The end goal, in the project's own words: "a software factory that assembles itself from its own failures."

Intro - what has been built, in plain words

DREAM FLOW

23 Nov 2026

DreamFlow helps one developer turn a rough request into a small, tested change that can be merged into the main project. This deck covers what is built, what is in progress, and where it is headed.

The goal: let one developer cover many kinds of work. The problem: assistants guess, and one person pays for every bad guess in review time.

Today's coding assistants write plenty of code, but they guess, drift and stall.

Part 1 · Progress - how it began

One person, **every discipline**

Drift

Instructions get lost: when work is handed on as a summary, the side rules often do not survive.

Bloat

Rules swell: the rules file every assistant had to read grew large and went out of date.

Stall

Work runs on: an overnight job kept merging changes and left the tests failing by morning.

Those lessons were learned the hard way, in three beats: a first harness, a stall, then a fresh start called DreamFlow.

Part 1 · Progress - the first attempt

A false start, **then DreamFlow**

First try created 4 Nov; DreamFlow's first change merged 15 Nov

First try

4 Nov: a harness, a helper system that keeps coding assistants working through a task list

Stalled

It grew large and complicated, then stopped making progress



15 Nov

A fresh start. Its first proposed change, checked and then merged, lands.

The first harness broke in two big ways: a timer made work wait, and assistants were handed far too much to read at once.

Part 1 · Progress - what was learned, 1

Gates and **context** build-up

Waiting

Work sat idle until a timer fired, instead of moving when it was ready.

Packed

in one session

Problem: a timer made requests wait.

A timer let work move only when the clock fired, not when the work was ready. Fix: drop the timer, so work moves as soon as it is ready.

Problem: assistants were handed too much to read.

Rules and logs were copied into every session. Fix: point to documents instead of copying them, and start each session from one entry file.

It also broke in three smaller ways. The standing rule since: every failure should become a safety check.

Part 1 · Progress - what was learned, 2

Stall, bloat, **pop-ups**

Stuck recovery

A stuck recovery held everything else still, because two timers fought each other.

Spec bloat

Specs, the short written plans for a change, grew too big because nothing capped their size.

Console pop-ups

Console windows popped up on screen, because launches ran visibly.

Takeaway

Every failure becomes a safety check, not only a habit.

≤5

files at most in one change

Short

a size cap on each written plan

Out of those lessons came one method. Every change, including changes to DreamFlow itself, follows these ten steps in order.

Part 1 · Progress - the method

Define. **Bound.** Evolve.

The ten-step method, in order

1. Define: write the acceptance test first
2. Question: challenge every requirement
3. Delete: remove what isn't needed
4. Simplify: make what's left smaller
5. Bound: 5 files, one piece of work at a time
6. Measure: judge nothing without a number
7. Accelerate: only what the numbers show is slow
8. Wire: some running code uses every part
9. Automate: only once it matches the manual way
10. Evolve: each failure becomes a gate or test

5 files
the most any one change may touch

10 steps
already in use on every change

Who decides what

The language model, the general-purpose program that proposes changes, is treated as a guess, never as proof. Taste, look, new ideas and the overall design stay with us.

Every change takes one path: a work item, a short plan and a test that must fail first, drafts, a build, the checks, then a merge.

Part 1 · Progress - one change's path

Six steps, **one path**

"Every failure becomes a gate, a test or a redesign."

~ Evolve, step 10 of the method ~

1 Issue

A tracked work item. The board is the one list of what is next.

2 Plan + first test

A short plan (five files at most) plus a test that must fail first.

3 Parallel drafts

Several drafts of the change are written side by side.

4 Build

The code is written in a separate copy; the locked tests cannot be changed.

5 CI + merge gate

The automatic checks must pass, then safety checks, before a merge.

6 Merge

If the proof holds, it merges itself; if not, limited retries, never a way around.

Because AI writes both the code and the tests, the red tests (written first, and made to fail before any code exists) are locked, so it can't grade its own homework.

