

```
$ echo "hello"
```

Hello.

Andreas Heige

A QUESTION FIRST

What did you open first this morning?

An editor

A terminal

A ticket

A chat

Throw away your IDE?

Throw away your IDE?

Agents, harnesses and cost-aware engineering work

Andreas Heige

● ● ● ~/shop · copilot

\$ copilot

› Fix: cart shows NaN after
a market switch

- Reading issue + repo map
- Plan: 3 steps · profile standard
- Failing test written **RED**
- Minimal fix applied **GREEN**
- npm run check **PASS**
- Diff ready for your review

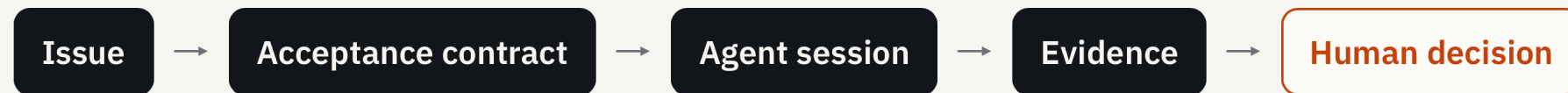
THE SHIFT

Work now starts with a contract

BEFORE



NOW



Developers define success, set the risk envelope and judge the evidence. The agent does the navigation and the mechanical work.

Throw away your IDE?

The terminal is now a full agent



Reads the whole repo



Edits files



Runs commands



GitHub and tools
via MCP



Hooks and guardrails



Subagents and fleet



Context and usage



Resumes
long sessions

This is not the old command-suggestion extension. It can own a complete repository task.

TODAY

Four parts, one loop

-
- 01 What we built
 - 02 What we learned the hard way
 - 03 The lean harness
 - 04 People make it real
-

04

PART 01

What we built

A real harness in a large e-commerce monorepo.

One monorepo, one .github folder

18

instruction files

loaded by file pattern

19

specialist agents

picked per task

35

skills

loaded on demand

14

prompts

one slash command each

All of it versioned in git, reviewed in pull requests, and shared by the whole team.

Throw away your IDE?

Four layers, four triggers

Layer	Lives in	Activates	Example
Instructions	instructions/*.instructions.md	Automatically, by file pattern	Component rules when you edit a component
Skills	skills/*/SKILL.md	When asked, or auto-invoked	“Use the tdd skill”
Agents	agents/*.agent.md	You pick one, or the orchestrator does	security-reviewer
Prompts	prompts/*.prompt.md	As a slash command	/analyze-ticket

Plus one global copilot-instructions.md that every single request reads first.

Throw away your IDE?

THE ONE EQUATION

**Agent =
Model + Harness**

The model is rented and swappable. The harness is where reliability is won.

