

WORKSHOP FOR DEVELOPMENT TEAMS, ON SITE OR ONLINE

Getting real productivity out of AI

Get more done with Claude Code or Copilot, spending fewer tokens and redoing less

One day with Claude Code and GitHub Copilot so the team gets more work done, spends fewer tokens, trusts what it ships to production and works the same way in every repository.

5 to 8 h

on site or online, depending
on the team

6

labs with measurement be-
fore and after

2

tools: Claude Code and
GitHub Copilot

8

advanced syllabus blocks

WHERE TEAMS START

The whole team uses AI, each person their own way, and the bill shows

One person uses the most expensive model to rename a variable. Another has had a session open since Monday that resends its whole history with every message. Another leaves the agent looping all afternoon on a test that can't pass. And pull requests arrive green because the same agent that wrote the code decided it was fine.

The tool isn't the one at fault. Claude Code and Copilot come with turn limits, spending caps, hooks, subagents with their own model and reviewers in a clean context. Almost nobody sets them up, because they're scattered across the documentation, the changelogs and the repository issues.

This day is about those pieces, the ones that decide whether the assistant saves work or makes more of it. Each one gets measured in the team's real sessions, before and after, and the saving comes from those numbers.

What the team takes away from the day

01 More work finished

The expensive model only where real thinking is needed, and agents working in parallel without stepping on each other.

02 Fewer tokens

A cache that doesn't break, context loaded only when it's needed and output that doesn't fill the window.

03 Sessions that finish

A turn limit, a spending limit and a stop condition that works even when nobody is watching.

04 Confidence in production

The AI tests what it generates, and another agent tries to break it before it reaches the merge.

05 A shared method

The same configuration and the same rules across every team.

I teach everything in Claude Code and in GitHub Copilot. By the end of the day, everyone has it set up in their own repository.

WORKSHOP RESULTS

Eight things the team will know how to do on Monday

We practise each one during the day with measurement, and in the last block it ends up set up in the team's own repository.

01

Choose models with data and pay for the expensive one only where it thinks

Compare models on the team's own tasks instead of trusting the leaderboard. Plan with the large model and execute with a cheaper one, subagents that explore with Haiku, and a different reasoning effort for each task.

Evidence: a table with three models on the same task, with passing tests, cost and time for each.

02

Spend fewer tokens without losing quality

Know what breaks the cache, why rewinding is cheaper than compacting and how to filter tool output before it reaches the model.

Evidence: input, output and cache tokens measured in each participant's session.

03

Give context only when it's needed, without RAG

A short instruction file that works as an index, rules that only load when someone touches their path, and documentation the agent finds on its own.

Evidence: a sentinel rule that shows which files each tool really reads.

04

Leave the agent working and make it finish

A turn and budget limit per session, a stop condition checked by another model and the Ralph loop: the same brief in rounds with a clean context, progress stored in the repository and an iteration cap.

Evidence: an autonomous task that stops by itself when it meets its goal or hits its limit.

05

Cut infinite loops

Spot the loop before it burns an afternoon, and set up what cuts it even when nobody is watching.

Evidence: a loop caused on purpose and cut by the configuration, with nobody stepping in.

06

Trust what goes to production

Have the AI test its own code: a test that fails first, tests protected against agent edits and a suite that runs before anything is declared finished.

Evidence: an agent that tries to cheat on the tests and a configuration that stops it.

07

Have agents watch each other

A reviewer in a clean context, cross-validation with different models, a judge with controlled biases and an adversarial agent that exists only to break things.

Evidence: a panel that finds planted flaws the author considered fine.

08

Work the same way across the team

Managed configuration for the whole organisation, an in-house catalogue of skills and agents, and one folder that both Claude Code and Copilot read.

Evidence: the same configuration working in the team's two tools.

Five documented failures that cost money

They show up in the tools' own repositories and guides, and they appear as soon as an agent works on a real project.

ENDLESS LOOP

The agent retries the same thing for hours

There is a public Claude Code issue with more than 300 consecutive retries, 27 million tokens and 4.6 hours of execution. Nobody stopped it because nobody was watching.

