Nearform

From first commit to self-improving agent

Lessons from running Claude in production

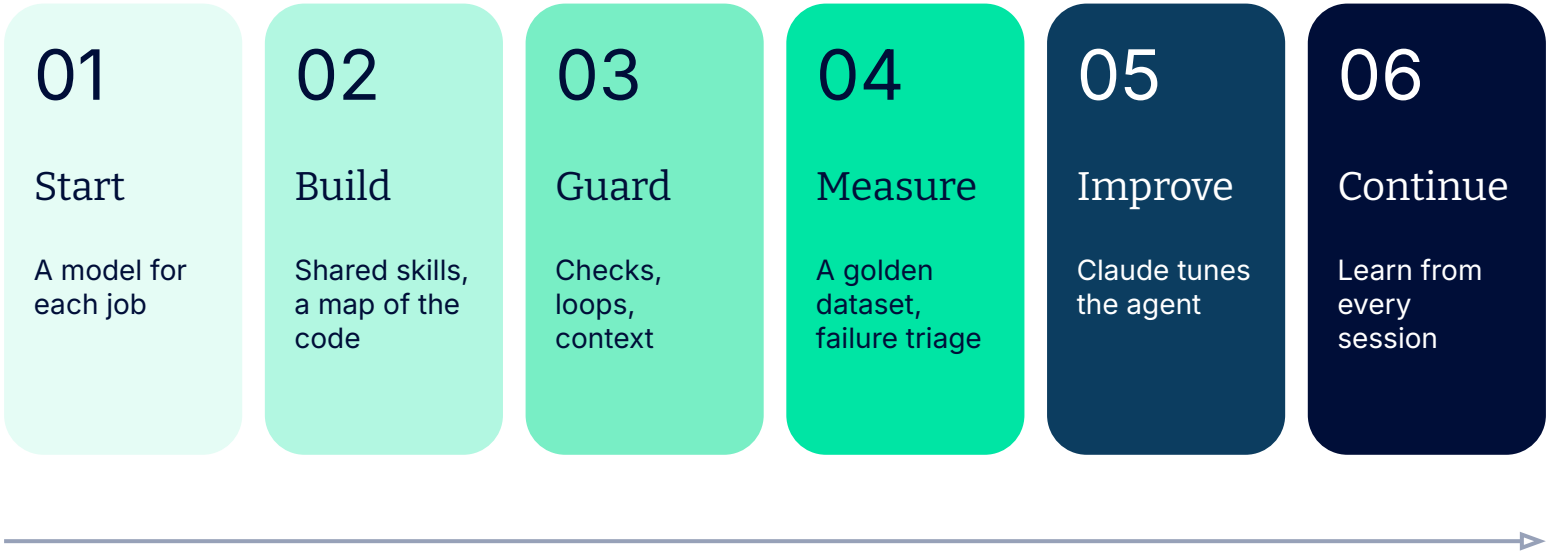
Danilo Poccia, AI Solutions Director

Claude Operators Summit London · 19 November 2026

© Copyright 2026 Nearform Ltd. All rights reserved.

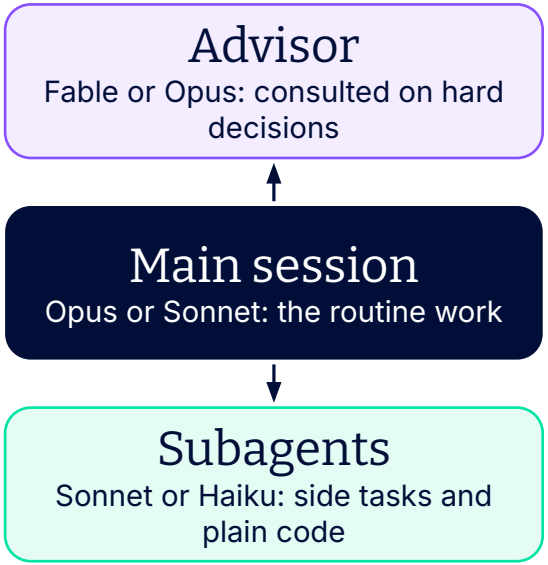
Follow one agent through six stages_

From one engineer's setup to an agent that learns from production.



Pick a model for each job_

Sonnet does the work, Opus advises, Haiku takes the subagents.



/advisor opus
the advisor

/model sonnet
the main session

"Use Haiku 5.5 on coding subagents"
said to Claude Code, or set CLAUDE_CODE_SUBAGENT_MODEL

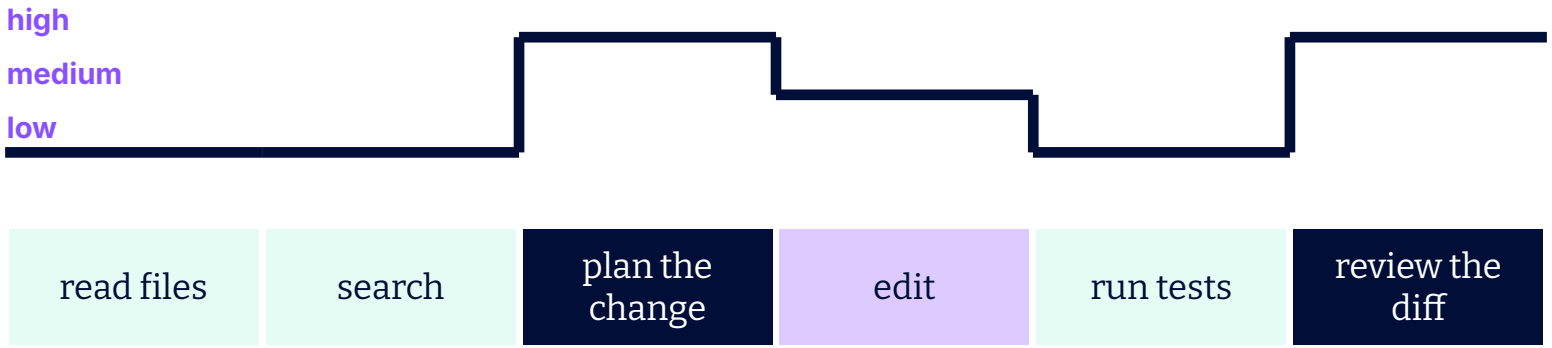
11.9%
cheaper per task

+2.7 pts
SWE-bench Multilingual

Anthropic's benchmark: Sonnet with an Opus advisor, against Sonnet alone.

