

Acessibilidade em IDEs com IA: por que leitores de tela ainda encontram barreiras

Por Leonardo Gleison Ferreira*

A corrida das IDEs (Ambientes de Desenvolvimento Integrado Quadrados ou de Segunda Geração) com IA — [GitHub Copilot](#), [Cursor](#), [Windsurf](#), [Trae](#) e [Claude Code](#) — mudou o fluxo de trabalho de quem escreve software. Para desenvolvedores cegos e com baixa visão que dependem de leitor de tela e navegação por teclado, porém, a evolução veio acompanhada de uma regressão de acessibilidade: recursos criados para ampliar o acesso frequentemente introduzem novas barreiras de teclado, de foco e de compatibilidade com tecnologia assistiva. Este texto analisa onde essas barreiras aparecem, como se relacionam com a [WCAG 2.2](#) e o que já é possível fazer a respeito.

O que mudou em relação ao autocompletar tradicional

Antes dos LLMs, o autocompletar de código já era razoavelmente acessível. Completar com Tab, navegar sugestões com as setas e inspecionar parâmetros de funções funcionava de forma previsível com leitores como [NVDA](#), [JAWS](#) e [VoiceOver](#); a experiência de um desenvolvedor cego ficava, dentro do possível, próxima à de qualquer outro.

Uma dificuldade anterior aos LLMs e ainda presente é o volume de código. Leitores de tela apresentam a informação de forma sequencial, linha a linha e símbolo a símbolo. Entender uma classe de 500 linhas ou um pull request com 20 arquivos modificados sempre exigiu esforço cognitivo maior do que a leitura visual, ainda que estratégias como navegação estrutural e breakpoints tornem a tarefa viável.

Os LLMs atacam exatamente essa assimetria: resumir uma função inteira em uma frase, explicar a lógica de um bloco e responder perguntas sobre o código sem exigir a travessia linha a linha. O potencial é grande; a execução ficou aquém.

Onde as barreiras aparecem

IDEs construídas sobre o núcleo do VS Code — hoje a base mais acessível disponível, segundo a própria [documentação de acessibilidade da Microsoft](#) — deveriam preservar o mesmo nível de suporte. Na prática, várias não preservaram, e três padrões se repetem.

O primeiro é a ruptura da navegação por teclado. Atalhos padronizados deixam de funcionar ou são sobrescritos sem alternativa equivalente, e combinações específicas de IA entram em conflito com teclas reservadas dos leitores de tela, obrigando a alternar constantemente entre modo de navegação e modo de foco para operações básicas.

O segundo é a inacessibilidade dos componentes específicos de IA. Editores embutidos que exibem sugestões sensíveis ao contexto frequentemente carecem de papéis, rótulos e estados ARIA, criam armadilhas de foco e não anunciam mudanças de conteúdo. Painéis de chat sofrem com gerenciamento inadequado de foco e ausência de anúncio de novas mensagens. Popups de sugestão multilinha apresentam anúncios verbosos e contraste abaixo do mínimo exigido — algo verificável com ferramentas como o [Contrast Checker do WebAIM](#).

O terceiro é a mudança imprevisível de contexto: sugestões que aparecem automaticamente e alterações de estado ao aceitar ou rejeitar uma proposta desorientam quem depende da navegação sequencial.

Mapeadas aos critérios, as falhas concentram-se nos quatro princípios da WCAG 2.2 — Perceptível, Operável, Compreensível e Robusto. Em termos práticos, funcionalidades centrais de IA, como aceitar sugestões, abrir o chat ou dispensar popups, ficam inacessíveis apenas por teclado, o que viola diretamente o critério 2.1.1.

O déficit fica mais evidente em uma categoria que a corrida recente ampliou para além das IDEs. Ferramentas como [Codex](#), [Antigravity](#) e [GitHub Desktop](#) não são, a rigor, IDEs, mas colocam o mesmo tipo de interface de conversa no centro do fluxo de trabalho — e é justamente nessa interface que a experiência de quem usa leitor de tela menos avançou. Há progresso real na leitura do conteúdo recebido, com regiões dinâmicas que anunciam a resposta conforme ela chega; a leitura, porém, cobre só parte da interação. A navegação detalhada — percorrer mensagens anteriores, retomar um trecho, revisar o que a ferramenta propôs antes de aceitar — segue pouco previsível e ainda depende de contornos manuais.

