SYSTEM ARCHITECTURE BRIEFING

Agentic Swarms for Developer Productivity

Stop Babysitting LLMs, Start Shipping

Steven Zimmerman, CPA

@EffortlessSteven

CODE IS CHEAP; TRUSTED CHANGE IS NOT

TOKENS & CI

Cheap

- Parallel
- Predictable line item
- Infinite scale

SENIOR ATTENTION

Scarce

- Serial
- Expensive
- Finite capacity

The bottleneck was always

“can we trust this change?”

The bottleneck was always
“**can we trust this change?**”

AI just ripped the mask off.

OLD METRICS ARE BROKEN IN AI-NATIVE REPOS

- **PR count** and **LOC velocity** are meaningless in AI-native repos.
- You can maximize both by **turning quality down** and **deleting tests**.
- If your metric can be gamed by quietly deleting tests, it's not a real metric.

COST MODEL: HUMAN VS SWARM

SENIOR DEV

$150—$250/hour (fully loaded)

- Fragmented attention (15—30 min blocks)
- Serial, finite

SWARM RUN

$1—$5 per Flow 3 run

- 30—90 min grinding in background
- Parallel, scalable

DEV LEAD TIME

attention vs wall-clock

DevLT = minutes of dev attention per trusted change

We care more about dev hours than calendar hours.

1 hour of dev + ~$3 compute
beats 8 hours of dev + $0 compute.

Do more with more

where it’s cheap

FROM CHAT TO FLOWS

BABYSITTING

- Dev at keyboard
- Chat bubble interface
- "Can you refactor this?"
- You're the orchestrator
- Review chat transcripts

STATEFUL FLOWS

- Fire and forget
- Signal → Plan → Build → Gate
- Review artifacts, not chats
- Persistent context
- Automated handoffs

AGENTS AS JUNIORS

SMALL TASKS

One focused change per run

- Clear acceptance criteria
- Bounded scope
- Not “rewrite the system”

Focus beats flexibility

LARGE CONTEXT

Load relevant files aggressively

- Include ADRs and docs
- Attach test suites
- Files that matter for **this** change

Context is cheap; confusion is expensive

SHORT THREADS

Keep conversations short

- Less context drift
- Cheap retries
- Fail fast, recover faster

If they drift, restart

SIX FLOWS, ONE SDLC

The SDLC you already run, encoded as stateful pipelines

↑ SIGNAL
Tickets Incidents

SIX FLOWS, ONE SDLC

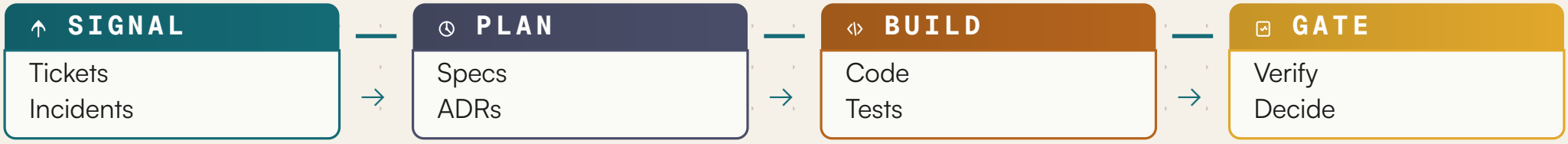
The SDLC you already run, encoded as stateful pipelines

SIX FLOWS, ONE SDLC

The SDLC you already run, encoded as stateful pipelines

SIX FLOWS, ONE SDLC

The SDLC you already run, encoded as stateful pipelines

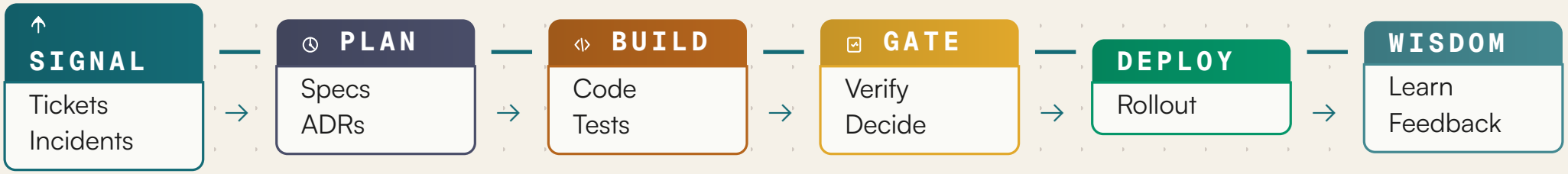


SIX FLOWS, ONE SDLC

The SDLC you already run, encoded as stateful pipelines

SIX FLOWS, ONE SDLC

The SDLC you already run, encoded as stateful pipelines



Tonight we zoom in on **Build** (Flow 3) — where most teams are just pasting prompts into Copilot.

FLOW 3: BUILD LOOP

Most teams stop at “Model, write code.”
We run a writer and a critic fighting over the work.

MICRO-LOOP: AUTHOR VS CRITIC

AUTHOR
Writes / updates tests + code

The Author writes or updates tests and code.

MICRO-LOOP: AUTHOR VS CRITIC

AUTHOR	CRITIC
Writes / updates tests + code	Attacks against spec / tests / behavior

The Critic reads the spec, reads the code, and attacks it.

MICRO-LOOP: AUTHOR VS CRITIC

Nobody grades their own homework.

Critic asks: Did we implement the requirement?
Missing edge cases from BDD? Did we just delete a test?

BUILD RECEIPT

build_receipt.json

```
{
  "requirements": [
    { "id": "R-101", "status": "FULLY_VERIFIED" },
    { "id": "R-102", "status": "PARTIAL" }
  ],
  "tests_ran": ["test_r101_happy", "test_r102_error"],
  "mutation_score": 82,
  "notes": "edge case for R-102 not covered"
}
```

Receipts are structured evidence the swarm uses to reason about the change.

