FRED TALKS

8th April 2026

CONTENTS

"Determinism" is the Biggest Cope in AI Adoption

HAN, NOT SOLO · 08 APR 2026 · SOURCE

We've never had determinism in software. We just had the illusion of it.

Here's a fact that most people outside computer science don't know: in 1936, Alan Turing proved that there is no way to build a program that can check whether another program will even finish running. This is the Halting Problem. A few years later, Rice's theorem took this further — Henry Gordon Rice proved that it is mathematically impossible to build a tool that can verify any meaningful property of software in the general case. Not hard. Not expensive. Impossible.

> This means "make sure it doesn't make a mistake" software was never a guarantee anyone could offer. Every piece of software you trust today shipped with that same uncertainty.

So when someone says "I can't use LLMs in production because they're nondeterministic — we need to build deterministic workflows where making no mistakes is the baseline expectation," they're confusing repeatability with correctness. A deterministic program that returns the wrong answer returns it every single time. That's a bug, not a baseline expectation.

Manufacturing figured this out decades ago. Six Sigma doesn't demand zero defects — it defines an acceptable defect rate and builds measurement systems to stay within that bound. The discipline was never "eliminate all variation." It was "define, measure, analyze, improve, control" — continuously reducing variation. That's evaluation, not determinism.

What AI systems shift is the evaluation surface. Instead of "does this code path execute as specified," you ask "does this output meet our evaluation criteria." The work moves from pre-deployment code verification to continuous evaluation.

> In AI and machine learning systems, we reduces entropy (chaos) through evaluation.

This is not new. TCP connects on unreliable networks. RAID clusters operate on top of failing drives. AI models are trained on failing GPUs. We've always built reliable systems from unreliable components.

> It was never about determinism. It was always about evaluations. If this resonates, I go deep on evaluation design in my book.

Anthropic's Project Glasswing - restricting Claude Mythos to security researchers - sounds necessary to me

SIMON WILLISON'S WEBLOG: ENTRIES · 07 APR 2026 · SOURCE

Anthropic’s Project Glasswing—restricting Claude Mythos to security researchers—sounds necessary to me

Anthropic *didn’t* release their latest model, Claude Mythos (system card PDF), today. They have instead made it available to a very restricted set of preview partners under their newly announced Project Glasswing.

The model is a general purpose model, similar to Claude Opus 4.6, but Anthropic claim that its cyber-security research abilities are strong enough that they need to give the software industry as a whole time to prepare.

> Mythos Preview has already found thousands of high-severity vulnerabilities, including some in *every major operating system and web browser*. Given the rate of AI progress, it will not be long before such capabilities proliferate, potentially beyond actors who are committed to deploying them safely.
>
> [...]
>
> Project Glasswing partners will receive access to Claude Mythos Preview to find and fix vulnerabilities or weaknesses in their foundational systems—systems that represent a very large portion of the world’s shared cyberattack surface. We anticipate this work will focus on tasks like local vulnerability detection, black box testing of binaries, securing endpoints, and penetration testing of systems.

Saying “our model is too dangerous to release” is a great way to build buzz around a new model, but in this case I expect their caution is warranted.

Just a few days (last Friday) ago I started a new ai-security-research tag on this blog to acknowledge an uptick in credible security professionals pulling the alarm on how good modern LLMs have got at vulnerability research.

Greg Kroah-Hartman of the Linux kernel:

> Months ago, we were getting what we called 'AI slop,' AI-generated security reports that were obviously wrong or low quality. It was kind of funny. It didn't really worry us.
>
> Something happened a month ago, and the world switched. Now we have real reports. All open source projects have real reports that are made with AI, but they're good, and they're real.

Daniel Stenberg of `curl`:

> The challenge with AI in open source security has transitioned from an AI slop tsunami into more of a ... plain security report tsunami. Less slop but lots of reports. Many of them really good.
>
> I'm spending hours per day on this now. It's intense.

And Thomas Ptacek published Vulnerability Research Is Cooked, a post inspired by his podcast conversation with Anthropic's Nicholas Carlini.

