Junior *to* Senior.

Patterns from twenty-three years and a thousand interviews.

A TALK BY ROB IVANOV

HELLO.

Rob.

Developer. Team lead. Fractional CTO.
Building and consulting from Bulgaria.
1000+ technical interviews.

23+ years writing code
Multiple roles, multiple stacks
Now: helping founders lead teams

// SECTION 01

What 'senior' actually means.

01

Independent.

You don't need someone to break down every task.

02

Product-thinking.

You ask 'why are we building this?' not just 'how?'

03

Team-lifting.

The team gets better because you're on it.

A senior often understands the request better than the request itself.

this is the move.

// SECTION 02

Executing *to* Deciding.

Don't wait for the title.

act like one.

Act senior. The name follows.

[MONITOR]

OK

Status: 200
All systems green.

[CLIENT]

???

CSS missing.
Page broken.

[ME]

I built a thing.

Visual diff monitor.
Nobody asked.

See a gap. Fill it. Don't wait for permission.

// SECTION 03

Handling ambiguity.

// junior

"Which library should I use?"

Others decide.

// senior

"I looked at three. I think this one fits. Here's why. Can I go ahead?"

Bring proposals, not blank questions.

Documentation.

// SPECS

what 'done' means

scope · acceptance criteria · edge cases · trade-offs
agreed before you write a line of code

future-you will thank you.

// DECISION LOG

what you chose & why

2026-03-12 · chose Postgres over Mongo
why: relational data, team familiarity

// SECTION 04

Communication is technical.

10 *seconds.*

Where am I? What's going on? What do I need?

Have it loaded before you open your mouth.

~~*"What am I learning?"*~~

"Who am I helping learn?"

this is the move.

// junior

~~"3 days."~~

Confident. Probably wrong.

// senior

"3 days if the API behaves. A week if it doesn't. Here's what I'll know by Wednesday."

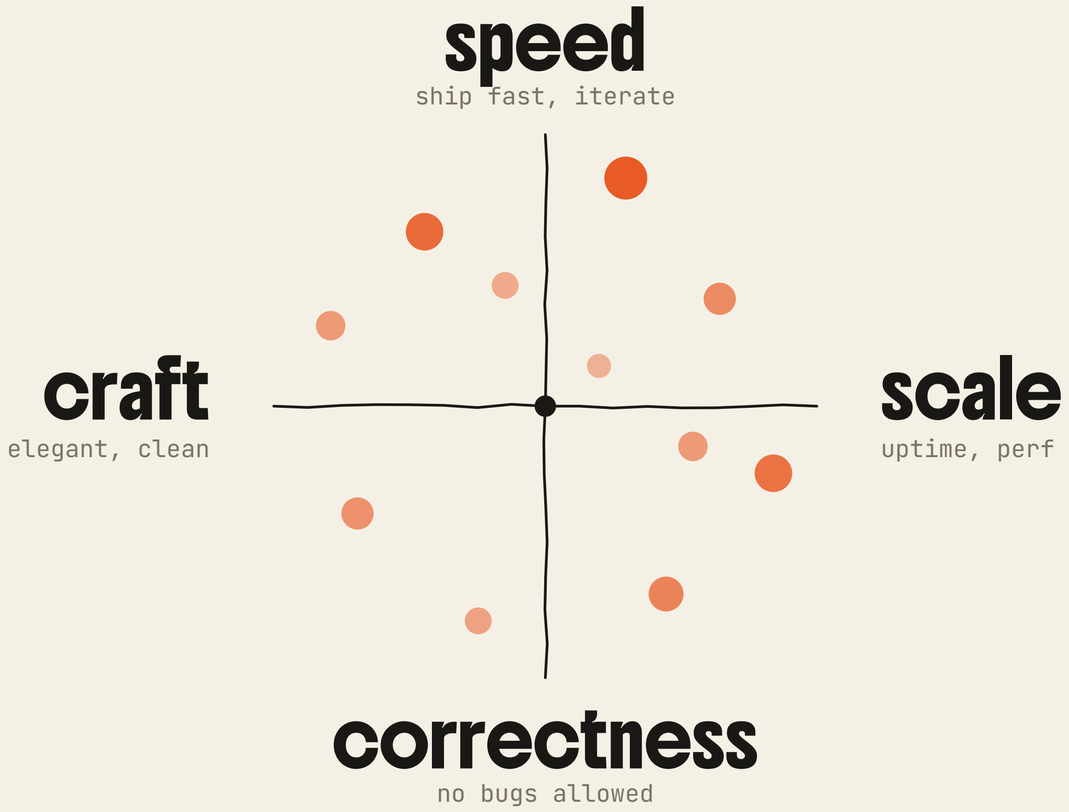
no surprises = trust

// SECTION 05

Code others can live with.

There is no "good code".

What matters depends on where the team is pointed.



Bad code is code that's hard to:

01 **read.**
you can't tell what it does

02 **maintain.**
every change breaks something else

03 **scale.**
doubles in cost when traffic doubles

04 **test.**
you can't isolate it from the world

05 **reuse.**
duplicated three times, drifting apart

06 **reason about.**
too many moving parts at once

you'll know it when you read it.

// SECTION 06

Technical judgment.

Know what's expensive.

