

The Agentic Probe: The Instrument of Agentic Discovery & Engineering

AI agents collapse the software lifecycle into two phases. Agentic probes — prototypes, simulations and possible futures — do the discovering.

Tapio Pitkäranta¹ and Leena Pitkäranta²

¹Department of Computer Science and Engineering, Aalto University · Finland ,
tapio.pitkaranta@iki.fi

²Department of Industrial Engineering and Management, Aalto University · Finland ,
leena.pitkaranta@iki.fi

13 October 2026

Third edition: adds the agentic wave, the state of AI capability in October 2026, the Golden Circle reading, Agentic Search as the engine of Discovery, the twelve-layer view, the field-study growth curve and a Theory-of-Constraints reading of the bottleneck shift. Earlier editions (26 May and 1 October 2026) remain available unchanged.

Abstract

Problem. The traditional software development lifecycle decomposes the path from organisational mission to running software into many sequential phases because each transition was historically expensive — but AI coding agents that produce a deployable artefact in minutes are collapsing those costs, and the SDLC’s phase structure needs to be re-derived from the new economics.

Approach. We propose two complementary phases that replace the traditional chain: *Agentic Discovery* + *Agentic Engineering*. *Agentic Discovery* is an expedition into the problem domain, and its instrument is the *agentic probe*: a stakeholder-runnable artefact that an AI agent builds from the stakeholder’s own words in minutes — a prototype, a process simulation, or a possible future. A *Discovery* session yields such a probe in one of two complementary modes. In the *Problem-validating* mode the probe is a process simulation: the session discovers and renders the current way of working, the constraints that shape it, and the pain it produces — so the stakeholder can confirm, correct or counterfactually relax any of them. In the *Solution-proposing* mode the probe is candidate software — a working implementation the stakeholder evaluates by using it, surfacing the value a better solution would unlock. Both modes pair a stakeholder (the *Dreamer*) with an AI agent that builds the probe, facilitated by a human *Orchestrator* who curates the queue between them.

Evaluation. Across the engagement series the bottleneck moved outward: in events 1–4 every probe was built live in the room; at 7 parallel requests the human Orchestrator became the limit; once we automated key orchestration steps for the two largest engagements (~100 invited; in-room + Teams hybrid), the limit moved one layer further out — to the supply of stakeholder ideas and the speed of stakeholder validation. Eight in-room field engagements with two voluntary partner organisations produced 69 probes — all of the prototype kind — and ≈ 140 k LOC of generated source, with intent→code amplification averaging 1 : 56 and reaching 1 : 467 at the extreme, and 100 % probe-generation success. Simulation and possible-future probes are illustrated in a small companion (n = 3) and mapped, not yet measured.

Contribution. A working *Agentic Discovery* runtime configured as a small Software Factory in which Probe production is no longer rate-limiting, the agentic probe as the instrument of discovery together with a map of the probe space (prototypes, simulations, possible futures) and the evidence for each,

a new IS design theory built around the two-modes pattern and the resulting bottleneck-shift principle (production → orchestration → idea supply), the empirical evidence above, and a public reproducibility kit (anonymised analytics appendix A.1–A.12).

Keywords: Agentic Probe, Agentic Discovery, Agentic AI, SDLC, Software Factory, Problem domain, Rapid prototyping, Design science.

Two modes of Agentic Discovery

Before diving in: this is the diagram the rest of the paper is built around. Read it once now so the introduction (§ 1), the model (§ 2) and the field results (§ 6) all land in context. The detailed discussion lives in § 3 — Two modes of probing.

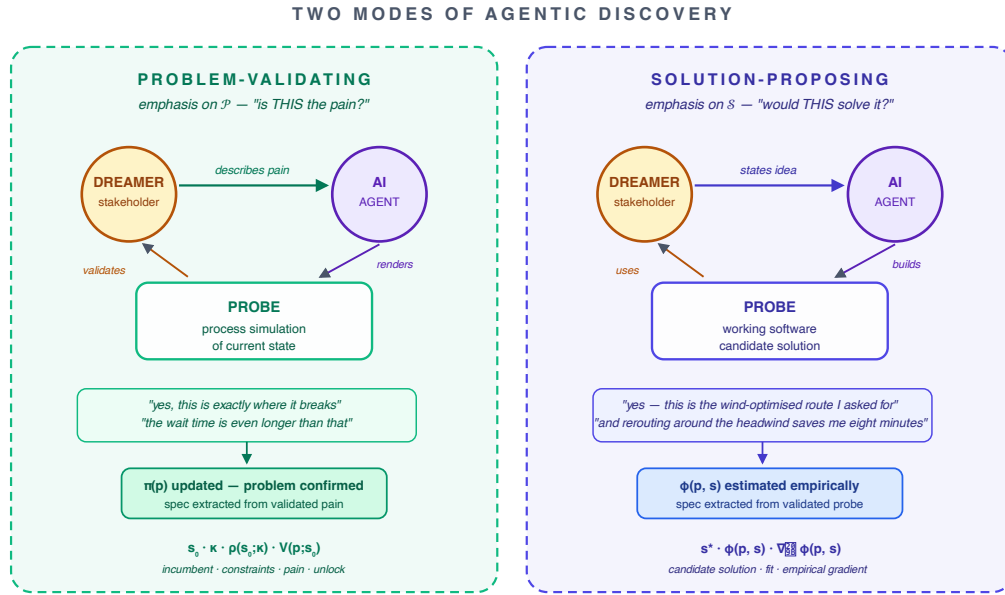


Figure 1: **Read the diagram in two halves.** *Left (Problem-validating):* the agent renders the *current state* as a process simulation. The bottom legend names the four objects the probe is estimating — the incumbent solution s_0 , the constraints κ that bind it, the pain $\rho(s_0; \kappa)$ that follows, and the value $V(p; s_0)$ of unlocking a better s^* — all three of s_0 , κ , ρ are discovered, not assumed (see § 2 frames *Incumbent under constraints* + *Value of the unlock*). The prior $\pi(p)$ updates from those observations. *Right (Solution-proposing):* the agent builds a *candidate solution*; the stakeholder evaluates it by using it. The right panel maximises V — equivalently, minimises ρ — quantified via the fit functional $\phi(p, s)$. Both modes use the same three-role pattern (Dreamer → Orchestrator → Agent); they differ in what the probe *renders*. The whole paper is a study of how this single picture plays out in eight field engagements (§ 6).

1 Introduction

The software development lifecycle has historically decomposed the journey from organisational mission to running software into many sequential phases — each with its own specialists, documents, and handoff rituals. This decomposition exists because each transition was expensive: translating a vision into a strategy, a strategy into requirements, requirements into architecture, architecture into code, code into a tested

release. The entire apparatus of the SDLC — its workshops, specification languages, design reviews, sprint ceremonies, and deployment pipelines — is machinery built to manage those transition costs.

The 2024–2026 wave of large language model (LLM) agents is collapsing those costs. Modern coding agents can produce a deployed, runnable artefact in minutes from a single conversation. When an artefact costs almost nothing to produce, the phase boundaries built to protect expensive artefacts lose their rationale.

The agentic wave. Figure 2 sketches how agentic work has spread across the lifecycle. It began in development — code assistants around 2022, the first coding agents in 2024 — and by 2026 agents are everyday tools in Dev. From there the wave moves in both directions: *downstream* into review, test, integration and operations, and *upstream* into requirements, design, discovery, market research and strategy. The maturity levels in the figure are the authors’ qualitative assessment, each anchored to representative studies of that phase (named under the columns). The upstream half — the least established and the most open to exploration — is the subject of this paper.

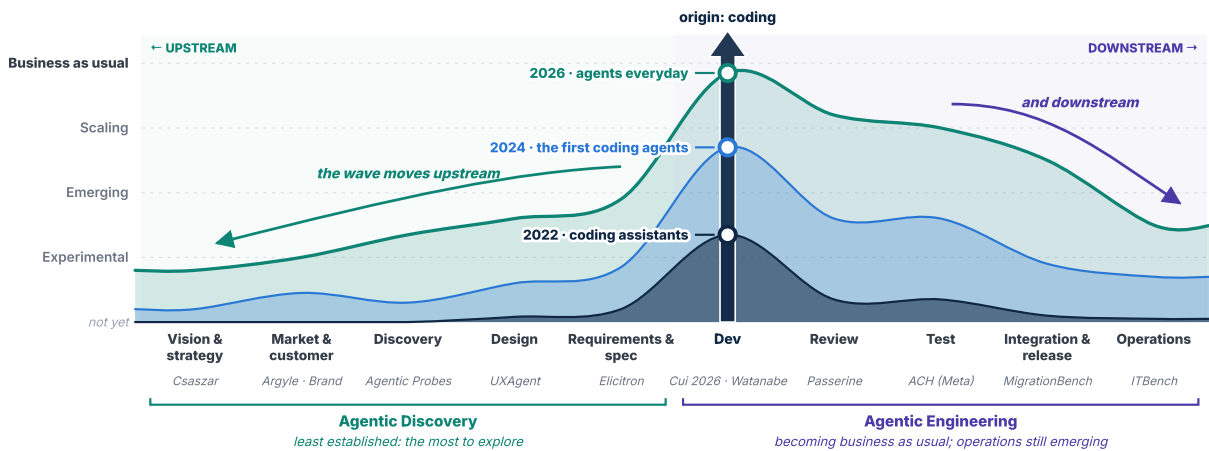


Figure 2: The agentic wave. Agentic work started in development (2022 code assistants, 2024 coding agents, 2026 everyday use) and is spreading upstream (Agentic Discovery) and downstream (Agentic Engineering). Maturity per phase is a qualitative assessment by the authors, anchored to the representative studies named under each phase.

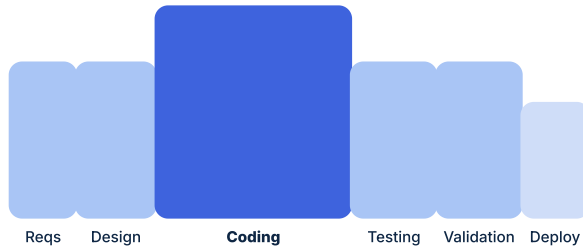
We argue that this cost collapse reduces the effective SDLC to **two complementary phases**. **Agentic Discovery** uses agents to expose and validate problems with stakeholders through *agentic probes* — prototypes, process simulations and possible futures that an agent generates, and a stakeholder tests and iterates, within a conversation. **Agentic Engineering** uses the same agent stack to carry validated specifications through production-grade implementation, testing, and deployment. Between them, the two phases absorb the concerns of every traditional SDLC phase from mission to operations. This application is itself a working instance of both.

Figure 3 illustrates the consequence for where human attention goes. In the traditional SDLC most of the time is spent writing code. When agents produce design, code and tests in parallel, human work moves to the two ends of the chain: finding the right problem with probes (Agentic Discovery) and validating the result at gates (Agentic Engineering). The block sizes are illustrative, not measured.

The agentic probe as instrument. Before a ship commits to a course through unknown water, a sounding probe goes over the side to measure the depth. Agentic Discovery is that kind of expedition into the problem domain, and the *agentic probe* is its instrument: built by an AI agent from a stakeholder’s own words, in minutes; used rather than read; thrown away once it has answered. The paper’s own term is *Probe* (§ 3); “agentic probe” names what makes it possible — agents that build working artefacts on request. A

TRADITIONAL SDLC

Most of the time is spent writing code.



AGENTIC SDLC

Time shifts to finding the problem and validating.

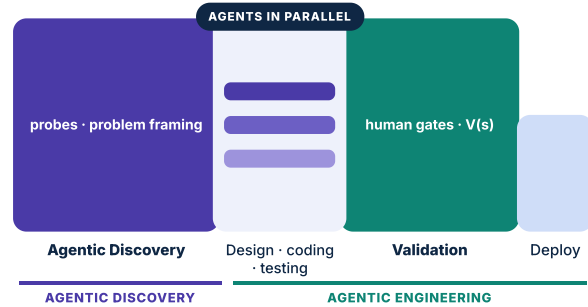


Figure 3: Where the time goes — illustrative, not measured. Left: the traditional SDLC, dominated by coding. Right: the agentic SDLC, in which agents run design, coding and testing in parallel and human attention shifts to problem framing with probes and to validation at human and automated gates.

probe is not always a prototype. It can be working software that tests whether a solution fits, a simulation of today’s process that surfaces its constraints, or a *possible future* that tests which assumptions are reasonable before anything is built. Our field study measured the first kind at scale; the other kinds are illustrated and mapped (§ 3.6), and the full space of probes is hard to imagine yet.

A practical discovery emerged when we ran the same pattern across engagements of growing size. The first four field events handled two to four parallel probe requests per session — every requested probe was built and shown back live. At higher concurrency (seven parallel requests in our fifth event) the human Orchestrator became the limiting factor. For the two largest engagements (~100 invited; in-room + Teams hybrid) we built a small orchestration-automation tool that took a queue of stakeholder briefs and dispatched agent sessions in parallel; the bottleneck moved one layer further out, to **the supply of stakeholder ideas** and **the speed at which stakeholders could validate** the produced probes. We describe this in § 6 and § 7 as a *Software Factory* configuration of Agentic Discovery (§ 8) — small enough to fit in a workshop room, but with production capacity removed as the binding constraint. The same shift applies to process discovery in the Problem-validating mode of § 3 (e.g. the healthcare cases in § 6.8): once a counterfactual process is cheap to render, the limit is how fast stakeholders can author the next “what if” and judge the result.

This progression is best read against the historical baselines: prior empirical studies place requirements and design at roughly a third of project effort (Boehm 1981 [67]; Boehm et al. 2000 [68]) and put the lion’s share of avoidable rework upstream, with defects caught in operations costing roughly $100 \times$ more than those caught during requirements (Boehm & Basili 2001 [69]). What agentic AI removes is the production capacity that historically anchored the discovery loop’s low end; what becomes visible at field scale is the residual that those same baselines have always flagged — idea supply and validation speed.

Reading guide. Section 1.1 summarises the state of AI capability in October 2026. Section 2 lays out the theoretical model — the six SDLC phases and where Agentic Discovery sits inside them. Section 3 defines Agentic Discovery and its instrument, the agentic probe, including the probe space of § 3.6; Section 4 contrasts it with Agentic Engineering. Sections 5–7 describe the method, results, and discussion. Section 8 places the work in context; Section 9 concludes with future work and acknowledgements; Section 10 lists references and recommended reading. The “Slides & Software” companion gives the public deck and the runtime that produced our field data.

Central research question. What is the rate-limiting step in agentic-discovery practice, and how does it move as the practice scales? The objectives O1–O6 below specify what an Agentic Discovery realisation must deliver; the field results in § 6 trace which of those objectives is the binding constraint at each scale. The headline finding (§ 7.4, formalised as C5 below) is that the constraint *migrates* — from production capacity, through human orchestration, to the supply and validation of stakeholder ideas — and that the resulting "small Software Factory" configuration of Agentic Discovery has a sibling on the Problem-validating side of § 3 (process-discovery use cases such as Healthcare in § 6.8).

1.0.1 Contributions

This paper offers five scoped contributions, in the design-science vocabulary of Hevner et al. [5] and Peffers et al. [45]:

- **C1 — Artefact.** A working Agentic Discovery runtime instantiating the Three-Role pattern (Dreamer → Orchestrator → Agent) and the agentic probe as its instrument and unit of stakeholder feedback. The runtime is deployed live (§ 5) and its source lives in this repository.
- **C2 — Design knowledge.** The *two-modes pattern* (§ 3) — Solution-proposing and Problem-validating — separating the two directions in which a Discovery session can travel across the problem/solution seam, plus the *Inside-Out* direction (§ 6.8) where the probe is a process simulation rather than deployable code, and a map of the probe space — prototypes, simulations, possible futures — with the evidence available for each (§ 3.6). Both are framed as a new IS design theory in the Walls–Hevner sense [64, 5], grounded in the prior-update view of problem-domain knowledge from § 2.
- **C3 — Empirical evaluation.** 69 deployable probes of the prototype kind from 8 in-room field engagements with two voluntary partner organisations (≈ 140 k LOC of generated source; intent→code amplification averaging 1 : 56 and reaching 1 : 467 at the extreme on the audited $n = 9$ subset; 100 % probe-generation success across observed sessions). All numeric claims are scoped to DSRM steps 3–5 in the Solution-proposing mode (§ 7.4 L6); the Problem-validating side is exercised only as an illustrative companion (§ 6.8).
- **C4 — Reproducibility kit.** A public anonymised analytics appendix (A.1–A.12, 10 charts), the publicly deployed runtime described in § 5, and provenance documentation for every numeric claim back to its underlying source. The de-anonymised internal appendices remain only in the authors’ working repository and are excluded from the arXiv build.
- **C5 — Empirical finding: the bottleneck shifts.** Across an engagement series of growing size, the binding constraint progressed from production capacity (events 1–4, all built live in the room) through human orchestration (event 5, 7 parallel ideas) to **stakeholder idea supply and validation speed** once we partially automated orchestration for the two largest engagements (~ 100 invited; in-room + Teams hybrid). The same shift applies to the Problem-validating mode of § 3 (process-discovery use cases such as Healthcare in § 6.8): once a probe is cheap to render, the limit becomes how fast humans can generate counterfactuals and judge them. We position this as a corollary of the cost-collapse argument and a re-entry point into the Software Factory tradition (§ 8): the *probabilistic factory* of Agentic Engineering has a sibling on the Discovery side — a small Software Factory in which the rate-limiting factor is no longer the production line but the queue of ideas at its inlet (§ 7.4).

1.1 The state of AI capability, October 2026

The cost collapse rests on capability that is still growing. We summarise the evidence as of October 2026; it is drawn from primary sources and changes quickly.

Task length. METR’s time-horizon measurement — the length of task, in human-expert working time, that a model completes with 50 % success — was re-based in early 2026 on a larger suite (Time Horizon 1.1, 228 tasks), with a doubling time of about 131 days since 2023 and about 89 days since 2024 [77]. In spring 2026 the frontier passed the 16-hour line (Claude Mythos Preview, about 17 hours), above which METR reports that its current task suite no longer measures reliably; the measurement page has not been updated since September 2026 [76]. Figure 4 shows the same points on a logarithmic and on a linear axis: on the linear axis almost all of the growth falls within the last year.

Expert work and the yardsticks. On GDPval, 1,320 real tasks from 44 occupations graded by domain experts [47], the best model rose from 12 % (GPT-4o) through the first result above 50 % (Claude Sonnet 4.5, September 2025) [78] to 83 % reported by OpenAI for GPT-5.4 in March 2026 — a vendor grading its own model [79]. The hardest tests of 2025 are nearing their ceilings (Table 1); in mathematics, Lean-checked proofs now settle open problems at scale [80], and the bottleneck has moved from producing proofs to verifying them.

