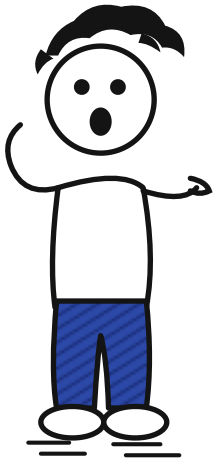
I get what `/loop` does now.

But Boris uses it to have Claude maintain his **whole** app - how does that work?



?

Auto-pacing

Crash fuzzer

Verification

Worktrees

Tuning

Dup unifier

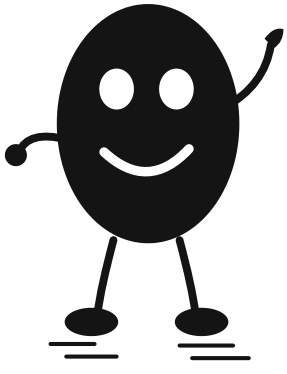
Routine workshop

Ralph loop

Squad

Harness

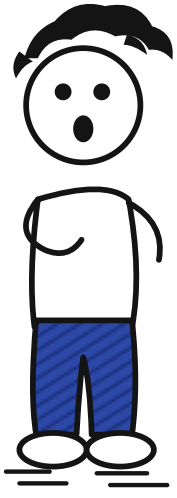
That's called **Loop Engineering**. Come on, I'll show you.



Hold on – I already know `/loop`. So what is **Loop Engineering** then?

`/loop` is just the building block: a loop that repeats a task.

Loop Engineering is the craft around it – making sure the loop can run **unattended for days** without causing harm.



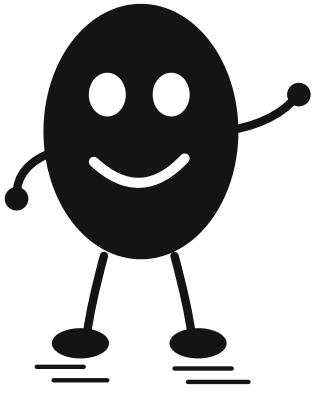
/loop

the building block

↓

Auto-pacing + verification + tuning + isolation

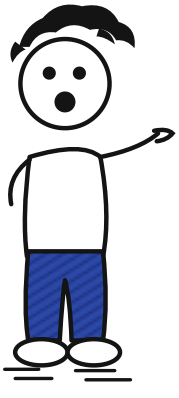
the craft around it

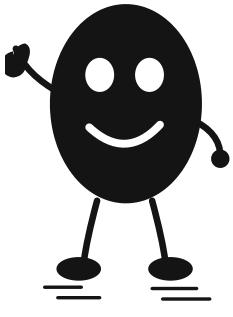


Give me an example of what that looks like in practice.

Boris Cherny runs several routines daily over his own apps, in a Slack channel.

A crash fuzzer taps around the app and fixes crashes, a dup unifier finds duplicate code, a dead-code remover cleans up - each routine with its own job.





Crash fuzzer

Dup unifier

Dead-code remover

Abstraction police

↓

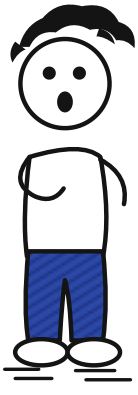
388 pull requests in a few weeks

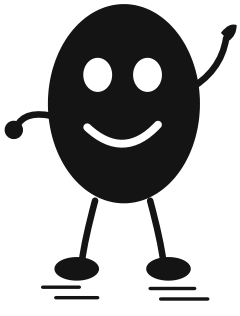
180 of them merged after code review + human review

Do I have to tell the loop every time how often it should run?

Not necessarily. Without a time given, Claude picks the interval itself.