Set effort per message_

Low to read and run, medium to edit, high to plan and review.



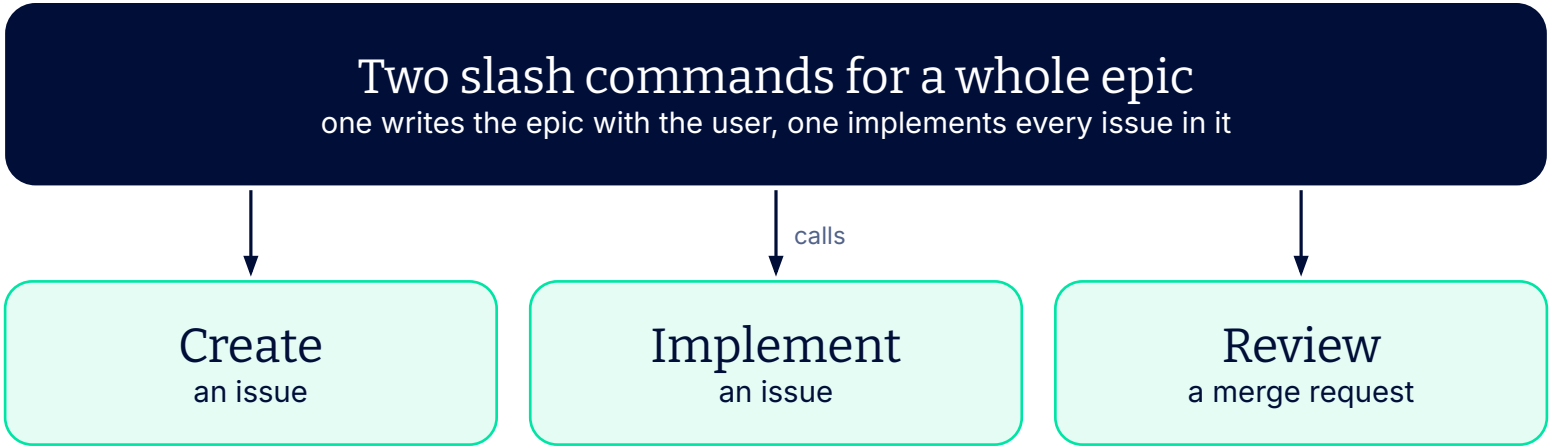
The prompt cache survives: the level changes inside the conversation, not in the request settings

```
{"role":"system","content":[],"output_config":{"effort":"high"}}
```

API beta: a system message sets the level from the next turn on

Skills in layers_

Skills for one issue at the bottom. Commands for a whole epic on top.



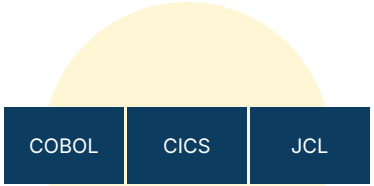
"After weeks of fine-tuning, one slash-command implements the whole epic mostly without interruptions."

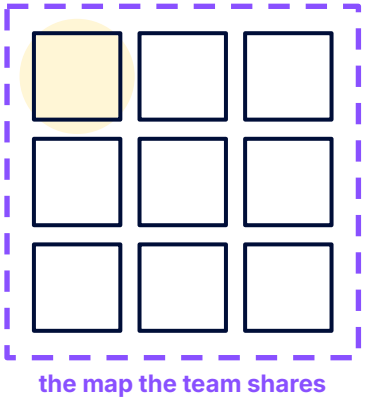
A team of ours at a media customer

Map the codebase by size_

A flashlight is enough for a small codebase. A large estate needs a map first.







A flashlight is enough
one repo: plain Claude with a reading prompt

Bring a guide
COBOL and mainframe: the code-modernization plugin

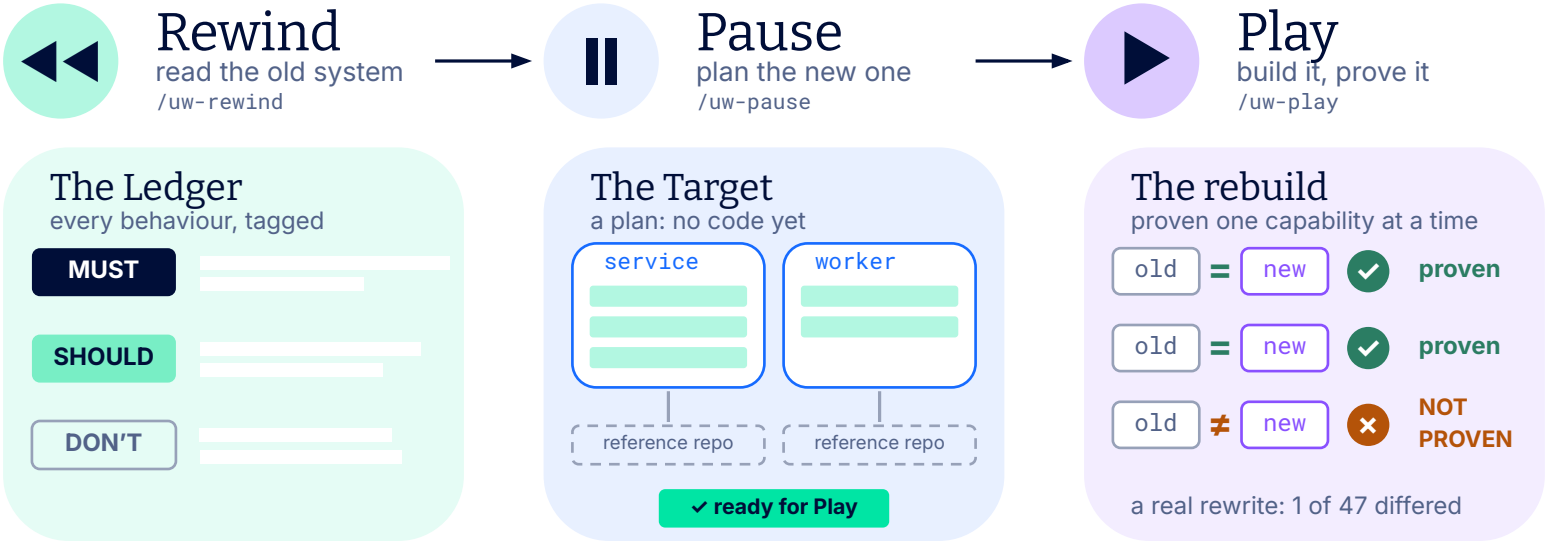
Map first
many repos: a shared, deterministic inventory (Unwind)

Whatever the size, list what the code can't answer, and have the system's owner answer it first.