Part 1 · Progress - safety checks

The agent can't grade **its own homework**

The automatic checks must pass

- CI is green: tests, style checks and type checks
- The lock on the first tests is unchanged
- No one edited the tests after they first failed
- The change touches 5 files or fewer
- No edits to standing rule files





This is the only set of parts treated as built. Everything else is in progress or planned.

Already running

A short written plan, tests written first and locked, early checks, and a design stage for bigger asks. The merge gate, and a rule that secrets, untrusted input, and outbound access are not held together. Automatic repair when a check fails, a pause when usage limits are hit, promotion and rollback, and a record of time and cost.

Running in part

Turning a kind of failure into a new piece of work has begun, but it is not finished yet. After a restart, work on the studio's build machine is taken over instead of being stopped.

Part 1 · Progress - what is built today

Built today, **some in part**

1. **Already in the main project**
2. **On the studio machine only**

This work is under way. It is not treated as built yet.

Part 2 · In progress - landed recently

Landed recently, **still in progress**

Now

work that has landed, still being finished

Next

not treated as built until the next milestone

Smoother drafting

Tighter limits on what a drafting pass can do, so each plan stays small and uses only the allowed tools.

Cleanup

A review of the structure, and unused parts being cleaned up.

Self-repair

A health watcher, a usage budget, restarts when a worker stops, and automatic re-checks.

Board reconciled

Checked against the code, so the list of work matches what exists.

Still being built: more self-repair, new quality checks, and a smoother path from a plan to a finished change.

Part 2 · In progress - still being built

Still being built

"All of this serves bar 1: eight hours stable, no hand rescues."

~ the next milestone ~

1 Failed-check router

Sends each failed check to the right fix, instead of leaving it stuck.

2 Stuck-work checks

Releases stale work, raises stuck items, and reloads code when needed.

3 Check before a push

A check on the developer's machine before code is sent: size, wiring, and timeouts.

4 Unused-code check

Catches orphan code, meaning code that nothing actually calls.

5 A shorter first pass

A shorter first look at the work, so the written plan can stay focused.

6 Focused test runs

Run only the tests a change can affect, check style first, and re-check the machine after each repair.

Priorities as of 23 Nov: paid work first. The near goal is bar 1, an eight-hour stable run with no hand rescues, targeted this week.

Paid work comes first. All of this serves bar 1, the next milestone: an eight-hour stable run with no hand rescues, meaning no one steps in by hand to unstick the system.

First
paid work first

Part 2 · In progress - priorities

Priorities **and bar 1**

Bar 1 is targeted this week. No calendar day is promised until that date is firmed up.

1. **Paid job delivery**
   Comes first; paid contract work is done by hand
2. **A later product offer**
   The second priority. This offer is not open yet.
3. **Everything else**
   The rest of the work, after the first two tiers

Where it is headed: reach bar 1, keep the studio services first, and build toward the end goal, a software factory that assembles itself from its own failures.

Part 3 · Vision - the road ahead

The road ahead, **step by step**

After bar 1: a split between a development line and a stable line, then a third work lane (planned)

Bar 1

Targeted this week: eight hours stable, no hand rescues

90 days

Services first: paid contract work, done by hand



Next

Planned: a later product offer. DreamFlow is not for sale today.

The aim is that a kind of decision could become the default once a clear pattern of accepted choices has been seen. That is a stated aim, not a result.

Planned, not yet built.

Part 3 · Vision - the decision dial

A decision dial, **not yet built**

Count

Layer 1 counts how often we accept each kind of decision.

Learn

Layer 2 is a small local model trained on those past decisions.

Explain

Layer 3 uses a language model only to offer and explain options.

Status as of 23 Nov 2026. DreamFlow is not for sale and not ready for client work. The model is treated as a guess, never as proof.

Close - the end goal

A software factory that assembles itself from its own failures.

Each failure should become a lasting improvement: a tighter gate, a better test or a simpler design, so the next pass is stronger than the last.