



Can AI Smell (Yet)?

**Anticipatory Judgment, Gap Closure,
and the Lovelace Objection**

Working Paper October 2026

Vidhu Shekhar

S. P. Jain Institute of Management & Research

vidhu.shekhar@spjimr.org

Can AI Smell (Yet)?

Anticipatory Judgment, Gap Closure, and the Lovelace Objection

Working Paper October 2026

Vidhu Shekhar

S. P. Jain Institute of Management & Research

`vidhu.shekhar@spjimr.org`

Abstract

Since Lovelace, people have asked whether a machine can originate anything beyond what it is given. AI now produces historically novel solutions in coding, mathematics, and science, but much of this can be understood as *gap closure*: solving a problem whose objective, criterion, and domain someone else supplied. Gap closure leaves a different capacity untested. Experienced programmers can sense that an architecture will become brittle before they can name the requirement that will expose it. I call this capacity *anticipatory smell* and define it behaviorally: an agent exhibits it to the extent that, unprompted, its design choices later prove cheaper to adapt to changes absent from the task context, net of the cost of the flexibility they add. A framework of four modes—specification execution, gap closure, anticipatory smell, and self-originated inquiry—separates capacities that require different evidence and relocates the Lovelace objection from novelty of output to the source of direction. An informational result shows that systematically lower adaptation cost under unrepresented change requires information available at the time of choice that predicts differences in that cost; enumerating more futures does not supply it. I hypothesize that one source of such judgment is a learning signal linking design decisions to their later consequences, and propose a test comparing models trained on the same project histories with decision–consequence relations paired, left in their ordinary dispersed form, or shuffled.

Keywords: Artificial intelligence; expertise; tacit knowledge; unawareness; creativity; Lovelace objection; software engineering; AI evaluation

1 Introduction

Ada Lovelace, in her notes on the Analytical Engine, wrote that the Engine had no pretensions to originate anything and could do only what it was ordered to perform (Lovelace, 1843). Turing replied that machines surprise their makers and that a machine which learns need not remain within what it was told (Turing, 1950). AI systems now prove mathematical results, write and revise software, generate scientific hypotheses, and carry out long sequences of actions toward supplied objectives. This is often read as settling the question in Turing’s favor. I argue that it does not. Solving a specified problem and judging well under an unspecified future are different capacities. A problem can be difficult while remaining externally posed, and solving it does not show whether an agent can judge well about matters that have not yet been posed as problems.

Software engineering gives a concrete and demanding case. Coding is already among AI’s strongest applications, yet some of the most consequential engineering judgments concern what happens after the current task is complete. An experienced programmer may inspect an architecture that satisfies its requirements, passes its tests, and shows no defect, yet judge that it “smells wrong.” The concern may attach to coupling, a rigid abstraction, or a choice that seems likely to constrain later development. She may be unable to name the requirement that will expose the weakness or reconstruct the observations behind her judgment. Even so, the judgment changes what she builds. A present choice has been shaped by sensitivity to a future that was never represented in the specification.

I call this capacity *anticipatory smell*. Accumulated experience shapes present judgment in response to future concerns that the task does not supply and that may not yet be represented at all. This differs from “code smell,” which concerns identifiable features of existing code that indicate deeper design problems (Fowler, 1999). It also differs from explicit foresight, where possible futures are made part of the decision problem before the choice is evaluated against them.

Much of what AI now achieves is better described as *gap closure*. A system derives a previously unknown solution while the problem, its criterion, and its domain are already given. Such achievements can be historically novel. A theorem may be proved for the first time or a faster algorithm discovered without the system having originated the problem or the direction of inquiry. Difficulty and the source of direction are separate properties of a task.

The argument draws on work on tacit knowledge, expert intuition, bounded rationality, and decision-making under radical uncertainty and unawareness. It asks a functional question about what agents do across repeated decisions. Is expert judgment of this kind recoverable

from artifacts in their ordinary form, or does it depend on experience that links earlier choices to their later consequences? Nothing in the argument assumes an intrinsic boundary between human and artificial cognition.

The paper makes three contributions. First, I define anticipatory smell by behavior and distinguish it from diagnostic code smell, explicit foresight, and codified design heuristics. I then use an informational condition to locate the empirical question. Advantage under unrepresented change, beyond what minimizing carrying cost can secure, requires information available at the time of choice that predicts differences in adaptation cost across designs. Second, I distinguish four modes of cognitive activity—specification execution, gap closure, anticipatory smell, and self-originated inquiry—and use them to reconsider the Lovelace objection in terms of the source of direction. Third, I propose a hypothesis about how smell is learned and a behavioral test that separates it from the view that ordinary software artifacts already provide a sufficient learning signal.

2 Novelty, Gap Closure, and the Source of Direction

Two properties of a cognitive task should be kept apart. One is the inferential gap between a specification and an acceptable solution. The other is the source of the problem itself. Most benchmarks measure progress on the first. Novel output tells us little about the second because an output can be new without requiring the agent to originate the problem or change the conceptual space in which it works. A novel image of a unicorn, for example, can be produced by combining concepts already available.

Boden (2004) gives this distinction a broader form. She separates psychological creativity, which is new to an individual, from historical creativity, which is new relative to prior human production. She also distinguishes combinational creativity from exploratory creativity within a structured conceptual space and transformational creativity that changes the space itself. Historical novelty and conceptual transformation therefore need not coincide. The first discovery of an admissible solution within an existing space can be historically new while remaining exploratory in structure.

I use *gap closure* for cases in which an agent derives a previously unknown solution while the focal problem, its evaluative criterion, and its domain are already available. The term concerns the structure of the task, not the method used to solve it. An agent may search, reason, simulate, recombine, or introduce new representations. A proof of an open conjecture may be unprecedented and may depend on an abstraction no previous mathematician considered, while still answering a problem supplied from outside.

OpenAI’s proposed solution to the Navier–Stokes existence and smoothness problem

(OpenAI, 2026) is an extreme case. A multi-agent system produced a proof, with a Lean formalization, that an initially smooth fluid at rest, driven by a smooth external force and retaining finite energy, can develop a singularity in finite time. OpenAI argues that this establishes the breakdown alternatives in the official formulation. The result awaits evaluation by the mathematical community. The Clay Mathematics Institute has said that the problem has apparently been settled and that its process for evaluating the work and assigning credit is deliberately unhurried (Clay Mathematics Institute, 2026).

The achievement, if it stands, is historically unprecedented. Yet the target was externally chosen. The Institute posed the problem in 2000 and specified the admissible forms of an answer. OpenAI reports that its researchers set the system on the open Millennium problems and later directed resources toward Navier–Stokes after agents resolved a related question for the Euler equations (OpenAI, 2026).¹ The achievement can therefore be both historically novel and a case of gap closure. AlphaEvolve (Novikov et al., 2025) has the same structure. It has improved best-known results on open mathematical and algorithmic problems, but it operates with an evaluation function supplied by the user. The criterion of success is given to the system.

Two clarifications mark the boundary of gap closure. The agent need not keep a fixed representation of the solution space. It may invent an abstraction, decomposition, or search strategy while solving the supplied problem. Changing how a problem is represented is different from originating the problem itself. Nor does gap closure depend on retrieval from training data. Generalization can produce outcomes the agent has never observed. The category tracks the source of direction, not whether the solution is novel or how it was found.

