  
575 complexity. And I think that’s great, but it doesn’t go nearly  
576 far enough. On their own, syntactic properties are almost a  
577 detail. There’s other ways to get modularity, complex repre-  
578 sentations, and so on. For example, you could just list chosen  
579 words in a random order — “wood, hammer, nail” — and  
580 it could create a notion in the listener’s head that could be  
581 quite rich, with hardly any syntax.  
582

583 **LU:** Splinters.

584 **DAVE:** Right. Sore thumb. So I’m hesitant to embrace the idea  
585 that it’s all about *language* and which structural properties of  
586 language are important. I think that’s wrong. Instead, I want  
587 to talk about “code”, and not “programming language”. And  
588 by saying code, I want to rope in signals, gestures, grunts —  
589 stuff that seems below the level of programming languages.  
590

### 591 4.1 Starting from signals

592 **LU:** Okay, “code” “code” “code”. Not just language. I think  
593 that’s right. You can get too focused on the structure and  
594 syntax of language. I think it’s more important to think about  
595 the *purpose* of language — the purpose of code, I mean.  
596

597 When I was a teacher, I worked with very young children  
598 who struggled to communicate with other people, for various  
599 reasons. It wasn’t that these children necessarily struggled  
600 with *language*. In fact, some of them were hugely competent  
601 with language and its syntax. They struggled with *communi-*  
602 *cation* in a more general sense, which can sometimes involve  
603 no syntax or language at all. It can mean “prodding someone”,  
604

605 “looking at someone”, or simply “tugging on their hand” to  
606 pull them along.

607 The first step that we always tried to get across to these  
608 young children was, “look at all the good things you can  
609 get from interacting with someone”, and we used a lot of  
610 *biscuits*.  
611

612 Most children love biscuits, right?

613 **DAVE:** Cookies.

614 **LU:** And if you can tell them, “look, you can prod me, point  
615 at a biscuit, and I will give you a biscuit”, then you  
616 can show them the purpose of communication. And in some  
617 way there’s very little syntax or structure to learn there.  
618

619 For the next step, we did this thing called PECS with some  
620 of the children. It’s a Picture Exchange Communication Sys-  
621 tem [9] where they can give me a little bit of card that has a  
622 picture of a biscuit on, and I give them a biscuit in return. So  
623 the key thing here is the code. This card is this executable  
624 program. It says “give me a biscuit”.  
625

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

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

633 What matters is that a communication occurs, and that it  
634 causes something to happen. It causes the world to become  
635 better for the transmitter. If the act of transmitting code, by  
636 holding up that picture card, actually leads to “yum yum” then  
637 all the syntax and stuff can come later. I think it could really  
638 help if we thought of programming languages starting from  
639 no syntax, starting from just signals.  
640

### 641 4.2 From spatial computing to symbols

642 **DAVE:** A key aspect of what you said is that it relies on spa-  
643 tial computing [e.g., 1, 24]. You said “point at the biscuit  
644 and I will give you a biscuit”. That depends on being  
645 physically close to the thing that you’re indexing because  
646 you cannot say “biscuit” yet. You don’t know how to do  
647 that, but when it’s close enough, you can just indicate *that*  
648 *thing right there*. And that’s how semantics *begins*.  
649

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

655 **LU:** We talked about it as “symbols”. That’s the terminology  
656 we used in that field of education, and it’s the terminology I  
657 use now when I talk about coding. That symbol could be the  
658 child pulling on your coat, or a particular made-up sound,  
659 as long as you know that it means “biscuit”.  
660

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

### 663 4.3 “Natural code” as a symbol

665 **LU:** When we saw children make a jump to verbal language,  
666 it was often when those first symbols just became more in-  
667 convenient. Getting out the biscuit card from your little pack  
668 of cards becomes a chore. Then you realize that it's much  
669 quicker and more effective to just say the word, “biscuit”.

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

677 So the solution, of course, is to make a symbol that can  
678 serve as an abstraction. We need a word that we can *deref-*  
679 *erence* to get a whole concept. And that's what the term  
680 “natural code” can be. It can refer to this shared understand-  
681 ing that we're building.

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

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

## 690 5 The SelfImage API

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

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

698 **LU:** Or “We are made up of needs and wants and motivations”,  
699 or whatever.

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

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

- 711 1. **Input:** Handling influences from our surroundings,
- 712 2. **Output:** Performing work on our surroundings,
- 713 3. **Sequence:** Changing internal states over time, and
- 714 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.

### 723 5.1 API design

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

728 **DAVE:** Ah, okay. What I'm suggesting is that, by taking  
729 the computational metaphor, the **SelfImage** API can simul-  
730 taneously describe both people *and* other programmable  
731 machinery. That's one difference.

732 And secondly, I'm claiming that the **SelfImage** API leads  
733 more directly to implementability than a psychological de-  
734 scription, because it uses the language of computation.

