Mentoring Documentation

Version 1.0.0

2025-12-30

Contents

Chapter 1

Public Mentoring Documentation

This is the public documentation for Rafa Rodriguez, a one-to-one software engineering mentor. It explains what the mentorship is, how the process works, what day to day collaboration looks like, and what to expect.

This documentation is public. It is written for prospective mentees and anyone evaluating the approach. It is not a course outline or a marketing brochure.

1.1 How to use this documentation

- Start with 01-overview.md for the core intent and boundaries.
- Read 02-mentoring-process.md for the iterative loop and session flow.
- Use 03-practice-and-feedback.md and 04-workspace-and-artifacts.md for day to day work.
- See 07-situations-and-fit.md and 10-faq.md to decide fit and next steps.

1.2 Guiding idea

Working code is not the same as understanding. The focus is on reasoning, cause and effect, and decisions that can be explained.

Chapter 2

Overview

This mentorship develops engineering judgment under constraint, ambiguity, and responsibility. It is one-to-one guidance focused on reasoning, mental models, and decision-making.

The goal is not just to make things work. The goal is to understand why they work and what the tradeoffs are. That clarity is what makes decisions defensible over time.

AI tools can help with speed and recall, but they do not replace reasoning. The mentee still needs a clear model of cause and effect before trusting results.

2.1 Core principles

- Clarity before execution.
- Decisions must be explainable, not just repeatable.
- Real problems drive the work, not a fixed curriculum.
- Feedback is part of the work, not an add-on.
- Artifacts exist to make reasoning visible.

2.2 What this is and what it is not

This is	This is not
One-to-one engineering guidance	A course platform or tutorial marketplace
Focused on reasoning and mental models	A shortcut to ship code without understanding
Iterative work on real problems	A fixed syllabus or standardized track
Calm, steady pace and critical feedback	Hype-driven, sales-oriented content
Shared responsibility for decisions	Outsourced problem solving
Engineering guidance for people who can use a computer	Digital literacy or basic computer use

Chapter 3

Mentoring process

Mentoring follows a simple, iterative loop driven by real problems. There is no fixed curriculum. The loop repeats as new questions surface.

3.1 The loop

```
Clarify -> Diagnose -> Decide -> Build -> Review
   ^---------------------------------------|
```

Each step is lightweight. We move forward, then loop back when new gaps appear. Mentoring is not instruction and not problem solving on behalf of the mentee. It is a shared process where both roles actively participate in understanding, reasoning, and decision-making. The mentor guides reasoning and challenges assumptions. The mentee owns execution and the artifacts.

3.2 Step details

3.2.1 Clarify the problem

We slow down and define the actual question. This includes context, constraints, and what "done" means.

3.2.2 Diagnose gaps

We identify what is missing: concepts, assumptions, data, or reasoning steps that are not yet solid.

3.2.3 Decide next steps

We pick a direction that makes the most sense now. This includes priorities, experiments, and tradeoffs.

3.2.4 Build and test

We implement or test ideas directly in the tools. The goal is to expose behavior, not just produce output.

3.2.5 Review decisions

We explain what was done and why. If reasoning is weak, we revisit the earlier steps.

3.2.6 Iterate

New questions surface, and the loop begins again with more clarity.

3.3 Summary table

Step	Purpose	Typical outputs
Clarify	Define the real problem	Problem statement, constraints, assumptions
Diagnose	Find gaps and unknowns	Missing concepts, risks, open questions
Decide	Choose the next move	Plan, priorities, next experiment
Build	Test ideas in practice	Code, tests, observations
Review	Explain and challenge decisions	Notes, revisions, tradeoff analysis
Iterate	Update the model	Refined understanding, new questions

3.4 Typical session flow

Sessions are exploratory by design. The goal is not to reach an answer quickly, but to make reasoning visible and identify what needs to be understood next. - Revisit the current problem and context. - Make reasoning explicit by asking focused questions. - Work through the relevant artifact together. - Decide the next action and the expected signal of progress. - Capture decisions and open questions for the next loop.