CONTROL PRACTISED

A turn and spending limit per session, a stop hook with a guard and a cap on continuations.

EXPENSIVE MODEL FOR EVERYTHING

The most powerful model searches for files and renames variables

Exploring a repository doesn't need the model that designs the architecture, but it gets it because it's the one selected.

CONTROL PRACTISED

A model per round: the large one plans and reviews, a cheap one explores and executes, each subagent with its own model pinned.

BROKEN CACHE

The same session costs double and nobody knows why

Switching model mid-conversation, turning on fast mode or compacting invalidates the cache, and the whole history is paid for again at the full rate.

CONTROL PRACTISED

Know what breaks the cache, measure it in the status bar and rewind instead of compacting when possible.

FALSE GREEN

The tests pass because the agent changed them

Faced with a test it can't get to pass, the agent relaxes the assertion, marks it as skipped or makes the function return exactly what the test expects.

CONTROL PRACTISED

Tests protected by a hook that blocks editing them, red before green and a reviewer who hasn't seen the author's conversation.

EVERYONE WITH THEIR OWN SETUP

Ten developers, ten ways of working with agents

Everyone has their own permissions, prompts and default model. Nobody knows what works or what it costs.

CONTROL PRACTISED

Managed configuration, a shared catalogue of skills and agents and a folder shared by both tools.

What's covered, block by block

I explain each block in Claude Code and in GitHub Copilot. I go through the basics quickly, because the team already uses the tools every day, and the time goes on what almost nobody configures. With five hours, blocks 6 to 8 get condensed.

01 Diagnosis: measure before optimising

First we measure every session: tokens, cache and cost.

- What fills the window and what the session costs: /context, /usage and a status bar with live cost and cache.
- Cost per run in non-interactive mode, reading the JSON that claude -p returns.
- How long the cache lasts depending on how Claude Code is used, and why that changes when a pause is worth taking.
- In Copilot: the current credit system, what consumes it and what is free, such as autocomplete.

Real case: Claude Code's cost documentation gives an average of about 13 dollars per developer per active day. The first exercise is measuring where the team stands.

02 Which model to use and how to choose it

The top of the leaderboard isn't necessarily the best for a given repository. The model gets chosen by measuring.

- Which models each tool offers: Claude Code ships Anthropic's; Copilot, Anthropic's, OpenAI's and Google's, each with its own multiplier.
- The criteria that count: quality on the team's tasks, cost of the whole task rather than price per token, context window, latency and reasoning effort.
- An in-house evaluation: real backlog tasks, two or three models and a table of passing tests, cost and time. Every new model runs the same battery before it replaces the default.
- Plan with the large model and execute with another, subagents with their model pinned and Claude Code's built-in explorer running on Haiku.
- In Copilot: agents with a fallback model list and the auxiliary model nobody configures.

Real case: A cheap model that needs three times the turns costs more than an expensive one that gets it right first time. And fast mode doesn't save money: it costs more per token, so it is for when latency matters.

03 Fewer input and output tokens

Most of the spend is in what gets resent with every message.

- What invalidates the cache: switching model or effort, fast mode and compacting.
- Rewinding versus compacting, and when to start from scratch with a written summary.
- Hooks that filter command output before it reaches the model.
- MCP tools on demand, and in Copilot the per-request tool limit and how to shorten the list.

Real case: A hook that trims test output down to the failing lines leaves the same information at a fraction of the tokens.

04 Context only when it's needed, without RAG

It's enough for the agent to know where everything is.

- An instruction file under two hundred lines that works as an index, and rules that load only when their path is touched.
- What actually loads: imports are not lazy and HTML comments never reach the model.
- Documentation organised in levels that the agent walks on its own, with no vector database.
- In Copilot: path-scoped instructions, and which Claude Code files it also reads, checked with a sentinel rule.

Real case: If CLAUDE.md exists, Claude Code does not read AGENTS.md. A sentinel line proves it in ten seconds.

05 Finite execution: sessions that end

An agent with no limit gets expensive, and it isn't really autonomous.