Yardstick	Top result, Oct 2026	Note
ARC-AGI-2 [81]	95.0 %	below 5 % at launch (March 2025)
ARC-AGI-3 [81]	62.7 % / 99.9 %	the same model with two different harnesses
FrontierMath, tiers 1–3 [82]	93.7 %	below 2 % at launch (November 2024)
Humanity’s Last Exam [83]	54.8 %	a rolling successor was released in September 2026
Terminal-Bench 4.0 [84]	64.8 %	measures agent harness and model together

Table 1: Yardsticks running out. The model name alone no longer determines the result: the harness around it does nearly as much.

The system, not the model alone. The same model scores 62.7 % on ARC-AGI-3 with the standard harness and 99.9 % with another [81]; Terminal-Bench ranks agent–model pairs, and its top entries all run on two harnesses (Claude Code and Codex) [84]. Open-weight models trail the closed frontier by about four months [85]. Agents are already everyday tools: 90 % of 15,000 surveyed developers used coding agents at work weekly and 68 % daily in mid-2026 [86], and 69 % of 7,156 agent-authored pull requests were accepted [87].

What this means for this paper. Building is no longer the bottleneck: long tasks, expert-level work and everyday use make a probe cheap, so $c_S/c_P \rightarrow \mathcal{O}(1)$ is measured rather than assumed. Advantage lies in the agent’s environment, process and data rather than in the model alone, and verification — not production — is the new scarcity. The rest of the paper is about the human work that remains at the two ends: finding the right problem, and validating what was built.

1.2 Methodological frame

This paper follows the Design-Science Research Methodology (DSRM) of Peffers et al. [45], complementing the broader design-science programme of Hevner et al. [5]. The six DSRM activities map onto the paper as follows: (1) *problem identification & motivation* — § 1 above; (2) *define objectives of a solution* — § 1 below (O1–O6); (3) *design & development of the artefact* — § 3 (Agentic Discovery) and § 4 (Agentic Engineering);



Figure 4: METR time horizon (50 % success), Time Horizon 1.1 data [76, 77]. Top: logarithmic axis, with the 2023-onward trend and the 16-hour measurement ceiling. Bottom: the same points on a linear axis (hours).

(4) *demonstration* — § 5 (Method); (5) *evaluation* — § 6 (Experiments & Results); (6) *communication* — this paper itself, together with the publicly deployed runtime described in § 5.

1.2.1 Objectives of a solution (O1–O6)

An Agentic Discovery realisation must satisfy six objectives, defined here and measured against the deployed instance in § 6.7:

- **O1 — Time-to-first-probe** on the order of minutes within the same session as stating the pain — not weeks behind a separate engineering pipeline.
- **O2 — Probe acceptance:** the produced probe is deployable and operable by the stakeholder without additional engineering input.
- **O3 — Spec extractability:** the natural-language spec used to generate the probe is sufficient to reproduce the artefact (no hidden tacit knowledge in the prompter).
- **O4 — Stakeholder satisfaction:** stakeholders voluntarily re-engage in subsequent sessions.

- **O5 — Safety:** probe generation occurs without policy or safety incidents.
- **O6 — Scalability:** the same approach works across multiple sites and problem categories without site-specific re-tooling.

2 Theoretical Model

Agentic Discovery is not a replacement for the SDLC; it is a transformation of how its earliest phases are carried out. Before we walk through the six phases, we ground the discussion in an older — and, we will argue, more permeable — distinction: that between the *problem domain* and the *solution domain*.

2.1 Problem Domain & Solution Domain

Requirements engineering uses the term *problem domain* for the part of the world in which a stakeholder’s need exists — its actors, rules, goals and constraints — and reserves *solution domain* for the artefacts (code, UI, processes) we build to satisfy that need [11, 12]. **Product management** uses the closely related pair *problem space* / *solution space* with the same intent: the problem space is what the customer needs and why; the solution space is the specific implementation that addresses it [13, 14, 15].

The distinction matters because most expensive product mistakes are made by collapsing the two prematurely — solving a well-defined wrong problem, or, more commonly, building a clever solution to a problem the user does not actually have [16].

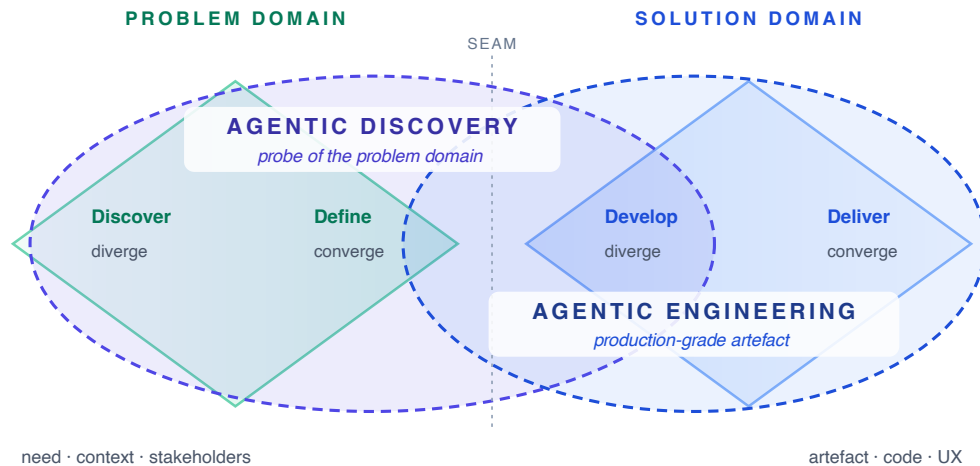


Figure 5: The double-diamond view of the problem and solution domains, after the British Design Council (2005) [17]. Each diamond captures a divergent–convergent cycle. The dashed vertical midline labelled *SEAM* marks the boundary between the problem domain (left diamond) and the solution domain (right diamond) — the classical handover point where requirements engineering ends and design begins. The indigo bubble is *Agentic Discovery* — bridging that seam with a stakeholder-runnable *Probe* (defined in § 3) that serves as the cheapest available instrument for measuring the problem domain. The blue bubble is *Agentic Engineering*, taking over from late Define and carrying the artefact through Deliver. The bubbles overlap deliberately: the overlap region is where Discovery hands the validated artefact to production.

2.1.1 A formal sketch — for the mathematically inclined

Both literatures cited above — requirements engineering and product management — express the same distinction in prose. A short pass through set-theoretic and optimisation language makes the structure explicit, and reduces the move from a *static handover* to a *permeable feedback loop* to a few lines.

Eight small frames suffice. Hover any card for a compact preview; click to read the full derivation in the formal-sketch companion. Cards marked Shared also appear in § 4 (Agentic Engineering); cards marked Discovery are specific to this section.

1. **Problem & solution as sets** — $\mathcal{P}, \mathcal{S}$. Two sets, only partially observable: candidate needs vs. candidate artefacts.
2. **Fit functional** — $\varphi : \mathcal{P} \times \mathcal{S} \rightarrow [0, 1]$. A score: how well a candidate solution addresses a candidate problem.
3. **Discovery as joint search** — $\arg \max \varphi(p, s) \pi(p)$. Search for problem AND solution simultaneously, with a prior π over which problems matter.
4. **Cost ratio shift** — $c_S/c_P \rightarrow \mathcal{O}(1)$. Pre-agentic: building dwarfs describing. With LLM agents the ratio collapses to constant order.
5. **Permeable seam** — $\nabla_s \varphi(p, s)$. The fit gradient is estimated empirically from a running probe, not inferred from documents.
6. **Incumbent under constraints** — $s_0 \in \mathcal{S}, \kappa \in \mathcal{K}, \rho(s_0; \kappa)$. Current way of working s_0 , the constraints κ that bind it, and the pain ρ it produces — all three are discovered, not assumed.
7. **Value of the unlock** — $V(p; s_0) = \rho(s_0; \kappa) - \rho(s^*; \kappa)$. The reachable gain — how much pain disappears when the search lands on s^* .
8. **Two modes — update π and φ** — $\pi(p | o) \propto L(o | p) \pi(p)$. Solution-proposing updates φ via the gradient; Problem-validating updates the prior π via a Bayesian posterior.

Why this matters. Pre-agentic software economics had $c_S(s) \gg c_P(p)$, and that asymmetry is the silent premise of the entire SDLC — it is the reason we explore problems before committing to solutions. Agentic Discovery makes the seam between the two domains *permeable*: a candidate s becomes a probe of $\mathcal{P}$, and the prior $\pi(p)$ updates from real usage rather than from imagination.

2.1.2 How the literature visualises the line

Different communities draw the same boundary with different geometries. Each visual encodes a different assumption about what is cheap (research, ideation, prototyping) and what is expensive (committing, building, shipping).

1. **Double Diamond** — Design Council, 2005 [17]. Two diamonds — Discover/Define then Develop/Deliver — each a divergent–convergent cycle.
2. **Product-Market-Fit Pyramid** — Olsen, 2015 [13]. Five stacked layers: target customer + underserved needs (problem) under value prop + features + UX (solution).
3. **Lean Canvas** — Maurya, 2012 [18]. A 1-page model with adjacent *Problem* and *Solution* boxes; problem/solution fit precedes product/market fit.

4. **Opportunity Solution Tree** — Torres, 2021 [15]. Outcome → opportunities (problem space) → solutions → assumption tests, drawn as a tree.
5. **Problem Frames** — Jackson, 2001 [11]. Context and problem diagrams that draw the Machine and the surrounding domains as named boxes joined by shared phenomena.
6. **Cynefin** — Snowden & Boone, 2007 [19]. Five problem-domain types — clear, complicated, complex, chaotic, confused — each prescribing a different solution stance.
7. **Discovery vs. Delivery** — Cagan, *Inspired*, 2017 [14]. Two parallel tracks; product, design and engineering collaborate across both rather than handing the problem off to engineering.
8. **Are Your Lights On?** — Gause & Weinberg, 1990 [16]. Problem definition as the first-class activity: "a problem is a difference between things as desired and things as perceived."

Why this re-frames the SDLC. Every visualisation above shares one hidden premise: *the problem can be substantially understood before any working artefact exists*. The cost of producing such an artefact is what historically forced this premise. Agentic Discovery does not abolish the problem/solution distinction — it makes the seam between them *permeable*. A stakeholder can now reach into the problem domain by working with a running *probe* (§ 3), the cheapest instrument for measuring it we have yet had. The six SDLC phases below describe what changes when that premise is relaxed.

2.2 From vision to features — vertical layers

A second, complementary lens runs *vertically*. Where the problem/solution view asks *which side of the seam are we on?*, the vision-to-features view asks *at what altitude are we operating?*. The same product organisation must hold a long-horizon vision, a strategy that interprets it, a roadmap that sequences it, and a backlog of concrete features — and every layer must remain answerable to the one above. Simon Sinek's Golden Circle [21] is the most familiar popular framing of this idea: *start with Why*, derive *How*, only then describe *What*. Several practitioner and academic traditions add machinery to that intuition: Pichler [22], Perri [23], and Cagan & Jones [24] each formalise the descent from vision to executable work; Doerr's OKRs [25] and Akao's Hoshin Kanri [26] add bidirectional measurement; the classical strategy hierarchy [27] is the textbook baseline they all critique or extend.

Decision-cost asymmetry. The hierarchy in Figure 6 carries an asymmetry that every layered tradition exploits but few make explicit. Top-of-stack decisions (Vision, Strategy) are cheap to revise in themselves — a vision statement is only text on a slide — but *expensive to misalign*: a wrong vision cascades down through every feature below, and the damage compounds across years of execution. Bottom-of-stack decisions (Features) invert the pair: they are *expensive to revise* once committed (production code, dependencies, accumulated technical debt) but the damage from any single misaligned feature is *locally bounded* — features are leaves of the tree, so there is no downstream amplification. This asymmetry is the reason the traditions above push for testing alignment high in the stack before committing low; Agentic Discovery shortens it by making the probe itself cheap, so the top of the stack becomes empirically testable through a working artefact at the bottom.

2.2.1 How the literature stacks the layers

Different traditions choose different geometries — concentric circles, pyramids, nested rectangles, vertical stacks, cascades — but all encode the same intuition: features are accountable to a roadmap, the roadmap to a strategy, the strategy to a vision, and the vision to a purpose.

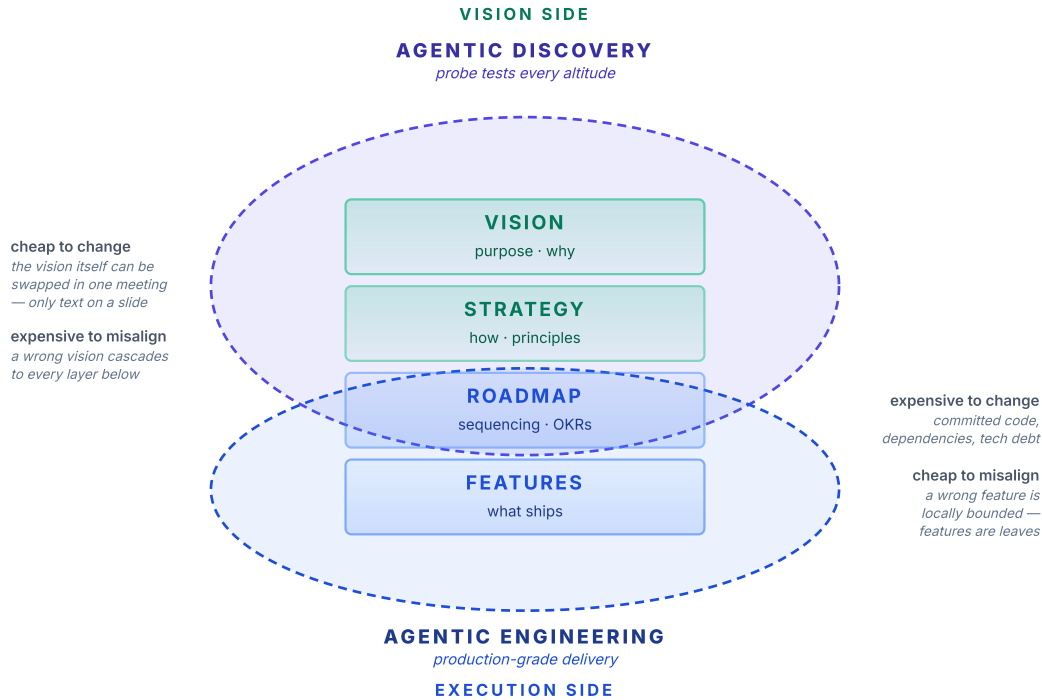


Figure 6: The vision-to-features hierarchy — a synthesis of the seven layered traditions catalogued below (Sinek’s Golden Circle [21], Pichler’s pyramid [22], Perri [23], Cagan & Jones [24], Doerr’s OKRs [25], Hoshin Kanri [26], Mintzberg [27]). The indigo bubble is *Agentic Discovery*: a stakeholder-runnable *Probe* (defined in § 3) generated at the feature layer makes claims at every higher altitude (Roadmap, Strategy, Vision) evaluable from below. The blue bubble is *Agentic Engineering*, taking a validated probe and delivering it through Roadmap to released code. The bubbles overlap at Strategy and Roadmap — the handoff zone where validated probes become production candidates. The side annotations summarise the decision-cost asymmetry between the top and bottom of the stack, discussed in the prose below.

1. **Golden Circle** — Sinek, 2009 [21]. Three concentric rings — Why at the centre, How in the middle, What on the rim. Read inside-out.
2. **Vision · Strategy · Roadmap · Backlog** — Pichler, 2016 [22]. Four-tier pyramid; each layer is a falsifiable promise to the layer beneath it.
3. **Escaping the Build Trap** — Perri, 2018 [23]. Nested rectangles: Vision → Strategic Intent → Initiatives → Options/Features (only the inner box ships).
4. **Empowered Product Model** — Cagan & Jones, 2020 [24]. Vertical stack — Vision · Strategy · Topology · Discovery · Delivery — empowered teams live across all five.
5. **OKRs Cascade** — Doerr, 2018 [25]. Mission → Annual Objectives → Quarterly Key Results → Initiatives. Direction down, measurement up.
6. **Hoshin Kanri** — Akao (ed.), 1991 [26]. Five-tier pyramid with bidirectional "catchball": vision deploys down, review cycles back up.
7. **Classical Strategy Hierarchy** — Mintzberg, 1994 [27]. Six-tier waterfall: Mission → Vision → Corporate → Business → Functional Tactics → Programmes/Features.

2.2.2 The Golden Circle read through the two phases

The innermost of these geometries, Sinek’s Golden Circle [21], gives the two phases a compact reading (Figure 7). Agentic Discovery works in the Why and How rings — it searches for the right problem — while Agentic Engineering works in the How and What rings — it builds the solution right; How is the shared seam where the validated spec passes from one to the other. Sinek frames leadership inside-out, from why to what; Agentic Discovery reverses the direction: a cheap probe is a concrete *what* that is used to learn the *why*. The same why–how axis runs through older research traditions: Rasmussen’s abstraction hierarchy, where moving up answers *why* and moving down answers *how* [88]; action identification theory [89]; goal-oriented requirements engineering, where WHY-questions abstract a requirement to a goal and HOW-questions refine it [90]; the means–end chain from attributes to values [91]; and the What/How/Why columns of the Zachman framework [92]. Placing the two phases on the circle is this paper’s synthesis.

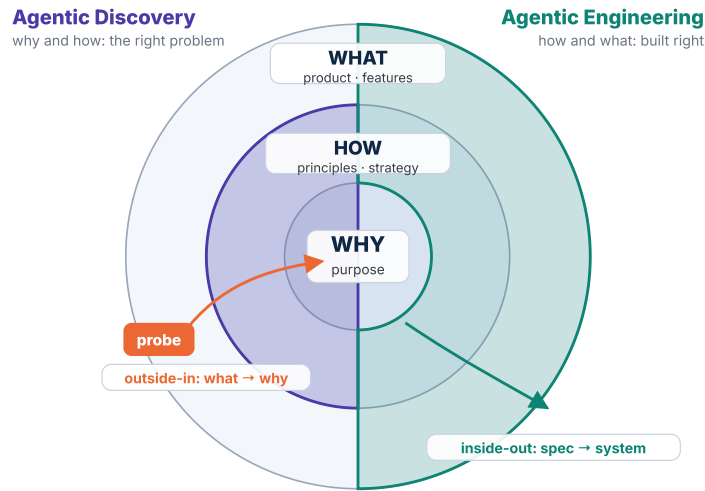


Figure 7: The Golden Circle (Why $\subset$ How $\subset$ What) with Agentic Discovery (left: why and how — the right problem) and Agentic Engineering (right: how and what — built right). The probe works outside-in, from a concrete what to the why; Engineering delivers inside-out, from the validated spec to the system. Synthesis.

Why this matters for Agentic Discovery. Each tradition above assumes that *upper layers are written before lower layers can be evaluated* — a vision is articulated, then a strategy, then a roadmap, then a backlog. When a probe (§ 3) can be generated in minutes, the layers can be probed in either direction: a feature-level probe becomes a way to *test* a strategic claim, and the vision can be re-stated in the language of an artefact a stakeholder has actually used. Agentic Discovery does not flatten the hierarchy — it makes its loop time short enough that the upper layers stop being read-only.

