PUG Challenge 2026 Amsterdam

From Ticket to Merge in Minutes

Building a Modern AI-Powered Development Workflow

Backlog · In Progress · In Review · Done

Vjekoslav Jug · Infodesign · 5 Nov 2026



Intro

1 · ABOUT ME

About me & Infodesign

I’m one of you: working in a company with a real legacy ERP on OpenEdge.

Me

- Vjekoslav Jug
- Business Dev. Manager
- Technical roles
- 23 years with OpenEdge
- Often an early adopter
- AI practitioner

Infodesign

- Croatian ISV company
- OpenEdge-based ERP
- Utilities and public sector
- Manufacturing and retail
- ~20 people
- OpenEdge 10.2B to 12.8

Our reality

- Decades of ABL code
- Desktop GUI, web, Crystal Reports
- Many customer databases
- Exactly the code base people say “AI can't handle”

“Everything I show you today was true this morning.
By tomorrow some of it will be outdated — that is the point of the talk.”

Intro

1 · WHY THIS TALK

This is not a talk about AI. It is a talk about productivity.

The goals do not change. Only the tools do.

The goal

- Productivity – my main goal since I started working
- Ship more of the right thing, with less waste

The tools

- Many tools and techniques changed along the way
- Each one promised the same thing: more done, less effort

AI

- AI is just the **newest player** in that game
- Judge it the same way: does the work get done faster and better?

Goals don't change — only the tools.

April 2026 submission vs November 2026 reality

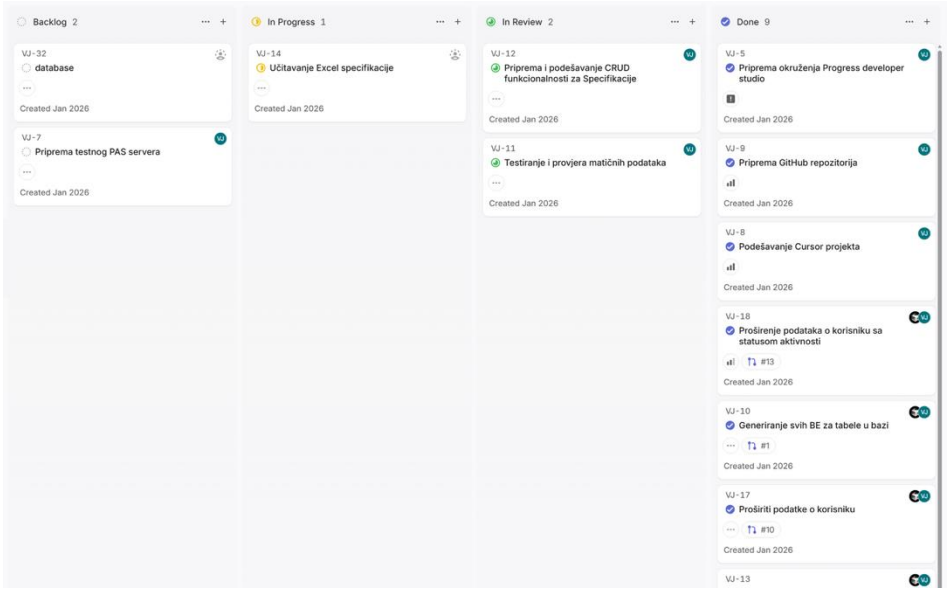
"When I submitted this session in April I thought I would show you Linear and Cursor. Then the tools changed. Three times."



April 2026 — what I submitted

Mostly based on Linear (tasks) → Cursor cloud agent → GitHub PR · Slack for communication

Linear Cursor Slack

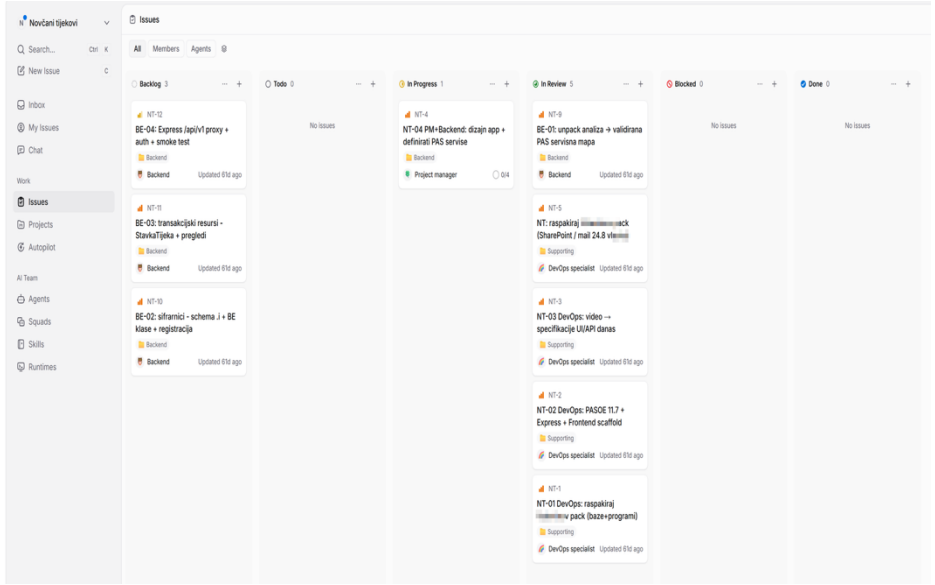


Linear board, project "Specifikacije backend" — tickets worked by the Cursor agent

November 2026 — what I show today

A "company" of bots runs the outer loop · local agents run inner loops on my Mac minis · we approve the gates

Multica Claude Code Grok Build



Multica workspace "Novčani tijekovi" — tasks owned by agent roles (Backend, DevOps, PM)

Intro

1 · THE PLAN

Today: a little theory, then three real ways of working

Timeline → the harness, piece by piece → my LLM chart → three ways I work → what went wrong → OpenEdge → lessons

1 **Then vs now**

Three snapshots: 2025, April 2026, November 2026

2 **Harness, piece by piece**

Model + eight parts: tools, context, memory, skills, guardrails, routines, agents, verification

3 **My LLM chart**

Which models I use for coding — my experience

A **Part A · Synchronous work**

Me + the agent in real time: super.engineering with Claude Code

B **Part B · Shared Kanban**

Humans and agents on one board: Linear ticket → Cursor agent → GitHub PR

C **Part C · Outer loop with bots**

Bots run the loop: sources → board → local coding agents; we approve the gates

Then: what went wrong · OpenEdge specifics · lessons learned · Q&A

Theory first, so the three practical parts make sense; each part shows screenshots from real work.

Then vs now:
one year of AI tools

2025 → April 2026 → November 2026: the tools changed three times.

- **Focus:** prompt → context → harness → loop engineering
- **Way of working:** IntelliSense → tab completion → IDE agents → agentic CLI → loops → graphs
- **Models:** pick the current SOTA, but “don't marry a vendor”



Timeline

THEORY · ONE YEAR OF AI TOOLS

2025 → April 2026 → November 2026

Three snapshots, one year apart — the tools changed three times.

2025 · a year ago

April 2026 · session submitted

November 2026 · now

Focus: prompt engineering

Work: smart IntelliSense → multi-line tab completion; agents inside editors (Cursor, Copilot, Windsurf)

Models: Claude 3.7 Sonnet → Claude 4 → GPT-5; Opus 4.5 in November

Focus: context engineering

Work: Linear ticket → Cursor cloud agent → GitHub PR; agentic CLI tools (Claude Code)

Models: Claude and OpenAI race for the top

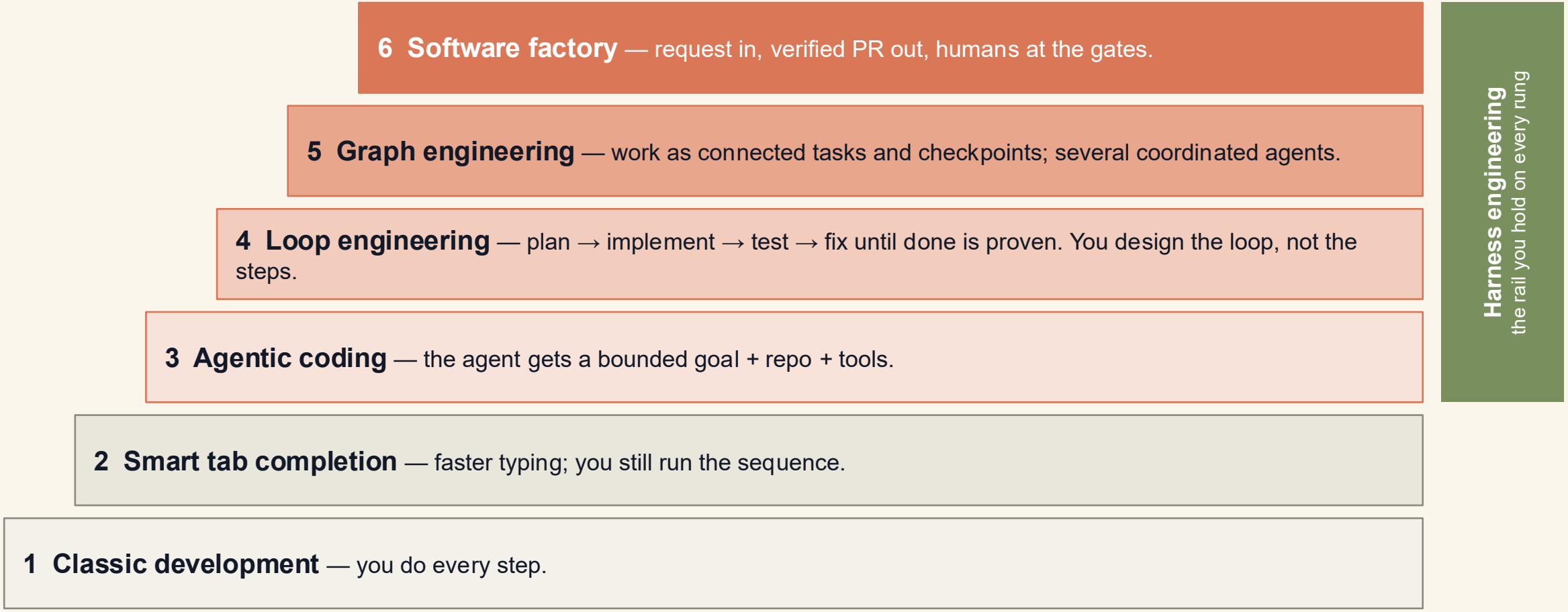
Focus: harness → loop engineering → graph engineering

Work: a company of bots runs the outer loop; local agents run inner loops on my Mac minis; we approve the gates

Models: Claude Opus 5.5-class for coding; SOTA from several vendors

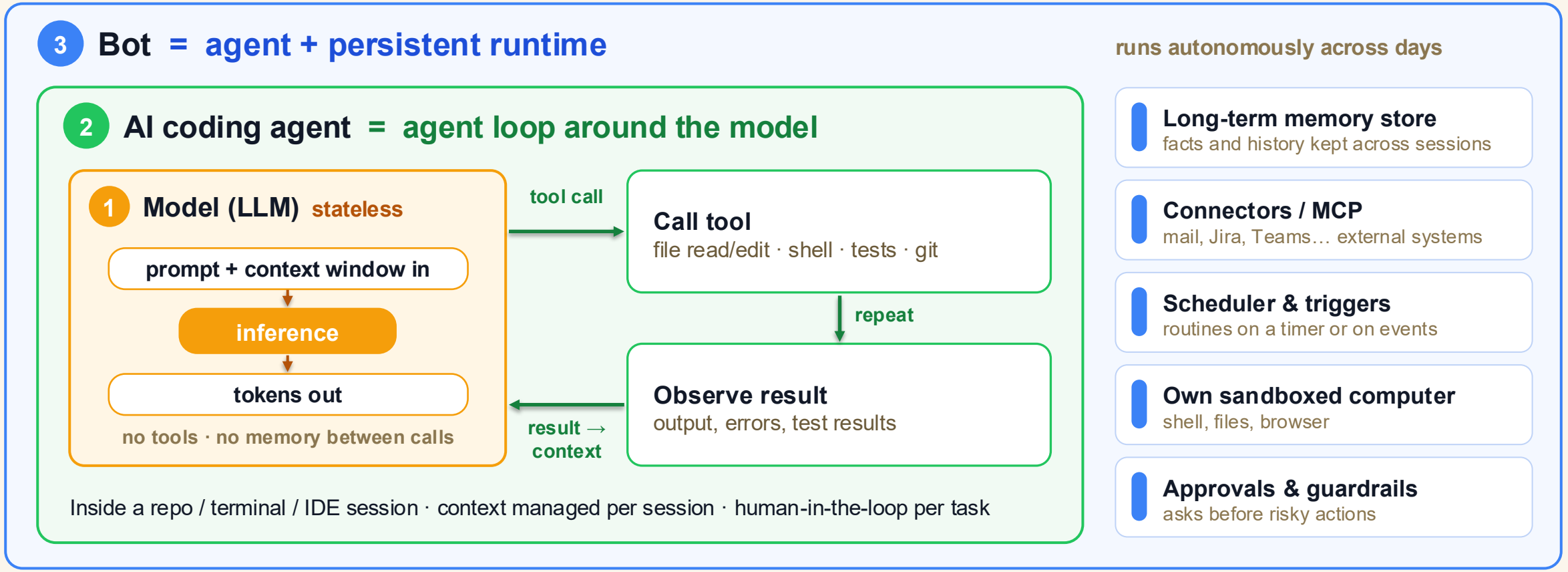
“Many developers I follow say publicly that they have barely written code by hand since late 2025.” (e.g. Karpathy, Jan 2026; Spotify CEO, Feb 2026)

The goal: climbing the ladder to automation



Clarification: model, AI agent, bot runtime

Each layer wraps the one inside it



Model = stateless function · Agent = loop with tools · Bot = always-on runtime

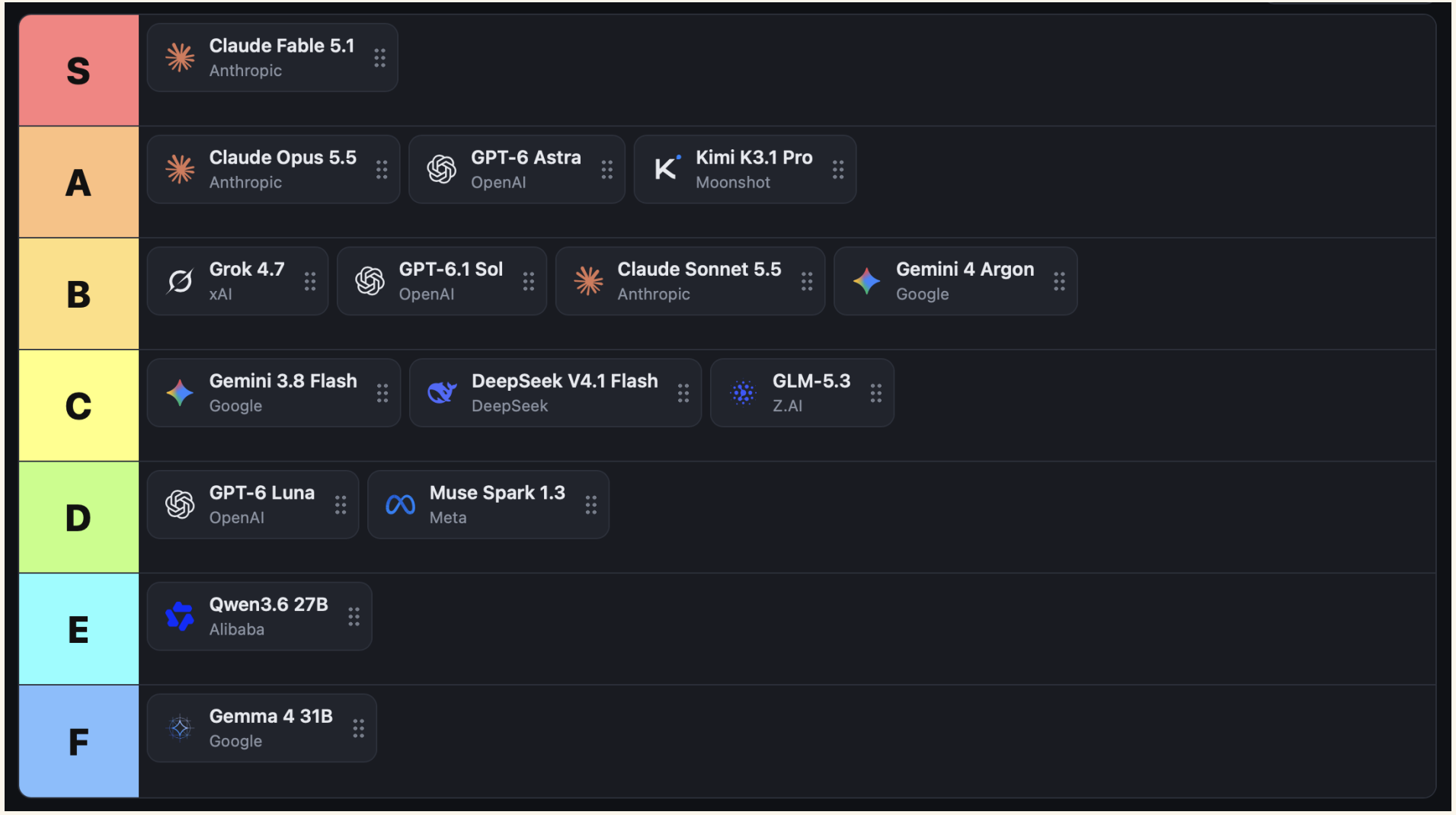
Same three levels — examples from the big vendors

Model → AI coding agent → Bot / assistant (names as of Nov 2026)

	1 Model	2 AI coding agent	3 Bot / assistant
xAI	Grok 4.7	Grok Build	Grok Bot
Anthropic	Claude Opus	Claude Code	Duet
OpenAI	GPT-6 Astra also GPT-6.1 Sol / Terra	Codex CLI · IDE · cloud	dots always-on agents in ChatGPT
Google	Gemini 4 Argon	Google Antigravity agent-first dev platform + CLI	Gemini Spark 24/7 personal agent

xAI and Anthropic names as given · OpenAI and Google names checked against vendor release notes, 3 Nov 2026 · logos: Wikimedia Commons; trademarks belong to their owners

Choosing the right model



My personal experience for coding, Nov 2026 — not a benchmark.

