Six Levels of Delegation

A Per-Task Taxonomy of Human–AI Autonomy and the Boundary Where the Operator's Understanding Stops Being Refreshed

Dirk Harms-Merbitz
LinuxToaster.com
dhm@linuxtoaster.com

Abstract

Taxonomies of automation ask how autonomous a system is. For general-purpose language models that is the wrong thing to measure. This paper measures something else: *where the operator's understanding of the work is last refreshed.* Somewhere on that scale is a boundary no autonomy scale marks. On one side of it an operator who missed something can find it by looking harder. On the other side looking harder does nothing.

An operator who reads the work is updated by the work; an operator who reads a test result is updated by an instrument. Between these two positions their knowledge changes in kind, not degree: it becomes bounded by what somebody thought to measure, in advance. Below the boundary, better oversight means paying more attention; above it, attention cannot help, because the missing information was never in the outputs at all. Kalman's notion of observability names this exactly, and it gathers a scattered family of results in the automation literature — surprises, complacency, skill decay — into consequences of one position rather than separate phenomena.

We locate the boundary within a six-level ladder derived from what can actually be handed over in knowledge work — the syntax, the thinking, the writing, the checking, the running, and the choosing. Each level is identified by the one decision the operator keeps, and that decision is also the last place their understanding of the work gets refreshed. The boundary falls between REVIEWER and APPROVER, which is why they are separate rungs here and one rung almost everywhere else.

Two consequences follow. First, the level belongs to the *task*, not the system: an operator may cross the boundary several times an hour, and the drift upward is easy to perform and easy not to notice. Second, most current AI products fix the operator at one level by their shape — a chat window affords COLLABORATOR, inline completion affords REVIEWER, an autonomous agent affords AUDIENCE — so changing level means changing application and losing context at the boundary. We give a case study of a command-line tool that spans all six levels within one binary by delegating composition to the Unix shell, as evidence that *interruptibility* makes the position recoverable rather than merely enumerable. The model is intended to be falsifiable; we state the experiments that would test it.

Keywords: human–AI interaction; levels of automation; delegation; agentic systems; automation complacency; observability; command-line interfaces

1 Introduction

Hand a piece of work to somebody else and you get it back faster, knowing less about it than if you had done it yourself. That trade is old, well documented, and not specific to machines.

Tannenbaum and Schmidt described it as a continuum of manager–subordinate authority in 1958 [20]; agile practice repackaged it as the seven levels of "delegation poker" [2]; human factors research has spent five decades measuring what happens to an operator who occupies the far end of it [3, 11, 5].

What is new with general-purpose language models is not the trade but its *granularity*. Prior automation taxonomies attach a level to a system. SAE J3016 assigns a level to a vehicle's driving automation feature [15]. Sheridan and Verplank's ten levels describe a supervisory control arrangement between a human and a teleoperator [17]. In both cases the level is a property of the deployment, chosen by designers and largely fixed in operation. An LLM-based tool inverts this. The same model, invoked through the same interface, can answer a question, draft a patch for review, edit two hundred files unattended, or run on a schedule with no human present — and the operator can choose among these several times in an afternoon, per task, with no change to the underlying system.

Once the level is a per-task choice, the interesting thing is not the level but a boundary inside it. One sentence carries the rest of this paper:

> Each level is identified by the one decision the operator keeps, and that decision is also the last place their understanding of the work gets refreshed.

In order of importance:

1. **A boundary that autonomy scales do not mark** (Section 5). Between reading the work and reading a test result, what the operator can know changes in kind: it becomes bounded by whatever somebody thought to measure, in advance. Below the boundary, missing something is a lapse of attention and more care recovers it. Above it, more care recovers nothing, because the thing that would have told you never reached the outputs. Observability [7] names this, and it gathers automation surprises, complacency, and skill decay [16, 19, 10, 3] under one cause.

2. **Every level has an invariant** (Section 4). Each level is identified by the one decision the operator still makes. This is a better handle than "how much autonomy," because an operator can say what their invariant is without consulting anybody, and because naming it also names the last point where their picture of the work was refreshed.

3. **A derivation, not an enumeration** (Section 3). The six levels come from asking what is actually available to hand over in knowledge work, rather than from cutting a spectrum into convenient bands. The derivation is what explains the ordering, and what forces reviewing and approving apart instead of leaving them adjacent on one scale.

