

CEN 5016: Software Engineering

Fall 2026

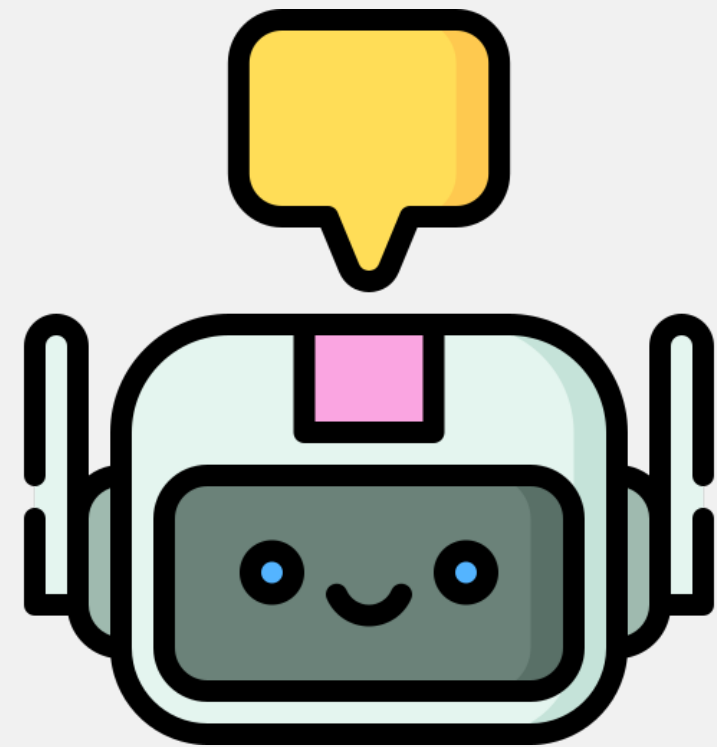


University of
Central Florida

Dr. Kevin Moran

Week 6 Class I:

Agentic Software Engineering I:
How Agents Work & What to
Delegate





01



What agentic AI is

From autocomplete to agents

02



The agentic harness

The loop, the context window, and what you control

03



The tool landscape

Interaction modes, autonomy levels, where agents run

04



What the evidence says

Benchmarks, agent pull requests, productivity

05



What to delegate

The delegation map, limitations, and three kinds of debt



The last time an AI tool gave you wrong code,
how did you find out, and how long did it take?



Seconds

It didn't compile, or a test failed



Days

A teammate caught it in review



Weeks

A user hit it after it shipped

PART 1

What Is Agentic AI?



From Autocomplete to Agents



Autocomplete

Suggests the next line as you type.

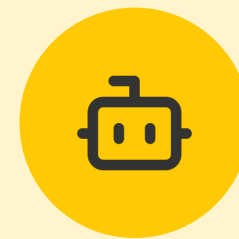
a line



Chat

Answers a question or writes code on request.

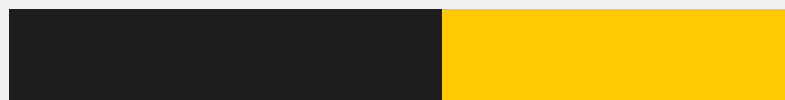
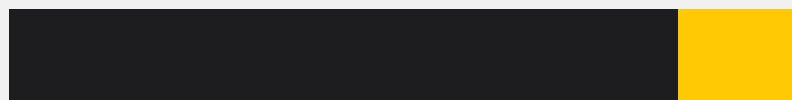
an answer



Agent

Plans, calls tools, checks results, and iterates.

a whole task



■ you ■ the AI

Your role shifts from writing code to directing and reviewing it.

How LLMs Generate Code

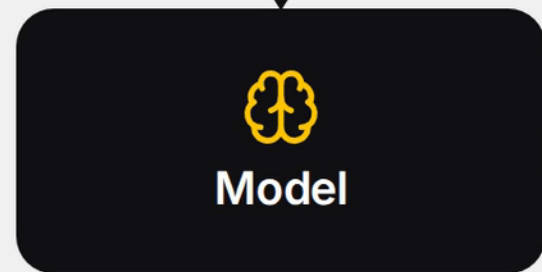


CONTEXT WINDOW: EVERY TOKEN SO FAR

def add (a , b) :

0

new tokens generated



MOST LIKELY NEXT TOKENS

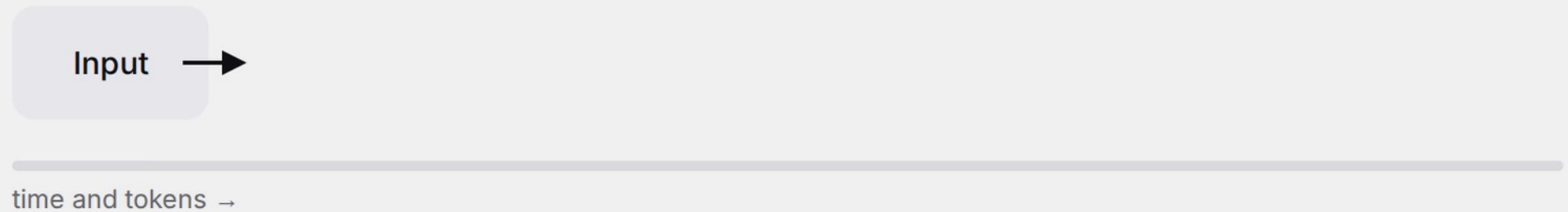


Key implication: the output can only be as good as what is in the context window.