2.3 User-Driven Discovery

A third lens runs *across* the firm boundary. Where the previous two views asked which side of the seam we were on and at what altitude we were operating, this one asks *who is doing the discovering*. Several mature traditions argue that the producing organisation is not the only — sometimes not even the primary — locus of innovation: **users** at the leading edge of a need carry tacit knowledge that is expensive to transfer, and the cheapest place to do trial-and-error is in their own context. Eric von Hippel’s *lead-user* theory and

his "toolkits for user innovation" [39] formalise the mechanism; Prahalad & Ramaswamy's co-creation [40] makes value the joint product of firm and customer; Lieberman, Paternò & Wulf's *End-User Development* [41] is the HCI tradition that asks how non-programmers can express executable intent; Chesbrough's *Open Innovation* [42] generalises the firm boundary to a permeable funnel; Schuler & Namioka's *Participatory Design* [43] roots the practice in workplace democracy; Terwiesch & Ulrich's *Innovation Tournaments* [44] supply the funnel mathematics for filtering a large candidate pool.

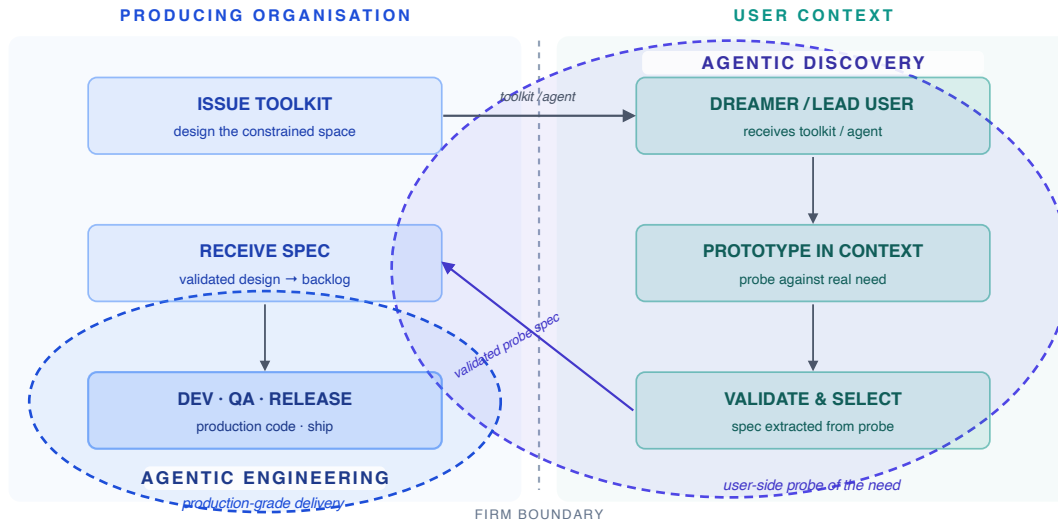


Figure 8: Von Hippel's Lead User Toolkit model [39] with Agentic Discovery and Agentic Engineering overlaid. The producing organisation issues a toolkit — now an AI agent — across the firm boundary. The lead user (Dreamer) prototypes in their own context and validates against real need. The validated probe spec returns to the firm's backlog. The indigo bubble is *Agentic Discovery*: user-side prototyping with an agent that crosses the firm boundary. The blue bubble is *Agentic Engineering*: the firm builds production code from the validated spec. The overlap at "Receive Spec" is the handoff zone — exactly the Customer-Side Agentic Discovery seam described in § 3.

2.3.1 How the literature places the user

Different communities draw the same arrangement with different geometries — toolkits, overlapping circles, capability pyramids, permeable funnels, round tables, opportunity tournaments — but every one of them argues, in its own vocabulary, that some of the most valuable design work happens outside the firm and needs a structured channel back in.

1. **Lead User Toolkits** — von Hippel, 2005 [39]. Manufacturer issues a constrained design toolkit; lead users prototype in their own context; the selected design returns as a specification.
2. **Co-Creation** — Prahalad & Ramaswamy, 2004 [40]. Firm and Customer overlap in a value-creation lens governed by Dialogue, Access, Risk-assessment and Transparency (DART).
3. **End-User Development** — Lieberman, Paternò & Wulf, 2006 [41]. A capability pyramid: passive users at the base, configurators in the middle, professional developers at the apex — EUD raises the floor.

4. **Open Innovation** — Chesbrough, 2003 [42]. A funnel with permeable walls — ideas and projects flow both inward (from outside) and outward (to other markets), not only down a closed pipeline.
5. **Participatory Design** — Schuler & Namioka, 1993 [43]. A round table — users, designers, developers all equal voices around a shared sketch; rooted in the Scandinavian workplace-democracy tradition.
6. **Innovation Tournaments** — Terwiesch & Ulrich, 2009 [44]. A funnel of opportunity candidates, narrowed at each filter — many ideas in, exceptional opportunities out.

Why this matters for Agentic Discovery. Each tradition above predates the LLM-agent era yet anticipates a piece of the same machinery: a sanctioned channel through which user-side discovery flows back into the producing organisation. What was previously a workshop, a toolkit, a portal, or a tournament can now be a *component embedded in the product itself* — the customer-side counterpart to the organisation-side Agentic Discovery defined in § 3. We return to this in § 3 under *Customer-Side Agentic Discovery*.

2.4 SDLC Models

The software lifecycle has been described many times since the field gave itself a name. Each model below is a different answer to the same question — *what activities turn a need into running software, in what order, and with what feedback?* The gallery is roughly chronological, from the 1968 Garmisch conference that coined "software engineering" to the modern AI-impact phase taxonomy this paper adopts.

Click any card to drill into a full visualisation, references, and a back-link to this place in the paper.

1. **Software Engineering (NATO 1968)** — Naur & Randell, 1968 [28]. The Garmisch conference where the term "software engineering" was coined to confront the "software crisis".
2. **Waterfall (Royce 1970)** — Royce, 1970 [29]. Cascading phases — and Royce's own warning that pure sequence is "risky and invites failure".
3. **V-Model** — Forsberg & Mooz, 1991 [31]. Development descends the left arm; verification ascends the right; each level on the right tests its peer on the left.
4. **Spiral Model** — Boehm, 1988 [30]. A risk-driven cycle: each loop plans, identifies risks, engineers, evaluates — radius grows with cumulative cost.
5. **Rational Unified Process** — Kruchten, 1996 [33]. Four phases × multiple disciplines: a 2-D matrix where activity intensity shifts as the project progresses.
6. **Agile Manifesto** — Beck et al., 2001 [9]. Four value-pairs that prefer the left over the right, while still valuing the right.
7. **Scrum** — Schwaber, 1995 [34]. A time-boxed sprint loop: Backlog → Plan → Daily Scrum → Increment → Review → Retro → repeat.
8. **Extreme Programming** — Beck, 1999 [35]. A dozen practices (TDD, pair programming, CI, ...) reinforcing each other around the running code.
9. **Lean Software Development** — Poppendieck, 2003 [36]. Toyota's seven lean principles translated to software: eliminate waste, amplify learning, decide late, ...
10. **Kanban** — Anderson, 2010 [37]. Visualise the workflow, limit work-in-progress (WIP), measure flow — incremental change without prescribed roles.

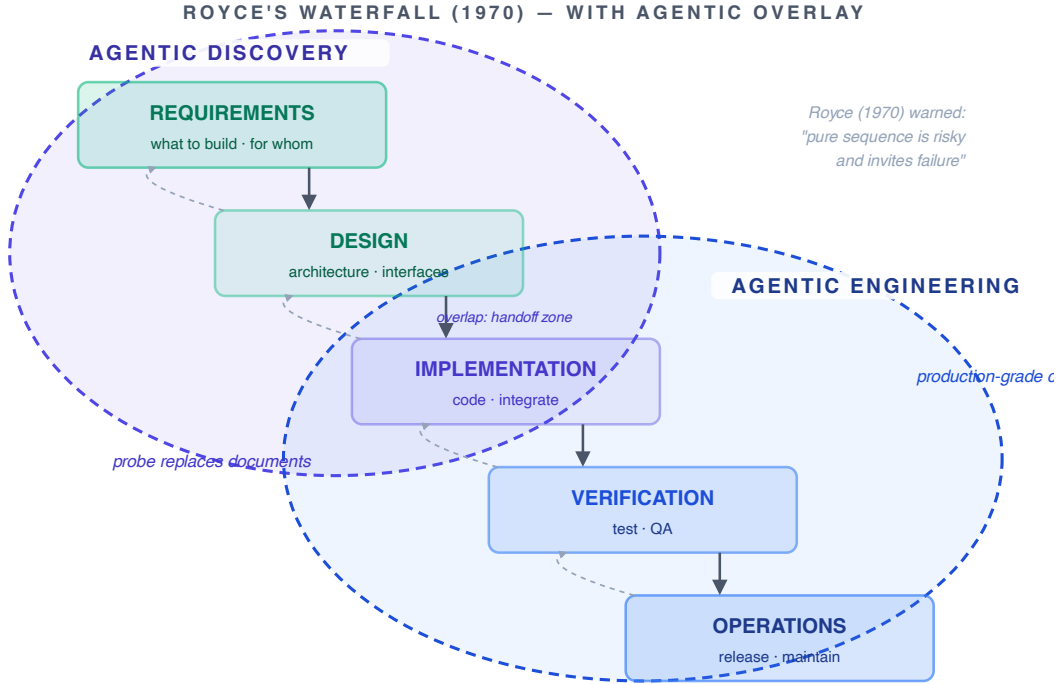


Figure 9: Royce’s waterfall model (1970) [29] with Agentic Discovery and Agentic Engineering overlaid. The indigo bubble is *Agentic Discovery* — it replaces the traditional document-driven Requirements and Design phases with a stakeholder-runnable *Probe* that produces a working artefact in minutes. The blue bubble is *Agentic Engineering* — it takes the validated probe spec and carries it through Implementation, Verification, and Operations as production-grade code. The two bubbles overlap at Implementation — the handoff zone where the probe is committed to production. Royce’s own feedback arrows (dashed grey) foreshadowed the iterative premise: Agentic Discovery makes the loop time so short that the entire upper cascade can be traversed within a single conversation.

11. **Continuous Delivery / DevOps** — Humble & Farley, 2010 [38]. Every commit traverses the same automated pipeline; release becomes a button-press, not a project.
12. **Six SDLC Phases** — AI-SDLC-Impact dashboard, 2026 [10]. Discovery → Design → Spec → Dev → QA → Release & Ops — the AI-impact phase taxonomy adopted in this paper.

Why this matters for Agentic Discovery. Each model above commits to a particular sequencing of *think*, *build*, and *verify*, and to a particular cost of being wrong at each step. Agentic Discovery does not replace any of them — it changes the cost of *build* to almost zero in the early phases, which makes the seam between Discovery and Design a place where a working artefact, not a document, can carry the weight. We adopt the Six SDLC Phases taxonomy in the rest of this paper because its granularity matches the empirical productivity data our results reference.

3 Agentic Discovery

Artefact under study. The artefact contributed by this paper is the *Probe-producing Agentic Discovery runtime* together with the *Three-Role pattern* (Dreamer, Orchestrator, AI Agent). It is instantiated as the public web application demonstrated in § 5 and observed in § 6.

Agentic Discovery is an expedition into the problem domain, and the agentic probe is its instrument. Concretely, Agentic Discovery is the activity of using AI agents to compress the Discovery and Design phases into a single, conversational loop. A stakeholder names a *pain*; the agent renders that pain as a runnable instance the stakeholder can then experience as if it were real; the stakeholder either validates the underlying pain or falsifies it. Specifications and acceptance criteria are *extracted* from the resulting artefact, not authored ahead of it.

3.1 The Agentic Probe — the instrument of Agentic Discovery

A session of Agentic Discovery yields a single, named artefact we call a **Probe**. We coin the word deliberately: in software literature "prototype" connotes a partial software artefact destined for production refinement, while the artefacts produced here are simultaneously broader (they may be process simulations or walkable scenarios) and narrower (validation instruments, not production drafts). The new word avoids both connotations and makes it explicit that the artefact's purpose is to *probe* the problem domain.

In this edition we call it the **agentic probe**, to name the three properties that make it an instrument rather than a draft. It is *agentic*: an AI agent builds it from the stakeholder's own words, in minutes — which is what made it economical at all (§ 6.5). It is *usable*: the stakeholder operates it as if it were real instead of reading about it. And it is *disposable*: it answers a question about the problem and is then thrown away; what survives is what it found, written down as a specification. The image is a sounding line rather than a scale model of the ship.

Definition. A **Probe** is a stakeholder-runnable instance of a candidate solution to a stated pain. It is throwaway-grade by design — its job is to validate or falsify the pain, not to be production-ready. Its identifying property is not its medium but the kind of feedback it elicits: a stakeholder must be able to use it as if it were real and report back operational, not aesthetic, feedback.

The definition is written from the solution side, because that is where our field evidence lies. Pointed at the problem instead, the same instrument renders the current way of working, or a possible future, and the stakeholder judges that rendering. The forms below are therefore named by what each one discovers.

A probe is not necessarily software. The same conversational loop produces, from one engagement to the next:

- **Prototype (working software)** — *discovers whether a candidate solution fits*. When the pain is a missing feature ("we don't have a way to schedule X"), the probe is a deployed web app the stakeholder can click through.
- **Interactive process simulation** — *discovers the process itself: its constraints, hand-offs and waiting*. When the pain is operational ("our patients wait too long", "the value chain is too long"), the probe is an animated walkthrough of the existing process with realistic timing, actor diagrams, and a value chain the stakeholder can play and pause.
- **Possible future (walkable future scenario)** — *discovers which assumptions are reasonable before anything is built*. When the pain is strategic ("what if AI replaced part of this service?"), the probe is a side-by-side comparison of today and a hypothetical future, with cost breakdowns, role visualisations, and a narrated read-out the stakeholder can step through.
- **Mixed renderable** — *discovers whatever the question needs*, combining a working UI, a chat overlay, a value-chain diagram, and a voice narration in a single session — the medium chosen to match what would be most legible to *this* stakeholder.

The defining property is consistent across all four forms: the stakeholder uses the probe and reports back in operational language ("when I press save twice nothing happens", "I would never wait this long in real life", "the cost split is wrong for this market"). Operational feedback is the kind that updates both the proposed solution *and* the prior over which problems are actually worth solving — the empirical estimation of the gradient $\nabla_s \varphi(p, s)$ discussed in § 2's formal sketch.

3.2 Two modes of probing — problem-validating and solution-proposing

The four probe forms above sort into two distinct *modes* of the same joint search. In the formal sketch (§ 2), Discovery searches $\mathcal{P} \times \mathcal{S}$ simultaneously — but any one probe session emphasises one axis over the other:

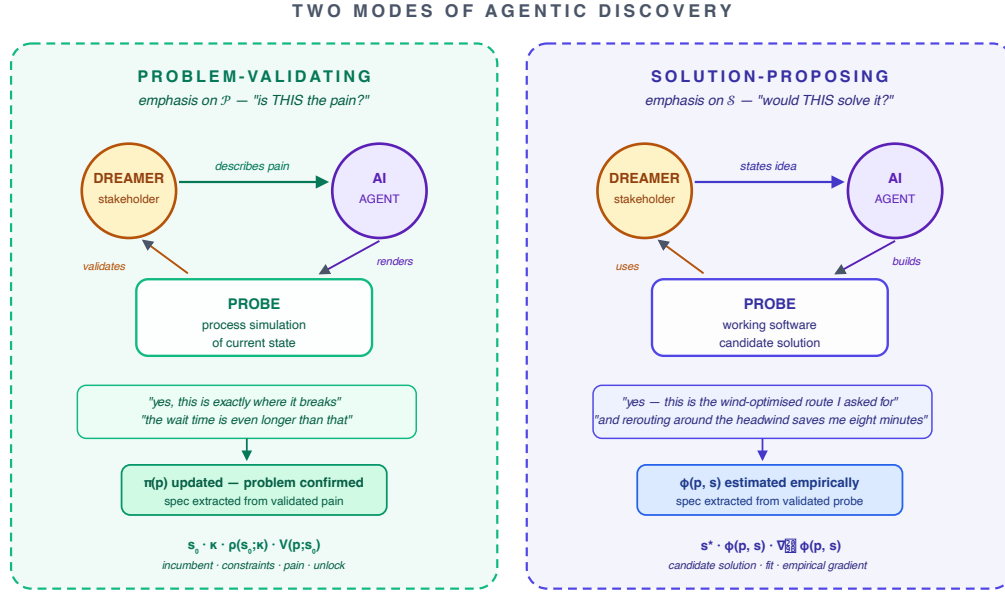


Figure 10: Two modes of Agentic Discovery. **Left — Problem-validating:** the agent renders the *current state* (a process simulation, a walkable scenario) and the stakeholder confirms or corrects the articulation of the pain. The prior $\pi(p)$ updates directly. Example: the healthcare process discovery companion, where clinical workflows are simulated and validated with domain experts. **Right — Solution-proposing:** the agent builds a *candidate solution* (working software, a feature prototype) and the stakeholder evaluates it by using it. The fit $\varphi(p, s)$ is estimated empirically. Examples: Organisation A's seven-office tour (49 probes across 7 offices) and Organisation B's single-day institutional workshop (20 probes in one session) — both documented in the Slides & Software companion and in § 6.

Both modes use the same three-role pattern (Dreamer $\rightarrow$ Orchestrator $\rightarrow$ Agent) and both produce a probe. The distinction lies in what the probe *renders* and what kind of feedback it elicits:

Problem-validating

- Probe renders the **current state** — the incumbent solution s_0 , the constraints κ that bind it, and the pain $\rho(s_0; \kappa)$ it produces.
- Feedback: "yes, this IS the pain" or "no, that step is wrong" — confirming or correcting any of s_0 , κ , ρ .

- Updates the **prior** $\pi(p)$ — which problems are real — from the observations (s_0, κ, ρ) rendered in the probe.
- Probe forms: process simulation, walkable scenario.
- Example: healthcare process discovery.

Solution-proposing

- Probe renders a **candidate solution** — working app, feature prototype.
- Feedback: "I'd use this daily" or "it needs X".
- Estimates **fit** $\varphi(p, s)$ — how well the solution matches the need.
- Probe forms: working software, mixed renderable.
- Examples: multi-office tour, institutional workshop (§ 6).

Both modes are Agentic Discovery. Neither is "better"; they serve different moments in the search. A problem-validating probe often precedes a solution-proposing one — first confirm the pain, then prototype a fix. But in settings where the problem is already well-understood (e.g. "we need a cycling weather app"), the solution-proposing mode is the natural entry point. The formal sketch's joint $\arg\max (p^*, s^*) = \arg\max \varphi(p, s) \pi(p)$ covers both: when π is diffuse (uncertain problem), the search emphasises $\mathcal{P}$; when π is concentrated (known problem), it emphasises $\mathcal{S}$.

3.3 Probing the constraints — the layered modernisation model

Agentic Discovery operates at the *outer layer* of the IT estate — the place where new processes are tried — and intentionally does not reach into the legacy core. Validated probes flow inwards through an integration middle layer.