4. **Product shape as level lock-in** (Section 6). The worst thing about current AI tooling is not that it performs badly at its level. It is that it deletes the other five, so changing level means changing application and restating the task somewhere that has never heard of it.

5. **An implementation that crosses the boundary in both directions** (Section 7). `toast` is a command-line tool that reaches all six levels from one binary by leaving composition, scheduling, and state to the shell. It is here because the claims above are otherwise cheap: telling operators to move between levels is empty if nothing lets them move, and telling them to design the instruments before the run only means something if the instruments are things they can write.

We are explicit at the outset about what this paper is not. It contains no user study, no controlled comparison, and no measurement of the fidelity costs it posits. It is a position paper with a working implementation attached. But it is meant to be wrong in ways that can be found out. It predicts that comprehension of the same artefact falls as the level rises; that level-4 operators miss defects outside test coverage that level-3 operators catch; and that operators asked to name their own level will sometimes get it wrong in the same direction. All three are measurable. Section 9 says how, and where the argument is most likely to break.

2 Related Work

2.1 Levels of automation

The canonical scale is Sheridan and Verplank's ten levels of automation, developed for undersea teleoperation, running from a computer that offers no assistance to one that acts entirely on its own and ignores the human [17]. Parasuraman, Sheridan, and Wickens later argued that a single scale is insufficient and proposed that automation be characterised along four *stages* of information processing — acquisition, analysis, decision, and action — each of which may be automated to a different degree [12]. Endsley and Kaber's taxonomy similarly distinguishes roles across monitoring, generating, selecting, and implementing [5]. SAE J3016 gives the most widely deployed instance of a level scale, assigning driving automation to six levels distinguished by which of the driving subtasks the system performs and whether the human is a fallback [15].

Our decomposition is closest in spirit to Parasuraman et al.: we too identify functionally distinct components of a task that can be delegated separately. We differ in two ways. First, our components are those of *knowledge work* performed at a computer rather than of information processing in supervisory control. Second, and more importantly, we treat the resulting level as a property of a task instance chosen at invocation time, not a property of a system fixed at design time. Shneiderman's human-centred AI framework makes a related move by separating human control from computer automation into two axes rather than one, arguing that high automation and high human control are compatible [18]. Our ladder can be read as a description of how an operator moves within such a space during ordinary work.

2.2 Delegation, before there were machines to delegate to

Tannenbaum and Schmidt's continuum of leadership behaviour [20] is, in substance, a levels-of-automation scale for human subordinates, and predates Sheridan and Verplank by twenty years. Its descendants in software practice — most visibly Appelo's delegation poker, which discretises the continuum into seven named levels applied per decision area rather than per person [2] — already contain the move we are making, namely that the level is assigned to a unit of work rather than to a relationship. The lineage matters because it shows the per-task view is not something LLMs invented. They only made re-assigning a level cost nothing.

2.3 Ironies, surprises, and skill decay

Bainbridge's "Ironies of Automation" [3] states the central problem of this paper in its classical form: automating the routine parts of a task leaves the human responsible for exactly the situations the automation could not handle, while simultaneously depriving them of the practice that would let them handle those situations. Sarter, Woods, and Billings catalogue the resulting

"automation surprises," in which operators are unable to answer basic questions about what the automation is doing and why [16]. Norman argued that the problem is generally not too much automation but inappropriate feedback — a system that acts without keeping its operator informed [10]. Parasuraman and Riley's use/misuse/disuse/abuse framework [11] and Skitka et al.'s work on automation bias [19] document how monitoring degrades in practice: operators under-detect failures of reliable automation and over-accept its recommendations. Endsley's later synthesis frames this as the "automation conundrum": increasing autonomy reduces the operator's situation awareness precisely as it increases the consequence of the operator's residual role [4]. Klein et al. argue that the fix is not more autonomy but better *team* properties, notably observability and directability [8].

None of these phenomena are new. What is new is that each one can now be pinned to a rung, and that the rung's invariant says which kind of understanding goes missing.

2.4 Language-model tooling

The tools that motivate this paper — interactive assistants, inline code completion, and autonomous agents — have been studied largely one at a time. Amershi et al.'s guidelines for human–AI interaction give design principles that assume an interactive, mixed-initiative setting [1, 6]. Empirical work on code completion reports substantial task-time reductions — a controlled experiment found a treated group completed an implementation task 55.8% faster [13] — alongside usability problems in verifying generated code [21]. Agent frameworks interleave reasoning with tool invocation [23] and, in the software engineering setting, operate against a repository with little or no human intervention [22]. To our knowledge these lines of work have not been placed on a common axis, and the tools themselves are rarely compared on the dimension we consider primary: which decisions they leave to the operator, and whether the operator may change that.

