O Último Editor

Depois de décadas ensinando pessoas a operar software, estamos construindo software que aprende a escutar.

Rafael M. Ehlers · 15 de julho de 2026
rafaelmehlers.com.br/essays/the-last-editor

Durante a maior parte da minha carreira, havia um editor no meio.

Na MailPoet, era um editor de newsletters. Na GravityKit, um editor de formulários. Na Octane AI, um editor de quizzes. Os produtos mudaram, os mercados mudaram e a tecnologia por baixo deles mudou, mas o arranjo básico permanecia familiar: uma pessoa chegava com uma intenção, e uma interface pedia que ela a traduzisse para a língua do software.

Escreva o assunto aqui. Adicione um campo ali. Escolha um layout. Defina uma condição. Conecte uma resposta a um resultado. Salve, visualize, publique.

O editor ficava entre o que a pessoa queria e o que o sistema conseguia entender.

No começo eu não percebi o padrão. Foram necessários anos de tickets de suporte, especificações de produto, releases, regressões e conversas com usuários frustrados até que a repetição se tornasse óbvia. Eu tinha passado boa parte da minha vida profissional dentro de variações do mesmo problema: como ajudar alguém a transformar uma ideia em estrutura sem que a estrutura parecesse um castigo.

Editores são produtos implacáveis. Toda aresta mal acabada aparece no fluxo de trabalho de alguém em questão de minutos. Um rótulo confuso não é apenas uma frase que poderia ter sido mais bem escrita. Ele vira uma campanha enviada com as configurações erradas, um formulário que não coleta a informação certa ou um fluxo de recomendação que produz o resultado errado. Editores expõem a distância entre o modelo mental da pessoa que usa o produto e o modelo interno do software. É nessa distância que vive o trabalho de produto.

O acordo do editor

O editor gráfico foi um dos grandes atos de tradução da computação.

Sistemas anteriores exigiam que as pessoas expressassem instruções por meio de comandos formais. As interfaces gráficas substituíram boa parte dessa sintaxe por objetos visíveis e ações reversíveis. Ben Shneiderman descreveu essa abordagem como manipulação direta: mantenha visíveis os objetos de interesse, deixe o usuário agir sobre eles de forma incremental e forneça feedback imediato sobre o resultado.[1] A ideia ficou tão comum que paramos de ver o quanto era radical. Documentos podiam ser editados mudando algo que parecia uma página. Arquivos

podiam ser movidos arrastando ícones entre pastas. Uma pessoa não precisava mais descrever cada operação na língua nativa da máquina.

Mas a manipulação direta redesenhou os termos da troca.

O usuário ainda precisava aprender o que o software considerava um objeto, quais ações estavam disponíveis, o que cada campo significava, como funcionava a hierarquia e qual sequência de operações produziria o resultado pretendido. O editor tornou o sistema visível, mas também tornou a ontologia do sistema inescapável.

Um editor de newsletters ensina que uma mensagem consiste em blocos, colunas, links, audiências e regras de envio. Um editor de formulários ensina que uma conversa pode ser representada como campos, validações, condições e entradas. Um editor de quizzes ensina que uma recomendação pode ser decomposta em perguntas, respostas, ramificações, pontuações e resultados.

Essas estruturas são úteis. São elas que tornam o software confiável, inspecionável e repetível. Elas também são trabalho.

O trabalho oculto de usar um editor é o trabalho de converter intenção em categorias que o software já sabe processar. A pessoa precisa decidir que parte de uma ideia pertence a qual campo. Precisa antecipar a interpretação do sistema antes de o sistema agir. Precisa aprender o produto bem o suficiente para pensar com ele.

Boa parte do design de produto tem sido, portanto, um esforço para tornar essa tradução menos dolorosa. Renomeamos rótulos, reorganizamos menus, reduzimos passos, adicionamos pré-visualizações, melhoramos padrões, expomos templates e removemos decisões que não precisam ser tomadas. Chamamos isso de usabilidade, mas por baixo existe uma ambição mais fundamental: encurtar a distância entre o que a pessoa quer dizer e o que a máquina exige.

Por décadas, essa distância foi tratada como uma propriedade da interface.

Agora ela está se tornando uma propriedade do modelo.

Quando a interface aprende a escutar

Modelos de linguagem introduzem um arranjo diferente.