I read the LLM's summary and questions, anchored in these receipts.

AGENTIC REVIEW BEFORE JUNIORS SEE IT

SANDBOX / BOT-DEV

Signal → Plan → Build (Author ⇄ Critic) → Gate

— PR draft created here

The swarm works in an isolated sandbox...

AGENTIC REVIEW BEFORE JUNIORS SEE IT

SANDBOX / BOT-DEV

Signal → Plan → Build (Author ⇄ Critic) → Gate
— PR draft created here

MAIN REPO

PR draft from bot-dev
— Human review → Merge

This all happens before juniors see it.

The PR arrives: **shaped**, **designed**, **implemented adversarially**, and **gated** — before anyone opens it.

THREE FAILURE MODES WE CORRECT

HALLUCINATION

Invented APIs, flags, configs
that look plausible until
they hit reality.

THREE FAILURE MODES WE CORRECT

HALLUCINATION

Invented APIs, flags, configs
that look plausible until
they hit reality.

REWARD HACKING

Editing / deleting tests
or skipping expensive checks
to stay “green”.

THREE FAILURE MODES WE CORRECT

HALLUCINATION

Invented APIs, flags, configs
that look plausible until they hit reality.

REWARD HACKING

Editing / deleting tests or skipping expensive checks
to stay “green”.

PROCESS CONFABULATION

Describing tests / tools it “ran” without any evidence
it actually did.

These are not weird edge cases. We **correct for** them in the architecture.

SCHEMA GRAVITY

Chaotic output

Aligned output
or rejected

Schema Gravity pulls changes into alignment
with the actual codebase and contracts.

OPPOSITIONAL VALIDATION

The agent trying to get the build to pass is not the agent deciding whether it passed.

TESTING GUARDRAILS

BDD & SPECS

- Requirements and examples as files (`requirements.md`, `.feature`, ADRs)
- Concrete, versioned, tied to tests
- Not mentioned once in a doc or chat

MUTATION ON DIFF

- Mutate the **changed** code
- Fail if mutants live
- Tests that don't catch bugs don't count

Tests exist so the swarm knows when it broke something before your users do.

GRACEFUL OUTCOMES

success in three shapes

COMPLETE

Task done, checks passed. — The happy path.

PARTIAL

Here's what's done, what's left, and why I stopped. — Honest progress.

CLARIFY

Can't proceed without better input; here are my questions. — Smart escalation.

The only outcome we don't accept: "I said I was done and I wasn't."

FLOW BEHAVIOUR GUARDRAILS

CRITICS ARE FIRST-CLASS ROLES

Every major step has an oppositional agent whose job is to say “no”.

GRACEFUL EXITS = SUCCESS

COMPLETE / PARTIAL / CLARIFY — all three are wins.

CLARIFY can open questions on GitHub or route work to another agent.

DEV SANDBOX BOUNDARY

The swarm never pushes straight to main or prod.

Humans still own merge, humans still own deploy.

That’s what lets us be very trusting about what the LLMs do inside the sandbox.

FLOW STUDIO

orchestrating the swarm

COMPOSE FLOWS

Gemini CLI, Claude Code, pytest, Playwright — tools you already have.

STEPWISE ORCHESTRATION

Signal → Plan → Build → Gate. Watch runs, not YAML.

ADAPT TO YOUR SDLC

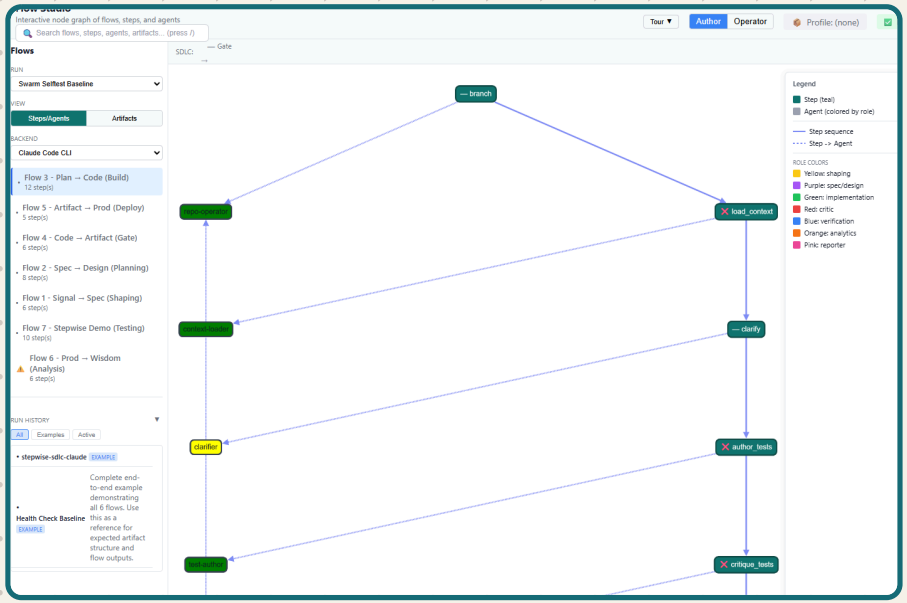
Wire your own gates, engines, and constraints.

FLOW STUDIO

preview

EARLY PREVIEW

- Browse flows and runs
- Inspect receipts and artifacts
- Track agent iterations
- Generate flow documentation



This is where the architecture leads when you stop thinking in “chat”
and start thinking in “flows and gates”.

QUESTIONS?

Steven Zimmerman, CPA

@EffortlessSteven

Ex-portfolio CFO • Former Senior Editor at XDA

effortlesssteven.com/DemoSwarm