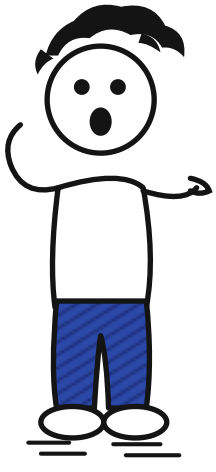
Ich hab jetzt kapiert, was `/loop` macht.

Aber Boris lässt Claude damit ja seine **ganze** App warten – wie geht das?



?

Auto-Pacing

Crash-Fuzzer

Verifikation

Worktrees

Tuning

Dup-Unifier

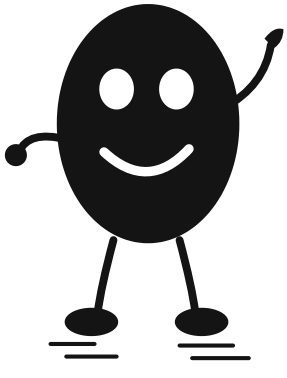
Routinen-Werkstatt

Ralph-Loop

Squad

Harness

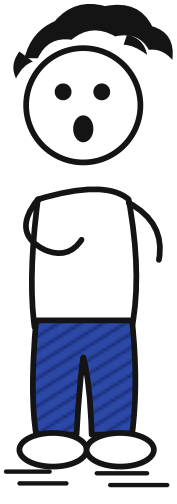
Das nennt man **Loop Engineering**. Komm', ich zeig's dir.



Moment – `/loop` kenn ich doch schon. Was soll **Loop Engineering** dann sein?

`/loop` ist nur der Baustein: eine Schleife, die einen Auftrag wiederholt.

Loop Engineering ist das Handwerk drumherum – dafür sorgen, dass die Schleife **tagelang unbeaufsichtigt** laufen darf, ohne Schaden anzurichten.



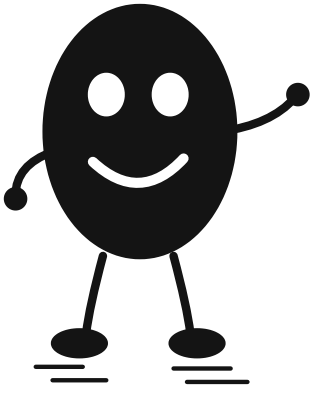
/loop

der Baustein

↓

Auto-Pacing + Verifikation + Tuning + Isolation

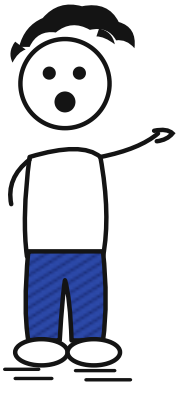
das Handwerk drumherum

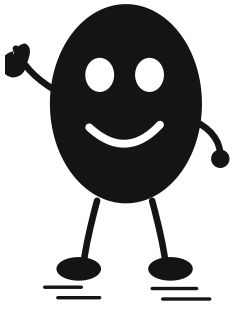


Gib mir mal ein Beispiel, wie das konkret aussieht.

Boris Cherny lässt in einem Slack-Kanal mehrere Routinen täglich über die eigenen Apps laufen.

Ein Crash-Fuzzer tippt in der App herum und behebt Abstürze, ein Dup-Unifier findet doppelten Code, ein Dead-Code-Entferner räumt auf – jede Routine mit ihrem eigenen Auftrag.





Crash-Fuzzer

Dup-Unifier

Dead-Code-Entferner

Abstraktions-Polizei

↓

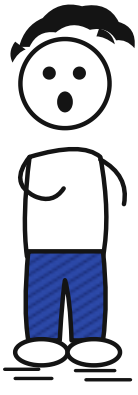
388 Pull Requests in wenigen Wochen

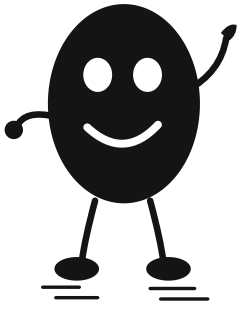
180 davon nach Code-Review + Mensch gemerged

Muss ich der Schleife jedes Mal sagen, wie oft sie laufen soll?

Nicht zwingend. Ohne Zeitangabe wählt Claude den Abstand selbst.