Em vez de abrir a tela correta, encontrar o controle correto e traduzir uma intenção para uma estrutura predefinida, a pessoa pode descrever o que quer na língua que já usa. Pode incluir contexto, incerteza, exemplos, exceções e pensamentos pela metade. Espera-se que o sistema infira a tarefa, identifique os detalhes relevantes e produza algo estruturado o bastante para virar ação.

A direção da tradução começa a se inverter. No editor tradicional, a pessoa faz a maior parte do trabalho interpretativo. Ela estuda a interface e remodela a intenção até que caiba. Numa interface agêntica, pede-se que o sistema faça mais desse trabalho. Ele escuta um pedido não estruturado, interpreta o que importa, decide o que pode ser inferido, faz perguntas quando necessário e converte o resultado em ações.

A linguagem natural muda quem carrega o fardo da estrutura.

Essa possibilidade existe como aspiração há muito tempo. A pesquisa sobre interfaces de iniciativa mista descreveu sistemas em que humanos e software contribuem juntos para uma tarefa, com a iniciativa mudando de acordo com a incerteza, o custo e o contexto.[2] Agentes conversacionais prometeram um modelo de interação ainda mais familiar, mas os primeiros sistemas decepcionaram repetidamente os usuários, porque a conversa comum cria expectativas que um software frágil não consegue sustentar. Luger e Sellen descobriram que as pessoas costumavam abordar assistentes conversacionais como se fossem colaboradores capazes, e depois precisavam fazer a engenharia reversa de suas limitações por tentativa e erro.[3]

Grandes modelos de linguagem ampliam o que pode ser expresso e interpretado, mas não removem o problema. O prompt é, ele próprio, uma forma de programação, mesmo quando a linguagem de programação se parece com prosa comum.[4] Estudos com não especialistas trabalhando com modelos de linguagem mostraram que os usuários ainda têm dificuldade de formar modelos mentais confiáveis do que um modelo consegue fazer, de como as instruções devem ser formuladas e de por que pequenas mudanças às vezes produzem resultados radicalmente diferentes.[5]

A interface pode parecer mais natural, mas a tradução não desapareceu. Ela ficou menos visível.

O editor não desapareceu

Quando uma pessoa diz “crie um quiz que recomende a rotina de skincare certa”, o sistema ainda precisa produzir estrutura.

Ele precisa decidir o que conta como uma pergunta útil, quais respostas são significativas, como os resultados diferem entre si, que informação deve ser solicitada e como o resultado deve ser representado. Se cria o quiz dentro de um produto, precisa em algum momento gerar os mesmos tipos de objetos que um editor teria exposto: perguntas, opções de resposta, condições, pontuações, blocos de conteúdo e regras de recomendação.

O esquema continua lá. A diferença é que o usuário pode não ver mais o momento em que sua intenção é traduzida para ele.

O editor pode desaparecer da superfície enquanto se torna mais poderoso por baixo. O sistema escolhe categorias em silêncio, resolve ambiguidades, fornece padrões, comprime contexto e

decide quais partes do pedido merecem uma representação durável.

Todo agente é, portanto, um editor.

Ele edita a linguagem do usuário num relato interno do que o usuário quis dizer. Seleciona alguns detalhes e descarta outros. Transforma possibilidades em parâmetros. Interpreta tom, prioridade e restrições implícitas. Quando age, age com base nessa representação editada.

O perigo não é apenas que o sistema produza um resultado ruim. Ele pode produzir um bom resultado para a interpretação errada.

Editores tradicionais tornavam muitas suposições visíveis. O usuário podia inspecionar o tipo de campo selecionado, a condição, a lista de destinatários, o layout ou a regra de pontuação. Um sistema conversacional pode esconder essas suposições dentro de uma resposta fluente. A fluência faz a interpretação parecer resolvida antes de ter sido examinada.

O editor antigo criava atrito porque expunha a estrutura.

O editor novo cria risco porque consegue esconder a estrutura bem demais.

O problema da interpretação invisível

Um modelo de linguagem não recebe a intenção diretamente. Ele recebe evidências da intenção.

A distinção importa. Pedidos humanos são incompletos porque o pensamento humano é contextual. Omitimos o que parece óbvio. Nos corrigimos no meio da frase. Usamos a mesma palavra de formas diferentes em situações diferentes. Pedimos uma coisa enquanto otimizamos silenciosamente para outra. Um colaborador humano habilidoso detecta parte disso pelo contexto compartilhado e pelos esclarecimentos. Um modelo reconstrói isso a partir da linguagem, das interações anteriores, das instruções de sistema, das ferramentas disponíveis e de padrões estatísticos aprendidos.