Short while things are actually happening – longer during quiet stretches. That's exactly what makes running for days practical: nobody has to keep adjusting an interval.





busy
short interval

quiet stretch
long interval

blocked
fallback wake-up

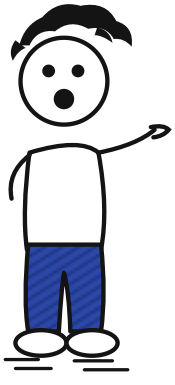
↻ ↻ ↻

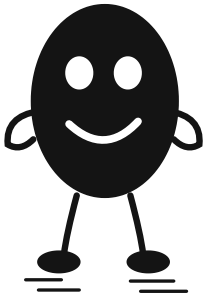
Claude decides for itself when the next run is worth it

This loop that just keeps planning, doing, checking – does that have a name too?

Yes – that's called a **Ralph-loop**: plan, execute, check, from the top.

Robust, but crude – without a brake it keeps burning tokens even after it's long since done. That's exactly what auto-pacing is for, and the verification that comes next.





Plan → Execute → Check

↻ from the top, until done

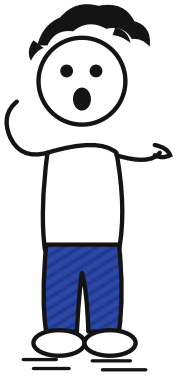
Ralph loop

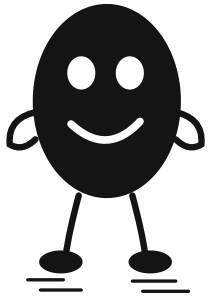
robust, but expensive in tokens without a brake

But if nobody's watching – how am I supposed to trust the loop?

That's exactly where Loop Engineering stands or falls: the loop has to be able to verify its own work **itself**, end-to-end.

Run the tests, have a second model review it – automated code review and security review – and only then offer the change as a pull request.





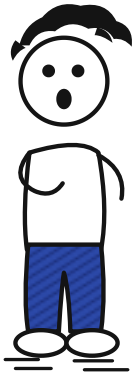
Change → Tests → Code review → Security review → Pull request

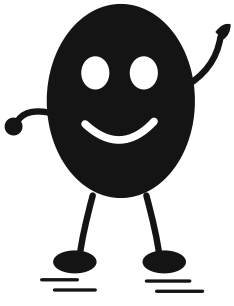
if a check fails, there is no proposal – no human has to look at it beforehand

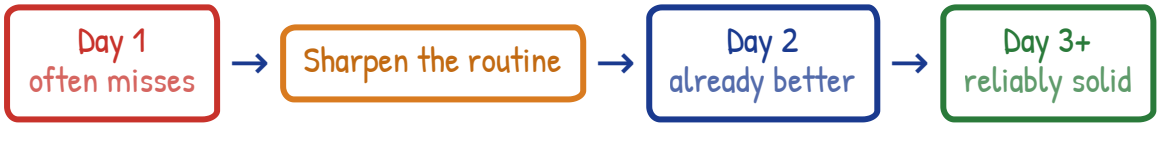
And what if the loop makes a mess of it at first?

Then you adjust the routine – or just ask Claude to improve it itself.

Sometimes one day is enough, sometimes it takes several attempts before the routine sits reliably. That's expected, not a failure.





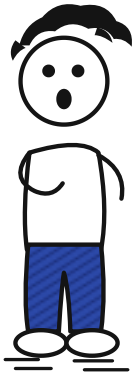


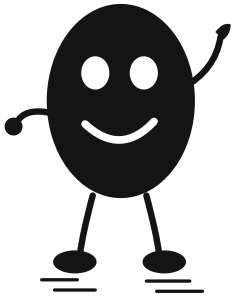
Claude gets feedback and tunes its own routine

Is that tuning from before systematic over time, or is it just guesswork?

The systematic version is called **hill climbing**: you measure quality with a fixed checklist – an eval – and adjust the routine on purpose.

No guesswork: every change gets tested against the eval, only what actually scores better stays in.





