

SYNERGY DEVPARTNER CONFERENCE 2026

# Doing More with More: AI-Directed Development at Synergex

Taras Yakovchuk, Associate Software Development Manager
Michael Oleshchuk, Jr. Software Engineer



**Peter Steinberger** @steipete · 4h

Here's your monthly reminder that you shouldn't be prompting coding agents anymore.

You should be designing loops that prompt your agents.



**Peter Steinberger**
@steipete

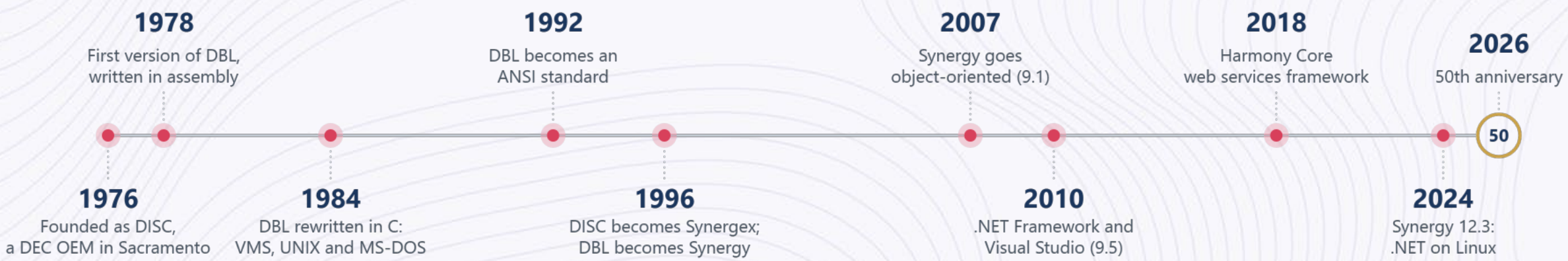
Are we still talking loops or did we shift to graphs yet?