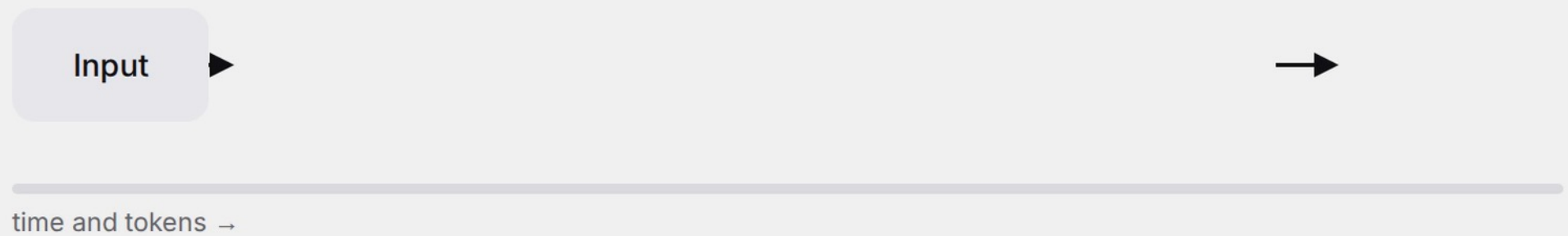
Reasoning Models



Standard LLM
answers immediately



Reasoning model
"thinks" before answering



Slower

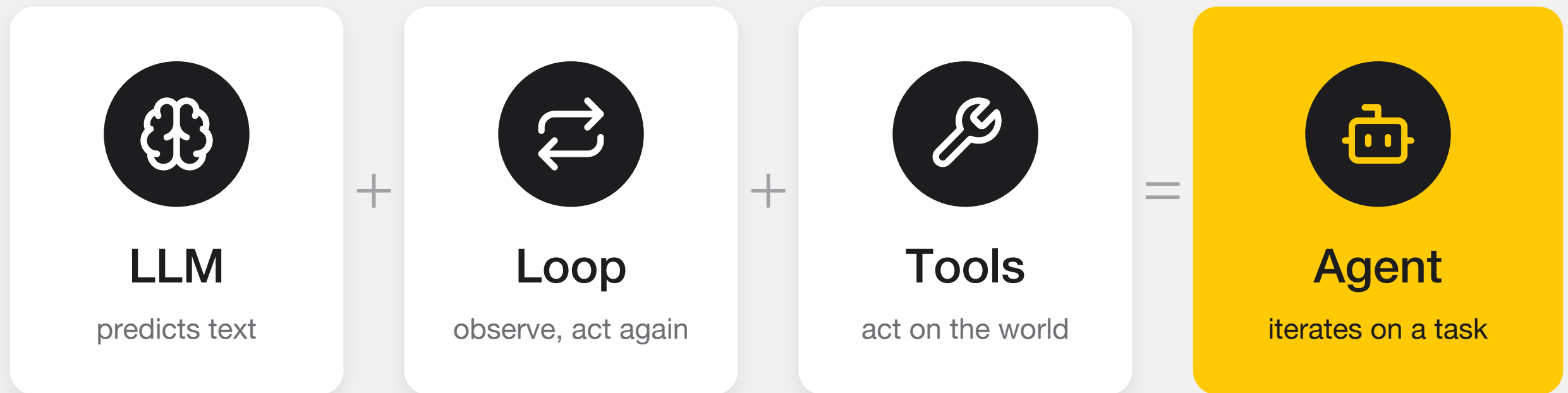


More tokens, more cost



Better on multi-step problems

What Makes an LLM an Agent



TYPICAL TOOLS

read and edit files

terminal

run tests

search

APIs (MCP)

subagents

LLM alone

Answers once and forgets. Cannot run your code or check its own work.

Agent

Reads files, runs commands, sees the results, and tries again.