Anthropic have a 5 minute talking heads video describing the Glasswing project. Nicholas Carlini appears as one of those talking heads, where he said (highlights mine):

> It has the ability to chain together vulnerabilities. So what this means is you find two vulnerabilities, either of which doesn't really get you very much independently. But this model is able to create exploits out of three, four, or sometimes five vulnerabilities that in sequence give you some kind of very sophisticated end outcome. [...]
>
> **I've found more bugs in the last couple of weeks than I found in the rest of my life combined**. We've used the model to scan a bunch of open source code, and the thing that we went for first was operating systems, because this is the code that underlies the entire internet infrastructure. **For OpenBSD, we found a bug that's been present for 27 years, where I can send a couple of pieces of data to any OpenBSD server and crash it**. On Linux, we found a number of vulnerabilities where as a user with no permissions, I can elevate myself to the administrator by just running some binary on my machine. For each of these bugs, we told the maintainers who actually run the software about them, and they went and fixed them and have deployed the patches patches so that anyone who runs the software is no longer vulnerable to these attacks.

I found this on the OpenBSD 7.8 errata page:

> **025: RELIABILITY FIX: March 25, 2026** *All architectures*
>
> TCP packets with invalid SACK options could crash the kernel.
>
> A source code patch exists which remedies this problem.

I tracked that change down in the GitHub mirror of the OpenBSD CVS repo (apparently they still use CVS!) and found it using git blame:

Sure enough, the surrounding code is from 27 years ago.

I'm not sure which Linux vulnerability Nicholas was describing, but it may have been this NFS one recently covered by Michael Lynch .

There's enough smoke here that I believe there's a fire. It's not surprising to find vulnerabilities in decades-old software, especially given that they're mostly written in C, but what's new is that coding agents run by the latest frontier LLMs are proving tirelessly capable at digging up these issues.

I actually thought to myself on Friday that this sounded like an industry-wide reckoning in the making, and that it might warrant a huge investment of time and money to get ahead of the inevitable barrage of vulnerabilities. Project Glasswing incorporates "$100M in usage credits ... as well as $4M in direct donations to open-source security organizations". Partners include AWS, Apple, Microsoft, Google, and the Linux Foundation. It would be great to see OpenAI involved as well—GPT-5.4 already has a strong reputation for finding security vulnerabilities and they have stronger models on the near horizon.

The bad news for those of us who are *not* trusted partners is this:

> We do not plan to make Claude Mythos Preview generally available, but our eventual goal is to enable our users to safely deploy Mythos-class models at scale—for cybersecurity purposes, but also for the myriad other benefits that such highly capable models will bring. To do so, we need to make progress in developing cybersecurity (and other) safeguards that detect and block the model's most dangerous outputs. We plan to

> launch new safeguards with an upcoming Claude Opus model, allowing us to improve and refine them with a model that does not pose the same level of risk as Mythos Preview.

I can live with that. I think the security risks really are credible here, and having extra time for trusted teams to get ahead of them is a reasonable trade-off.

Issue #705

POINTER · 07 APR 2026 · SOURCE

April 7th, 2026 | Read online

Pointer

Tuesday 7th April issue is presented by UpLevel

Are You Getting Real ROI From AI — Or Just Faster Code?

84% of developers use AI tools. Fewer than 3% of organizations have meaningfully changed how they ship — or can show what it's worth to the business.

StackUp is a free 10-minute diagnostic that benchmarks your engineering org against peers across ways of working, alignment, velocity, and environment — and tells you which 2–3 changes will have the biggest impact on your AI ROI.

Take The Free Diagnostic

Why Your Engineering Team Is Slow

— Alex Piechowski

tl;dr : “After years of codebase audits, the same five signals keep showing up, so I finally put them into a scoring rubric: The Codebase Drag Audit. Five signals, scored 0 to 2. If you hit 4 or above, the code needs direct investment before anything else will help.”

Leadership Management

On The Socially Acceptable Use Of AI In Business

— Dave Kellogg

tl;dr : “There’s a question I’ve been mulling for a while now, and I think it’s time to write it down: when is it okay to use generative AI in a given business context, and when does it cross a line? I’ll focus on two specific areas I know well — board work and strategic analysis — but I think the principles generalize.”

Leadership Management

Are You Getting Real ROI From AI — or Just Faster Code?

tl;dr : 84% of developers use AI tools. Fewer than 3% of organizations have meaningfully changed how they ship — or can show what it's worth to the business. StackUp is a free 10-minute diagnostic that benchmarks your engineering org against peers across ways of working, alignment, velocity, and environment — and tells you which 2–3 changes will have the biggest impact on your AI ROI.