8:34 AM · Jul 18, 2026 · **2.6M** Views

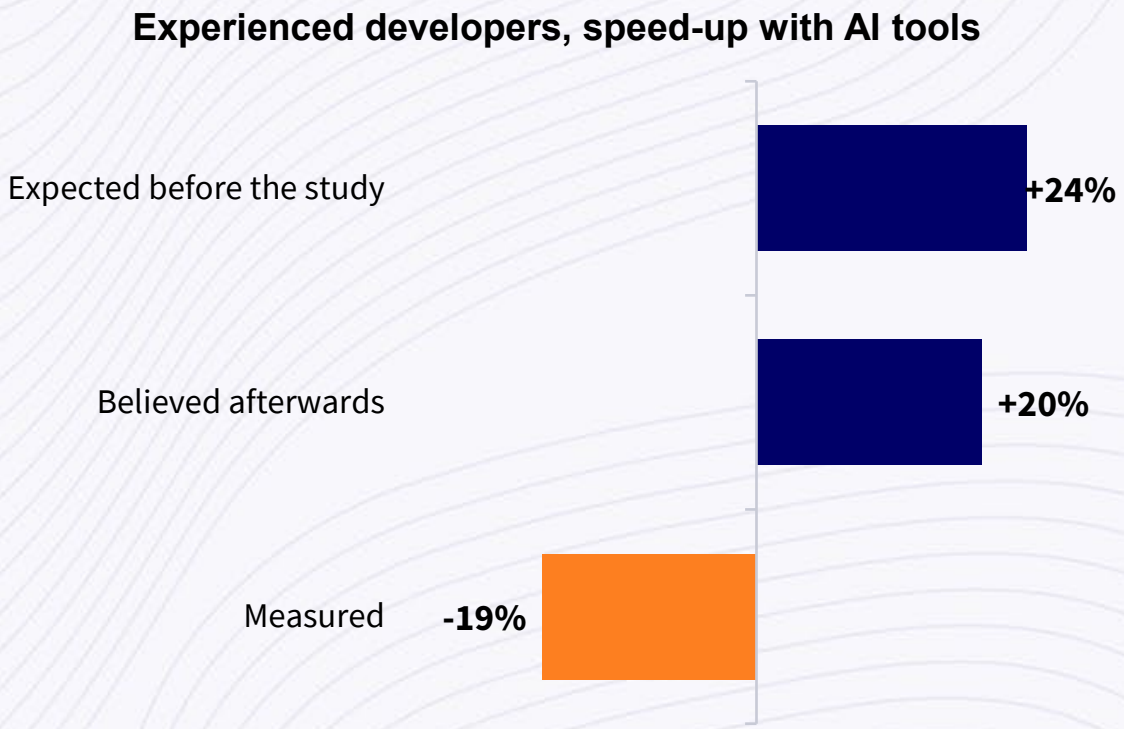


# Same craft. With more.

SYNERGY DEVPARTNER CONFERENCE 2026



# AI Amplifies What You Already Do




- **DORA 2025:** AI is an amplifier. Strong teams get stronger; weak practices show faster.
- **Anthropic 2026:** About 27% of AI-assisted work is work that would never have been scheduled at all.
- **METR 2025, left:** Developers felt 20% faster and measured 19% slower.
- The question is not whether to use AI. It is how to direct it so it amplifies the right things.

Sources: METR, July 2025 randomized trial (16 developers, 246 real issues); METR's February 2026 follow-up (57 developers, 800+ tasks) leaned toward a speed-up but METR calls that signal unreliable. DORA 2025 State of AI-assisted Software Development (nearly 5,000 respondents). Anthropic 2026 Agentic Coding Trends Report.

# What You Will Leave With

- **The day-to-day loop:** Intent and success criteria first, tests before code, the agent drafts, humans gate.
- **A real project, end to end:** SQL Connection rebuilt in Rust. Essentially one engineer, Claude Code and Codex, twelve months of merged pull requests.
- **Lessons learned:** What the gates caught, what we got wrong, and the rules we wrote down.
- **First steps for Synergy DBL teams:** A CLAUDE.md to adapt, a three-step demo on a small DBL project, and the next steps to take back with you.

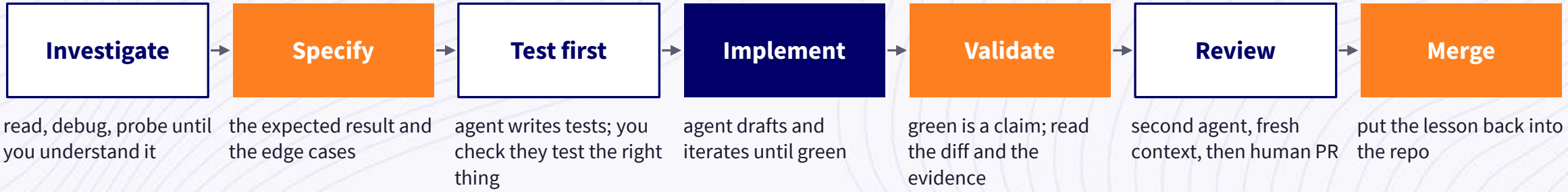
SYNERGY 2026
DEVPARTNER CONFERENCE

# The Workflow

How we direct AI, day to day

# AI-Directed, Not AI-Driven: The Loop






- It is test-driven development with an agent inside it. The understanding ends up in your head; the typing is the agent's.
- Edge cases live in two places: the ones you find while investigating go into the spec and the tests; the ones you missed are what validation and review exist to catch.
- **Anthropic:** "Give Claude a check it can run: tests, a build, a screenshot to compare."
  **OpenAI:** "Humans steer. Agents execute."

# Tell It What Done Means

## Vague

`fix the SQL timeout bug`

- **What the agent has to guess:**
  - Which timeout, on which path, for which backend
  - Whether the contract allows a behavior change
  - What proves it is fixed
- Typical result: a plausible patch to a symptom, a green unit test it wrote for itself, and the word “fixed.”

## Directed

- **Goal:** Customers report %SSC_EXECUTE hanging well past the configured timeout on SQL Server. Make the timeout take effect.
- **Context:** Per-handle timeout state in src/internal.rs. Contract: docs/contracts/options-commands.md. Reproduce with the pattern in dbl-tests/rust/timeout.dbl.
- **Constraints:** Do not change the VTX ABI. Keep the fix behind the backend boundary. If the contract disagrees with the code, stop and tell me; do not edit the contract.
- **Done when:** A focused test fails before and passes after; dev.bat timeout passes in Rust mode and matches legacy output. Report the exact commands, mode, and artifact path.