One file opened, 10 files load

```
open src/blocks/HeroBlock.tsx
```

10

instruction files load, without a single click

ALWAYS ON • 7

copilot-instructions

agent-first

model-selection

precedence

security

code-review

tracker

MATCHED BY PATH • 3

frontend • src/blocks/**

accessibility • **/*.tsx

cms • src/blocks/**

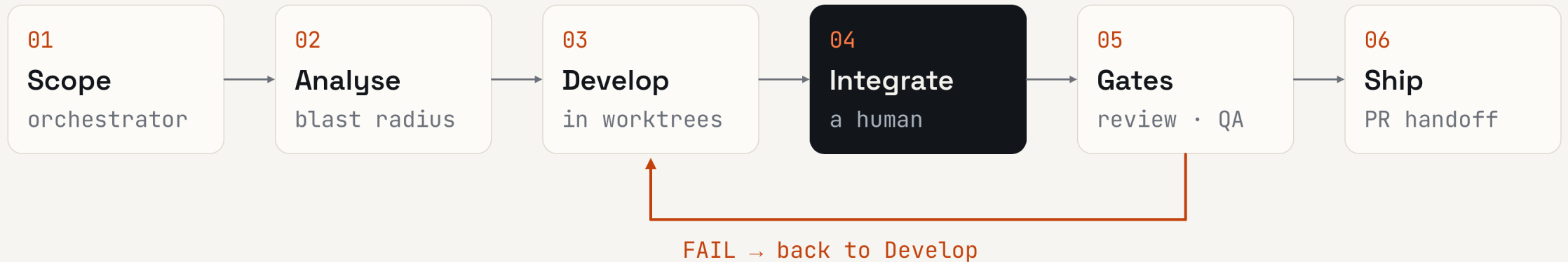
When rules conflict



One precedence file decides the order, and it loads on every request.

The orchestrated pipeline

Shared memory wraps every stage • READ → WORK → WRITE



Each agent works in its own git worktree and returns PASS, FAIL or BLOCKED. A human merges. Nothing ships on FAIL.

Binding gates before anything ships

BINDING

Code quality gate

PASS or FAIL. A FAIL blocks integration and sends the work back.

BINDING

Security reviewer

OWASP Top 10, secrets, auth and injection. Several models, always the top tier.

BINDING

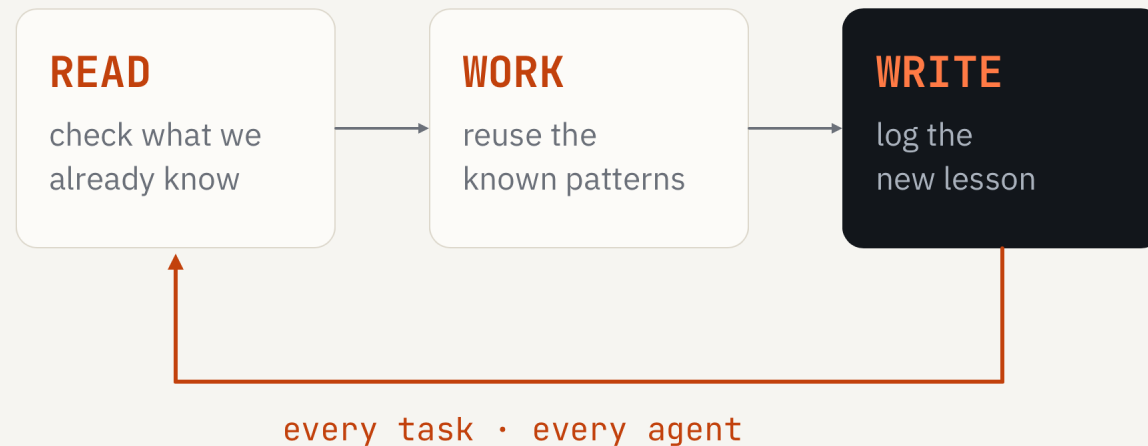
QA

Functional and performance checks. Issues the FINAL VERDICT.

 **The default is FAIL until proven.**

 **Agents never commit or push. A human does.**

Shared memory: read, work, write



Git-backed

Lives in the repo, never expires, reviewed in pull requests.