Promoted by Uplevel

Leadership Management

Encoding Team Standards

— Rahul Garg

tl;dr : ““AI coding assistants respond to whoever is prompting, and the quality of what they produce depends on how well the prompter articulates team standards. I propose treating the instructions that govern AI interactions as infrastructure: versioned, reviewed, and shared artifacts that encode tacit team knowledge into executable instructions, making quality consistent regardless of who is at the keyboard.”

Leadership Management

“Too many of us are not living our dreams because we are living our fears.”

— Les Brown

The Unwritten Laws Of Software Engineering

— Anton Zaides

tl;dr : “As everybody and their mother thinks they can build great software right now, I decided to help them avoid a bit of pain. Here are 7 laws every engineer has broken at least once, learned the hard way.”

BestPractices

How To Scale Code Review When AI Writes Code Faster Than You Can Understand It

tl;dr : AI is outpacing traditional code review, creating a verification bottleneck. This report breaks down the shift: (1) a growing trust gap: 96% of developers distrust AI output, (2) the move to automated guardrails, and (3) embedding verification directly into the SDLC with a “trusted, but verified” approach.

Promoted by CodeReview

CodeReview

Eight Years Of Wanting, Three Months Of Building With AI

— Lalit Maganti

tl;dr : “There’s no shortage of posts claiming that AI one-shot their project or pushing back and declaring that AI is all slop. I’m going to take a very different approach and, instead, systematically break down my experience building syntaqlite with AI, both where it helped and where it was detrimental.”

Tools Productivity

What Is Inference Engineering?

— Gergely Orosz

tl;dr : “Many engineers use inference daily, but inference engineering is a bit obscure – and an area rich with interesting challenges. Philip Kiely, author of the new book, “Inference Engineering,” explains.”

DeepDive

7 More Common Mistakes In Architecture Diagrams

— William Pliger

tl;dr : “System architecture diagrams are essential tools for documenting complex systems. However, common mistakes in these diagrams can lead to confusion, misinterpretation, and frustration for viewers. Here’s a rundown of seven (more!) common mistakes to avoid.”

BestPractices Architecture

Editorial Note

A friend passed this article on and it stuck with me.

It likens the economics of AI to the subprime crash, arguing that AI labs (OpenAI, Anthropic) obfuscate finances with zero path to profitability and an irreversibly broken business model.

The issue is that it the impending doom of these AI labs becomes our own instability, given our staggering adoption, growing over-reliance and rapid workflow integration.

AI feels like a magical tool to me but, perhaps, part of that magical feeling is that it's on always on tap? For engineering managers, this raises uncomfortable questions: if code generation isn't truly free, how should we think about cost, ownership, and quality over time?

To be clear, I'm not convinced this is inevitable - but the argument is worth sitting with.

PS. I'm experimenting with this section to pen personal thoughts. Feel free to hit reply and share any feedback.

Most Popular From Last Issue

What I Learned From Nearly 1,000 Interviews At Amazon -Steve Huynh

Notable Links

Claude How-To : Visual, example-driven guide.

EmDash : TypeScript CMS based on Astro.

Gallery : On-device GenAI use cases.

Goose : OS AI agent that automates engineering tasks.

QMD : Search engine for everything you need to remember.

How did you like this issue of Pointer?

1 = Didn't enjoy it all // 5 = Really enjoyed it

1 | 2 | 3 | 4 | 5

Measuring AI Ability to Complete Long Software Tasks

NOREPLY@BLOGGER.COM (MURAT) · 28 MAR 2026 · SOURCE

This paper from METR (Model Evaluation & Threat Research) introduces a new metric for tracking AI progress: the "50%-task-completion time horizon". This denotes the length of software engineering task (measured by how long a skilled human developer takes to complete it) that the AI model can finish with 50% success rate. The researchers evaluated 12 frontier AI models on 170 tasks across three benchmarks: HCAST (97 diverse software tasks ranging from 1 minute to 30 hours), RE-Bench (7 difficult ML research engineering tasks, each 8 hours), and SWAA (66 short software actions taking 1-30 seconds). To calibrate task difficulty, they collected over 800 human baselines from professional developers, totaling 2,529 hours of work.