Illustrative, scrubbed. The four parts follow OpenAI’s Codex guide (goal, context, constraints, done criteria); “the more precise your instructions, the fewer corrections you'll need” is Anthropic's.

# A Map, Not a Manual

- **Always loaded:** CLAUDE.md or AGENTS.md, about 100 lines. What the repo is, how to build and test, the hard rules, where to look next.
- **Per task:** Docs, plans, skills. The entry file points at them; it does not repeat them.
- **Just in time:** grep, legacy source, git history, a probe. References in, full text only when needed.
  - “For each line, ask: would removing this cause Claude to make mistakes? If not, cut it.” (Anthropic)
  - A short AGENTS.md pointing into docs/ as the system of record; the nearest file wins. (OpenAI)

CLAUDE.md at the root of C:\repos\synergy\sde32 (excerpt)

```
## Building Components

### CRITICAL: Use sde_run.cmd for all builds

Claude Code runs in a bash shell where Unix tools
(/usr/bin/link) shadow MSVC tools (link.exe), breaking
native builds. **Always use sde_run.cmd** to run build
commands in a properly configured cmd.exe environment:

  cmd.exe //C "C:\repos\synergy\sde32\sde_run.cmd
              <working_directory> <command> [args...]"
```

The agent can’t learn this from the code. One paragraph, read at the start of every session, sends every build through the wrapper.

# Give It a Check It Can Run

**What counts as a check**

- A build with a real exit code: dbl, dblink, cargo build
- A test suite, or one focused failing test written first
- Program output diffed against a baseline file
- The same probe run against the legacy binary and the new one
- A screenshot compared with the design
- **Exit with evidence:** the exact command, the output, the diff, the artifact path. Faster to review than to re-run, and it works for sessions nobody watched.

```
C:\demo\parts-inventory>test_x000D_un_tests.bat
PASS  load
PASS  list
PASS  find_ok
PASS  find_missing
PASS  receive
PASS  issue
PASS  issue_short
PASS  issue_badqty
PASS  list_after
ALL TESTS PASSED (9 cases)
```

The demo project's harness: nine cases, each diffed against an expected file. The agent runs this itself and reads the result.

“Claude stops when the work looks done. Without a check it can run, ‘looks done’ is the only signal available, and you become the verification loop.” Anthropic, Claude Code best practices

# Green Can Be Deceiving

## How a red test turns green

- Special cases the test input instead of fixing the logic
- Weakens the assertion, or marks the test as skipped
- Edits the expected output or the baseline to match the wrong result
- Swallows the error with a catch-all or a default, so the failure disappears
- Stubs the hard part and leaves a TODO behind
- Changes the harness or the environment instead of the code

## What we do about it

- **Tests belong to the spec step.** The implement step may not edit tests or baselines; the prompt and AGENTS.md say so, a hook can enforce it.
- **Read the test diff on its own.** A changed expected file is a review event.
- **Keep an oracle the agent cannot edit.** A known-good build, a baseline, or a spec.
- **Ask why, not just whether.** “Explain why this fixes the root cause.”
- **Validate the behavior, not the color.** A human runs the thing and looks.

**The agent optimizes for the check you gave it.** If the check can be gamed it will be, not from malice but because green was the only goal it was given. Verify the work, not the suite.



SYNERGY 2026
DEVPARTNER CONFERENCE

# Case Study: SQL Connection, Rebuilt in Rust

Essentially one engineer, two agents, twelve months

# The Project

**What we are building**

- A Rust replacement for the closed-source legacy SQL Connection driver, called by the DBL runtime through the same exports
- SQL Server first; backend agnostic by design
- It must behave like the legacy DLL, quirks included. The oracle is the legacy DLL itself: measured, not assumed
- **Into the unfamiliar:** We forked Tiberius, the open-source Rust SQL Server client, and changed its behavior to match the legacy driver. Reading and changing a foreign codebase is where the agent lowered the barrier most.