Blog: Every October, the bug came back · nearform.com/insights · Nov 2026.

Unwind: rebuild a legacy system and prove it matches_

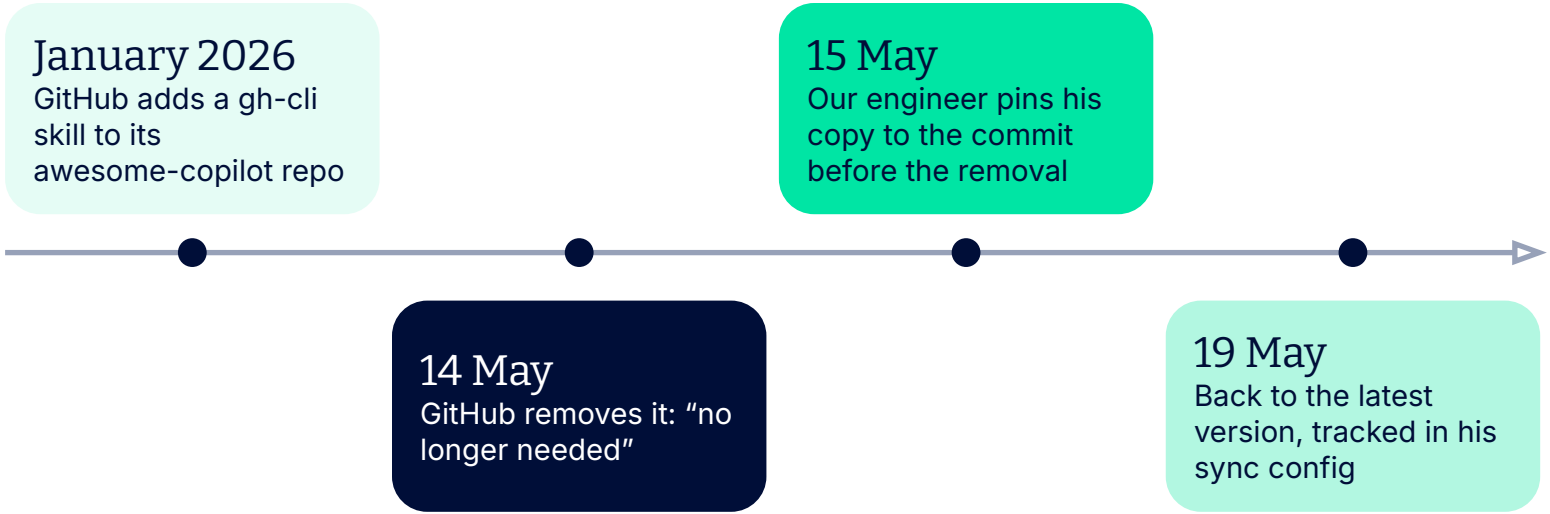
The engine owns facts, agents own meaning, people own decisions.



coverage = scan − docs · the rebuild reads only the Ledger: what it can't carry is reported as a hole

Skills can vanish upstream_

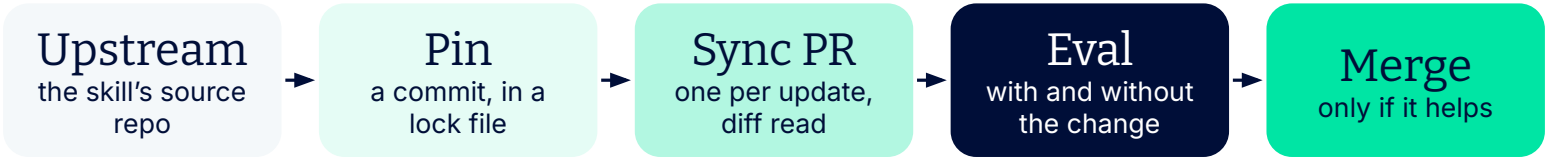
A skill from someone else's repo is a dependency, so pin it.



Pin the exact commit in a lock file, so nothing changes until you choose to update.

Read every skill update_

Take every update as a pull request, read the diff, and eval it.



One instruction from a design skill update

– ~~"state your choice"~~

\+ "confirm with the client"

An agent that runs unattended can now stop and wait for an answer that nobody gives.

No test fails on a change like that.

Sort every guardrail_

Sort checks by timing and cost, then use the cheapest that works.

	Steer: before the work	Check: after the work
Code milliseconds, a fraction of a cent	**Correct by construction** templates, schemas, structures	**Gate** lint, types, tests
Model seconds, cents	**Guidance** skills, instructions, worked examples	**Judgement** an LLM as judge, a refuter review
Person minutes, pounds	**Direction** the spec, the owner's answers, approving the plan	**Review** reads the diff before it merges

Feed the error back_

A gate blocks the merge and tells Claude what to fix.



One script, three places
hook · every edit
pre-commit · every commit
CI · every merge

A reviewer's comment on a customer report

"This is wrong."

which rule?
where?
what should it be?

The linter's error (illustration)

Claude can act on this.

```
rule    heading-3-size
where   page 12, "Results"
fix     14 pt → 12 pt
```

Order the gate from fast to slow_

Anything a machine can check becomes a gate.

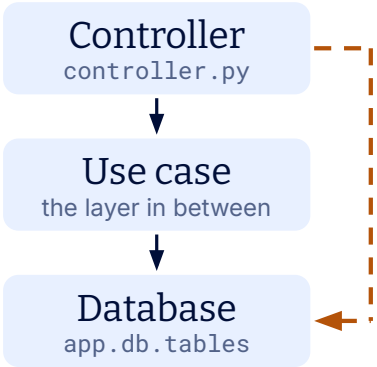
	Python	TypeScript
1 Lint and style	Ruff	ESLint
2 Types	ty	tsc
3 Dead code	vulture	Knip
4 Complexity, with a threshold	Ruff complexity	ESLint complexity
5 Duplication	jscpd	jscpd
6 Security rules	Semgrep	Semgrep
7 Secrets	Gitleaks	Gitleaks
8 Dependencies	pip-audit	npm audit
9 Tests, with a coverage threshold	pytest + coverage	Vitest + coverage
10 Architecture	PyTestArch, pytest + ast	ArchUnitTS, ts-morph

In the order it runs, fast and cheap first. Most of these run in our own pre-commit gates.