Most evaluations of AI begin after someone has selected the focal problem. The system receives a problem and its output is judged against a criterion. Benchmarks vary in how much they specify. Some give an exact task and scoring rule; others give a broad objective and leave the agent to decompose the task, choose tools, and revise its plans. In both cases, the direction of activity originates outside the system.

The language of autonomy can obscure this distinction. An agent may set intermediate goals, critique its own work, change strategy, and decide which tools to use. These are genuine forms of autonomy in execution, but they occur within a supplied objective. That differs from originating the direction of inquiry, where the focal problem itself is not selected for the agent.

Self-originated direction does not mean creation ex nihilo. Human inquiry also arises from existing beliefs, incentives, social settings, biological drives, and available concepts. The narrower question is who selected the problem as the focal task. A researcher who notices

¹OpenAI recognizes the priority of Alpböge and Buckmaster on a forced Euler result (OpenAI, 2026).

an anomaly while working on something else and decides that it warrants investigation has redirected the inquiry, even though both the anomaly and her reasons for caring about it have prior causes.

Schön (1983) calls this prior act *problem setting*, the process of deciding what in an indeterminate situation counts as the problem, what ends matter, and what is relevant. Rittel and Webber (1973) make a related point about planning problems, whose formulation cannot be separated cleanly from their solution. Simon (1973) argued that ill-structured problems need no special faculty because they acquire structure through the same processes used to solve well-structured problems. The criterion here is compatible with Simon’s argument. What matters is who selected the focal problem, not whether structuring it requires a distinct cognitive mechanism.

The Lovelace objection has also been turned into a test. Bringsjord et al. (2001) proposed that an artificial agent passes a Lovelace Test if it produces an output whose production its designers cannot explain, setting the condition strictly enough that few if any systems pass (see Riedl, 2014). That test concerns the designers’ knowledge of how an output was produced. It does not establish where the direction of inquiry came from. Designers may understand how a system searches while being unable to predict the result it will find. They may still have supplied the problem, the criterion, and the decision to pursue it.

I therefore read the surviving force of Lovelace’s objection as a question about direction. Turing was right that machines can produce novelty and surprise. Gap closure now demonstrates both. But novelty of output does not tell us who established the inquiry that produced it. Nor should originality depend on whether an outcome was derivable from the agent’s information. That would deny originality to a mathematician who proves a new theorem from established axioms. The source of direction gives us a criterion that can be applied to human and artificial agents alike.

The claims here are functional. Anticipatory smell and self-originated inquiry are defined by what an agent does across repeated occasions, not by what accompanies the behavior internally. This leaves open whether functional equivalence captures everything a human expert experiences as intuition. Nothing in the argument requires that it does. The advantage of the functional approach is that the same evidence can be required of human and artificial agents. Rolla (2026) argues that behavioral competence cannot establish whether an artificial system is a cognizer because cognition requires embodied, self-maintaining life. Floridi (2023) describes such systems as agents without intelligence. I take no position on either claim. The narrower question is what behavior would count as evidence for the capacities distinguished here.

Anticipatory smell lies between supplied execution and self-originated inquiry. The task

remains externally given, but considerations absent from the specification affect how the agent performs it. A programmer asked to implement a feature may choose a more modular architecture because the supplied design seems brittle under uses she cannot yet identify. She has not originated the project or its objective. What enters endogenously is a consideration that changes how she meets it.

3 The Motivating Case: Anticipatory Smell in Software Engineering

Software engineering makes the distinction between present performance and future adaptability visible. Functional correctness is judged against requirements that already exist. Architectural quality often becomes apparent only as a system encounters requirements, interfaces, loads, users, and organizational constraints that were absent when the design was chosen. Lehman (1980) observed that software in use must continually adapt or become less satisfactory, and that its structure degrades under adaptation unless effort is made to preserve it. Two implementations can satisfy the same specification at time t yet impose very different adaptation costs at $t + k$. Technical debt captures part of this temporal structure (Cunningham, 1992; Kruchten et al., 2012). Anticipatory smell concerns later costs that were not yet articulated when the design was chosen.

The term “code smell,” introduced by Kent Beck in the chapter he co-wrote for Fowler (1999), refers to observable features of existing code that suggest deeper design problems. Examples include duplicated code, long methods, large classes, and feature envy. These smells are diagnostic. A visible feature of the present artifact points to a structural problem that may warrant refactoring. The cue can be named, catalogued, and sometimes detected automatically. Anticipatory smell is different. Its object is a present design choice, and its concern is how that choice will fare under conditions not yet specified. The expert may respond before there is a catalogued symptom to diagnose or a particular future to consider.

3.1 Anticipatory Smell

I define the construct by behavior.

Definition 1. An agent exhibits *anticipatory smell* to the extent that, across repeated decisions, (i) considerations absent from the current task context affect its choices; (ii) applicable codified rules leave structurally different admissible alternatives open; and (iii) among those alternatives, its choices are associated with lower adaptation costs under later change than

the choices that would otherwise have been made, net of the carrying cost of the flexibility they add.

Condition (i) separates smell from execution of a supplied criterion. It concerns the current task context, not the agent’s history. Dispositions acquired through training or practice, like an engineer’s internalized professional norms, are not part of the specification. Condition (iii) gives the construct its predictive content. Anyone can feel vague unease, and such unease can reflect error, bias, or superstition. One lucky choice is not enough. Smell requires a recurring relationship between judgment and later adaptability, even when the agent cannot state which future condition it is responding to. Defining smell this way also leaves its mechanism open. A system that showed the behavior would possess the capacity regardless of how it acquired it.

Table 1 separates anticipatory smell from nearby concepts. Diagnostic smell operates on an articulated symptom, technical debt on an attributable later cost, and scenario analysis on represented futures. Anticipatory smell concerns a future consideration that was not supplied in the task. The agent may still generate possible futures for itself; the stronger case arises when its choice also proves robust to changes that remained unrepresented when the choice was made.

	Diagnostic code smell	Technical debt	Scenario analysis	Codified heuristic	Anticipatory smell
Object of judgment	Present artifact	Design choice with later cost	Enumerated futures	Present artifact against a rule	Present choice under an open future
Future concern supplied?	Not required	If deliberate	Yes	Encoded in rule	No
Cue articulable?	Usually	Once attributed	Yes	Yes	Not required
Trigger	Inspection	Trade-off or attribution	Prompted	Rule match	Unprompted
Validated by	Refactoring need	Realized cost	Scenario coverage	Rule conformance	Adaptation cost, net of carrying cost

Table 1: Anticipatory smell and its neighboring constructs.

The closest neighbor, and the one on which condition (ii) depends, is the *codified design heuristic*. Parnas’s criterion for modular decomposition is the canonical example. A module should hide a design decision that is likely to change so that later change remains confined to

one place (Parnas, 1972). Baldwin and Clark (2000) and Sullivan et al. (2001) later gave this principle an economic interpretation, treating modularity as a portfolio of options whose value lies in the ability to substitute one module without redesigning the rest. Such heuristics are the articulated residue of earlier engineering judgment, and a system can learn and apply them from text. If a rule settles the choice, applying it is not evidence of anticipatory smell, even when the application is unprompted.