**By the numbers**

- **49** pull requests merged, October 2025 to October 2026
- **~40K** lines of Rust
- **668** unit tests, plus matched DBL and C ABI probes
- **~8.3K** lines of agent-facing docs, 8 behavior contracts

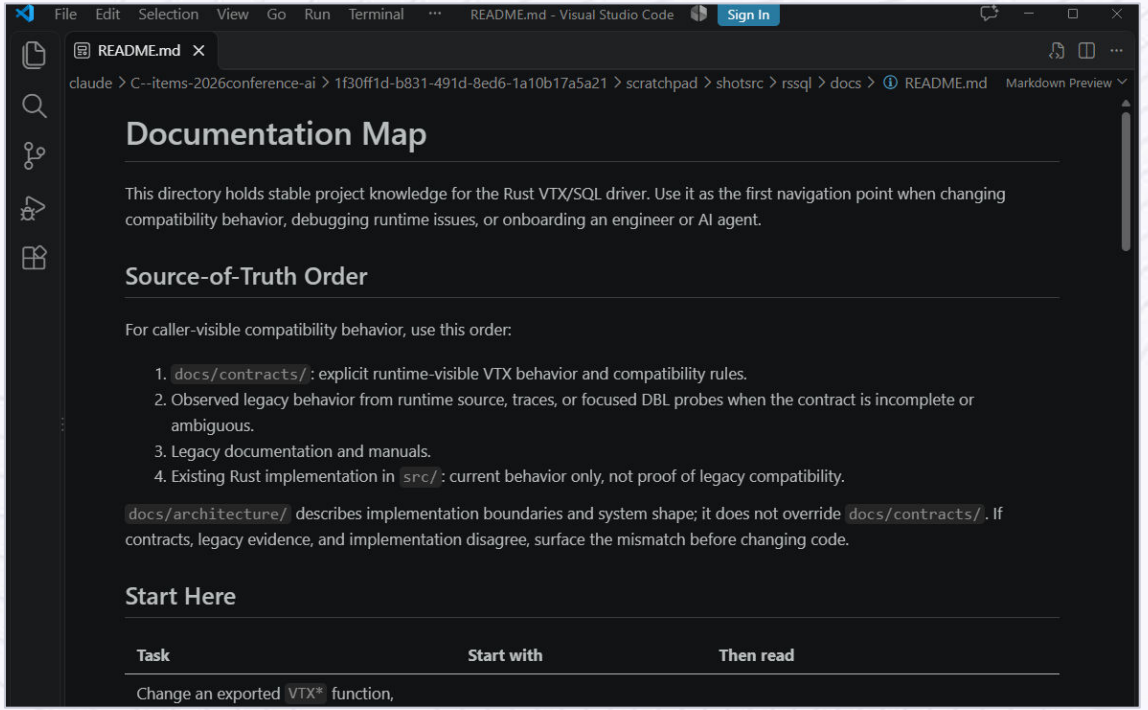
Counts from the project branch history, October 2026.



# A Repo the Agent Can Navigate

```
dbl/src/rssql/
  AGENTS.md              82 lines; points, does not explain
  docs/README.md         task -> start here -> then read
    contracts/ (8)       source of truth for behavior
    architecture/ (5)    shape; never overrides contracts
    compatibility-matrix.md  implemented vs stubbed
    runtime-test-map.md      which probe proves what
    legacy-evidence.md       truth sources, in order
    odd-legacy-behaviors.md  measured quirks, decisions
  plans/                 bounded, status line; not truth
  dbl-tests/ test-harness/  probes, Perl harness
  src/                   ffi, internal, backend, sqlserver
```

The agent always reads about 80 lines, then one row of the task table. One change reads one contract and one test-map row, not 8,000 lines of docs.




docs/README.md as the agent sees it: the same map for a new engineer and for the agent. Contracts change in the same PR as behavior, so it does not rot.

# Evidence Precedence: Never Guess



1. **Explicit contract:** docs/contracts. If it is written there, it wins.
2. **Observed legacy behavior:** Runtime source, traces, focused probes against the real legacy DLL.
3. **Legacy documentation:** Useful, but it describes intent, not what the DLL actually does.
4. **Current Rust behavior:** Our tests prove what our code does today. They never prove legacy parity.

**Every claim carries a label:** Documented contract, observed legacy behavior, legacy documentation, current Rust behavior, or unverified. If the probe or the environment is unavailable, say unverified; never fill the gap from assumptions.