2.5 Unix composition

Our case study rests on the design tradition of Unix: small programs communicating through text streams, with composition supplied by the shell rather than by any single program [14, 9]. The relevance is not nostalgia. Composition, scheduling, and persistence in that tradition are external to the tool, which means a tool can be moved between levels of autonomy by changing the shell expression that invokes it, without the tool acquiring any new interface for the purpose.

3 Deriving the Ladder

Rather than partition a spectrum, we ask what is available to be handed over. Consider a person doing knowledge work at a computer. The work decomposes into parts they hold separately, and each part can be released independently of the others only up to a point.

The syntax.
The operator knows what they want but not how to express it: the flag, the incantation, the regular expression. Delegating this changes nothing about who decides what happens.

The thinking.
The operator knows the goal but not the approach. Delegating this means receiving proposals about how to proceed and choosing among them.

L	Role	Delegated	Invariant: what is still yours
1	One-shot	the syntax	every action
2	Collaborator	the thinking	every direction
3	Reviewer	the writing	every change
4	Approver	the checking	every result
5	Audience	the running	the system
6	Farmer	the choosing	the environment

Table 1: The six levels, the component of the work delegated at each, and the invariant that identifies it. The invariant is both the operator's remaining control point and the last place their model of the work is refreshed.

The writing.
The operator knows the change they want but no longer performs the keystrokes that make it. Delegating this ends the incidental learning that comes from writing a system by hand.

The checking.
The operator stops reading the produced work and instead reads *signals* about it: a test suite, a type checker, a green build.

The running.
Nobody is at the prompt. The work is initiated by a schedule or a loop rather than by a person.

The choosing.
The operator stops selecting which work is done next, retaining only the conditions under which work is generated.

Two observations follow. First, these are ordered, though the order is a dependency relation rather than a total ranking. One cannot delegate the checking while still typing every command, because the artefact being checked is being produced by hand; one cannot leave the prompt unattended while still choosing each next move. The ordering is therefore a partial order induced by dependency, which happens to be a chain over these six. That dependency is why this is a ladder and not a menu. It is also the claim most likely to be wrong, and Section 9 says why.

Second, the list is not homogeneous. Two of the five transitions are unlike the others, and distinguishing them is the substantive structural claim of this paper.

The transition from delegating the writing to delegating the checking (levels 3 to 4) changes what the operator can know. Before it, they are updated by the work itself; after it, by an instrument that measures the work. The transition from delegating the checking to delegating the running (levels 4 to 5) changes where the operator stands: the first four delegations are components of a task they are still holding, whereas the last two are the task itself, and then the selection of tasks — positions *outside* the work rather than further along inside it. Past the second transition the instrument can no longer be adjusted while the work is under way, and the operator cannot intervene mid-run.

The first is a boundary of kind; the second is a boundary of time. The second compounds the first rather than repeating it. Figure 1 shows both.

Mapping the six delegations onto the operator's resulting role gives the ladder in Table 1.