Kurz, solange sich gerade etwas tut – länger, wenn Ruhe ist. Genau das macht tagelanges Weiterlaufen erst praktikabel: niemand muss ständig ein Intervall nachjustieren.





viel los
kurzer Abstand

Ruhephase
langer Abstand

blockiert
Fallback-Weckruf

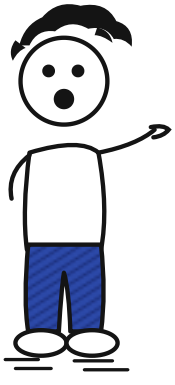
↻ ↻ ↻

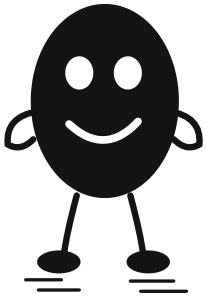
Claude wählt selbst, wann der nächste Durchlauf sich lohnt

Diese Schleife, die einfach immer wieder plant, macht, prüft – hat das auch einen Namen?

Ja – das nennt man einen **Ralph-Loop**: planen, ausführen, prüfen, von vorn.

Robust, aber roh – ohne Bremse frisst er Token weiter, auch wenn er längst am Ziel ist. Genau dafür gibt's Auto-Pacing, und die Verifikation, die als Nächstes kommt.





Planen → Ausführen → Prüfen

↻ von vorn, bis fertig

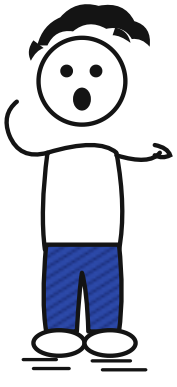
Ralph-Loop

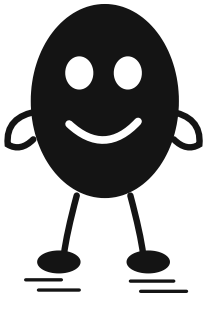
robust, aber ohne Bremse teuer im Token-Verbrauch

Und wenn dabei niemand zuschaut – wie soll ich der Schleife dann trauen?

Genau da steht oder fällt Loop Engineering: die Schleife muss ihre eigene Arbeit **selbst** prüfen können, end-to-end.

Tests laufen lassen, ein zweites Modell gegenlesen – automatisches Code-Review und Sicherheits-Review – und erst danach den Vorschlag als Pull Request anbieten.





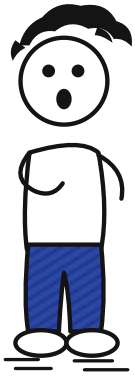
Änderung → Tests → Code-Review → Sicherheits-Review → Pull Request

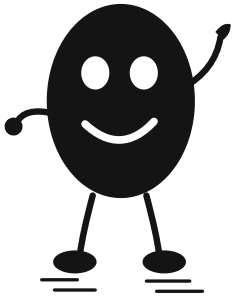
fällt eine Prüfung durch, gibt es keinen Vorschlag – kein Mensch muss vorher draufschauen

Und wenn die Schleife am Anfang noch Mist baut?

Dann justierst du die Routine nach – oder bittest Claude direkt darum, sie selbst zu verbessern.

Manchmal reicht ein Tag, manchmal braucht es mehrere Anläufe, bis der Ablauf zuverlässig sitzt. Das ist eingeplant, kein Fehlschlag.





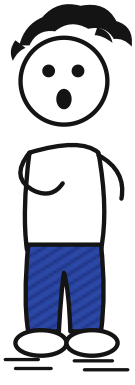
Tag 1 trifft oft daneben → Routine schärfen → Tag 2 schon besser → Tag 3+ sitzt zuverlässig

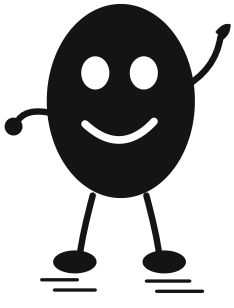
Claude bekommt Rückmeldung und tunt den eigenen Ablauf nach

Ist das Nachjustieren von eben auch mit der Zeit systematisch, oder rät man da nur rum?

Systematisch heißt **Hill Climbing**: man misst die Qualität mit einer festen Prüfliste – einem Eval – und schraubt gezielt an der Routine.

Kein Rätselraten: jede Änderung wird gegen den Eval getestet, nur was wirklich besser abschneidet bleibt drin.