Parnas's criterion also shows where judgment remains. The rule says to hide decisions that are likely to change, but it cannot identify which decisions in a particular system are fragile. Experience must do that work. Smell therefore matters when the applicability or weighting of a heuristic is itself part of the judgment, where rules conflict, apply imperfectly, or leave several structurally different choices open. Condition (ii) puts the boundary there instead of asking the unobservable question of whether a learned rule caused the choice.

Anticipatory smell should arise only where the environment contains learnable regularities. Proposition 1 formalizes that limit. Feedback on earlier decisions is one possible way of learning those regularities, and its role is treated below as a hypothesis rather than a requirement. Smell should not be expected in a domain new to the agent, in an unstable environment, or where the available information contains no signal about later adaptability. An expert may still feel uneasy in such settings, but the account does not predict that the feeling will track later outcomes. No explicit representation of the later state is required at the time of judgment.

Consider a programmer choosing between two architectures for a new service. Both satisfy the specification and perform similarly under current load. One spreads assumptions about data structure and authentication across the application; the other localizes them behind narrower interfaces. The programmer prefers the second even though there is no requirement for another data store or identity provider. She need not predict either change. The distinction is between forecasting a future requirement and choosing a structure that remains easier to change when an unrepresented requirement arrives.

That does not make generality good in itself. Flexibility has a carrying cost. An interface that is never substituted or an abstraction built for a variation that never arrives can make a system larger and harder to understand. The engineering maxim against building what is not yet needed exists for this reason (Beck, 1999). Anticipatory smell includes knowing where not to add flexibility. Any measure of it must count the cost of unused generality alongside the savings when flexibility is later used.

3.2 Evidence from Coding Agents

Recent studies of coding agents bear indirectly on this distinction, though none measures anticipatory smell itself.

Thillen et al. (2026) construct a benchmark of large multi-file refactorings mined from open-source repositories. Frontier agents perform well when the desired refactoring is specified in detail and worse when given only a focus area and asked to identify the refactoring chosen by human developers. The difficulty therefore appears to lie partly in deciding *what* should be changed rather than carrying out a supplied change. But the benchmark does not test anticipatory robustness directly. Its target is agreement with the developers’ choice, not lower adaptation cost under later change. The focus area is also supplied, so the open condition tests elicited rather than unprompted anticipation.

Two field studies examine agents in ordinary use. Ottenhof et al. (2026) compare agent-authored pull requests across 86 Java projects with a developer baseline. Agent refactorings are dominated by annotation-related changes, while developers more often make structural changes. Horikawa et al. (2025), analyzing more than fifteen thousand agent-generated refactoring instances in Java pull requests, likewise find that agents mainly perform low-level, consistency-oriented edits rather than high-level design changes. Field data come closer to unprompted behavior, but neither study observes all the instructions the agents received. Horikawa et al. (2025) also find that refactoring was an explicit target in roughly a quarter of agent commits. The populations differ in project and period as well, since Ottenhof et al.’s developer baseline comes from projects that predate agentic tools.

Recent work on generated-code quality points in the same direction. Zhu et al. (2026) report increasing bloat and coupling as model capability rises and find that functional correctness and detailed prompting do not eliminate this architectural degradation. Their measures are not longitudinal, so the result does not establish anticipatory smell, but it reinforces the distinction between present task success and maintainability. Taken together, these studies are consistent with a gap between carrying out a specified structural change and identifying an unstated structural concern. They do not isolate that gap or show that current systems lack anticipatory smell. Stronger evidence would be unprompted design choices that later prove easier to adapt to changes absent from the original task. The behavioral test developed below is designed to measure that.

4 Theoretical Foundations

Anticipatory smell brings together two ideas from different traditions. The first is tacit expertise, where judgment draws on knowledge the decision-maker cannot fully state. The second is decision-making under an open future, where relevant future states cannot all be specified in advance. Together they explain how experience can shape a present choice even when neither the full basis of the judgment nor the future condition to which it responds is explicitly represented.

4.1 Tacit Knowledge and Expert Judgment

Polanyi (1966) argues that human knowledge exceeds what can be stated as propositions or rules. Skilled action depends on particulars that guide judgment without becoming its direct object. An experienced engineer may give reasons for preferring one design without exhausting the basis of the judgment. Years of failed abstractions, difficult migrations, operational incidents, and successful redesigns change how a present structure is perceived.

Collins (2010) distinguishes tacit knowledge by why it remains unarticulated. Some could be stated but are not; some is carried in bodily skill; some is sustained by collective social practice. The distinction matters here. If anticipatory smell is tacit only because links between design choices and later costs were never recorded together, then those links may in principle be recoverable from records that preserve them. Collins argues, however, that tacit knowledge sustained by social practice is unavailable to machines that have not been socialized into that practice (Collins, 2018). That stronger claim matters when considering whether software records alone are enough.

Dreyfus and Dreyfus (1986) also place the move from explicit rules to situational responsiveness at the center of expertise. Novices rely on context-free rules; with experience, judgment becomes responsive to whole situations and distinctions that resist a fixed rule set. Dreyfus and Dreyfus take this to show that machines cannot attain expertise. I use their account of how expertise develops but treat the conclusion about machines as an empirical question.

The term *compression* is useful in a limited sense. A long history of experience affects present judgment through something much smaller than the set of episodes from which it was learned. An engineer does not replay earlier projects before deciding that an abstraction is too rigid. Experience has changed how present cues are read.

Tacit judgment can also be wrong. Kahneman and Klein (2009) argue that reliable intuition requires learnable regularities and repeated opportunities to receive feedback. Experience alone is not enough when outcomes are noisy, causal structure is unstable, or feedback is

absent. Klein (1998) reaches a similar conclusion through recognition-primed decision-making. Experts often recognize a plausible course of action and mentally simulate it instead of comparing a full set of alternatives, but the quality of that recognition depends on earlier situations in which cues, actions, and outcomes were connected closely enough to be learned.

Ghoshal’s “smell of the place” supplies the paper’s metaphor (Ghoshal and Bartlett, 1994, 1997; Ghoshal, 1997). It describes how people respond to the aggregate character of a workplace without reducing it to a list of policies. Anticipatory smell has the same partly inarticulate character, but extends it through time. The judgment concerns how a present design may fare under conditions that have not yet arisen.

4.2 Uncertainty with an Incomplete State Space

Tacit expertise explains how judgment can draw on knowledge that is not fully articulated. It does not explain why that matters when the future itself is incompletely specified.

Under risk, the relevant states can be specified and assigned probabilities. Knightian uncertainty arises when those probabilities are not known (Knight, 1921). Kay and King (2020) use *radical uncertainty* for cases in which the difficulty also includes identifying the relevant states. Lane and Maxfield (2005) call this *ontological uncertainty*, where the uncertainty concerns which entities, relations, and states matter at all.

Decision theory captures part of this problem through *unawareness*. Karni and Viero (2013) model agents whose state space expands as they become aware of new consequences or acts. Grant and Quiggin (2013) consider agents who learn from experience that they are likely to encounter contingencies they cannot yet describe and may therefore value caution or flexibility. An agent can therefore respond to the likelihood of contingencies it cannot presently specify.