// CHEAP

renaming a variable

adding a button

writing more tests

refactoring one file

// EXPENSIVE

a re-render that cascades

a bundle size you can't shrink

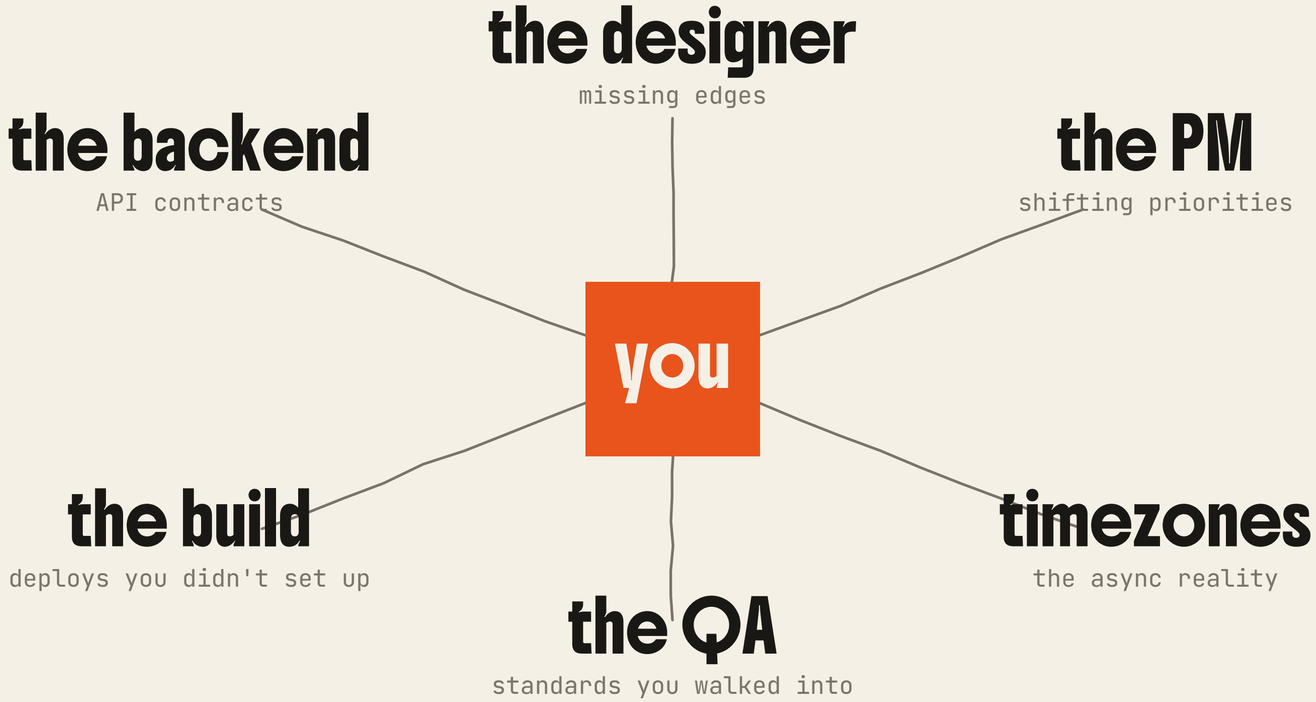
a state shape that locks you in

the abstraction you can't undo

the difference is what you remember.

Your job is the system.

You don't need to own them.
You need to ask them the right questions.



Read more code than you write.

// JUNIOR

writes a lot, reads a little

// SENIOR

reads a lot, writes deliberately

the ratio flips. that's the job.

The trust you build by Tuesday.

01 *you receive a vague ticket*
→ you reply with three options + a recommendation

02 *you spot a problem at standup*
→ you flag it written, with what you'd try first

03 *you finish ahead of estimate*
→ you say so + what you'd pick up next

remote trust is built in writing.

// SECTION 07

AI changes the game. (But not the rules.)

Nobody knows.

// CAMP A

"Nobody will write code in 6 months."

(they said this 6 months ago)

VS.

// CAMP B

"It's all hype. We're in a bubble."

(also not super reliable)

I'm somewhere in the middle. So should you be.

You are not the output.

You are the result of the effort.

— B. Sanderson (paraphrased)

Abstraction layers.

AI is the next one. We've been here before.

AI / Agents	you, today
Frameworks (React, etc.)	everyone, now
High-level langs (Python, JS)	1990s onward
C	1970s onward
Assembly	1950s onward
Machine code	the dawn of time

you are here

PUBLIC SERVICE ANNOUNCEMENT.

Please do not fully vibe code your job away.

your name. your code. your reputation.

```
// it's a slippery slope
```

// SECTION 08

What I saw in 1000+ interviews.

JUNIOR dives into the code.

"I recognize this. Let me start typing."

MID slows down. clarifies.

"What about edge cases? Let me think first."

SENIOR asks why first.

"Who is this for? What's the real constraint?"

I don't know.

But here's how I'd find out.

— every senior candidate, ever

juniors bluffed.
we always saw it.

// SECTION 09

Debugging your growth.

Tomorrow.

Pick one task. Before you write code, write down:

01 ☐ **What 'done' actually means.**

02 ☐ **The biggest unknown.**

03 ☐ **One decision you'll make yourself.**

Three lines. Two minutes. Every task this week.

you'll be working senior before they call you one.

Thank you.

Questions?

rob.top
@robXtop