Eval definieren → Routine anpassen → gegen Eval testen

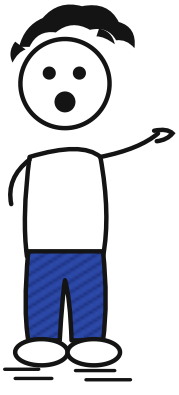
↗ Schritt für Schritt bergauf

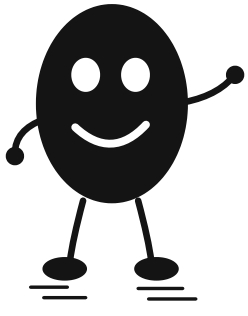
nur behalten, was der Eval bestätigt – das ist Hill Climbing

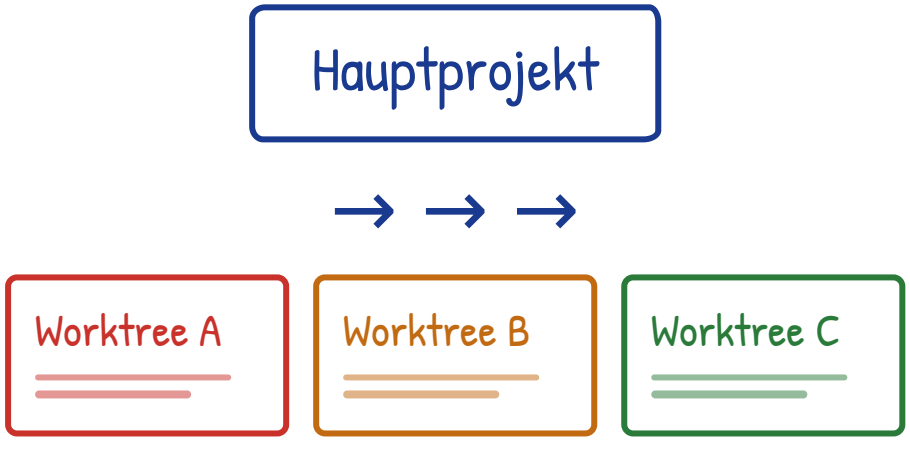
Laufen da nicht mehrere Schleifen im selben Ordner durcheinander?

Nein – jede Routine arbeitet in ihrer eigenen Arbeitskopie, einem **Worktree**.

So stören sich parallele Läufe nicht gegenseitig, und ein missratener Versuch reißt nicht den Hauptstand mit runter.





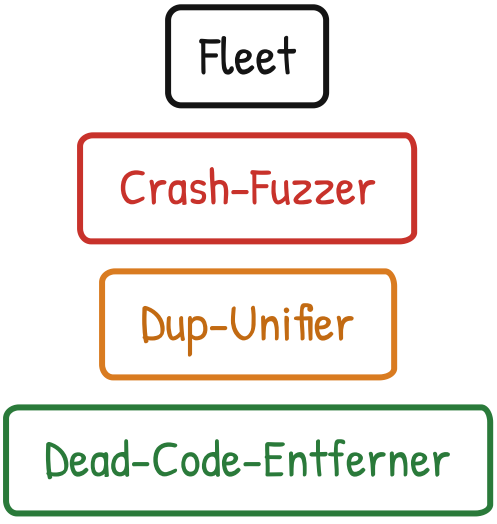


jede Schleife für sich, nichts überschreibt sich gegenseitig

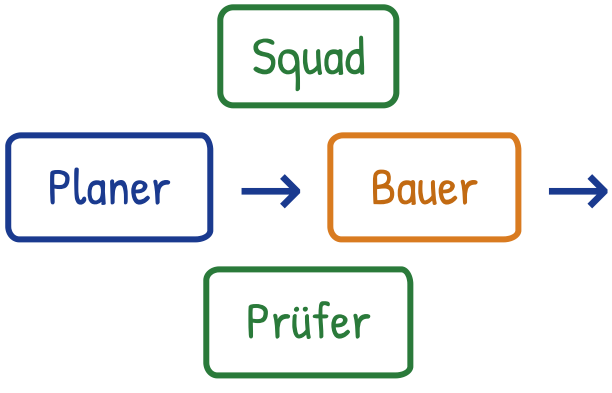
Die Worktrees von eben – laufen die dann alle als eigene, einzelne Claudes?

Meistens ja: jede Routine ihr eigener Claude – viele davon gleichzeitig nennt man eine **Fleet**.