Software often evolves under these conditions. Some changes can be forecast and written into requirements. Others become relevant only after new technologies, regulations, security threats, user practices, or complementary systems appear. Anticipatory smell can be understood as a learned disposition toward structures that have proved easier to adapt across earlier changes, without requiring the later change to have been represented in advance.

5 Why Enumeration Does Not Close an Open Future

Searching more extensively over represented futures and remaining robust to futures never represented are different achievements. The distinction applies to any agent.

Suppose an agent at time t represents a finite set of future states $R_t = \{f_1, \dots, f_n\}$, and

let $C(a, f)$ be the cost of action a under state f . Standard decision rules evaluate actions over R_t . With probabilities, an agent may minimize expected cost $\sum_i p_i C(a, f_i)$. Without them, it may use minimax, regret, robustness over admissible distributions, or performance across generated ensembles of plausible futures (Lempert et al., 2003). Exhaustive evaluation may remain infeasible even within R_t (Simon, 1955). More computation can ease that constraint and enlarge R_t by generating scenarios, analogies, or possible failures. But these methods still operate on states that have entered the decision problem.

Let $F_{t,T}$ denote, from an ex post perspective, the states between t and a later horizon T that turn out to matter for the decision made at t , and let

$$U_{t,T} = F_{t,T} \setminus R_t$$

denote those that matter but were not represented when the decision was made. The substantive assumption is that in an open environment $U_{t,T}$ remains non-empty however far R_t is enlarged. Relevant states may arise from technologies or regulations that did not yet exist, changing user behavior, or interactions whose importance becomes apparent only later. The state space may also depend on present choices. An architecture can enable applications that generate new requirements, for example, and a platform can attract users whose behavior changes its design problem. Under this assumption, no search over R_t guarantees coverage of $F_{t,T}$.

5.1 Robustness to Unrepresented Change

A single case illustrates the problem but cannot establish anticipatory smell. Suppose two architectures a_1 and a_2 satisfy the specification at t . A requirement $f_j \in U_{t,T}$ later arrives and $C(a_2, f_j) < C(a_1, f_j)$. That outcome may be luck. The claim of anticipatory smell is about a pattern across decisions. For the unrepresented states that later matter, define

$$\Delta(a_1, a_2) = \sum_{f \in U_{t,T}} w_f [C(a_1, f) - C(a_2, f)] - [K(a_2) - K(a_1)], \quad (1)$$

where w_f weights the importance of each state and $K(a)$ is the carrying cost of the design over the horizon, including generality that is never used. A positive Δ means that a_2 is more robust than a_1 . An agent exhibits anticipatory smell to the extent that its unprompted choices produce positive Δ systematically across repeated decisions. Here the formal boundary is the agent’s own represented state space, whereas Definition 1 uses the task context. Because internal representation is not observable, Section 7.3 uses absence from the task context as a behavioral proxy. The carrying-cost term prevents overengineering from winning by adding

flexibility.

When is such an advantage possible? Let $\mathcal{A}_t$ be a finite candidate set of designs that satisfy the specification at t , and define

$$D(a) = \sum_{f \in U_{t,T}} w_f C(a, f)$$

as the weighted adaptation cost of design a under the unrepresented states that later arise. Then

$$\Delta(a_1, a_2) = D(a_1) - D(a_2) - [K(a_2) - K(a_1)].$$

From the standpoint of t , both $D(a)$ and $K(a)$ are uncertain. Let I_t denote all information available at t , including records of earlier projects.

Proposition 1 (Informational condition for anticipatory robustness). *Suppose that I_t contains no information that distinguishes designs by their expected cost under unrepresented change:*

$$\mathbb{E}[D(a) \mid I_t] = \mathbb{E}[D(a') \mid I_t] \quad \text{for all } a, a' \in \mathcal{A}_t. \quad (2)$$

Let a^K minimize $\mathbb{E}[K(a) \mid I_t]$ over $\mathcal{A}_t$, and let $\hat{a}$ be the choice of any agent whose choice is determined by information in I_t , together with randomization independent of the realized future. Then

$$\mathbb{E}[\Delta(a^K, \hat{a}) \mid I_t] = \mathbb{E}[K(a^K) \mid I_t] - \mathbb{E}[K(\hat{a}) \mid I_t] \leq 0.$$

Proof. Conditional on I_t , let the agent choose each $a \in \mathcal{A}_t$ with probability $\pi(a)$, independently of the realized future. By (2),

$$\mathbb{E}[D(\hat{a}) \mid I_t] = \sum_a \pi(a) \mathbb{E}[D(a) \mid I_t] = \mathbb{E}[D(a^K) \mid I_t].$$

Likewise,

$$\mathbb{E}[K(\hat{a}) \mid I_t] = \sum_a \pi(a) \mathbb{E}[K(a) \mid I_t].$$

Substituting these terms into $\Delta(a^K, \hat{a})$ gives the equality. The inequality follows because a^K minimizes expected carrying cost. $\square$

The proposition locates the empirical question. If nothing available at t distinguishes designs by their expected adaptation cost under unrepresented change, the only expected advantage comes from carrying cost. The best policy is therefore to minimize that cost. Systematic positive Δ relative to a^K requires (2) to fail on a non-negligible share of decisions. Something knowable at t must predict which designs will later be cheaper to adapt.

That information cannot be a description of the particular change, because the change is not represented. It must instead connect properties of designs—such as coupling, reversibility, localization of assumptions, or preserved optionality—to recurring features of the environments in which those designs are later changed. This is how robustness without foresight is possible. The relevant outcome is adaptation cost, not scenario accuracy. The logic resembles real-options accounts of modularity (Baldwin and Clark, 2000; Sullivan et al., 2001) and ecological rationality, where simple heuristics can generalize well because they depend less on the particulars of past data (Gigerenzer and Brighton, 2009).

The same condition explains why anticipatory smell requires learnable regularities. If no regularity connects design properties to later adaptation cost, condition (2) holds and no agent can systematically outperform the carrying-cost minimizer. This is the formal counterpart of the feedback condition for reliable intuition in Kahneman and Klein (2009).

The result does not depend on how a choice is formed. It therefore binds human and artificial agents alike, including agents that generate possible futures before deciding. Those generated futures become part of R_t . Robustness to states that remain in $U_{t,T}$ still requires a design-level regularity that helps beyond the futures explicitly represented.

5.2 Where Agents Can Differ

The proposition binds every agent at the moment of choice. Differences can arise in what an agent extracts from available information and in how its awareness changes over time.

First, records in I_t may contain regularities that a particular agent has not learned. Presence in the records is not enough. To gain anticipatory robustness, the agent must extract a relation between present design properties and later adaptation costs. This is where competing accounts of the learning signal matter, as do the arguments of Dreyfus and Collins that some forms of expert knowledge may not be recoverable by a machine.

Second, agents may differ after t . A state in $U_{t,T}$ eventually becomes visible, and an agent may or may not notice it, connect it to an earlier decision, and revise commitments while revision remains cheap. In the terms of Karni and Vierø (2013), agents can differ in how their awareness grows. Each time an agent connects a later state to an earlier choice, it gains another opportunity to learn which design properties make adaptation easier. That feedback can shape decisions made after t .