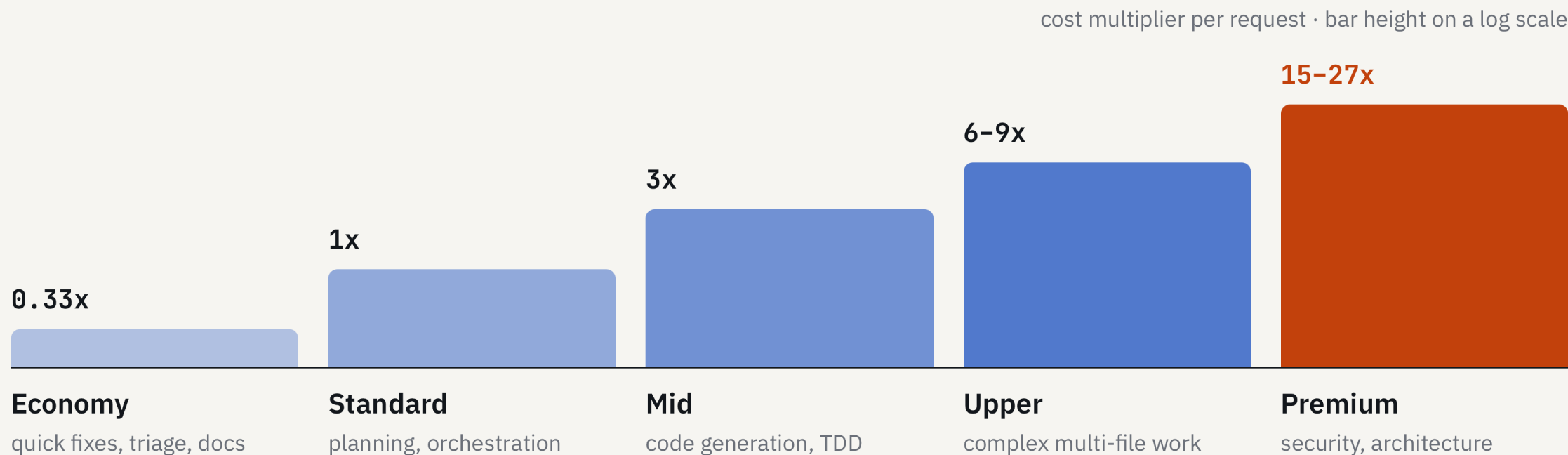
Index first

A small index is always read. One domain file only when it matches. Roughly 8 KB instead of 40 KB+.

Enforced in layers

Instructions, the shared agent protocol, every skill, and a PR checkbox for the humans.

Five model tiers: start cheap, escalate



Now built in

We hand-built start-cheap-escalate. GitHub now ships it as **HydraFusion's Cascade**.

The harness keeps itself current

```
# simplified
name: update-model-catalog
on:
  schedule:
    - cron: "0 0 1,15 * *" # 1st and 15th
  workflow_dispatch: {}
jobs:
  sync:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@11bd719 # v4.2.2
      - run: python
        .github/utils/update-model-catalog.py
```

Prices change, the harness notices

A workflow re-syncs the model catalogue twice a month, or on demand.

Same discipline as any CI

Pinned action SHAs and least-privilege permissions.

The harness is code

So it gets a pipeline, reviews and tests too.

One prompt, a whole workflow

- 01 Reads the ticket and linked designs over MCP
- 02 Maps affected files and existing patterns
- 03 Waits for your yes before every task

● ● ● ~/shop · copilot

```
$ copilot
> /analyze-ticket SHOP-4567
• Ticket: Add size guide drawer to the product page
• Figma link found → reading the design
• Searching the codebase for affected files
• Wrote .copilot-tracking/tickets/SHOP-4567/
  analysis.md · investigation.md
  implementation-plan.md · todos.md
Task 1 · Create SizeGuide component
Test: opens on click, ESC closes it
> Proceed with Task 1? [yes/no]
```