Manchmal arbeiten mehrere Claudes aber auch **zusammen** an einer einzigen Aufgabe, mit verteilten Rollen – das ist ein **Squad**: einer plant, einer baut, einer prüft.



viele, unabhängig, parallel

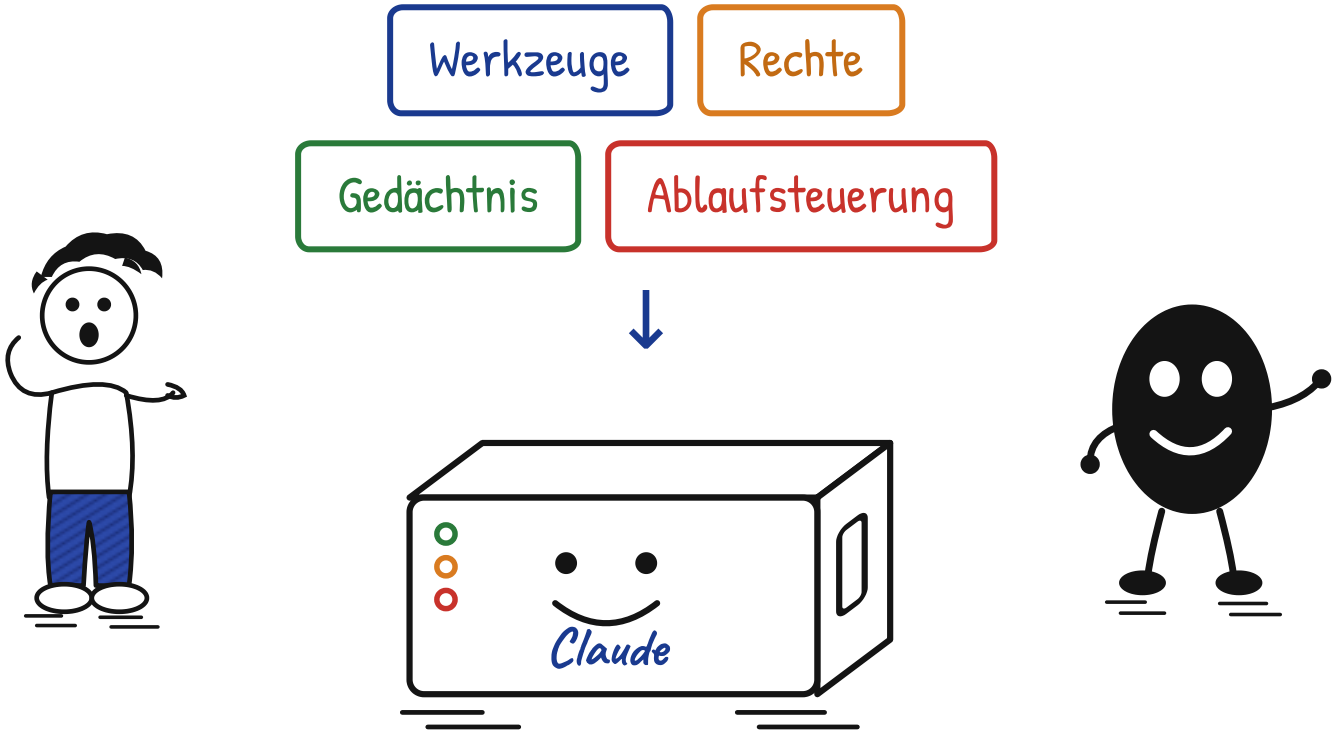


wenige, mit verteilten Rollen, an einer Aufgabe

Auto-Pacing, Verifikation, Tuning, Isolation – hat dieses ganze Gerüst auch einen Namen?

Ja – die Summe aus Werkzeugen, Rechten, Gedächtnis und Ablaufsteuerung um das Modell herum heißt **Harness**.

Loop Engineering ist genau das: einen Harness bauen, in dem Claude tagelang unbeaufsichtigt arbeiten darf.