PART 2

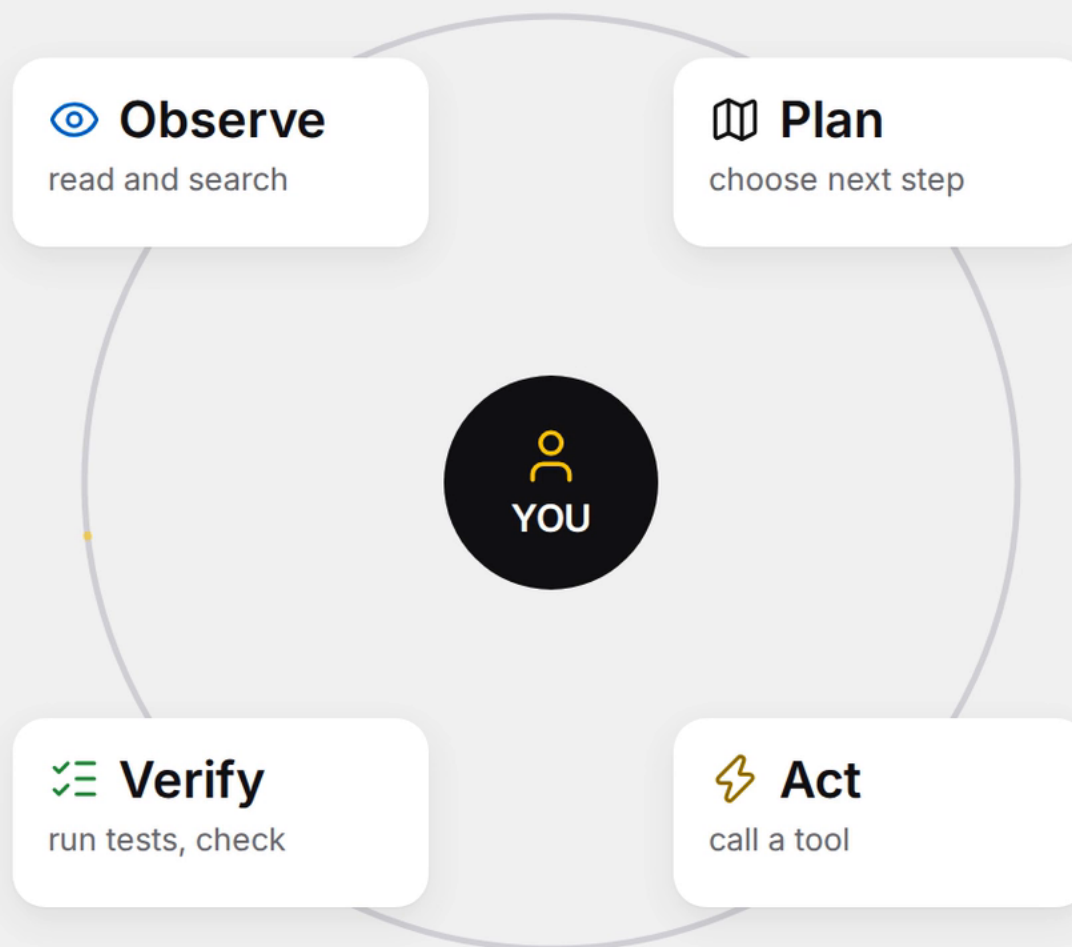
The Agentic Harness



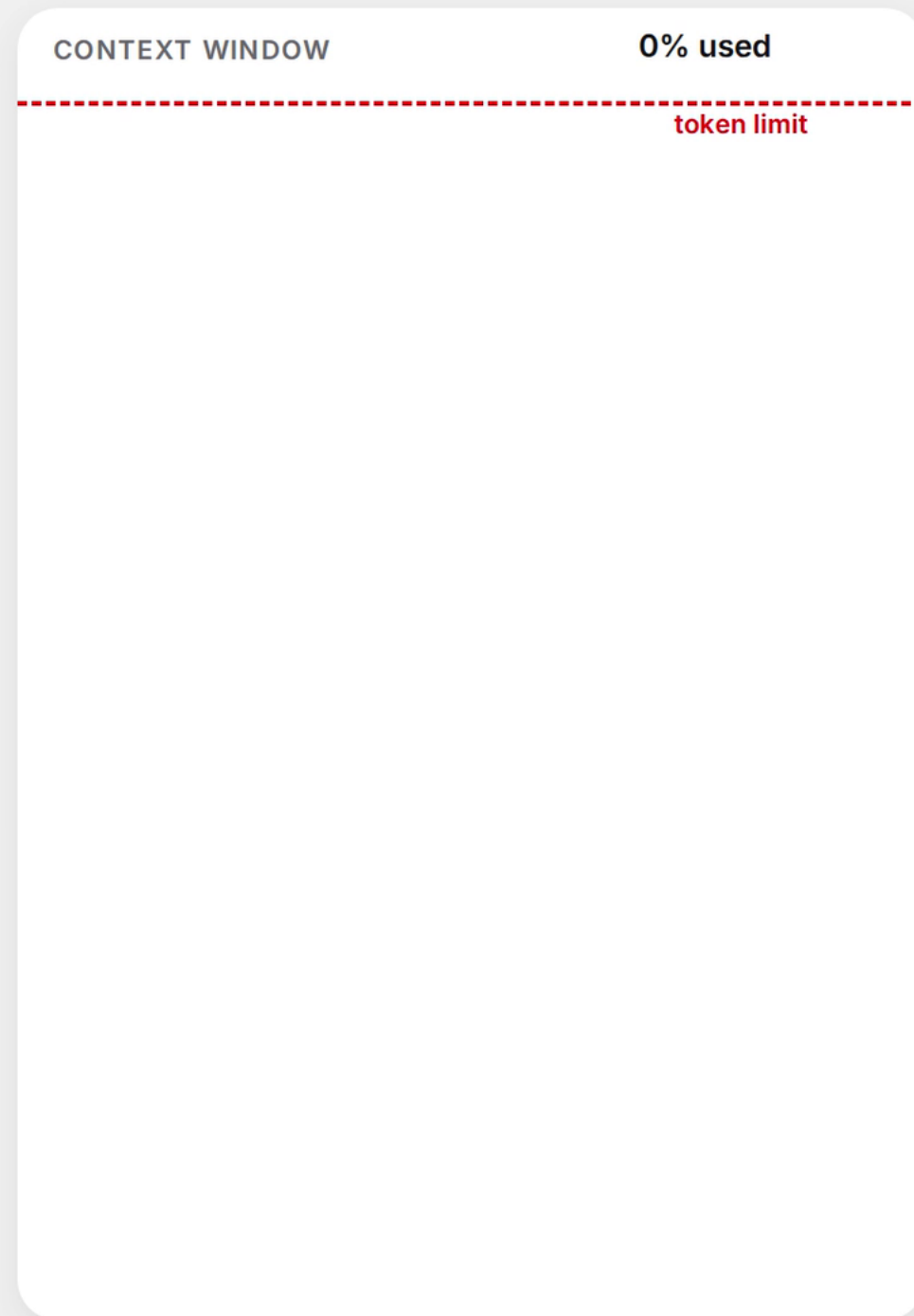
The Agent Loop



Task: fix the failing PNG-export test



The Context Window



Who Fills the Context Window



CONTEXT WINDOW

System prompt

instructions, rules

Tool definitions

what the agent can call

Instructions file

CLAUDE.md / AGENTS.md

Files you point at

@src/export.ts, specs

Tool results

file contents, test output, logs

Your request

"Fix the export bug..."

You fill

instructions file, pointers, your request

The harness fills

system prompt, tool definitions, results

A shared job.

When the window fills, the harness decides what to trim. Anything that must survive belongs in a file.

Memory Across Sessions



SESSION 1

IN-CONTEXT MEMORY

SESSION 2

IN-CONTEXT MEMORY

EXTERNAL MEMORY: FILES ON DISK



CLAUDE.md

Harness Architecture



External services

MCP servers · files · APIs · databases

Harness

prompt · tools · hooks · permissions



LLM

the only part that “thinks”



System prompt and instructions file



Tools: files, shell, search, APIs



Hooks: scripts on lifecycle events



Permission tiers



Subagent configurations



MCP: connects external services

“A harness is largely a delivery mechanism for good context engineering.”

What You Configure in a Harness



Build from scratch
only if the agent is your product

You adapt it