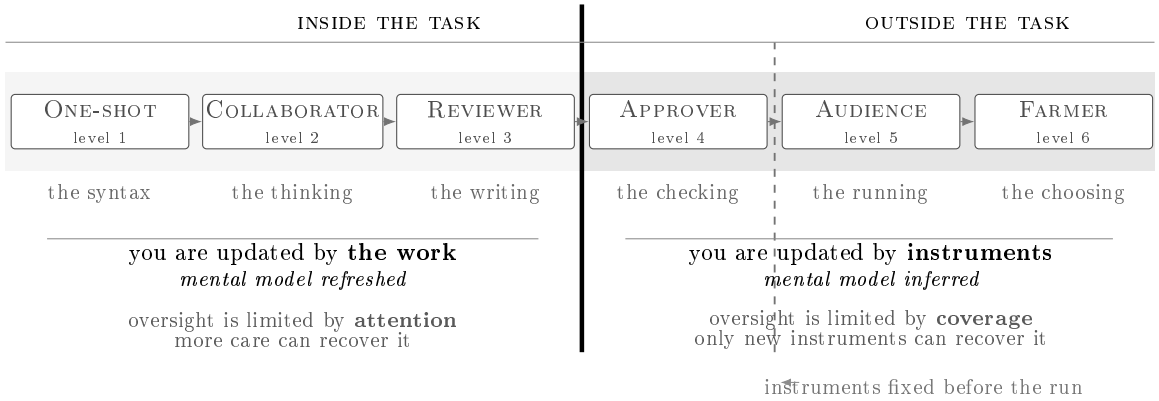

Figure 1: The ladder and its two boundaries. The heavy line is the boundary of *kind*: to its left the operator is updated by the work, to its right by an instrument that measures the work, so a defect that no instrument tests for leaves no trace in what the operator sees. The dashed line is the boundary of *time*: beyond it nobody is at the prompt, the instruments can no longer be revised while the work runs, and the operator cannot intervene. Which side of the heavy line you are on is the thing worth knowing about your own position; the number of rungs is not.

Why reviewer and approver are two rungs. Existing scales tend to collapse these. We keep them apart because reading the work and reading a signal about the work are different acts with different failure modes. A reviewer who misses a defect failed to notice something that was in front of them. An approver who misses a defect was never shown it: the defect fell outside what the instrumentation was built to detect. The first failure is a matter of attention, the second of coverage, and no amount of additional care at level 4 converts one into the other. The gap between these two rungs is, in our experience, where most surprises originate, and Section 5 argues that it is a boundary rather than a gap.

4 The Six Levels

We now describe each level by its invariant, its characteristic interaction, and the fidelity it costs. Examples are drawn from software work because that is the domain we know best and because it makes the signals concrete; we discuss generality in Section 8.

4.1 Level 1: One-shot

Invariant: every action. One command in, one answer out; nothing happens that the operator did not ask for. The model supplies vocabulary — the name of the flag, the shape of the query, an interpretation of a log — and returns control immediately. This is the level at which delegation costs the operator nothing in understanding, because they have still performed the act and merely acquired the wording for it. It is also, in the tools we survey, the level least well served: the dominant interaction forms all sit at level 2 or above.

4.2 Level 2: Collaborator

Invariant: every direction. The system proposes; the operator decides what is worth doing next. Context persists across turns, so the exchange is a dialogue rather than a series of independent queries. The operator has given up the approach but retains the trajectory, and because every next move is theirs, they generally still understand the answer they end up with. This is the level that chat interfaces afford, and it is the level at which the mixed-initiative literature applies most directly [6, 1].

4.3 Level 3: Reviewer

Invariant: every change. The system writes; the operator reads all of it before it takes effect. The characteristic loss here is the incidental knowledge that came from typing: an operator who reviews a codebase for a year knows it differently from one who wrote it. The characteristic risk is review fatigue. Nobody decides to stop reading diffs. The fortieth file in a forty-file change gets skimmed, then the next change opens with the same forty files and the skimming starts at the tenth, and the level has become level 4 without anyone choosing to climb.

An underexplored option at this level is to delegate the review as well, to a second model instance with a different instruction and *without* access to the reasoning that produced the artefact. Independence of the review from the generation is a structural property worth preserving, and it is easy to obtain when the interface between stages is a text stream.

4.4 Level 4: Approver

Invariant: every result. The operator stops reading the work and reads whether it passed. A type checker reporting no errors across two hundred files tells the operator that the code satisfies the type checker; it does not tell them that the type checker covers what matters, and it does not surface a warning emitted mid-run unless the operator's pipeline was built to surface it. This is a defensible place to work, and much professional practice already lives here. The requirement is that the operator knows they are here, because the natural reading of a green check is stronger than the check's actual warrant — a form of automation bias documented long before language models [19, 11].

4.5 Level 5: Audience

Invariant: the system. Nobody is at the prompt. The operator wrote the loop, the capability allowlist, and the stopping condition, and then left. Everything they subsequently learn about the work arrives through instrumentation they specified in advance. The invariant is no longer a per-item decision but a design. Control is exercised once, in advance, and the quality of that one act determines everything that can later be known or corrected. When something goes wrong at three in the morning on Friday, the decision that governs it was made on Tuesday, by whoever wrote the allowlist and the stopping condition.

4.6 Level 6: Farmer

Invariant: the environment. Level 5 automates tasks; level 6 grows deliverables. The operator seeds an environment — an outline, a voice, a set of permitted tools, a schedule — and returns for a harvest: a manuscript, a plan, a codebase. They no longer choose the tasks, their order, or

their timing; they choose what grows and where. The retained decision is the most leveraged of the six and the furthest from the work, and the operator's model of the artefact at harvest time is whatever the artefact and its notes tell them.