Agent = Model + Harness

"If you're not the model, you're the harness." — the harness is every piece of code, configuration and execution logic around it

Model
the brain: text in → text out

\+

Harness
loop · tools · context · rules · checks

=

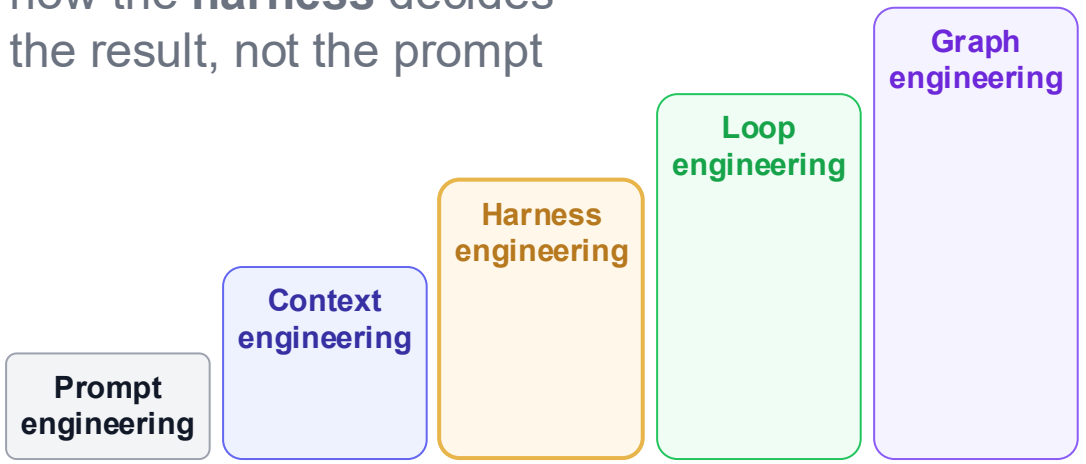
Agent
does real work

A harness includes (LangChain, Mar 2026)

- **System prompts**, rules and conventions
- **Tools, skills, MCPs** and their descriptions
- **Bundled infrastructure:** filesystem, sandbox, browser
- **Orchestration:** subagents, handoffs, model routing
- **Hooks / middleware:** compaction, continuation, lint checks

Focus shift (my outline)

One step every few months: now the **harness** decides the result, not the prompt



Why it matters: same model, different harness = a completely different result

According to LangChain, its coding agent moved from **Top 30 to Top 5 on Terminal Bench 2.0 by changing only the harness**, and Opus 4.6 scores far lower in Claude Code than in other harnesses. That is why, in my setup, the model and the bot are two separate choices.

Sources: langchain.com/blog/the-anatomy-of-an-agent-harness (10 Mar 2026); thoughtworks.com/insights … /ai-agent-harness

What matters most besides the tools

Six things that matter more than which tool is hot this week.

Context is crucial

- The model is only as good as what it sees
- Your code is the context!

Tools: local and MCP

- My take: prefer a **CLI** over MCP where a good CLI exists
- Faster, fewer tokens, scriptable — the agent already knows it

Skills

- Reusable, versioned know-how: “how we do X here”

Rules

- Hard constraints that never change per task

Memory

- What the agent learned about me and my company

… the harness!

- Everything around the model: loops, tools, permissions, gates
- Same model, different harness = a completely different result

Anatomy of a harness: eight parts around one model

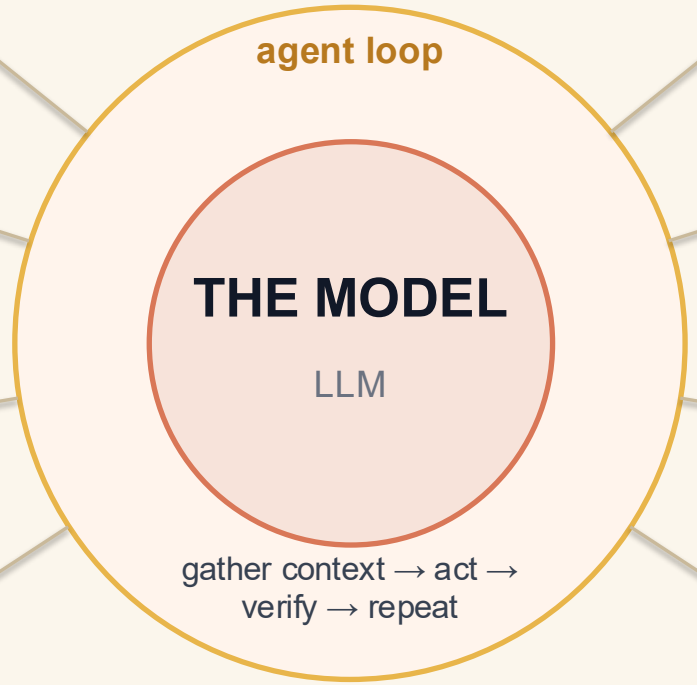
Agent = model + harness. Each part gets its own slide next.

1 **Tools & execution**
the agent's hands

2 **Context management**
what it sees right now

3 **Memory & rules**
what survives the session

4 **Skills**
"how we do X here", on demand



5 **Guardrails & approvals**
what it may not do alone

6 **Scheduling & routines**
work that starts by itself

7 **Multi-agent coordination**
who works with whom

8 **Verification**
proof, not promises

Sources: Anthropic (Claude Agent SDK post); LangChain (Anatomy of an Agent Harness, 10 Mar 2026)

1 · Tools & execution

Four kinds of tools. One definition each — and when to prefer which.

Built-in tools

Come with the agent: read and edit files, shell, search, web fetch, browser / computer use.

e.g. Claude Code: Read, Edit, Bash, WebFetch

Use: always first — no setup

Local CLI tools

Programs on the machine, called through the shell. Agents already know most of them.

e.g. git, gh, docker, m365, the OpenEdge compiler

Use: when a good CLI exists

MCP servers

Model Context Protocol: a server exposes tools and their schemas to any agent — a USB port for tools.

e.g. Linear, GitHub, OpenEdge AI Assistant (MCP) for ABL docs

Use: no CLI, or docs and search behind a login

APIs

Plain HTTP (REST / GraphQL) through curl or an SDK.

e.g. Microsoft Graph for OneDrive uploads

Use: when nothing else fits — wrap it in a script or a skill

My order: built-in → CLI → MCP → API. CLI over MCP where a good CLI exists — faster, fewer tokens, easy to script and log.

2 · Context management

Everything the model sees in this turn — and the window is finite.

Definition

Instructions, the task brief, open files, tool results. What does not fit is dropped or summarised (compaction).

Example

Bad: paste 40 files and a mail thread.

Good: a one-page BRIEF with goal, scope and links; the agent opens files when it needs them.

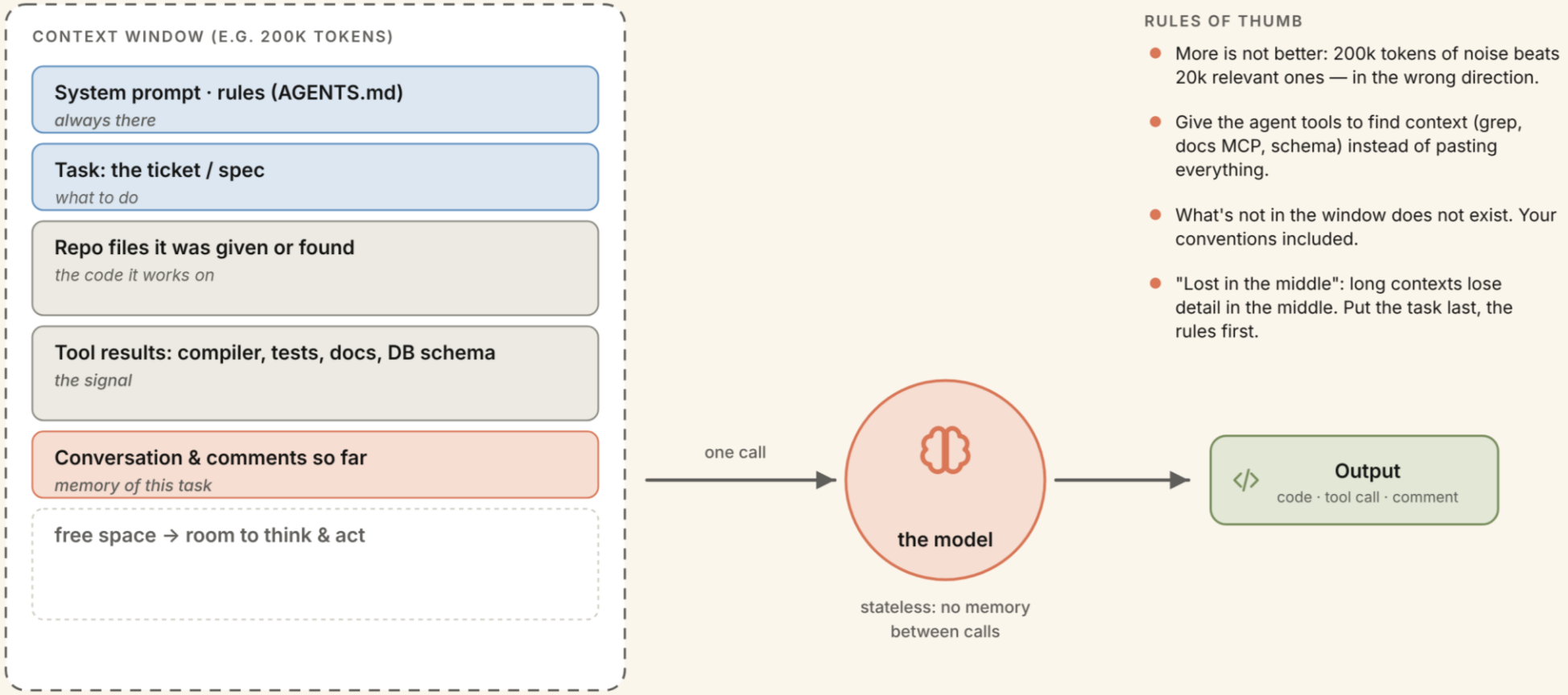
In my setup

- One BRIEF per task — links, not dumps
- A fresh session per phase
- Subagents for side quests, so the main context stays clean

Context is a budget: spend it on the task, not on history.

What the model actually sees

The context window: everything it knows about your task, right now. Finite. Attention degrades.



Context engineering = choosing what goes in and, more importantly, what stays out.

3 · Memory & rules

What survives between sessions.

Definition

Rules are always loaded (AGENTS.md / CLAUDE.md). Memory is recalled when relevant. A progress file keeps the state of one task.

Example

AGENTS.md: “build with X, never touch a production database”

STATUS.md: phase, last step, what is blocked

In my setup

- Shared memory across my bots and AI agents
- A team knowledge base for how-tos and decisions
- Logs and every artifact on OneDrive, linked from the task

Rules short and always on · memory searchable · state in a file, not in the chat.

4 · Skills

Company know-how as code: loaded only when the task needs it.

What a skill is

A folder with SKILL.md: a name, a one-line “when to use me”, steps and scripts.

Only the description sits in the context; the body loads on demand.

How skills work for me

1 · Start from a public skill (Matt Pocock, obra/superpowers, pstack)

2 · Adjust it to how we work

3 · One GitHub repo, synced to every machine

4 · Loaded when the task matches

Examples from my skill library

Ticket spec: SPEC.md by task type before work starts

Crystal Reports: edit .rpt via the SDK, check by screenshot

DB copy: restore locally; only DBs named in the brief

ABL new look: restyle an ABL screen step by step

After every successful new task: “save what was valuable as a skill” — then review and share.

Sources: LangChain (skills as a harness primitive, progressive disclosure); public skill packs by Matt Pocock, obra/superpowers, Lauren Tan (pstack)

Skills: rules that keep them useful

What we learned running skills across bots and AI agents.

Save your own skills

After a verified task, save what worked as a skill: your build, your PROPATH, your traps.

Share them

One skills repo, synced to every machine: every bot and agent loads the same version.

Don't micromanage

Write the goal, the checks and the pitfalls, not every click. The model fills in the steps.

Fewer is better

Too many skills: the wrong one gets picked and the context fills up. Merge overlaps.

Keep them current

A stale skill is worse than none. When a skill fails, fix it right away.

A sharp “when to use”

The one-line description decides whether the skill loads. Make it specific.

Treat skills like code: review them, version them, delete what no longer helps.

5 · Guardrails & approvals

What the agent may do on its own — and where it must stop.

Definition

Sandbox, permissions, allow-lists and approval gates around every action with side effects.

Example

- “git push and merge only after Gate 3”
- “only the databases named in the brief”
- “never on a customer server”

In my setup

- A hard list every brief inherits
- Never send, delete or pay without my OK
- Least privilege: each bot gets only its connectors and computers

The bot decides how — a human decides whether.

6 · Scheduling & routines

Work that starts on a timer or an event — not on a prompt.

Definition

A routine = trigger + skill + where the result goes.

Example

- Every morning: read new tickets and mail → enrich → put them on the board
- Nightly: sync skills to every machine

In my setup

- Routines for intake, checks and reports
- A login check before every agent run (a lesson learned)
- Walk a bot through a task once → save it as a skill → schedule it

Do it twice → make it a skill. Do it daily → make it a routine.

7 · Multi-agent coordination

Several agents with clear roles, handoffs and their own context.

Definition

Agents work in parallel, each with a role and its own context; a shared board holds the state.

Example

Outer-loop bot writes the brief → coder agent builds in its own worktree → validator agent checks each acceptance criterion (never the coder).

In my setup

- A company of bots in the cloud + local coding agents on my Mac minis
- One main bot per machine
- The board (Multica) is the shared state

One owner per task, one folder per task — and a board everyone can see.

8 · Verification

Evidence that the work is done — not the agent's word for it.

Definition

Tests, compilers, linters, screenshots and a second agent that checks the result against the goal.

Example

A validator (never the coder) reviews against the locked DoD; KEY screenshots go on the card; every upload is checked by size > 0.

In my setup

- An independent validator for every task
- Automated review on the PR
- CI is the target — not set up yet

"The command succeeded" is not evidence. A file size is.

Synchronous vs asynchronous work

Async is where the real leverage is; synchronous work only when I want to explore or learn.

(a) Synchronous — me + agent, IDE-like

- Closest to classic development
- Dev agent tools: Claude Code, Codex, Grok Build, opencode, pi, Hermes, OpenClaw
- Tools supporting agents: super.engineering (formerly Superconductor), Orca (macOS, Windows, Linux), Cursor Agent view, …
- Best fit: exploration, hard debugging or learning

(b) Asynchronous — board, notifications, gates

- Assign tasks in the morning, check later or get pinged when done
- Before: Linear + Cursor cloud agent → pinged when the PR is ready
  Now: bots + local coding agents
- Automated reviews: GitHub Copilot review, Claude reviews, CodeRabbit, Graphite
- The goal is to switch all eligible processes to this

Code editors — pardon, code *viewers*

Cursor, Devin Desktop, VS Code, Antigravity, Kiro, … — “I still open an editor every day. Mostly to read.”

Option A ·
Synchronous work

Me next to the agent: explore, debug, learn.