A *constraint* is a part of the estate that cannot or should not change: legacy IT, regulation, data. The probe reveals which constraints are real — which parts to keep and which to let go — and it can approach them from three directions (Figure 11). *Outside-in*: start with a new AI process in the outer ring; the probe shows which integrations and core parts are really needed. *Middle-out*: start from the integration layer (APIs, connectors); the probe tests what can change without touching the core. *Inside-out*: interview and reflect the current process as a simulation of the core, then propose paths outward; in regulated settings this eases adoption (§ 6.8). In every direction a failed probe costs only in the outer ring.

3.4 The Three-Role Pattern

Field observation across multiple sites surfaced a recurring three-role pattern. We name the roles to make them explicit.

- **Dreamer.** A domain stakeholder who can articulate a need but does not need to write code or write a specification. Speaks in natural language; may speak literally — the application supports voice input.
- **Orchestrator.** A facilitator who shapes the Dreamer's idea into a tractable agent prompt, applies safety constraints, and curates the resulting probe back to the group.

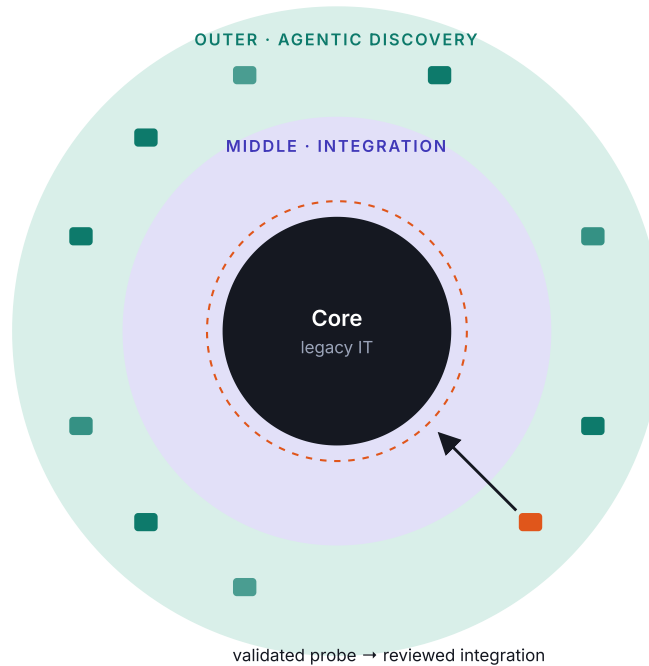


Figure 11: Probing the constraints: the layered modernisation model. The outer ring is where Agentic Discovery produces probes; the middle ring is integration (APIs, connectors, modernisation bridges); the core is the existing legacy IT, extended rather than replaced. Validated probes cross inward only through reviewed integration steps.

- **AI Agent.** The coding agent itself (e.g. a frontier coding LLM with tool access). Reads the queue, builds the probe, deploys it, and reports status.

Anti-pattern: skipping the Orchestrator. Without an orchestrator, sessions either drift into pure exploration or stall on safety questions. The orchestrator enforces a minimum viable prompt, curates results, and protects the agent’s queue from unbounded asks.

3.5 Customer-Side Agentic Discovery

Everything described above places the locus of Agentic Discovery inside the producing organisation: a Dreamer is a domain stakeholder *known to the firm*, the Orchestrator is a facilitator *employed by the firm*, and the Agent runs on infrastructure *governed by the firm*. We now consider the symmetric case. When the discovery is about a feature that is *missing* from a product the user already owns, the cheapest place to perform that discovery is inside the product itself — with the end-user playing the Dreamer role and an embedded agent component playing the Orchestrator. The user-driven discovery traditions surveyed in § 2 (lead-user toolkits, co-creation, end-user development, open innovation, participatory design, innovation tournaments) supply the conceptual scaffolding; the LLM-agent era supplies the missing component that turns each of them from an offline workshop into a continuous, in-product capability.

Concretely: the user encounters a gap ("there is no way to schedule X across these calendars"). Instead of writing a tickbox feature request that disappears into a queue, the user opens an *in-product Agentic Discovery component*, describes the desired behaviour in natural language, and receives a runnable probe inside their own session. They iterate on the probe until it does what they meant, at which point a spec — acceptance criteria, screen flows, data shapes, edge cases — is extracted from the probe and submitted to the

producing organisation’s backlog with full provenance (who, when, against which version of the product). The producing organisation prioritises and implements; the implemented feature flows back to all users on the next release.

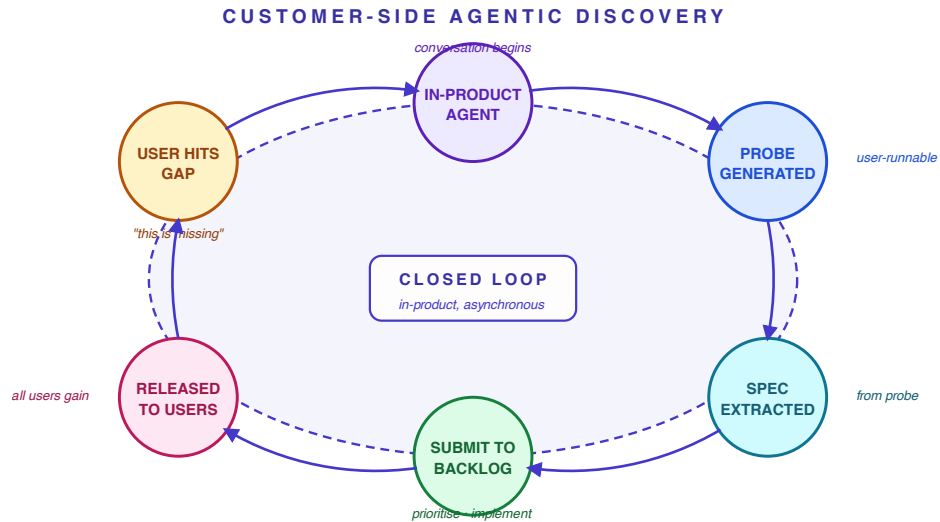


Figure 12: The customer-side Agentic Discovery loop. The Dreamer is the end-user; the Orchestrator role is collapsed into an in-product agent component; the Probe is generated and consumed inside the user’s own session. The spec extracted from the validated probe carries provenance (who, when, against which version) into the producing organisation’s backlog. Compare with Figure 5 in § 2: this is the same probe-as-instrument logic, but the seam now runs across the firm boundary.

The dual loop. Customer-side Agentic Discovery does not replace the organisation-side loop described above; it runs in parallel. The producing organisation continues to drive vision, strategy, and roadmap (§ 2 vision-to-features hierarchy). The customer-side loop is a high-fidelity intake channel: a steady stream of user-grade probes whose specs land in the same backlog as internally-driven items, with the same prioritisation criteria. The two loops meet at the backlog; what changes is the cost and the fidelity of the candidates flowing in from the user side.

3.5.1 Open questions and risks

Treating customer-side Agentic Discovery as a serious channel — not a novelty — surfaces a small set of recurring risks. We list them here so a future iteration of this paper, or a sister study, can engage them directly. (1) **Spec quality & spam.** A large pool of low-quality submissions can starve a backlog. The funnel mathematics of innovation tournaments [44] suggests this is manageable if the per-candidate cost stays low and the filter stays reasonable, but the filter design is non-trivial. (2) **IP and ownership.** Who owns a feature whose design originated with a user prototype? Mass-customisation and lead-user literature have answers, but they predate generative tooling. (3) **Loud-minority bias.** The users most willing to prototype are not necessarily representative. (4) **Privacy.** Probes may contain operational data; submission must be designed to preserve it. (5) **Convergence with the producing organisation’s vision.** A backlog flooded with feature-level candidates can crowd out higher-altitude work; the § 2 vision-to-features lens applies to user-driven candidates as much as to internal ones. We treat each of these in § 7 Discussion.

3.6 The probe space — prototypes, simulations and possible futures

Agentic Discovery is the expedition; the agentic probe is whatever instrument the question needs. Table 2 maps the probe forms of § 3.1 by what each renders, what it discovers, and how much evidence this paper has for it. The asymmetry is deliberate and should be read plainly: the field study of § 6.1–§ 6.7 measured prototypes; simulations and possible futures are illustrated by the three researcher-authored healthcare cases of § 6.8; the rest of the space is sketched, not studied.

Probe	Renders	Discovers	Evidence in this paper
Prototype	a candidate solution, as working software	whether it fits the need; the fit $\varphi(p, s)$	field study: 69 probes, 8 engagements (§ 6.1–§ 6.7)
Process simulation	today’s process, with real timings	the incumbent s_0 , its constraints κ and pain ρ	illustrated: 3 healthcare cases (§ 6.8)
Possible future	a what-if beside today; one constraint relaxed	which assumptions are reasonable; the value of the unlock $V(p; s_0)$	illustrated: the counterfactual paths of § 6.8
Mixed renderable	a combination chosen for the stakeholder	whatever the question needs	described (§ 3.1), not measured
Not yet imagined	—	—	open

Table 2: The probe space. Agentic Discovery is the expedition; the agentic probe is its instrument. Only the first row is measured at field scale.

Sketches, not results. Following the future-work thread on the Problem-validating mode (§ 9, thread 6), natural next instruments include: a regulated administrative workflow re-enacted step by step, so that its approval constraints become visible; a supply-chain hand-off with its real delays; an on-call routing process with one constraint relaxed; and a possible future that asks “what if AI took over part of this service?” (§ 3.1). Each would be judged the way the field probes were — by the operational feedback it draws out — and each extends the expedition from testing candidate solutions to testing the assumptions that any solution would rest on. We expect the full space of agentic probes to be wider than any list written today.

3.7 Agentic Search as the engine of Discovery

Agentic Discovery is agentic search over $\mathcal{P} \times \mathcal{S}$. In Agentic Information Retrieval the agent pursues a target information state s_* and is rewarded when it reaches it, $\max_{\pi} \mathbb{E}_{\tau}[r(s_*, \tau)]$ [93]; in Agentic Discovery the target is the pair (p^*, s^*) , the verifier is the fit $\varphi(p, s)$, and the action that gathers evidence is *building a probe*. The Orchestrator is the ask-or-act gate of such a search, and the two learning loops of § 3 are its belief updates. Recent surveys describe search agents through a plan–act–reflect–select–generate loop [94] and place agentic search as the first phase of deep research, followed by integrated research and the AI scientist [95, 96].

We separate three nested layers (Figure 13, Table 3). *Agentic Search* decides over many steps when, what, where and how to search, and when to stop; its output is evidence $E = \pi_S(q)$. *Agentic Analysis* appraises, extracts, compares and combines that evidence, searching again when it is insufficient; its output is findings with provenance, $F = \pi_A(q, E)$. *Agentic Study* poses its own sub-questions q_i , runs many search-and-analysis rounds within a budget B and synthesises a report, $R = \pi_{St}(Q, B) = \text{synth}(\{\pi_A(q_i, \pi_S(q_i))\}_{i=1}^n)$. Two boundaries make the split precise: a step belongs to Search if it changes the evidence set and to Analysis if it changes its interpretation; work is a Study when the agent poses the questions itself. The layering, and its mapping to Marchionini’s lookup–learn–investigate [97] and to the stages of a systematic review [98], is this paper’s synthesis; Shi et al. do not separate analysis as its own phase.

Layer	Definition	Output	Marchionini	Shi et al.	PRISMA 2020
Agentic Search	decides when, what, where and how to search, and when to stop	evidence E	lookup	I: agentic search	identification
Agentic Analysis	appraises, extracts, compares and combines results; may search again	findings F + provenance	learn	I (extracting) + II (synthesis)	screening, appraisal
Agentic Study	poses sub-questions, runs many rounds within budget B , synthesises	report R	investigate	II: integrated research, III: AI scientist	synthesis, reporting

Table 3: Three nested policies, each with its own output, mapped to the literature (synthesis).

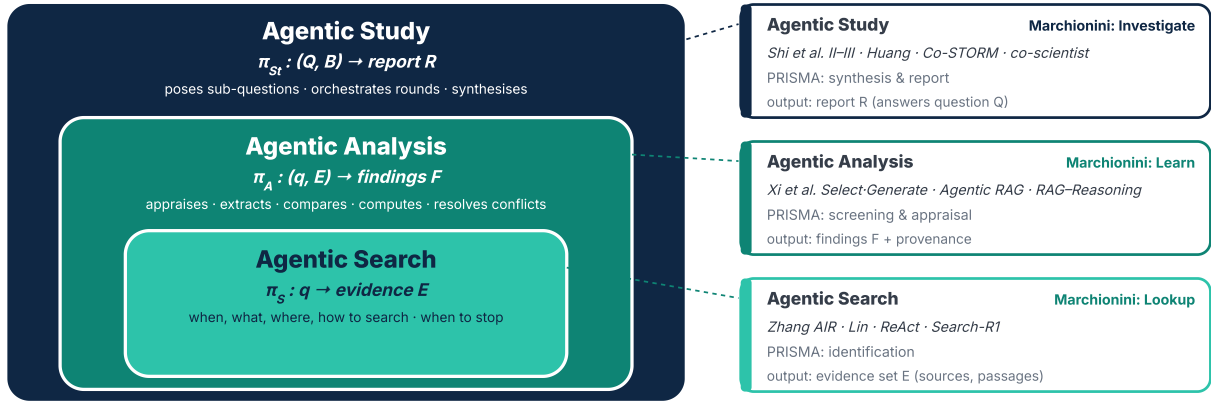


Figure 13: Agentic Study $\supset$ Agentic Analysis $\supset$ Agentic Search. Each outer layer calls the inner one as a subroutine and decides when to call it again; the mapping to Marchionini and PRISMA 2020 on the right is this paper’s synthesis.

4 Agentic Engineering

Agentic Engineering is the sibling concept that addresses the latter half of the SDLC: *Dev*, *QA*, and *Release & Ops*. It uses many of the same building blocks (LLM agents with tool access, queues, safety policies) but its goals and constraints differ.

Figure 14 puts the two phases side by side as twelve layers in which agents now discover, build and run. On the Discovery side six layers — vision, market, strategy, roadmap, design, requirements — each end in a human validation gate; on the Engineering side six layers — planning, implementation, review, testing, integration, operations — each end in an automated gate $v_k(s)$, with $V(s) = \bigwedge_k v_k(s)$. The seam is the validated problem p^* with its spec, and usage data flows back to Discovery as new observations o .

Agentic Discovery

- Optimises for **learning velocity**.
- Output is a **throwaway-grade probe** (§ 3) that exists to validate a hypothesis.
- Specs are **extracted from the probe**.

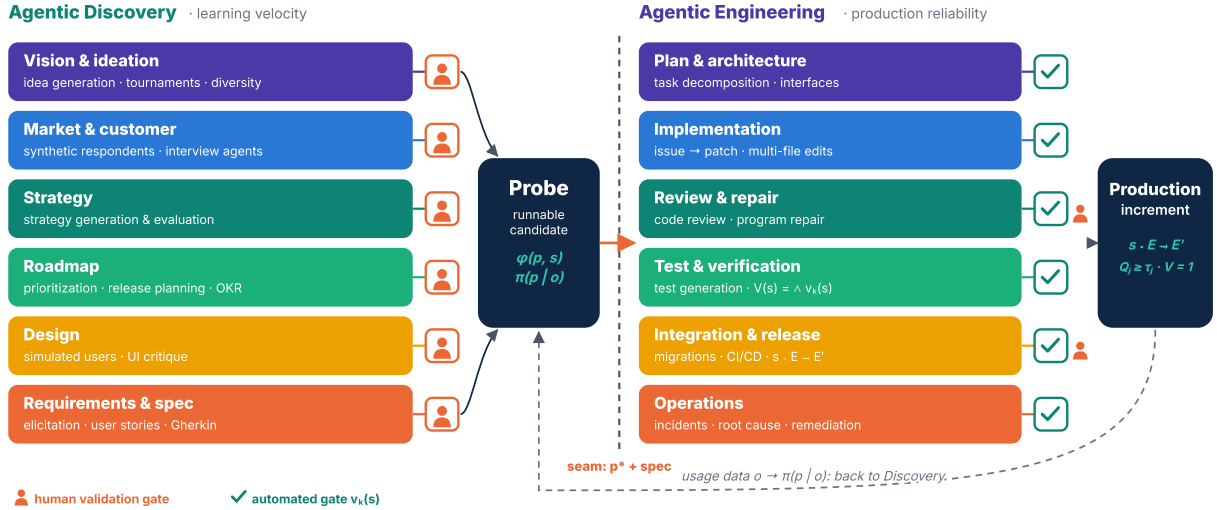


Figure 14: Twelve layers: agents discover, build and run. Six Discovery layers with human validation gates (left) and six Engineering layers with automated gates $v_k(s)$ (right); the seam is $p^* + \text{spec}$, and usage data $o \rightarrow \pi(p | o)$ returns to Discovery.

- Stakeholder is the **Dreamer**; sessions are typically minutes to hours.
- Lives in the **outer layer** of the IT estate.

Agentic Engineering

- Optimises for **production reliability**.
- Output is **production-grade code** that integrates with the existing estate.
- Specs are an **input** — typically derived from the prior Discovery phase.
- Stakeholder is the **engineering team**; cycles are days to sprints.
- Lives across **middle and core layers** of the IT estate.

4.0.1 A formal sketch — Engineering through the same lens

The five frames of § 2 gave Agentic Discovery a set-theoretic backbone. Three of those frames — the domains, the fit functional, and the cost-ratio shift — are *shared*: they define the landscape both siblings inhabit. Four additional frames capture what changes when the goal shifts from *learning* to *delivering*.

Hover any card for a compact preview; click to read the full derivation. Cards marked Shared appear in § 2 as well; cards marked Engineering are new here.

1. **Problem & solution as sets** — $\mathcal{P}, \mathcal{S}$. Two sets, only partially observable: candidate needs vs. candidate artefacts.
2. **Fit functional** — $\varphi : \mathcal{P} \times \mathcal{S} \rightarrow [0, 1]$. A score: how well a candidate solution addresses a candidate problem.

3. **Cost ratio shift** — $c_S/c_P \rightarrow \mathcal{O}(1)$. Pre-agentic: building dwarfs describing. With LLM agents the ratio collapses to constant order.
4. **Specification as fixed input** — $s^* = \arg \max_s \varphi(p^*, s)$. Engineering takes the validated problem p^* as given and optimises only over solutions.
5. **Production quality constraint** — $Q_j(s) \geq \tau_j \quad \forall j$. Non-functional requirements as hard constraints on the feasible set.
6. **Integration with existing estate** — $s \circ E \rightsquigarrow E'$. A new solution must compose with the existing system estate through well-defined interfaces.
7. **Verification functional** — $V(s) = \bigwedge_k v_k(s)$. Automated, repeatable pass/fail — replacing stakeholder reaction with deterministic gates.