Define eval → Adjust routine → Test against eval

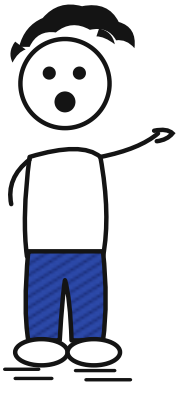
↗ step by step, uphill

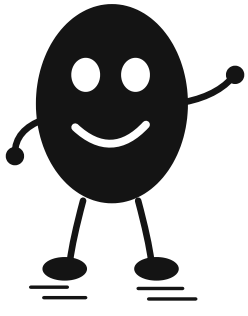
keep only what the eval confirms – that's hill climbing

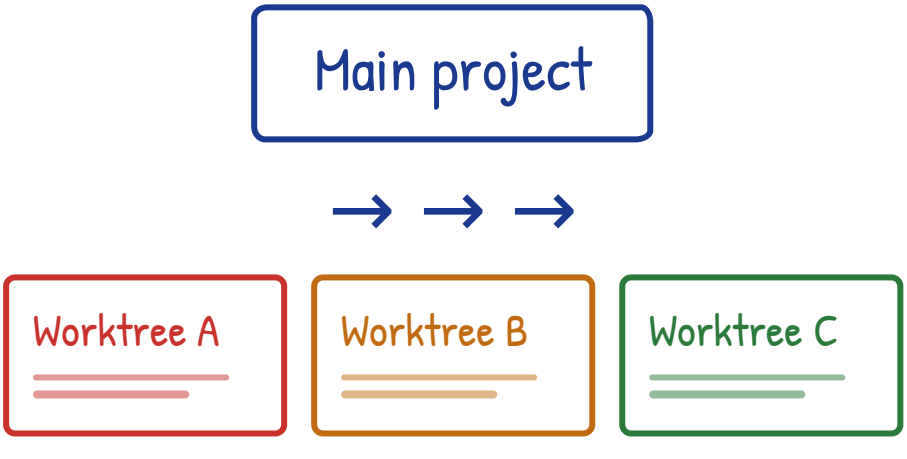
Don't several loops in the same folder get in each other's way?

No – each routine works in its own copy of the project, a **worktree**.

That way parallel runs don't interfere with each other, and a failed attempt doesn't drag the main branch down with it.





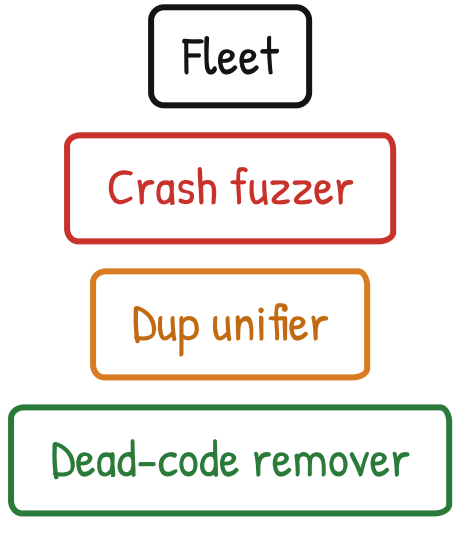


each loop on its own, nothing overwrites anything else

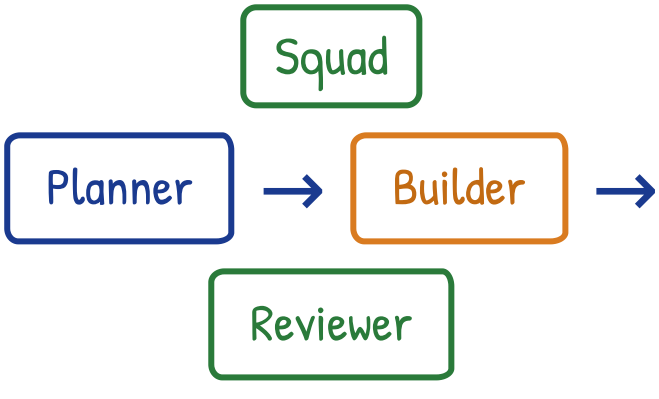
The worktrees from before – do they each run as their own, separate Claude?

Mostly yes: each routine its own Claude – many of them at once is called a **fleet**.