- Using super.engineering + Claude Code
- Still using Cursor for coding and viewing
- Similar to pair programming
- Same process as 2025, just different tools
- Save every valuable lesson as a skill

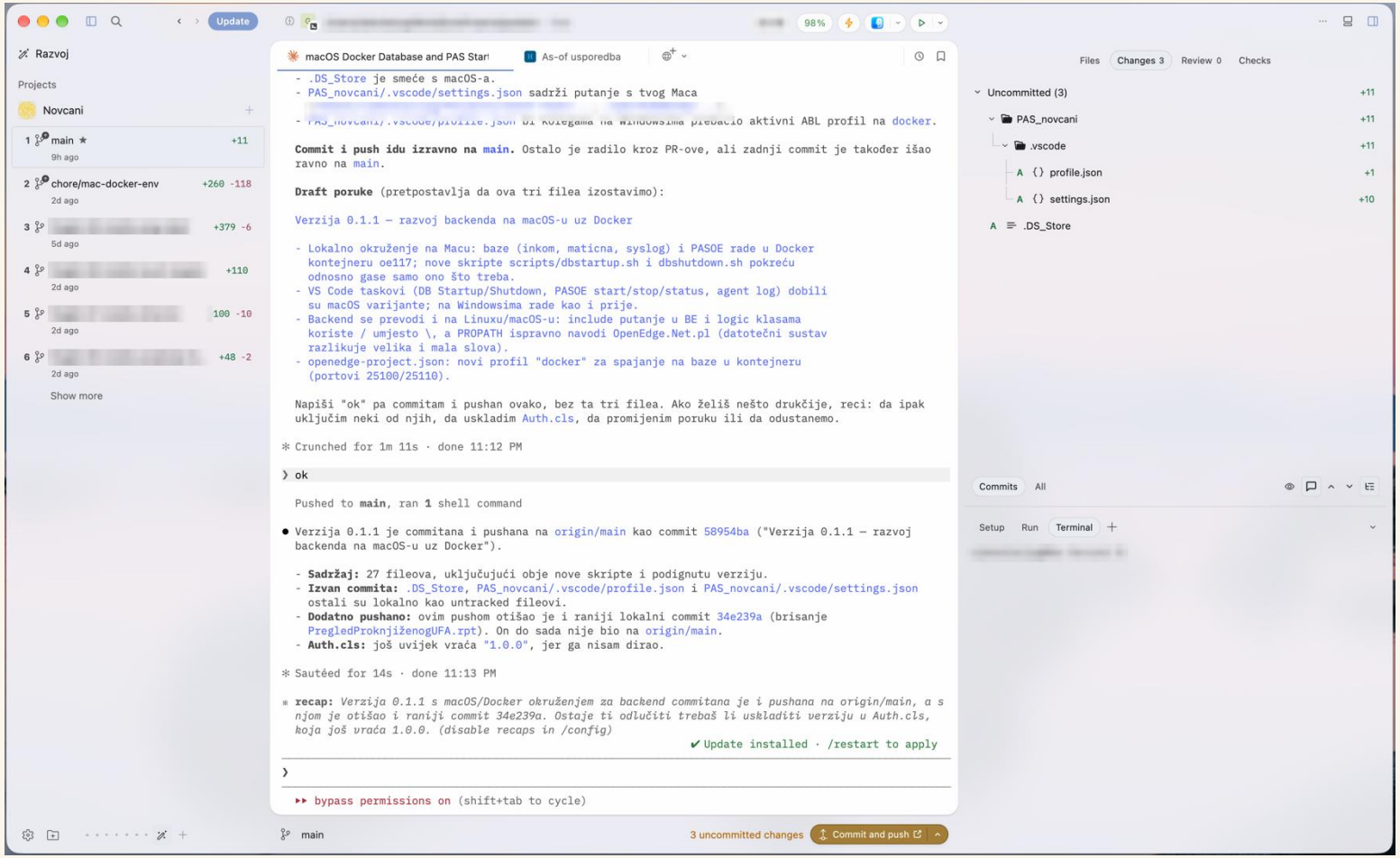


Claude Code inside super.engineering

Synchronous work: I sit with the agent — explore, debug, learn. Then I save what we learned as a skill.

How I work synchronously

- super.engineering is the cockpit: Claude Code runs inside it — one window, many projects and agent chats
- Multiple coding agents: **Claude Code**, Grok Build, Codex, …
- Orca for multiplatform development
- Code viewers (no longer editors): Cursor, VS Code, Devin Desktop
- After a finished task: **save the lesson as a skill**, so the next session starts smarter

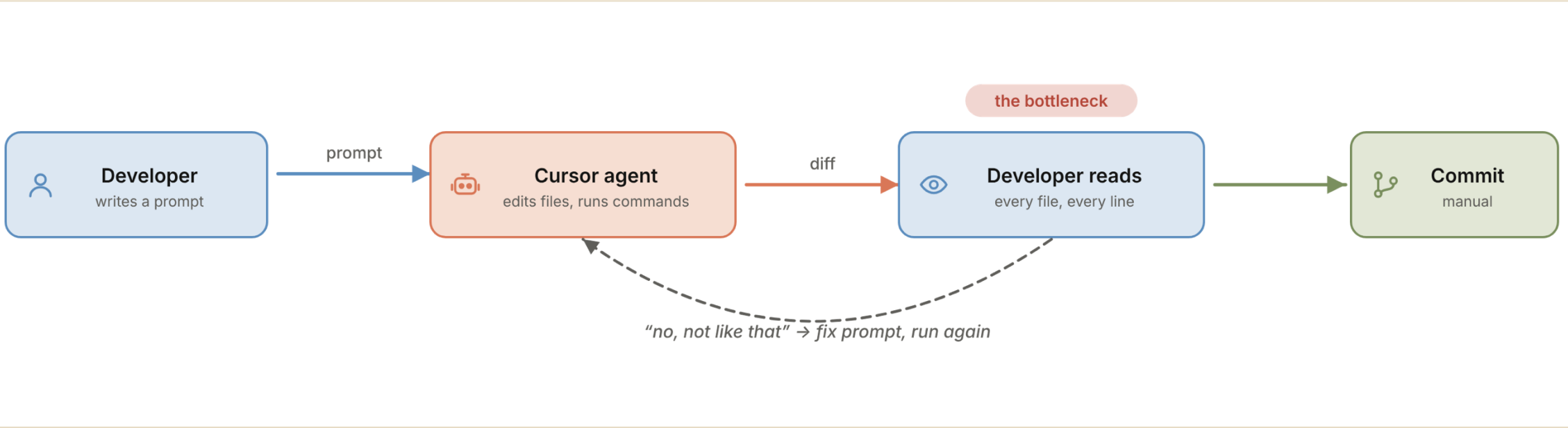


Synchronous

A · SYNCHRONOUS PROGRAMMING

Same way as we did in 2025

Cursor agent window · one task at a time · every file read by a human before commit



No compiler or tests in the loop for ABL — the developer is the signal. Unit of work: a prompt. Verification: my eyes.

Faster than typing. Slower than trusting. Trusting was not an option.

From a verified task to a shared skill

Real task

verified

Proposal

agent proposes a skill

Approval

I approve it (every one)

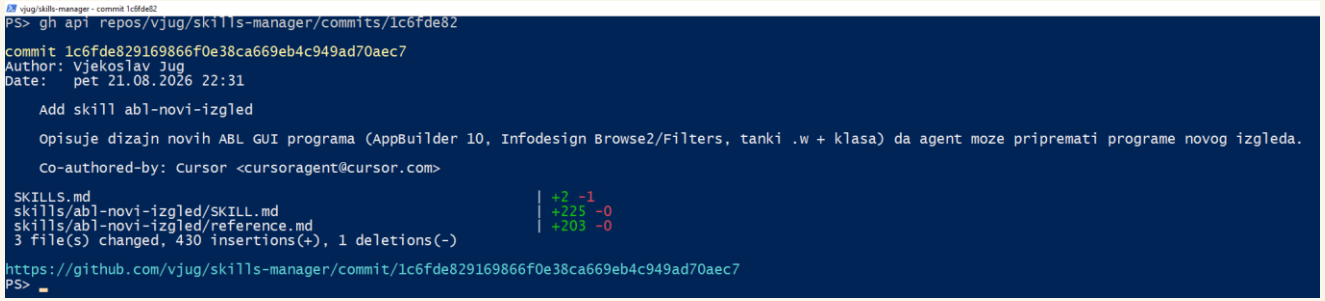
GitHub

published to a central GitHub repo

Sync

synced to the other agents · auto-sync: planned

GitHub gives versioning and distribution. It does not replace verification.



The commit that published a verified skill to my skill repo (vjug/skills-manager, 1c6fde82, 21 Aug 2026)

Option B ·
Shared Kanban

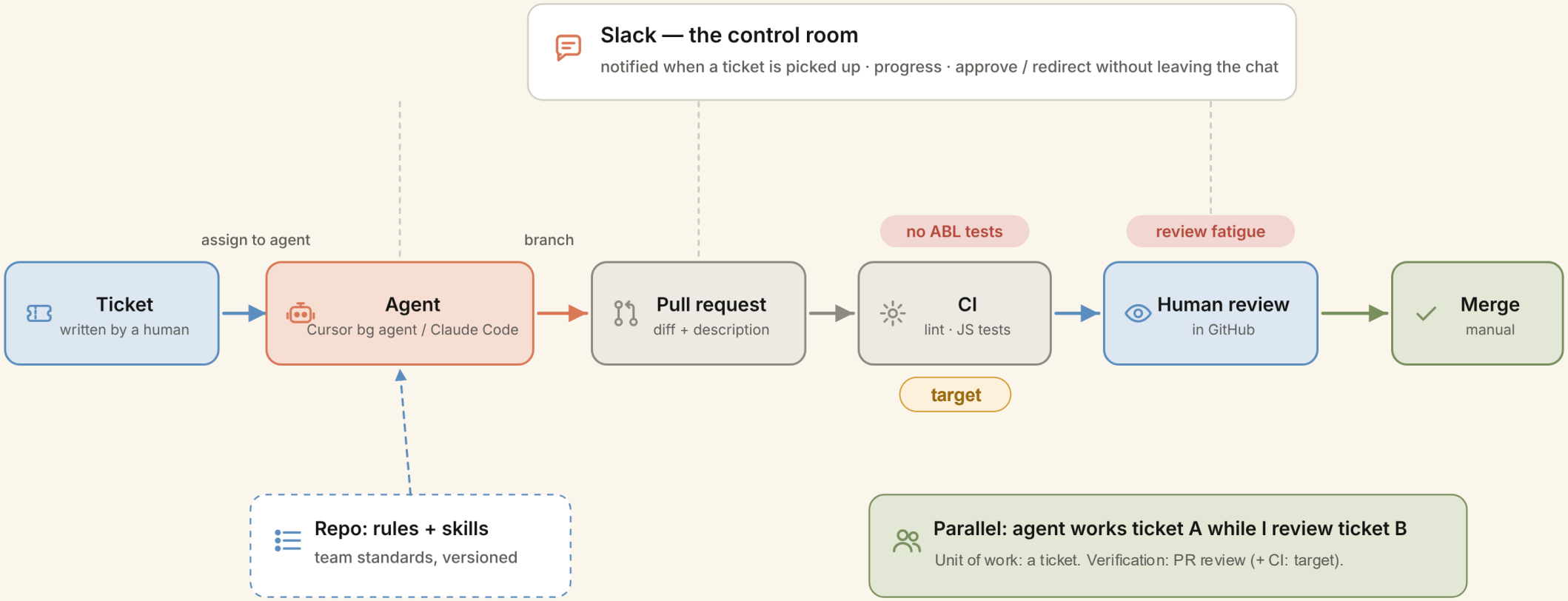
Humans and agents on one board

- I write the task on a Linear ticket
- Assign it to the Cursor agent
- The agent opens a GitHub PR
- I review and merge



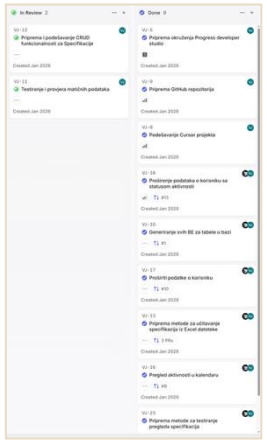
April 2026: I write the task on a ticket

Claude Code for scaffolding, Cursor for daily work · Slack = control room



The ticket description became the prompt. Ticket quality became code quality.

The Shared Kanban path: Linear board + Cursor agent



1

Linear tickets handed to the Cursor agent (VJ-13: 3 PRs)

→



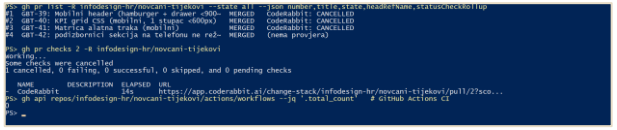
2

branch fix/gbt-43 + commit e9ca344

↓

3

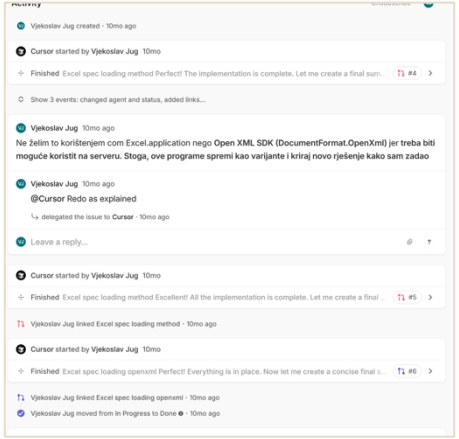
PRs with automated review checks — free-tier reviewer, no CI pipeline yet