Essa reconstrução é uma interpretação.

Enquanto a interpretação permanece dentro de uma resposta transitória, seus erros podem ser temporários. Quando ela se torna uma configuração salva, uma ação executada ou uma memória persistente, as consequências duram mais. Quanto mais autonomia um agente recebe, mais importante se torna separar o que o usuário disse do que o sistema inferiu.

É por isso que a melhor interface de agente não pode ser simplesmente uma caixa de texto em branco seguida de uma execução confiante.

Ela precisa de formas de tornar a interpretação inspecionável sem forçar o usuário de volta ao trabalho completo da configuração manual. Precisa saber quando prosseguir, quando resumir seu entendimento, quando expor a estrutura que criou e quando a incerteza justifica mais uma pergunta.

A pesquisa em interação humano-IA tem retornado repetidamente a essas necessidades. As diretrizes propostas por Amershi e colegas incluem deixar claras as capacidades e limitações, apoiar a correção eficiente, mostrar informação contextualmente relevante e permitir que os usuários descartem ou ajustem comportamentos indesejados.[6] O trabalho de Horvitz sobre iniciativa mista trata igualmente a autonomia não como um interruptor binário, mas como uma decisão governada pela incerteza, pelo valor esperado e pelo custo de errar.[7]

Esses princípios se tornam mais importantes, não menos, quando a interface parece não exigir esforço.

Um agente que faz mais trabalho em nome do usuário também precisa tornar mais fácil para o usuário entender, revisar e reverter esse trabalho.

De controles a compromissos

As questões de design de um editor eram, em grande parte, espaciais.

Onde o controle deve aparecer? Como deve se chamar? Quais opções pertencem ao mesmo grupo? O que deve estar visível por padrão? Como o usuário pode pré-visualizar o resultado?

As questões de design de um agente são cada vez mais epistêmicas.

O que o sistema acredita que o usuário quer? Quais detalhes vieram diretamente do usuário, e quais foram inferidos? Qual é o grau de confiança? Que informação deve persistir? O que deve ser esquecido? Qual ação exige confirmação? Quando a personalização se torna uma suposição da qual é difícil escapar?

A unidade de design se desloca do controle para o compromisso.

Um campo é um compromisso visível. Seu rótulo diz ao usuário que tipo de informação pertence ali. Um toggle é um compromisso visível. Seu estado pode ser inspecionado e alterado. Uma condição num editor visual é um compromisso visível. O usuário pode ver que um evento vai disparar outro.

Um agente frequentemente cria compromissos de forma invisível.

Ele pode decidir que “deixe mais amigável” significa frases mais curtas, linguagem mais calorosa, menos avisos ou mais entusiasmo, ou que “recomende a melhor opção” significa maximizar a conversão, minimizar o preço ou corresponder a uma preferência declarada. Uma preferência temporária pode ser lembrada como se fosse estável, um exemplo tratado como uma regra.

O futuro do design de produto depende de dar forma a esses compromissos.

Nem toda decisão interna precisa de um painel, um campo ou um modal. Reconstruir o editor antigo em torno de cada inferência destruiria o valor da nova interface. Mas interpretações

consequentes precisam de pegas. Precisam de resumos, pré-visualizações, proveniência, desfazer, correção e, às vezes, consentimento explícito.

A interação pode começar em conversa e depois se tornar visual quando a precisão importa. Um usuário pode descrever um fluxo de trabalho em linguagem simples, inspecionar a estrutura gerada, ajustar um ramo diretamente e voltar à conversa para explicar uma mudança mais ampla. A pesquisa que combina manipulação direta com sistemas programáticos argumenta há muito tempo que os dois modos têm forças complementares: a manipulação direta oferece imediatismo e feedback, enquanto a representação programática oferece abstração e reúso.[8] Agentes de linguagem adicionam um terceiro modo, a interpretação, mas não apagam os dois primeiros.

As interfaces mais fortes não vão forçar uma escolha entre conversar e editar.

Vão deixar cada modo aparecer onde ele é mais forte.

O último editor

O título deste ensaio é intencionalmente enganoso.

Não haverá um editor final depois do qual a edição desaparece. As pessoas continuarão precisando de superfícies precisas para inspecionar e moldar coisas complexas. Planilhas, canvases, linhas do tempo, construtores de regras, código e formulários continuarão úteis, porque a visibilidade é útil.