For agents facing the same information, differences in anticipatory judgment can arise from what they extract from experience and from how they notice and learn from later developments. No agent escapes the informational condition above.

6 Four Modes of Cognitive Activity

The preceding distinctions yield four modes of cognitive activity: specification execution, gap closure, anticipatory smell, and self-originated inquiry. They differ in what the environment supplies and what the agent contributes. They are modes, not levels, and they do not nest. Anticipatory smell operates within a supplied task, while an agent capable of originating a problem need not possess reliable anticipatory judgment.

Mode	What is supplied	What the agent generates	Software example	Evidence required
1. Specification execution	Problem, requirements, and criterion	A solution satisfying the specification	Implement a function to pass stated tests	Successful task completion
2. Gap closure	Focal problem, criterion, and domain	A previously unknown solution or representation	Resolve an open issue or devise a faster algorithm	Novel solution within a supplied problem
3. Anticipatory smell	Current task; future concern not supplied	A solution shaped by concerns absent from the specification	Fix a bug while localizing a fragile assumption	Unprompted choices with lower later adaptation costs, net of carrying cost
4. Self-originated inquiry	No externally specified focal problem	Selection or reframing of the problem	Treat the module’s design, rather than the reported bug, as the task	Endogenous change in the focal task; cross-task persistence as supporting evidence

Table 2: Four modes of cognitive activity.

Modes 1 and 2 share an externally supplied focal problem and criterion. They differ in the inferential distance between specification and solution. Benchmarks such as SWE-bench (Jimenez et al., 2024) can test demanding planning, inference, debugging, tool use, and search within these modes. High performance shows that a system can cross a large gap between a supplied problem and an acceptable solution. It does not show how the system responds to considerations omitted from the task.

6.1 Degrees of Anticipation

Mode 3 begins when a future-oriented consideration absent from the supplied specification affects the choice. Three prompt conditions matter.

- (a) *Explicit foresight.* The system is instructed to enumerate future states and evaluate the current decision against them. The future is part of the assigned problem.
- (b) *Elicited anticipation.* The system receives an open instruction such as “identify hidden long-term risks” or “design for possible future change.” Particular futures remain unspecified, but the instruction to look for them is supplied.
- (c) *Unprompted anticipation.* The system receives an ordinary task and introduces a concern about future adaptability, fragility, or optionality that was absent from the prompt. It may represent particular futures for itself, or the concern may remain unstated.

Only the third condition can provide evidence of anticipatory smell as defined here, because the future concern itself was not supplied. Whether the agent explicitly represents particular futures is a separate question. A self-generated future becomes part of the agent’s represented state space; later robustness to changes it did not represent provides stronger evidence.

Stated concern may also be decoration. A system trained to be thorough can mention extensibility without changing what it builds. Such remarks count as evidence only when the artifact changes in the direction the concern implies.

When the concern is not stated, anticipatory judgment must be inferred from a pattern of choices. This requires a reference distribution of designs that satisfy the same specification without anticipatory instruction. Evidence for smell requires the agent’s choices to prove more robust than that reference across tasks rather than in isolated cases.

Standing instructions also matter. A system prompt to “write maintainable and extensible code” supplies a quality criterion, so behavior produced under it is elicited anticipation. Dispositions acquired during training are not part of the task specification, although condition (ii) of Definition 1 excludes cases where a codified rule already settles the choice. A behavioral test should therefore use tasks without quality instructions in the system context. It should also distinguish absence of concern from deference to the requested scope (Appendix A).

6.2 Self-Originated Inquiry

Mode 4 concerns who selects the focal problem. Endogenous subgoals within a supplied objective, such as deciding to inspect a database or write a test, are forms of autonomy in execution and do not qualify. The problem itself must not already have been selected as the task to pursue.

Systems designed to generate their own goals test this boundary (Molinaro et al., 2026; Lu et al., 2026). An instruction to propose a problem specifies the activity but may leave

the problem itself open. Lu et al. (2026), for example, compare a template-based system that begins from human-provided code with a more open-ended template-free system. The contrast is one of supplied direction: whether a selected problem counts as self-originated depends on how tightly the brief fixes the focal question. Origination in this sense says nothing about whether the resulting direction is valuable.

The boundary between Modes 3 and 4 lies in what changes. In Mode 3, the objective remains supplied and an anticipatory concern changes how it is met. Asked to fix a bug, an agent may do so in a way that also localizes the fragile assumption behind it. In Mode 4, the agent changes what it treats as the problem. It may conclude that the module, rather than the individual bug, should be redesigned and make that redesign the task. The two modes can occur together, but they require different evidence. Mode 3 appears in properties of the delivered solution; Mode 4 appears in a change in the focal problem.

One possible marker for Mode 4 is the *persistence of open concerns*. An agent may maintain a question it generated itself across tasks and act on it when later information arrives. Current benchmarks are usually episodic, which makes this difficult to observe. Persistence is not sufficient by itself, since an agent can also persist in a supplied question, but it provides supporting evidence when the concern was self-generated.

The four modes are not a scale of value. Specification execution can be consequential, gap closure can produce major advances, and self-originated inquiry can produce poor or harmful directions. The point of the framework is evidential. Achievement in one mode should not be treated as evidence for another without the behavior that the latter requires.

7 Is Smell Recoverable from Artifacts?

If expert smell is compressed experience, a natural objection follows. Large models trained on software corpora already compress experience at scale, so perhaps anticipatory smell will emerge with enough data and capacity. That possibility cannot be ruled out by definition. The question is what signal the judgment is learned from. In the terms of Proposition 1, what allows an agent to learn the relation between present design choices and their later adaptation costs?

7.1 Two Accounts of the Learning Signal

The *artifact-distribution account* holds that the required information can be recovered from software artifacts as they ordinarily appear:

$$\text{artifacts and ordinary histories} \longrightarrow \text{statistical learning} \longrightarrow \text{expert-like judgment.} \quad (3)$$

Reused abstractions, refactorings, documentation, review comments, issue threads, version histories, and conventions all contain traces of earlier engineering judgments. On this account, a sufficiently capable learner can recover the relevant regularities from those traces, including links between decisions and consequences that are only implicit. Explicitly pairing a design decision with what happened later should therefore add little once scale and capacity are sufficient.

The *trajectory-linkage account* makes a different claim. Reliable intuition depends on feedback that connects earlier choices to later outcomes. An engineer who works with a system for years observes such trajectories. A choice made at t meets a new requirement at $t + 1$, a migration at $t + 2$, and perhaps a failure under an unanticipated use at $t + 3$. Across projects, these episodes provide a signal linking properties of the earlier design to the cost of changing it later:

$$\text{decision at } t \longrightarrow \text{later environment} \longrightarrow \text{adaptation cost} \longrightarrow \text{updated structural judgment.} \quad (4)$$

Silver and Sutton (2025) argue more generally for agents that learn from streams of their own experience and its consequences rather than only from static human data. The claim here is narrower. Later software evolution may teach an agent which properties of an earlier design made it easier or harder to change when conditions arrived that were not specified at the time. A related result outside software comes from Tong et al. (2026), who train models to judge scientific work using later citation outcomes and report transfer to future-year papers and unseen fields. This is consistent with later outcomes providing a useful learning signal, though it does not test whether explicit trajectory linkage adds information beyond the same material in dispersed form.