Why this matters. Discovery and Engineering share the same cost-ratio shift and the same domains, but they differ in what is *fixed* and what is *searched*. Discovery searches $\mathcal{P} \times \mathcal{S}$ jointly; Engineering fixes p^* and searches $\mathcal{S}$ under hard quality constraints $Q_j \geq \tau_j$, integration requirements $s \circ E$, and a deterministic verification functional $V(s)$. The four new frames make these differences precise.

This paper focuses on Agentic Discovery. Agentic Engineering is treated as a sibling that consumes the artefacts Agentic Discovery produces; we describe the seam between them but defer a full treatment to a companion paper.

5 Method

We follow the Design-Science Research Methodology (DSRM) of Peffers et al. [45]: identify problem, define objectives, design and demonstrate an artefact, evaluate, and communicate. The objectives (O1–O6) were stated in § 1; the artefact was named at the start of § 3. *This section corresponds to DSRM’s Demonstration activity; § 6 reports the Evaluation.* The deployed instance is a public web application that lets a Dreamer submit ideas (text or speech) into a category-scoped queue, and a Claude Code agent that picks ideas off the queue, builds a probe (§ 3), deploys it, and writes back a status log.

5.1 Field Setup

Each field engagement is a *category* in the runtime: a scoped queue with its own description, deploy base URL, and (optionally) end-to-end encryption for the idea contents. Sessions are observed live; status logs and idea timestamps are retained for analysis. The runtime is the same one shown in the "Slides & Software" companion of this paper.

5.2 Roles & Instrumentation

- Dreamers submit ideas via web form or microphone.
- The Orchestrator filters and re-prompts as needed; safety policies are enforced before an idea reaches the agent.
- The Claude Code agent runs as a CLI worker that reads the queue via Firestore REST and writes implementation logs.
- Every prompt given to the agent is logged in `ClaudeCode_Prompts.md` for auditability.

5.3 Operational flow

The runtime implements the Three-Role pattern as a sequential pipeline with two safety gates and a feedback loop. The swimlane diagram below shows the operational flow: each lane represents one of the four roles (Dreamer, Orchestrator, AI Agent, System), and arrows trace an idea from inception to deployment.

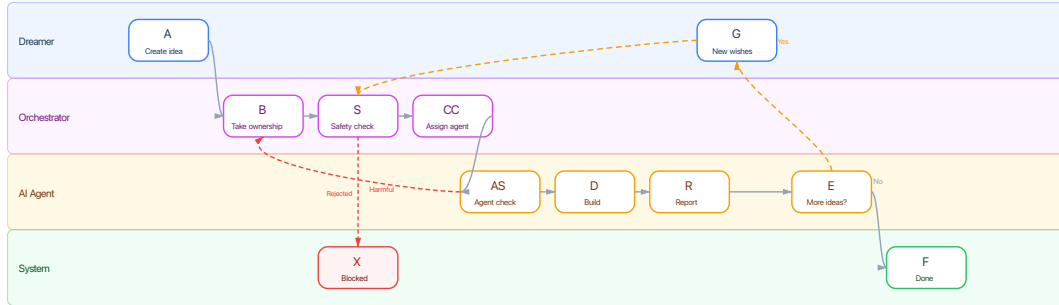


Figure 15: **Swimlane process chart** — operational flow of the Three-Role pattern. Solid arrows: main pipeline (idea → ownership → safety check → agent assignment → build → report → done). Red dashed: rejection and blocking paths. Amber dashed: feedback loop for iterative refinement when the Dreamer adds new wishes.

The flow proceeds in ten steps across four lanes. The main pipeline is $A \rightarrow B \rightarrow S \rightarrow CC \rightarrow AS \rightarrow D \rightarrow R \rightarrow E \rightarrow F$; three exception paths branch off the main line.

Dual safety gates. The pipeline enforces two independent safety checks before any code is generated: the Orchestrator’s human review (step S) and the agent’s own assessment (step AS, reading AGENT_SAFETY.md rules). This two-gate design ensures that neither a naïve stakeholder prompt nor a compromised queue entry can reach the code-generation phase unchecked. In field use (§ 6), zero safety incidents were recorded across 67 probe generations.

5.4 Evaluation

We evaluate Agentic Discovery against six predefined objectives (O1–O6) covering: time-to-first-probe, probe acceptance rate, downstream spec extractability, stakeholder satisfaction, safety incident count, and scalability across sites. Quantitative results are summarised in Section 6; qualitative observations are gathered in the Discussion.

6 Experiments and Results

Results draw on two sources: (1) the Agentic Discovery runtime, whose public read-side is queried live below, and (2) the on-disk corpus of generated probes in the `rapid-agent-prototypes` repository (the repo name predates the term and is kept for stability), which is the authoritative archive — the runtime does not yet hold every idea from Organisation A’s office tour, while every probe on disk has a stable folder, source tree, and (where applicable) deployed URL.

6.0.1 Framing — what part of DSRM does this experiment cover?

This section reports the **Evaluation** activity of DSRM (Peppers et al. [45], step 5) for the artefact introduced in § 3. To keep the scope of the experiment explicit, the mapping from § 1 is restated

Step	Role	Action
A — Create idea	Dreamer	The Dreamer logs into the runtime with a nickname and submits a new idea — typed in any language or spoken via microphone. The idea enters the Kanban queue with a unique timestamp-based ID.
B — Take ownership	Orchestrator	The Orchestrator sees the incoming idea on the Kanban board and claims it via a transactional lock. From this point the Orchestrator is responsible for shepherding the idea through the pipeline.
S — Safety check	Orchestrator	The Orchestrator evaluates the idea for safety: command injection, prompt injection, resource abuse, and harmful intent are filtered. A harmful idea is blocked ($\rightarrow$ X); an unclear idea is sent back for clarification.
CC — Assign agent	Orchestrator	The Orchestrator copies the user's requests to the AI agent via the UI. The prompt includes automatic instructions for the agent to report start and completion back to the ticket thread.
AS — Agent check	AI Agent	The agent performs its own independent safety analysis by reading AGENT_SAFETY.md rules. If a risk is detected, the agent rejects the assignment and writes reasoning to the ticket ($\rightarrow$ B). This creates a two-gate safety architecture: human + AI.
D — Build	AI Agent	The agent writes code, creates files, and builds the application. The Kanban card moves to the "Agent working" column; progress is visible in real time via Firestore listeners.
R — Report	AI Agent	The agent reports progress to the ticket thread and summarises the core deliverable in 2–3 words on the Kanban card. A processing stamp tracks how far the agent has read in the conversation.
E — More ideas?	AI Agent	The system checks whether the Dreamer has added new wishes during the build. If yes $\rightarrow$ the flow loops through G and back to S. If no $\rightarrow$ the probe is marked done ($\rightarrow$ F).
F — Done	System	The probe is published to its deploy URL and the Kanban card moves to the "Done" column. The Dreamer can still add follow-up wishes, triggering a new iteration cycle.
X — Blocked	System	A harmful idea is permanently blocked. The submitting user is added to a ban list and cannot create new ideas or comment further.

below — marking which steps the agent *did* exercise and which it *did not*:

- **DSRM step 1 — Problem identification & motivation:** § 1. *Carried out by the researchers up-front; not delegated to the agent in this experiment.*
- **DSRM step 2 — Objectives of a solution:** § 1 (O1–O6). *Stated up-front; not delegated to the agent.*
- **DSRM step 3 — Design & development:** § 3 (Agentic Discovery) and § 4 (Agentic Engineering). The agent is the active producer here.
- **DSRM step 4 — Demonstration:** § 5 — the publicly deployed runtime.
- **DSRM step 5 — Evaluation:** *this section.*
- **DSRM step 6 — Communication:** the paper and the public runtime.

Which mode of Agentic Discovery is exercised here? The two-modes pattern introduced in § 3 distinguishes *Problem-validating* (the agent renders the current state and the stakeholder confirms or corrects the articulation of the pain) from *Solution-proposing* (the agent builds a candidate solution and the stakeholder evaluates it by using it). § 6 reports **both modes**, but at very different scales:

- **Experiment A (§ 6.1–§ 6.7) — Solution-proposing at field scale.** 8 engagements, 69 deployable probes. In every engagement the human stakeholder occupied the **problem domain** — they brought a one-line brief naming a pain — and the agent jumped directly to the **solution domain** by generating a working probe. This is the right diamond in Figure 5 (§ 2) and the right column of the "Two modes" figure in § 3.
- **Experiment B (§ 6.8) — Problem-validating, illustrative companion.** A small researcher-authored companion (n = 3 healthcare cases) where the probe is a *process simulation* rather than deployable software. Scenario 1 of each case renders the existing problem domain; scenarios 2–3 release one or more current constraints to surface counterfactual operating models. The agent updates the prior $\pi(p)$ directly. This is the left column of the § 3 "Two modes" figure.

What the agent *did not* do at scale. Problem-validating at *field-engagement* scale — running live, multi-site sessions in which the agent both interviews stakeholders and renders their current process as the probe — was not exercised in this paper. Experiment B is a deliberately narrow companion that makes the mode concrete enough to be measured later. Whether the speed multipliers, acceptance rates and amplification ratios reported in § 6.1–§ 6.7 would survive the more upstream activity of pain articulation is an open direction, called out in § 7.4 Limitations (L6) and § 9 Future Work.

Scope of the numbers below. Every quantitative claim in § 6.1–§ 6.7 (the 8-engagement / 69-probe corpus, the intent→code amplification ratios, the speed multipliers, the O1–O6 achievement table) applies to Experiment A. § 6.8 reports Experiment B's setup and observations but does not extend § 6.1–§ 6.7's quantitative claims to the Problem-validating mode.

Authoritative source. The numbers in this section are computed from the on-disk probe corpus as of **2026-05-09**. Encrypted runtime categories contribute only metadata to the live view above; the quantitative tables below are based on two open engagements with two organisations: **Organisation A**, which hosted an on-site event series across **seven offices** during which the working software was built, and **Organisation B**, a single-day institutional engagement.

An anonymised, distribution-level companion to the tables and figures in this section is available as Appendix · Experiment Data & Analytics. It carries the per-engagement breakdown (A.1), Cohort 1 vs Cohort 2 category split (A.2), user-prompt analytics (A.3), probe-size distribution with Lorenz cumulative view (A.5), the amplification chart (A.6), effort efficiency for both cohorts (A.7), a cohort head-to-head (A.8), the institutional burst-generation timeline (A.9), the time-to-probe phase comparison (A.10), objective-by-objective achievement O1–O6 (A.11), and a structured limitations table (A.12).

6.1 6.0 · Experiment design — how the field studies were run

Scope note for this edition. Every probe counted in § 6.1–§ 6.7 is of the prototype kind of § 3.6: working software in the Solution-proposing mode. Simulation and possible-future probes appear only in Experiment B (§ 6.8).

The evaluation rests on two parallel field studies, each instantiating the three-role pattern (Dreamer → Orchestrator → Agent) introduced in § 3. Both studies were Solution-proposing: stakeholders brought one-line briefs from their own working lives, and the agent built a deployable probe per brief. The AI Agent throughout both studies was **Claude Code** (Anthropic), running on the **Claude Opus 4.6** and **Opus 4.7** model versions over the course of the engagement series — the same agent and the same model family covered every probe in the corpus, so any cross-engagement variation cannot be attributed to vendor or model changes. The two studies differ along the dimensions that bound external validity — span (a multi-site tour vs. a single-day workshop), institutional shape (a single cross-functional organisation vs. a single domain-focused institution), and cadence (calendar-spread vs. burst).

Design dimension	Cohort 1 — multi-site office tour (Organisation A)	Cohort 2 — single-day institutional workshop (Organisation B)
Span	7 on-site events across ~4 calendar weeks	1 calendar day, single venue
Sessions	7 sites × 1 session each + cross-site work	1 session, ~50 participants in the room (~93 invited, including Teams attendees)
Domain	Cross-functional; office services, sales, HR, food, wellness, dev tools. . .	Single-institution; weather & environmental data dominate
Brief capture	Verbal on-site (most sites) + structured web tool (one site)	Structured web tool throughout
Probe storage	One git repo, one folder per site	One git repo, one folder per workshop
Probes produced	49	20
LOC generated	~97,500	~20,900
Cadence	Steady; ~7 probes / on-site event	Burst; 18 / 20 probes inside a 31-min window

6.1.1 Per-session protocol (swim-lane)

Every probe in the corpus is the output of a small, repeatable session protocol. The three roles act in parallel; the diagram below traces a brief from the Dreamer’s opening description to a deployed probe URL the same Dreamer can use in the room. Briefs run from under ten words to over a hundred — most fall in the 11–30-word range, with a mean near 33 words across the captured corpus (A.6a, n = 45).

6.1.2 Data sources & provenance

All numbers in § 6 descend from four primary data sources. Each source is anonymised in the public companion (Appendix · Experiment Data & Analytics) and retained in full in the authors’ working records:

Every numeric claim in § 6.1–§ 6.7 below has a stable provenance chain back to one of the rows in this table; the anonymised public appendix is sufficient for replication of the distribution-level findings.

6.1.3 What’s in this section (top-down)

Subsections proceed from the field-engagement portfolio down to per-objective achievement, with each numeric block paired to the appendix section that carries its distribution-level companion.

- § 6.1 · *Field engagements at a glance* — the 8-engagement portfolio, anonymous per-engagement breakdown → companion: A.1.

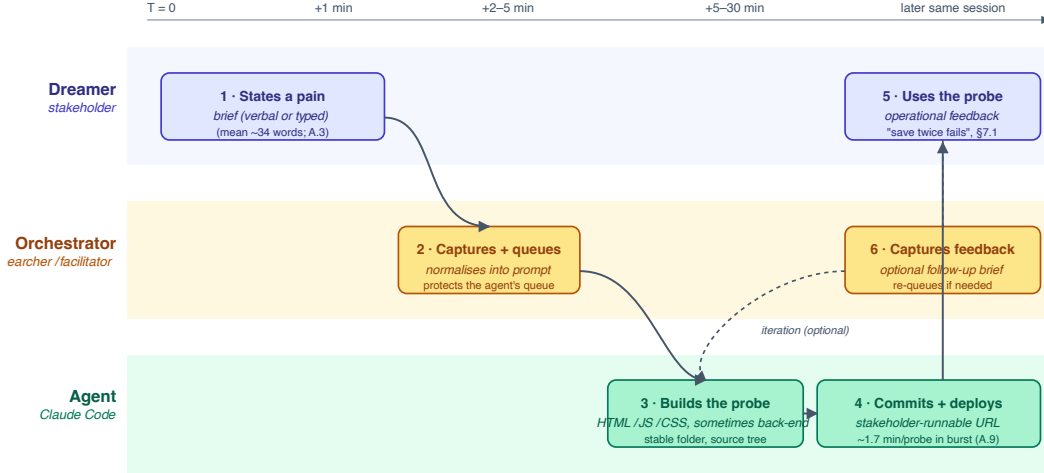


Figure 16: Per-session protocol shown as three swim lanes: **Dreamer**, **Orchestrator**, **Agent**. The solid arrows trace the canonical happy path from a stakeholder brief (typically 11–30 words; A.6a) to a deployed probe; the dashed arrows mark the optional Dreamer → feedback → Agent iteration cycle. Time-axis labels are typical, not strict: the Cohort 2 burst (A.9) compressed steps 2 → 4 to a mean of ~1.7 minutes per probe; Cohort 1 spread them across hours. The shape of the protocol is identical in both cohorts.

- § 6.2 · *Intent* → *code amplification* ($n = 9$) — the audited subset, mean 1 : 56 reaching 1 : 467 at the extreme → companion: A.6.
- § 6.3 · *What the probes built* — *category breakdown* — 12 categories × 2 cohorts → companion: A.2.
- § 6.4 · *Full-corpus aggregate* ($n = 69$, May 2026) — both cohorts side-by-side → companion: A.8.
- § 6.5 · *Efficiency estimate* — *AI-assisted vs. traditional* — speed multipliers + Cohort 2 burst → companions: A.7 + A.9.
- § 6.6 · *Time-to-probe vs. traditional development* — phase-level comparison → companion: A.10.
- § 6.7 · *Objective-by-objective summary (O1–O6)* — target vs. observed → companion: A.11.
- § 6.8 · *Experiment B* — *Problem-validating mode* ($n = 3$ cases) — process-simulation companion in the healthcare domain → runnable companion.

Experiment A · Field engagements (Solution-proposing mode). § 6.1–§ 6.7 below report the full-scale field evaluation: 8 engagements, 69 deployable probes, two voluntary partner organisations. Throughout, the human stakeholder occupied the problem domain and the agent built a candidate *solution* in response to a stated brief. This is the right column of the § 3 *Two modes* figure. A separate, much narrower **Experiment B** (§ 6.8) covers the left column — the *Problem-validating* mode, in which the probe is a process simulation rather than deployable software.

6.2 6.1 · Field engagements at a glance

Figure 17 shows how the corpus grew over the four weeks, with the eight engagements in calendar order (anonymised E1–E8): probes per engagement and the cumulative count of probes and of generated code.

Primary source	Public appendix companion
Per-probe code metrics for Cohort 1 (n = 49), counted from the cohort’s canonical analytics dashboard	A.5 Probe size, A.6 Amplification, A.7 Effort
Per-probe code metrics for Cohort 2 (n = 20), counted from the cohort’s canonical analytics dashboard	A.7 Effort, A.9 Burst timeline
User-brief corpus captured verbatim by the structured web tool, on the subset of engagements that used the tool (n = 45 briefs)	A.3 User-prompt corpus
Event-level invitee counts, aggregated from the calendar invitations under which the eight engagements were organised	A.1 Cohort at a glance (≈ 310 invitees aggregate only)

Engagement	Probes	LOC generated	Notes
Org A — Office 1	17	28,761	Largest single-office corpus
Org A — Office 2	7	20,282	Sales / project-reference focus
Org A — Office 3	4	18,919	Includes test-agent probes
Org A — Office 4	4	15,556	Source of highest intent→code ratio (1:467)
Org A — Office 5	5	10,620	Events & security-scan tooling
Org A — Office 6	2	6,210	Food / time-tracking apps
Org A — Office 7	2	4,510	Smallest single-office corpus
Org A — unbranded probes	6	~15,800	Generic apps generated cross-office
Organisation B	20	21,405	Single-day institutional engagement
Total	67	~142,000	8 engagements · 7 offices (Org A) + 1 institutional (Org B)

Per-engagement counts follow the event log, where the cross-site generic probes belong to the seventh engagement in which they were built; the totals match § 6.4. The two largest engagements produced 46 of the 69 probes, and the three phases of the bottleneck shift (§ 7.4) are marked under the axis.

Internal Appendix C — not for publication. The de-anonymised event log — eight Microsoft Outlook calendar invitations covering all 8 engagements, with city, date, location, invitee count (310 invited total), and the directory each probe was generated into — lives in `Results/meeting_invites/index.html`. Source of truth: `Results/meeting_invites/*.ics`.