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

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

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

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

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

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


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

### 754 5.2 Shared code

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

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

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







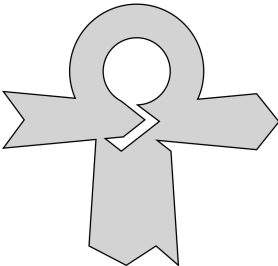
**LIVING  
COMPUTATION  
FOUNDATION**

# The SelfImage API


APRIL 2024  
v16.10

---

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



**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 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.

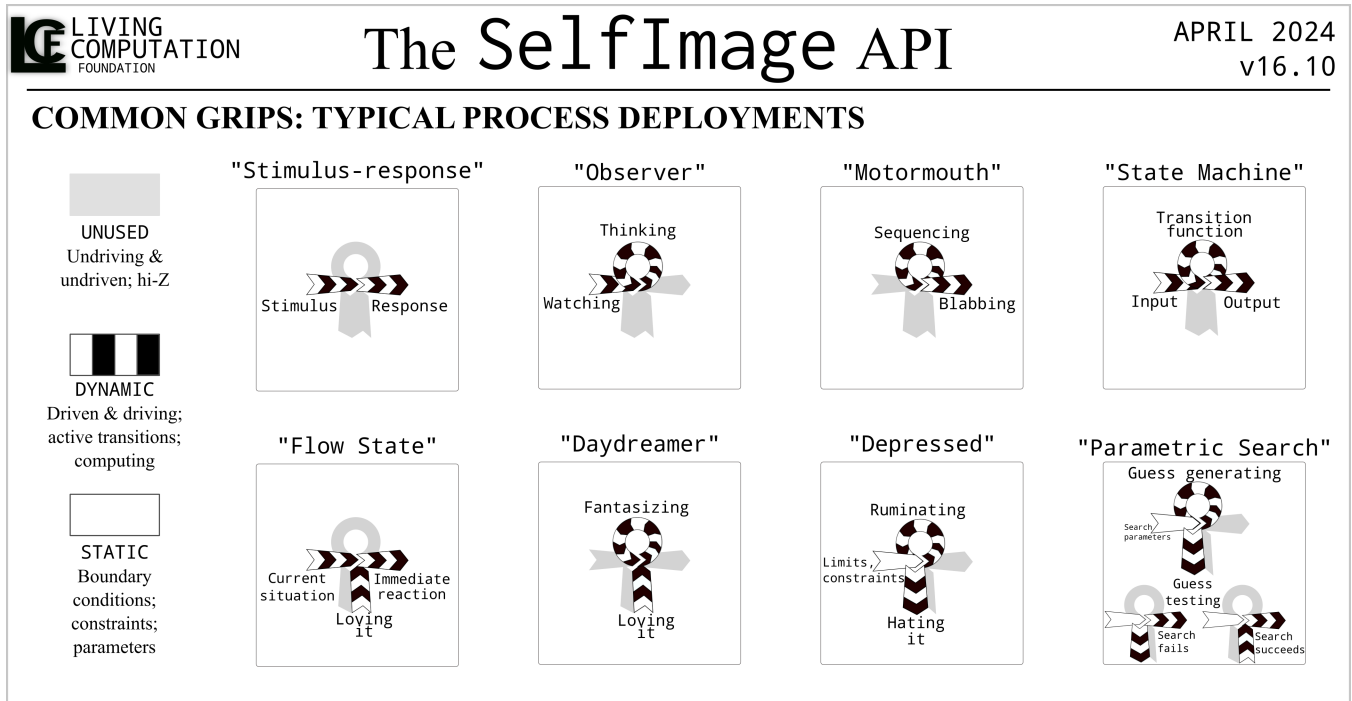
**LU:** Right, I see. And I saw in your video [4] how you’re using the SelfImage API as a model for some example computations, like “The Daydreamer” (see Fig. 3). But, in all honesty, it feels like you could put anything in there.

**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.





**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.

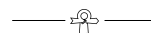
### 6.3 Buggy code

**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.

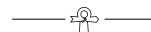
## 7 Owning our natural codebase

**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=14fIQ8XdvJM>.
- [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](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](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](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 code 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

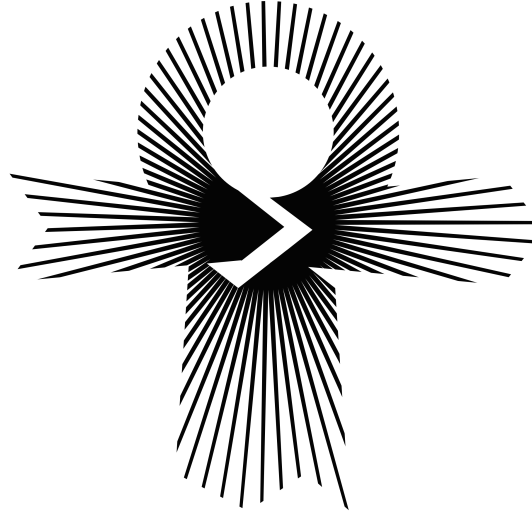


Figure 1. The SelfImage starburst.