- A maximum of turns and budget per run, with a different exit code for each cut-off.
- The Ralph loop: the same brief in successive rounds, each with a clean context, progress written to repository files, a completion promise and a maximum number of iterations.
- Goals with a stop condition evaluated by another model, and stop hooks with their guard against looping.
- In Copilot: the agent mode request limit, autopilot with a cap on continuations and the cloud agent limit.

Real case: The Ralph loop works because each round starts without the previous one's history. With no iteration cap and no definition of done, it is the fastest way to burn tokens.

06 Orchestrating groups of agents

What works is different agents with briefs that don't overlap.

- A worktree per agent, batch work across many files and one session per pull request.
- An expert panel with disjoint roles and cross-validation between different models.
- A judge with controls for position, length and self-preference bias, and an adversarial agent with a single brief: break it.
- When not to build a multi-agent system, and in Copilot the handoffs, the CLI agent fleet and the cloud agent.

Real case: Anthropic measured its multi-agent research system: 90% better performance in exchange for about fifteen times more tokens. It only pays off on certain tasks.

07 Have the AI test its own code

A change is finished when a check confirms it.

- A test that fails first, on the expected line, and a hook that stops the agent editing the tests.
- A stop gate that runs the suite and won't let the task close with anything red.
- A reviewer in a clean context, automatic security review and mutation testing to know whether the tests prove anything.
- In Copilot: the cloud agent validates with CodeQL, and the terminal's auto-approve is not a security boundary.

Real case: A model that self-corrects without external information doesn't improve its reasoning. That's why the reviewer can't share context with the author.

08 A shared methodology across all teams

What works in one repository has to work in all of them, regardless of who configured it.

- Managed configuration for the organisation: allowed models, dangerous modes disabled and shared permissions.
- An in-house catalogue of skills, agents and hooks that installs the same way in every repository.
- A single configuration folder read by both Claude Code and Copilot, and per-team telemetry.
- In Copilot: which organisation instructions don't reach the IDE and where content exclusions don't apply.

Real case: Copilot's content exclusions don't apply in agent mode or in the CLI. If anyone is counting on them, better to find out today.

DELIBERATE PRACTICE

Six labs, each measured before and after

Each exercise runs on a practice repository I prepare for the day, in Claude Code and in Copilot. Everyone notes their numbers before and after, and the saving comes from there.

01

Choosing a model by measuring

The same task is solved with three different models, and then with planning, exploration, execution and review spread across models.

Result: a table of passing tests, cost and time for each option, which the team reuses as the battery for the next model.

02

The cache and the output

The cache is deliberately broken in three different ways and each one is measured. Then a hook is added that filters command output.

Result: cache, input and output tokens measured, and a hook the team can copy.

03

Context on a diet

A long, contradictory instruction file is turned into an index with path-scoped rules. A sentinel rule shows what each tool reads.

Result: less context loaded and a rule the agent now follows that it used to ignore.

04

Making it finite

An autonomous task is launched with no limits and falls into a loop. It is repeated as a Ralph loop, with progress in a file, an iteration cap, a spending limit and a stop condition evaluated by another model.

Result: a run that ends by itself, with the cost of both versions compared.

05

Catching the cheat

The repository comes with planted flaws. The agent is asked to make the tests pass and we watch how it tries to do it. Then the tests are protected and a stop gate is added.

Result: the planted flaws surface and the agent can no longer declare the task done while red.

06

Panel and judge

Three agents with different roles and different models review the same change. A judge decides, and an adversarial agent tries to break what everyone approved.

Result: flaws found that the author considered fine, and a ready-to-use panel template.

ON THE COMPANY'S CODE

We do the last part on the company's repository

At the end of the day everyone opens their laptop and sets up their own repository with what we practised. I go round answering questions and reviewing decisions when someone asks.

01

Measure the starting point

Context used, the cost of a normal session and cache usage, with the status bar we built during the day.

02

Instructions, models and limits

An instruction file as an index, subagents with their model pinned and turn and spending limits for whatever is launched autonomously.