LLMs como tecnologia de ampliação de acessibilidade

Apesar das deficiências de implementação, os LLMs possuem capacidades que, integradas corretamente, reduzem barreiras de forma significativa. A mais relevante é a sumarização e explicação de código: converter complexidade sintática em descrições semânticas em linguagem natural permite navegar por uma base de código em unidades conceituais, em vez de percorrer arquivos linha a linha. A isso somam-se a assistência à navegação contextual, respondendo a consultas semânticas como "onde a lógica de autenticação é implementada", e a adaptação personalizada, com ajuste dinâmico da verbosidade das descrições e detecção automática de conflitos entre atalhos de teclado e comandos de tecnologia assistiva.

O que pode ser feito

Para que esse potencial se realize, a acessibilidade precisa deixar de ser tratada como preocupação pós-lançamento. A funcionalidade central da IDE deve atender à WCAG 2.2 Nível AA antes do desenvolvimento de recursos de IA, e os componentes de IA devem nascer com metadados ARIA, operabilidade por teclado e compatibilidade com leitores de tela. Papéis apropriados, nomes acessíveis, gestão previsível de foco e regiões dinâmicas para respostas assíncronas são requisitos, não diferenciais. As convenções de teclado

devem ser preservadas e estendidas, com comandos remapeáveis e fáceis de descobrir, além da validação das combinações em relação aos conjuntos de [NVDA](#), [JAWS](#), [VoiceOver](#) e [Orca](#). O controle deve ficar com o usuário: verbosidade e frequência configuráveis das sugestões, alternâncias por recurso e perfis de modo foco que suprimem interrupções não críticas. Testes automatizados com [axe-core](#) e [Lighthouse CI](#) integrados à pipeline, somados a testes manuais com usuários de leitor de tela a cada release, fecham o ciclo.

É nessa lacuna que se posiciona o [Assistente](#), uma iniciativa de código aberto concebida por e para pessoas cegas. A ferramenta cumpre dois papéis simultâneos: oferece um caminho prático para quem precisa usar LLMs hoje, sem depender de que a interação das grandes plataformas amadureça primeiro, e funciona como demonstrador de referência, mostrando que navegação acessível em interfaces de conversa é viável — e servindo de parâmetro para que empresas desenvolvam soluções de interação mais robustas para esse público.

Vale registrar também a dimensão organizacional, porque a maior barreira para acessibilidade em software raramente é técnica. O mercado de IDEs com IA vive uma disputa aberta, em que todas as empresas correm para inovar e capturar mercado, e o público de desenvolvedores cegos é pequeno demais para competir, no backlog, com a urgência de um recurso novo que disputa liderança de mercado — enquanto a barreira que exclui esse público é invisível justamente para quem decide o que entra no sprint. Não é má-fé — é a dinâmica de mercado operando como se sabe que opera. O risco é que, nessa corrida pela liderança em IA, ou por qualquer coisa que venha a substituir as IDEs, quem depende de tecnologia assistiva fique para trás: uma tecnologia com potencial enorme de ampliar a acessibilidade acaba entregue de forma apenas parcialmente acessível. Do lado da implementação, a maioria das correções é trivial — papel, rótulo, ordem de tabulação, contraste, descrição de imagem. O gargalo não está no código, e sim em tornar o problema visível para quem decide.

Considerações finais

A capacidade técnica de construir IDEs com IA inclusivas existe hoje. Converter representações visuais-espaciais de código em descrições sequenciais e semânticas é precisamente o que os LLMs fazem bem, e isso ataca a assimetria cognitiva entre compreensão visual e compreensão por leitor de tela. O que falta é compromisso organizacional para priorizar acessibilidade lado a lado com inovação. Quando isso acontece, os ambientes resultantes não servem melhor apenas aos desenvolvedores cegos: ganham usabilidade, flexibilidade e resiliência para todos.

* *Leonardo Gleison Ferreira é engenheiro de software no [Nubank](#), consultor em tecnologia assistiva para pessoas com deficiência visual e criador de conteúdo no [Canal Inclunet](#) — projeto de acessibilidade e inclusão que mantém, com a professora Camila Domingues, desde 2020. Cego, dedica a carreira a tornar a tecnologia acessível, tema que também aborda em inclunet.com.br.*