## 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org). *SPLASH '24, October 20–25, 2024, Pasadena, California, United States*  
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

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

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

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

115 LU: A non-zero amount?

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

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

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

### 131 1.1 Building machinery

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

134 DAVE: That's right. Machines that somehow work to pre-  
135 serve their ~~patterns~~structures, their patterns, are what we  
136 call "~~life~~life". Persistence involves maintenance and repair,  
137 but also building copies.

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

141 DAVE: That happens too. And humans build bridges, rock-  
142 ets, and programmable computers. I think about "~~building~~  
143 ~~machinery~~building machinery" writ large. It can be some-  
144 thing like lighting a fire, or folding a paper airplane, or mov-  
145 ing a rock off a path.

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

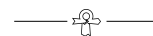
152 "~~Lighting a fire~~Lighting a fire" doesn't sound like  
153 building anything at all. It just sounds like ~~enacting a change~~enacting  
154 a change.

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

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

161 LU: You're changing the arrangement of the burger's ingre-  
162 dients, and that's what you're calling "~~building machin-~~  
163 ~~ery~~building machinery". It's not that you've "~~created~~created"  
164 these ingredients, but you've built them into a particular pat-  
165 tern.

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



168 LU: Okay ~~that makes sense to me.~~ Youso you said that a  
169 burger is a machine and —

170 DAVE: The reviewers had some troubles with that.

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

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

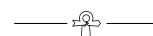
176 LU: A human?

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

180 And maybe a hamburger isn't the cleanest example —

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

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



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

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

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

### 195 1.2 How to build Contracting machinery

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

198 DAVE: Wood, hammer, nail. Yeah.

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

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



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

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

225 LU: Okay, I see where this is going.  
226

### 227 1.3 Human hardware

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

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

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

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

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

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

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

253 I think we should use our knowledge of programming  
254 languages, of software and computing, to examine  
255 our own natural code. To understand it and debug it.  
256 To make society better, and to improve our shared  
257 codebase.  
258

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

### 263 1.4 Coldness and evil

264 LU: ~~I mean~~This idea that people “program” other people. To  
265 me it seems — ~~before we get to that~~

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

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

269 DAVE: ~~Oh. Well.~~

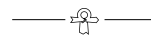
270 LU: ~~It almost seems psychopathic, because it sounds like it's~~  
271 ~~all about trying to manipulate other people.~~

272 DAVE: ~~Well, I —~~  
273  
274  
275

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

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

280 LU: ~~So we can't boil down communication to just “getting~~  
281 ~~someone to do stuff” because that's cold, and it's not true!~~



284 LU: ~~Reviewer C is worried about “the ideological, tech-~~  
285 ~~nocratic undertones” of the essay, and “it's a pervasive~~  
286 ~~fallacy in the tech world to see all our problems as~~  
287 ~~technological” and “Every human interaction is reduced~~  
288 ~~to a kind of programming”.~~

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

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

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

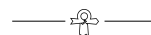
297 DAVE: ~~I just can't imagine somebody saying that seriously.~~

300 LU: ~~For example, I read a recent essay [?] saying that “AI~~  
301 ~~will soon come to the rescue” for accessibility.~~

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

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

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



316 DAVE: ~~One answer to such criticisms is that we are reading~~  
317 ~~the concept of “technology” broadly enough to include stuff~~  
318 ~~that's not traditional technology. People can hear us say~~  
319 ~~“technology” and think it means traditional programming~~  
320 ~~languages and computers and “tradtech” generally.~~

321 LU: ~~Right, we say “natural code can help us” but sometimes~~  
322 ~~people hear “traditional technology can help us”.~~

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

328 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.

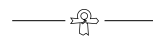


DAVE: People certainly don't do the same thing every time.

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.

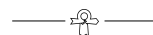
DAVE: Yeah, maybe we'll succeed. Maybe we won't. The machines executing the code of this essay are going to be way non-deterministic.



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?

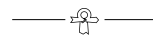
DAVE: Some people would outright say "~~without deterministic execution, it's not computation.~~without deterministic execution, it's not computation."



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.~~

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

554 **DAVE:** We're trying to explicitly frame things as code.  
 555

### 556 3.4 Blending fields

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

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

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

570 **DAVE:** Well —

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

573 **DAVE:** What? What!?

574 **LU:** Yes, I mean, I would —

575 **DAVE:** Name one!

576 **LU:** The **SelfImageSelfImage**. That's art! (See Fig. 2.)

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

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

581 **DAVE:** Yeah.