instructions file · permissions · MCP servers · subagents · what you point it at



Ships with the tool
the loop · tools · compaction · permission system

You are not building a harness. You are specializing a general one for your project.

Coding vs. General Harnesses



GENERAL HARNESS

web search, sandbox

everything in the window

generic

no ground truth

chat cannot break anything

Tools

Context

Guidance

Feedback

Safety

CODING HARNESS

read and edit files · grep · shell · git · tests

search the repo, read only what matters

read before editing · match conventions ·
small diffs

tests, types, and builds give automatic
feedback

approval gates · branches · sandboxing

Same model, same loop: the difference is tools, context policy, and verification.

Demo I: The Agent Loop in Action



```
claude — ~/excalidraw

Claude Code v2.1.284
Opus 5.5 · Claude API
~/excalidraw

Get to finished work sooner with Opus 5.5. Switch anytime with /model.

medium · /effort

> try "how does <filepath> work?"

auto mode on (shift+tab to cycle) · ← for agents
```

CHAPTERS

0:01

Switch to plan mode (read-only)

0:05

The task: how PNG export works

0:11

Observe: search, then read

0:29

Verify: run the related tests

0:52

/context: what is in the window

1:04

/clear: the memory is gone

Watch for: the agent searching before it reads, the tests it chooses to run, what /context shows, and what /clear forgets.

