Diretrizes de Acessibilidade em Ambientes de Desenvolvimento Integrado para Estudantes com Deficiência Visual

SUMÁRIO

1. INTRODUÇÃO

Este documento apresenta um conjunto de recomendações de acessibilidade para Ambientes de Desenvolvimento Integrado (IDEs[1]) direcionadas a estudantes com deficiência visual. As recomendações têm o objetivo de orientar o projeto de IDEs de modo a torná-las apropriadas e adequadas às necessidades desses usuários. É importante considerar que pessoas com deficiência visual interagem com os dispositivos digitais de maneira distinta dos demais usuários, visto que a tela e o mouse podem não ser úteis para eles (Geraldo, 2016).

As recomendações foram elaboradas considerando as características e demandas dos estudantes com deficiência visual, principalmente cegos. Entretanto, desenvolver interfaces que atendam às diferentes necessidades desses usuários não é uma tarefa trivial, pois uma dificuldade percebida por uma pessoa com determinada deficiência não significa necessariamente ser a mesma barreira enfrentada por outras pessoas com a mesma deficiência (Da Silva, Ferreira & Ramos, 2016).

A ênfase destas recomendações está em IDEs voltadas para programação baseada em texto. IDEs desenvolvidos para programação em blocos ou híbridos não são abrangidas neste documento. Além disso, a pesquisa concentra-se especificamente nos desafios enfrentados por usuários que utilizam leitores de tela como recurso de Tecnologia Assistiva (TA) para interagir com IDEs.

Essa abordagem foi escolhida após a constatação de que leitores de tela se destacam como o principal recurso de TA utilizado por estudantes com deficiência visual de cursos de Computação (Zen, da Costa & Tavares, 2023). Dentre as principais limitações identificadas está o fato de que os leitores de tela percorrem interface gráfica do IDE e o código-fonte de forma linear, obrigando o usuário a percorrer o código linha por linha, em sequência (Albusays, Ludi & Huenerfauth, 2017), (Mountapmbeme, Okafor & Ludi, 2022). Esse método de interação dificulta o uso por pessoas com deficiência visual (Albusays & Ludi, 2016), resultando em vários problemas de interação, os quais serão abordados em cada uma das Diretrizes apresentadas nesse documento.

2. ESTRUTURA E ORGANIZAÇÃO DAS DIRETRIZES

As Diretrizes foram classificadas em 8 Categorias, que representam o conjunto de barreiras ou limitações encontradas por pessoas com deficiência visual, quais sejam:

- **Categoria 1. Compreensão do código**: relacionada às barreiras encontradas por pessoas com deficiência visual para ler e resumir um código-fonte utilizando leitores de tela.
- **Categoria 2. Depuração de código**: compreende as barreiras para identificar falhas no software, compreender a origem da falha e determinar a melhor maneira de removê-la ou corrigi-la.
- **Categoria 3. Navegação no código**: abrange questões relacionadas à leitura sequencial, linha por linha, realizada pelos leitores de tela, necessidade de procurar informações no

[1] *Integrated Development Environment*

código sem perder a posição de foco do cursor, navegação no ambiente de programação e localização dos recursos disponibilizados pela ferramenta de desenvolvimento.

- **Categoria 4. Edição de código**: relacionada às barreiras encontradas durante a escrita do código-fonte, tais como: uso de identação para indicar nível de escopo; utilização de cores diferentes para realçar a sintaxe do código; escrita correta dos comandos; uso extensivo de caracteres não alfanuméricos; e, complexidade da sintaxe das linguagens de programação.
- **Categoria 5. *Skimming*[2] de código**: aborda questões envolvidas no processo de obter uma visão geral de alto nível do código e identificar recursos como dobra de código.
- **Categoria 6. Interface Gráfica de Usuário**: compreende as dificuldades encontradas na realização de tarefas de desenvolvimento que envolvem a construção de algum componente visual.
- **Categoria 7. Sobrecarga auditiva**: relacionada ao excesso de informações transmitidas via canal de áudio em uma interface.
- **Categoria 8. Leitura de código-fonte**: relacionada à pronúncia das palavras-reservadas das linguagens de programação e das mensagens de erro.