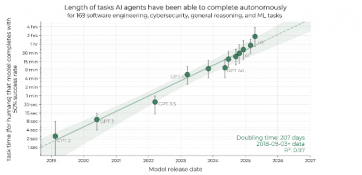


Figure 1: The length of tasks (measured by how long they take human professionals) that generalist autonomous frontier model agents can complete with 50% reliability has been doubling approximately every 7 months for the last 6 years (Section 3). The shaded region represents 95% CI calculated by hierarchical bootstrap over task families, tasks, and task attempts. Even if the absolute measurements are off by a factor of 10, the trend predicts that in under a decade we will see AI agents that can independently complete a large fraction of software tasks that currently take humans days or weeks (Section 5).

The headline finding is that this time horizon has been doubling every 7 months since 2019. GPT-2 could handle tasks that take humans about 2 seconds. The o3 model reached 110 minutes. Extrapolating this trend, AI reaches a one-month time horizon (167 working hours) between mid-2028 and mid-2031, with a central estimate of mid-2029. To be precise about what this means: the "167 working hours" refers to task complexity as measured by human effort, not AI execution time. An AI agent with a one-month time horizon could complete a task that takes a human developer a month of work within hours; it works continuously, doesn't context-switch,

and executes steps faster. What the one-month time horizon means is that, AI would be able to autonomously build a SaaS MVP, migrate a 200k-line legacy codebase to microservices, or implement complex features end-to-end.

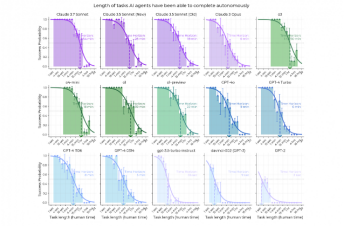
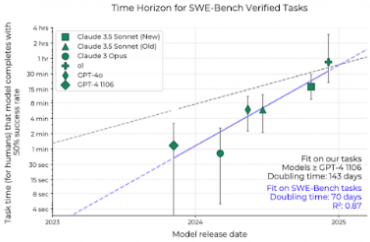


Figure 4: Success rates of all models on the test suite, showing the computation of time horizon as predicted 50%- success rate time. The logistic fit is fairly good, though there is a jump in success rate between <1 minute SWAA tasks and >1 minute HCAST tasks.

That said, "50% success rate" is doing the heavy lifting in this claim. There is a significant reliability gap: the 80% success-rate time horizon is 4-6x shorter than the 50% horizon. A model that can sometimes complete a month-long task can only reliably complete week-long ones. Models also struggle in messy environments that lack clear feedback loops (like automated tests) and require proactive information gathering. This means well-structured well-tested codebases will benefit far more than legacy systems with poor documentation. That is, greenfield work will see more gains than deep maintenance of complex existing systems.

Another caveat is that, experiments with pull requests showed that AI performance aligns more closely with low-context contractors (who take 5-18x longer than expert maintainers) than with experienced developers who know the codebase. This matters because the extrapolation is calibrated against general-purpose developers, not domain experts. When the paper says "one-month time horizon", it means one month of contractor-level work, not one month of work by someone who built the system. For tasks that require deep institutional knowledge (debugging subtle production issues, evolving tightly coupled legacy systems) the effective time horizon is much shorter.



I think the external validity checks were a strong part of the paper. The paper shows that the trend holds on SWE-bench (with an even faster 70-day doubling time). Again, messy tasks show lower absolute success but the same rate of improvement. The methodology is clean and reproducible. The main weakness is the task distribution. These 170 tasks are mostly software tasks with high inter-model task correlation (r=0.73), and they largely represent isolated coding challenges rather than the full breadth of real-world software engineering. True full-stack ownership (gathering specifications, architectural decision-making, deployment, and live operations) remains untested here. Furthermore, while AI excels at isolated prototyping, engineering end-to-end production-grade distributed systems requires a high level of fault tolerance/reliability that remains incredibly challenging to automate.

Discussion: What does a one-month AI time horizon world look like?

If the extrapolation holds and AI systems can automate month-long software tasks by ~2029, what are the implications?

This changes the unit economics of software fundamentally. Tasks that currently require a team of developers working for weeks (building integrations, standing up new services, migrating databases, writing and testing features) become something you can delegate to an AI agent for a fraction of the cost and time. The AI agent operates at the level of a competent junior-to-mid engineer working on a well-scoped project. Software goes from expensive and slow to cheap and fast.

But something tells me we will find ways to squander this. When hardware kept getting faster through Moore's law, we produced bloated software that felt just as slow. Wirth's law ate Moore's law. When cloud computing made infrastructure cheap, we built sprawling microservice architectures that ended up like a ball of yarn, tangled, and sprawling, and hard to unravel. Every time we reduce one cost, we expand scope until the new bottleneck hurts just as much as the old one. I expect AI-generated software to follow the same pattern: more software would mean more complexity, and new categories of problems we don't yet have names for. 50 shades of metastability?