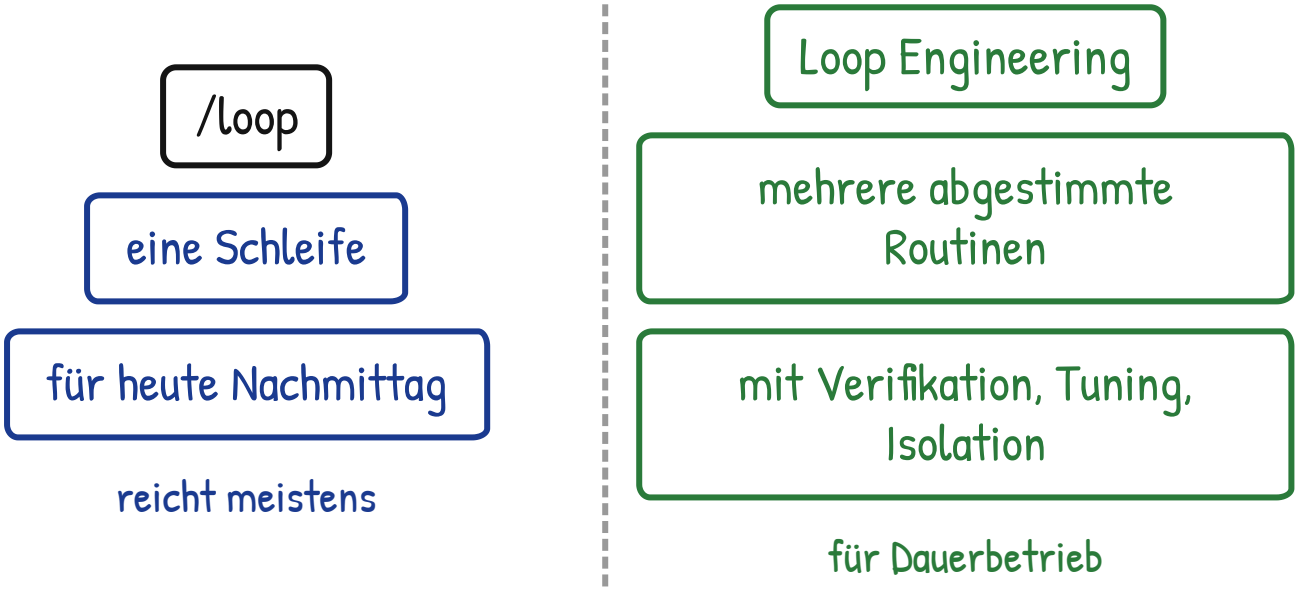


alles zusammen: der **Harness**

Wann lohnt sich das ganze Handwerk dann – reicht `/loop` nicht meistens?

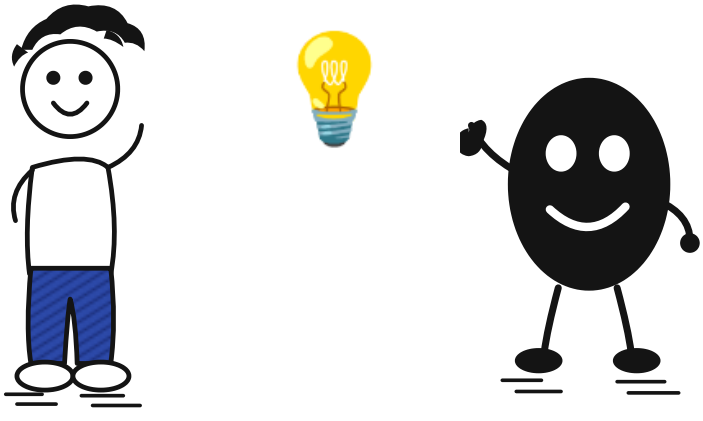
Für einen einzelnen Nachmittag reicht `/loop` tatsächlich völlig.

Loop Engineering lohnt sich, sobald mehrere solcher Routinen dauerhaft nebeneinander laufen sollen – dann wird aus einzelnen Schleifen eine kleine Werkstatt, die sich selbst instand hält.



Jetzt macht der Slack-Kanal von Boris endlich Sinn.

Genau darum geht's.



/loop – der Baustein, wiederholt einen Auftrag
Ralph-Loop – planen, ausführen, prüfen, von vorn
Auto-Pacing – Claude wählt den Abstand selbst
Verifikation – Tests, Review, erst dann ein Vorschlag
Hill Climbing – Tuning mit Eval statt Rätselraten
Worktree-Isolation – jede Schleife für sich
Squad / Fleet – verteilte Rollen oder viele parallel
Harness – Werkzeuge, Rechte, Gedächtnis, Ablaufsteuerung
Loop Engineering – all das als Werkstatt im Dauerbetrieb