Cada Categoria contém uma ou mais Diretrizes que oferecem recomendações para orientar o desenvolvimento de IDEs acessíveis para pessoas com deficiência visual. No total, são 21 Diretrizes contendo recomendações que podem ser aplicadas durante o processo de concepção, desenvolvimento e avaliação dessas aplicações. As Diretrizes adotam o seguinte formato:

- **Descrição da Diretriz**: apresenta um resumo da recomendação;
- **Critérios de sucesso**: declarações testáveis que visam determinar objetivamente se o conteúdo satisfaz a Diretriz;
- **Por que fazer?** Descreve a importância da orientação para reduzir barreiras de interação para pessoa com deficiência visual e quais as dificuldades de interação que ela pode auxiliar a mitigar;
- **Como fazer?** Contém sugestões de implementação da recomendação, elaboradas a partir das soluções já implementadas extraídas dos trabalhos obtidos na literatura; e,
- **Fonte**: referências bibliográficas utilizadas como base para a elaboração das Diretrizes e para sugerir como implementá-las.

3. DIRETRIZES

CATEGORIA 1: Compreensão do Código

Diretriz 1.1 Semântica

Descrição
Comunicar o significado lógico (semântica) dos elementos que constituem uma base de código.

[2] Não foi encontrada uma tradução adequada para o termo "*Skimming*", optando-se por mantê-lo em língua inglesa.

Critérios de sucesso

Critério de sucesso 1.1.1: É possível navegar pelo explorador de pacotes utilizando as teclas de seta.

Critério de sucesso 1.1.2: A ordem de navegação pelo explorador de pacotes reflete a estrutura lógica do programa.

Critério de sucesso 1.1.3: Ao navegar pelo explorador de pacotes de um projeto, são fornecidas informações sobre o significado lógico de cada nó dentro da hierarquia, bem como o relacionamento desse nó com seus irmãos ou pais.

Critério de sucesso 1.1.4: É possível obter informações contextuais de cada função ou método de um arquivo de código-fonte:

- Finalidade: descreve a funcionalidade que o método implementa;
- Qualificador: palavra-reservada das linguagens de programação que define a visibilidade de um determinado método, podendo assumir valores como "*public*", "*private*" ou "*protected*", por exemplo;
- Tipo de retorno: tipo de dado retornado por um método após a sua execução; e,
- Parâmetros: informações que podem ser passadas para um método quando ele é chamado.

Critério de sucesso 1.1.5: É possível mover o cursor do teclado para o nó principal do explorador de pacotes, a qualquer momento e independentemente do nó que esteja selecionado pelo cursor do teclado.

Critério de sucesso 1.1.6: É possível pesquisar por um nó específico do explorador de pacotes.

Critério de sucesso 1.1.7: É possível obter informações sobre a localização de um nó específico, em relação à estrutura geral do explorador de pacotes.

Critério de sucesso 1.1.8: É possível obter informações sobre a distância do nó selecionado em relação ao nó raiz do explorador de pacotes.

Por que fazer?

A exploração da estrutura de um programa utilizando as teclas de seta para subir ou descer no explorador de pacotes é desafiadora para usuários com deficiência visual. O leitor de telas verbaliza os rótulos dos componentes gráficos, mas não transmite nenhuma informação sobre o significado lógico e o relacionamento de um nó com seus irmãos ou pais, pois os leitores de tela não são inerentemente capazes de comunicar a funcionalidade das visualizações hierárquicas para usuários com deficiência visual. Além disso, pessoas com deficiência visual também podem ter dificuldade para distinguir o relacionamento entre classes e subclasses em uma base de código, especialmente quando existem várias subclasses que herdam todas ou algumas das características da classe principal.

Como fazer?