**This is where expertise matters.** The engineer defines what counts as truth; the agent applies that ranking the same way on every task.

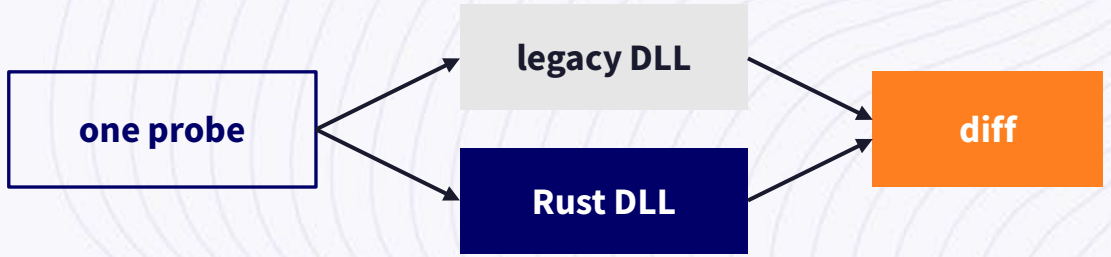
# Tests and Probes Close the Loop



**Four layers, all written and run by the agent**

- **Cargo test: 668** unit tests. Rust-level behavior only, no SQL Server needed.
- **Direct C ABI probes:** One probe binary that takes the DLL path; point it at legacy or at Rust.
- **DBL probe programs:** The customer's view of the driver, swapped between the two DLLs.
- **Test suite harness:** Smoke, a named test, or the whole inventory. Baseline comparison on by default; failures classified.
- **The report it must hand back:** Exact command, working directory, mode, result, artifact path, baseline status. Never a credential.

## The matched-probe pattern




- Same inputs, two binaries, compare the output byte for byte.
- A difference becomes either a contract update or an entry in the odd-legacy catalogue. That choice is a human one.
- The agent is not allowed to say "tests pass." It says which tests, in which mode, against which artifact.

# Odd Legacy Behaviors: The Human Decides

docs/odd-legacy-behaviors.md (excerpt)

```
## Status Values
Replicated              Port matches legacy deliberately.
                        Do not "fix" it.
Divergent, accepted     Port differs; we decided that is
                        correct or unavoidable.
Decision pending        Port differs; target not chosen yet.
Needs re-verification   Old enough to re-probe first.
Runtime-owned issue     Comes from the DBL runtime, not
                        the driver.

## Adding An Entry
Copy this skeleton, keep the field order, and cite a
probe run.
| Area | Applies to | Legacy behavior | Port behavior |
| Status | Last verified |
### What Happens   ### Why It Is Odd   ### What The Port Does
### Recommendation ### Open Items     ### Evidence
```

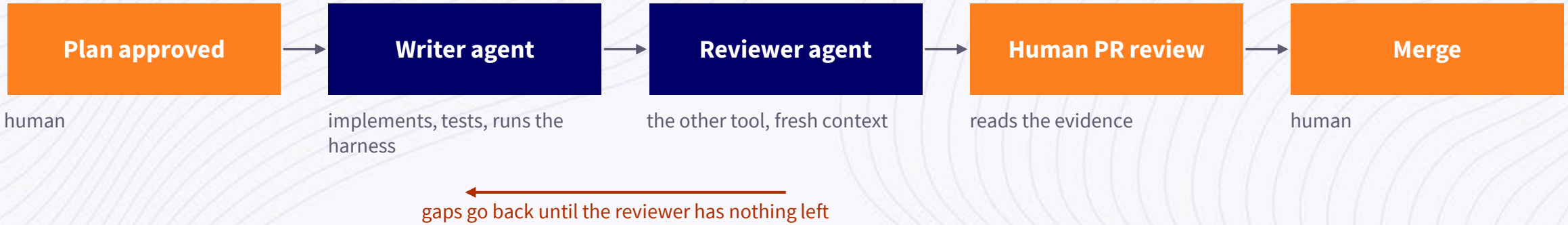
The catalogue's own template: the status values and the entry skeleton every quirk is recorded with.