5 The Cost of Climbing

Every rung up buys leverage and costs fidelity. The cost is not evenly distributed across the rungs, and it is not paid in money or compute.

5.1 The invariant is also the refresh point

At each level, the decision the operator retains is the decision they must be informed enough to make, and therefore the point at which their picture of the work is updated. An operator at level 2 who decides each next move is continuously re-reading the problem. An operator at level 3 who reads each change is updating on the artefact but no longer on the process. An operator at level 4 is updating on a summary statistic. An operator at level 5 is updating on whatever their logging captured. The invariant is thus diagnostic. Naming it tells the operator what they are deciding, and by doing so tells them where their picture of the work stops being refreshed. It turns a question nobody can answer ("how much autonomy is right here?") into one anybody can ("which decision am I keeping, and will I know enough to make it?").

5.2 The boundary of kind: observation and instrumentation

Up to and including level 3, the operator reads the work. From level 4 they read a measurement of it. That is a different loss from the one that comes after it, and the two are worth keeping apart.

Control theory names the relevant property. Kalman's notion of observability asks whether a system's internal state can be reconstructed from its outputs over time [7]. An unobservable mode is one whose behaviour leaves no trace in the measured signals; no amount of attention to those signals will reveal it, because the information is absent rather than obscured. The analogy is imperfect — a work process is not a linear dynamical system, and we use the term descriptively rather than formally — but the structural point transfers exactly. From level 4 onward the operator is a state estimator restricted to a chosen output map, and any property of the work not touched by that map is invisible in principle.

Below the boundary, an operator who missed something can find it by reading more carefully; the binding constraint is attention, and effort relieves it. Above the boundary, effort does not: an approver who reads the passing test suite ten times learns nothing about the property no test asserts. The binding constraint is coverage, and only new instruments relieve it. This is why we treat reviewer and approver as separate rungs rather than adjacent points on one scale, and why a scale that reports only "how autonomous" loses the distinction that matters most to the person holding responsibility.

Norman made essentially this argument about aviation automation thirty-five years ago: the problem is not that the system acts, but that it acts without keeping the operator informed [10]. Three of Klein et al.'s ten challenges for automation that can function as a team member concern exactly this: being directable, making current state obvious, and communicating status and intentions [8]. The problem belongs to one transition rather than to automation at large, and that transition can now be crossed and re-crossed inside a single afternoon.

5.3 The boundary of time: when the instruments can be chosen

The move from level 4 to level 5 does not repeat this transition; it compounds it. The operator was already reading instruments at level 4, but could still add one — run a different check, read a diff, stop and look — because they were present while the work proceeded. From level 5 nobody is at the prompt. The output map is fixed before the run, so a gap discovered during execution cannot be closed during execution, and the operator cannot intervene between the error and its consequences.

Below level 4, improve oversight by paying attention; at level 4, by adding instruments during the work; from level 5, by designing the instruments before it, since that is the last moment at which the choice exists. It follows that the most consequential engineering at levels 5 and 6 is not the prompt or the model but the instrumentation and the stopping condition, which is a claim the current tooling literature does not much reflect.

5.4 Compounding: the skill that would have caught it

The two costs interact, and this is the mechanism we regard as most serious. It is difficult to detect a flawed conclusion one did not participate in reaching, partly because detection depended on the reasoning that was skipped, and partly because nothing in the instrumentation was watching for it. Efficiency improves while the judgement that would have caught the error quietly atrophies. This is Bainbridge's irony [3] restated for knowledge work, and it is sharpened here by the per-task framing: because the level can now be chosen many times a day at negligible cost, the drift upward is easy to perform and easy not to notice. An operator who never decided to become an approver may nonetheless have become one by Thursday.

None of this is an argument against climbing. It is an argument for climbing deliberately, per task, with the trade named.

6 Product Form as Level Lock-In

There is a question nobody asks of an AI tool: not how well it performs at its level, but how many levels it lets you occupy.

By that measure the current crop does badly, in a specific way. The shape of the product picks the operator's level before they have typed anything:

- A **chat window** places the operator at collaborator and keeps them there. It is awkward to use for a single-shot lookup and structurally incapable of unattended operation.
- **Inline completion** places the operator at reviewer. The artefact is produced and presented for acceptance; the interaction has no natural form for "discuss the approach first" or "do this across two hundred files."
- An **autonomous agent** places the operator at audience and asks for trust. Intermediate state may be displayed, but it is displayed as a narrative rather than offered as a value that the operator can capture, filter, or branch on.