The two accounts agree that software corpora contain histories. Version control, issue threads, pull-request discussions, and post-mortems often preserve the ingredients of sequence (4). The disagreement is over whether their ordinary form is enough. A design decision and the later condition that exposes its consequences may be separated by months of unrelated commits and scattered across files and discussions. The trajectory-linkage account holds that this weak linkage matters.

Hypothesis 1. *Holding model capacity, training compute, and source material constant, training in which earlier design decisions are paired with their performance under later, previously unspecified conditions produces greater unprompted anticipatory robustness than training on the same material in its ordinary dispersed form.*

The hypothesis is about linkage, not additional content. Pairing makes the decision–consequence relation more directly observable. The hypothesis is that this improves learning. The artifact-distribution account predicts that sufficiently capable models should recover much the same relation without it.

7.2 Discriminating Predictions

The accounts can be compared using three training conditions built from the same project histories, with content and token counts matched:

- In the *paired* condition, design decisions are placed with the later changes, adaptation costs, and discussions associated with their consequences.
- In the *natural* condition, project histories remain chronological and are segmented into ordinary training windows. A decision and its consequence share a context only when chronology happens to place them together.
- In the *shuffled* condition, later consequences are reassigned to unrelated earlier decisions, preserving the material while removing the decision–consequence relation.

Pairing should co-locate existing material rather than add explanations of the connection. Training and evaluation repositories should also be separated at the project level so that the test measures transfer rather than recovery of known project histories. Pairs can be identified through co-change analysis and line-level provenance or by expert annotation. Because deciding which later changes resulted from which earlier decisions is itself a form of retrospective judgment, mined pairs should provide the primary test; comparison with curated pairs can estimate the effect of annotation.

The shuffled condition is a floor. Paired training should beat it whenever the decision–consequence relation carries useful information. The important comparison is paired versus natural training. Let AR_u denote anticipatory robustness under unprompted evaluation. The artifact-distribution account predicts

$$AR_u(\text{paired}) \approx AR_u(\text{natural})$$

at sufficient scale, while Hypothesis 1 predicts

$$\text{AR}_u(\text{paired}) - \text{AR}_u(\text{natural}) > 0. \quad (5)$$

The difference should appear in the unprompted condition. An advantage that appears only when the system is asked to think about future adaptability would show improved elicitable knowledge, not a stronger anticipatory disposition. The hypothesis is weakened if paired and natural training are indistinguishable, if the advantage appears only under elicitation, or if it shrinks toward zero as models become larger. The last result would suggest that larger models can recover from dispersed histories what smaller ones need explicitly paired. If all three conditions achieve similarly high robustness, the result favors a stronger artifact-distribution account in which artifact content itself is enough. If all three perform poorly, neither account receives support.

The experiment can be run through fine-tuning of open-weight models and does not require control over pretraining. It tests whether linkage adds to whatever pretraining has already supplied. Comparisons across model sizes can also test whether any benefit from pairing falls with scale, though not at the scale of the original pretraining corpus.

7.3 A Behavioral Test

Measuring AR_u requires decisions made with information available at time t and evaluation against requirements that arrived later. Historical software repositories provide such cases without requiring the evaluator to invent the future. A project can be reconstructed at time t using only the requirements, issues, and interfaces then available. The agent produces or revises a design, which is later tested against requirements that appeared after t and were absent from the original task.

The agent’s internal representation is not observable, so the test uses absence from the task context as a behavioral proxy for $U_{t,T}$. A later state qualifies for evaluation when it was absent from the task context at t . A future that was present in the context but overlooked by the agent receives no credit. This proxy cannot establish that the agent itself failed to represent the state internally, which is why robustness to unrepresented change remains the stronger theoretical case.

Robustness is measured relative to a reference set $\mathcal{A}_0$ of designs satisfying the same specification, produced without anticipatory instruction by a diverse set of agents. Let $\bar{C}_0(f_j)$ be the mean cost of adapting those designs to requirement f_j , and let $\bar{K}_0$ be their mean

carrying cost. Then

$$\text{AR}(a_i) = \sum_j w_j [\bar{C}_0(f_j) - C(a_i, f_j)] - [K(a_i) - \bar{K}_0]. \quad (6)$$

The carrying-cost term captures complexity added for flexibility that the later requirements never use. Without it, maximal abstraction could score well by anticipating everything. The adaptation and carrying-cost terms should therefore be reported separately. A design can beat the reference set by carrying less unused generality, which requires no predictive information. Evidence for anticipatory smell requires both a positive adaptation-cost advantage and a positive net score.

The test should also vary the prompt. In an explicit-foresight condition, the agent first generates possible future requirements. In an elicited condition, it is asked to consider long-term adaptability. In the unprompted condition, it receives only the contemporaneous task, with no quality instruction in the system context. Better performance only in the first two conditions is evidence of foresight or elicited anticipation. Better performance in the third is evidence consistent with anticipatory smell.

Tasks should be coded in advance, by raters blind to later outcomes, for whether applicable design rules leave structurally different choices open. This follows condition (ii) of Definition 1. The strongest cases are those where the design that later proves easier to adapt was not already dictated by a codified heuristic. Tasks can also be coded for dependence on practice-specific conventions, which allows the test to probe Collins’s claim that some relevant knowledge depends on participation in a social practice.

Two threats require separate controls. First, a model may have seen the later history of a public project during training and reproduce that history rather than anticipate it. Second, measured adaptation cost depends on who performs the adaptation. An artificial adapter may favor code written in its own idiom, while a human may favor familiar styles. Appendix A gives controls for both problems.

The same design tests Hypothesis 1. Models trained under the paired, natural, and shuffled conditions are evaluated on the unprompted task, and prediction (5) is tested directly. Thillen et al. (2026) show that mined repository histories can support reproducible evaluation of large refactorings. The present test extends that approach from recovering human refactoring choices to measuring adaptation to requirements that appeared after the original design.

Trajectory linkage is not the only possible account of anticipatory judgment. Dreyfus and Collins both argue, in different ways, that some of what experts learn may be unavailable to a machine. Proposition 1 locates the issue. Any agent that succeeds must acquire some relation between present design properties and later adaptation cost. It does not tell us what

kind of experience, learner, or embodiment is sufficient to acquire it.

Dreyfus and Dreyfus (1986) argue that expertise depends on responsiveness to whole situations that cannot be reduced to rules and context-free features. They take this to rule out machine expertise. The argument here treats that conclusion as empirical. A learner shaped by feedback need not be well described as applying explicit rules. Dreyfus later argued that even neural learners would fail to generalize as humans do without a human-like body and background (Dreyfus, 1992). The behavioral test cannot settle that general embodiment claim, but it can ask whether trajectory-linked records are sufficient in software, where later adaptation costs provide an observable consequence of earlier design choices.