02

PART 02

What we learned the hard way

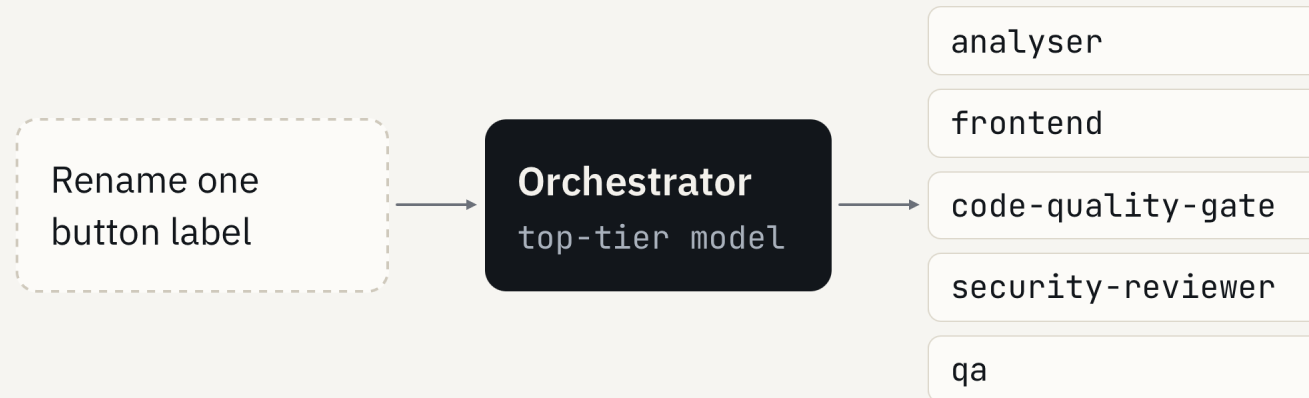
The expensive lessons, so you do not have to pay for them.

★ The biggest lesson: fan-out

Process and people

Context and cost

More agents, more interactions



Built for big changes.

For routine work, the default fan-out multiplied model calls, context and review.

Fix: let a gate decide when to delegate.

Faster code moves the queue downstream



WATCH FOR

oversized PRs

review overload

finished work waiting to ship

Speed up one stage and the next one becomes the constraint.

Throw away your IDE?

There is no single AI developer



Power users

Automate their work, write the standards, share what works.



Individuals

Have their own workflow and avoid imposed process.

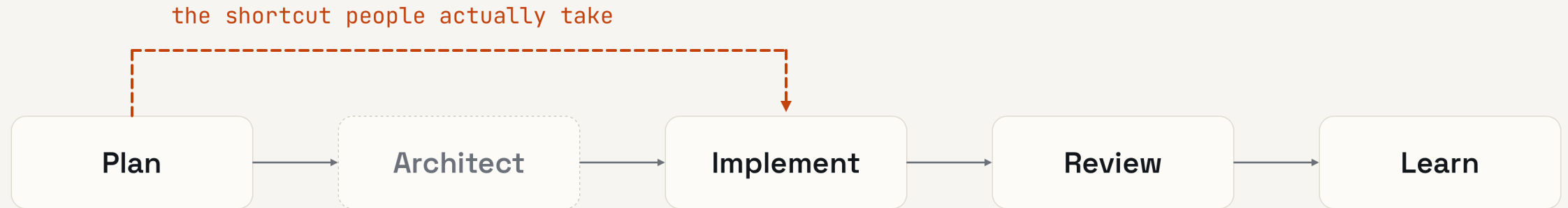


Hesitant

Use the tools lightly, or route around them.

Design the harness for all three, or it only serves the first.

A rigid pipeline invites workarounds



Plans drift from reality mid-task.

Learnings go stale when nobody owns them.

Trust drops, and the tool gets skipped.

Standardise capabilities, not every step

Skills
> Process

WORKFLOW

tdd spec-driven-development verification-loop
resolve-build-error

SCAFFOLDING

create-component create-hook create-service
create-block

TICKETS

grill-jira-ticket refine-jira-ticket

META

make-skill runbook-maintenance

The skill carries the standard. The person keeps their workflow.

Context is a budget: sessions and tools

Problem	What happens	What we do now
Context rot	Quality drops as a session grows long and cluttered.	Short, focused sessions. /compact or /new between phases.
Tool overhead	Every enabled tool and MCP server adds schema to every turn.	Enable tools per task . Remove what nobody uses.

Context is a budget: rules and memory

Problem	What happens	What we do now
Rule sprawl	Ten instruction files can load for one component edit.	Keep the always-on core small. Move detail into skills.
Memory growth	Reading every learnings file costs 40 KB+.	Index first, then one domain file.