None of these afford movement. The ladder exists regardless — the operator still needs all six positions during a working week — so it is climbed by switching applications. The cost of that is not inconvenience but state: each tool has its own history, its own notion of project context, its own memory. The chat window has never heard of the file the agent just edited. Dropping a

rung to look at something more closely means explaining the whole task again, somewhere that was not there for any of it.

6.1 Interruptibility

What separates a ladder you can climb from a list of rungs is *interruptibility*: being able to stop, look, and start again at a different level without losing the work. That takes at least (i) intermediate state that exists as an inspectable value rather than as a rendered narrative; (ii) a machine-readable outcome the operator can branch on; (iii) the ability to halt a running process and recover its partial results; and (iv) capabilities that the operator grants explicitly rather than receives by default.

Autonomy that cannot be interrupted is not a higher rung on the same ladder. It is a different structure, because there is no path from it back to the levels below.

7 Case Study: Crossing the Boundary in Both Directions

Three of the claims above cost nothing to make. Choose a level per task — with what? Design the instruments before the run — from which menu, written by whom? Come back down when the result looks wrong — back down to what, if the work only exists inside somebody's product? Each one needs a tool in which the thing is expressible, and until there is one the argument is a preference.

`toast` is that tool. What follows is evidence about what can be built, not about what works better. Section 9 states the competing interest and the evaluation this section is not.

7.1 Design

`toast` implements none of composition, scheduling, persistence, or concurrency. Each is delegated to the shell. The tool reads standard input, writes standard output, edits a named file in place when given one, sets an exit code, and reads dot-files from the working directory for context. Every level above is then reached by a shell expression rather than by a mode of the program.

- **Context** comes from `.crumbs`, a plain-text file discovered by walking up the directory tree, which the model may also append to. Memory is therefore a file the operator can read, edit, diff, and delete.
- **Capabilities** come from `.tools`, an allowlist with one permitted command per line. Absent the file, no tool invocation occurs. This is the mechanism behind requirement (iv) above: autonomy is opt-in and enumerated by the operator.
- **Personas** are symbolic links to the same binary; the invocation name selects a system prompt. A pipeline of differently-named links is therefore a pipeline of differently-instructed model instances, each seeing only its predecessor's standard output.
- **Termination** is signalled through the exit code: the process exits non-zero when the model emits a completion token, so an ordinary shell loop terminates when the work is judged finished rather than after a fixed count.

7.2 The levels in practice

Level 1. A pipe supplies the data and the operator supplies the question; control returns immediately.

```
$ ps aux | toast "anything suspicious"
```

Level 2. Invoked with no arguments, the tool opens an interactive session whose history persists in a local file, so the conversation resumes rather than replays.

Level 3. Review is a pipeline stage. Because each stage sees only the previous stage's output, the reviewing instance never sees the instruction that produced the artefact, giving independence between generation and critique for free:

```
$ cat auth.py | Dev "fix the token refresh" \
              | Paranoid "what is wrong with this patch"
```

Level 4. In-place editing across a file set, gated on a checker. The operator reads the exit status of `mypy`, not the diffs:

```
$ find . -name "*.py" -exec toast {} "add type hints" \;
$ mypy .
```

The honest description of this command is that 199 files changed and two were read. The tool makes that ratio visible, which is itself a design choice.

Level 5. Scheduling is `cron`; iteration is a shell loop whose condition is the tool's exit code. The allowlist governs what the unattended process may do:

```
$ printf 'df\njournalctl\nsystemctl\n' > .tools
$ while toast server.py "refactor until clean; print DONE when finished"; do :; done
```

Level 6. The operator seeds a directory — an outline, a persona file, an allowlist — schedules the loops, and returns to a directory of files. Harvesting uses the same ordinary tools as inspection at level 1, which means a level 6 artefact can be dropped back to level 1 for examination without leaving the shell.

7.3 What the case study does and does not show

It shows that the six levels can be reached from one artefact, that moving between them is an edit to a command line rather than a change of application, and that context (`.crumbs`), instruction (persona), and permission (`.tools`) survive the move because they are files in the working directory rather than state inside a product. It also shows that the four interruptibility requirements are satisfiable by construction when composition is external: pipes give inspectable intermediates, exit codes give branchable outcomes, process signals give halting, and the allowlist gives explicit capability grants.