Collins (2010, 2018) places the boundary in social practice. His collective tacit knowledge is acquired through participation in a form of life, and he argues that an unsocialized machine cannot possess it. Some anticipatory judgment may have this character because the cost of changing software depends on who maintains it, with what tools, and under which conventions. Collins’s idea of interactional expertise nevertheless allows that some tacit knowledge can be acquired through immersion in the language of a practice without full participation (Collins and Evans, 2007). If trajectory-linked training improves robustness in general but not in practice-dependent cases, that pattern would support this distinction.

A stronger account might make anticipatory smell depend on endogenous attention, intrinsic motivation, or self-generated purpose. These could affect what an agent notices and which later outcomes it connects to earlier choices (Stanley and Lehman, 2015). The trajectory-linkage hypothesis does not require that stronger claim. It makes a narrower prediction and can fail. If linkage helps, it identifies one source of anticipatory judgment. If it does not, these alternatives become more important.

8 Implications

Different capabilities require different evidence.

8.1 AI Evaluation and Machine Creativity

Most AI benchmarks begin after the evaluator has selected the problem, specified an objective, and set a criterion of success. They therefore provide direct evidence about specification execution and gap closure. Problems arise when performance on such tasks is also taken as evidence of judgment or problem sensing. Evaluation should vary how much structure the agent is given. Future concerns can be specified, elicited through a general instruction, or left unstated. Performance differences across these conditions show how much of the judgment

came from the agent rather than the task.

The same point applies to human evaluation. Examinations and professional tests supply the problem, criterion, and time available. They can measure difficult problem solving without measuring anticipatory judgment or problem setting. Strong examination performance and strong architectural judgment can therefore come apart in the same engineer.

Scientific discovery raises a similar issue. As systems increasingly generate, test, and write up hypotheses (Lu et al., 2026), claims of independent discovery should specify what the system contributed. Solving a stated problem, recognizing an anomaly as important, and choosing a new direction are different acts. Anticipatory smell has an analogue in the researcher who notices that an anomaly matters before she can say why. Evidence for that capacity would lie in unprompted recognition of consequential anomalies.

The same distinction bears on machine creativity. Much of that debate has focused on whether outputs are historically new, predictable from inputs, or explainable by designers. Gap closure can satisfy such criteria while leaving the source of direction unchanged. Self-originated inquiry instead asks whether the system changes what it treats as the problem. That can be studied behaviorally, especially when self-generated concerns persist across interactions.

8.2 Software Engineering

A system can perform well on current coding benchmarks while producing designs that later prove costly to change. Such benchmarks provide little evidence either way about future adaptability. A poor local implementation is often exposed quickly. An architectural commitment may reveal its cost only after other components, users, and processes have come to depend on it.

This matters as coding systems take on larger parts of software development. Faster generation also increases the rate at which architectural commitments are made. Their quality therefore matters independently of the effort required to write the code. Longitudinal evaluation of the kind proposed here would measure that quality directly. Hypothesis 1 also suggests a training intervention: preserve the links between design decisions and the migrations, failures, and refactorings that later reveal their consequences.

8.3 Limits and Transfer

A disposition learned from past feedback will not remain useful under every novel future. Anticipatory smell requires recurring structure. If later changes bear no relation to what earlier experience could teach, human and artificial judgment should both fail. The same

condition applies to organizations. When decision-makers leave before seeing the consequences of their choices, the feedback from which such judgment might form is weakened.

Robustness to a withheld instance of a familiar kind of change may reflect ordinary generalization. Stronger evidence comes from changes that differ in kind from those represented during learning. In the terms of Chollet (2019), the difficulty of generalization depends on the relation between the test case and the agent’s prior experience. A system trained on transitions from single-tenant to multi-tenant architectures has learned something narrower if its advantage disappears under a new security model, regulatory constraint, or platform interaction. The empirical question is how far the learned disposition transfers beyond the changes that produced it. The same question applies to human expertise.

9 Conclusion

AI can produce solutions that are difficult, valuable, and historically unprecedented while working on problems whose objectives were supplied from outside. I have called this *gap closure* and separated it from anticipatory smell, where unprompted choices prove better adapted to later change, and from self-originated inquiry, where the agent selects or reframes the problem itself. These capacities require different evidence.

Anticipatory robustness has an informational condition. Beyond what can be gained by minimizing carrying cost, an agent can systematically choose designs that adapt better to unrepresented change only if something available at the time of choice predicts differences in later adaptation cost. Enumerating more possible futures does not by itself provide that information. This limit applies to human and artificial agents alike.

The Lovelace objection therefore survives in a narrower form. Turing was right that machines can produce novelty and surprise. Novel output, however, does not settle where the direction of inquiry came from. That question can be made behavioral by asking whether an agent changes what it treats as the problem without another actor having selected that problem as the focal task.

Whether current systems possess anticipatory smell remains open. I have proposed that one source of such judgment may be learning signals that preserve the link between design decisions and their later consequences. The competing artifact-distribution account predicts that ordinary software histories are already enough. The proposed training comparison and behavioral test allow those accounts to be separated empirically. Dreyfus and Collins suggest further places where the trajectory-linkage account may fail. On this evidence, the “yet” in the title is an empirical question.

A Protocol for the Behavioral Test

This appendix records the main controls needed to implement the test in Section 7.3.

Contamination. Three approaches can reduce contamination: repositories whose relevant history post-dates the model’s training cutoff, following contamination-controlled code benchmarks (Jain et al., 2025); private codebases used under data agreements; and evaluator-authored evolution scenarios whose later requirements are withheld until the design is fixed. Each should include a memorization probe. The probe must distinguish anticipation from knowledge of what happened. Predicting that a brittle identity layer will cause trouble is not contamination. Reproducing later issue titles, identifiers that did not exist at t , or specific later implementation choices is stronger evidence that the future was already known.

The adapter. Each design is adapted by a fixed panel blind to its provenance. The panel should include human developers and agents from more than one model family. Agreement across adapters is reported. Adaptation cost is measured through lines and files changed, interfaces modified, regressions introduced, and time to passing tests.

The human baseline and the realized path. The historical architecture at t provides a natural human comparison, but effort inferred from commits is not directly comparable with adapter-measured cost. The historical architecture should therefore be passed through the same adapter panel. The commit record can then serve as a check on whether the panel’s estimates track actual development effort.

Historical replay has two further limits. Because the future state space is partly endogenous to the architecture, it measures robustness to the path that occurred, not to the futures each alternative design might have produced. Designs resembling the historical architecture may therefore inherit some advantage. Replay also selects projects that survived long enough to receive later requirements, which may underrepresent costly forms of change. Evaluator-authored evolution scenarios can partly address both problems.

Reference set and carrying cost. The composition of $\mathcal{A}_0$ must be reported because it determines the reference point. Carrying cost can include code added only to support generality, layers of indirection, interfaces or extension points with one implementation or none, and the time an unfamiliar developer needs to make an unrelated change. The weights w_j , the aggregation rule for C , the components of K , and the heuristic set used to code condition (ii) should be fixed before outcomes are observed. Component-level results should be reported alongside the aggregate score.