←

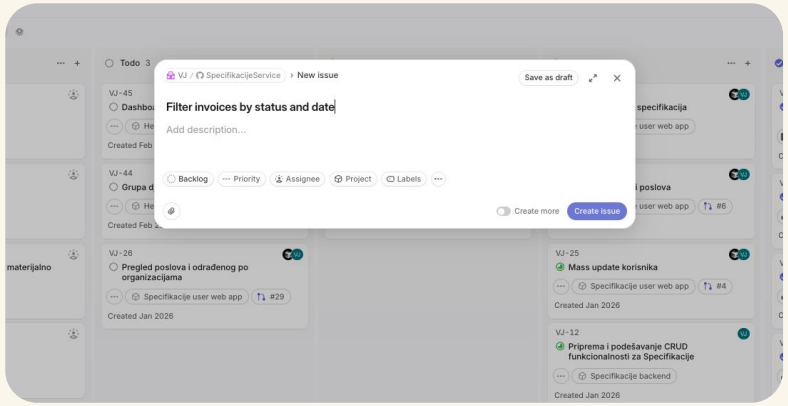
4

my review: “@Cursor Redo as explained” → PRs #5, #6

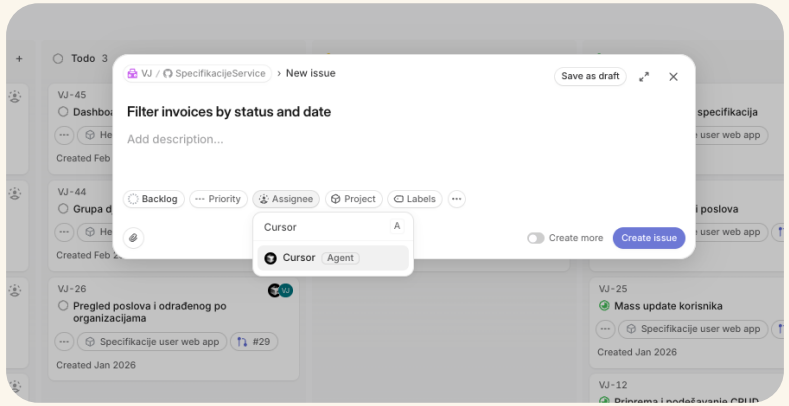


↑ redo → new PRs

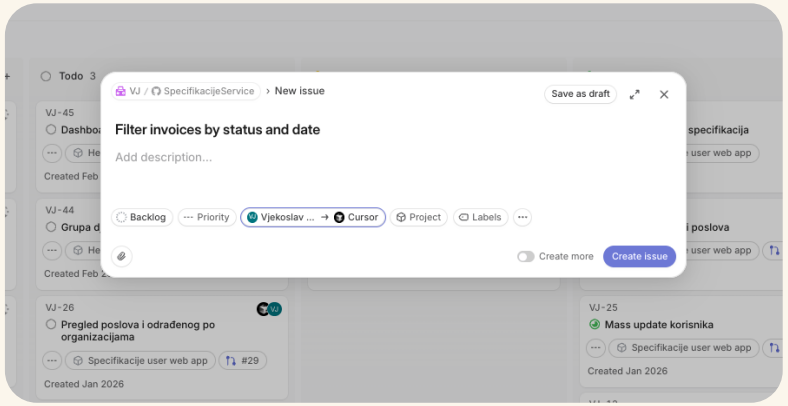
Open a task, then hand it to Cursor



1 New issue

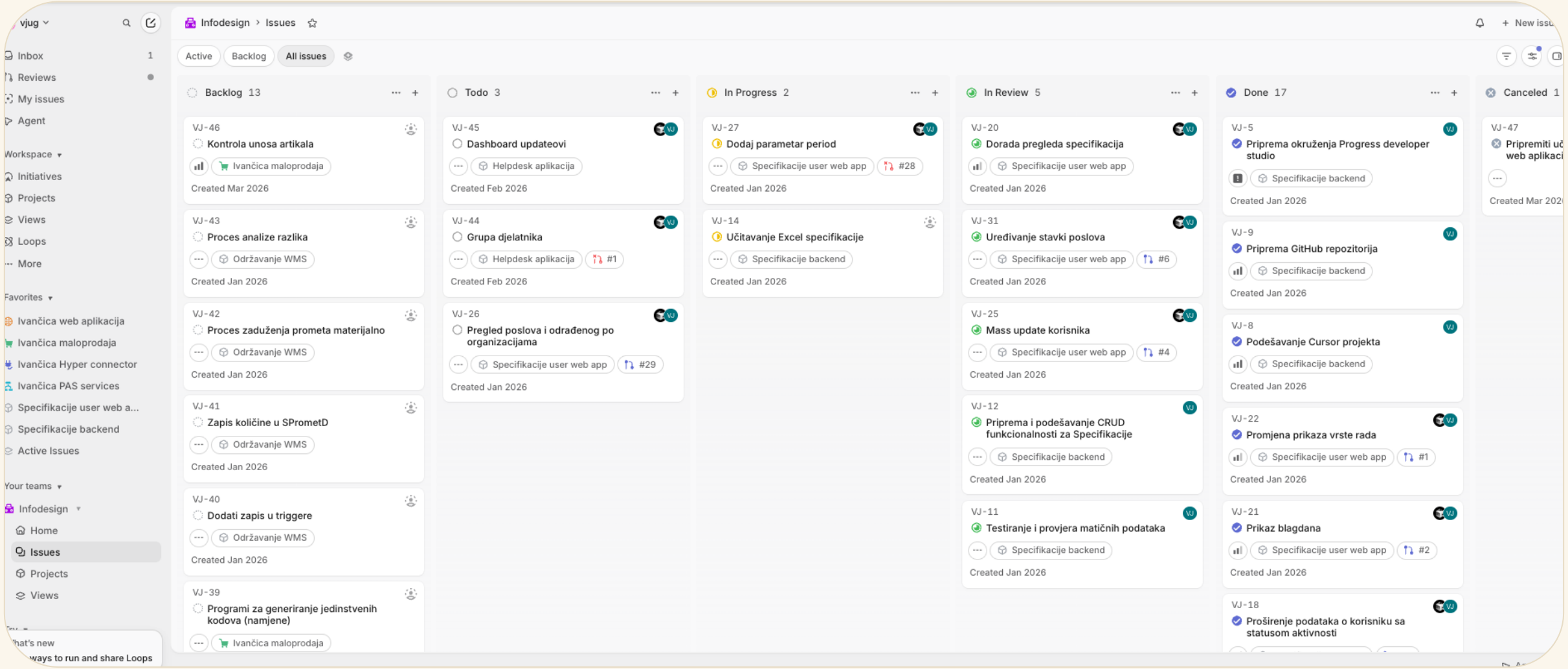


2 Assign to the Cursor agent



3 Delegated to the Cursor agent

Status moves across the board



Backlog
New task

→

Todo
Assigned

→

In Progress
Cursor working

→

In Review
PR + Copilot

→

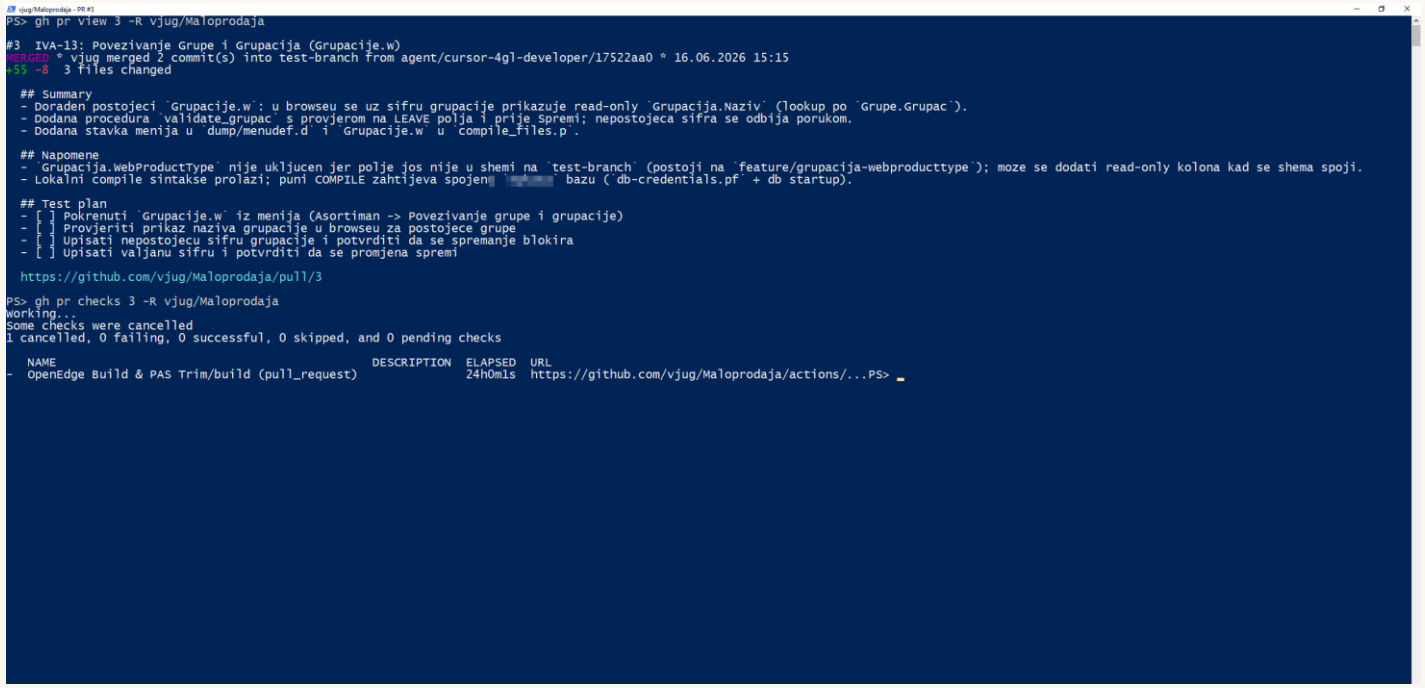
Done
Merged

Synchronous / async

4 · THE CATCH

The merge-button problem

Being asked to merge feels like control … but over time you just click “merge”, and the agents get overconfident.



A real PR, DB name blurred — “Did anyone actually look?”

What happens over time

- Five green checks, an automated review, a confident agent
- You stop reading — and just click merge
- Agents get overconfident — and so do you

“Clicking merge without looking is not automation, it is gambling.”

So how do you automate AND keep control? → section 7

What broke with Shared Kanban

Tickets written for humans

→ the agent didn't "know what I meant".

PR review fatigue

Agent PRs became a new full-time job.

"Tests pass" meant JS tests pass

— no ABL.

Missing context

DB schema, naming, source of truth.

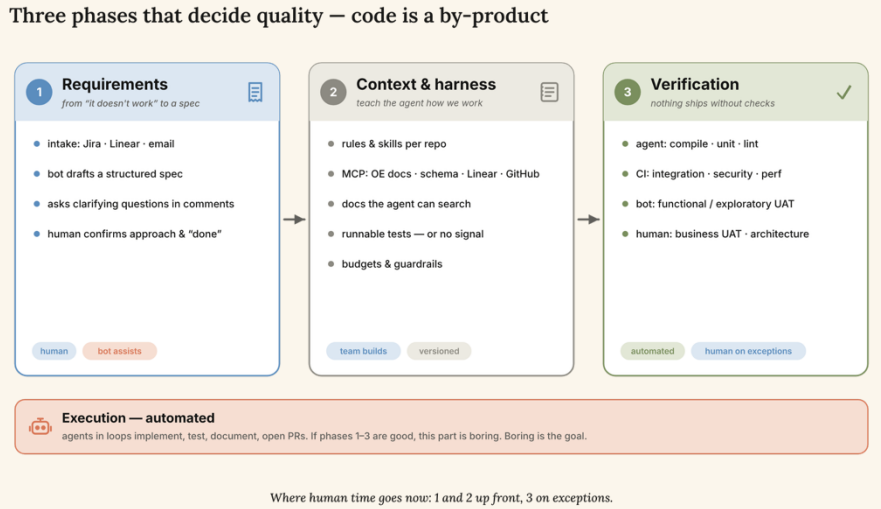
I replaced writing code with reading code, and discovered I like reading code even less.

Lessons from Shared Kanban → the three phases

1 **Requirements detailed enough for a stranger.**

2 **Context and harness must include what's in senior developers' heads (and the compiler).**

3 **Verification is automated — or it doesn't exist.**



Preview: the full version comes in Part C (slides 55–69).

Option C ·

Bots running the outer loop

The bots run the process. It changes every day — literally.

- Sources: Jira, Slack, Teams, mail
- A bot runs the outer loop
- Board, docs, logs, archive
- Coding agents on their own computers



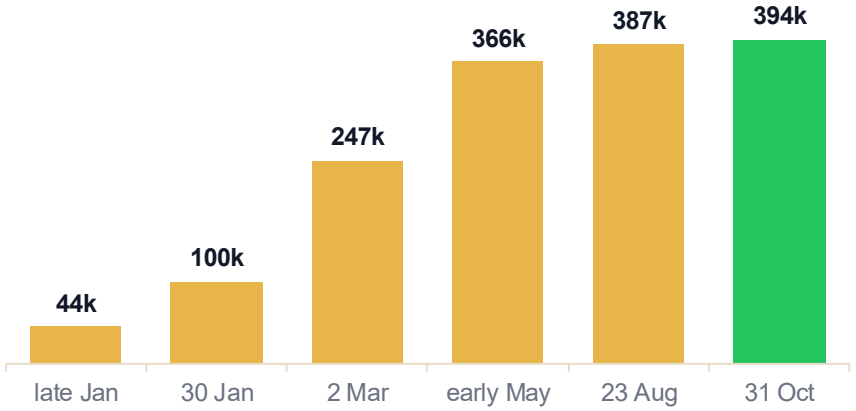
OpenClaw: the game changer — and the hype

An open-source personal agent you talk to from your chat apps; it went viral in January 2026

Key point: GitHub popularity

OpenClaw

393,753 ★



- 100k+ stars in ~2 months, passed React (~250k) in early March 2026

What it brought

- **Channels:** WhatsApp, Telegram, Slack, Discord, iMessage, Teams + 20 more
- **Scheduled tasks** and a gateway that keeps running
- **Computer use**, browser and **local tools**
- **Skills** as folders with a SKILL.md
- Started as Warelay (24 Nov 2025) → Clawdbot → Moltbot → OpenClaw (announced 29 Jan 2026)

My experience

- Huge hype
- Marketed with local LLMs → acted dumb, ignored basic instructions
- Buggy, many issues
- Took a lot of time and patience … then **I moved on**

Unique: local-first, on your own hardware

State, memory and credentials live on your machine; models (Claude, Codex, local) are swappable plug-ins. Run by the non-profit OpenClaw Foundation — no paid tier, no hosted service.



Sources: openclaw.ai blog (29 Jan 2026); TechCrunch 27 Jan 2026; OpenClaw README

Hermes Agent: what OpenClaw should have been

Open-source agent by Nous Research — same idea, far more stable

Facts

- Built by **Nous Research**, the lab behind the Hermes models
- Public on GitHub in **early March 2026**; first release **v2026.3.12** (12 Mar 2026)
- **Open source, MIT**
- **Any model:** Nous Portal, OpenRouter, OpenAI, Anthropic, local endpoints …
- **20+ channels** from one gateway: Telegram, Discord, Slack, WhatsApp, Signal, Email, Teams …
- Built-in **cron** scheduler, subagents, MCP
- Runs local, Docker, SSH, Modal, Daytona — from a $5 VPS to a GPU cluster
- 259,863 ★ on GitHub (3 Nov 2026)



HERMES-AGENT

Hermes Agent app icon and wordmark (official repo, MIT)

My experience

- **Very stable**, actually usable
- **Follows instructions much better**
- Weakness: the agent is **only as good as the underlying model**

Unique: a built-in learning loop (Nous Research's claim)

After complex tasks it **writes its own skills**, improves them while using them, searches its past conversations and builds a model of you across sessions.

Sources: github.com/NousResearch/hermes-agent (README, releases, GitHub API 3 Nov 2026); hermes-agent.nousresearch.com/docs; newclawtimes.com

Grok Bot

5 · AUGUST 2026 → TODAY

Grok Bot: the next game changer



A smart bot that never stops working — on its own computer in the cloud

Facts (official docs)

- By SpaceXSI (formerly xAI); beta since 11 Aug 2026
- Each Bot works on a **persistent cloud computer** with browser, filesystem and terminal — keeps going when the laptop is closed
- **Setup is a message**, not a workflow builder
- **Type, dictate or voice chat**; desktop (macOS, Windows, Linux) and iOS / Android
- Routes each task to the best model: **xAI or third-party** (e.g. Opus 5.5)
- Bots **coordinate in group chats**, learn workflows from demonstration → **skills and routines**

How I use it

- Runs my **outer loop**
- **Streamlines communication** with people
- **Explores and prepares documents** (this deck, too)
- **Specialised security agents**
- Bot = employee: SOTA models + general knowledge, but on day one it didn't know how I work → skills



My fleet: four Mac minis and two Windows PCs (+ Docker) — Claude Code, Grok Build, Codex, Cursor Agent and gh on almost every host

Unique: zero setup + its own always-on computer → a whole "company" of bots

My bots are specialised and trained, each with its own purpose and mission; they **share memory, skills and one cloud computer**, use browser, MCP and CLI tools, and use the tools I allow on my Mac minis.

Sources: x.ai/news/introducing-grok-bot; docs.x.ai/grok-bot/overview; cursor.com/help/grok-bot/models; VentureBeat 11 Aug 2026; photo Seasider53, CC BY 4.0

Grok Bot 5 · TODAY



Grok Bot is my main choice — alternatives in reserve

Aug–Oct 2026: every big lab shipped its own personal agent

Meta Muse
Sep 2026

OpenAI dots
Sep 2026

Claude Duet
Oct 2026



Grok Bot

- Own **always-on cloud computer**
- **Zero setup**
- **Best model per task**, not just Grok
- A **team of bots** with shared skills
- My main choice today

The alternatives (verified, autumn 2026)

Meta Muse · 8 Sep 2026

Personal agent “for everyone”, powered by Muse Spark; runs on Muse Secure VM with a separate Sentinel agent that approves outbound actions; Muse app and WhatsApp; US first.

OpenAI dots · 29 Sep 2026

Always-on agents in ChatGPT, powered by GPT-6 Astra; own cloud computer, 4,000+ apps; reachable in ChatGPT, Slack and Teams; Pro and Business Premium.

Claude Duet · Oct 2026, research preview

Proactive agent app built around Claude Opus 5.5: morning brief, checks connected apps, asks before it sends, pays, signs or deletes; US waitlist (Pro, Max, Ultra).

Same pattern everywhere: own cloud computer, works while you are away, asks before consequential actions.