Most relevant to Section 5.2, it shows that the boundary can be crossed downward as well as upward. The instrument at level 4 is an ordinary command the operator wrote (`mypy`, a

test runner, a linter), so adding an instrument is editing a command line rather than requesting a feature; and because every intermediate is a file or a stream, a result that looks wrong can be dropped back to level 1 for direct reading without restating the task elsewhere. The same property holds at the top of the ladder: a level 6 harvest is a directory of plain files, so the artefact and the notes the system kept for itself are inspectable by the same means as anything else. Whether operators actually descend when they should is an empirical question we do not answer; what the design establishes is that the descent is available at all, which is the property the products surveyed in Section 6 lack.

It does not show that operators in fact choose levels more appropriately with such a tool, that the fidelity costs we describe are smaller under it, or that the design is usable by anyone not already fluent in a shell. That last point is a real limitation rather than a rhetorical concession: the design purchases level mobility with a prerequisite that excludes most users of AI tools, and we do not know whether the property can be preserved in a graphical interface.

8 Discussion

Generality beyond software. The derivation in Section 3 refers to syntax, writing, and checking, which are readily interpreted in software but are not specific to it. Editorial work, legal drafting, and data analysis admit the same decomposition, with the checking step supplied by a citation check, a compliance review, or a validation script. The boundary of kind should transfer wherever the check can be automated and the work cannot be simultaneously read. We expect the ordering claim to be weakest in domains where the artefact is short enough that reading it is never the bottleneck.

Choosing a level. The taxonomy suggests a rule of thumb rather than a formula: choose the highest level at which the invariant is a decision you can actually make with what you will know, and no higher. Where consequences are severe, irreversible, or poorly instrumented, that argues for a lower rung regardless of the model's capability, because the binding constraint is the operator's information rather than the model's competence.

Implications for tool design. If this is right, the question to ask a product is not "how autonomous is it" but "which levels does it support, and what does it cost to move between them?" That belongs on the box. A tool that does one level well and forecloses the rest has not stayed out of the operator's business. It has chosen their side of the boundary for them, and said nothing about doing so.

9 Limitations and Threats to Validity

No empirical evaluation. Every causal claim in Section 5 — that fidelity declines with level, that the decline changes in kind at the 3/4 boundary, that unnoticed upward drift occurs — is a hypothesis. We adopt them from an adjacent literature [3, 4, 19] in which they are established for supervisory control of physical systems, and we do not establish that they transfer.

The ordering may be weaker than a chain. We argue the levels are dependency ordered, but we can construct partial counterexamples: an operator may delegate the checking (level 4) for a mechanical subtask while still writing the substantive code by hand, which mixes rungs within

one activity. The correct statement may be that the ordering holds per delegated component rather than per session, in which case a session occupies a set of levels rather than a point. We regard resolving this as the first piece of work the taxonomy needs.

Selection of six. A different decomposition would yield a different count. We do not claim six is canonical; we claim the derivation is stated explicitly enough to be argued with, which enumerative taxonomies usually are not.

Competing interests. The author is the developer of `toast` and is affiliated with Linux-Toaster.com, which distributes it commercially. The case study in Section 7 is therefore not independent, and we have confined it to claims about what is constructible. It is not a comparative evaluation, and no performance, productivity, or preference claim is made on the tool's behalf. The framework in Sections 3–6 is intended to stand or fall on its own; a reader who rejects the case study entirely should still be able to accept or reject the boundary argument on the evidence given for it.

Empirical programme. The framework implies specific studies: (i) a within-subjects experiment measuring comprehension of an artefact produced at each level, holding the artefact constant, to test the fidelity gradient; (ii) a defect-detection study contrasting level 3 and level 4 operators on defects inside and outside test coverage, to test the discontinuity claim; (iii) a longitudinal observation of level selection in real work, to measure whether upward drift occurs and whether operators can accurately report their own level; and (iv) a comparison of level-mobile and level-locked tooling on tasks requiring more than one level, measuring the cost of transitions.

10 Conclusion

The question posed by current AI tooling is usually framed as how autonomous a system can become. We have argued that this is the wrong unit and the wrong question. The unit is the task, not the system; the question is where the operator remains in the control loop, and whether they can move.