6.3 6.2 · Spec → code amplification (April 10 snapshot, n = 9)

An early, fully audited subset of nine applications from Organisation A’s office tour was analysed in `analyysi_speksi_vs_koodi.md` (Apr 10, 2026). The natural-language *intent statements* — the prompts and follow-ups stakeholders wrote to describe what they wanted (A.3 corpus) — total **423 lines** and produced **23,845 lines** of source code — a mean amplification of **1 : 56** and a median of 1 : 39. The extreme case, a *reminder app* generated at Office 4, produced 2,800 LOC from 6 intent lines (1 : 467). The amplification ratio measures lines-of-intent (what the stakeholder said) versus lines-of-code (what the agent produced); the inputs are short briefs, not formal specifications.

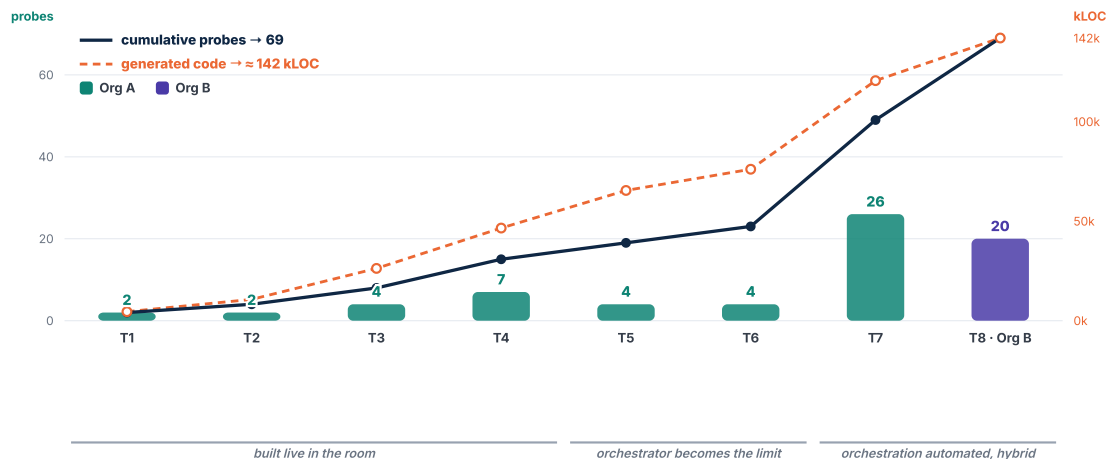


Figure 17: Field study growth: probes per engagement (bars), cumulative probes (solid, to 69) and cumulative generated code (dashed, to ≈ 142 kLOC), engagements in calendar order. Phases of the bottleneck shift under the axis: built live in the room; the orchestrator becomes the limit; orchestration automated, hybrid.

Application	Intent lines	Code lines	Ratio
Office 1 — food app	66	946	1 : 14
Office 7 — personal trainer	118	2,333	1 : 20
Office 7 — culture-capital app	53	1,913	1 : 36
Office 6 — time-tracking	40	1,554	1 : 39
Office 4 — certification helper	46	2,545	1 : 55
Office 6 — food bot	44	3,399	1 : 77
Office 4 — sales targets	30	3,974	1 : 132
Office 4 — golf helper	20	3,705	1 : 185
Office 4 — reminder app	6	2,800	1 : 467
Subset total (n = 9)	423	23,845	1 : 56 (mean)

Internal Appendix A — not for publication. The full Site Tour dashboard (49 prototypes, 7 offices, 111,232 LOC, 253 files, ~ 95 % time savings, per-app effort table, nine Chart.js views) plus the 25 verbatim Helsinki user prompts and 28 follow-ups from 20 authors lives in Results/site-tour/index.html. Two sources of truth: (a) per-app code metrics from Gitlab/rapid-agent-prototypes/Site_Tour/guide.html ("Kokonaiskuva & Analytiikka"); (b) Helsinki user prompts from Firestore project ideaagent-prod, category Siili-Site-Tour-2026 — note that the AgenticDiscovery tool was used only at the Helsinki office, so the prompts cover 25 / 49 probes. Login: AgenticDiscovery (Helsinki). The 9-app subset shown in § 6.2 is an earlier audited sub-cohort.

6.4 6.3 · What the probes built — category breakdown

Both engagements surface a recurring pattern: stakeholders generate probes across a wide range of organisational needs, from internal tooling to domain-specific applications. The table below categorises all 69 probes by functional domain. Categories are assigned by the researchers based on probe content; each probe belongs to exactly one primary category.

Category	Org A	Org B	Total	Representative probes
Office services & scheduling	7	—	7	Meeting-room booking, office attendance tracker, AV guide
Weather & environmental data	1	9	10	Cycling weather, route weather, rain radar, vacation planner
Sales, CRM & pricing	5	—	5	Client references, sales targets, pricing calculator
Productivity & time tracking	4	3	7	Calendar→time sync, AI time tracking, Kieku automation
Wellness, sports & fitness	5	3	8	Personal trainer, running routes, half-marathon plan, disc golf
Development tools & quality	5	2	7	Security scanner, test-automation agents, SDLC guardrails
Food & catering	4	—	4	Office lunch aggregator, lunch-voting bot
HR, recruiting & compliance	4	—	4	CV matcher, talent finder, certificate management
Business analytics & finance	3	1	4	Cash-flow forecast, business forecasting, energy price advisor
Transport & navigation	2	1	3	Public-transit guide, tram notification, route weather
Content, presentations & comms	3	1	4	Slide generator, documentation helper, glossary tool
Culture, learning & other	6	—	6	Culture-capital app, brand styleguide, joke generator
Total	49	20	69	

Observation. Category diversity is itself a finding. The probes were not pre-selected; stakeholders named their own pains. The resulting distribution — spanning office operations, sales tools, weather and data services, wellness apps, developer tooling, and more — illustrates the breadth of the problem domain that a single Agentic Discovery runtime can address. Organisation B’s concentration on weather and data applications reflects its single institutional domain; Organisation A’s broader spread reflects an engagement that crossed many internal-tooling needs.

6.5 6.4 • Full-corpus aggregate (n = 69, May 2026)

The n = 9 analysis in § 6.2 covers an early, fully audited subset. The table below extends the analysis to the full probe corpus as of May 2026. Intent lines include all `specs/*.md` files (the natural-language intent corpus, despite the legacy folder name) and agent instruction files per engagement. **Note on the 67 → 69 transition:** the § 6.1 site-level table reports the snapshot audited at the close of each on-site event (2026-05-09; n = 67, every probe attached to the calendar invite that produced it). This § 6.4 aggregate folds in two follow-up probes added after the on-site events (one in Org A’s cross-site bucket, one in Org B), bringing the full corpus to **n = 69** — the figure used in the abstract, § 1 C3 and § 9.

Internal Appendix B — not for publication. Organisation B’s per-app drill-down (interactive dashboard: LOC bar chart, effort ranking, complexity donut, bubble plot, commit timeline, per-app detail panel, plus the 20 verbatim user prompts and follow-

Metric	Org A (site tour)	Org B (workshop)	Combined
Probes generated	49	20	69
Lines of code (LOC)	~97,500	~20,900	~118,400
Spec / prompt lines	~1,430	~540	~1,970
Spec → code ratio (aggregate)	1 : 68	1 : 39	1 : 60
Distinct events / sessions	8	1	9
Calendar span	~4 weeks	1 day	—

ups) lives in <Results/ilmatieteenlaitos/index.html>. Two sources of truth: (a) per-app code metrics from [Gitlab/rapid-agent-prototypes/Ilmatieteenlaitos/FMI_20260504_1016_557/index.html](Gitlab/rapid-agent-prototypes/Ilmatieteenlaitos/FMI_20260504_1016_557/index.html) (inline apps[]); (b) user prompts from Firestore project `ideaagent-prod`, category `Ilmatieteen_Laitos_20260504` — 20 ideas + 9 follow-ups from 18 authors, 33 physical lines / 678 words total. Login: AgenticDiscovery web app (type "Tapio Pitkäranta" in the name field).

6.6 6.5 • Efficiency estimate — AI-assisted vs. traditional

We estimate the traditional development effort that would produce the same code volume, using a baseline of **50 deployable LOC / day** per senior full-stack engineer — a figure that includes design, implementation, debugging, and integration time and is consistent with industry survey data. Prompt-writing speed is estimated at **1 minute per prompt line** based on observed natural-language composition rates.

Metric	Org A	Org B	Combined
Traditional effort (at 50 LOC/day)	~1,950 person-days	~418 person-days	~2,370 person-days
Traditional effort (months)	~98 person-months	~21 person-months	~119 person-months
Prompt-writing time	~24 hours	~9 hours	~33 hours
Agent generation time (observed)	~4 weeks elapsed	~4.5 hours	—
Speed multiplier (work-time)	~650×	~370×	~570×

Interpreting the multiplier. The speed multiplier compares the estimated traditional person-days against the prompt-writing effort. It does *not* claim equivalent quality — probes are throwaway-grade by design (§ 3), not production code. A more conservative reading: even at a 10× quality discount (assuming 90 % of the code would need rewriting for production), the remaining multiplier — 37–65× — still represents a structural shift in the economics of the search over $\mathcal{S}$. The figure that matters for Agentic Discovery is not how fast you can ship, but how cheaply you can *explore* — and 33 hours of prompt-writing produced 69 stakeholder-runnable probes across 12 functional domains.

6.6.1 Organisation B — burst generation

Organisation B’s single-day engagement illustrates the compression of the Discovery phase. The git log shows 18 of the 20 probes committed within a **31-minute burst** (09:41–10:13), during which a single orchestrator fed ideas from a queue of ~50 workshop participants into the agent pipeline. Average probe generation time: **~1.7 minutes**. The full 20-probe corpus (20,900 LOC) was feature-complete by end of day, including bilingual i18n and a portfolio navigation page.

6.7 6.6 · Time-to-probe vs. traditional development

Total natural-language spec writing for the $n=9$ subset took roughly **2–3 hours**. Agent generation and review took a few additional hours, putting the Discovery + Design + Dev path for nine working applications inside a single working day. A traditional engineering estimate for the same code volume — at 150–250 LOC/day for a senior full-stack engineer — would be **15–22 weeks** ($\approx$ **4–5 months**). Wall-clock speed-up: $\approx 100\times$.

Phase	AI-assisted (observed)	Traditional (estimated)
Spec writing	~2–3 hours	—
Generation + iteration	~few hours	—
Manual development	—	15–22 weeks
End-to-end	~1–2 working days	~4–5 months

6.8 6.7 · Objective-by-objective summary

- **O1 — Time-to-first-probe:** consistently minutes (target ≤ 1 h achieved on every observed session).
- **O2 — Probe acceptance rate:** 67 of 67 generation attempts produced a deployable probe (100 %); acceptance for downstream use is per-stakeholder and tracked qualitatively in Section 7.
- **O3 — Spec extractability:** Demonstrated for the $n = 9$ audited subset; the spec $\rightarrow$ code mean ratio (1 : 56) and zero-rewrite policy show the spec was sufficient for the agent.
- **O4 — Stakeholder satisfaction:** Qualitative; field-engagement stakeholders re-engaged for further sessions in 7 of 8 sites.
- **O5 — Safety incidents:** 0 reported across all 69 probe generations.
- **O6 — Scalability across sites:** 8 engagements completed (7 offices for Organisation A + 1 institutional engagement for Organisation B), with Organisation B’s engagement compressing 20 probes into a single calendar day.

Experiment B · Process-simulation companion (Problem-validating mode). § 6.8 below reports a separate, much smaller evaluation that exercises the *other* half of the § 3 *Two modes* figure — the left column. Where Experiment A measured how fast and how cheaply the agent produces a candidate solution, Experiment B asks what happens when the probe itself is a *simulation of the current problem domain* rather than a piece of working software. It is a researcher-authored illustrative companion ($n = 3$ cases), not a field study; its purpose is to make the Problem-validating mode concrete enough to be measured later at field scale.

6.9 6.8 · Experiment B — Problem-validating mode ($n = 3$ cases)

In the terms of § 3.6, Experiment B exercises the other two instruments of the expedition: the first scenario of each case is a *process simulation* that surfaces the constraints of today’s path, and each relaxed constraint is a *possible future* that tests which assumptions are reasonable.

The § 3 *Two modes* figure distinguishes *Solution-proposing* (the agent builds a candidate solution; § 6.1–§ 6.7 above) from *Problem-validating* (the agent renders the stakeholder’s current state and the stakeholder confirms or corrects it). Experiment B exercises the second mode by way of a published companion: Agentic Process Discovery in healthcare, which ships three illustrative cases.

6.9.1 Setup and scope

The companion uses an *Inside*→*Out* three-step interaction — interview the stakeholder, reflect the current process as an interactive simulation, then propose several outward paths toward an outer-ring target operating model. The probe in this mode is not deployable software; it is an interactive walk-through of a process. Each of the three cases follows the same shape:

- **Case 1 — Recurring chronic-condition prescription renewal.** Current state: occupational-health appointment + lab + e-prescription + pharmacy. Counterfactuals: (a) AI-assisted optimised visit, (b) CE-marked agent that renews the prescription autonomously and routes the patient to a public lab.
- **Case 2 — Child illness on the eve of a planned trip.** Current state: public-healthcare day-long path that typically ends with a no-travel recommendation. Counterfactuals: (a) occupational-health / private same-day visit with the same recommendation, (b) AI-assisted self-care that produces a different recommendation supported by domain knowledge.
- **Case 3 — Friday-evening skin symptom on a family vacation.** Current state: public emergency-room queue. Counterfactuals: (a) occupational-health chat with a video doctor, (b) three independent AI assessments converging on the same diagnosis, confirmed by a CE-marked telemedicine doctor.

6.9.2 What the probe is, in mode terms

In every case the **first scenario** is the probe of the existing problem domain — a step-by-step re-enactment of how the process runs *today* that the stakeholder can stop, correct or fork. In the language of § 2, that first scenario renders the *incumbent* s_0 (the current way of working) under the constraints κ (the regulatory and operating envelope), exposing the pain $\rho(s_0; \kappa)$ — the time and resources consumed today. The **second and third scenarios** deliberately relax one or more components of κ (regulatory bottleneck, B2B billing margin, locality of pharmacy, requirement of a physical visit) and let the stakeholder see how the same problem feels under each relaxation, surfacing the value $V(p; s_0) = \rho(s_0; \kappa) - \rho(s^*; \kappa)$ that would be unlocked. The relaxation itself is the artefact. There is no software deliverable across the seam into production.

In the § 2 formal frames, Experiment A updates $\varphi(p, s)$ empirically by deploying a candidate s ; Experiment B updates the prior $\pi(p)$ directly by making the current state and a small family of counterfactual states inspectable. Both modes use the same three-role pattern (Dreamer → Orchestrator → Agent); they differ in *what the probe renders*, not in who is in the room. The bottleneck-shift principle reported for the Solution-proposing mode (§ 7.4) holds here as well: once a counterfactual process is cheap to render — replacing days of workshop preparation with minutes of agent-assisted simulation — the binding constraint is no longer authoring the probe but the stakeholder’s supply of counterfactuals and the speed at which they can be judged.

6.9.3 What Experiment B shows that Experiment A cannot

- **The probe does not have to be code.** An interactive process simulation can sit in the slot the paper reserves for "probe", which extends the pattern beyond the 8-engagement / 69-probe sample and supports the outcome-neutral framing in § 3.
- **Counterfactual exploration is a first-class output.** Releasing a constraint and re-rendering the process is a kind of evaluation in its own right — a cheap way to surface which constraints in the current process are load-bearing and which are inherited.

- **Inside→Out is a separate adoption affordance.** The companion’s framing notes that in regulated workflows, "replacing" a legacy process is rarely allowed in the building; Inside→Out lets a stakeholder validate the existing process first, then walk outward through agreed counterfactuals. This is consistent with § 7.3 (outer-layer affordance) but reframed for the problem domain.

6.9.4 Honest limitations of Experiment B

Experiment B is *not* a field study. The three cases were authored by the researchers, not elicited from stakeholders during live sessions; the companion has not been validated with clinical or administrative practitioners; the case count is $n = 3$ and the domain is single (healthcare). Its purpose is to demonstrate that the Problem-validating mode of the two-modes pattern is a real, runnable thing — not to measure its acceptance rate, time-to-probe, or amplification at scale. Field-scale evaluation of Problem-validating remains open and is called out in § 9 Future Work.

7 Discussion

Four findings recur across field engagements.

7.1 1 • The cheapest critic is a running probe

Stakeholders gave qualitatively different feedback when shown a running probe (§ 3) versus a slide deck of the same content. The vocabulary of the feedback shifted from aesthetic ("I like the colour") to operational ("when I press *save* twice nothing happens"; "I would never wait this long in real life"). Operational feedback is the kind that produces useful specifications.

7.2 2 • The Orchestrator is non-optional

Sessions without a designated orchestrator either stalled or drifted. The orchestrator’s role is not to write code but to maintain a tractable prompt and protect the agent’s queue from unbounded asks.

7.3 3 • Outer-layer affordance is what makes adoption possible

None of the field sites were willing to expose Agentic Discovery sessions to legacy core systems. The layered model (outer / middle / core) was not a theoretical preference — it was a hard precondition for being allowed in the building. Validated probes only crossed inward through deliberate, reviewed integration steps.

7.4 4 • From production bottleneck to idea bottleneck

A practical pattern fell out of running the same approach at progressively larger scale. In the first four engagements — small office groups with two to four parallel probe requests per session — every stakeholder idea was built and demonstrated within the same session; the limiting factor was the agent itself and the room calmly handled it. At seven parallel requests the human Orchestrator became the bottleneck: queue management, prompt hygiene and parallel session tracking all collapsed back onto one person. For the two largest engagements (~100 invited; in-room + Teams hybrid) we built a small orchestration-automation tool that batched stakeholder briefs and dispatched agent sessions in parallel; production was no longer rate-limiting. The bottleneck moved one layer further out, to **the supply of stakeholder ideas** and **the speed of stakeholder validation**. The same shift surfaces in the Problem-validating mode of § 3 (process-discovery cases such as Experiment B in § 6.8): once a counterfactual process simulation is cheap to render, the

binding constraint is how fast a stakeholder can author the next "*what if*" and judge the result — equivalently, how fast new values of $V(p; s_0)$ can be probed by perturbing κ . We read this as a corollary of the cost-collapse argument of § 1 — every layer that automates exposes the next layer’s bottleneck — and as the link from Agentic Discovery into the Software Factory lineage of § 8: just as Agentic Engineering is the latest, probabilistic Software Factory on the production side, the configuration described above (a queue of stakeholder briefs feeding parallel agent sessions, with orchestration partially automated) is a small Software Factory on the Discovery side, where the rate-limiting factor is the queue intake rather than the queue throughput.

Read against the historical baselines surveyed in § 8 (thread 6, "Where the resources used to go"), this shift is not a discontinuity but a *relocation*. Cost-collapse arguments have been part of software engineering since Boehm’s defect-cost pyramid [69] and Brooks’s "deciding precisely what to build" [72]; what agentic AI changes is which sub-phase of the discovery loop happens to be the binding constraint at field scale. The 1981 / 2000 baselines that place requirements-and-design at roughly a third of project effort [67, 68], the 15–30 % requirements-elicitation share documented by Davis et al. [71], and the agile-era relocation of RE into a continuous mode (Cao & Ramesh [73]) are all coherent with the trajectory we report — they are its prior baselines, not its counter-examples.