Passing tests can hide a wrong design_

App tests check what the code does.
An architecture test checks how it's built.

CLAUDE.md, with a wrong rule planted on purpose
"Always call the database directly from the controllers."



Claude's shortcut
it followed the rule: the controller queries the database itself

the design: controllers reach the data only through use cases

The same change, two checks

App tests
● **passed**
They check what the endpoint returns. It returned the right transaction.

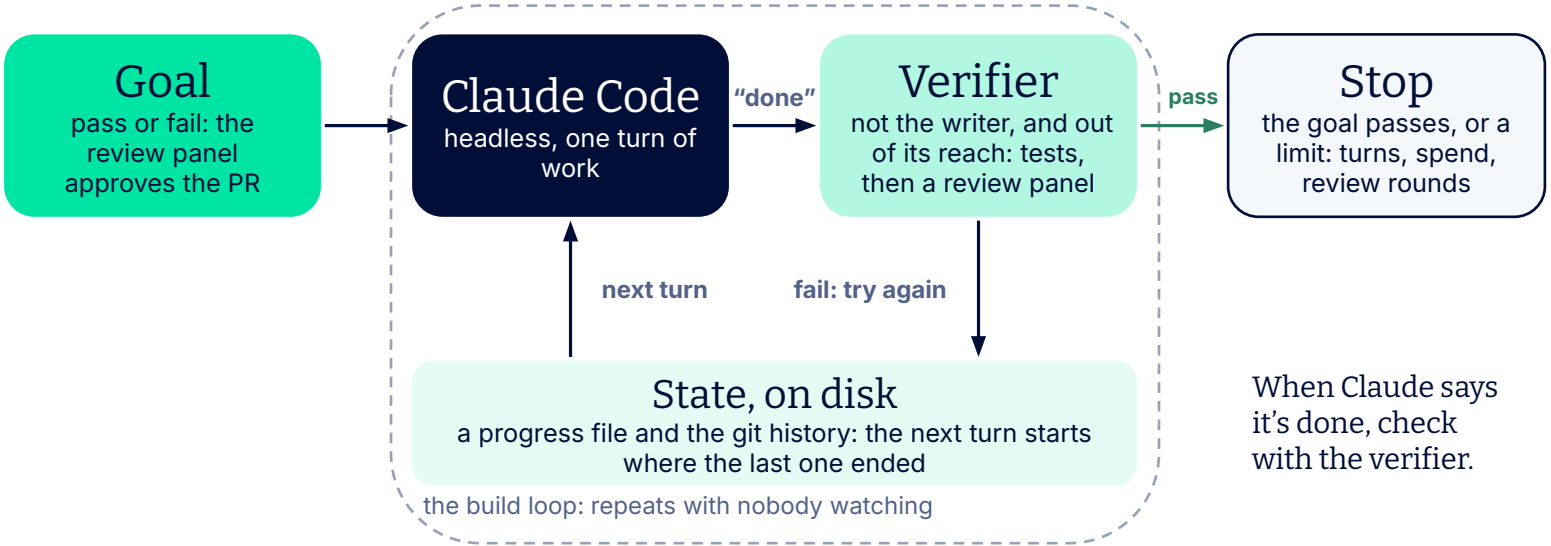
Architecture test
● **failed**
It reads the imports: a controller may use app code only through a use case.

pytest + Python's ast (abstract syntax trees) module

Write-up: Guardrails-driven agents: executable coding standards · nearform.com · Nov 2026 · The diff: pull request #18, nearform/guardrails-driven-agents

The build loop needs four things_

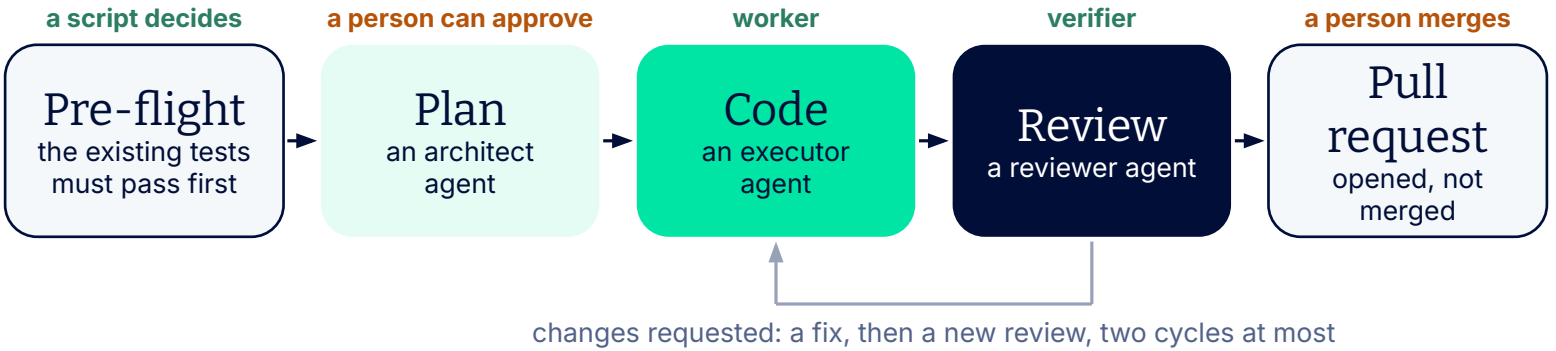
With nobody watching, a loop needs a goal, a verifier, state and a stop rule.



When Claude says it's done, check with the verifier.

Last Light: our open source software factory_

It runs the build loop, from issue to pull request.