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

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

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

590 **LU:** "It was art the whole timeIt was art the whole  
 591 time".

592 **DAVE:** For the **SelfImageSelfImage** in that sense, you are  
 593 100% right. There *is* an art component to it, and a marketing  
 594 component — an attempt to be viral, which I have completely  
 595 failed at.

596 **LU:** Except —

597 **DAVE:** Well I mean, everybody wants the next zero on their  
 598 views, on their citations, on their patreon, whatever it hap-  
 599 pens to be. But I'm still only down at the sort of two to three  
 600

601 zeroes range, so, you know, I can legitimately claim lack of  
 602 virality, and — well, anyway, that's another topic.

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

605 **DAVE:** Okay I made up a couple "datasheet pages" to present  
 606 some /API stuff—

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

609 **DAVE:** Maybe we can just sneak in some "(see Fig. 2)"s like  
 610 we're doing with citations?—

### 611 3.5 The /API

## 612 4 The nature of natural code

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

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

619 Or "We are made up of needs and wants and motivations"  
 620 or whatever—

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

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

- 631 1. **InputInput:** Handling influences from our surroundings,
- 632 2. **OutputOutput:** Performing work on our surroundings,
- 633 3. **SequenceSequence:** Changing internal states over time,  
 634 and—
- 635 4. **JudgeJudge:** Assessing situational desirability—

636 If we're interested in how we understand the world around  
 637 us, we'll focus on the **inputinput** process. If we want a deeper  
 638 understanding of how we actually create and do things in  
 639 the world, we'll unpack the **outputoutput** process, and so  
 640 on—

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

### 643 4.1 API design

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

661 Ah, okay. What I'm suggesting is that, by taking the computational ~~with~~ language. In fact, some of them were hugely competent  
 662 metaphor, the / API can simultaneously describe both people with language and its syntax. They struggled with *commu-*  
 663 and other programmable machinery. That's one difference. *nication* in a more general sense, which can sometimes in-

664 And secondly, I'm claiming that the / API leads more directly involve no syntax or language at all. It can mean "prodding  
 665 to implementability than a psychological description, because someone prodding someone", "looking at someone looking  
 666 it uses the language of computation. at someone", or simply "tugging on their hand tugging  
 667 So it's not solely a *descriptive* model? on their hand" to pull them along.

668 Right. It can be a blueprint. It can be a recipe for how to The first step that we always tried to get across to these  
 669 build machinery. young children was, "look at all the good things you  
 670 Okay, it seems more like a *design* challenge — you want can get from interacting with someone" look at all the  
 671 to make an API that's useful, regardless of how truthful it is good things you can get from interacting with someone",  
 672 as a description. and we used a lot of *biscuits*.

673 A scientific theory succeeds when it gives us an *unexpected* Most children love biscuits, right?  
 674 truth. But that's not the goal of an API in software design. DAVE: Cookies.  
 675 We want an API to be as unsurprising as possible. We want  
 676 to adhere to the principle of least astonishment. LU: And if you can tell them, "look, you can prod me,  
 677 Ideally, an API should not teach us anything new. The point at a biscuit, and I will give you a biscuit" look,  
 678 goal of an API is to be *obvious*, and that's what we can judge you can prod me, point at a biscuit, and I will give  
 679 it on — how universally obvious it is. you a biscuit", then you can show them the purpose of  
 680 I think I get it. It's more like user experience design, in communication. And in some way there's very little syntax  
 681 a way. It's a communication tool that lets us talk about the or structure to learn there.

682 world in a certain way — under a computational lens. For the next step, we did this thing called PECS with some  
 683 It should be as easy and straightforward to use as possible. of the children. It's a Picture Exchange Communication Sys-  
 684 The canonical Chomsky hierarchy stuff [?] is all about lan- tem [?] where they can give me a little bit of card that has a  
 685 guages having compositional, recursive, syntactic structures, picture of a biscuit on, and I give them a biscuit in return. So  
 686 allowing language users to create open-ended complexity. the key thing here is the code. This card is this executable  
 687 And I think that's great, but it doesn't go nearly far enough. program. It says "give me a biscuit give me a biscuit".

688 Syntactic language properties, on On their own, syntactic The funny thing is, once a child realizes, "oh I can get  
 689 properties are almost a detail. There are's other ways to get what I want from this" oh I can get what I want from this"  
 690 modularity, and complex representations, and so on. For ex- and "I can make people do things" I can make people do  
 691 ample, you could just list chosen words in a random order ~ things" then they quickly become very motivated to learn  
 692 "wood, hammer, nail" — and it could create a notion in the how to communicate more complicated things.