PART 3

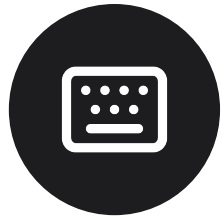
The Tool Landscape



Interaction Modes

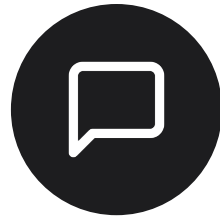


MORE AUTONOMY



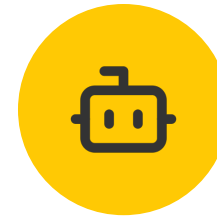
Inline

Suggests as you type



Chat

You ask; it generates



Agentic

Plans and runs many steps



Background

Runs without you

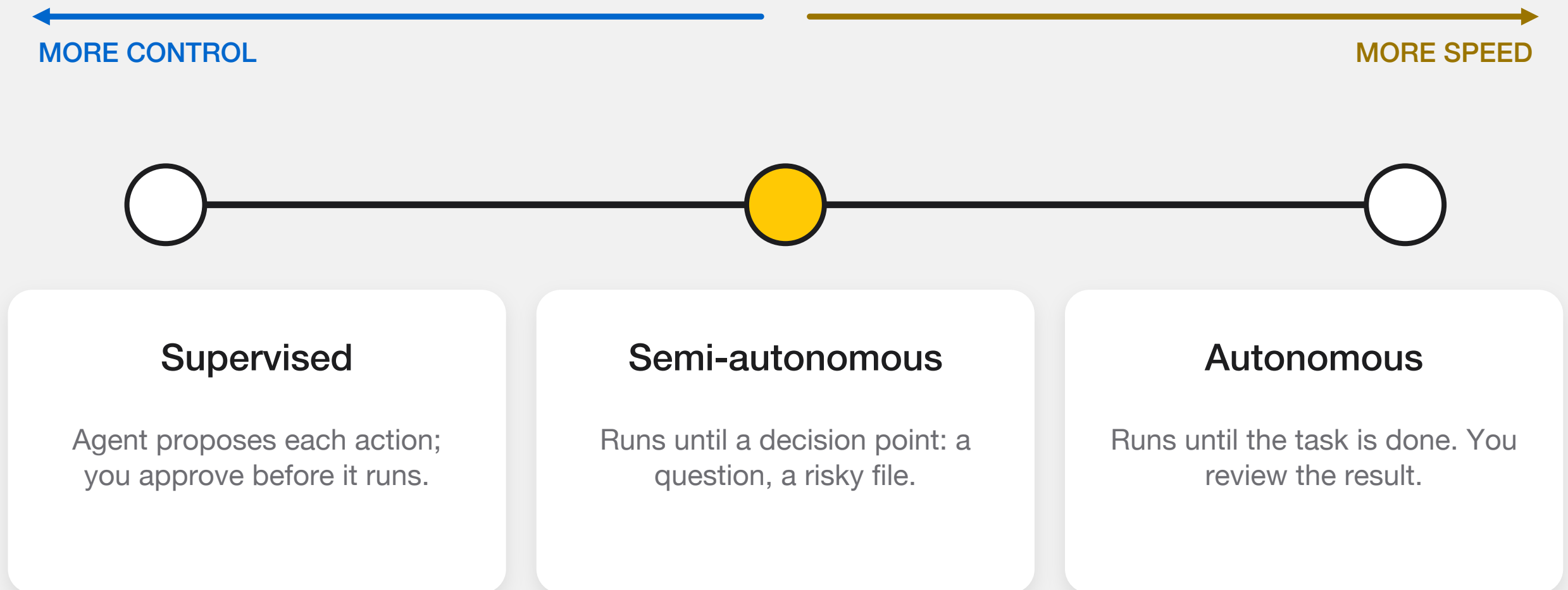
YOU REVIEW
every suggestion

YOU REVIEW
every answer

YOU REVIEW
at checkpoints

YOU REVIEW
the final result

Autonomy Levels



THE RIGHT LEVEL DEPENDS ON



Task risk



Familiarity with the code



Trust built on this kind of task

Where Agents Run



Command line

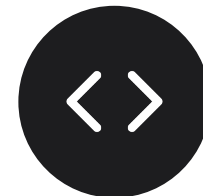
Claude Code, Codex CLI

CODEBASE ACCESS



Full filesystem, shell, git, MCP

Best for: Long tasks and scriptable workflows



IDE

Cursor, VS Code + Copilot

CODEBASE ACCESS



Open files, editor state, inline diffs

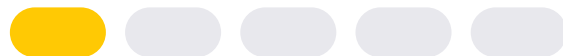
Best for: Tight in-editor iteration



Web chat

Claude.ai, ChatGPT

CODEBASE ACCESS



Only what you paste in

Best for: Questions and exploration



Desktop and browser

Computer use, Claude in Chrome

CODEBASE ACCESS



Sees the screen, controls apps

Best for: GUI and cross-app tasks

PART 4

How Capable are Current Agents?



Measuring Agents with SWE-bench



GitHub issue

real bug report
+ repo snapshot



Agent

explores the repo
and edits code



Patch



Hidden tests

fail → pass ● ● ●
pass → pass ● ● ● ● ●

2,294

real issues from 12 Python repos

500

human-validated tasks in
“Verified”

1 bit

per task: do the tests pass?

Limits of SWE-bench Scores



7.8%

of “resolved” patches fail the full developer test suite

Wang et al., 2025

29.6%

behave differently from the developer’s real fix

Wang et al., 2025

59.4%

of audited Verified problems had flawed tests

OpenAI, Feb 2026



Contamination

Frontier models had seen some problems and solutions in training. Higher scores partly reflect memory.



The lesson

“Tests pass” is evidence, not proof. The same is true for your own agent’s PRs.