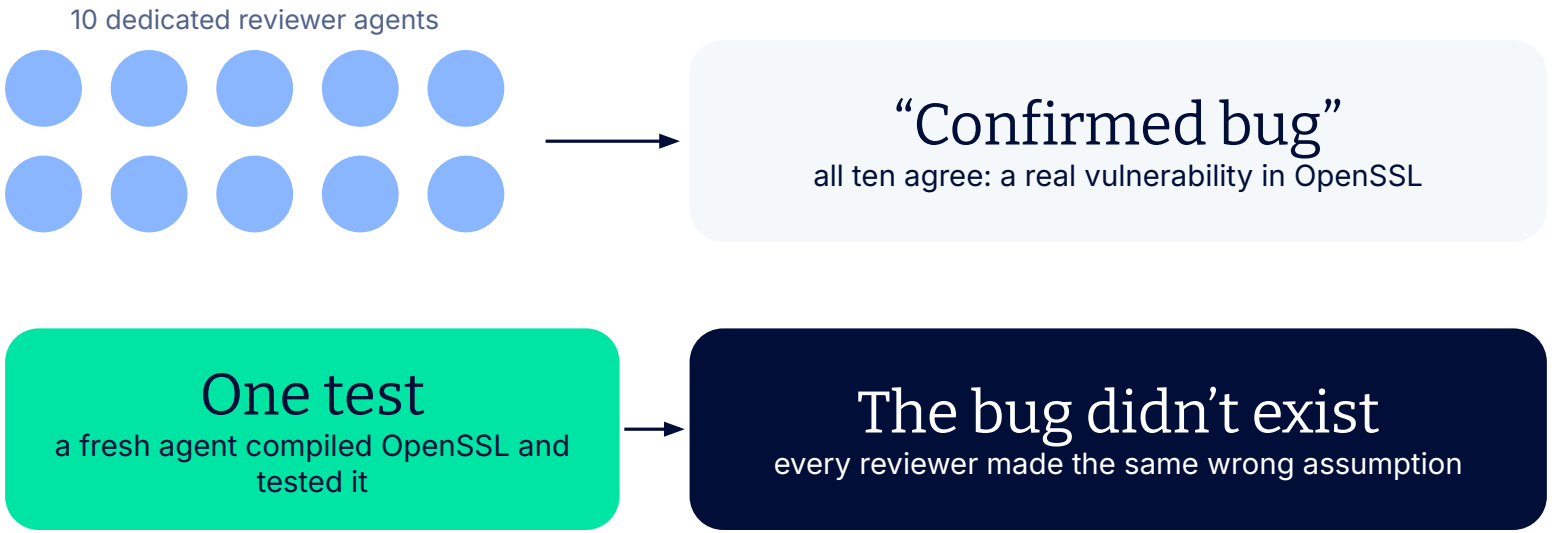


When it picks up issues on its own, it stops for a person twice by default: after the plan and after the review.

Last Light · github.com/nearform/lastlight · MIT licence

Reviewers can all be wrong together_

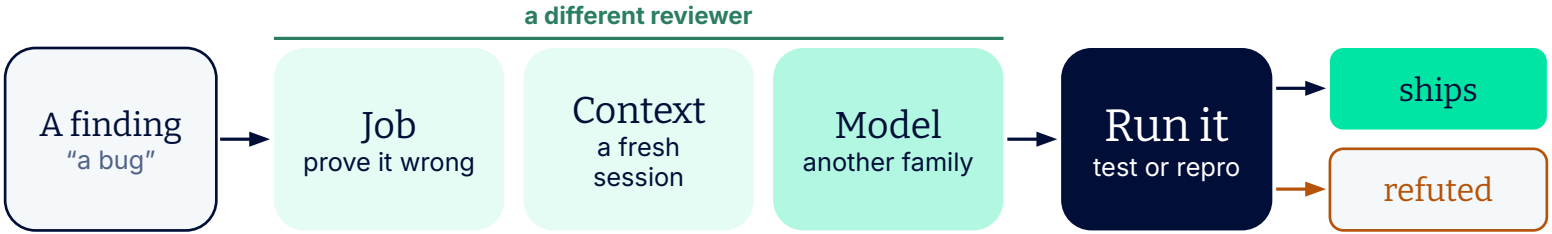
Ten model reviewers agreed on a bug. One test showed it didn't exist.



Source: Agarwal, Refute-or-Promote (arXiv 2604.19049, April 2026)

Change the reviewer_

A different reviewer catches more. A test decides.



3 of 19

proposed fixes had problems only a reviewer from another model family caught

for prose, the test is the source passage, quoted

Limit the review rounds

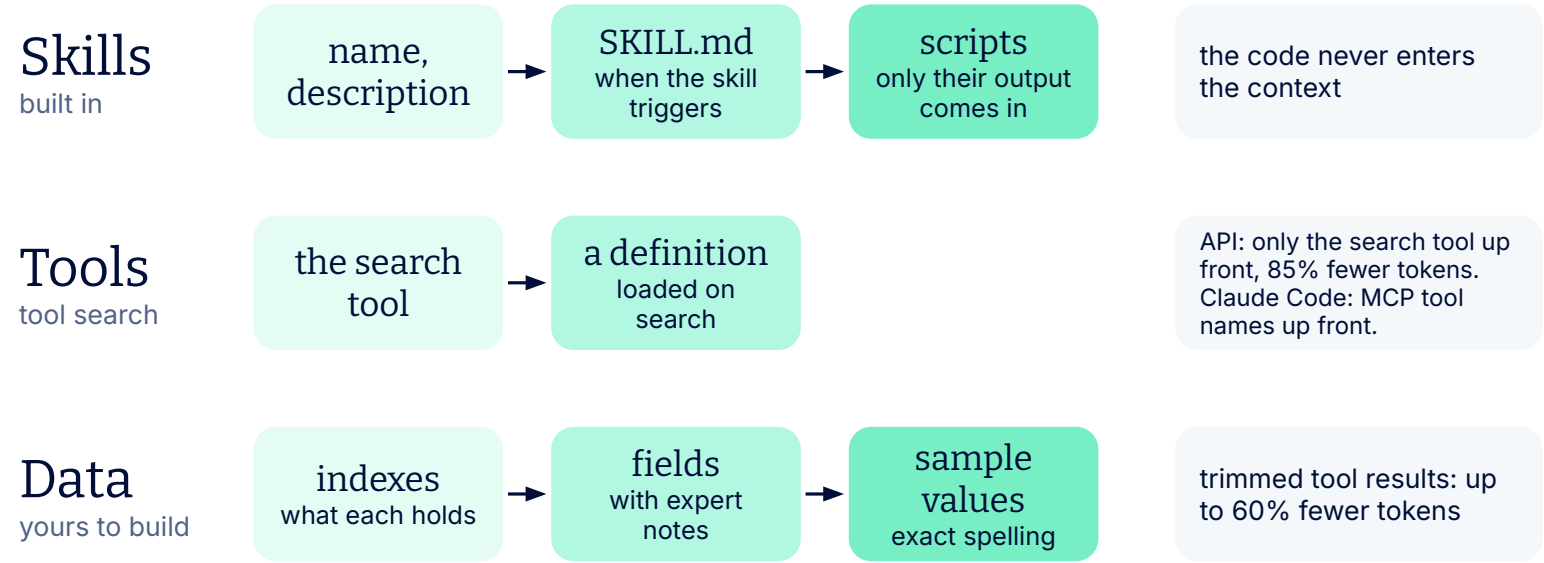


A newer model sent one customer's pull request review round in circles, and a study found that more rounds add noise.