O último editor não é o último produto de edição.

É o último momento em que se espera que o humano faça toda a tradução sozinho.

Durante a maior parte da história do software, a máquina esperava no fim de uma sequência cuidadosamente estruturada. A pessoa tinha que chegar com o comando certo, os campos certos, a configuração certa e a representação certa. O editor ajudava, mas ainda colocava o fardo da legibilidade sobre o humano.

Agentes prometem mover essa fronteira.

Uma pessoa pode começar com a linguagem como ela realmente ocorre: incompleta, contextual, provisória e viva. O sistema pode ajudar a transformá-la em algo operacional. Pode propor estrutura em vez de apenas exigi-la, preservar a intenção através de múltiplas representações, notar a informação que falta e pedi-la, e carregar o contexto de uma ação para a seguinte.

Arquiteturas recentes de agentes já tratam memória, reflexão, planejamento e uso de ferramentas como componentes do comportamento, e não como recursos de uma única resposta.[9] Esse desenvolvimento torna a interface mais do que um lugar onde instruções são

digitadas. Ela se torna o palco de uma negociação contínua entre a intenção humana e a interpretação da máquina.

A pergunta não é mais se o software consegue entender linguagem natural bem o suficiente para remover alguns campos.

A pergunta é se ele consegue assumir mais do trabalho de entender sem tomar silenciosamente posse do que o usuário quis dizer.

Projetando sistemas que escutam sem nos reescrever

Escutar não é passivo.

Escutar é selecionar, organizar e interpretar. Um sistema que escuta bem não se limita a reter mais palavras. Ele constrói um relato utilizável do que aquelas palavras tentavam realizar. Mas como esse relato pode estar errado, a escuta precisa continuar prestando contas a quem fala.

Isso cria um princípio de design simples:

> O sistema pode propor a estrutura, mas o usuário deve manter a autoridade sobre o significado.

Autoridade exige mais do que um diálogo de confirmação. Exige que o sistema preserve distinções que são fáceis de colapsar:

- o que o usuário disse explicitamente;
- o que o sistema inferiu;
- o que o sistema gerou como sugestão;
- o que foi aceito;
- o que permanece incerto;
- o que será lembrado;
- que ação virá a seguir.

Essas distinções são o novo equivalente dos campos rotulados e dos controles visíveis. Nem sempre são partes da tela. São propriedades da relação entre a pessoa e o sistema.

A IA centrada no humano tem defendido sistemas que aumentam a capacidade enquanto preservam um controle humano significativo.[10] Em produtos agênticos, controle significa poder ver e mudar os compromissos que importam, especialmente quando a interpretação do sistema começa a moldar o comportamento futuro. Não exige aprovar cada passo, o que reduziria a autonomia a burocracia.

Um bom agente deve reduzir o trabalho de operar o software sem reduzir a capacidade da pessoa de se reconhecer no resultado.

Deve estruturar sem distorcer, e inferir sem fingir que inferência é fato. Deve lembrar sem transformar uma afirmação temporária numa identidade permanente, e agir sem fazer da própria conveniência a definição da intenção do usuário.

Esse é um problema de produto mais difícil do que adicionar uma caixa de chat a um aplicativo existente. Ele exige repensar onde a estrutura vive, quando ela se torna visível e quem tem a autoridade final para revisá-la.

A interface muda de lado

Passei anos ajudando pessoas a aprender a lógica dos editores.

O trabalho me ensinou a respeitar os formulários, campos, condições e controles que tornam um sistema confiável. Também me mostrou quanto esforço o software comum pede das pessoas que o usam. Cada ticket de suporte carregava um pequeno registro de uma tradução fracassada: alguém sabia o que queria, mas o caminho da intenção até a estrutura era mais difícil do que deveria ser.

Agentes de linguagem tornam outro arranjo possível.

O usuário pode começar mais perto do pensamento. O sistema pode se mover para mais perto da estrutura. Entre os dois, a interface se torna menos um formulário a preencher e mais uma negociação a refinar.

Mas o editor permanece.

Ele não é mais apenas a superfície visível onde uma pessoa manipula objetos de software. É também o processo invisível pelo qual o sistema interpreta uma pessoa, propõe uma representação e converte essa representação em ação.

Nossa tarefa não é eliminar esse editor.

É torná-lo digno de confiança.

Referências

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↩