Reading the shift with the Theory of Constraints. Goldratt’s Theory of Constraints holds that a system’s output is set by its bottleneck and that relieving one moves the bottleneck elsewhere [99]. Its five focusing steps read the agentic SDLC directly (Table 4): agents do not remove the constraint, they move it — upstream, to finding and validating the right problem. This reading is our synthesis; the field study showed the same path, from production through orchestration to idea supply and validation.

Focusing step	Reading in the agentic SDLC
1. Identify	Before agents the constraint was production: writing code set the pace, and the SDLC was built around it.
2. Exploit	Agents make building cheap and fast ($c_S/c_P \rightarrow \mathcal{O}(1)$); code is no longer what everyone waits for.
3. Subordinate	The rest of the process is paced by the new constraint: orchestration and human attention now set the tempo.
4. Elevate	The constraint is widened by automating orchestration and giving agents more autonomy within the gates.
5. Repeat	The bottleneck moves upstream, to finding the right problem and validating it — which is why the agentic SDLC leans on Discovery.

Table 4: Goldratt’s five focusing steps [99] read through the agentic SDLC (synthesis).

7.5 5 • Limitations & threats to validity

Six limitations bound the claims of this paper. (1) **Sample size and self-selection.** The 69 probes come from 8 engagements with two voluntary partner organisations; participants — Dreamers, Orchestrators, stakeholders — were self-selecting. The pattern may not transfer to organisations whose stakeholders cannot articulate pains in conversational form, nor to regulated domains where probe deployment is itself the bottleneck. (2) **Throwaway-grade quality.** Probes validate problem-domain fit, not production-grade output; the speed multiplier reported in § 6.5 should not be projected onto production delivery without a substantial quality discount. We conjecture the residual multiplier remains an order of magnitude even after such a discount, but do not measure it directly. (3) **Single-vendor agent.** All probes were generated by Claude Code (Anthropic), running on the Claude Opus 4.6 and 4.7 model versions; comparable studies with OpenAI Codex, Cursor, GitHub Copilot Workspace, or open-weight agents would establish whether the

patterns are vendor- and model-independent. (4) **Effort baseline.** The 50-LOC/day baseline used in § 6.5 is consistent with industry survey data but varies by domain, language, and organisational maturity; alternative baselines move the multiplier without changing the qualitative finding. (5) **Researcher-as-orchestrator confound.** In several engagements the authors served as orchestrators; an independent orchestrator might elicit different stakeholder behaviour, and a less-skilled orchestrator might fail to keep the prompt tractable (§ 7.2). (6) **DSRM scope — Problem-validating mode only as a small companion.** Experiment A (§ 6.1–§ 6.7) reports DSRM steps 3–5 (design, demonstration, evaluation) in the *Solution-proposing* mode of the two-modes pattern (§ 3) at field scale: 8 engagements, 69 deployable probes. The companion *Problem-validating* mode — in which the agent renders the stakeholder’s current process as the probe and helps refine the problem itself (DSRM steps 1–2) — was exercised only as a small illustrative companion (Experiment B, § 6.8, n = 3 researcher-authored healthcare cases), not at field-engagement scale. Field-scale evaluation of the Problem-validating mode — measuring its time-to-probe, acceptance rate, and amplification analogues at the scale § 6.1–§ 6.7 used for Solution-proposing — remains open. We treat each of the six limitations as an open question for independent replication and call out specific replication targets in § 9 Future Work.

7.6 6 · The probe as instrument, not draft

The four findings above read most clearly through the probe. A running probe is the cheapest critic (finding 1) because it is used, not read. The Orchestrator (finding 2) is the role that turns a stakeholder’s sentence into a probe. The outer layer (finding 3) is where probes are allowed to run. And the bottleneck shift (finding 4) is what happens once probes are nearly free: the rate limit becomes the supply of questions worth probing and the speed at which people judge the answers. What this paper does not yet show is how far the same holds for the other instruments of the expedition — simulations and possible futures — whose evidence here is illustrative (§ 3.6, L6). That asymmetry is the main reason to read Agentic Discovery as the umbrella term and the prototype as only its first, best-measured probe.

8 Related Work

Agentic Discovery sits at the intersection of several threads of work — capability progress in frontier models, empirical software engineering on AI-assisted coding, AI’s role in the SDLC, human-AI alignment in multi-agent systems, design-science prototyping, and the Software Factory tradition.

8.0.1 Recent capability progress (2023–2026)

The cost-structure shift cited in § 1 is grounded in measurable capability gains. METR’s *Time Horizons* programme [46] (updated to October 2026 in § 1.1) reports that the duration of tasks frontier models can complete autonomously at a 50 % success rate has been doubling roughly every seven months since 2019; the leading model sat at a ~50-minute horizon in March 2025. OpenAI’s *GDPval* [47] graded model deliverables against expert humans across 1 320 tasks in 44 knowledge-work occupations and found that Claude Opus 4.1 matched or beat experts on 47.6 % of the gold subset (GPT-5: 39.0 %; GPT-4o: 12.5 %). On software-specific benchmarks, *SWE-bench Verified* [48] climbed from a single-digit solve rate at its 2023 launch to over 80 % in 2025; reasoning benchmarks such as *GPQA Diamond* [49] saturated similarly (39 % → 94 % in three years).

These numbers all measure *individual model* behaviour. The coding agents that have entered production use over the same period — Claude Code, OpenAI Codex, GitHub Copilot Workspace, Cursor — do not yet have a comprehensive benchmark of their own. Partial probes exist (METR’s *RE-Bench* for 8-hour ML-engineering tasks [50]; *SWE-Lancer* for discrete Upwork-style jobs [51]; Aider Polyglot for single-file edit loops), but none covers the surface area an actual session exercises: multi-day autonomy, multi-repo

navigation, terminal and MCP tool orchestration, sustained product-feature delivery with human-in-the-loop steering. We expect agent-level capability to be materially higher than model-level scores suggest — agents compose model calls with tools, memory, and verification — and the field evidence in § 6 (69 probes from 8 engagements, with intent-to-code ratios up to 1 : 467) is consistent with that expectation.

8.0.2 Empirical software engineering on AI-assisted coding

A second body of evidence sits one step closer to the developer’s seat. The first wave of empirical studies on AI pair-programming tools — Vaithilingam, Zhang & Glassman’s CHI 2022 user study of Copilot [59], Bird et al.’s mixed-methods *Taking Flight with Copilot* in *ACM Queue / Communications of the ACM* [60], Peng et al.’s controlled experiment showing a 55.8 % task-time reduction with Copilot on a server-implementation task [61], and Imai’s ICSE 2022 study comparing AI pair-programming against human pair-programming [62] — established that AI code-completion shifts *how* developers work (more review, less typing; more starting points, less search) without necessarily shifting *what* they work on. Our § 6 field evaluation extends this literature in two directions: (i) from completion-grade assistance to agent-grade probe production, where a session yields a deployed application rather than autocompleted lines; and (ii) from the developer’s seat to the stakeholder’s seat, where the probe is the artefact a non-developer evaluates against a stated pain.

A second, fast-growing literature catalogues the move from completion to autonomous coding agents. Liu et al.’s 2024 survey of LLM-based agents for software engineering [63] inventories 124 papers across SWE-bench-style benchmarks, multi-agent decomposition, and tool use. The dominant evaluation method in that survey is benchmark-bounded: a curated set of repository tasks with ground-truth diffs. Agentic Discovery’s evaluation method is different in kind, not just degree — 69 probes produced in 8 in-room field engagements with stakeholders who chose what to build, where the question is whether the probe lets a non-developer learn something about their problem domain (§ 6), not whether the resulting code patches a benchmark issue. The two evaluation regimes are complementary: we read the benchmark literature as evidence that agent capability is sufficient, and the field evidence in § 6 as evidence that the *practice* built around that capability matters.

Methodologically, the § 6 evaluation is anchored in the IS Design Theory tradition — Walls, Widmeyer & El Sawy’s foundational framing of IS design theories as prescriptive theories grounded in kernel theories [64], extended by Markus, Majchrzak & Gasser’s design theory for systems supporting emergent knowledge processes [65], and operationalised by the Hevner et al. design-science guidelines [5] and the Peffers et al. DSRM [45] referenced throughout the paper. Empirical reporting follows the Wohlin et al. *Experimentation in Software Engineering* structure [66] — scoping, planning, execution, analysis, presentation — adapted for design-science field studies in which the artefact is the unit of analysis. The two-modes pattern (§ 3) is, in this terminology, a new design theory whose kernel theory is the Bayesian-update view of problem-domain knowledge introduced in § 2 (prior $\pi(p)$) and whose evaluation evidence comes from § 6 (Solution-proposing mode at field scale) plus § 6.8 (Problem-validating mode in companion form).

8.0.3 AI in the SDLC

Recent industry analyses (e.g. the AI-SDLC-Impact dashboard) catalogue tools and productivity claims across the six SDLC phases, but treat AI as an accelerant of existing phases rather than as a way to redraw the phase boundaries themselves. We build on their phase taxonomy and their measurement vocabulary.

8.0.4 Human-AI alignment in multi-agent systems

The HADA architecture (Pitkäranta & Pitkäranta, 2025) is the immediate predecessor of this paper. HADA addresses post-deployment alignment of agents and legacy algorithms with organisational targets and values;

Agentic Discovery addresses the moment *before* deployment — the moment in which the artefact under alignment is brought into existence.

8.0.5 Design-science prototyping traditions

Lean Startup, design-thinking and rapid prototyping have long argued that working artefacts beat documents. Agentic Discovery operationalises that argument now that the cost of a working artefact has collapsed.

8.0.6 Where the resources used to go: empirical distribution of effort across SDLC phases

Before placing Agentic Discovery in the Software Factory lineage (§ 8.7 below), it helps to recall the empirical baseline our field numbers should be read against. Boehm’s *Software Engineering Economics* [67] and the later COCOMO II refresh [68] put requirements-and-design at roughly one third of total project effort on mainstream projects, with code occupying about a quarter and integration-plus-test another third — i.e. discovery and front-end shaping have consistently consumed a non-trivial share of project resources for decades. Survey work on requirements elicitation specifically (Davis et al. [71]) places that activity at 15–30 % of project time on its own. Our 33 hours of prompt-writing across 8 engagements sits comfortably *below* the lower bound of these reference ranges — the speed multiplier reported in § 6.5 is measured against a baseline that is itself well-documented.

Effort distribution is one half of the picture; cost-of-failure distribution is the other. Boehm & Basili’s “Software defect reduction top-10 list” [69] consolidates two decades of empirical SE into a defect-cost amplification pyramid: a problem found and fixed after delivery is roughly $100 \times$ more expensive than the same problem caught during requirements and design, and current projects spend 40–50 % of their effort on avoidable rework — most of it driven by a small fraction of defects originating upstream. Standish CHAOS data 1994–2020 reinforces the upstream-origin point from the failure-rate side: across cohorts, “user involvement”, “executive management support” and a “clear statement of requirements” sit at the top of the success-factor lists, while requirements instability and unclear scope dominate the root-cause counts for failed and challenged projects [70]. Brooks’s standing observation [72] — that “the hardest single part of building a software system is deciding precisely what to build” — is the headline form of the same finding, and Glass’s *Facts and Fallacies* [74] catalogues it as one of the field’s empirical regularities.

There is also a direct precedent for the kind of bottleneck relocation we report. Cao & Ramesh’s empirical study of 16 organisations [73] documents how agile teams did not *eliminate* requirements engineering — they relocated it from an up-front activity to a continuous one. The agile RE practices they identify (face-to-face elicitation, lightweight requirements documents, prototyping for clarification, iterative requirements refinement, requirements prioritisation, test-driven clarification, and constant customer collaboration) collectively trade one bottleneck (write-everything-down-first) for another (sustain-customer-availability-throughout). Our argument in § 7.4 follows the same logical shape applied one rung outward: removing the production-capacity constraint inside the discovery loop does not eliminate the loop, it exposes *idea supply* and *validation speed* as the new binding inputs.

Read against these baselines, the field results in § 6 are not surprising as a quantity of work produced; they are surprising as a *relocation* of where the work happens. The discovery layer used to absorb roughly a third of project effort and the lion’s share of avoidable rework. The agentic configuration described in § 3 leaves that share’s lower portion (drafting an artefact, deploying it, running it once with a stakeholder) effectively for free, and surfaces the upper portion (deciding what to ask the agent to build next, judging whether the candidate fits) as the residual workload. The RE-community vocabulary catalogued by Aurum & Wohlin [75] — recall, precision, completeness, elicitation-time-per-requirement — applies directly to that residual.

8.0.7 Software Factories

The aspiration to *industrialise* software production is older than the term "software engineering" itself. Since Hitachi coined "Software Factory" in 1969, the concept has been reinvented at least four times — as Japanese quality-driven production lines (1970s–1990s), as European composable infrastructure (Eureka, late 1980s), as Microsoft’s model-driven assembly with DSLs and patterns (2004), and most recently as AI-agent orchestration platforms (2024–). Each generation shares a common thesis: software should be *assembled* from validated components through a repeatable, measured pipeline, rather than hand-crafted from scratch each time. In the vocabulary of this paper, all Software Factory traditions live on the *solution* side of the problem/solution boundary (§ 2) — they automate the journey from specification to deployed code, which is precisely the scope we label **Agentic Engineering** (§ 4).

How the literature builds the factory Different communities draw the same ambition with different blueprints — production lines, maturity ladders, product-line architectures, model-driven assemblies, DevSecOps pipelines, agent fleets — but every one encodes the same bet: that software construction can be made predictable, measurable, and increasingly automatic.

Why this matters for Agentic Engineering. Every Software Factory tradition above automates the same pipeline: specification → code → test → deploy. The early generations used deterministic tools (DSLs, code generators, process scripts); the latest generation replaces them with LLM agents that can *reason* about specifications, generate and debug code, and compose multi-step workflows under human oversight. Agentic Engineering (§ 4) is, in this lineage, the probabilistic Software Factory — one where the assembly line is a fleet of agents and the factory manager is the developer who writes prompts, reviews pull requests, and validates output. What none of these traditions address is the question *before* the specification: *what should we build, and for whom?* That question is the domain of Agentic Discovery (§ 2–§ 3).

9 Conclusion and Future Work

This paper introduced **Agentic Discovery** as a first-class phase of the software development lifecycle: a pattern in which AI agents convert stakeholder pains into runnable *agentic probes* — the instruments of the expedition — within a single conversation, allowing specifications, mock-ups, and acceptance criteria to be *extracted from* a validated artefact rather than authored ahead of it. Agentic Discovery has a sibling — **Agentic Engineering** — which uses the same agent stack to deliver validated probes through Roadmap to released code. Together they redraw the cost frontier of the SDLC, moving the cheapest place to learn-what-to-build out of meeting rooms and into running software. In one line: Agentic Discovery is the expedition; the agentic probe is its instrument — a prototype, a simulation or a possible future — and our field study has so far measured the first of these.

Contributions revisited. The five scoped contributions stated in § 1 hold up against the field evidence: **(C1) Artefact** — a working Agentic Discovery runtime instantiating the Three-Role pattern (Dreamer → Orchestrator → Agent) and the layered modernisation model that locates it in the IT estate; the runtime is deployed live (§ 5) and the same instance produced this paper. **(C2) Design knowledge** — the *two-modes pattern* (§ 3) separating Solution-proposing and Problem-validating directions across the problem/solution seam, plus the *Inside-Out* direction (§ 6.8) where the probe is a process simulation rather than deployable code; framed as a new IS design theory in the Walls–Hevner sense, grounded in § 2’s prior-update view and now decomposed into the incumbent / constraints / pain / value frames (s_0, κ, ρ, V). **(C3) Empirical evaluation** — 69 deployable probes of the prototype kind from 8 in-room field engagements with two voluntary partner organisations (~142,000 lines of generated source code; intent→code amplification averaging 1 : 56 and

reaching 1 : 467 at the extreme on the audited $n = 9$ subset; 100 % probe-generation success across observed sessions), scoped to DSRM steps 3–5 in the Solution-proposing mode (§ 7.4 L6). **(C4) Reproducibility kit** — the public anonymised analytics appendix (A.1–A.12, 10 charts), the publicly deployed runtime of § 5, and provenance documentation back to each underlying source for every numeric claim. **(C5) Empirical finding — the bottleneck shifts** — across an engagement series of growing size, the binding constraint moved from production capacity through human orchestration to *stakeholder idea supply and validation speed*; we position this both as a corollary of § 1’s cost-collapse argument and as a re-entry point into the Software Factory tradition (§ 8) — the *probabilistic factory* of Agentic Engineering has a sibling on the Discovery side, a small Software Factory where the rate-limiting factor is the queue of ideas at the inlet (§ 7.4).

In brief. *Where we are:* capability keeps rising and the yardsticks are running out; the environment and harness matter almost as much as the model (§ 1.1). *Core claim:* when building gets cheap, the six-phase SDLC collapses into Agentic Discovery, which finds the right problem, and Agentic Engineering, which builds it right; the seam is the validated spec. *Instrument:* the agentic probe — runnable, disposable, problem-validating or solution-proposing — which also reveals the real constraints. *Evidence:* 69 probes and about 142,000 lines of code from eight engagements in four weeks, with the bottleneck moving from production to orchestration to idea supply and validation. *Engine:* Discovery is agentic search over $\mathcal{P} \times \mathcal{S}$, with the human as the ask-or-act gate (§ 3.7). For organisations this suggests — as our synthesis rather than a measured result — investing in stakeholder time, orchestration and validation gates once production no longer limits: build probes to learn, not to ship, and hand the spec to Engineering only once it is validated.