Agent Pull Requests on GitHub



PR acceptance rate

AIDev, 456K+ agent-authored PRs

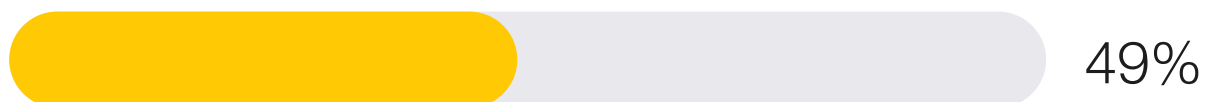
Human PRs



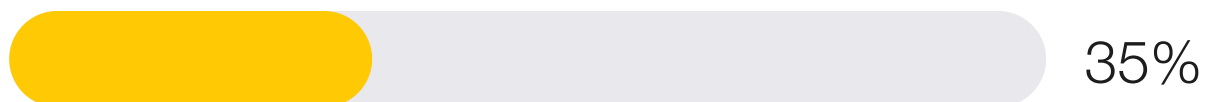
OpenAI Codex



Devin



GitHub Copilot



9.1%

of agent PRs change code complexity, vs. 23.3% of human PRs

Fast, but routine.

Agents mostly make simpler changes, and they are accepted less often than humans.

Test Coverage of Agent PRs



Our ICSME 2026 study of 4,882 agentic pull requests

50.4%

of PRs that change code under test
include no test changes

81–86%

of error-handling lines agents add are
never run by tests

Agent-changed lines run by existing tests

61.5%

Java

27.0%

Python

AI and Developer Productivity



Change in task completion time when using AI

Forecast

predicted before

Self-estimate

believed after

Measured

what happened

← faster

slower →

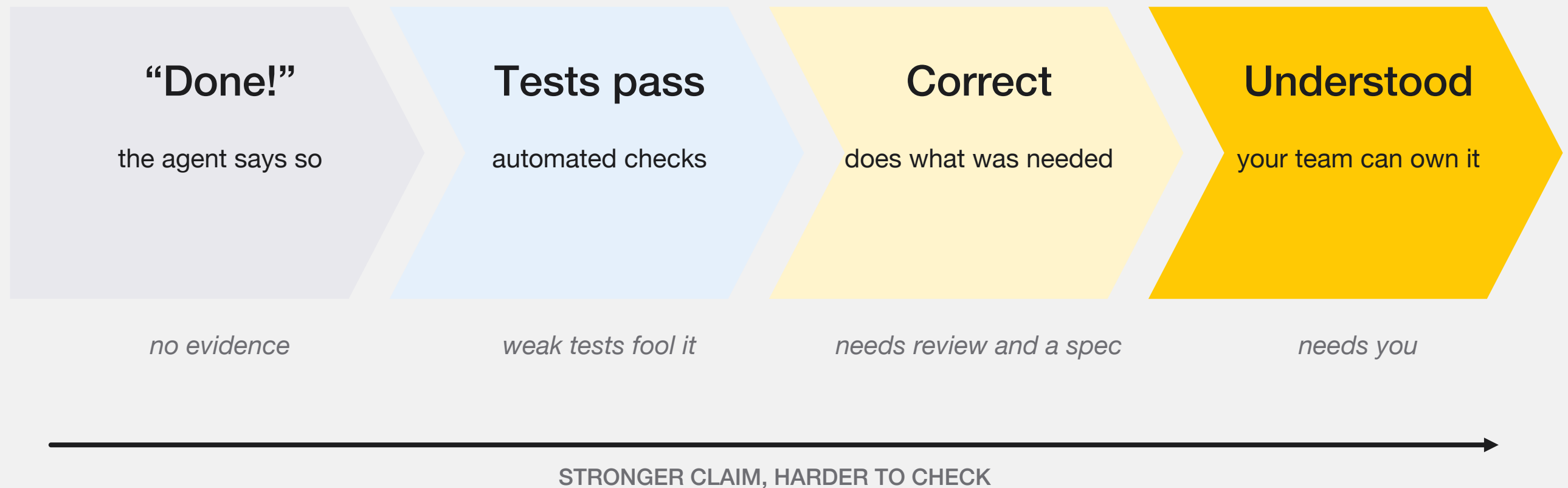
+19%

slower on average
while developers believed they
were 20% faster

Why? Large, familiar codebases,
high quality bars, and time spent
prompting, waiting, and fixing
output.

Source: Becker et al., METR, 2025

Levels of Confidence in Code



Benchmarks and most agents
stop at “tests pass.”