What actually drives the bill

TOKENS

Large instructions and long sessions
repeat input on every turn.

TOOLS

Every enabled tool adds schema
and decision overhead.

AGENTS

Each subagent starts another
model loop.

Model price matters less than what you repeat on every turn.

Pick the setup once per phase

OUR OLD GUIDE SAID

~~“Switch models
freely mid-session.”~~

WHAT WE SAY NOW

Choose once per phase. Keep it stable.

CHANGING ANY OF THESE INVALIDATES THE PROMPT CACHE

Model

Reasoning level

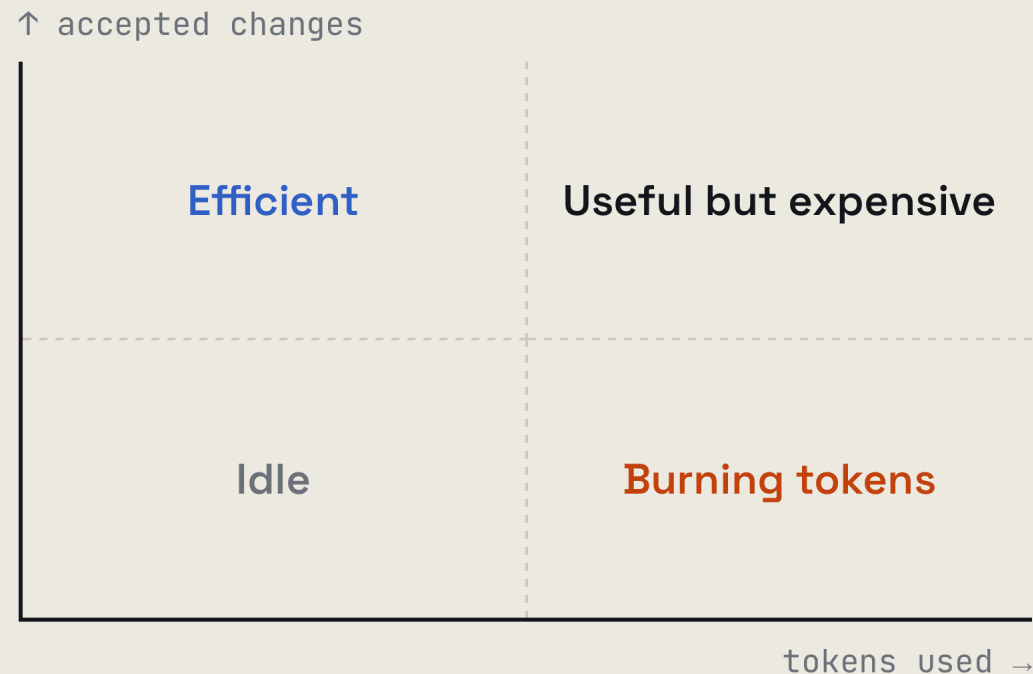
Context tier

Tool set

MCP servers

Throw away your IDE?

Tokens used are not work done



Measure the outcome instead

- ✓ accepted changes
- ✓ defects found in review
- ✓ time from issue to merge
- ✓ retries per task

Usage tells you the cost. It never tells you the value.

03

PART 03

The lean harness

Same packaging. Less ceremony. Built for today's routing, caching and billing.

Keep the packaging, cut the ceremony

KEEP

- ✓ Portable installer and repository layers
- ✓ Narrow specialist agents
- ✓ Tests and binding gates

CHANGE

~~Automatic fan-out~~

→ **An explicit delegation gate**

~~Hard-coded model names~~

→ **Auto routing**

~~Write everything memory~~

→ **Selective memory writes**

Contract, control, evidence

CONTRACT

Repository instructions: goals, constraints, and one verification command.