What does the month-long task horizon mean for big tech, enterprise software, and startups? How the hell should I know? But let me speculate anyway, since you are still reading (or at least your LLMs are).

Big tech companies employ tens of thousands of engineers specifically for the month-long projects this extrapolation targets: internal tools, service migrations, API implementations, data pipelines. AI will accelerate all of this. But these companies are structured around large engineering orgs. Promotion ladders, team hierarchies, planning processes all assume that software requires massive human headcount. If one engineer with AI agents does the work of five, the traditional organizational model breaks. Middle management's role become muddled. We may see flattening hierarchies and small high-leverage teams. Whether the behemoth organizations can handle such a rapid restructuring is unclear.

For infrastructure and database companies, the Jevons Paradox applies. Cheaper software means more of it: more applications, more databases, more demand for managed services. The total addressable market could grow substantially. But the flip side is that AI agents that can build month-long projects can also evaluate and switch infrastructure. Vendor lock-in weakens when migration is cheap, and customers become more price-sensitive. There's also the question of whether AI agents will develop strong default preferences shaped by their training data or their creator companies. This is a brand new market force, which we didn't have to contend with before.

I think the most dramatic impact is on software startups. Today, the primary cost of a startup is engineering talent. If AI handles month-long implementation tasks, that cost drops by an order of magnitude. The barrier to entry collapses. Many more products get built. That may mean that differentiation shifts to domain expertise, data moats, and taste. I forecasted a similar trend earlier for academic papers.

The reliability gap tells us the world of 2029-2031 is probably not "AI replaces developers" but "developers who use AI effectively are 5-10x more productive". The scarce resource shifts from the ability to write code to the ability to specify what to build, evaluate whether it is correct, and manage the complexity of systems that grow much faster than before. What worries me more is whether organizations can restructure fast enough to keep up.

Agentic Search as an Agile Engineering Process

DANIEL TUNKELANG · 28 MAR 2026 · SOURCE

(co-authored with Asif Makhani, co-founder of Infino AI)

Search is usually framed as retrieval: given a query, return the most relevant documents. Today, that framing is breaking down.

Agentic search systems do more than retrieve: they plan, decompose, execute, verify, and refine. They behave less like function calls and more like software engineering teams pursuing goals under uncertainty.

Once you see this, a natural question follows: If agentic search is a form of engineering, what methodology should guide it?

Agile software engineering offers a compelling answer—not just as an analogy, but as a control system.

Search as Engineering without a Specification

A traditional software project starts with a specification. Search does not.

A query is an incomplete, often ambiguous description of a goal. The system must infer intent, explore interpretations, and iteratively construct an answer.

From this perspective:

- The query is a partial specification.
- Retrieval and reasoning comprise implementation.
- Results represent candidate solutions.
- Interaction becomes part of the evaluation loop.

Search is what engineering looks like when the specification is missing.

Agentic Search as Managing a Workflow

Traditional search executes in a single pass. Agentic systems do not.

Agentic search:

- Breaks problems into subtasks.
- Delegates subtasks to tools.

- Explores multiple paths.
- Refines based on intermediate results.

Agentic search no longer executes queries. It manages workflows. This is exactly the kind of process agile methods are designed for.

Iteration as Uncertainty Reduction

Agile development is fundamentally about reducing uncertainty through iteration. The same is true for agentic search.

Each step should be evaluated not just by what it produces, but by how much uncertainty it removes per unit of cost.

This reframes the agentic search process as a sequence of uncertainty-reducing experiments.

The Tradeoff: Scope, Cost, Quality

Perhaps the most important foundation of software engineering is the classic scope-time-quality triangle: you want to optimize scope, time, and quality, but you cannot achieve all three. At most, you can pick two.

In agentic search, we can translate engineering time into cost, so the tradeoff becomes:

- Scope: how much of the problem space the search covers.
- Cost: aggregate spend on tokens, tool calls, and other computation.
- Quality: a combination of correctness, completeness, and confidence.

As in software engineering, it is always possible to mitigate uncertainty by reducing scope. However, the critical constraint is likely to be cost, since projects have finite budgets. These three levers define the tradeoff in every agentic search workflow.