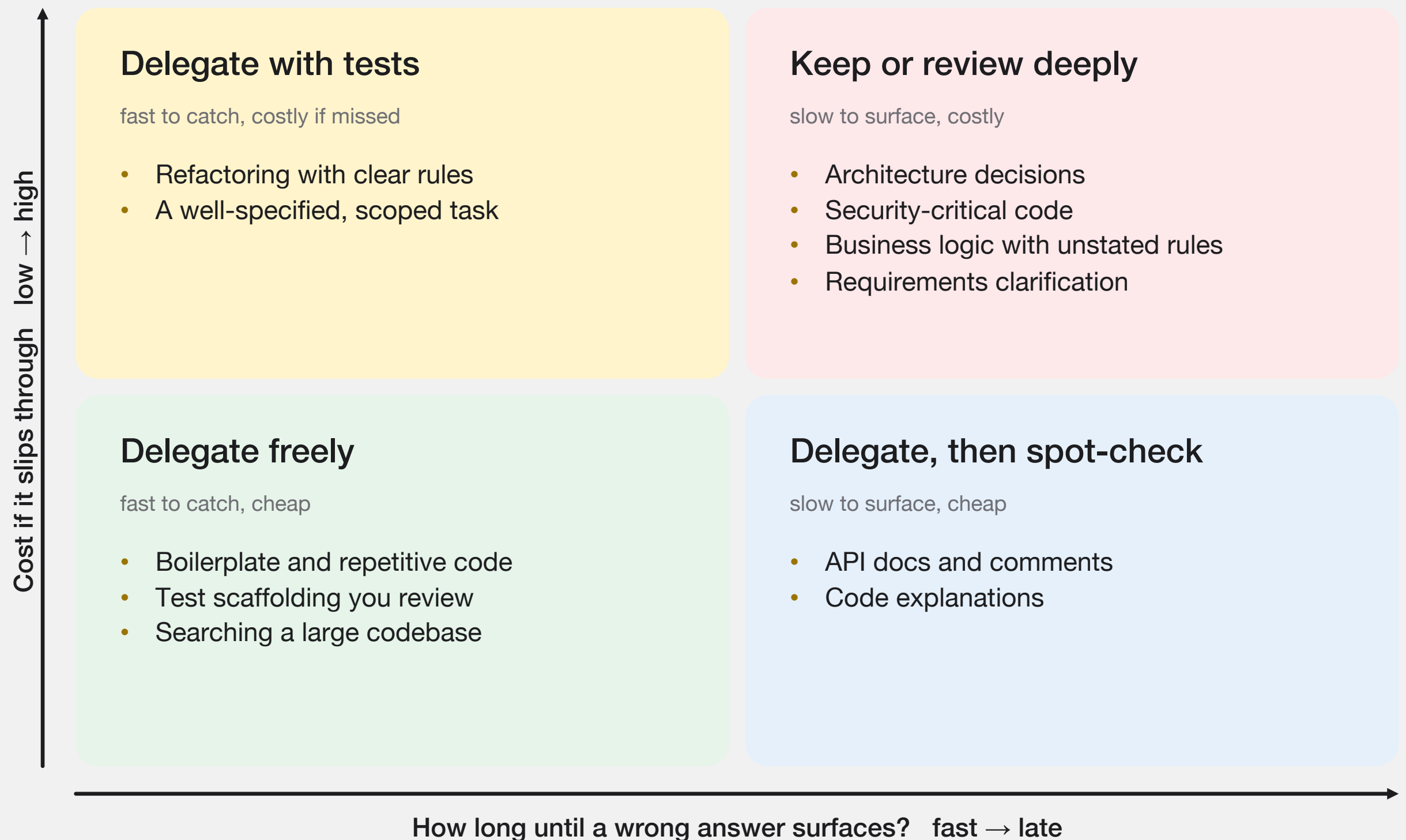
Your job as the engineer
is “correct” and “understood.”

PART 5

What to Delegate to an Agent



Deciding What to Delegate



Activity: Classify These Tasks



Add a keyboard shortcut to
Excalidraw's toolbar

Change how Gitea expires session
tokens

Write unit tests for Actual Budget's
CSV importer

Decide whether Outline should
switch editor frameworks

Rename a utility function across
Chatwoot

Document Medusa's checkout API



Technical Limitations of Agents



Hallucinations

Confident, plausible, and wrong. Worst on obscure libraries, internal conventions, and recent API changes.



Context drift

In long sessions the agent loses the original goal and optimizes for the wrong thing.



Intent blindness

It solves what you said, not what you meant. Vague specs compound.

