The Last Editor

After decades of teaching people how to operate software, we are building software that learns how to listen.

Rafael M. Ehlers · rafaelmehlers.com.br · July 2026

For most of my career, there was an editor in the middle.

At MailPoet, it was a newsletter editor. At GravityKit, it was a form editor. At Octane AI, it was a quiz editor. The products changed, the markets changed, and the technology underneath them changed, but the basic arrangement remained familiar: a person arrived with an intention, and an interface asked them to translate it into the language of software.

Write the subject line here. Add a field there. Choose a layout. Define a condition. Connect one answer to one outcome. Save, preview, publish.

The editor stood between what a person wanted and what the system could understand.

I did not notice the pattern at first. It took years of support tickets, product specifications, releases, regressions, and conversations with frustrated users before the repetition became obvious. I had spent a large part of my working life inside variations of the same problem: how to help someone turn an idea into structure without making the structure feel like punishment.

Editors are unforgiving products. Every rough edge appears in someone's workflow within minutes. A confusing label is not merely a sentence that could have been better written. It becomes a campaign sent with the wrong settings, a form that fails to collect the right information, or a recommendation flow that produces the wrong result. Editors expose the distance between the mental model of the person using the product and the internal model of the software.

That distance is where product work lives.

The bargain of the editor

The graphical editor was one of computing's great acts of translation.

Earlier systems required people to express instructions through formal commands. Graphical interfaces replaced much of that syntax with visible objects and reversible actions. Ben Shneiderman described this approach as direct manipulation: keep the objects of interest visible, let the user act on them incrementally, and provide immediate feedback about the result.[1] The idea became so ordinary that we stopped seeing how radical it was. Documents could be edited by changing what looked like a page. Files could be moved by dragging icons between folders. A person no longer needed to describe every operation in the machine's native language.

But direct manipulation did not eliminate translation. It redesigned the terms of the exchange.

The user still had to learn what the software considered an object, which actions were available, what each field meant, how the hierarchy worked, and which sequence of operations would produce the intended result. The editor made the system visible, but it also made the system's ontology unavoidable.

A newsletter editor teaches you that a message consists of blocks, columns, links, audiences, and sending rules. A form editor teaches you that a conversation can be represented as fields, validations, conditions, and entries. A quiz editor teaches you that a recommendation can be decomposed into questions, answers, branches, scores, and outcomes.

These structures are useful. They are what make software reliable, inspectable, and repeatable. They are also work.

The hidden labor of using an editor is the labor of converting intention into categories the software already knows how to process. The person must decide which part of an idea belongs in which field. They must anticipate the system's interpretation before the system acts. They must learn the product well enough to think with it.

Much of product design has therefore been an effort to make this translation less painful. We rename labels, reorganize menus, reduce steps, add previews, improve defaults, expose templates, and remove decisions that do not need to be made. We call this usability, but underneath it lies a more fundamental ambition: shorten the distance between what the person means and what the machine requires.

For decades, that distance was treated as a property of the interface.

Now it is becoming a property of the model.

When the interface learns to listen

Language models introduce a different arrangement.

Instead of opening the correct screen, finding the correct control, and translating an intention into a predefined structure, a person can describe what they want in the language they already use. They can include context, uncertainty, examples, exceptions, and half-formed thoughts. The system is expected to infer the task, identify the relevant details, and produce something structured enough to act on.

The direction of translation begins to reverse.

In the traditional editor, the person does most of the interpretive work. They study the interface and reshape the intention until it fits. In an agentic interface, the system is asked to do more of

that work. It listens to an unstructured request, interprets what matters, decides what can be inferred, asks questions when necessary, and converts the result into actions.

Natural language does not merely replace buttons with sentences. It changes who carries the burden of structure.

This possibility has existed as an aspiration for a long time. Research on mixed-initiative interfaces described systems in which humans and software contribute to a task together, with the initiative shifting according to uncertainty, cost, and context.[2] Conversational agents promised an even more familiar interaction model, but early systems repeatedly disappointed users because ordinary conversation creates expectations that brittle software cannot sustain. Luger and Sellen found that people often approached conversational assistants as if they were capable collaborators, then had to reverse-engineer their limitations through trial and error.[3]

Large language models improve the breadth of what can be expressed and interpreted, but they do not remove the problem. Prompting is itself a form of programming, even when the programming language resembles ordinary prose.[4] Studies of non-experts working with language models have shown that users still struggle to form reliable mental models of what a model can do, how instructions should be phrased, and why small changes sometimes produce radically different outcomes.[5]

The interface may feel more natural, but the translation has not vanished.

It has become less visible.

The editor did not disappear

When a person says, “Create a quiz that recommends the right skincare routine,” the system must still produce structure.