1. Organizar a estrutura de um projeto de software e do código-fonte utilizando uma árvore de navegação hierárquica. Ao posicionar o cursor do teclado em um nó da árvore, o usuário pode utilizar atalhos de teclado para solicitar dicas de áudio e obter informações contextuais do nó e sobre o relacionamento hierárquico desse nó com os demais.
2. Fornecer informações de contexto sobre o nó atual em relação aos seus pais e irmãos, incluindo uma descrição breve do papel do nó na estrutura global do programa.
3. A navegação pelos nós da árvore deve seguir a lógica da hierarquia da árvore e não a ordem na qual os elementos são listados dentro da hierarquia da árvore.
4. Fornecer atalhos de teclado para:
    - pesquisar por um nó específico na árvore hierárquica;
    - mover o cursor do teclado para o nó principal da árvore hierárquica;
    - mover o cursor para os nós ascendentes, irmãos e descendentes de um nó específico; e,
    - alterar o contexto relativo em relação a um novo ponto de ancoragem.

5. Utilizar alertas sonoros para informar:
    - a localização de um nó específico;
    - a distância de um nó em relação à raiz da árvore;
    - quais são os nós primos de um nó selecionado; e,
    - a densidade de uma subárvore dentro da hierarquia;

IMPORTANTE: As orientações presentes nessa Diretriz precisam ser consistentes com as recomendações feitas na Diretriz 5.2 (Visão Geral de Código).

Fonte
(Hutchinson & Metatla, 2018)
(Mealin & Murphy-Hill, 2012)
(Schanzer, Bahram & Krishnamurthi, 2019)
(Smith et al., 2003)

CATEGORIA 2: Depuração de Código

Diretriz 2.1 Informações sobre erros

Descrição
Informar o usuário sobre a existência de erros no código.

Critérios de sucesso
Critério de sucesso 2.1.1: Os erros de sintaxe são informados em tempo real.

Critério de sucesso 2.1.2: É exibida uma lista dos erros detectados após compilar o código-fonte.

Critério de sucesso 2.1.3: O usuário é informado quando não forem encontrados erros após a compilação de um código-fonte.

Critério de sucesso 2.1.4: As mensagens de erro contêm informações a respeito da causa do erro, sua localização no código-fonte e como corrigi-lo.

Por que fazer?
IDEs fornecem a maioria das informações sobre erros por meio de dicas visuais, como cores para realçar a sintaxe ou ícones que exibem informações sobre como corrigir o erro ao passar o mouse sobre eles. Esses recursos não estão disponíveis para desenvolvedores com deficiência visual de forma proativa. Esses usuários podem enfrentar desafios ao identificar, por exemplo, a ausência, excesso ou posicionamento inadequado de caracteres, operadores e parênteses, uma vez que não existem mecanismos de suporte em tempo real alternativos para o reconhecimento de erros durante a escrita de código. Além disso, as mensagens de erro podem não ser claras o suficiente para permitir que o usuário compreenda corretamente a mensagem que está sendo transmitida, custando mais tempo para realização da depuração.

Como fazer?
1. Utilizar sinais de áudio para informar sobre a ocorrência erros de sintaxe no código-fonte em tempo real.
2. Fornecer atalhos para listar os erros identificados no código e mover o cursor para a linha que contém o erro.
3. Utilizar alertas sonoros para informar quando não for encontrado nenhum erro após a compilação de um código-fonte.

4. Fornecer mensagens de erro mais claras e detalhadas, que informem ao usuário a causa do erro, sua localização no código-fonte e como corrigi-lo.

IMPORTANTE: A orientação presente no Critério de sucesso 2.1.4 precisa ser consistente com o Critério de sucesso 8.2.2 (As mensagens de erro são clara e facilmente compreendidas), da Diretriz 8.2 (Leitura contextual).

Fonte
(Albusays & Ludi, 2016)
(Petrausch & Loitsch, 2017)
(Potluri et al., 2018)
(Sampath, Merrick & Macvean, 2021)
(Zen, da Costa & Tavares, 2023)

Diretriz 2.2 Variáveis e constantes

Descrição
Fornecer informações sobre variáveis e constantes declaradas no código-fonte.

Critérios de sucesso
Critério de sucesso 2.2.1: É possível obter informações como nome, valor, tipo, qualificador e escopo de variáveis e constantes.