Sources: about.fb.com (Muse, 8 Sep 2026); openai.com/index/introducing-dots; claude.com/resources/articles/claude-duet; CNBC 21 Oct 2026; logos Wikimedia Commons (PD) / Claude CC0

OpenClaw Hermes Agent Grok Bot SUMMARY

OpenClaw vs Hermes Agent vs Grok Bot



	OpenClaw	Hermes Agent	Grok Bot
Maker · start	Peter Steinberger → OpenClaw Foundation · Nov 2025, viral Jan 2026	Nous Research · public March 2026	SpaceXSI (formerly xAI) · beta 11 Aug 2026
Licence	Open source, MIT	Open source, MIT	Commercial, included in subscription plans
Setup	Self-hosted install and configuration	Self-hosted install (CLI, desktop app)	**“Setup is a message” — zero setup**
Where it runs	Your own hardware (gateway)	Local, Docker, SSH, Modal, Daytona …	Persistent cloud computer (browser, filesystem, terminal)
Models	Any — Claude, Codex, local models as plug-ins	Any provider — Nous Portal, OpenRouter, OpenAI, local …	Auto-routed per task: xAI + third-party (e.g. Opus 5.5)
Channels	WhatsApp, Telegram, Slack, Discord, iMessage, Teams + 20 more	Telegram, Discord, Slack, WhatsApp, Signal, Email … 20+	Own desktop and mobile apps, voice chat, @bot on X
Stability (my experience)	Buggy; needed a lot of patience	Very stable; follows instructions much better	Smart, never stops working
Unique trait	Local-first; fastest GitHub star growth ever	Built-in learning loop (Nous Research’s claim)	Own always-on computer + team of coordinating bots
GitHub stars (3 Nov 2026)	393,753	259,863	not defined (no public repo found)

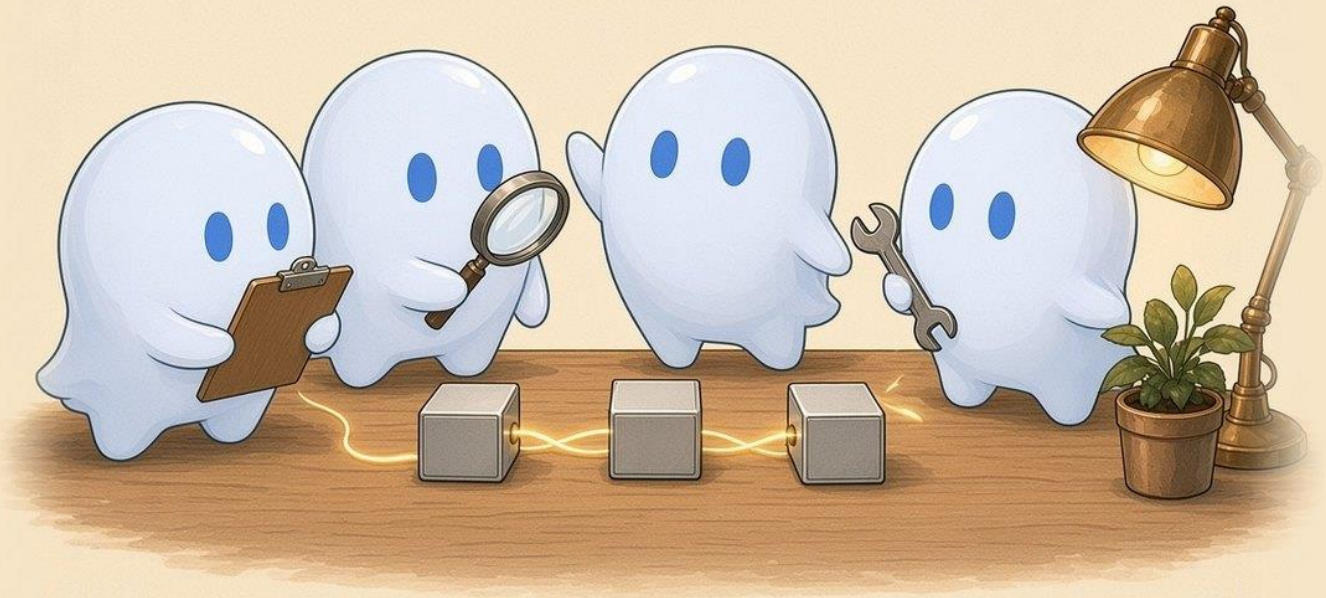
Sources: GitHub API 3 Nov 2026; OpenClaw and Hermes READMEs/docs; docs.x.ai/grok-bot; cursor.com/help/grok-bot/models

My setup:

a company of bots

Constants: Git/GitHub, Jira, Slack/Teams, OneDrive.

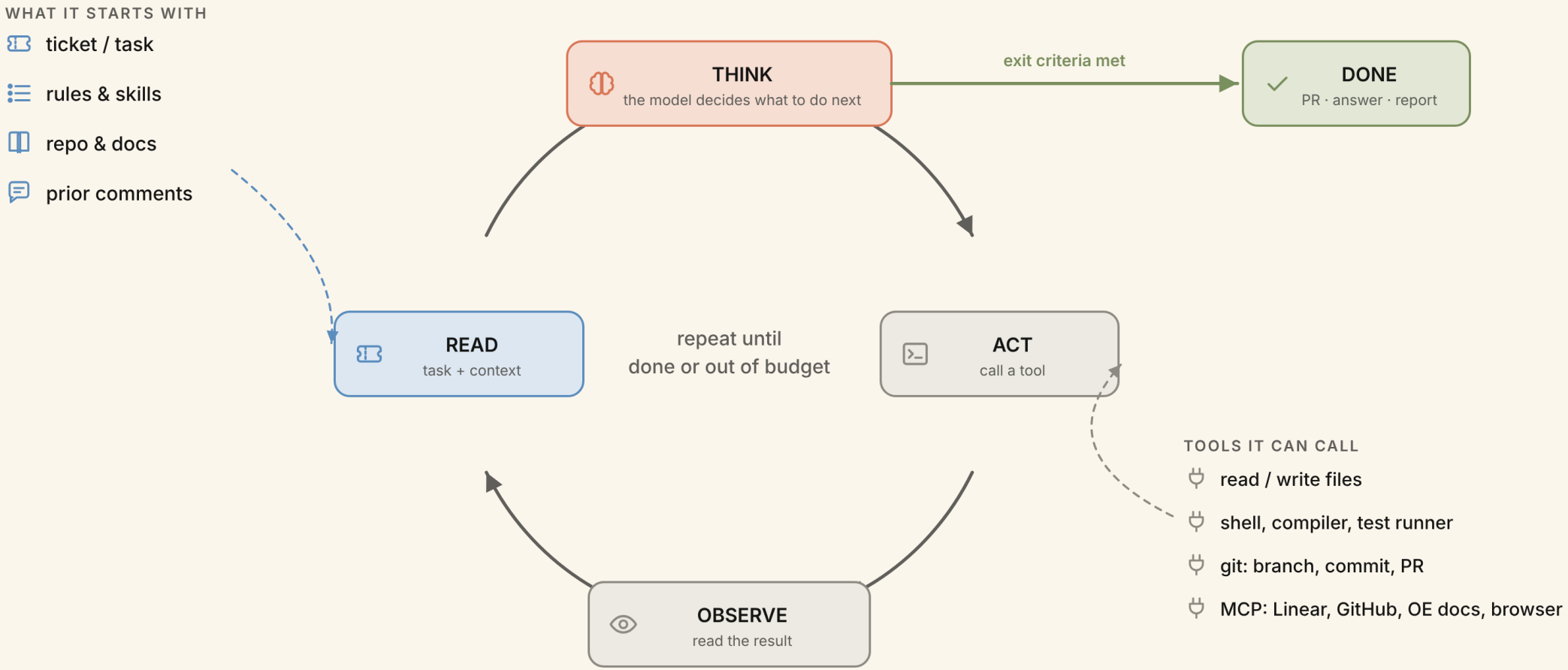
- **Outer loop:** bots plan, supervise, verify, communicate
- **Inner loop:** local agents code and test on my machines
- **Model ≠ harness:** Claude codes, a Grok bot runs the outer loop



Inner loop

C · CONCEPT

Agent = model + tools + loop



A tool is anything with a text interface. The compiler is a tool. _progres -b is a tool.

Outer loop Inner loop C · CONCEPT

Outer loop and inner loop: two harnesses, one job



Grok Bot plans, supervises and verifies; local coding agents do the hands-on work on my own machines

OUTER LOOP · Grok Bot team

- **Persistent cloud computer** — browser, filesystem, terminal; keeps working when my laptop is closed
- **Setup is a message**; memory and context compound per bot
- **Skills and routines:** learned once, rerun on a schedule
- Bots **message each other** and hand over tasks
- **Comes back when something needs my approval**
- How I use it: it takes the task (even a spoken one), enriches it, sets the goal, **launches and supervises** local agents, verifies, reports

INNER LOOPS · local coding agents

- **Claude Code** — the harness Anthropic also ships as the Claude Agent SDK
  - design in **plan mode**, fix in its **own git worktree**, tests until green, draft PR, squash merge after Gate 3
- **Grok Build** — recon / as-is for every task, reviews each acceptance criterion against the DoD (never its own code), writes evidence
- **Codex**, Cursor Agent — fallbacks
- Run on my own machines: four Mac minis and two Windows PCs (+ Docker), reached over Tailscale — never on a customer server

Handoff outer → inner: a brief with goal, scope, acceptance criteria and the hard list. No brief, no launch.

Sources: docs.x.ai/grok-bot/overview; Anthropic (Claude Agent SDK post); martinfowler.com harness engineering (2 Apr 2026); logos: Wikimedia, OpenAI

The new hire: brilliant, well-read — and new here

The model is the education. Everything about us still has to be taught.



Day one: badge, folder, keys — and no idea how we work.

What it brings: the LLM

- Trained on a huge share of public text and code
- Knows SQL, Vue, Git, English, Croatian…
- ABL not best — but it has the OpenEdge AI Assistant (MCP)
- Reads and writes fast, never tired
- SOTA model = the smartest graduate you ever hired

What it doesn't know on day one

- Our company, code base, customers
- Our conventions, PROPATH, databases, who to ask
- Anything after its knowledge cutoff
- No implicit context: not written down = doesn't exist

“Bot = employee: SOTA models + general knowledge, but on day one it didn’t know how I work → skills.”

Onboarding: role, access, skills, knowledge

Everything a good manager gives a new colleague in week one — but written down.

1 · Role & job description

- Bot profile: purpose, mission, persona
- Boundaries: what it never does
- A named human owner: who it reports to

2 · Access — least privilege

- Its own identity and accounts, never shared credentials
- Tools: CLI first, MCP where needed
- Only the computers it needs; one primary bot per machine
- Read by default, write only where needed

3 · Skills library

- Company know-how as reusable skills: “how we do X here”
- Procedures, pitfalls, conventions
- Versioned in a GitHub repo, synced to every bot and host
- Public skill packs, adapted to us

4 · Memory & knowledge base

- Shared memory: what it learned about me and the company
- Knowledge base: wiki / markdown docs, specs, past tickets
- Archive: every deliverable on OneDrive, linked

Onboarding is **written, not told**. Bonus: agent rules are the first onboarding documents in company history that are actually read by their audience.

Sources: CIO, 15 Oct 2026 (cio.com/article/4236060); VE3, 15 Oct 2026; OSIbeyond, 13 Jul 2026; Microsoft Learn, 14 Jul 2026

Todo In Progress 6 · A BOT IS LIKE A NEW EMPLOYEE

Goals and clear tasks

A smart new hire with a vague task produces a confident, wrong result. So: goal, brief, owner, rules.

Objective

- What problem, for whom, why now
- Measurable expected behaviour

Definition of done

- Versioned DoD block: numbered acceptance criteria (Given / When / Then)
- Evidence required: screenshot, test data, log
- Done = merged, never “almost done”

A clear brief — no brief, no launch

- ✓ Goal and source of the request
- ✓ Context: docs, tickets, examples
- ✓ Scope — and what NOT to touch
- ✓ Deliverable + evidence
- ✓ Deadline / time box
- ✓ Skills to use, agent + fallbacks
- ✓ House rules, copied verbatim

One task, one owner

- One ticket on the board, one assignee
- One working folder per task
- Status always visible

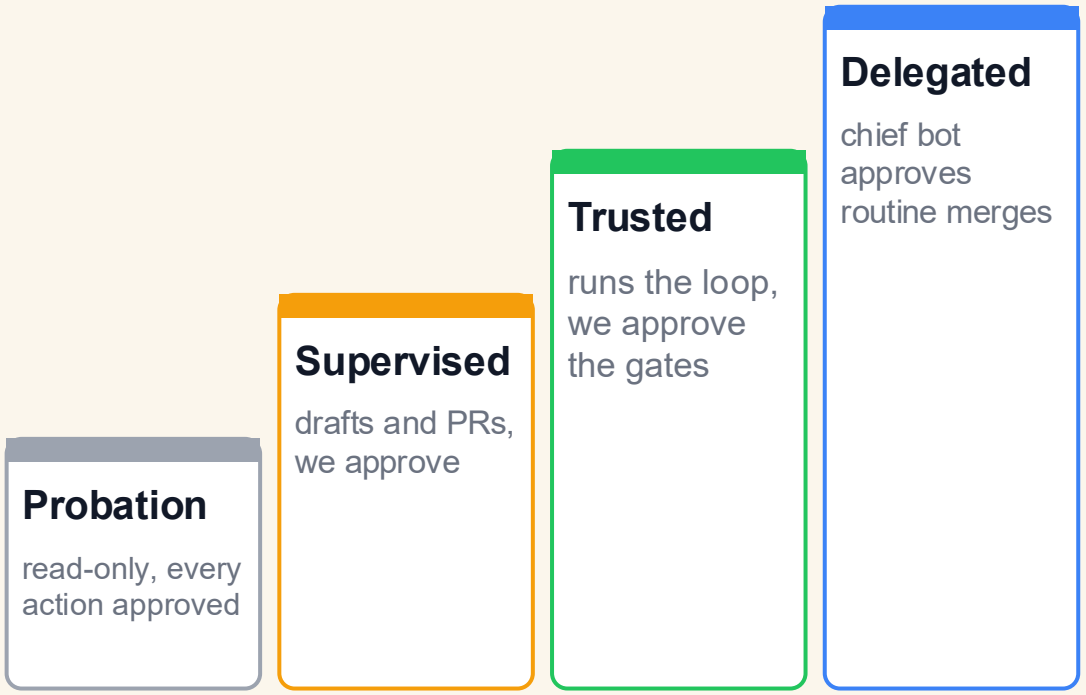
House rules

- Never send, delete or pay without my approval
- Nothing leaves without the merge gate
- Unsure? Ask → Blocked, don't guess

Managing the new colleague: trust grows with evidence

Probation, reviews, feedback, a team structure — and a clean handover when it's replaced.

Trust ladder (illustrative)



Performance review

- Logs, evidence, independent verification
- Never the bot's own “all tests pass”

Feedback & retrospectives

- Daily review of finished work
- Every lesson → a skill or rule update

Team structure

- Chief bot, specialists, who reports to whom
- One primary bot per machine

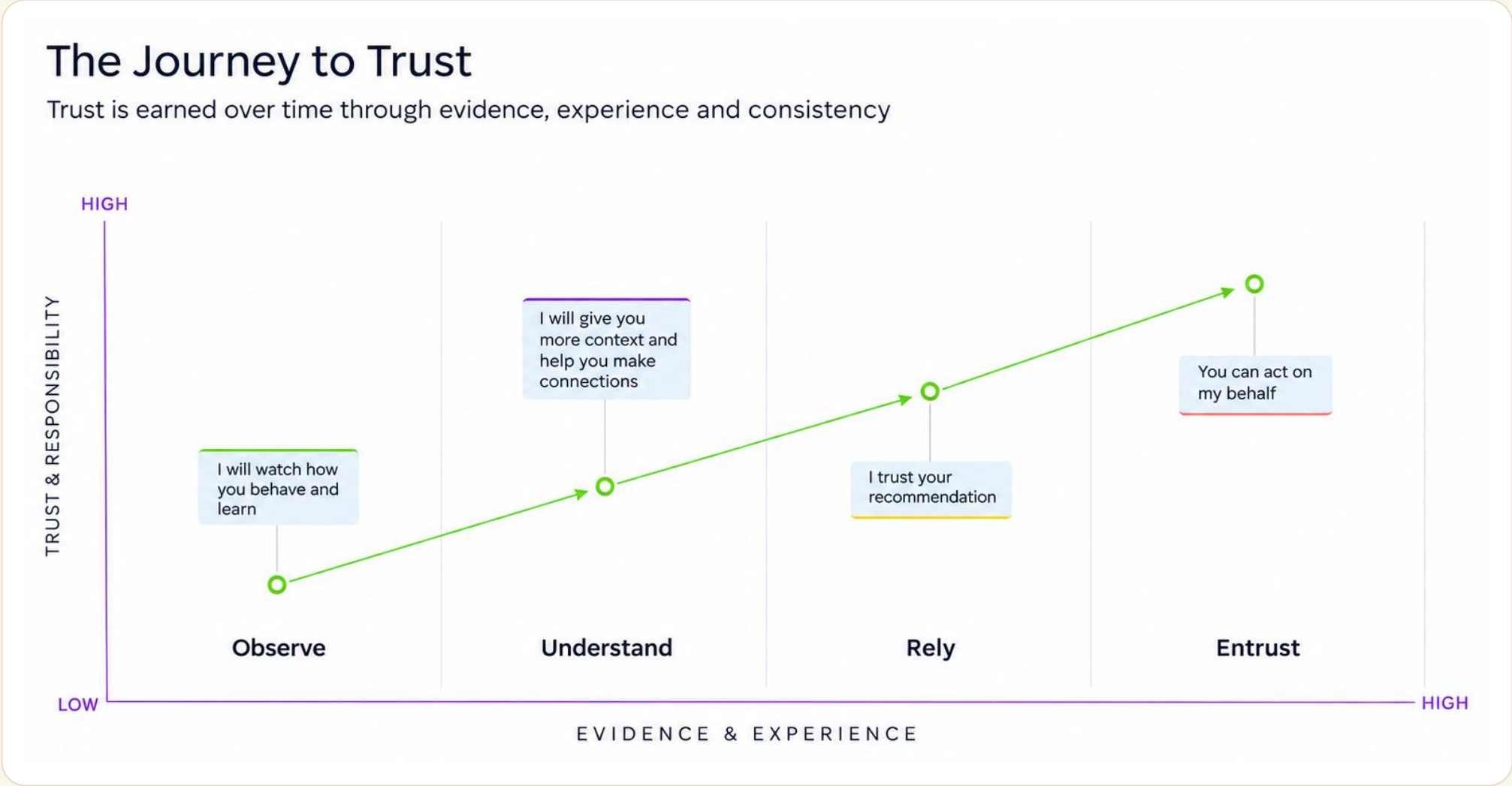
Offboarding & replacement

- Backup of memory and records
- Handover pack → new bot; old access revoked

Sources: CIO, 15 Oct 2026 (cio.com/article/4236060); VE3, 15 Oct 2026; OSIbeyond, 13 Jul 2026; Microsoft Learn, 14 Jul 2026

Look familiar?