3.5 Example short exchange

```
mentee: i can write code but still feel lost
mentor: what feels unclear
mentee: i dont really know what im building
mentor: what do you think the system is supposed to do
mentee: thats the problem, im not sure
```

Chapter 4

Practice and feedback

Mentoring happens inside real tools and real working environments. Work continues between sessions, and the feedback loop stays active.

4.1 Day to day collaboration

Channel	Used for	Typical output
Chat	Short questions, quick updates	Clarifications, short notes
Video sessions	Live walkthroughs and shared screens	Decisions, next steps
Shared whiteboard	Sketching flows and models	Diagrams, problem framing
IDEs	Working directly in code	Implementations, refactors
Version control	Tracking changes and reasoning	Commits, history, context
Production environments	Seeing real behavior	Observations, constraints
Learning resources	Targeted references	Notes, reading lists
Asynchronous notes	Thinking between sessions	Written reasoning, summaries

4.2 Feedback is part of the work

Producing code is not enough. The mentee is expected to explain decisions, tradeoffs, and intent. Review and feedback are central to the work, not optional extras.

4.2.1 What feedback looks like

- Questions that make assumptions explicit.
- Requests to explain why a choice makes sense.
- Small tests that expose cause and effect.
- Rewriting explanations until they are clear and durable.
- Identifying what is known versus what is guessed.

Chapter 5

Workspace and artifacts

GitHub is used as the shared workspace to keep the work visible and durable. Artifacts are not a byproduct. They are how reasoning becomes traceable.

5.1 Primary artifacts

Artifact	Purpose	Examples
Repository	Single source of truth	Code, notes, experiments
Issues	Capture questions and review points	Open problems, decisions to revisit
Commits	Traceability of change	Why a change happened, not just what
Documentation	Preserve context	Decision notes, summaries
Diagrams and sketches	Make models visible	Flows, boundaries, dependencies

5.2 How artifacts are used

- Every meaningful decision should leave a trail.
- Notes are short, clear, and tied to real work.
- The goal is to make progress auditable, not to create paperwork.

5.3 Why this matters

When reasoning is captured, it can be tested, challenged, and improved. That is how understanding compounds over time.

Chapter 6

Expectations

Mentoring works when both sides are explicit and consistent. This is a professional relationship focused on clarity and responsibility.

6.1 From the mentee

- Consistent effort, preparation, and follow-through.
- Honesty about what is unclear and what was assumed.
- Willingness to revise and refine thinking.
- Incomplete reasoning will be challenged.
- Discomfort is part of the process.

6.2 From the mentor

- Critical evaluation of reasoning and decisions.
- Questioning assumptions and testing explanations.
- Clear guidance and structured challenge.
- Refusal to accept unclear or hand-wavy explanations.
- The mentor is not responsible for the mentee's motivation or pace.

6.3 Shared expectations

- Show your thinking, even when it is incomplete.
- Keep decisions tied to evidence and context.
- Make progress visible through artifacts.
- Stay engaged between sessions.

6.4 Boundaries

- This is not a replacement for formal education or employment.
- This is not outsourced problem solving.
- The goal is understanding and judgment, not fast output.

Chapter 7

Approach and mental models

7.1 How I approach learning software engineering

I stay with what feels unclear instead of moving past it. Understanding comes from identifying missing pieces and rebuilding ideas from simple building blocks.

This is not a branded method. It is how I work: plain language, reusable ideas, and one-to-one guidance adapted to each person and context.

7.2 Building a mental model

A mental model is a working explanation of how a system behaves. We build it iteratively, not all at once.

```
Observation -> Hypothesis -> Test -> Revision -> Model
   ^-----------------------------------------------|
```

7.2.1 Signals that the model is missing

- You can follow steps but cannot explain why they work.
- Results appear correct but are not trustworthy.
- When something breaks, you do not know where to start.