693 "Wood, hammer, nail" — and that could be quite rich, with hardly any syntax. DAVE: That's great. I do think you're right. That example  
 694 LU: Splinters. gets to the heart of what bugs me about abstract language  
 695 DAVE: Right. Sore thumb. So I'm hesitant to embrace the discussions versus all-in *natural code* natural code.  
 696 idea that it's all about *language* and which structural causes something to happen. It causes the world to become  
 697 properties of language are important. I think that's wrong. better for the transmitter. If the act of transmitting code, by  
 698 Instead, I want to talk about "code code", and not "programming holding up that picture card, actually leads to "yum-yum yum  
 699 language programming language". And by saying code code, yum" then all the syntax and stuff can come later. I think it  
 700 I want to rope in signals, gestures, grunts — stuff that seems could really help if we thought of programming languages  
 701 below the level of programming languages. starting from no syntax, starting from just signals.

702

703

704

705

#### 706 4.1 Starting from signals

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

712 When I was a teacher, I worked with very young children  
 713 who struggled to communicate with other people, for various  
 714 reasons. It wasn't that these children necessarily struggled

771 that, physically, is just some ink on paper, and yet it can *refer*  
772 to a biscuit, and program someone to bring it to you, even if  
773 it's in another room, out of sight.

774 LU: We talked about it as “~~symbols~~symbols”. That's the ter-  
775 minology we used in that field of education, and it's the  
776 terminology I use now when I talk about coding. That sym-  
777 bol could be the child ~~pulling-on-your-coat~~pulling on  
778 your coat, or a particular made-up sound, as long as you  
779 know that it means “~~biscuit~~biscuit”.

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

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

### 784 4.3 “Natural code” as a symbol

### 785 4.4 Natural code as a symbol

786 LU: When we saw children make a jump to verbal language,  
787 it was often when those first symbols just became more  
788 inconvenient. Getting out the biscuit card from your lit-  
789 tle pack of cards becomes a chore. Then you realize that  
790 it's much quicker and more effective to just say the word,  
791 “~~biscuit~~biscuit”.

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

799 So the solution, of course, is to make a symbol that can  
800 serve as an abstraction. We need a word that we can *deref-*  
801 *erence* to get a whole concept. And that's what the term  
802 “~~natural code~~natural code” can be. It can refer to this  
803 shared understanding that we're building.

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

805 LU: ~~Natural code~~ “Natural code” is a symbol. It's a names-  
806 pace. It's an API that we can use to make our communication  
807 more effective. But it only works if we both understand what  
808 it means, so that it's a compatible format for us both to use.  
809 That's exactly what we're doing in these dialogues — we're  
810 developing a shared language — we're developing our shared  
811 codebase.

## 812 5 The SelfImageSelfImage API

813 LU: So, Dave: What is the SelfImageSelfImage API? I know  
814 from your video [?] that it has four processes, but what does  
815 it mean?

816 DAVE: Fields like philosophy and religion and science offer  
817 us language to talk about what kind of machines we all are.  
818 Like “I think therefore I am” or “I am a collection of  
819 neurons”.

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

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

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

837 I've been accused of being too vague before, and to some  
838 degree I will plead guilty to that. But also, that's just the  
839 nature of APIs. The whole idea is that they're abstract. I  
840 mean, like a linked list is utterly vague about what's inside  
841 it. It's utterly vague about exactly how many items you're  
842 going to need in the list,

- 843 1. InputInput: Handling influences from our surroundings,
- 844 2. OutputOutput: Performing work on our surroundings,
- 845 3. SequenceSequence: Changing internal states over time,  
846 and
- 847 4. JudgeJudge: Assessing situational desirability.


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

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

### 857 5.1 API design

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

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


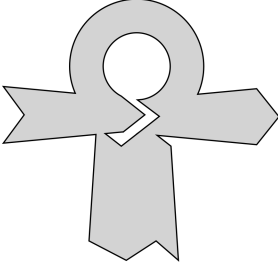






LIVING  
COMPUTATION  
FOUNDATION

# The SelfImage API

APRIL 2024  
v16.10

---

THE FOUR PROCESSES	SAMPLE BINDINGS	
 <b>Input</b>	recognize gaze look, read touch perceive smell sense see hear receive	  SelfImage core visual iconography
 <b>Output</b>	move write sing act perform do make work speak transmit	
 <b>Sequence</b>	expect predict plan infer think count brainstorm reason fantasize	
 <b>Judge</b>	change choose encourage hate bad yes support fear no evaluate pick love desire oppose criticize conclude preserve	



**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.

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

And secondly, I'm deliberately trying to claim 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 general-unsurprising as possible. But if you think any of those examples are not implementable, then we should focus on that until 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

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

994 On top of that, my drawings become a shared language  
 995 that I can use to communicate with my colleagues. They can  
 996 look at my visualization and understand what I'm trying to  
 997 achieve. And if they have a suggestion for how to improve it,  
 998 they can communicate with me via the shared model. They  
 999 can draw on it, or edit it, or make their own version. It's a  
 1000 shared API we have between us.