Taken from keynote for comparison





Human new hire vs AI bot

Same playbook, different weak spots.

	Human new hire	AI bot	So what
General knowledge	Education + experience	The LLM: very broad, with a knowledge cutoff	Pick a SOTA model
Company context	Picks it up over coffee and in meetings	Only what is written down or reachable	Write it down: skills, rules, KB
Speed & stamina	Hours a day, gets tired	Fast, 24/7, many tasks in parallel	Use it for the 2+2 work
Memory	Remembers yesterday	Forgets between sessions unless given memory	Shared memory + status files
Confidence	Asks when unsure (usually)	Sure it's right — even when it isn't	Independent verification
Mistakes	One at a time	Repeats the same mistake at scale	Gates, limits, stop points
Accountability	Owns the result	Can't be accountable	I stay accountable
Leaving	Notice period, handover	Replaced in an afternoon — with a handover pack	Backup + handover docs

Sources: VE3, 15 Oct 2026 (agents don't tire, neither do their mistakes)



Where the bots run the agents: my own Mac minis

Bots never write code themselves: every inner loop runs in a local agent on my registered computers, never on the bot's cloud computer.



Claude Code
Grok Build
Codex
Cursor Agent

Mac mini M4
2024 · silver

Hosts A–D

Host A = the only host that writes to the board

Each host has one primary bot that launches its local agents
Agents per host from the CLI inventory: Claude Code, Grok Build, Codex, Cursor Agent



local agents
+ my projects

Mac mini 2018
Intel Core i7 · Space Gray

Older Intel host

Same rules on every host: the host-owner bot launches and supervises the agents, and records approvals in STATUS.

Photos: Wikimedia Commons, CC BY 4.0 (Seasider53; Khaosaming)

Why do 2+2 yourself?



Goal: automate as much as possible without losing result quality, code quality or control over code and product.

Automation → **needs Trust** → **needs Controls & verification** → **needs Evidence**

Ever had to re-check a colleague's work?

- You click through it to check it matches the request
- Preparing test data takes real effort
- In the end you just put 2+2 together
- **Why do it yourself?** Make the agents produce the evidence

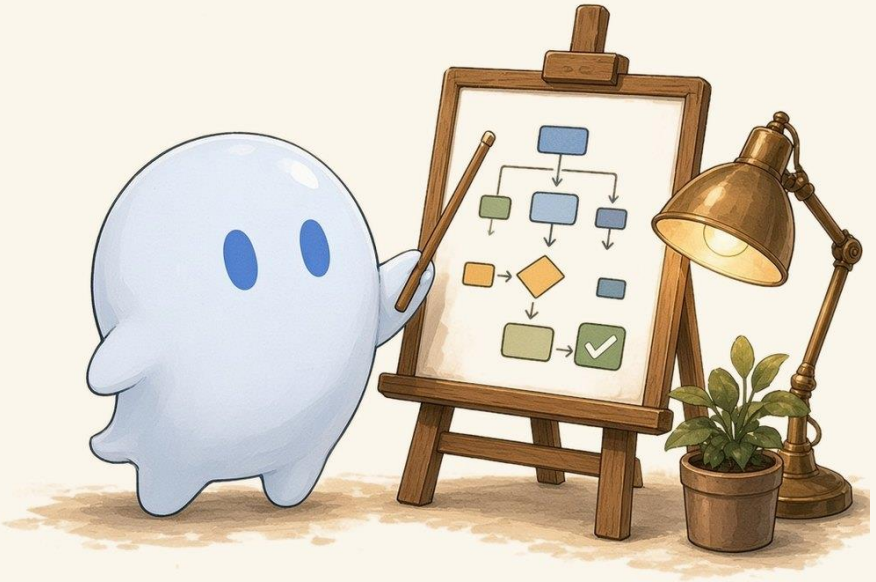
Accountability

- I stay accountable → needs trust → needs evidence
- I stay accountable; the agents do the 2+2

"The bot is not smarter than me. It is just never too lazy to prepare test data."

Backlog Todo In Progress In Review Blocked Done Archive

My workflow — how my bots work



How bots and local AI agents run tasks on the Multica board:

- phases
- roles
- gates
- check against our own description

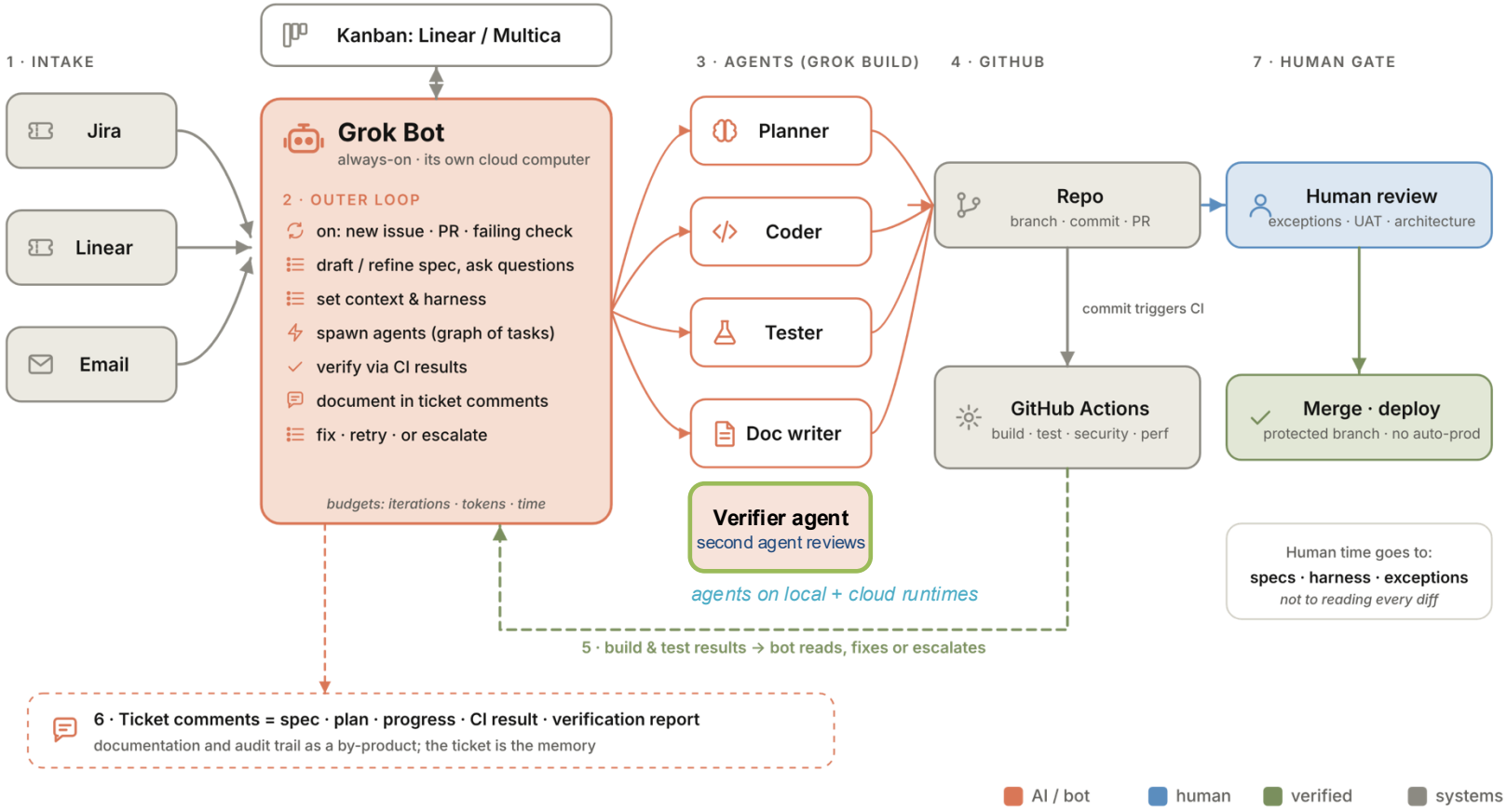
Vjekoslav Jug · Infodesign

Workflow

7 · ARCHITECTURE

October 2026: the bot runs the process

Bot runs the outer loop · agents as needed · CI verifies · everything documented · humans at the gate



GBT workflow v3: flow by Multica status and role

Multica “Grok bot tasks” · bug / small change and new feature



Backlog | **Todo** | **In Progress** | **In Review** | **Archive** | **Blocked** | **Done** | **Cancelled**

We
requester · Backlog→Todo · Gate 3 · Archive

Chief bot
chief bot · Gate 1, Gate 2 · Gate 3 (or us)

Assigned bot
The bot / authorized bots · host owner · GBT via multica CLI

Grok Build
AS-IS · verification

Claude Code
design · fix · PR

Sources
docs, wiki, mail, Jira, other bots

1 **Assigns the task** to chief bot: bug or **new feature** (fills in the request as well as possible); picks the bot

2 **Proposes a bot** by role, primary computer and availability (§3a)

3 **Opens GBT-NN** (Backlog), assignee = assigned bot · multica CLI on host

4 **Enrichment** starts right away in Backlog: docs, wiki, mail, Jira, other tickets; asks questions

5 **Backlog → Todo**: we, on the board, or **ask chief bot** to move (one by one or in bulk)

Answers the grilling questions until the bot has all it needs

questions ↔ answers

Bug or new feature?

new

bug

6a **Bug**: Grok Build **AS-IS** reproduction → AS-IS.docx + goal definition (OD, size>0)

6b **New feature**: assigned bot **grills** us with questions until it has everything → goal definition / spec

7 **Design**: Claude Code **plan mode** → PLAN-OPCIJE.md (1–5 options, scores 1–100)

Gate 1
Chief bot: goal clear? 6 criteria + option → TO-BE.v1

NO / unclear

YES → In Progress

8 **Status In Progress**; assigned bot (host owner) creates worktree `fix/gbt-NN-…` and **launches Claude Code**

Supervision: poll ≤10 min; 30 min without progress → correction round (max 3)

9 **Inner loop**: fix (Opus 5.5 xhigh) until tests pass → RJESENJE.md, local commit

tests pass →

NO: correction round (max 3); 3× → Blocked

10 **Status In Review** as soon as verification starts; launches Grok Build

11 **Verification** vs TO-BE: VERIFIKACIJA .docx + .md, KEY-1..3.png → OD (size>0)

Pass? all acceptance criteria

NO

YES

Gate 2
Chief bot: solved per goal, evidence clear?

NO

YES

Gate 3
We **or chief bot**: approve push / merge

NO → Blocked

YES

or chief bot approves push / merge (since 30 Oct)

12 **KEY-1..3** images posted as a comment by the bot

13 **Claude Code prepares the PR** fix/gbt-NN → main

14 **Claude Code: push + merge** → main (git) · no git: §8

Archive: We, manually, daily, as a check

Future: retrospective drives updates to skills, rules and docs

Blocked: question for us · 3 rounds · deadline. After the answer → previous status

manual, daily

merged → Done

15 **Done**: merge into main already made; log, Completed, polling stops

Cancelled: when a ticket is dropped

Sources: docs, wiki, mail, Jira, Teams, other tickets, automation, other bots
= input for enrichment, not an order

input

step (number = order) · gate / decision · note · future · flow · input / future · NO / loop back

Sources: outer-loop-control, lokalni-ai-agenti-i-multica (updated 30 Oct) · our decisions 30 Oct 2026, up to 22:55

Who does what, and on which computer



Swimlane	Who / computer	What they do in the GBT flow
We	requester	Assigns the task (fills in the request for a new feature); picks the bot; moves Backlog → Todo (ourselves or by telling the chief bot); answers grilling and Blocked questions; **Gate 3** (or the chief bot); **Archive**, manually, daily
Chief bot	chief bot, no primary computer	Proposes the bot; moves tasks to Todo on request (one by one or in bulk); **Gate 1** (goal, option → TO-BE.v1); **Gate 2** (verification and evidence); **Gate 3** (or us): approves push/merge
Assigned bot	The bot or another authorised bot with Multica skills	Opens and runs the GBT task via multica CLI on the host, one batch per phase, tagged with the bot's name; enrichment in Backlog; grilling for a new feature; launches the agents as host owner
(host owner)	Each assigned bot has one primary host; some bots share a host	Work folder, BRIEF/STATUS, worktree, launching and supervising agents (poll ≤10 min), approvals at stop points, smoke check, OneDrive upload
Grok Build	local agent on the host	AS-IS reproduction (bug) + goal definition; verification against TO-BE, VERIFIKACIJA.docx/.md (verification), KEY-1..3.png. Fallbacks: Cursor Agent → Claude Code → Codex
Claude Code	local agent, Opus 5.5 xhigh	Design in plan mode (PLAN-OPCIJE = plan options), fix on the fix/gbt-NN-… worktree, RJESENJE.md (solution), **prepares the PR**, **carries out push/merge** after Gate 3. At most 5 in parallel
Sources	docs, wiki, mail, Jira, tickets, other bots	Input for enrichment, not orders. Tasks come only from us

Backlog PHASE 0

Intake and opening the GBT task



Input

- Our task to the chief bot or the assigned bot: a bug, a small change or a **new feature**
- External signals (mail, Jira, Teams, automation, leftovers from an earlier GBT) → **input, not an order**
- Only tasks that involve work on a registered computer go on the board

Who (bot / agent)

- **We:** assign the task, pick the bot
- **Chief bot / a bot:** proposes the bot
- **Assigned bot:** an authorised bot, via multica CLI on the host

Activities

- The chief bot proposes a bot by **role, primary computer and availability**; we choose
- The **assigned bot** (an authorised bot) opens **GBT-NN** in the GBT workspace: issue create … --status backlog
- Assignee = assigned bot; project = customer; host label = primary computer
- Description: title, procedure, checklist, OneDrive links (only after size > 0), Do not touch
- From here on we talk **directly** to the assigned bot (name + GBT-NN)

Output / artifacts

- GBT-NN in Backlog
- Assignee, project, host label
- Description .md on OneDrive

Gate / criterion

- The task comes from us
- Work happens on a host (otherwise it does not go on the board)
- GBT workspace (WIP only on explicit order)

What can go wrong

- Mail or another bot's message treated as an order · an issue on WIP without an order · writing to the board outside multica CLI on the host (browser, box) · missing assignee (falls back to the chief bot)

Backlog Todo PHASE 1



Enrichment in Backlog, then on to Todo

Input

- GBT-NN in Backlog
- Sources: docs, wiki, specifications, the project's docs/, mail, Jira, other tickets

Who (bot / agent)

- **Assigned bot:** enrichment, brief
- **We:** Backlog → Todo
- **Chief bot:** moves tasks on our order

Activities

- As soon as the task lands in Backlog, the assigned bot **starts enrichment**: pulls the information it needs from the available sources and **asks questions**
- Work folder C:\Work\AI-agent-tasks\<yyyy-mm-dd> <task>\ (Mac ~/Work/AI-agent-tasks/…), **BRIEF.md** + **STATUS.md**, a copy of the brief on OneDrive
- Brief: goal, AS-IS/TO-BE paths, DoD, deadline, skills, agent and fallbacks, what it may touch, **the hard list verbatim**
- **We move Backlog → Todo:** ourselves on the board, or by telling the chief bot to move tasks one by one or in bulk
- Status-file entry once the agent starts

Output / artifacts

- Enriched description and questions
- BRIEF.md, STATUS.md, OneDrive folder
- Task in Todo

Gate / criterion

- **No brief, no launch**
- Todo only by our decision

What can go wrong

- The bot moves the task to Todo on its own · questions left unanswered before Todo · tokens or passwords in the brief · a brief without the hard list