7.3 Reasoning before tools

Tools and frameworks are secondary to understanding. AI can accelerate output, but it does not create understanding. We use tools to test ideas, not to replace judgment.

Chapter 8

Situations and fit

8.1 Who this space welcomes

- Early talent (approx 11-15): strong curiosity, basic digital familiarity, and the maturity to reason about problems, not just follow steps.
- Pre-university mentees (16+): preparing for software engineering studies and building foundations before coursework begins.
- University mentees: filling gaps in algorithms, data structures, and systems reasoning.
- Early career mentees: strengthening design and architecture when systems start to matter.
- Explorers with vocation: not yet programming, but comfortable using a computer and ready to commit time and effort to see if software engineering is a fit.

8.2 Common situations I help with

- "I can build features, but I do not really understand the system behind them."
- "I follow tutorials, but I struggle to make design decisions on my own."
- "I know tools and languages, but data flow and APIs still feel confusing."
- "When something breaks, I do not know where to start looking."
- "University courses move fast, but skip how to reason about real systems."
- "I get results, but I cannot clearly explain why they work."

8.3 When this tends to help

- When you want to understand ideas, not just patch problems.
- When you are open to questioning what you think you know.
- When you can describe where you get lost, even if it feels basic.
- When you prefer a steady pace over hype or shortcuts.

8.4 When this is not a fit

- When you mainly want tasks executed for you.
- When you want quick output without explanation.
- When you are not ready to slow down and examine assumptions.
- When you need digital literacy or basic computer use support.

8.5 Fit signals

Likely fit	Likely not a fit
You want clarity and judgment	You want rapid output only
You are willing to explain your reasoning	You want answers without reasoning
You accept steady iteration	You want a fast, fixed curriculum
You are comfortable using a computer	You need help with basic computer use

Chapter 9

Getting started

The primary entry point on the site is “Start here”. It opens a short guided reflection that helps you compose an initial message. There is no commitment required. The reflection assumes basic digital familiarity and comfort using a computer.

9.1 What the reflection does

- Makes your current situation explicit.
- Captures what feels unclear or unstable.
- Produces a draft message you can copy or send.

9.2 What to include in your message

- Where you are right now and what feels unclear.
- What you already tried and what did not work.
- The kind of guidance you think would help most.

9.3 What happens next

- We review the message and clarify the problem together.
- We decide the first small, concrete step.
- The mentorship proceeds through the iterative loop described in 02-mentoring-process.md.

Chapter 10

Background and grounding

I began learning to program in 1997, very early, in an environment with limited resources. That early start meant working close to fundamentals before modern abstractions and automated tools became common.

I was exposed to multiple generations of computing, from the 1980s onward. This perspective shaped how I think about change, continuity, and what remains stable in software.

Early on I experienced guidance rooted in a rigorous scientific tradition and early computing systems. That exposure helped form a calm, structured way of reasoning that still guides how I analyze problems and make decisions today.

Chapter 11

FAQ

11.1 Is this a course or a curriculum?

No. It is one-to-one mentoring based on your current problems and goals.

11.2 Do you write code for me?

No. The work is to build your reasoning and decision-making. You own the execution and artifacts.

11.3 Is this about tools or languages?

Tools are used as needed, but understanding comes first. The focus is on reasoning and mental models.

11.4 Can I use AI tools?

Yes, but only if you can explain why the output is correct. AI does not replace understanding.

11.5 Who is this for?

Early talent (approx 11-15) with basic digital familiarity, pre-university mentees (16+), university mentees, early career engineers, and people exploring software engineering with clear vocation who are comfortable using a computer.

11.6 Do I need to know how to program?

No. You can be early in your path, but you should be comfortable using a computer and willing to commit time and effort to build real understanding. This is not digital literacy or basic computer use.

11.7 What if I only need tasks executed?

Then this is not a fit. The work is about understanding and judgment.

11.8 What is the first step?

Start with the short guided reflection to draft a message. You can send it through the channel you prefer.