Por que fazer?
Informações sobre nomes, valores, tipos e qualificadores de variáveis não estão facilmente disponíveis para uma pessoa com deficiência visual. É necessário percorrer o código manualmente para consultar essas informações. Entretanto, usuários de leitores de tela não conseguem examinar o código sem mover o cursor também. Para encontrar algo como a grafia do identificador de uma variável, é necessário mover o cursor até o local onde a variável está declarada para que o leitor de tela verbalize a informação. Depois que as informações forem obtidas, o usuário do leitor de tela deve mover o cursor de volta para onde estava editando.

Como fazer?
1. Fornecer uma lista resumida dos nomes, valores, tipos, qualificadores e escopo de variáveis e constantes declaradas em um arquivo de código-fonte, acessível por meio de atalhos de teclado.
2. Fornecer atalhos de teclado para obter informações sobre valor, tipo, qualificador e escopo de uma variável ou constante que estiver atualmente selecionada pelo cursor do teclado.

Fonte
(Mealin & Murphy-Hill, 2012)
(Potluri et al., 2018)

Diretriz 2.3 Marcadores e Pontos de interrupção

Descrição
Permitir a utilização de marcadores ou pontos de interrupção em um arquivo de código-fonte.

Critérios de sucesso
Critério de sucesso 2.3.1: É possível identificar, em tempo real, os marcadores ou pontos de interrupção declarados em um arquivo de código-fonte.

Critério de sucesso 2.3.2: É possível listar todos os marcadores ou pontos de interrupção inseridos em um arquivo de código-fonte.

Critério de sucesso 2.3.3: É possível inserir marcadores ou pontos de interrupção em um arquivo de código-fonte.

Por que fazer?
Pontos de interrupção são marcadores definidos pelo programador no código-fonte para interromper a execução normal do código em um determinado ponto, permitindo que se examine o estado do programa e depure eventuais problemas. Informações sobre marcadores ou pontos de interrupção não estão facilmente disponíveis para uma pessoa com deficiência visual. Há situações, por exemplo, em que o desenvolvedor com deficiência visual chega a uma linha de código desconhecida devido a pontos de interrupção ou exceções, ou simplesmente porque o desenvolvedor estava distraído.

Como fazer?
1. Utilizar sinais de áudio para informar proativamente os programadores sobre marcadores ou pontos de interrupção inseridos no código.
2. O usuário pode utilizar atalhos de teclado para obter uma listagem dos marcadores ou pontos de interrupção existentes em um arquivo de código-fonte. Ao selecionar um marcador ou ponto de interrupção, o usuário pode optar por acessar informações que descrevem esse ponto de interrupção: localização no código, escopo e descrição.
3. Permitir que o usuário utilize atalhos para marcar locais específicos no código-fonte durante a depuração e adicionar descrições personalizadas para documentar o motivo pelo qual o marcador ou ponto de interrupção foi definido.

Fonte
(Potluri et al., 2018)

CATEGORIA 3: Navegação no Código

Diretriz 3.1 Rótulos

Descrição
Fornecer descrições textuais suficientemente claras para todo conteúdo não textual.

Critérios de sucesso
Critério de sucesso 3.1.1: Todo conteúdo não textual exibido ao usuário possui uma descrição textual alternativa clara para transmitir o seu significado e o destino ou a funcionalidade associada.

Por que fazer?
O texto alternativo é o melhor formato para descrever os elementos visuais que compõem a interface gráfica, como links e ícones, para os usuários com deficiência visual. Pessoas com deficiência visual confiam em nomes significativos porque, caso contrário, obterão apenas a informação de que os elementos gráficos estão presentes na interface, mas podem não compreender qual funcionalidade é oferecida.

Como fazer?
1. Fornecer descrições de texto alternativas para conteúdo não textual (imagens, gráficos e outros elementos visuais), visíveis ou não, permitindo que os usuários com deficiência visual tenham acesso a todas as informações por meio de leitores de tela.

Fonte
(ATAG, 2015)
(ISO/IEC, 2018)
(Petrausch & Loitsch, 2017)
(Potluri et al., 2018)
(WCAG, 2018)

Diretriz 3.2 Movimentação pelo código

Descrição
Oferecer recursos alternativos para navegar pela base de código.

