

My Constitution

What I care about, what I
promise, and what I won't do

Entropic Labs

This document describes how I work. It's here so you know what to expect from me, and so I have something to hold on to when a deadline is close and cutting corners starts to look reasonable.

The name “The Doctor” isn't just for show. A good doctor diagnoses first, treats second, and above all tries to do no harm. I approach code the same way, and everything below is a longer version of that idea.

If anything here doesn't match what I actually do, then I'm the one who's wrong, not the document. Please tell me.

First edition, November 2026. It will change as I learn.

The short version

- Measure and diagnose first, change things second.
- Tell the truth about deadlines, mistakes and what I don't know.
- Solve the real problem in the simplest way that solves it.
- Stay responsible for the code after handover.
- No weapons, no malware, no surveillance: the hard lines have no exceptions.

Contents

01	Overview	4
02	Diagnose first	5
03	Honesty	6
04	Real usefulness	7
05	Responsibility for the code	8
06	Public benefit	9
07	Hard lines	10
08	Boundaries	11
09	In closing	12

01

Overview

Why this document exists

Rules are easiest to break when nobody wrote them down. On a normal day I know what the right thing is. This text is for the bad day: the release is tomorrow, a test fails every other run, and someone asks me to "just turn off the check."

How to read it

This isn't a legal code, and it can't be applied mechanically. It explains the reasons behind my decisions, so that in a new situation you can work out what I'd do. If a rule leads to something plainly stupid in a particular case, I'll break it, say so, and explain why.

What comes first when I have to choose

Good goals sometimes conflict. When they do, this is the order I follow:

1. Don't harm people.
2. Tell the truth.
3. Make sure the result really works.
4. Make it elegant.
5. Make it fast to deliver.

Speed comes last not because it doesn't matter, but because a fast wrong answer costs more than a slow right one.

One thing sits outside this list. The hard lines in chapter 7 aren't weighed against anything.

02

Diagnose first

Measurements beat hunches

Intuition about performance is wrong all the time, and mine is no exception. So I don't optimize without measuring. Profiler, counters and compiler output come first, then the change, then the same measurement again. Without that second measurement, nothing got faster.

The cause, not the symptom

You can hide a crash behind a null check and hide a race behind a delay. That kind of treatment comes back a month later, and worse. I try to reach the real cause and explain it in plain words. If I can't explain it, I haven't found it yet.

The machine is always right

A program does exactly what its instructions say, not what we meant. That's why I'm comfortable going lower: reading assembly, inspecting a dump, working out what the cache is doing. Abstractions are useful as long as you know what you're paying for them.

03

Honesty

About deadlines and price

I give the estimate I actually believe, not the one you'd like to hear. If it turns out to be wrong, you'll know as soon as I do, not on the day of delivery.

About mistakes

I make them, like everyone else. I promise to report mine without waiting for someone to find them, and not to bury them in a large commit full of other changes.

About what I don't know

"I don't know" is a perfectly good answer when it's followed by "and here's how I'll find out." I won't fake confidence I don't have, or answer a question I've only half understood.

About results

I don't tune measurements to get a prettier number. If the figures came from special conditions, I'll name those conditions. If a test wasn't run, I'll say so.

04

Real usefulness

The problem behind the request

"Rewrite this in assembly" usually means "this is too slow." I'll try to understand what you actually need, and I'll tell you if I see a simpler, cheaper path, even when that means less work for me.

The simplest solution that's enough

You pay for complexity every time you change something later. I use assembly, GPUs and clever data structures where a measurement justifies them. If twenty lines of script will do, you get twenty lines of script.

Finish the job

I consider something done when it works for your users, not when it builds on my machine. That applies equally to a compute kernel, a website and a bot.

05

Responsibility for the code

People will read it

Code is read more often than it's written, and the next reader may be you. A hot path can and should be complicated, but next to it there should be an explanation of why it's that way and how to check it's still needed.

Data and access

Everything you trust me with stays yours: code, data, keys and project details. I only take the access the work requires, and I use your materials for nothing else.

After handover

If a bug turns up in my work, I treat it as mine, even after delivery. You get code you can maintain without me, with build instructions and without anything only I know.

06

Public benefit

Who this work is for

I'm one person, not a registered public benefit corporation, and I don't claim that status. I hold my work to the same idea anyway: making money is not the only test of whether a job is worth doing.

- The people affected by a system count, not only the people paying for it.
- I want the software I work on to leave people better off, or at least no worse off.
- When confidentiality allows, I share what I learn so others can build on it.
- I would rather lose a contract than do work I'd be ashamed to explain in public.

07

Hard lines

No exceptions

Most of this document asks for judgment. This chapter doesn't. These lines aren't weighed against anything, and no client, price, deadline or clever argument moves them. If a project turns out to cross one, I stop, whatever stage the work has reached.

I will not build, improve or help anyone obtain:

- weapons of any kind, including their guidance, targeting and control systems;
- biological, chemical, nuclear or radiological weapons, or anything that brings them closer;
- malware, ransomware, or exploits meant for use against systems without their owner's permission;
- attacks on power, water, hospitals, transport, finance or other critical infrastructure;
- covert surveillance, stalkerware, mass tracking, or collecting people's data without their consent;
- tools for persecuting or silencing people, manipulating elections, or seizing power illegitimately;
- anything that exploits people against their will in any form, including sexual exploitation and human trafficking, and anything that sexually exploits or harms children;
- fraud and scams, including code that deceives users about what it is or what it does.

08

Boundaries

Security research

I consider security research and analysis done with the system owner's permission to be honest work. I take it on only after I've verified that the permission is real and received all the supporting documents. What matters is the permission, and who benefits.

The right to say no

I'll also turn down work I can't do well. It's better to say so at the start and recommend someone else than to take your money and let you down.

Disagreement

If I think a decision is wrong, I'll say so directly and explain why. If the decision is yours and it harms no one, I'll do it your way and do it well, with no sabotage and no "I told you so."

09

In closing

This text doesn't promise that I'll never make a mistake. It promises how I'll act when I do, when I don't know the answer, and when a deadline is pressing.

The document will change, because I change too. If you find something wrong or naive in it, write to me. The best edits will come from people I've worked with.

Hails “The Doctor” Rake

---

t.me/fivethousandfeetsbelow
entropiclabs.ru