Scope and deference. Coding agents are often trained to stay within the requested scope. An agent may therefore notice a concern and decline to act on it. Robustness should be reported separately for choices among implementations of equal scope and for choices that

expand the task. A post-task question can ask whether the agent noticed concerns it did not act on. The answer is useful only as a diagnostic of deference and should not count as evidence of anticipatory smell.

Sample size. Adapter variance may be large relative to the effects of interest. The number of projects should therefore be set by a power analysis using variance estimates from a pilot.

References

- Baldwin, C. Y. and Clark, K. B. (2000). *Design Rules, Volume 1: The Power of Modularity*. MIT Press.
- Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.
- Boden, M. A. (2004). *The Creative Mind: Myths and Mechanisms*. Routledge, 2nd edition.
- Bringsjord, S., Bello, P., and Ferrucci, D. (2001). Creativity, the Turing test, and the (better) Lovelace test. *Minds and Machines*, 11(1):3–27.
- Chollet, F. (2019). On the measure of intelligence. *arXiv preprint arXiv:1911.01547*.
- Clay Mathematics Institute (2026). Navier–Stokes announcement. News statement, 11 September, <https://www.claymath.org/news/navier-stokes-announcement/>.
- Collins, H. (2010). *Tacit and Explicit Knowledge*. University of Chicago Press.
- Collins, H. (2018). *Artificial Intelligence: Against Humanity’s Surrender to Computers*. John Wiley & Sons.
- Collins, H. and Evans, R. (2007). *Rethinking Expertise*. University of Chicago Press.
- Cunningham, W. (1992). The WyCash portfolio management system. In *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA ’92)*, pages 29–30.
- Dreyfus, H. L. (1992). *What Computers Still Can’t Do: A Critique of Artificial Reason*. MIT Press.
- Dreyfus, H. L. and Dreyfus, S. E. (1986). *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. Free Press.
- Floridi, L. (2023). AI as agency without intelligence: On ChatGPT, large language models, and other generative models. *Philosophy & Technology*, 36(1):15.

- Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley. With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts.
- Ghoshal, S. (1997). The individualized corporation: An interview with Sumantra Ghoshal. *European Management Journal*, 15(6):625–632.
- Ghoshal, S. and Bartlett, C. A. (1994). Linking organizational context and managerial action: The dimensions of quality of management. *Strategic Management Journal*, 15(S2):91–112.
- Ghoshal, S. and Bartlett, C. A. (1997). *The Individualized Corporation: A Fundamentally New Approach to Management*. HarperBusiness.
- Gigerenzer, G. and Brighton, H. (2009). Homo heuristicus: Why biased minds make better inferences. *Topics in Cognitive Science*, 1(1):107–143.
- Grant, S. and Quiggin, J. (2013). Inductive reasoning about unawareness. *Economic Theory*, 54(3):717–755.
- Horikawa, K., Li, H., Kashiwa, Y., Adams, B., Iida, H., and Hassan, A. E. (2025). Agentic refactoring: An empirical study of AI coding agents. arXiv preprint arXiv:2511.04824.
- Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. (2025). LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*.
- Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. (2024). SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*.
- Kahneman, D. and Klein, G. (2009). Conditions for intuitive expertise: A failure to disagree. *American Psychologist*, 64(6):515–526.
- Karni, E. and Vierø, M.-L. (2013). “reverse Bayesianism”: A choice-based theory of growing awareness. *American Economic Review*, 103(7):2790–2810.
- Kay, J. and King, M. (2020). *Radical Uncertainty: Decision-Making Beyond the Numbers*. W. W. Norton.
- Klein, G. (1998). *Sources of Power: How People Make Decisions*. MIT Press.
- Knight, F. H. (1921). *Risk, Uncertainty and Profit*. Houghton Mifflin.

- Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21.
- Lane, D. A. and Maxfield, R. R. (2005). Ontological uncertainty and innovation. *Journal of Evolutionary Economics*, 15(1):3–50.
- Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076.
- Lempert, R. J., Popper, S. W., and Banks, S. C. (2003). *Shaping the Next One Hundred Years: New Methods for Quantitative, Long-Term Policy Analysis*. RAND Corporation.
- Lovelace, A. A. (1843). Notes by the translator. *Scientific Memoirs*, 3:666–731. Appended to her translation of L. F. Menabrea, “Sketch of the Analytical Engine Invented by Charles Babbage, Esq.”; Note G.
- Lu, C., Lu, C., Lange, R. T., Yamada, Y., Hu, S., Foerster, J., Ha, D., and Clune, J. (2026). Towards end-to-end automation of AI research. *Nature*, 651:914–919.
- Molinaro, G., Collins, A. G. E., Oudeyer, P.-Y., and Colas, C. (2026). Self-generated goals in the age of foundation models: Artificial autotelic agents and their implications for cognitive science. *Current Opinion in Behavioral Sciences*, 72:101684.
- Novikov, A. et al. (2025). AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv preprint arXiv:2506.13131.
- OpenAI (2026). On the Navier–Stokes millennium prize problem. Research publication, 8 September, <https://openai.com/index/navier-stokes-solution/>.
- Ottenhof, L., Penner, D., Hindle, A., and Lutellier, T. (2026). How do agents refactor: An empirical study. In *Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026), Mining Challenge Track*, pages 797–801.
- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058.
- Polanyi, M. (1966). *The Tacit Dimension*. Doubleday.
- Riedl, M. O. (2014). The Lovelace 2.0 test of artificial creativity and intelligence. arXiv preprint arXiv:1410.6142.
- Rittel, H. W. J. and Webber, M. M. (1973). Dilemmas in a general theory of planning. *Policy Sciences*, 4(2):155–169.

- Rolla, G. (2026). The embodiment challenge for artificial intelligence. *Philosophy & Technology*, 39:125.
- Schön, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books.
- Silver, D. and Sutton, R. S. (2025). Welcome to the era of experience. *Google AI*, 1(11):6.
- Simon, H. A. (1955). A behavioral model of rational choice. *Quarterly Journal of Economics*, 69(1):99–118.
- Simon, H. A. (1973). The structure of ill structured problems. *Artificial Intelligence*, 4(3–4):181–201.
- Stanley, K. O. and Lehman, J. (2015). *Why Greatness Cannot Be Planned: The Myth of the Objective*. Springer.
- Sullivan, K. J., Griswold, W. G., Cai, Y., and Hallen, B. (2001). The structure and value of modularity in software design. In *Proceedings of the 8th European Software Engineering Conference held jointly with the 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 99–108.
- Thillen, A., Mündler, N., Raychev, V., and Vechev, M. (2026). CodeTaste: Can LLMs generate human-level code refactorings? In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 120544–120572. PMLR.
- Tong, J., Li, M., Li, H., Yang, Y., Mou, Y., Ma, W., Xi, Z., Chen, H., Liu, X., Cheng, Q., Zhang, M., Chen, Q., Ge, W., Guo, Q., Ying, T., Sun, T., Zheng, Y., Chen, X., Zhao, J., Ding, N., Huang, X., Jiang, Y., and Qiu, X. (2026). AI can learn scientific taste. arXiv preprint arXiv:2603.14473.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.
- Zhu, Y., Tsantalis, N., and Rigby, P. C. (2026). AI-generated smells: An analysis of code and architecture in LLM and agent-driven development. arXiv preprint arXiv:2605.02741.