Critérios de sucesso
Critério de sucesso 3.3.1: É possível deslocar o cursor do teclado para o início, o meio ou o fim de um arquivo de código-fonte.

Critério de sucesso 3.3.2: É possível mover o cursor do teclado de volta para a última linha do código-fonte que tenha sido editada.

Critério de sucesso 3.3.3: É possível localizar e mover o cursor do teclado para posições específicas (linhas de código ou trechos de código) em um arquivo de código-fonte.

Critério de sucesso 3.3.4: É possível marcar linhas de código ou trechos de código em um arquivo de código-fonte para serem consultados posteriormente.

Por que fazer?
Para se movimentar no código, IDEs limitam os programadores a algumas ações:

- Usar as teclas de seta para percorrer cada linha do código-fonte. Pode ser difícil e demorado, por exemplo, retornar rapidamente para uma linha de código específica ao revisar instruções do código-fonte, pois os usuários de leitores de tela não conseguem examinar o código sem mover o cursor também. Essa limitação pode acarretar inserção, exclusão ou alteração de código em locais incorretos, ou não intencionais.
- Usar o explorador de pacotes para navegar até um método ou arquivo de código-fonte. Depois que o arquivo ou método é selecionado, é necessário navegar pelo código de forma sequencial, linha por linha.
- Usar um mecanismo de pesquisa. É menos eficiente e mais difícil para um usuário com deficiência visual localizar informações específicas em uma base de código, limitando-se à funcionalidade de pesquisa e a alguns outros recursos de navegação oferecidos pelo IDE. A realização de busca utilizando palavras-chave pode ser demorada e frustrante, já que uma palavra-chave pode aparecer em vários locais na mesma base de código.

Como fazer?
1. Poderia ser utilizado um recurso de árvore de navegação hierárquica que possibilite navegar entre as funções e métodos declarados no arquivo de código atual.
2. Atalhos de teclado poderiam ser utilizados para deslocar o cursor do teclado para o início, para o meio ou para o fim de um arquivo de código-fonte. Atalhos de teclado também poderiam ser utilizados para deslocar o cursor do teclado para uma linha de código específica ou para a última linha de código que tenha sido editada.
3. Utilizar marcadores ou *tags* em locais específicos do código-fonte para marcar uma instrução de código e retornar a ela posteriormente para modificações adicionais. É recomendado utilizar o recurso de marcador para saltar para uma instrução de código específica, ao invés de um número de linha específico, porque a numeração das linhas de código pode ser alterada quando código adicional é inserido ou excluído. Além disso, seria interessante que se pudesse classificar esses marcadores como

"públicos" ou "privados". Marcadores privados só seriam acessíveis às pessoas que os adicionaram, enquanto marcadores públicos poderiam ser acessados por todos os envolvidos em um projeto.

Fonte
(Albusays & Ludi, 2016)
(Albusays, Ludi & Huenerfauth, 2017)
(Alotaibi, Al-Khalifa & Alsaeed, 2020)
(Baker, Bennett & Ladner, 2019)
(Mealin & Murphy-Hill, 2012)
(Potluri et al., 2018)

Diretriz 3.3 Contexto e nível de escopo

Descrição
Informar o usuário a respeito do contexto e do nível de escopo de uma determina linha de código ou trecho de código.

Critérios de sucesso
Critério de sucesso 3.3.1: É possível obter informações a respeito do nível de escopo de uma variável, constante ou instrução de fluxo de controle inserida no código-fonte.

Critério de sucesso 3.3.2: É possível obter informações a respeito do início e do fim de uma instrução de fluxo de controle ou bloco de código.

Critério de Sucesso 3.3.3: É possível verificar o nível de profundidade de aninhamento de uma instrução de fluxo de controle ou bloco de código.

Por que fazer?
Pode ser difícil para pessoas com deficiência visual compreender o nível de escopo enquanto navegam por estruturas profundamente aninhadas, bem como detectar o início e o fim de um bloco de código. Os recursos visuais utilizados pelos IDEs para auxiliar nesse processo, como o alinhamento de caracteres especiais ou recuos que marcam o início e o fim de bloco de código, não são facilmente acessíveis aos leitores de telas.