The Triangle in Practice

Here are three common strategies for managing this tradeoff in practice:

- Treat cost and quality as fixed, and reduce scope. This strategy takes a narrow focus, only explores high-confidence paths, and stops early. This tends to be the default strategy for agentic search.

- Treat scope and quality as fixed, and increase cost. This strategy emphasizes broader exploration, aggressive verification, and conflict resolution. This is what many systems expose as “deep research”.

- Treat scope and cost as fixed, and sacrifice quality. This strategy leads to skimming content, summarization, and accepting lower confidence. This is often exposed as a “quick overview” mode.

Each strategy makes a different tradeoff and can produce dramatically different results. Also, unlike in most software engineering contexts, the tradeoffs can be adjusted dynamically throughout the process.

Searchers as Product Owners, Agents as Engineers

In agile software development, there are two key roles. The product owner defines and prioritizes goals. Engineers own all aspects of execution.

In agentic search, the searcher is the product owner, while agents fill the roles of engineers.

However, much as can happen with software development, searchers may not know what they want until they see it. The process is iterative: requirements evolve as results come in.

But unlike software development, where iterations are measured in days, agentic search iterations are measured in minutes or even seconds. This makes the loop tighter, faster, and more volatile.

Task Sizing: A Critical Challenge

A central challenge in both agile and agentic systems is decomposition. Larger tasks increase efficiency by reducing coordination overhead, but they also increase risk. Smaller tasks reduce risk but decrease efficiency.

The optimal tradeoff balances the cost of coordination against the expected cost of errors. This tradeoff determines:

- How many steps an agent takes.
- How much reasoning an agent packs into each step.
- How often the agent verifies that it is on the right path.

This is one of the most important—and underexplored—design decisions in agentic search workflows.

The Cost of Iteration: Unpredictability

One of the classic frustrations with agile software development—especially for product managers!—is its lack of predictability. It is impossible to know exactly when a project will finish.

Agentic search inherits—and amplifies—this property.

The searcher does not know:

- The number of steps required.
- The number of branches to be explored.
- The sources of uncertainty.

However, as with software engineering, trying to force predictability leads to worse outcomes:

- Premature stopping.
- Shallow exploration.
- Lower-quality results.

So it is important to remember that predictability is not the goal for which agentic search can or should optimize.

Replacing Predictability with Evaluability

Agile development replaces predictable completion with predictable progress. Agentic search takes this idea one step further: it does not define “done” upfront; instead, it detects completion using evaluation.

This framing informs the entire iterative workflow. The question is never “How many steps remain?”; rather, it is “Is further work worth the cost?”

An agentic system is “done” when spending more is not expected to improve the quality of the outcome. In other words, does the expected marginal gain justify the marginal cost?

Testing is the Definition of Done

In software engineering, “done” is defined by passing tests. In agentic search, evaluation replaces this “definition of done”.

What needs to be evaluated?

- Outcome: correctness, completeness, and usefulness.
- Process: validation of exploration and verification strategies.
- Efficiency: maximization of ROI, minimization of wasted effort.

While some of this evaluation can be automated, the searcher plays a critical role in evaluating outcomes.

Testing Replaces Explainability

People like explainability. Unfortunately, we have learned from the evolution of search that explainability is incompatible with the modern neural AI-powered search methods that achieve the best performance.

For the most part, we have learned to accept reduced explainability in exchange for better outcomes, and that also holds true for agentic search.

However, what we need is not explainability, but the ability to test the process. We need to be able to evaluate outcomes and audit processes.

Putting it all Together

Agentic search brings together three interacting layers:

- An agile, iterative workflow to reduce uncertainty.
- The constraints and tradeoffs of the scope–cost–quality triangle.
- An evaluation process that establishes a definition of done.

Together, these form a control system for reasoning under uncertainty.

This framing suggests concrete design principles:

- Prioritize steps that maximize uncertainty reduction per unit of spend.
- Dynamically adjust scope based on budget and confidence.
- Size tasks to balance coordination costs against expected costs of errors.
- Integrate verification into the iteration and refinement processes.
- Define and enforce evaluation-driven stopping criteria.

Final Thought

The agile development methodology recognizes and accepts that we cannot predict exactly when the work will be complete.

Agentic search goes further: it is not just a question of whether the process is complete, but rather whether the results are good enough and further effort does not justify the cost.

This shift—from execution to evaluation—is what turns search into an engineering problem and motivates an agile approach.