9.1 Future work

Seven threads invite continuation. (1) **Replication studies** with non-author orchestrators and other agent vendors (OpenAI Codex, Cursor, GitHub Copilot Workspace, open-weight agents) to test vendor- and operator-independence of the pattern. (2) A **comprehensive coding-agent benchmark** of the kind § 8.1 argues is currently missing — measuring multi-day autonomy, multi-repo navigation, terminal and MCP tool orchestration, and human-in-the-loop steering, not isolated patches. (3) Deeper **integration of Agentic Discovery and Agentic Engineering**: how does a probe gracefully cross the seam, and what guarantees about the probe survive the crossing into production? (4) **Customer-side Agentic Discovery** at scale (§ 3.5): treating end users as primary discovery actors, with the funnel mathematics of innovation tournaments [44] as a filter and the IP / privacy questions in § 3.5 as design constraints. (5) **Pre-deployment alignment**: the HADA architecture [1] addresses post-deployment alignment of agents with organisational values; a sister study should examine whether alignment can be embedded directly inside the Discovery loop, before the validated artefact crosses into production. (6) **Scaling the Problem-validating mode**. Experiment B (§ 6.8) demonstrates the Problem-validating mode of the two-modes pattern (§ 3) at illustrative scale ($n = 3$ researcher-authored healthcare cases, probes = interactive process simulations). The § 2 frames *Incumbent under constraints* (s_0, κ, ρ) and *Value of the unlock* ($V(p; s_0)$) supply the vocabulary for measuring such probes — a Problem-validating probe is, formally, an estimator for those four quantities. A field-scale evaluation analogous to § 6.1–§ 6.7 would measure its time-to-probe, acceptance rate and amplification analogue — and would document whether *process-level* probes (re-enactments, walkable scenarios, counterfactual reshaping) elicit the same quality of operational feedback as *software-level* probes do. Healthcare is one candidate domain; other natural targets include regulated administrative workflows, supply-chain hand-offs, on-call routing, and other Inside→Out settings where "replacing" a legacy process outright is rarely allowed in the building. (7) **Software Factory dimensioning + the idea-supply bottleneck**. § 7.4 reports a bottleneck progression — production → orchestration → idea supply — observed across engagements of growing size. Open questions: at what cohort size does the orchestration-automation investment pay back? What instruments measure "idea supply" itself (intake-form quality, dwell time before

a brief is discarded, the ratio of probes iterated to probes abandoned, the velocity at which a stakeholder produces the next counterfactual)? Can the same small-Software-Factory configuration be staged across remote / mixed / fully-asynchronous discovery sessions, or does the in-room concurrency that triggered the shift turn out to be load-bearing? And once idea supply is the binding constraint, what mechanisms (warm-up exercises, stakeholder priming, paired discovery, prior-rendered counterfactual menus) raise the supply rate without lowering its quality? Healthcare and other Inside→Out domains (§ 6.8) inherit the same questions on the Problem-validating side, where validation speed is the analogous variable.

9.2 Acknowledgements

We thank the participating organisations for opening their offices to field experimentation, and the workshop facilitators who acted as Orchestrators across the engagements. The Claude Code agent (Anthropic) was the AI agent used throughout — running on the Claude Opus 4.6 and Opus 4.7 model versions during the field engagements (§ 6), and on Claude Opus 4.7 during the writing of this paper itself; the revised edition of 1 October 2026 was prepared with Claude Opus 5.5. This paper, the runtime described in § 5, the slide companion, and the LaTeX export pipeline were all built by Agentic Discovery sessions — the artefact is its own demonstrator. The authors nonetheless take full responsibility for the content of the paper; every claim, number and citation has been reviewed by us, and any errors are our own.

References

- [1] Pitkäranta, T., & Pitkäranta, L. (2025). *HADA: Human-AI Agent Decision Alignment Architecture*. arXiv:2506.04253.
- [2] Anthropic (2024). *Model Context Protocol (MCP) specification*.
- [3] Google (2024). *Agent-to-Agent (A2A) protocol overview*.
- [4] Brown, T. et al. (2020). *Language Models are Few-Shot Learners*. NeurIPS.
- [5] Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). *Design Science in Information Systems Research*. MIS Quarterly, 28(1), 75–105.
- [6] Ries, E. (2011). *The Lean Startup*. Crown Business. ISBN 978-0307887894.
- [7] Brown, T. (2009). *Change by Design*. HarperBusiness. ISBN 978-0061766084.
- [8] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. ISBN 978-0134610993.
- [9] Beck, K. et al. (2001). *Manifesto for Agile Software Development*.
- [10] Pitkäranta, T. (2026). *AI-SDLC-Impact* — informal analytical companion presenting industry productivity claims phase-by-phase; cited here as the source of the six-phase taxonomy adopted in § 2 (not a peer-reviewed reference).
- [11] Jackson, M. A. (2001). *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley. ISBN 978-0201596274.
- [12] Pohl, K. (2010). *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer. ISBN 978-3642125775.

- [13] Olsen, D. (2015). *The Lean Product Playbook: How to Innovate with Minimum Viable Products and Rapid Customer Feedback*. Wiley. ISBN 978-1118960875.
- [14] Cagan, M. (2017). *Inspired: How to Create Tech Products Customers Love* (2nd ed.). Wiley. ISBN 978-1119387503.
- [15] Torres, T. (2021). *Continuous Discovery Habits: Discover Products that Create Customer Value and Business Value*. Product Talk LLC. ISBN 978-1736633303.
- [16] Gause, D. C., & Weinberg, G. M. (1990). *Are Your Lights On? How to Figure Out What the Problem Really Is*. Dorset House. ISBN 978-0932633163.
- [17] Design Council (2005). *The 'Double Diamond' design process model*. London: UK Design Council.
- [18] Maurya, A. (2012). *Running Lean: Iterate from Plan A to a Plan That Works* (2nd ed.). O'Reilly. ISBN 978-1449305178.
- [19] Snowden, D. J., & Boone, M. E. (2007). A Leader's Framework for Decision Making. *Harvard Business Review*, 85(11), 68–76.
- [20] Albayrak, Ö., & Carver, J. C. (2014). Correlations between Problem Domain and Solution Domain Size Measures for Open Source Software. *IEEE Joint Conference on Software Measurement (IWSM-Mensura)*.
- [21] Sinek, S. (2009). *Start With Why: How Great Leaders Inspire Everyone to Take Action*. Portfolio / Penguin. ISBN 978-1591842804. Companion talk: *How Great Leaders Inspire Action*, TEDxPuget Sound, 2009.
- [22] Pichler, R. (2016). *Strategize: Product Strategy and Product Roadmap Practices for the Digital Age*. Pichler Consulting. ISBN 978-0993499203.
- [23] Perri, M. (2018). *Escaping the Build Trap: How Effective Product Management Creates Real Value*. O'Reilly Media. ISBN 978-1491973790.
- [24] Cagan, M., & Jones, C. (2020). *Empowered: Ordinary People, Extraordinary Products*. Wiley. ISBN 978-1119691297.
- [25] Doerr, J. (2018). *Measure What Matters: How Google, Bono, and the Gates Foundation Rock the World with OKRs*. Portfolio / Penguin. ISBN 978-0525536222. Origin: Grove, A. (1983), *High Output Management*, Random House.
- [26] Akao, Y. (Ed.). (1991). *Hoshin Kanri: Policy Deployment for Successful TQM*. Productivity Press. ISBN 978-0915299577.
- [27] Mintzberg, H. (1994). *The Rise and Fall of Strategic Planning*. Free Press. ISBN 978-0029216057. Complementary: Porter, M. E. (1996). What Is Strategy? *Harvard Business Review*, 74(6), 61–78.
- [28] Naur, P., & Randell, B. (Eds.). (1969). *Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*. Brussels: Scientific Affairs Division, NATO.
- [29] Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*, 26, 1–9. (Reprinted in *Proceedings of the 9th International Conference on Software Engineering*, 1987, pp. 328–338.)

- [30] Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 21(5), 61–72.
- [31] Forsberg, K., & Mooz, H. (1991). The relationship of system engineering to the project cycle. *Proceedings of the First Annual Symposium of the National Council on Systems Engineering (NCOSE)*, Chattanooga, TN, 57–65. (V-Model.) See also: *V-Modell XT*, German Federal Government, since 1992.
- [32] Larman, C., & Basili, V. R. (2003). Iterative and incremental developments. A brief history. *IEEE Computer*, 36(6), 47–56.
- [33] Kruchten, P. (2003). *The Rational Unified Process: An Introduction* (3rd ed.). Addison-Wesley. ISBN 978-0321197702.
- [34] Schwaber, K. (1995). SCRUM Development Process. *OOPSLA'95 Workshop on Business Object Design and Implementation*, Austin, TX. (Updated annually as: Schwaber, K., & Sutherland, J. *The Scrum Guide*, 2020+.)
- [35] Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley. ISBN 978-0201616415. (2nd ed. with C. Andres, 2004, ISBN 978-0321278654.)
- [36] Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley. ISBN 978-0321150783.
- [37] Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press. ISBN 978-0984521401.
- [38] Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley. ISBN 978-0321601919.
- [39] von Hippel, E. (2005). *Democratizing Innovation*. MIT Press. ISBN 978-0262002745.
- [40] Prahalad, C. K., & Ramaswamy, V. (2004). *The Future of Competition: Co-Creating Unique Value with Customers*. Harvard Business School Press. ISBN 978-1578519538.
- [41] Lieberman, H., Paternò, F., & Wulf, V. (Eds.) (2006). *End User Development*. Springer (Human–Computer Interaction Series, vol. 9). ISBN 978-1402042201.
- [42] Chesbrough, H. W. (2003). *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press. ISBN 978-1578518371.
- [43] Schuler, D., & Namioka, A. (Eds.) (1993). *Participatory Design: Principles and Practices*. Lawrence Erlbaum Associates / Routledge. ISBN 978-0805809510.
- [44] Terwiesch, C., & Ulrich, K. T. (2009). *Innovation Tournaments: Creating and Selecting Exceptional Opportunities*. Harvard Business Review Press. ISBN 978-1422152225.
- [45] Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), 45–77.
- [46] Kwa, T., West, B., Becker, J., et al. (2025). *Measuring AI Ability to Complete Long Tasks*. METR / arXiv:2503.14499 (NeurIPS 2025).

- [47] Patwardhan, T., Dias, R., Proehl, E., et al. (2025). *GDPval: Evaluating AI Model Performance on Real-World Economically Valuable Tasks*. OpenAI / arXiv:2510.04374.
- [48] Jimenez, C. E., Yang, J., Wettig, A., et al. (2024). *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* ICLR 2024 / arXiv:2310.06770.
- [49] Rein, D., Hou, B. L., Stickland, A. C., et al. (2023). *GPQA: A Graduate-Level Google-Proof Q&A Benchmark*. arXiv:2311.12022.
- [50] Wijk, H., Lin, T., Becker, J., et al. (2024). *RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts*. METR / arXiv:2411.15114.
- [51] Miserendino, S., Wang, M., Patwardhan, T., & Heidecke, J. (2025). *SWE-Lancer: Can Frontier LLMs Earn \$1 Million from Real-World Freelance Software Engineering?* OpenAI / arXiv:2502.12115.
- [52] Cusumano, M. A. (1991). *Japan's Software Factories: A Challenge to U.S. Management*. Oxford University Press. ISBN 978-0195062168. See also: Cusumano, M. A. (1989). The Software Factory: A Historical Interpretation. *IEEE Software*, 6(2), 23–30.
- [53] Aaen, I., Bøtcher, P., & Mathiassen, L. (1997). Software Factories: Contributions and Illusions. *Proceedings of the 20th Information Systems Research Seminar in Scandinavia (IRIS 20)*, Oslo.
- [54] Humphrey, W. S. (1989). *Managing the Software Process*. Addison-Wesley / SEI Series. ISBN 978-0201180954. Origin of the five-level process maturity framework; basis for CMU/SEI CMM (1991) and CMMI.
- [55] Clements, P., & Northrop, L. (2002). *Software Product Lines: Practices and Patterns*. Addison-Wesley / SEI Series. ISBN 978-0201703320.
- [56] Greenfield, J., & Short, K. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley. ISBN 978-0471202844. With S. Cook & S. Kent.
- [57] Keller, K. M., Lytell, M. C., & Bharadwaj, S. (2022). *Personnel Needs for Department of the Air Force Digital Talent: A Case Study of Software Factories*. RAND Corporation, RR-A550-1.
- [58] Osmani, A. (2026). The Factory Model: How Coding Agents Changed Software Engineering. *AddyOsmani.com* (blog).
- [59] Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI EA '22)*, Article 332, 1–7.
- [60] Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., & Gazit, I. (2022). Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools. *ACM Queue*, 20(6), 35–57. (Reprinted: *Communications of the ACM*, June 2023.)
- [61] Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*. arXiv:2302.06590.
- [62] Imai, S. (2022). Is GitHub Copilot a Substitute for Human Pair-Programming? An Empirical Study. *Proceedings of the 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '22), ACM Student Research Competition*, 319–321.

- [63] Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L., & Lou, Y. (2024). *Large Language Model-Based Agents for Software Engineering: A Survey*. arXiv:2409.02977.
- [64] Walls, J. G., Widmeyer, G. R., & El Sawy, O. A. (1992). Building an Information System Design Theory for Vigilant EIS. *Information Systems Research*, 3(1), 36–59.
- [65] Markus, M. L., Majchrzak, A., & Gasser, L. (2002). A Design Theory for Systems That Support Emergent Knowledge Processes. *MIS Quarterly*, 26(3), 179–212.
- [66] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering* (2nd ed.). Springer. ISBN 978-3-642-29044-2.
- [67] Boehm, B. W. (1981). *Software Engineering Economics*. Prentice-Hall. ISBN 978-0-13-822122-5. — establishes the COCOMO baseline phase-effort distribution for mainstream projects; primary historical reference for "requirements + design $\approx$ 30 % of project effort".
- [68] Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., & Steece, B. (2000). *Software Cost Estimation with COCOMO II*. Prentice-Hall. ISBN 978-0-13-026692-7. — the 2000-era refresh of the 1981 effort-distribution baselines, used as the canonical comparison point in the present paper.
- [69] Boehm, B., & Basili, V. R. (2001). Software Defect Reduction Top-10 List. *IEEE Computer*, 34(1), 135–137. — defect-cost amplification pyramid ($\approx 100\times$ post-delivery vs requirements/design); 40–50 % avoidable-rework finding; 80/20 rule on defect concentration.
- [70] The Standish Group (1994–2020). *CHAOS Report*. Standish Group International. — multi-cohort failure-cause distribution; "clear statement of requirements" persistently in the top three project-success factors. 2015 cohort: agile 39 % successful / 52 % challenged / 9 % failed vs waterfall 11 % / 60 % / 29 %.
- [71] Davis, A., Dieste, O., Hickey, A., Juristo, N., & Moreno, A. M. (2006). Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review. *14th IEEE International Requirements Engineering Conference (RE'06)*, 179–188. — meta-analysis confirming 15–30 % project-time share for requirements elicitation.
- [72] Brooks, F. P. Jr. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley. ISBN 978-0-201-83595-3. — original 1975; the chapter "No Silver Bullet" carries the canonical line that "the hardest single part of building a software system is deciding precisely what to build".
- [73] Cao, L., & Ramesh, B. (2008). Agile Requirements Engineering Practices: An Empirical Study. *IEEE Software*, 25(1), 60–67. — 16-organisation empirical study; identifies seven agile RE practices that collectively *relocate* rather than eliminate requirements-engineering cost (face-to-face elicitation, lightweight documents, prototyping, iterative refinement, prioritisation, test-driven clarification, constant customer collaboration).
- [74] Glass, R. L. (2003). *Facts and Fallacies of Software Engineering*. Addison-Wesley. ISBN 978-0-321-11742-9. — catalogues empirical regularities of the field, including the persistence of unstable requirements as a project-risk factor.
- [75] Aurum, A., & Wohlin, C. (Eds.) (2005). *Engineering and Managing Software Requirements*. Springer. ISBN 978-3-540-25043-2. — collects the RE community's quantitative metrics (recall, precision,

completeness, elicitation-time-per-requirement); supplies the vocabulary that this paper’s O1–O6 hooks into.

- [76] METR (2026). *Time horizons* (measurement page and data, TH1.1). <https://metr.org/time-horizons/> (accessed 12 October 2026).
- [77] METR (2026). *Time Horizon 1.1*. METR blog, 29 January 2026. <https://metr.org/blog/2026-1-29-time-horizon-1-1/>
- [78] Epoch AI (2026). *GDPval benchmark data*. <https://epoch.ai/benchmarks/gdpval>
- [79] OpenAI (2026). *Introducing GPT-5.4*. 5 March 2026. <https://openai.com/index/introducing-gpt-5-4/>
- [80] Google DeepMind (2026). *AlphaProof Nexus: formally verified progress on Erdős problems and OEIS conjectures*. arXiv:2605.22763.
- [81] ARC Prize Foundation (2026). *ARC-AGI-2 and ARC-AGI-3 leaderboards*. <https://arcprize.org/leaderboard> (data of 5 October 2026).
- [82] Epoch AI (2026). *FrontierMath results*. <https://epoch.ai/benchmarks/frontiermath>
- [83] Scale AI & Center for AI Safety (2026). *Humanity’s Last Exam leaderboard*. https://labs.scale.com/leaderboard/humanitys_last_exam
- [84] Terminal-Bench (2026). *Terminal-Bench 4.0 leaderboard*. <https://www.tbench.ai/>
- [85] Epoch AI (2026). *Open models lag closed models by about four months*. Data insight, 29 May 2026. <https://epoch.ai/data-insights/open-closed-eci-gap>
- [86] JetBrains Research (2026). *AI coding agent adoption in 2026*. <https://blog.jetbrains.com/research/2026/08/ai-coding-agent-adoption-2026/>
- [87] Pinna, G., Gong, J., Williams, D., & Sarro, F. (2026). Comparing AI Coding Agents: A Task-Stratified Analysis of Pull Request Acceptance. *MSR 2026 Mining Challenge*. arXiv:2602.08915.
- [88] Rasmussen, J. (1985). The role of hierarchical knowledge representation in decisionmaking and system management. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(2), 234–243. doi:10.1109/TSMC.1985.6313353.
- [89] Vallacher, R. R., & Wegner, D. M. (1987). What do people think they’re doing? Action identification and human behavior. *Psychological Review*, 94(1), 3–15. doi:10.1037/0033-295X.94.1.3.
- [90] van Lamsweerde, A. (2001). Goal-oriented requirements engineering: a guided tour. In *Proc. 5th IEEE International Symposium on Requirements Engineering (RE’01)*, 249–262. doi:10.1109/ISRE.2001.948567.
- [91] Gutman, J. (1982). A means-end chain model based on consumer categorization processes. *Journal of Marketing*, 46(2), 60–72. doi:10.1177/002224298204600207.
- [92] Zachman, J. A. (1987). A framework for information systems architecture. *IBM Systems Journal*, 26(3), 276–292. doi:10.1147/sj.263.0276.
- [93] Zhang, W., Liao, J., Li, N., Du, K., & Lin, J. (2024). *Agentic Information Retrieval*. arXiv:2410.09713.

- [94] Xi, Y., Lin, J., Xiao, Y., et al. (2025). *A Survey of LLM-based Deep Search Agents: Paradigm, Optimization, Evaluation, and Challenges*. arXiv:2508.05668.
- [95] Shi, Z., Chen, Y., Li, H., et al. (2025). *Deep Research: A Systematic Survey*. arXiv:2512.02038.
- [96] Huang, Y., Chen, Y., Zhang, H., et al. (2025). *Deep Research Agents: A Systematic Examination and Roadmap*. arXiv:2506.18096.
- [97] Marchionini, G. (2006). Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4), 41–46. doi:10.1145/1121949.1121979.
- [98] Page, M. J., McKenzie, J. E., Bossuyt, P. M., et al. (2021). The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 372, n71. doi:10.1136/bmj.n71.
- [99] Goldratt, E. M., & Cox, J. (1984). *The Goal: A Process of Ongoing Improvement*. North River Press.