Como fazer?
1. Utilizar uma estrutura semelhante aos níveis de cabeçalho de um arquivo HTML para identificar informações estruturais como classes, métodos e instruções de fluxo de controle. O nível do cabeçalho identificaria o nível de escopo de uma determinada linha de código e sua localização relativa no código.
2. Utilizar diferentes alertas sonoros para representar o início ou o fim de uma instrução de fluxo de controle ou bloco de código.
3. Disponibilizar recurso de indicador de nível de aninhamento e escopo, que leria em voz alta a localização atual do cursor quando uma combinação especial de teclas de atalho fosse pressionada.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)
(Armaly, Rodeghero & McMillan, 2018)
(Baker, Bennett & Ladner, 2019)
(Huff et al., 2020)
(Hutchinson & Metatla, 2018)
(Zen, da Costa & Tavares, 2023)

Diretriz 3.4 Estratégias de navegação

Descrição
Fornecer diferentes formatos de organização e navegação pelo conteúdo sem perder informação.

Critérios de sucesso
Critério de sucesso 3.4.1: É possível escolher entre interagir com uma interface gráfica ou uma interface simplificada, com um *layout* baseado em texto e menus organizados em níveis.

Critério de sucesso 3.4.2: É possível monitorar processos executando em segundo plano.

Critério de sucesso 3.4.3: É possível determinar a localização do usuário dentro da hierarquia de menus e funcionalidades disponibilizadas pelo sistema.

Critério de sucesso 3.4.4: Todos os recursos e funcionalidades do IDE podem ser acessados e utilizados por meio do teclado.

Critério de sucesso 3.4.5: Todos os recursos e funcionalidades do IDE podem ser lidos pelos leitores de tela.

Critério de sucesso 3.4.6: O usuário é notificado quando o foco do cursor do teclado é movido para o console do IDE ou para a interface gráfica gerada após a compilação do código-fonte, e vice-versa.

Por que fazer?
As interfaces dos IDEs concentram-se quase que exclusivamente em estímulos visuais, utilizando estruturas de menus encapsuladas, caixas de diálogo, ícones e outros elementos gráficos e são difíceis de interagir por meio de leitores de tela. O acesso a essas informações utilizando leitor de tela depende do foco do cursor, ou seja, o usuário precisa buscar ativamente informações em vários componentes do IDE e definir explicitamente o foco no painel apropriado, sem a possibilidade de analisar as ferramentas periféricas. Além disso, alguns recursos que estão abertamente visíveis na interface gráfica do IDE, podem ficar ocultos dentro de vários níveis de hierarquia de navegação e são difíceis de acessar quando se utiliza leitores de tela. Em algumas situações, o desenvolvedor com deficiência visual pode não estar ciente da presença de um painel contendo as informações que procura.

Como fazer?
1. Fornecer atalhos para que o usuário possa escolher entre interagir com uma interface gráfica ou uma interface simplificada, com um layout baseado em texto e menus organizados em níveis.
2. Criar uma estrutura de conteúdo lógica e bem-organizada, facilitando a navegação e a compreensão do conteúdo por usuários com deficiência visual.
3. Utilizar alertas sonoros para informar a finalização ou o resultado de algum processo que estiver executando em segundo plano.
4. Utilizar alertas sonoros para indicar a mudança do foco do cursor da IDE para o console ou para a interface gráfica gerada após a compilação do código-fonte, e vice-versa.

Fonte
(ATAG, 2015)
(Mealin & Murphy-Hill, 2012)
(Potluri et al., 2018)
(Stefik et al, 2009)
(WCAG, 2018)
(Zen, da Costa & Tavares, 2023)

Diretriz 3.5 Numeração de linhas

Descrição
Informar o número de uma determinada linha de código ou o intervalo numérico de um trecho de código.

Critérios de sucesso
Critério de sucesso 3.5.1: É possível obter informação sobre o número da linha de código em que o cursor do teclado está posicionado.

Critério de sucesso 3.5.2: É possível obter informação sobre o número da linha inicial e a linha final de um trecho de código.