The six levels come from what is available to hand over. Each is identified by the one decision the operator keeps, which is also the last place their understanding gets refreshed — so leverage and fidelity trade against each other rung by rung, and somewhere in the middle the operator stops watching work and starts watching signals. Most current products fix the level by their shape, and that immobility, not performance at any one level, is what is most wrong with them. A tool that leaves composition to the shell can span the whole ladder, and interruptibility is what makes the climb reversible.

The older discipline is instructive here. Unix was never principally about automation; it was about keeping the person at the appropriate point in a loop and giving them the means to move. AI tooling deserves the same discipline. The framework here is stated in a form that can be tested and, if it is wrong, corrected.

References

[1] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, and E. Horvitz. Guidelines for human–AI inter-

action. In *Proc. CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019. doi:10.1145/3290605.3300233

[2] J. Appelo. *Management 3.0: Leading Agile Developers, Developing Agile Leaders*. Addison-Wesley, Boston, MA, 2011. See ch. 5, "How to Empower Teams," pp. 119–146, for the seven delegation levels.

[3] L. Bainbridge. Ironies of automation. *Automatica*, 19(6):775–779, 1983. doi:10.1016/0005-1098(83)90046-8

[4] M. R. Endsley. From here to autonomy: Lessons learned from human–automation research. *Human Factors*, 59(1):5–27, 2017. doi:10.1177/0018720816681350

[5] M. R. Endsley and D. B. Kaber. Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics*, 42(3):462–492, 1999. doi:10.1080/001401399185595

[6] E. Horvitz. Principles of mixed-initiative user interfaces. In *Proc. CHI Conference on Human Factors in Computing Systems*, pages 159–166, 1999. doi:10.1145/302979.303030

[7] R. E. Kalman. On the general theory of control systems. In *Proc. First International Congress of the IFAC on Automatic and Remote Control*, Moscow, volume 1, pages 481–492. Butterworths, London, 1960.

[8] G. Klein, D. D. Woods, J. M. Bradshaw, R. R. Hoffman, and P. J. Feltovich. Ten challenges for making automation a "team player" in joint human-agent activity. *IEEE Intelligent Systems*, 19(6):91–95, November–December 2004. doi:10.1109/MIS.2004.74

[9] M. D. McIlroy, E. N. Pinson, and B. A. Tague. UNIX time-sharing system: Foreword. *Bell System Technical Journal*, 57(6):1899–1904, July–August 1978.

[10] D. A. Norman. The 'problem' with automation: Inappropriate feedback and interaction, not 'over-automation'. *Philosophical Transactions of the Royal Society of London, Series B: Biological Sciences*, 327(1241):585–593, 1990.

[11] R. Parasuraman and V. Riley. Humans and automation: Use, misuse, disuse, abuse. *Human Factors*, 39(2):230–253, 1997. doi:10.1518/001872097778543886

[12] R. Parasuraman, T. B. Sheridan, and C. D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems and Humans*, 30(3):286–297, May 2000. doi:10.1109/3468.844354

[13] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv:2302.06590, 2023.

[14] D. M. Ritchie and K. Thompson. The UNIX time-sharing system. *Communications of the ACM*, 17(7):365–375, July 1974.

[15] SAE International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Recommended Practice J3016_202104, revised April 2021 (first issued January 2014). https://doi.org/10.4271/J3016_202104

[16] N. B. Sarter, D. D. Woods, and C. E. Billings. Automation surprises. In G. Salvendy, editor, *Handbook of Human Factors and Ergonomics*, 2nd edition, pages 1926–1943. Wiley, New York, 1997.

[17] T. B. Sheridan and W. L. Verplank. Human and computer control of undersea teleoperators. Technical report, Man-Machine Systems Laboratory, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, July 1978. DTIC accession ADA057655.

[18] B. Shneiderman. Human-centered artificial intelligence: Reliable, safe and trustworthy. *International Journal of Human–Computer Interaction*, 36(6):495–504, 2020. doi:10.1080/10447318.2020.1741118

[19] L. J. Skitka, K. L. Mosier, and M. Burdick. Does automation bias decision-making? *International Journal of Human–Computer Studies*, 51(5):991–1006, 1999. doi:10.1006/ijhc.1999.0252

[20] R. Tannenbaum and W. H. Schmidt. How to choose a leadership pattern. *Harvard Business Review*, 36(2):95–101, March–April 1958.

[21] P. Vaithilingam, T. Zhang, and E. L. Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, article 332, 7 pages, 2022. doi:10.1145/3491101.3519665

[22] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652, 2024. arXiv:2405.15793.

[23] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. arXiv:2210.03629.