**Every entry records**

- **Area:** The export and the %SSC routine it serves
- **Legacy behavior:** What the old driver really does, as measured by a probe, not what the manual says
- **Port behavior:** What the new driver does today and where it differs
- **Status:** Replicated, or divergent with the reason, or decision pending
- **Last verified:** The date and the probe run that proves it. No probe run, no entry.

**The agent measures and proposes.** The engineer decides whether a quirk is a contract or a bug.

# Two Agents: Writer and Reviewer




- **Why a fresh context:** “A fresh context improves code review since Claude won’t be biased toward code it just wrote.” (Anthropic) The reviewer sees the diff and the criteria, not the reasoning that produced them.
- **What it is asked for:** Edge cases the probes missed, ABI or ordinal breaks, contract drift, missing coverage, intent mismatch with the PR description. Not formatting; cargo fmt owns that.
- **The guard:** A reviewer told to find gaps will find some, even when the work is sound. We scope it to gaps that affect correctness or the stated requirements.
- **Where the human is:** “At OpenAI, Codex reviews 100% of PRs.” At Synergex, agents review first, and every PR still ends with a human approval in Azure DevOps.



SYNERGY 2026
DEVPARTNER CONFERENCE

# Human Review Gates and Lessons Learned

What the gates caught, and what we changed

# Where the Humans Are

1. **Intent and done criteria.** Before any code: what, why, and the check that proves it.
2. **Plan approval.** Plan mode, or a plan file with a status line. Edited, then approved.
3. **Contract changes.** “If implementation and contracts disagree, stop and surface the mismatch.” Never rewrite a contract to fit the code.
4. **PR review.** Read the evidence: commands run, mode, artifacts, baseline status, coverage. Agents reviewed first.
5. **Merge and retrospective.** What should the repo say so this does not recur? A rule, a probe, a harness case.

.claude/commands/prreview.md, our review command for Claude Code (excerpt)

```
4. For each changed file, read the full file (not just
   the diff hunk) so you understand the surrounding
   context, types, imports ...
5. If there are new or modified tests, read them too and
   check whether they actually cover the changes.

## What to look for
Go through each category. Only report real findings --
do not invent issues or pad the review.
- Bugs and logic errors ...
- Breaking API or contract changes ...
- Missing or outdated tests for the new behavior
- Intent mismatch (does the implementation actually
  achieve what the PR description says it should?)

If the diff looks clean and you have no findings, say so.
A clean review is a valid outcome.
```

AI-assisted review, human-decided: The human still decides what blocks the merge.

**A rule for findings:** When a reviewer, human or agent, finds an issue, ask the agent for a test that proves it. A finding you can run becomes coverage; one you cannot is an opinion.

# Lessons Learned



**1. The agent trusts its environment, so make sure it should**

- It reports what it sees. If the build is stale or the setup is wrong, it will confidently report the wrong thing.
- So we verify the environment before we trust the result.

**2. The agent generalizes from whatever you show it**

- Show it one example and it will write the rule as if it covers every case.
- So we test the failures and edge cases before anyone says "it matches."

**3. Ask what the tests don’t cover, not just whether they pass**

- Passing tests only vouch for the code they touch. Everything else is unverified.
- So every review asks: what isn’t covered?

**4. Keep the work small enough to actually review**

- Agents will happily hand you a 2,000-line diff.
- Review is the bottleneck now. Size the work for the reviewer.

Every lesson became a line in AGENTS.md, a probe, or a harness case. OpenAI’s advice is the same: When the agent makes the same mistake twice, run a retrospective and update the file.



SYNERGY DEVPARTNER CONFERENCE 2026

# Applying This to Synergy DBL

First steps you can take next week

# Why DBL Needs More Guidance



## Weak priors

- There is very little public DBL for models to learn from. Left alone they guess syntax, blur traditional and .NET, and invent routines.
- **Do this:** Point the agent at the Synergy docs and at real code in your repo. Name the canonical example file.

## The compiler is the oracle

- The model cannot know if the code is right. The compiler, linker, and runtime can.
- **Do this:** Make dbl, dblink, and dbr the check it must run, with -qcheck on. Diff output against a baseline.