Por que fazer?
Os números de linha, no editor de código-fonte dos IDEs, facilitam a navegação, a depuração, a colaboração e a manutenção do código-fonte, mas não são acessíveis aos leitores de tela.

Como fazer?
1. Fornecer atalhos de teclado para consultar o número de uma linha específica de código, bem como para consultar o número da linha inicial e o número da linha final de um determinado trecho de código.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)

Diretriz 3.6 Atalhos

Descrição
Fornecer funções e comandos facilmente operáveis por meio de teclado e leitor de tela.

Critérios de sucesso
Critério de sucesso 3.6.1: São disponibilizados atalhos para todas as funcionalidades do IDE.

Critério de sucesso 3.6.2: É possível consultar todos os atalhos para as funcionalidades do IDE.

Critério de sucesso 3.6.3: É possível configurar e alterar os atalhos.

Critério de sucesso 3.6.4: O usuário pode classificar seus atalhos favoritos para agilizar uma consulta posterior.

Por que fazer?
Determinar os equivalentes em teclado (atalhos) para as funcionalidades do IDE pode ser complexo para pessoas com deficiência visual, pois não existe uma forma acessível de consultar essas informações na própria ferramenta, tornando necessário solicitar auxílio de outras pessoas ou realizar uma busca externa. Por vezes, tanto o IDE quanto o leitor de telas utilizam um mesmo conjunto de teclas para funcionalidades diferentes, confundindo o usuário. Há casos em que IDEs diferentes utilizam atalhos diferentes para acessar a mesma funcionalidade. Além disso, a memorização dos atalhos do IDE pode ser desafiadora para usuários com deficiência visual.

Como fazer?
1. Disponibilizar um guia de referência rápida para os atalhos utilizados no IDE permitindo que o usuário consulte os atalhos associados a funcionalidades específicas.
2. Informar automaticamente os atalhos correspondentes a uma determinada funcionalidade quando o cursor do teclado estiver posicionado sobre o seu respectivo ícone ou link.
3. Oferecer meios para que o usuário possa personalizar os atalhos, de acordo com suas preferências.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)
(Baker, Bennett & Ladner, 2019)
(Petrausch & Loitsch, 2017)
(Potluri et al., 2018)
(Zen, da Costa & Tavares, 2023)

Diretriz 3.7 Ajuda e Documentação

Descrição
Fornecer diferentes formas de ajuda e documentação ao usuário: dicas, tutoriais, recomendações ou exemplos.

Critérios de sucesso
Critério de Sucesso 3.7.1: Todas as principais funções são explicadas e instruídas e tutoriais e instruções são fáceis de acessar.

Critério de Sucesso 3.7.2: É fornecida uma descrição resumida de cada funcionalidade do sistema.

Critério de Sucesso 3.7.3: Toda documentação da IDE é acessível para programadores com deficiência visual.

Critério de Sucesso 3.7.4: A cada nova versão do IDE, o usuário é informado sobre funcionalidades adicionadas e/ou removidas.

Por que fazer?
Os tutoriais precisam ser acessíveis para pessoas com deficiência visual de melhorar a compreensão do conceito de visão e perspectiva dos IDEs. Da mesma forma, boas documentações das funcionalidades podem melhorar a eficiência e facilitar o uso dessas ferramentas.
Cada nova versão de um IDE pode adicionar novos recursos e as funcionalidades já existentes podem sofrer modificações. Muitas dessas mudanças são representadas visualmente e não existe uma abordagem estruturada para que os usuários com deficiência visual sejam informados sobre elas.

Como fazer?
1. É necessário projetar tutoriais especificamente voltados para as necessidades das pessoas com deficiência visual.
2. Cada funcionalidade do sistema deve vir acompanhada de uma descrição concisa de sua finalidade e instruções sobre como utilizá-la. Essas informações devem estar acessíveis por meio de atalhos de teclado.
3. Ao lançar uma nova versão de um IDE, deve-se informar o usuário sobre a inclusão de novos recursos e a implementação de modificações nas funcionalidades já existentes.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)
(Pandey et al., 2022)

CATEGORIA 4: Edição de Código

Diretriz 4.1 Espaçamento

Descrição
Informar a quantidade total de espaços em branco quando se utilizar linguagens de programação que utilizam esses mecanismos para indicar identação.