It must decide what counts as a useful question, which answers are meaningful, how outcomes differ, what information should be requested, and how the result should be represented. If it creates the quiz inside a product, it must eventually generate the same kinds of objects an editor would have exposed: questions, answer options, conditions, scores, content blocks, and recommendation rules.

The schema is still there.

The difference is that the user may no longer see the moment in which their intention is translated into it.

This is the central tension of agentic product design. The editor can disappear from the surface while becoming more powerful underneath. The system silently chooses categories, resolves

ambiguity, supplies defaults, compresses context, and decides which parts of the request deserve durable representation.

Every agent is therefore an editor.

It edits the user's language into an internal account of what the user meant. It selects some details and discards others. It turns possibilities into parameters. It interprets tone, priority, and implied constraints. When it acts, it does so on the basis of that edited representation.

The danger is not only that the system may produce a bad output. It may produce a good output for the wrong interpretation.

Traditional editors made many assumptions visible. The user could inspect the selected field type, the condition, the recipient list, the layout, or the scoring rule. A conversational system can conceal those assumptions inside a fluent response. Fluency makes the interpretation feel settled before it has been examined.

The old editor created friction because it exposed structure.

The new editor creates risk because it can hide structure too well.

The problem of invisible interpretation

A language model does not receive intention directly. It receives evidence of intention.

The distinction matters. Human requests are incomplete because human thought is contextual. We omit what seems obvious. We revise ourselves mid-sentence. We use the same word differently across situations. We ask for one thing while quietly optimizing for another. A skilled human collaborator detects some of this through shared context and clarification. A model reconstructs it from language, prior interactions, system instructions, available tools, and learned statistical patterns.

That reconstruction is an interpretation.

When the interpretation remains inside a transient answer, its mistakes may be temporary. When it becomes a saved configuration, an executed action, or a persistent memory, the consequences last longer. The more autonomy an agent receives, the more important it becomes to separate what the user said from what the system inferred.

This is why the best agent interface cannot simply be a blank text box followed by confident execution.

It needs ways to make interpretation inspectable without forcing the user back into the full labor of manual configuration. It must know when to proceed, when to summarize its understanding, when to expose the structure it created, and when uncertainty justifies another question.

Human-AI interaction research has repeatedly returned to these needs. Guidelines proposed by Amershi and colleagues include making capabilities and limitations clear, supporting efficient correction, showing contextually relevant information, and allowing users to dismiss or adjust unwanted behavior.[6] Horvitz's work on mixed initiative similarly treats autonomy not as a binary switch, but as a decision governed by uncertainty, expected value, and the cost of being wrong.[7]

These principles become more important, not less, when the interface appears effortless.

An agent that does more work on the user's behalf must also make it easier for the user to understand, revise, and reverse that work.

From controls to commitments

The design questions of an editor were largely spatial.

Where should the control appear? What should it be called? Which options belong together? What should be visible by default? How can the user preview the result?

The design questions of an agent are increasingly epistemic.

What does the system believe the user wants? Which details came directly from the user, and which were inferred? How confident is it? What information should persist? What should be forgotten? Which action requires confirmation? When does personalization become an assumption that is difficult to escape?

The unit of design shifts from the control to the commitment.

A field is a visible commitment. Its label tells the user what kind of information belongs there. A toggle is a visible commitment. Its state can be inspected and changed. A condition in a visual editor is a visible commitment. The user can see that one event will trigger another.

An agent often creates commitments invisibly.

It may decide that "make it friendlier" means shorter sentences, warmer language, fewer warnings, or more enthusiasm. It may decide that "recommend the best option" means maximizing conversion, minimizing price, or matching a stated preference. It may remember a temporary preference as if it were stable. It may treat an example as a rule.

The future of product design depends on giving these commitments shape.

Not every internal decision needs a panel, field, or modal. Rebuilding the old editor around every inference would destroy the value of the new interface. But consequential interpretations need handles. They need summaries, previews, provenance, undo, correction, and sometimes explicit consent.

The interaction may begin in conversation, then become visual when precision matters. A user might describe a workflow in plain language, inspect the generated structure, adjust one branch directly, and return to conversation to explain a broader change. Research combining direct manipulation with programmatic systems has long argued that the two modes have complementary strengths: direct manipulation offers immediacy and feedback, while programmatic representation offers abstraction and reuse.[8] Language agents add a third mode, interpretation, but they do not erase the first two.

The strongest interfaces will not force a choice between talking and editing.

They will let each mode appear where it is strongest.

The last editor

The title of this essay is intentionally misleading.

There will not be a final editor after which editing disappears. People will continue to need precise surfaces for inspecting and shaping complex things. Spreadsheets, canvases, timelines, rule builders, code, and forms will remain useful because visibility is useful.

The last editor is not the last editing product.

It is the last moment at which the human is expected to perform all of the translation alone.