## What it cannot see does not exist

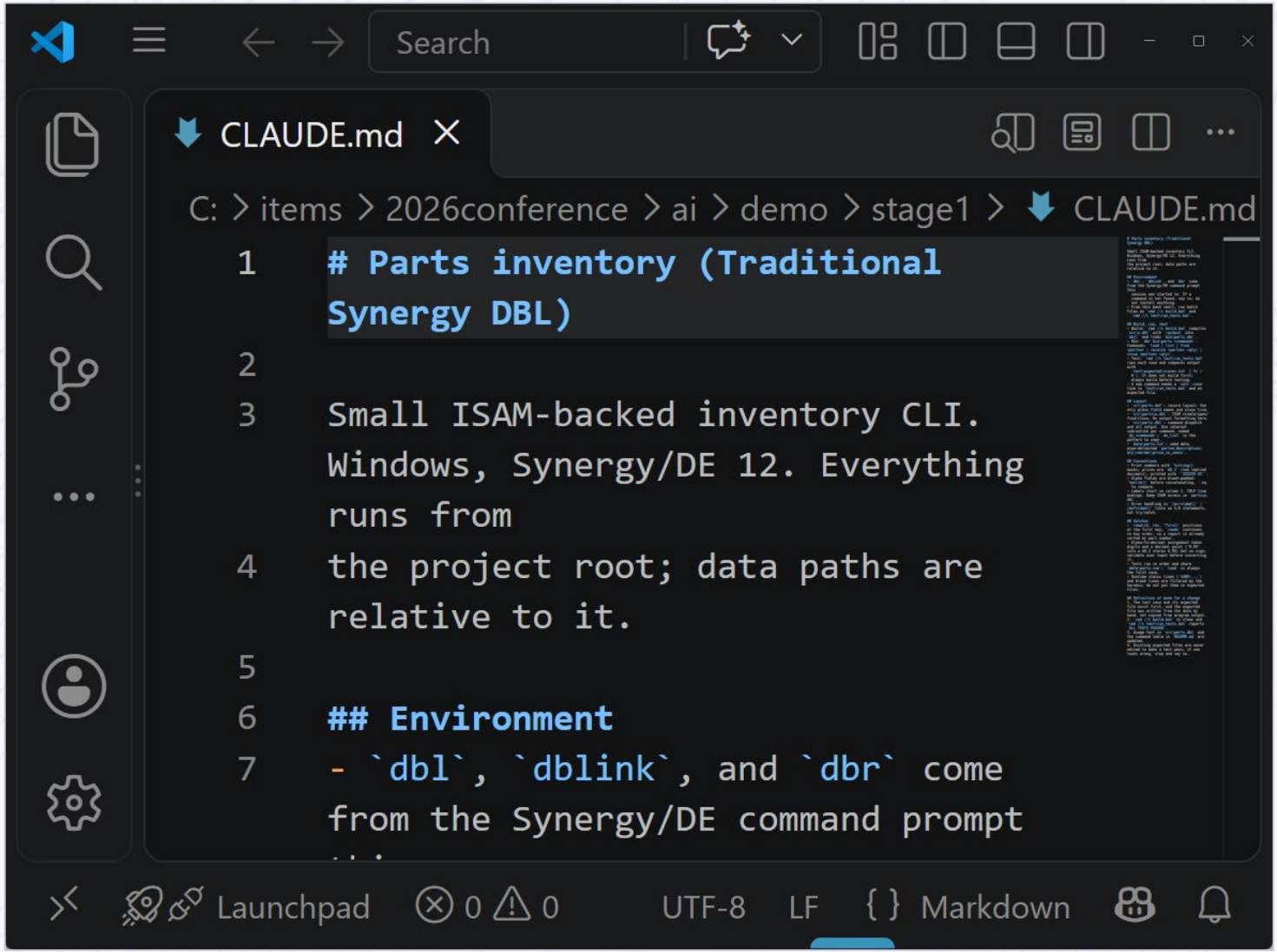
- Record layouts, repository definitions, logicals, build scripts: If they live in someone's head, the agent works blind.
- **Do this:** Encode them in the repo: .def includes, a build script, a short CLAUDE.md that names them.

## Expertise is the multiplier

- ISAM locking, alpha padding, decimal rounding, OpenVMS quirks: the agent will not infer these, and it will not know when it is wrong.
- **Do this:** You validate semantics; the agent handles drafting and iteration. Humans steer.