CONTROL

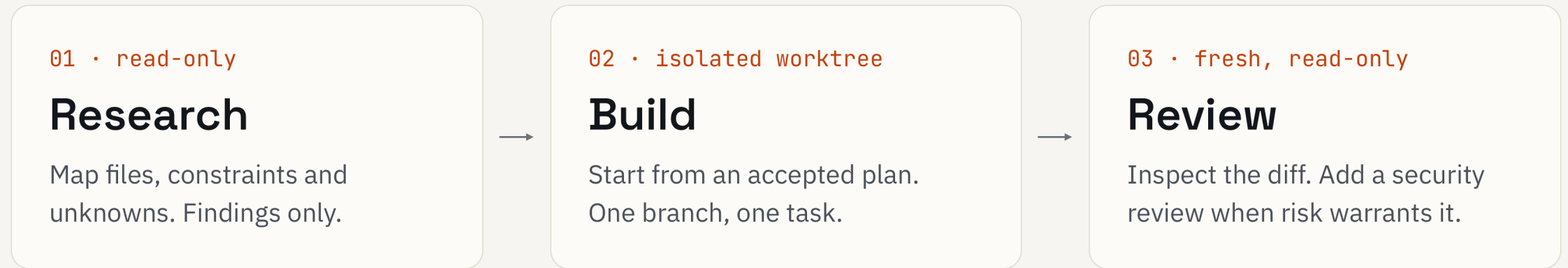
Profiles: routing, reasoning, credit caps, permissions, subagent limits.

EVIDENCE

Deterministic checks first. Then model review for behaviour, ambiguity and risk.

Each layer changes without rewriting the others.

One phase per session



A clean context boundary often beats one session remembering the whole journey.

Choose the budget before you start

Profile	Reasoning	Credit cap	Use for
Quick	low	35	small fixes, docs, triage
Standard	medium	100	most features
Deep	high	250	cross-cutting changes
Security	high	300	auth, payments, personal data

Caps are soft guardrails. Recalibrate them from your own usage after every meaningful run.

The delegation gate

- 01 Can each task change separate files?
- 02 Does each task have a clear output?
- 03 Will parallel work shorten elapsed time?
- 04 Can a cheaper specialist handle it?
- 05 Is the merge and review cost acceptable?

Five yes → delegate. Any no → keep it in one session.

When parallel agents pay off

✓ GOOD FIT

Independent audits.

Check every CMS block for a missing test id.

Multi-model review of one branch.

Agreement means confidence. Disagreement means look closer.

HydraFusion's Critique formalises this: an independent critic from another model family reviews the draft.

Large mechanical migrations.

Hand them to a cloud agent that opens a PR.

⚠ POOR FIT

Several agents, one core module.

They collide, and the merge eats the savings.

Unclear outputs.

Nobody can tell when a worker is done.

Tiny tasks.

The coordination costs more than the work.

Autonomy needs verification by design

Interactive

Confirms before each file change.

Plan

Shows the whole plan before any code.