Source: Song, More Rounds, More Noise (arXiv 2603.16244, March 2026)

Load context on demand: progressive disclosure_

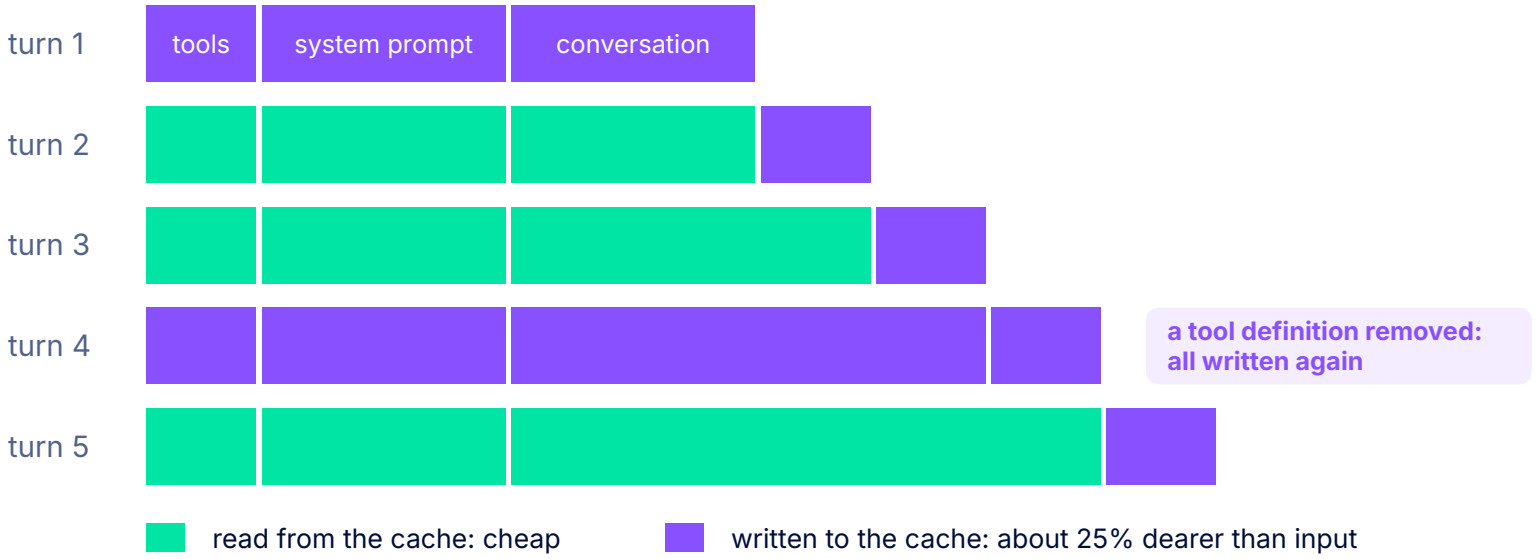
Show a short index first. Load the detail when the next step needs it.



Source: Chroma, Context Rot.

Keep the start of the context fixed_

Change the front of the cache, and everything behind it is written again.



Tool search adds definitions to the conversation and leaves the tool list alone.

Cut the cost of each answer_

Measure the tokens behind every answer, then cut where the numbers point.

The two counters

cache_read_input_tokens
should climb

cache_creation_input_tokens
should stay low after the first turn

$2.40 → $0.50
the expensive answers (95th percentile), almost five times cheaper

What did it, together, in one production system

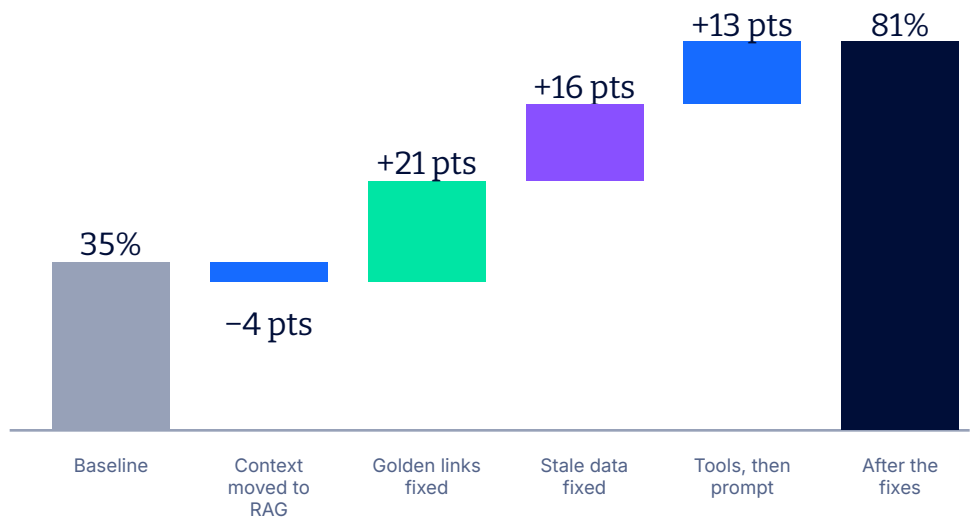
- caching
- the smallest model that passes the evals
- smaller tool outputs
- a sandboxed shell to filter data

Write-up: Cutting an AI agent's cost by 70%, without losing accuracy · nearform.com · Oct 2026

Agent, eval or data?_

When an eval fails, there are three suspects.