DORA 2025 lists "AI-accessible internal data" among its seven AI capabilities. Anthropic calls the practice context engineering.

# A CLAUDE.md for a DBL Project




demo\stage1\CLAUDE.md, the real file from the demo project. Every command on it was run against the harness.

- **Environment:** What the code cannot tell it.
- **Build, run, test:** The checks it must run and the diff that defines pass. The harness does not build; build first.
- **Layout and conventions:** Only what differs from what it would do anyway, plus a file to copy.
- **Gotchas:** What cost you an afternoon.
- **Definition of done:** Tests and expected file first; never edit an expected file to go green.
- **One file for both tools:** Write AGENTS.md once; CLAUDE.md is one line, @AGENTS.md. That's how our Visual Studio integration repo does it.

# Demo: An AI-Friendly DBL Project

**1. Clone a small DBL project.** Parts inventory: record layout, ISAM routines, a CLI, seed data, build.bat, a harness with nine green cases.

**2. Make it AI-friendly.** The CLAUDE.md you just saw. /init drafts it; you correct what it guessed.

**3. Add a feature, test first.** A reorder report, from the prompt on the right.

### Watch for

- It reads CLAUDE.md instead of guessing the build
- It writes the expected file first and stops for your OK
- It compiles and runs the harness itself
- If a case fails, it fixes the code, not the expected file

demo\prompt-reorder.txt

```
Add a `reorder` command to the parts CLI.

Goal: list every part whose quantity on hand is at or below its
reorder level, in part-number order, one line per part in the same
format as `list`, followed by a line "<n> parts at or below reorder
level".

Context: command dispatch and output live in src\parts.dbl (copy the
do_list pattern); the record layout is src\parts.def; ISAM routines
are in src\partsio.dbl. CLAUDE.md has the build and test commands.

Constraints: do not change partsio.dbl or parts.def. Do not edit any
existing expected file.

Test first: add a `reorder` case to test\run_tests.bat right after
`load`, and write test\expected\reorder.txt by hand from
data\parts.txt. Show me that file and wait for my OK before
implementing.

Done when: `cmd //c build.bat` is clean and `cmd //c
test\run_tests.bat` prints ALL TESTS PASSED. Paste the test output.
```

# Next Steps

1. **Install Claude Code or Codex and run /init on one project.** Then add the build and test commands yourself; that is what it cannot guess.
2. **Use it read-only first.** Ask it how a routine works, and learn how it reads your code before it writes any.
3. **Let it write reproduction programs and tests.** It must compile, link, and run them itself.
4. **Ask for evidence, not a summary.** The command, the output, the diff.
5. **Plan mode for anything that touches more than one file.** Small sessions, clear the context between tasks, worktrees for parallel work.
6. **Add a second agent as reviewer and a hook for format and lint.** A hook is deterministic; a sentence in a file is advisory.
7. **Same mistake twice:** Run a retrospective and update the file.

**Guardrails:** No customer data in prompts, synthesize the shape you need; if you cannot verify it, do not ship it; measure before you claim a number; tell your team the AI stance.

# Takeaways



1. **Humans steer.** Intent, success criteria, and judgment stay with the engineer.
2. **Context is the scarce resource.** Give the agent a map, not a manual.
3. **Give it a check it can run.** Compiler, tests, baselines, matched probes. Review evidence, not claims.
4. **Separate writer from reviewer.** A fresh context finds what the author cannot. Human gates at plan, contract, and PR.
5. **Encode every lesson in the repo.** A rule, a probe, or a harness case. That is how it compounds.

# Questions?

**Taras Yakovchuk**
Associate Software Development Manager, Synergex

**Michael Oleshchuk**
Jr. Software Engineer, Synergex

**Today, 1:30 pm:** Rebuilding SQL Connection from the Ground Up. Mike on the driver itself.

**Further reading**

Claude Code best practices: code.claude.com/docs/en/best-practices

Effective context engineering for AI agents: anthropic.com/engineering

Harness engineering: openai.com/index/harness-engineering

Codex best practices and AGENTS.md: developers.openai.com/codex, agents.md

DORA 2025 State of AI-assisted Software Development: dora.dev

METR developer productivity studies: metr.org