But sometimes several Claudes work **together** on a single task, with roles split up – that's a **squad**: one plans, one builds, one reviews.



many, independent, parallel

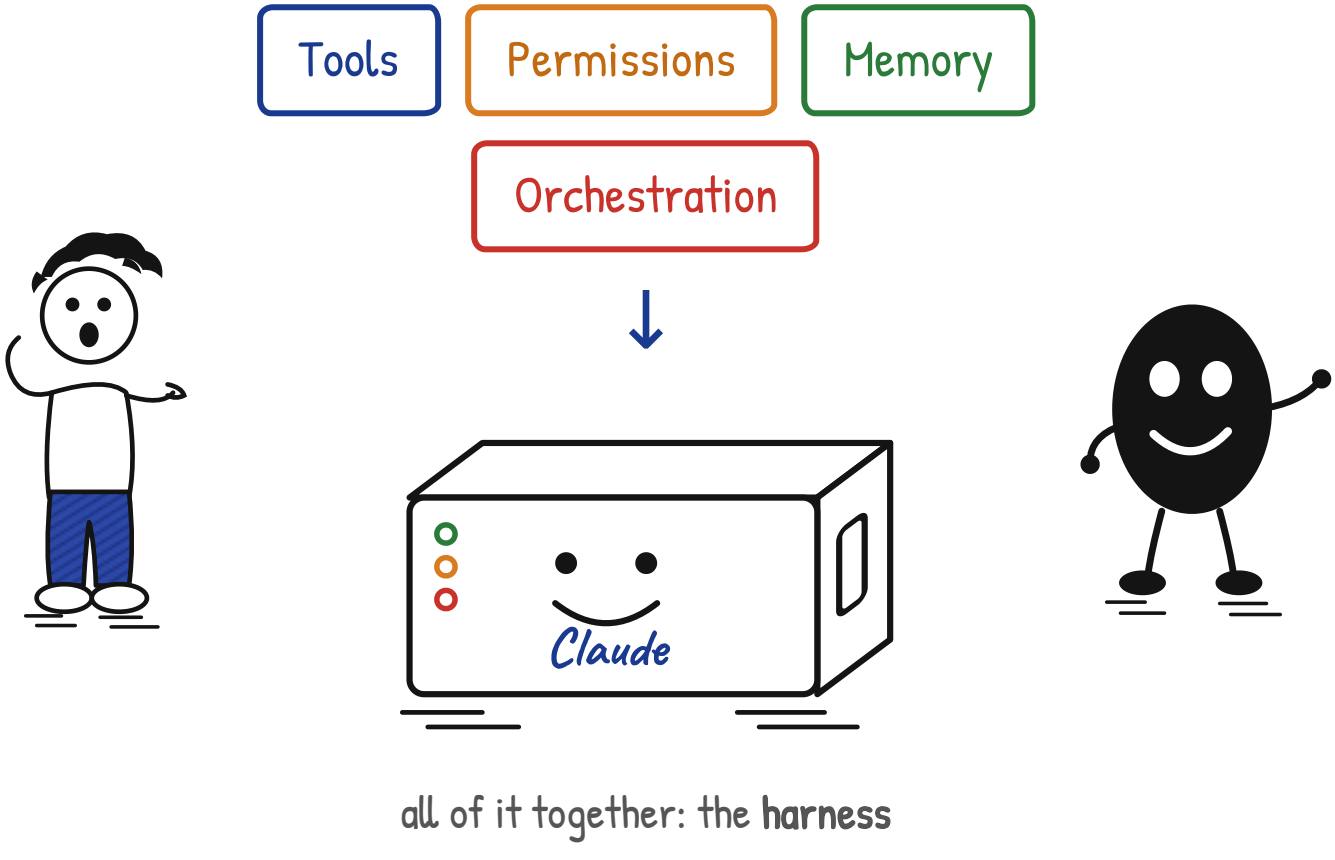


a few, with split roles, on one task

Auto-pacing, verification, tuning, isolation – does this whole scaffolding have a name too?