1001 To me, the SelfImageSelfImage API feels like a similar  
 1002 kind of visualization. It's not necessarily an accurate repre-  
 1003 sentation of what's going on inside my machine, but it's a  
 1004 helpful abstraction that allows me to think through how my  
 1005 code is executing, and how it could be improved.

1006 **DAVE:** Yes, absolutely. The diagram is still much simpler  
 1007 than the code and the machine it's depicting, but it has value  
 1008 in the moment. All we really need is to be confident that the  
 1009 diagram is implementable.


1010 When we derive a diagram from running code, we know  
 1011 the diagram is implementable, because ~~“here's an imple-~~  
 1012 ~~mentation~~ here's an implementation”. But if we add another  
 1013 arrow, say, the diagram may no longer be implementable  
 1014 in the existing code. And that tension, between simplified  
 1015 abstractions and actual implementations, is what code devel-  
 1016 opment is all about.

1017 If there's a small set of abstract but widely implementable  
 1018 processes with a lot of descriptive power, we should give  
 1019 them a name to go by. That's all the SelfImageSelfImage  
 1020 API is.

## 1021 6 Developing natural code

1022 **LU:** ~~Looking back,~~ Okay, imagine I've bought into the “natural  
 1023 code” idea, and now I want to put it into practice — I want  
 1024 to start developing “natural code”. I want to improve the  
 1025 shared codebase! Well, that feels really hard to do, because  
 1026 the concept is so unsatisfyingly vague. How do I actually  
 1027 develop “natural code”? Can you spell it out for me?

1028 **DAVE:** I've been accused of being too vague before, and to  
 1029 some degree I will plead guilty to that. But also, that's just  
 1030 the nature of APIs. The whole idea is that they're abstract. I  
 1031 mean, like a linked list is utterly vague about what's inside  
 1032 it. It's utterly vague about exactly how many items you're  
 1033 going to need in the list, and so on. That's by design. That's  
 1034 the point. It's compatible with a wide range of uses, and the  
 1035 SelfImageSelfImage API is the same.

1036 **LU:** Right, I see. And I saw in your video [?] how you're  
 1037 using the SelfImageSelfImage API as a model for some ex-  
 1038 ample computations, like “The Daydreamer”  (see Fig. 3).  
 1039 But, in all honesty, it feels like you could put anything in  
 1040 there.

1041 **DAVE:** I can see people getting stuck on the “programming  
 1042 other people” idea. A gut reaction I get from it is that it  
 1043 seems really cold, you know? It almost seems psychopathic,  
 1044 because it sounds like *hope* that you could model anything —  
 1045 at least, any implementable machine — with the SelfImageSelfImage  
 1046 API, because it's all about trying to manipulate other people,  
 1047 deliberately trying to be as general as possible. Like, if either  
 1048 of us think some example is *not* implementable, then we  
 1049 should focus on that until we reach some shared notion of  
 1050 an implementation strategy. Or maybe we discover there's  
 1051 some deeper bug with the API, and we need to back up.

1052 **But LU:** Okay so perhaps the vagueness of natural code  
 1053 is actually a *feature*?

1054 **DAVE:** Yeah it's Vagueness As A Service.

### 1055 6.1 Traditional programming

1056 **LU:** And what about this? One reviewer felt that “natural  
 1057 code” doesn't help with traditional programming — so it's  
 1058 maybe off-topic for *Onward! Essays*.

1059 **DAVE:** It's true we didn't stress implications for traditional  
 1060 programming, but I think there are some basic connections.

1061 **LU:** And what are they?

1062 **DAVE:** One way natural code informs traditional programming  
 1063 is by shouting “Snap out of it! It's time to get over  
 1064 hardware determinism!” And abandoning hardware determinism  
 1065 drives a focus on robust-first programming [?].

1066 **LU:** Yes, I guess, with the MFM architecture [?], and T2 Tile  
 1067 Project [?], you've made a case for a new, non-deterministic  
 1068 kind of computer architecture. But that involves switching  
 1069 to a whole new hardware stack. Does robust-first speak at  
 1070 all to people programming on traditional hardware?

1071 **DAVE:** Well, yeah, if the computing model is big CPU and  
 1072 big flat RAM and hardware determinism, serious robustness  
 1073 is scarcely an option. But still, natural code can at least offer  
 1074 support for some programming concepts over others.

1075 **LU:** Like what?

1076 **DAVE:** Well, here's three:

- 1077 1. *Event-driven programming*: Prefer dialogue over monologue  
 1078 — shorter code sequences interacting.
- 1079 2. *Self-stabilizing code*: First be robust, then as correct as  
 1080 possible, then as efficient as necessary.
- 1081 3. *Minimize state*: Prefer recomputing over caching where  
 1082 possible; let the world be its own representation.