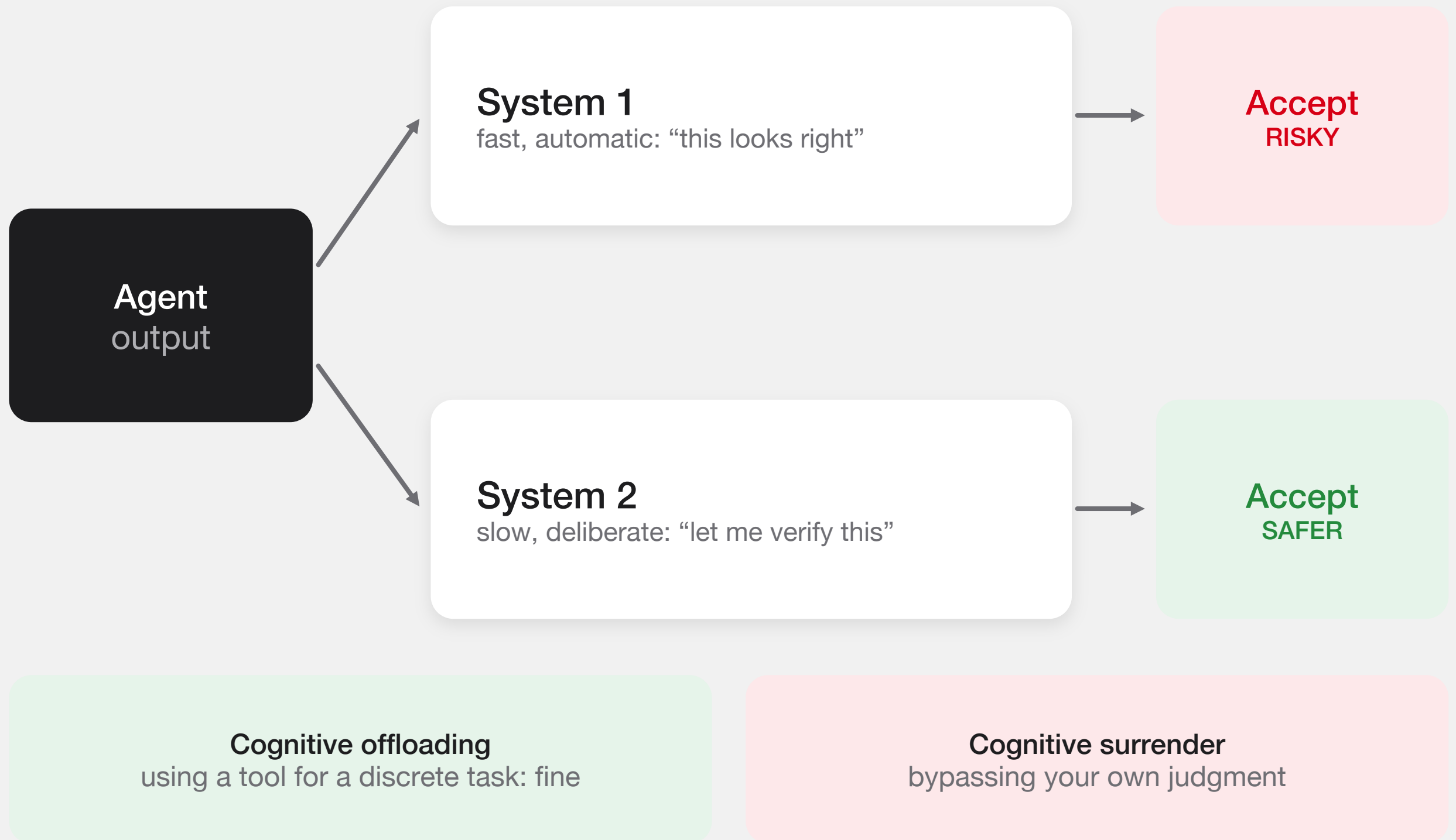


Context limits

It cannot hold a large codebase at once, so it must choose what to read, and may choose wrong.

Common cause: thin training data, and a second model shares the same blind spots.

Cognitive Surrender



Technical, Cognitive, and Intent Debt



AI may reduce technical debt
**while accelerating cognitive
and intent debt.**



Intent debt

lives in artifacts

goals, constraints, and decisions never
written down

< > Technical debt

lives in the code

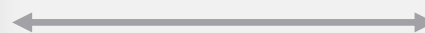
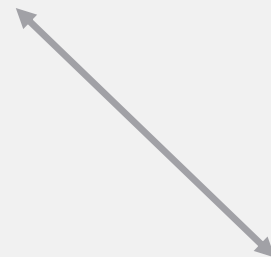
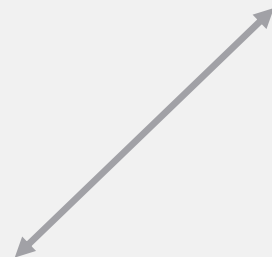
messy code, shortcuts, compromises



Cognitive debt

lives in people

the team's shared understanding erodes



amplify / repair

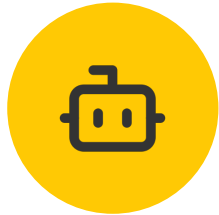


The code may work.

The team may not understand it.

And no one wrote down what it was supposed to do.

This is the core risk of AI-assisted development.



An agent is a model, a loop, and tools

and it is only as good as its context.



Benchmarks and "feelings" overstate correctness

"Tests pass" and "I feel faster" are not proof. Measure and verify.



Delegate by risk

How fast would a mistake surface, and what would it cost?

Next time: workflows, verification, security, and responsibility



- Chaparro, O. “Introduction to Agentic Software Engineering.” CSCI 435/535 lecture slides, William & Mary, 2026.
- Jimenez, C. E., et al. “SWE-bench: Can Language Models Resolve Real-World GitHub Issues?” ICLR 2024.
- Wang, Y., Pradel, M., & Liu, Z. “Are ‘Solved Issues’ in SWE-bench Really Solved Correctly?” arXiv:2503.15223, 2025.
- OpenAI. “Why we no longer evaluate SWE-bench Verified.” Feb 2026.
- Li, H., Zhang, H., & Hassan, A. E. “The Rise of AI Teammates in SE 3.0.” arXiv:2507.15003, 2025 (AIDev).
- “Test Coverage Analysis of Agentic Pull Requests.” ICSME 2026 (NIER). arXiv:2607.18057.
- Becker, J., et al. “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity.” METR, 2025.
- Shaw & Nave. Cognitive surrender (as cited in Chaparro, 2026).
- Storey, M.-A. “From Technical Debt to Cognitive and Intent Debt.” arXiv:2603.22106, 2026.