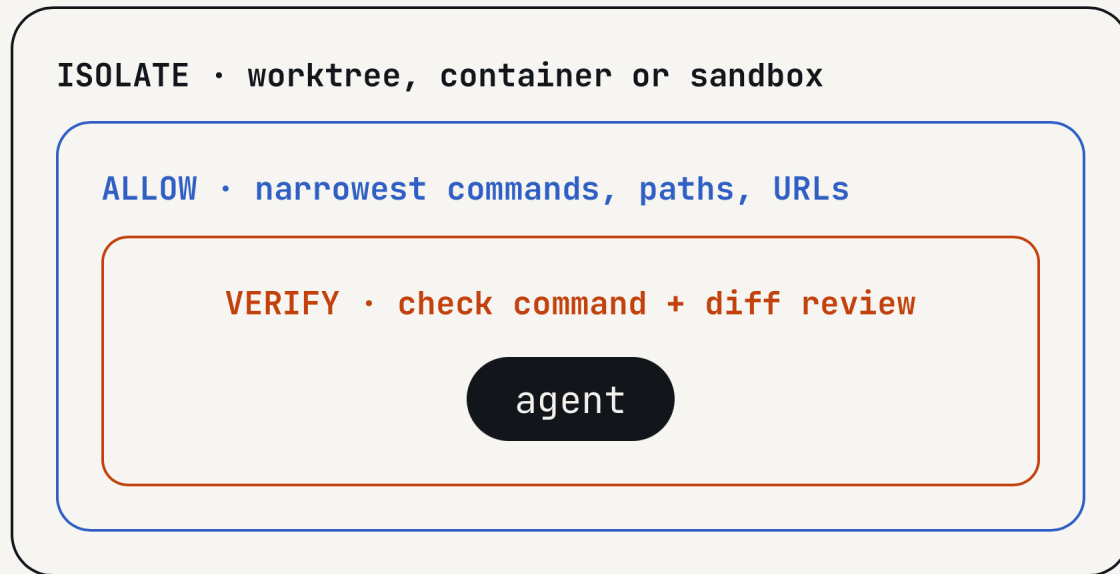
Autopilot

Runs to done on a well-scoped task.

Autopilot is not unchecked. The checks are part of the loop.

Human checkpoints stay mandatory for auth, payments, personal data and infrastructure.

The safety envelope



Freedom inside a small, observable boundary.

Never trade reviewability for convenience. No allow-all, no yolo mode.

CI is the referee



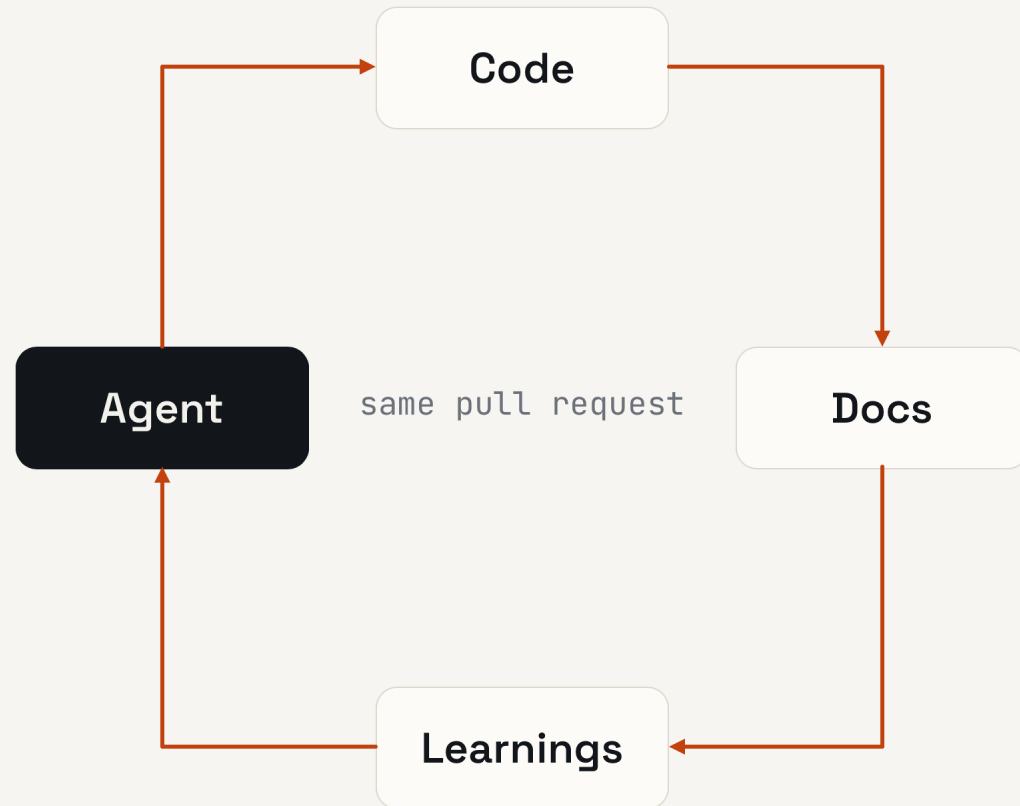
Deterministic checks first

Build, types, lint, tests and a Trivy scan that fails on CRITICAL or HIGH. Models review what tests cannot see.