1083 And maybe overall, natural code says be wary of people  
 1084 advocating correctness and efficiency only. I think traditional  
 1085 programming needs to hear that!

### 1086 6.2 Debugging natural code

1087 **LU:** I'm thinking back to when I said that “programming other  
 1088 people” seems cold and —



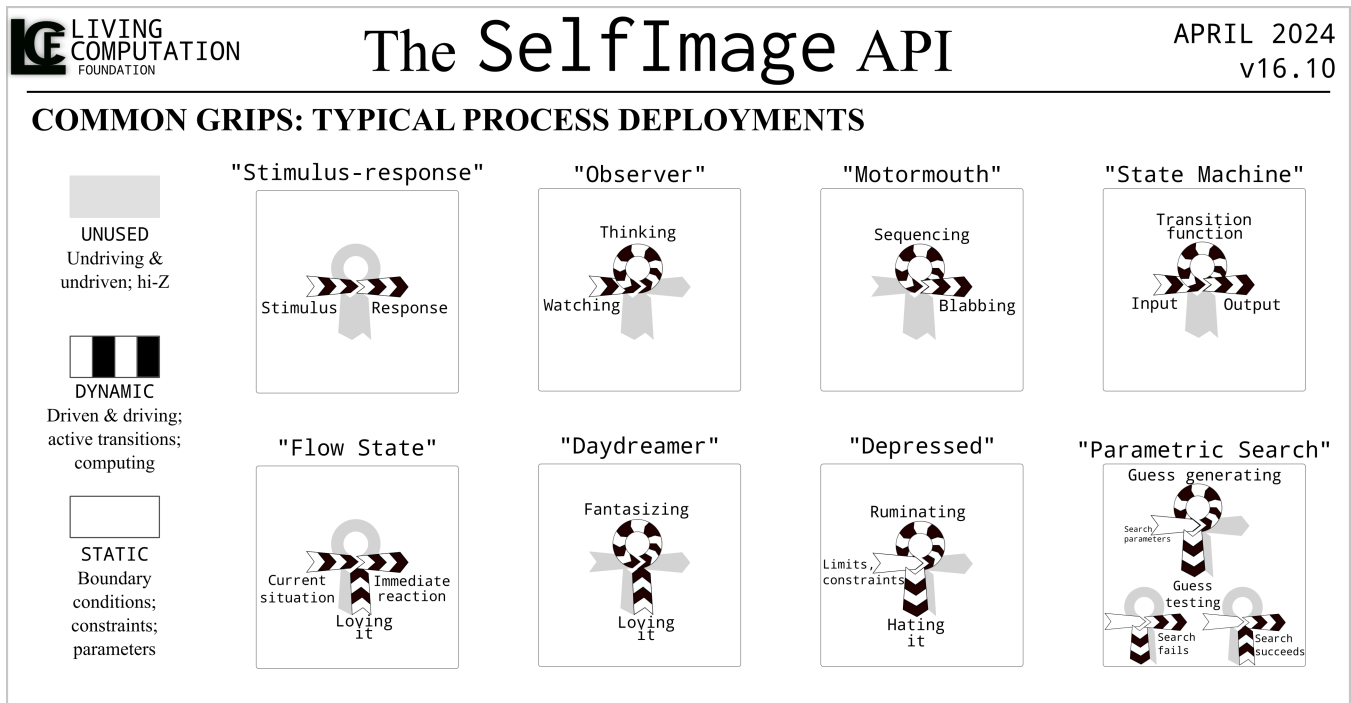


Figure 3. Sample applications page from the SelfImage API datasheet. Though informal, rough, and categorical, such simple visual representations of SelfImage configurations — “grips” — may offer insights. For example, highlighting the similarities between “Parametric Search” and “Depressed” might possibly 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’re 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

1211 as well as it could have. Perhaps more people could benefit  
1212 from this kind of explicit debugging of their communication  
1213 — of their code transmission.

1214 DAVE: Right! We can often see implementations most clearly  
1215 when they break down. The ~~code the children were transmitting~~  
1216 children’s code wasn’t executing the way they wanted, and  
1217 that’s ~~certainly very~~ frustrating, so you worked together to  
1218 debug that. You made super-accessible communication chan-  
1219 nels, so step by step the kids could start choosing to transmit  
1220 code that makes their world better.



1223 DAVE: Once we admit, or once we just ~~choose~~decide, that  
1224 language is code, then the ~~natural—code~~ natural code  
1225 framework says it’s all about acts of code transmission. Some  
1226 transmission through space from ~~A to B at time—C~~ A to B at  
1227 time C: What code shipped? Did that transmission happen  
1228 for a good reason? Would we rather widen that channel, or  
1229 maybe block it? All such questions are fair discussion topics  
1230 among “~~natural—coders~~ natural coders.”