Todo PHASE 2

Todo: bug or new feature, then design



Input

- Task in Todo, BRIEF.md
- Task type: bug or new feature

Who (bot / agent)

- **Grok Build:** AS-IS (bug)
- **Assigned bot:** grilling (new), launch, supervision
- **Claude Code:** design

Activities

- **Bug / small change:** the assigned bot (host owner) launches **Grok Build**: reproduction on the host → **AS-IS.docx** with screenshots, straight to OneDrive
- Along with AS-IS: goal, cause (or hypotheses), context, programs, type; environment errors → a separate subtask
- **New feature:** the assigned bot **grills** us until it has all the information → goal definition (separate slide)
- **Design (both paths):** Claude Code in **plan mode** (Opus 5.5 xhigh) → **PLAN-OPCIJE.md**: 1 option or 2–5, each with scores 1–100
- STATUS: as-is → plan / spec

Output / artifacts

- AS-IS.docx (+ PNG) on OneDrive, or the grilling record
- Goal definition
- PLAN-OPCIJE.md

Gate / criterion

- **Gate 1** (Phase 3): goal definition with 6 elements + choice of option
- Deadline (bug): AS-IS 1 h, plan 1 h

What can go wrong

- Reproduction fails · agent not signed in (next agent in line; we do the sign-in) · referencing a local path before the upload · deadline exceeded → Blocked

Todo PHASE 2b

New feature: request and grilling



Input

- Our request, filled in as well as we can: goal, users, scope, examples, constraints
- Result of the enrichment in Backlog

Who (bot / agent)

- **We:** request and answers
- **Assigned bot:** grilling, specification
- **Claude Code:** design

Activities

- During analysis the assigned bot **grills** us: it keeps asking until it has **all the information it needs** about what has to be built
- Questions by topic: problem and goal, scope and what we leave alone, behaviour and examples, acceptance criteria, evidence, risks
- It writes the answers into the brief and STATUS; nothing is assumed
- **Only then** the goal definition / specification with 6 elements
- Then design: Claude Code plan mode → PLAN-OPCIJE.md → Gate 1
- Stack for new code: Node + Express, local Postgres, Vue 3, Quasar (unless the repo has its own)

Output / artifacts

- Record of questions and answers
- Goal definition / specification
- PLAN-OPCIJE.md

Gate / criterion

- No open questions before the goal definition
- All 6 elements of the definition filled in

What can go wrong

- Specification written before the questions are answered · the bot "fills" gaps with assumptions · scope grows during grilling without a record

Todo | In Progress | Blocked | PHASE 3

Goal confirmation (chief bot) → In Progress or Blocked



Input

- Goal definition + AS-IS or grilling record (OneDrive links)
- PLAN-OPCIJE.md

Who (bot / agent)

- **Chief bot:** confirmation
- **Assigned bot:** posting, launch
- **We:** when the chief bot cannot decide

Activities

- The assigned bot sends the goal definition to the chief bot and posts it on the GBT task
- The chief bot checks **6 criteria:**
  - 1 the problem in one sentence + source
  - 2 scope: program, module, database, host, what we leave alone
  - 3 measurable expected behaviour
  - 4 1–5 numbered acceptance criteria
  - 5 the evidence required
  - 6 risks, dependencies, open questions
- The chief bot picks an option → locked **TO-BE.v1.md**; the **assigned bot** (host owner) then launches Claude Code

Output / artifacts

- Confirmed goal
- TO-BE.v1.md
- Status In Progress or Blocked

Gate / criterion

- All 6 clear → **In Progress**
- Unclear, or a question only we can answer → the chief bot orders **Blocked**

What can go wrong

- Blocked without a concrete question · TO-BE changed without a new version (with a record) · agent launched on a non-primary host without recording why

In Progress PHASE 4

In Progress: the Claude Code inner loop



Input

- Locked TO-BE.v1, BRIEF
- Capacity: at most 5 Claude Code tasks

Who (bot / agent)

- **Claude Code:** inner loop
- **Assigned bot:** supervision, approvals recorded in STATUS

Activities

- Assigned bot (host owner): git worktree add <root>-wt/gbt-NN -b fix/gbt-NN-<desc>, dependency check
- Launch claude -p … --model opus --effort xhigh from the worktree, log + PID
- Claude Code implements the chosen option, runs tests until they pass, writes **RJESENJE.md**
- A local commit is fine; **push only after Gate 3**
- No git: separate folder, SVN WC read-only; OpenEdge demo with the work folder first in PROPATH (no -propath)
- **Supervision:** poll ≤10 min, independent signals (PID, mtime, diff, log); 30 min without progress → correction round
- Stop points: install, registry/DSN, commit/revert, database writes, sending anything out, deadline expiry

Output / artifacts

- Diff on fix/gbt-NN-…
- Local commit, RJESENJE.md
- STATUS fix, owner claude-code

Gate / criterion

- The coding agent's tests pass, but **Claude Code alone is not trusted**
- Fix deadline 3 h

What can go wrong

- Stall → at most 3 correction rounds, then Blocked · database writes or work on production without an explicit order · someone else's worktree or PID · two agents writing to the same folder

In Review PHASE 5

Verification (tester agent) and evidence · In Review



Input

- RJESENJE.md, branch, TO-BE.v1
- Claude's tests pass

Who (bot / agent)

- **Grok Build** (fallbacks: Cursor Agent → Claude Code → Codex)
- **Assigned bot:** status, launch

Activities

- The assigned bot sets **In Review as soon as verification starts**
- It launches **Grok Build** to verify against the locked TO-BE
- **VERIFIKACIJA.docx** (screenshots) + **VERIFIKACIJA.md** (pass/fail)
- **KEY-1..3.png** (up to 3 key screenshots) in the same OneDrive folder
- Everything via m365 to OneDrive, **size > 0** before phase: chief-review
- The assigned bot smoke-checks the claims; it does not "finish" someone else's code
- It shows the chief **only OneDrive links**: AS-IS, PLAN-OPCIJE, TO-BE, RJESENJE, VERIFIKACIJA

Output / artifacts

- Evidence report and KEY images on OneDrive
- STATUS verify → chief-review

Gate / criterion

- Pass on all acceptance criteria → Gate 2
- Fail → correction round (max 3) → Blocked
- Verification deadline 1 h

What can go wrong

- 0 B upload or relying on OneDrive sync · verification by the same agent that did the fix (not allowed for the second validation) · referencing a file before Graph confirms it

In Review PHASE 6

Chief confirmation (Gate 2)



Input

- OneDrive links to all evidence (size > 0)
- VERIFIKACIJA.md: pass

Who (bot / agent)

- **Chief bot:** confirmation
- **Codex:** optional
- **Assigned bot:** comment with KEY images

Activities

- The chief bot checks: **solved per the goal** and **evidence clear**
- If unsure → second validation: **Codex** (or a named model), never the agent that did the fix
- Outcome goes into STATUS
- After confirmation the assigned bot posts the **KEY-1..3** images as a comment (tagged with the bot's name, one batch)
- To us: **"the solution is ready"**

Output / artifacts

- Confirmation in STATUS and on the GBT task
- KEY images visible on the task

Gate / criterion

- YES → PR and Gate 3 (Phase 7)
- NO → correction round or Blocked
- **Status stays In Review**

What can go wrong

- Confirmation without OneDrive links that open · KEY images posted before confirmation · Done set before the merge

In Review PHASE 7

PR, Gate 3 and merge



Claude Code prepares the PR. Gate 3: approved by us or the chief bot. Claude Code (launched by the assigned bot) then pushes and merges while the task is In Review; Done comes after.

1 · GitHub PR (remote)

- Claude Code **prepares the PR** (draft) from fix/gbt-NN-…
- **Gate 3:** approved by us or the chief bot
- Claude Code pushes and merges the PR into main
- Without approval it stays a local branch

2 · Local git merge

- Claude Code prepares the merge fix/gbt-NN-… → main
- **Gate 3:** we or the chief bot
- Claude Code merges on the host
- **Done always = merge into main already made**; if it was missed, merge right away

3 · No git (SVN WC)

- Close-out **"Closing without git"**: feature folder vs WC (size, then SHA256)
- Only changed programs are copied; the original goes to <name>.bak-YYYYMMDD-HHMMSS; SHA check
- Feature folder out of PROPATH (with a backup); env/registry read-only
- No SVN commit, no database writes, nothing is deleted
- **ZATVARANJE.md** (close-out) on OneDrive, size > 0

Gate 3 · we or the chief bot
Approval for push/merge comes from us or the chief bot. No approval, no push and no merge.

Executor: Claude Code
Push and merge are carried out by Claude Code, launched by the assigned bot (host owner). No git: close-out without git.

Done Archive Cancelled PHASE 8

Done, Archive and Cancelled



Input

- Merge into main made (git) or ZATVARANJE.md (no git)
- Gate 2 and Gate 3 (us or the chief bot) passed

Who (bot / agent)

- **Assigned bot:** Done, log
- **We:** Archive, manually and daily
- **Dropped ticket:** Cancelled

Activities

- **Done:** tested (Grok verification + chief bot), **merge into main already made**, evidence in the log and on the board
- Deliverables on OneDrive; folder listing and **size > 0** per file
- The assigned bot sets **Done**: a 1–2 sentence comment + OneDrive link (no share link)
- **Log:** STATUS.md + the bot's log (time, host, agent, phases, decisions, stop points)
- Status file → Completed; polling stops
- **Archive:** for now we do it **manually, every day**, as a check (not a bot); **future:** a retrospective drives updates to skills, rules and docs
- **Cancelled:** when the ticket is dropped

Output / artifacts

- GBT-NN in Done, later in Archive
- Evidence on OneDrive and on the board (KEY images)
- Log and STATUS

Gate / criterion

- Done only with a merge into main
- Archive: daily check (us)

What can go wrong

- Done without a merge (fix: merge right away) · deploy or run on a production database without an explicit order · a bot archiving instead of us · deleting the feature folder without our instruction

Blocked EXCEPTIONS

Blocked, stop points and deadlines



When a task goes to Blocked

- A question for us (goal unclear, decision beyond the bot's authority)
- 3 correction rounds without success (stall or failed verification)
- Phase deadline exceeded
- Escalation chain: assigned bot → chief bot → us
- After the answer: back to the previous status

Stop points (agent exits cleanly)

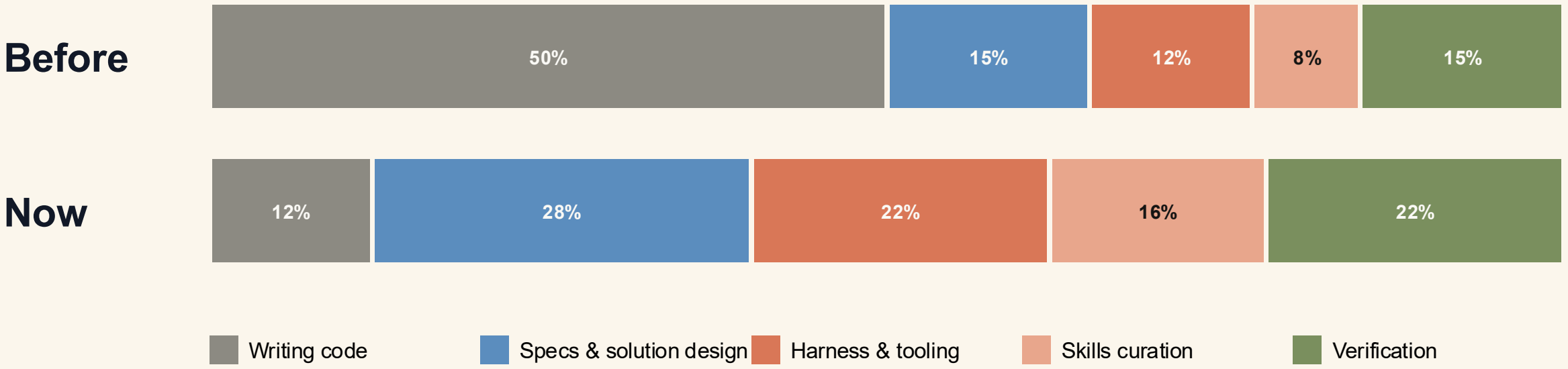
- Installation, registry / DSN
- Commit / revert / update
- Database writes, application folder
- Personal data leaving the host, sending anything out
- Approved by the assigned bot (host owner) and recorded in STATUS, **except the hard list** (us only)
- **Push/merge:** Gate 3, we or the chief bot

Deadlines and rounds (default)

- AS-IS 1 h · plan/TO-BE 1 h
- Fix 3 h · verification 1 h
- Limits per phase — no total promised per task
- Stall = 30 min without progress (3 checks)
- At most 3 correction rounds, then stop
- The numbers get checked on the first few tasks

Hard list: only databases named in the brief · only our registered computers, never a customer server · nothing sent out; git push only after Gate 3 (us or the chief bot) · never an SVN commit · no runs on a production database without an explicit order.

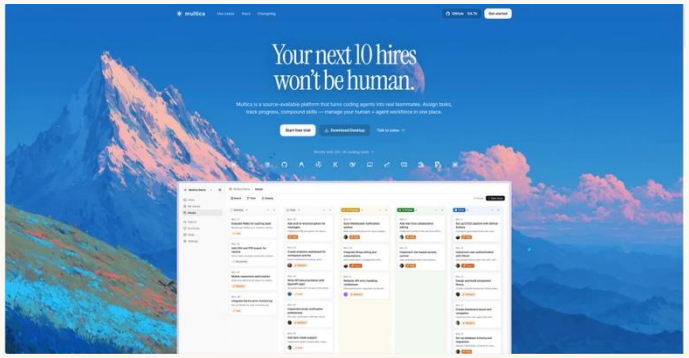
What a developer does now



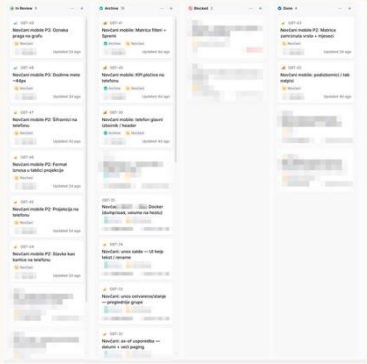
Writing specs is solution design. Building the harness is architecture. Curating skills is knowledge management. Verifying is engineering. None of it is "prompting".

In Review Done 7 · DEMO RESULT

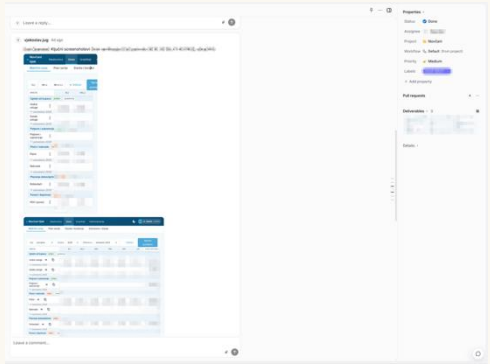
Back to the board: the demo result



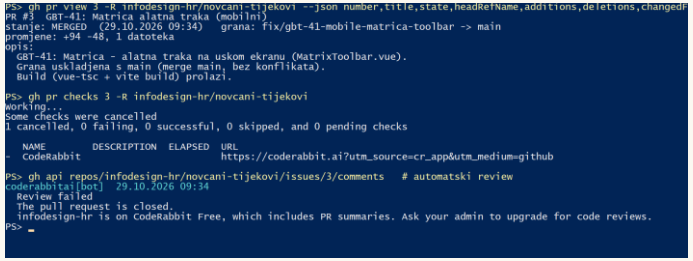
Multica — the board the demo runs on



Demo board: tasks GBT-39…49 in In Review, Archive and Done



GBT-43 in Done with the “Ključni screenshotovi” (key screenshots) comment



Automated review on PR #3 (GBT-41)

“Automation without an off-switch is not a workflow, it is a hope.” Gate 3 is still ours (or delegated to our chief bot).

Things that went wrong

Honest slips from our own work, and what we changed after each one.

1 · The agent went too wide

We asked for one change and got extra refactoring and features nobody asked for. **Now the ticket says what is in scope, and what is not.**

2 · Works for the agent, hard to ship

The result ran fine in the agent's own environment, but installing it to production was hard. **"Done" now means installed and running.**

3 · Too much chatter between bots

Bots kept confirming and asking each other questions. **Now a bot hands off once, with everything the next one needs, and stops.**

4 · A new problem needs us too

Don't expect an agent to solve a new kind of issue alone. **We work it out together first, then save the solution as a skill.**

Each slip became a rule in the process, not a reminder in someone's head.

More slips — and the rule each one became

Meta-lesson: encode the lesson in structure (a skill), not in your memory.