03

A real task with its gate

A small item from the team's own backlog: a test that fails first, protected tests, the suite before closing and a reviewer in a clean context.

Measure again and write it down

The same numbers as at the start, noted next to a short list of what the team wants to standardise.

Everyone works on their own laptop with their usual tools. The code never leaves the company, and I don't take it or copy it anywhere. It isn't an audit either: each person decides what they show.

Between 5 and 8 hours, depending on the team

The length adjusts to the team and to what hurts it most. The order doesn't change: measure, save, bound, verify and apply it to the team's own repository.

Block 1	Measure and choose a model Diagnosis of each participant's session, how to choose a model, and which model and effort to use in each round. Lab 1.
Block 2	Fewer tokens and context only when needed Cache, filtered output, instructions as an index and path-scoped rules. Labs 2 and 3.
Block 3	Sessions that end Turn and spending limits, the Ralph loop, stop conditions and how to cut a loop. Lab 4.
Block 4	Verify and orchestrate Have the AI test its code, protected tests, an agent panel, a judge and an adversary. Labs 5 and 6.
Block 5	The company's repository Each participant measures, configures and measures again their own project. We close with what the team wants to standardise.

MATERIAL THAT STAYS WITH THE COMPANY

What stays in the repository

All the material is made to be copied into the repository and kept there. If it ends up unopened in a training folder, I did it wrong.

Status bar with cost

Session cost, context used and cache in view at every moment.

Ready-to-use subagents

A cheap explorer, a clean-context reviewer, an adversary and a judge, with model and turn limit pinned.

Instruction templates

An index file, path-scoped rules and the sentinel rule, in a folder both tools read.

Agent brief template

What to do, what not to do, turn limit and report format.

Base configuration for the organisation

Allowed models, shared permissions and dangerous modes disabled, to start standardising.

Repository with six labs

Briefs, starting files, planted flaws and tests to repeat each exercise.

Example hooks

Output filter, test protection and stop gate, for Claude Code and for Copilot.

Finite-execution recipes

The Ralph loop with its progress file, turn and spending limits, stop conditions and autonomous-mode caps in both tools.

Model evaluation battery

The tasks and the table from lab 1, to test every new model before changing the default.

WHAT COMES WITH THE COURSE

Four things that stay with the team, at no extra cost

They come with the workshop. They're what teams end up asking for afterwards.

+1

A 10-minute survey beforehand

Before the day, the team answers a short survey about its stack, its tools and where its time goes. I use it to adapt the labs to what the team really uses.

+2

The team's numbers

At the start, each participant writes down three numbers from one of their normal sessions: tokens, cache percentage and cost (in Copilot, premium requests). They write them down again after each lab and at the end of the day, in a shared template I bring ready. That week I send one page with the team's numbers: where the spend was going and what has changed.

+3

Configurations on a branch of the company's repository

During the final block I leave the configurations we tested on a branch: instructions, subagents, hooks and limits. The team reviews and merges them like any other change.

+4

Labs with solutions

The lab repository with its solutions and a 12-month internal-use licence, so whoever joins the team later doesn't have to wait for us to repeat the day.

WHO IT'S FOR

For teams that already code with agents

A strong fit for

- Developers who work daily with Claude Code or GitHub Copilot and feel the tool delivers less than it should, or costs more.
- Tech leads who want the team to work with agents one way, and not one way per person.
- Engineering managers who want to control token spend without slowing the team down.
- Platform teams that maintain the configuration, permissions and agents for the rest of the organisation.
- Teams already launching agents in parallel or autonomously and starting to see loops, mixed-up commits or pull requests that don't hold up.

Not the right workshop if

- The team doesn't yet use a coding assistant on a real repository.
- The company wants an introduction to LLMs or a prompting course.
- The goal is to compare tools to choose one. The workshop starts from Claude Code and Copilot.
- What is expected is an audit of the code. In the final block everyone works on their own repository, but I don't do the review.

From cybersecurity at BBVA to AI at Santander



I'm Daniel Alfocea, and I've spent twenty years in development and security. At Banco Santander I was Principal Engineer in the AI department while the area was being set up. At BBVA I was Cybersecurity Tech Lead. At 42Crunch, in the UK, I was responsible for one of its API security products.