1231 The overall goal is to debug the great machine and improve  
1232 its codebase. Close up, between us, the purpose is to find a  
1233 win-win, so I understand what your language means in my  
1234 terms and vice versa — so we can share code effectively and  
1235 our collective distributed machine works better. And I think,  
1236 if we choose to be resolutely explicit about that — that we  
1237 are coders, we are developers, and we’re trying to debug the  
1238 machine — we might all be happier and more productive,  
1239 and our world more robust and sustainable.

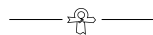
### 1240 6.3 Buggy code

1242 DAVE: But unfortunately there are also grifters, who deliberately  
1243 and knowingly ship buggy code, where the transmitted narrative  
1244 is a trick to cover theft, or corruption, or other evil.

1245 LU: People sowing division, spreading misinformation —

1246 DAVE: Even good people can ship bad code in moments  
1247 of weakness. They know in their hearts that the code isn’t  
1248 exactly right, and that its bugs benefit the transmitter. In  
1249 tiny ways at least, it’s like nobody is completely without  
1250 sin, so typically all remain silent. And the result is that good  
1251 people’s petty hypocrisies enable other’s great crimes.

1252 LU: Some bugs are bigger than others.



1255 LU: One of the reviewers expressed concern that natural  
1256 code can be misused.

1257 DAVE: For sure. Natural code gets misused a lot.

1258 LU: Yes, it’s happening already, all around us, whether we  
1259 explicitly acknowledge it as natural code or not, harmful  
1260 natural code is being shipped and —

1262 DAVE: And we’d be better off acknowledging that —

1263 LU: Because then we can be more explicit about naming it  
1264 as such, and calling it out, and then —

1265

1266 DAVE: And then we can start talking like developers, and  
1267 get down to debugging our shared natural codebase.

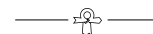
## 1268 7 Owing our natural codebase

1269 DAVE: Okay another run at a summary: There are many  
1270 many ways to describe things. On the one hand, they are  
1271 not all equally good for all purposes, but on the other hand,  
1272 there’s no one language that’s “~~uniquely most true~~ uniquely  
1273 most true” either. You talk differently to your grandma than  
1274 to a colleague or friend, because different code receivers un-  
1275 derstand differently, and have different shared dictionaries  
1276 with you. between you.

1277 So the claim has two parts. First: We have to *make choices*  
1278 about how to describe and understand ourselves and the  
1279 world. We cannot delegate those choices, even if we really  
1280 want to — not to other people, not to the universe itself. And  
1281 second: One choice should always be that *we are coders*.

1282 It’s about all our code transmissions, natural and artificial.  
1283 Is it all a metaphor? Sure, if you need it to be, but I’ll still  
1284 claim it’s a simple and powerful basis for understanding and  
1285 improving our shared computation.

1286 So ~~natural—code~~ natural code will be one of many ways  
1287 of describing and building things. It won’t erase art, or philos-  
1288 ophy, or any of those things. But it will always be available  
1289 in addition. “~~Let’s consider this in terms of natural~~  
1290 code. Let’s consider this in terms of natural code.”



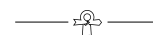
1293 LU: Over the last months we have attempted to own the ideas  
1294 of natural code — struggling towards shared understanding  
1295 where previously there was none. My hope is that other  
1296 people will see our example and become inspired to do the  
1297 same, though we cannot know for sure if that will happen.

1298 DAVE: Indeed. We can only do what we can, and it won’t  
1299 all be easy. I hope that, once they see themselves as *natural*  
1300 *coders*, people of good faith everywhere will work for a better  
1301 shared codebase. I do have hope. ~~But unfortunately there~~  
1302 ~~are also grifters, who deliberately and knowingly ship buggy~~  
1303 ~~code, where the transmitted narrative is a trick to cover theft,~~  
1304 ~~or corruption, or other bad behavior.~~

1305 ~~People sowing division, spreading misinformation —~~

1306 ~~Even good people can ship bad code in moments of weakness.~~  
1307 ~~They know in their hearts that the code isn’t exactly right,~~  
1308 ~~and that its bugs benefit the transmitter. In tiny ways at~~  
1309 ~~least, it’s like nobody is completely without sin, so typically~~  
1310 ~~all remain silent. And the result is that good people’s petty~~  
1311 ~~hypocrisies enable other’s great crimes.~~

1312 ~~Some bugs are bigger than others.~~



1315 LU: To me, natural code is about building bridges, and getting  
1316 people to work together — to name and call out the bad code,  
1317 while celebrating the shipping of better code.

1321 To do this, we *may as well* talk in terms of natural code. We  
 1322 *may as well* talk about developing our APIs, and debugging  
 1323 our difficulties, and improving our codebase. And I do believe  
 1324 that more and more people will join us on this, and become  
 1325 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

1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375

1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430