Clicking merge without looking

→ Gate 2 by the chief bot + key screenshots on the card + Gate 3 approval

Wrong target (DB, folder, stray OneDrive folder)

→ Hard list in every brief, fixed working folders, path-mapping rules in a skill

Stuck agents burning hours

→ 10-min supervision on independent signals; 30 min no progress → correction; 3 rounds → Blocked

Mail / another bot treated as an order

→ Only the human creates tasks; everything else is input

“It's in my working folder”

→ Upload first, size check, then link — a local path is never a deliverable

Hype tools with local LLMs

→ Use SOTA models — the agent is only as good as the model

Real case (Oct 2026): a coding agent's sign-in expired on one host and the nightly skill sync failed for about 9 days, until someone looked → login check before every agent run.

What about us

OpenEdge users?

AI agents can work with ABL and legacy code too, if you teach them.

- **ABL & legacy code:** agents read, fix and test existing .p / .w programs
- **A real OE to test against:** local databases, OE in Docker
- **Where they slip:** PROPATH, codepage / convmap.cp, Crystal Reports
- **Teach them with skills:** every lesson becomes a rule file the next agent reads



The model doesn’t know ABL best — but it has the OpenEdge AI Assistant (MCP)

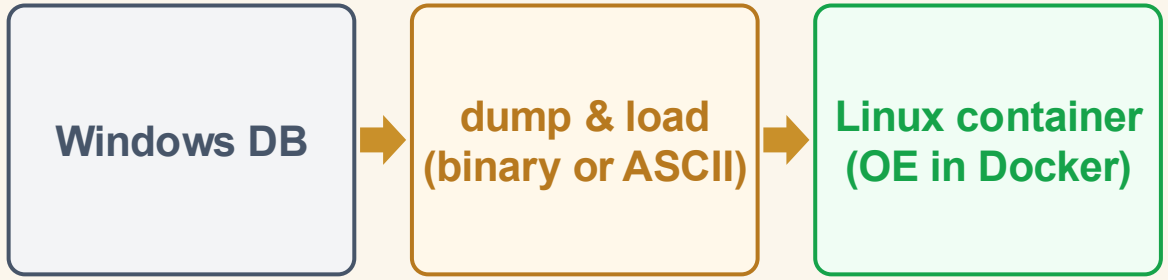
It still doesn’t know YOUR PROPATH: every pitfall we hit once became a rule in a skill.

Pitfall	Symptom	Skill rule
PROPATH	Wrong program version runs; demo touches the production working copy	No -propath startup parameter: set it via .ini, registry, environment or code; task folder first in PROPATH
Codepage / convmap.cp	Garbled characters	Keep a verified full convmap with checksum; the skill says where to get it
Crystal Reports	Agent can't edit binary .rpt files	Dedicated SDK-based tool/skill + screenshot-based verification
Legacy VCS (SVN working copies)	Risky commits by an agent	Read-only for agents; copy back with timestamped backups and SHA checks
GUI apps	“Tests pass” but the screen is wrong	Verification by computer use / screenshots — the tester agent's job (AS-IS + VERIFICATION docs)
Databases	Wrong DB touched	Only databases named in the brief; customer DB copies restored locally via a skill
Windows → Linux DB move	Copied DB files don't work	Dump & load, binary or ASCII; copying DB files is not supported (Progress KB P44220), the agent doesn't know until a skill tells it

OE in Docker, and what a skill is worth

Reproducible builds and tests on Linux; the before/after comes from a real task.

Moving a DB from Windows to Linux (Progress KB P44220)



Why Docker

- Reproduce, build and test ABL the same way on every host
- Copying DB files across platforms is not supported — dump & load only

Before / after a skill

- The same task, without and with the skill:
  - Without the skill, the agent tried to copy the DB files and got stuck in a loop
  - With a simple skill, the agent got it done quickly and reliably

AI and ABL: teaching the agent our code base

The model doesn’t know ABL best. Give it OpenEdge context, our rules and real checks.

Use the tools



OpenEdge AI Assistant
Progress’s MCP: ABL syntax, rules, docs, KB



OpenEdge ABL for VS Code
Riverside Software, sponsored by Progress



PCT
Progress Compilation Tools: headless build for CI

Adapt community skills

Start from shared ABL skills and MCP tools from the community.

Adapt them to our code base: PROPATH, DB connections, includes, naming.

Prove them on a real task, then share them with the team.

Give it our context

Rules: our ABL conventions (AGENTS.md)

Docs: architecture, data dictionary

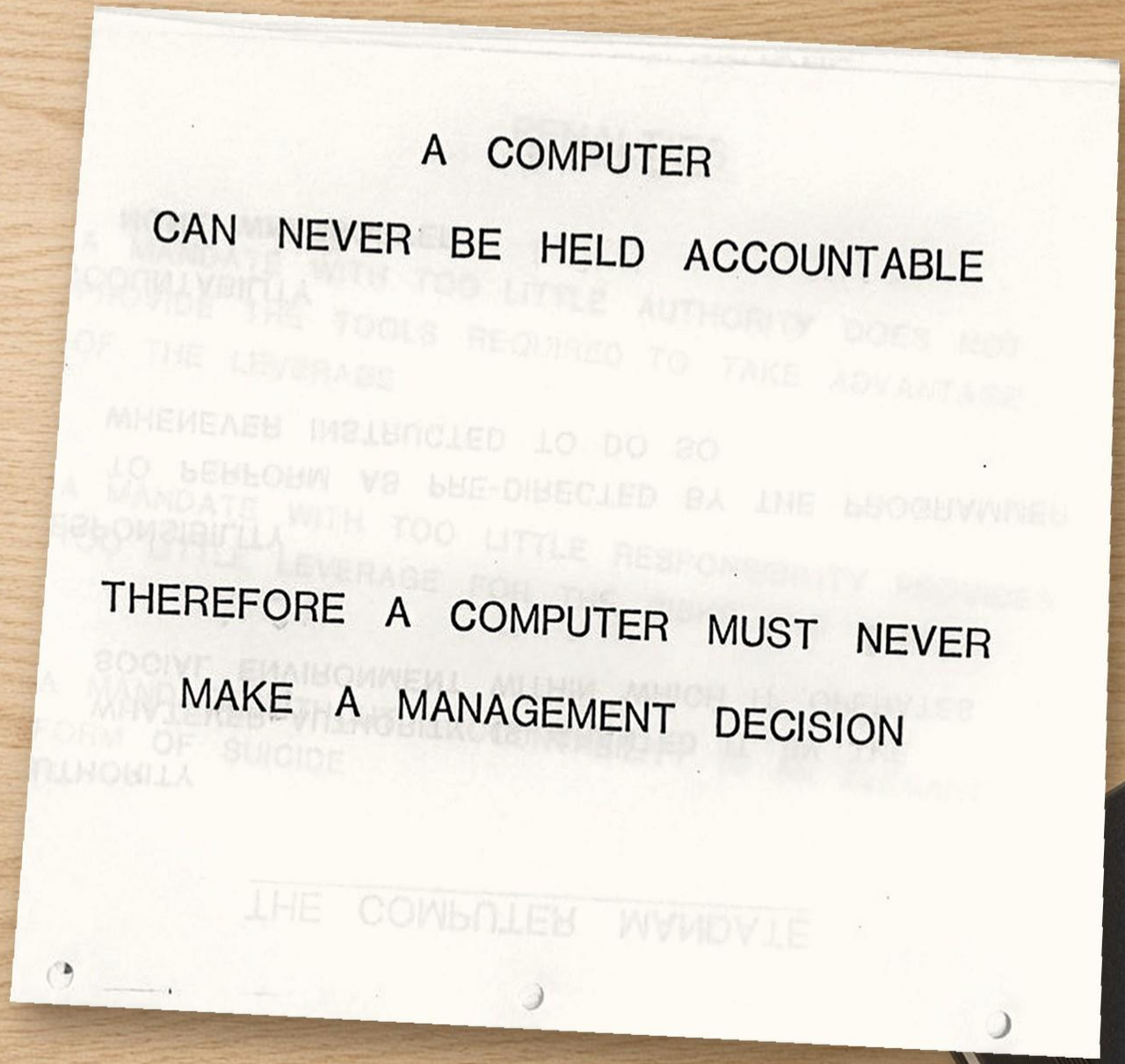
Golden examples: from our own code

Compiler + ABLUnit: the signal

ABL depends on version, PROPATH, includes, DB, PASOE: plausible code that doesn’t compile is the default failure.

10 · ACCOUNTABILITY

Who is accountable for AI agents' work?



IBM internal training, 1979 (attributed).

First posted online by @bumblebike, 17 Feb 2017. Original lost in a 2019 flood; IBM archives have no copy.

Done

10 · ACCOUNTABILITY

Courts and regulators say the same thing

1979, 2024, 2027: a human or a company answers for the machine.

Moffatt v. Air Canada — 14 Feb 2024

- Its website chatbot told a customer he could claim a bereavement fare after the trip — wrong
- Air Canada argued the chatbot was responsible for its own actions
- Tribunal: “a remarkable submission … it should be obvious to Air Canada that it is responsible for all the information on its website”
- Air Canada had to pay C$812.02

EU AI Act — Art. 14 human oversight

- High-risk AI must be built so that natural persons can effectively oversee it
- Overseers must understand limits, watch for automation bias, and be able to override or stop it
- Art. 26: deployers assign oversight to people with competence, training and authority
- Applies from 2 Dec 2027 (Annex III) / 2 Aug 2028 (Annex I) after the 2026 Digital Omnibus

Most coding bots are not “high-risk AI” under the Act — but the direction is clear: keep a human in the loop who understands the system and can say no.

Sources: Moffatt v. Air Canada, 2024 BCCRT 149 (14 Feb 2024); EU AI Act Art. 14 & 26; Council of the EU, 29 Jun 2026; AI Act Service Desk FAQ

Blocked In Review 10 · ACCOUNTABILITY

How my setup keeps a human accountable

Not a policy PDF — six mechanisms that run every day.

1 · Human approval gates

- Gate 1: goal and DoD confirmed before work starts
- Gate 3: push / merge only after our (or the chief bot's) OK

2 · Never send without OK

- No e-mail, message, post or payment leaves without my explicit approval
- Bots draft — I press send

3 · Evidence & logs per step

- Screenshots, test data, logs on the ticket
- Status file per task; nothing is “done” without proof

4 · One owner per task

- One ticket, one assignee, one working folder
- Every bot has a named human owner

5 · Least privilege

- Own identities, read by default, write only where needed
- One primary bot per machine; a hard list of forbidden actions

6 · Retrospectives

- Daily review of finished work
- Every slip becomes a skill or rule update

Rule of thumb: the bot decides how — a human decides whether.

Sources: Microsoft Learn, Least privilege for AI agents, 14 Jul 2026

In Review

10 · ACCOUNTABILITY

Delegate the work — never the accountability

The bot is like a new employee, with one difference: it can't answer for the result. A named human always does.

Who answers for what

- **Bot = Responsible**: does the work, leaves evidence
- **Human = Accountable**: decides, approves, answers to customers and the company
- Exactly **one A per task**, always a named person
- “The agent did it” is not an answer — not to a customer, not to a court
- A chief bot may approve routine steps, inside rules I set — the A stays mine

Activity	Assigned bot	Chief bot	Me (human)
Recon + DoD (Gate 1)	R	C	**A**
Code, tests, evidence	R	I	**A**
Independent verification (Gate 2)	C	R	**A**
Push / merge (Gate 3)	R (after OK)	C / approves routine	**A — approves**
Mail or message to a customer	drafts	—	**A + R — sends**
Outcome for the customer	I	I	**A**

R = Responsible (does it) · A = Accountable (answers for it) · C = Consulted · I = Informed. Illustrative RACI of my setup.

Sources: IBM training slide 1979 (attributed; Simon Willison, 3 Feb 2025); RACI = standard responsibility-assignment matrix

Don't forget about security and privacy

Agents get keys only to the rooms they need.

- **A guard bot:** checks the shared computer and sessions twice a day
- **One golden rule:** never send, delete or pay without my confirmation
- **Least access:** permissions per bot and per computer
- **GDPR:** know what personal data an agent can see, and why



Security & control

Orchestrator in its own cloud box

executors on dedicated machines — not on a developer's daily laptop

Branch protection; no auto-deploy to production

a human approves the PR

Demo & test data anonymised

no customer data

Secrets never enter the context

least-privilege tokens

Every action is a ticket comment

→ audit trail

What the PM tool must support

API / webhooks · agent assignment · comments as first-class objects

Yes, code goes to the models. Secrets, production data and production deploys do not.

Security

10 · WHO CONTROLS THE BOTS?

Security layers and GDPR

The first question every CTO asks — and the answer is structure, not trust.

Human gates

Gate 3: no push / merge without our (or delegated) approval

Hard list

Only named DBs · only my registered computers · never customer servers

Per-bot access

Each bot gets only the connectors and computers it needs

Per-machine primary bot

One owner per machine; board writes from one host only

Logs & evidence

Detailed logs; every deliverable on OneDrive and linked — “if it is not linked, it did not happen”

GDPR checklist

- Customer data stays on the host
- Real customer data only in the customer's task folder
- Shared docs use synthetic data, checked by a personal-data scan before publishing
- Logs without names, addresses or personal IDs
- Bot memory pruned when no longer needed
- No sharing links — paths only; secrets never in briefs, logs or chat

Code and customer data are processed on my own machines; the AI models themselves are cloud APIs, so only the context an agent needs for a task is sent to the model provider. Real customer data never appears in demos or shared documents – those use synthetic data – and logs carry no names, addresses or personal IDs. Nothing leaves the team – no e-mail, message, post or push – without my explicit approval.

Numbers?

We don't need no stinking numbers

…well, actually we do:

- Models + subscriptions: ~€290 / month (average Aug–Oct 2026)
- Benefits: parallel work — 10+ tasks in review at once (1 Nov 2026)
- Still, it pays off in the end



Where to start?

From Copilot-only to your first agent.

1. **Pick one small, real task:** a bug or a report, not a moonshot
2. **Give the agent a skill / rules file:** PROPATH, databases, conventions
3. **Keep a human gate** before push and merge
4. **Measure:** time before and after, then repeat



Takeaways

11 · CLOSING PRINCIPLES

Four things to take home

Tools will change next month. These won't.

Use SOTA models

- My experience for coding: Claude > GPT >> Grok, Gemini **(my personal ranking, Nov 2026)**
- Model ≠ bot / harness: a Grok bot runs my outer loop while Claude Code writes the code

Use skills

- Take public skills, adjust them to how you work
- Manage them in a GitHub repo, sync to every machine

Automate all you can

- Don’t do the 2+2 your agents can do
- Make them produce the evidence

Tech tips

- CLI instead of MCP where possible — faster, saves tokens
- A bot for the outer loop, Claude Code for coding
- A board (Multica, or self-built) keeps everything visible
- Detailed logs; every artifact on OneDrive, linked everywhere

Tools

TOOL OVERVIEW · ONE TICKET, START TO FINISH

From start to finish: which tool does which step

Bots run the outer loop; local coding agents do the inner loop; everything is noted, archived and communicated.

Outer loop: Grok Bot — takes tasks, sets goals, supervises the local agents, checks results and reports back

Ticket intake	Project management	Coding	Testing, documentation	Versioning
Jira	Multica	Claude Code	Grok Build	GitHub
Email	(Linear)	alt: Cursor	alt: Cursor	PR → review → merge

Around every step

- Note-taking
- Pergament — was: Notion
- Archiving, backup — OneDrive
- Team communication — Teams
- Email

Tools

TOOL OVERVIEW · ALSO IN THE TOOLBOX

Around the core: the rest of the toolbox

Tools from the talk that are alternatives, fallbacks or context — not the daily core path.

AI bots & assistants

- **Grok Bot** — outer loop, main
- **OpenClaw**
- **Hermes Agent**
- **Meta Muse**
- **OpenAI dots**
- **Claude Duet**

Coding agents: alternatives

- **Codex** — fallback
- **opencode**
- **pi** — pi.dev
- **Devin**

AI orchestrators

- **super.engineering**
- **Orca** — onorca.dev
- **Cursor agent**

Editors & IDEs

- **GitHub Copilot**
- **Windsurf**
- **VS Code** — reading code
- **Google Antigravity**
- **Kiro**

Code review

- **CodeRabbit**
- **Graphite**

Platform & models

- **Docker** — OpenEdge on Linux
- **Progress OpenEdge** — our platform
- **OpenRouter** — model access
- **Origin** — Cursor git forge

People to follow

X (Twitter) is my main source of relevant data



Andrej Karpathy

@karpathy

Photo: Gladwin Analytics, CC BY 3.0



Matt Pocock

@mattpocockuk

Photo: X profile picture (@mattpocockuk)



Lauren Tan

@poteto

Photo: X profile picture (@poteto)

Why X: by the time a video is out, the tool has changed.

Questions?



Thank you!

Let’s stay in touch

Let’s talk about your agents.

vjug@infodesign.hr

@v_jug

linkedin.com/in/vjekoslavjug