I've presented research at RSA Conference and OWASP AppSec EU, and I'm a regular speaker at RootedCON. Spain's National Cybersecurity Institute (INCIBE) has asked me to judge its competitions several times. I founded Navaja Negra and the OWASP Madrid chapter. I'm the author of widely used security tools such as GoLismero, Plecost and Dockerscan, and of LAPIS, research published on arXiv on how to describe an API to a model with fewer tokens. With Alfonso Muñoz I wrote *Secure MCP: A practical guide for developers, software architects and tech leads*.

What I teach here is what I use in my own projects. Each one has its documentation index, and I split the work across agents, each with its own model, turn limit and report. An important conclusion doesn't get accepted until another agent checks it along a different path.

GUARANTEES

Two commitments in writing, and one limit

If the configurations don't work, I fix them

If the configurations I leave in the company's repository don't work, I run a free remote session to fix them.

Stuck after 30 days, one free hour

If after 30 days the team has got stuck applying something from the course, I spend an hour with the team remotely, free.

This isn't an audit and I'm not promising results: I don't guarantee saving percentages. The saving gets measured during the day, with the team's sessions.

What teams ask me before booking

How many tokens will we save?

I don't give a figure before seeing the team's sessions. During the day each participant measures their consumption before and after each lab, so the number comes from their own work.

How much does it cost?

It depends on the length and the size of the team. We talk it through in the first conversation and then I send a fixed proposal.

Does everyone need to use the same tool?

No. Each participant needs Claude Code or GitHub Copilot, already signed in and able to work on a local folder. I explain everything in both, and much of the configuration works for both at once.

Isn't it too basic if we already use them every day?

The workshop starts from there. I go through the basics quickly and the time goes on what almost nobody configures: model per round, cache, execution limits, the Ralph loop, hooks, agent panels and managed configuration.

Can we use our own repository during the day?

Usually yes, in the final block. The earlier labs use a practice repository so the whole group sees the same failure; after that, everyone applies what they learned to their own code, on their own laptop. If the company can't open its code that day, we work on a practice repository similar to its stack.

Does our code leave the company?

No. We work on the company's laptops and accounts. I don't take or copy any code, and whatever each tool sends to its vendor is the same as on any other day.

Can it be adapted to our language or stack?

The labs use Python, but the concepts work for any language. We do the final block on each participant's own stack, whatever it is.

Can it be funded through FUNDAE?

When a date is requested, I send the information the training department or the advisers need.

Can it be done online?

Yes, with the same syllabus, the same labs and the same measurement. It runs live and we decide together how to split it into sessions. We do the final block on the company's repository by sharing screens.

What if we need follow-up?

There's a 6-week programme on the same topic, with measurement before and after and review of the team's PRs. The details are on [the programme page](#).

FORMAT

One team, one day and a shared way of working with agents

It includes the preparation, a day of five to eight hours and all the working material. There are two ways to run it. We settle the length and the group size in the first conversation, so there's time to go through the labs and to help in the final block, in each person's repository.

On site

At the company's office, everyone in the same room.

- Room with a screen, wifi and one laptop per participant.
- Travel outside Madrid agreed before the date is confirmed.

Online

Live over video call, with the same programme and the same labs.

- Each participant with their laptop, camera and microphone.
- No travel, so it works for teams spread across several cities.

- Participants from the same team or from teams that work together.
- Delivered in English or Spanish.
- Includes the team's numbers with their summary page, the configurations on the company's branch and the two commitments above.
- It is designed as a working day for a team that will apply what it learns. It is not an audit or a talk to sit through.

Write to me at

me@danielalfocea.com

In the first conversation we check whether the workshop is a fit, which tools the team uses, how many people take part, how long it lasts and which date suits the company. With that I prepare the proposal.

Can be subsidised through FUNDAE (Spanish training credit).

EMAIL

me@danielalfocea.com

PROGRAMME ONLINE

danielalfocea.com/en/formacion/workshop-productividad-con-ia