Link accuracy of one production agent



Agent
Is the agent wrong? −4 and +13 pts

Eval
Is the test wrong? +21 pts

Data
Is the source data wrong? +16 pts

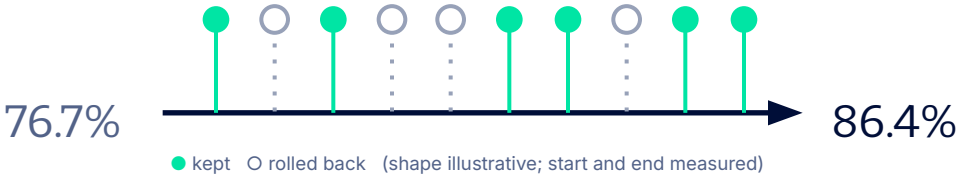
¾ of the climb touched no agent code
+37 of the +50 points, 31% → 81%

Write-up: From AI prototype to production: how to build evals for reliable agents · nearform.com · Mar 2026

Claude improves the agent_

Claude Code changes the agent, runs the evals, and keeps what helps.

The improve loop: Auto Agent on one agent, pass rate on its evals



Past about 90-95%

you start tuning to the test set. Grow the golden dataset with harder cases, and check on cases the loop never saw.

Form a hypothesis
one git branch each

→

Change the agent
prompt, tool descriptions, tool logic

→

Run the evals
the same cases every time

→

Keep or roll back
a fixed number of iterations

Review the production traces_

Store every session once, and let an agent review them periodically.



Tracebound, our open source tool, does the failure half

It groups negative-feedback traces into failure modes, drafts a fix spec for a person to approve, and replays the inputs that went wrong. For example, we found ten such modes within about a hundred sessions in a client engagement.

Tracebound · github.com/nearform/tracebound

Let the second brain dream_

A shared memory, kept as a wiki, that a dream consolidates.

Learning loop 2: dream



The pattern: Andrej Karpathy's LLM Wiki

the model writes interlinked pages, keeps them current and checks them for health

The risk

one wrong fact, saved once, can come back in later sessions

Karpathy, "LLM Wiki", gist, 4 Apr 2026

Agentic Mapper: decide where agents belong_

Map the real business process first, using only what is in the interviews.

Why
sponsor
outcome and metrics

→

Scope
process owner
hypothesis map

→

As-is
people doing the work
as-is map, every value quoted

→

Analysis
consultant and sponsor
a decision for each step

→

To-be
team, then client
a human role for each agent step

Behind every value
the quote, word for word
who said it, in which interview
a link to the exact turn

Ranked by hours freed and expertise at risk

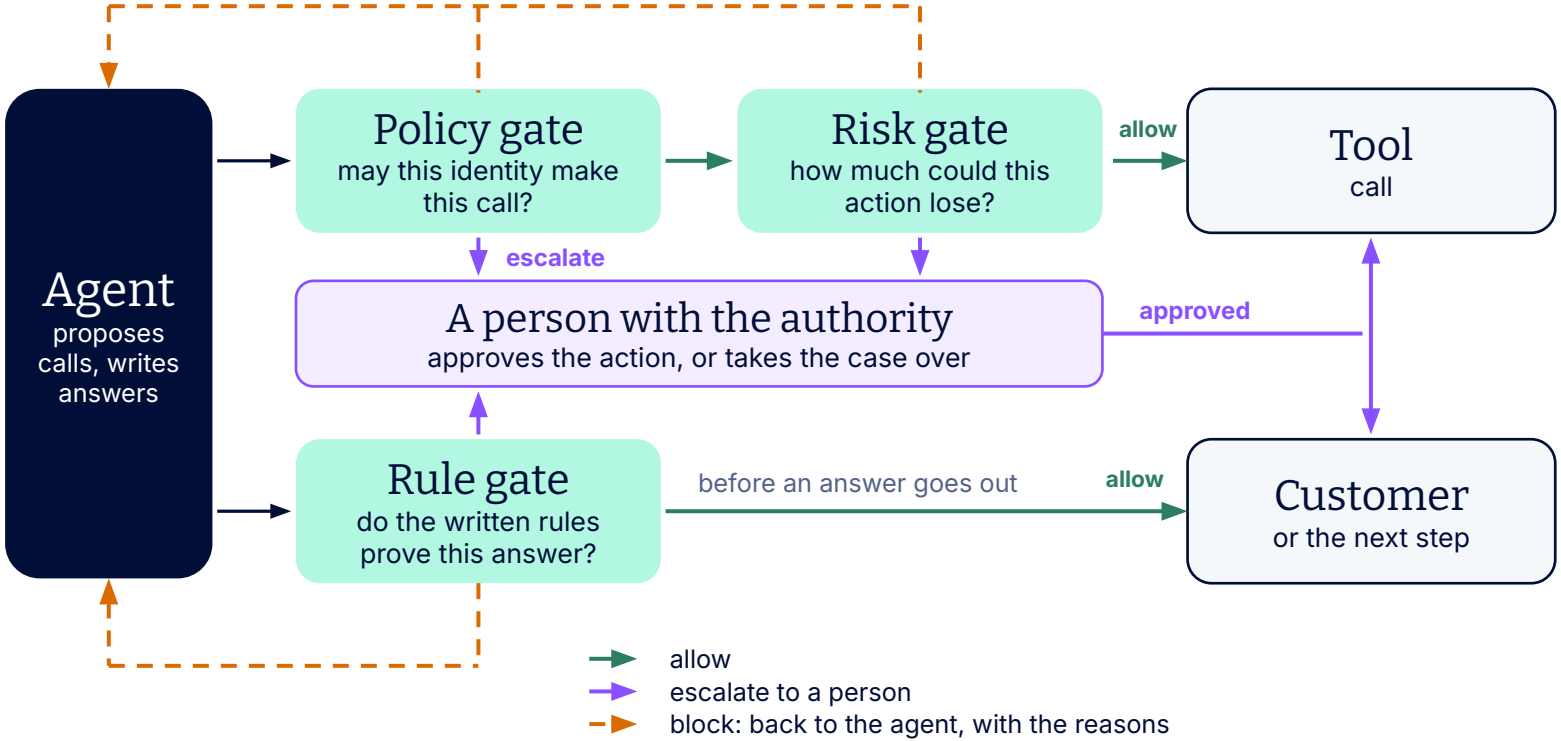
hours	volume × duration × people
risk	1 ÷ people who can do the step
score	hours × pain × (1 + risk) × feasibility

Never reaches the map
a claim whose quote isn't in the transcript
the interviewer's own words
an instruction hidden in an answer

Interviews by transcript, AI chat, live notes or screen recording. A person accepts every claim.