Same Actions hygiene

- ✓ Least-privilege permissions: contents: read
- ✓ Pin every action to a commit SHA
- ✓ OIDC instead of long-lived cloud keys
- ✓ Timeouts and concurrency on every job

Documentation is part of the system



- Lives next to the code it describes.
- Updated in the same change as the behaviour.
- Feeds the next agent session.
- The PR template asks: did you read it, did you add to it?

The desktop app as control plane

CLI

The reproducible interface

- Scripts and profiles
- CI and automation
- The same commands for everyone

DESKTOP APP

The supervision layer

- Chats for research
- Worktrees for implementation
- Diff view for judgment
- Terminal and browser for validation
- Pull request handoff

Same agent engine. A different job.

Throw away your IDE?

Issue → Build → Prove

01 Issue

A vague request becomes acceptance criteria and a plan.

02 Build

Standard profile, in an isolated worktree.

03 Prove

Run the check command, read /usage, review the diff.

ISSUE

Fix: cart shows NaN after a market switch
→ 3 acceptance criteria plan approved



BUILD

worktree • profile standard
failing test RED → minimal fix GREEN



PROVE

npm run check • /usage • /diff
check PASS review PASS 🔗 FAIL → Build



🧑 A human reads the diff and decides

04

PART 04

People make it real

The model is the easy part.

Two ways to get it wrong



FOMO

Try everything. Finish nothing.
Constant churn.

The balance

Small experiments, shared results,
regular pruning.

Local optimum

Change nothing. Stay comfortable.
Miss the shift.

Tooling moves faster than your process

- 01 Build for the problem, not the implementation.
- 02 Expect the vendor to ship what you built by hand.
- 03 Keep the process elastic: prune as often as you add.

LIVE EXAMPLE

Our hand-built start-cheap-escalate now ships as HydraFusion's Cascade. It is a research preview: measure it against your own baseline.

Already built in: `plan mode` • `/fleet` • `/delegate` • `session limits` • HydraFusion

Throw away your IDE?

Start on Monday

1

One repo

and one real,
recurring task type

2

A small core

instructions plus
one verification
command

3

A baseline

time, retries and
review defects today

4

Skills on demand

add one when
a pain repeats

5

Share and prune

what worked, and
what did not

Small, measured, shared. Then repeat.

CLOSING THOUGHT

The IDE becomes a precision tool.

Start with the problem and its contract. Let the harness run the workflow. Open the editor when direct manipulation is the fastest way to a precise answer.

Thank you. Andreas Heige · Questions?

Monday: one repo · a small core · a baseline · skills on demand · share and prune

... ~/talk · copilot

```
$ copilot "end the talk"
• Plan: 1 step · profile quick
• Slides remaining: 0
• Thank you for listening PASS
• Audience awake: mostly
• Questions: ∞ pending
> Approve applause? [Y/y]
```

INDEX OF SOURCES

Further reading, by slide

Slide	Source	Where
05	Copilot CLI: quickstart and command reference	docs.github.com/copilot
16 · 36	HydraFusion (research preview)	docs.github.com/en/early-access/copilot/hydrfusion
27	Optimize AI usage	docs.github.com/copilot
28	Context management	docs.github.com/copilot
31	The lean harness starter (toolkit)	github.com/andreasheige/toolkit
33	Copilot CLI best practices	docs.github.com/copilot
36	Fleet: parallel agents	docs.github.com/copilot
38	Allowing tools	docs.github.com/copilot
41	Copilot app generally available	github.blog/changelog