Yes – the sum of tools, permissions, memory, and orchestration around the model is called a **harness**.

Loop Engineering is exactly that: building a harness Claude can work in, unattended, for days.

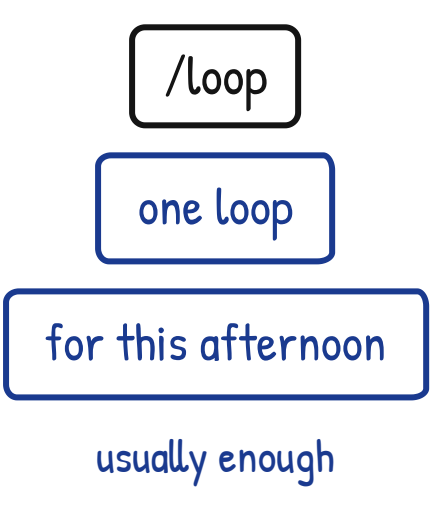


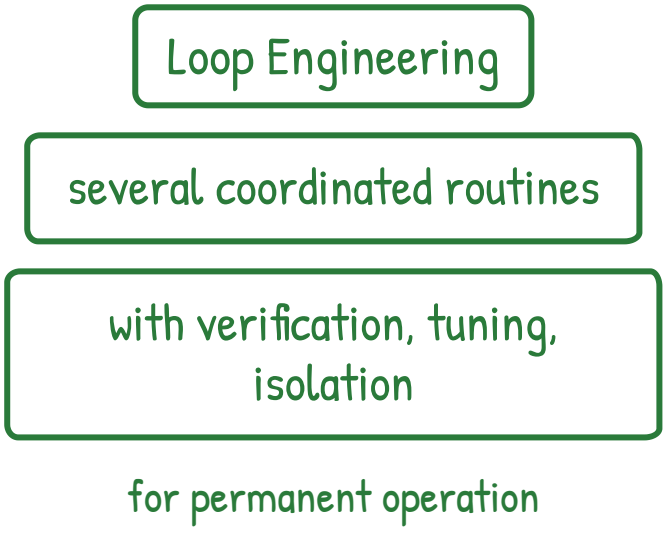
all of it together: the **harness**

So when is all this craft actually worth it – isn't `/loop` enough most of the time?

For a single afternoon, `/loop` really is completely enough.

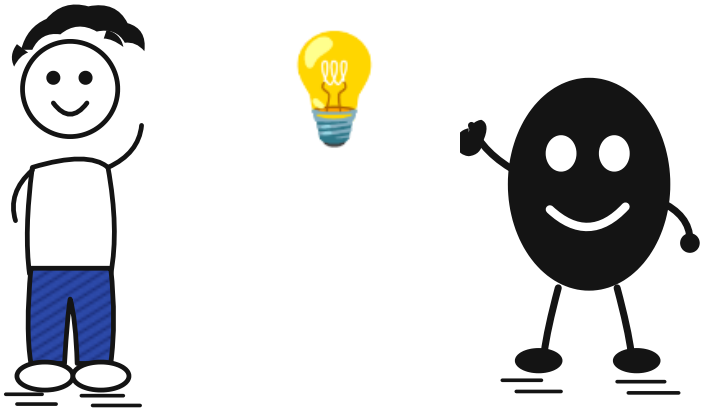
Loop Engineering pays off once several such routines are meant to run alongside each other permanently – then individual loops turn into a small workshop that maintains itself.





Now Boris' Slack channel finally makes sense.

That's exactly the point.



/loop – the building block, repeats a task

Ralph loop – plan, execute, check, from the top

Auto-pacing – Claude picks the interval itself

Verification – tests, review, only then a proposal

Hill climbing – tuning with an eval instead of guesswork

Worktree isolation – each loop on its own

Squad / fleet – split roles, or many in parallel

Harness – tools, permissions, memory, orchestration

Loop Engineering – all of it as a workshop in permanent operation