For most of software history, the machine waited at the end of a carefully structured sequence. The person had to arrive with the right command, the right fields, the right configuration, and the right representation. The editor helped, but it still placed the burden of legibility on the human.

Agents promise to move that boundary.

A person may begin with language as it actually occurs: incomplete, contextual, provisional, and alive. The system can help turn it into something operational. It can propose structure instead of merely demanding it. It can preserve intent across multiple representations. It can notice missing information and ask for it. It can carry context from one action into the next.

Recent agent architectures already treat memory, reflection, planning, and tool use as components of behavior rather than features of a single response.[9] That development makes the interface more than a place where instructions are entered. It becomes the site of an ongoing negotiation between human intention and machine interpretation.

The question is no longer whether software can understand natural language well enough to remove a few fields.

The question is whether it can assume more of the work of understanding without quietly taking ownership of what the user meant.

That is the line product designers will have to hold.

Designing systems that listen without rewriting us

Listening is not passive.

To listen is to select, organize, and interpret. A system that listens well does not merely retain more words. It builds a usable account of what those words were trying to accomplish. But because that account can be wrong, listening must remain accountable to the speaker.

This creates a simple design principle:

> The system may propose the structure, but the user must retain authority over the meaning.

Authority requires more than a confirmation dialog. It requires that the system preserve distinctions that are easy to collapse:

- what the user explicitly said;
- what the system inferred;
- what the system generated as a suggestion;
- what was accepted;
- what remains uncertain;
- what will be remembered;
- what action will follow.

These distinctions are the new equivalent of labeled fields and visible controls. They are not always parts of the screen. They are properties of the relationship between the person and the system.

Human-centered AI has argued for systems that increase capability while preserving meaningful human control.[10] In agentic products, control will not mean manually approving every step. That would reduce autonomy to bureaucracy. It will mean being able to see and change the commitments that matter, especially when the system's interpretation begins to shape future behavior.

A good agent should reduce the work of operating software without reducing the person's ability to recognize themselves in the result.

It should structure without distorting.

It should infer without pretending that inference is fact.

It should remember without turning a temporary statement into a permanent identity.

It should act without making its own convenience the definition of the user's intention.

This is a harder product problem than adding a chat box to an existing application. It requires rethinking where structure lives, when it becomes visible, and who has the final authority to revise it.

The interface changes sides

I spent years helping people learn the logic of editors.

The work taught me to respect the forms, fields, conditions, and controls that make a system dependable. It also showed me how much effort ordinary software asks from the people using it. Every support ticket carried a small record of failed translation: someone knew what they wanted, but the path from intention to structure was harder than it should have been.

Language agents make another arrangement possible.

The user can begin closer to thought. The system can move closer to structure. Between them, the interface becomes less like a form to complete and more like a negotiation to refine.

But the editor remains.

It is no longer only the visible surface where a person manipulates software objects. It is also the invisible process by which the system interprets a person, proposes a representation, and commits that representation to action.

Our task is not to eliminate that editor.

It is to make it worthy of trust.

The interface does not vanish.

It changes sides.

References

1. Shneiderman, B. (1983). “Direct Manipulation: A Step Beyond Programming Languages.” *Computer*, 16(8), 57–69. https://doi.org/10.1109/MC.1983.1654471↩
2. Horvitz, E. (1999). “Principles of Mixed-Initiative User Interfaces.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 159–166. https://doi.org/10.1145/302979.303030↩
3. Luger, E., & Sellen, A. (2016). “‘Like Having a Really Bad PA’: The Gulf between User Expectation and Experience of Conversational Agents.” In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 5286–5297. https://doi.org/10.1145/2858036.2858288↩
4. Reynolds, L., & McDonell, K. (2021). “Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm.” *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3411763.3451760↩

5. Zamfirescu-Pereira, J. D., Wong, R. Y., Hartmann, B., & Yang, Q. (2023). “Why Johnny Can’t Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts.” In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3544548.3581388↩

6. Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., & Horvitz, E. (2019). “Guidelines for Human-AI Interaction.” In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3290605.3300233↩

7. Horvitz, E. (1999). “Principles of Mixed-Initiative User Interfaces.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 159–166. https://doi.org/10.1145/302979.303030↩

8. Chugh, R., Hempel, B., Spradlin, M., & Albers, J. (2016). “Programmatic and Direct Manipulation, Together at Last.” In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 341–354. https://doi.org/10.1145/2908080.2908103↩

9. Park, J. S., O’Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). “Generative Agents: Interactive Simulacra of Human Behavior.” In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. https://doi.org/10.1145/3586183.3606763↩

10. Shneiderman, B. (2020). “Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy.” *International Journal of Human–Computer Interaction*, 36(6), 495–504. https://doi.org/10.1080/10447318.2020.1741118↩