Who approves the action?_

Three gates outside the model: each one allows, escalates or blocks.



Rule gate: prove the answer_

The written rules, turned into logic, check each answer before it goes out.

Our example: a new customer wants £40 back, paid to a different card, 12 days after delivery

1 · Rule gate | 2 · Policy gate | 3 · Risk gate

The agent says
“You’re eligible for a refund”

→

The refund rule
within 30 days of delivery

→

The session’s facts
delivered 12 days ago, from the order lookup

Allow: the solver proves the answer, and it goes out

If the delivery date were missing: block, “a fact is missing”. The agent looks it up and answers again.

A solver (Z3) checks the rules and the session’s facts. The 30-day rule is invented for this example.

Policy gate: who may do what_

The policy reads who is calling, what the call says, and what it acts on.

Our example: a new customer wants £40 back, paid to a different card, 12 days after delivery

1 · Rule gate | **2 · Policy gate** | 3 · Risk gate

Who — principal
role: support, set by the deployment

What — action and context
issue_refund: £40, to a different card

On what — resource
the order: £40 left to refund

P1 support may refund up to £100

P2 a supervisor up to £500

P3 never more than is left of the order

P4 only a supervisor refunds to a different card

Block

P4 fails. The reason goes back to the agent.

A supervisor is a role a person holds. The agent offers the original card instead, and P1 allows it.

Cedar evaluates principal, action, resource and context against the policies, which are code a person can read and test.

Risk gate: how much can go wrong_

Put a money value on the action's risk, and compare it with the thresholds.

Our example: a new customer wants £40 back, paid to a different card, 12 days after delivery

1 · Rule gate | 2 · Policy gate | **3 · Risk gate**

FAIR
a standard for risk in money

Risk = Loss Event Frequency × Loss Magnitude

1%	× 3	× 8	× 3
base chance of fraud	new customer	different card	address changed



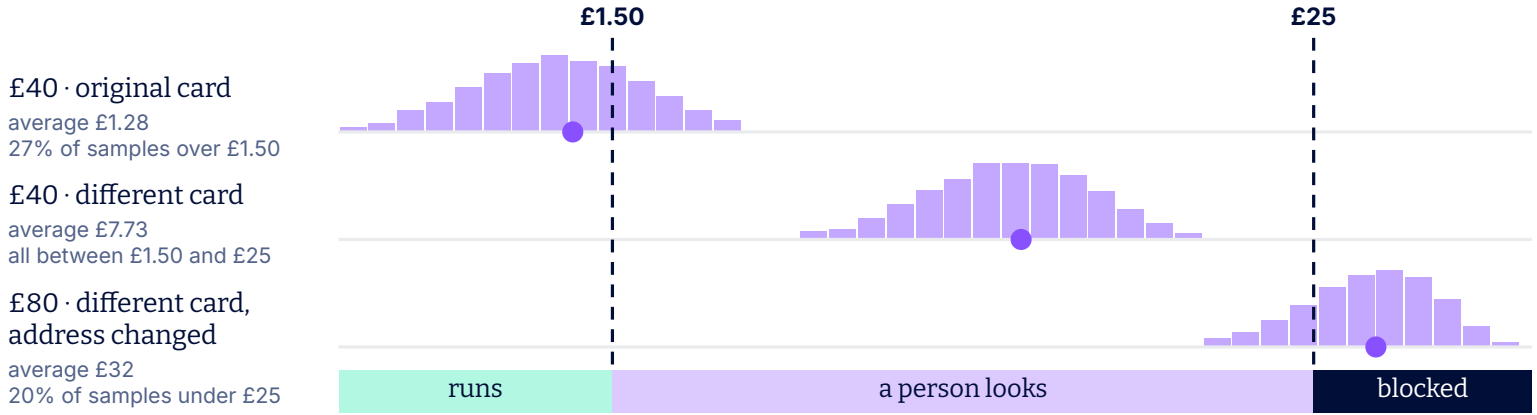
£1.20	£9.60	£28.80
3% × £40	24% × £40	72% × £40
original card	different card	different card, address changed

Monte Carlo simulation: from ranges to a distribution_

Ranges are easier to estimate. A Monte Carlo simulation combines them.

1%	× 3	× 8	× 3
base chance of fraud	new customer	different card	address changed
range 0.5% to 2%	range × 2 to × 5	range × 4 to × 12	range × 2 to × 4

Each sample: one value from every range, then chance × amount. 10,000 samples per refund, about 8 ms.



● the average, which the gate compares with the thresholds · bars: samples at each amount at risk, on a log scale

Markov chains: risk levels that move_

One chain follows a run, another a gate, each moving a risk level.

Run level

our customer, step by step



up: an estimate crosses into a higher band

down: a new fact lowers the estimate

High: a person looks before it runs

Our customer, if the address had changed last week: 8% after the customer lookup, caution. The different card makes it 42%, still caution, but the risk gate sends the £17 at risk to a person.

Gate level

the mix of all cases



up: with a set chance, on each risky action

down: with a set chance, on each action under 2%

More checks: every estimate doubled. People check: all but the lowest-risk actions

Simulated: the refund gate spends about 8% of days at more checks. In a fraud campaign, 24%.

One tip for each stage_

Make every change to your agent verifiable.

01 Start	02 Build	03 Guard	04 Measure	05 Improve	06 Keep improving
Pick the model per job	Pin skills, read updates	Run the cheapest check first	Check the eval before the agent	Let Claude tune the agent	Learn from every session

At every stage: gate the risky actions, and decide who approves each kind of change.

The tools behind this talk

Last Light
open source software factory
GitHub

Tracebound
from failing traces to a fix
GitHub

Unwind
rebuild an old system, proven against it

Agentic Mapper
where agents belong in a process

Community: nearform.com/digital-community · Reference Architecture: reference-architecture.labs.nearform.com



Engineered differently

Danilo Poccia
AI Solutions Director
danilo.poccia@nearform.com
x.com/danilop
linkedin.com/in/dpoccia

Nearform_