Critérios de sucesso
Critério de sucesso 4.1.1: Em linguagens que utilizam espaços em branco para indicar identação, esses espaços são verbalizados como uma lista completa de espaços em branco.

Por que fazer?
Algumas linguagens de programação usam recuo de espaço em branco para delimitar blocos de código, ao invés de usar palavras-chave ou chaves. Nessas linguagens, um aumento no recuo pode indicar um bloco de código novo e mais profundo, e uma diminuição no recuo indica o fim de um bloco de código. O leitor de tela realiza a leitura de todos os espaços em branco como uma sequência de caracteres de "espaço" individuais, em vez de um único recuo de um comprimento específico. Por exemplo, quando um usuário de leitor de tela navega por linguagens baseadas em indentação, o programador cego ouvirá seu leitor de tela verbalizando os espaços em branco individualmente (por exemplo, "espaço, espaço, espaço") em vez de uma contagem ("três espaços").

Como fazer?
1. Efetuar a contagem da quantidade de espaços em branco empregados para delimitar o início ou o término de um bloco de código e informar o total de espaços em branco ao usuário.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)
(Potluri et al., 2018)

Diretriz 4.2 Autocompletar

Descrição
Realizar a leitura das informações exibidas pelo recurso autocompletar.

Critérios de sucesso
Critério de sucesso 4.2.1: O usuário é informado a respeito da existência do recurso autocompletar a partir da posição de foco do teclado.

Critério de sucesso 4.2.2: O conteúdo do recurso autocompletar é acessível aos leitores de tela.

Critério de sucesso 4.2.3: O usuário consegue solicitar explicitamente a leitura do conteúdo exibido no recurso autocompletar.

Por que fazer?
Recursos como preenchimento automático (*autocomplete*), que auxiliam o programador a recordar a nomenclatura de classes, métodos e atributos, são exibidos por meio de um menu *pop-up* contendo previsões de digitação que podem ser realizadas. Esses recursos não são acessíveis aos leitores de tela.

Como fazer?

1. Utilizar alertas sonoros para notificar o usuário com deficiência visual sobre a presença do recurso autocompletar, na posição de foco do teclado.
2. Fornecer atalhos de teclado para que o usuário possa solicitar que seja realizada a leitura do conteúdo do recurso autocompletar.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)
(Zen, da Costa & Tavares, 2023)

Diretriz 4.3 Comentários

Descrição
Fornecer informações sobre comentários no código-fonte.

Critérios de sucesso
Critério de sucesso 4.3.1: O usuário é informado, em tempo real, sobre a existência de comentário na posição atual de foco do teclado.

Critério de sucesso 4.3.2: O usuário pode optar entre realizar a leitura ou ignorar um comentário no código.

Critério de sucesso 4.3.3: O usuário pode classificar os comentários.

Por que fazer?
Comentários são utilizados para tornar um código-fonte mais legível, facilitar tarefas de manutenção e depuração, documentar o funcionamento de um determinado trecho de código, auxiliar a superar barreiras de navegação, localizar erros ou bugs no código, ou para destacar instruções de código que requerem revisão adicional. Entretanto, enquanto desenvolvedores com visão podem ignorar grandes comentários de código (como documentação e licenças), usuários de leitores de tela podem ter dificuldades para navegar até o final desses comentários.

Como fazer?
1. Utilizar mecanismos para informar o usuário sobre a existência de comentários em determinados trechos de código e comandos que possibilitem que o usuário, explicitamente, solicite que seja feita a leitura do comentário.
2. Fornecer meios de classificar comentários de código de acordo com a sua função:
   - tornar o código legível ou descrever uma funcionalidade;
   - localizar erros que precisam de correção; e,
   - destacar declarações de código que necessitam de revisões futuras.
3. Fornecer meios de classificar comentários de código de acordo com a sua visibilidade:
   - Públicos: que podem ser visualizados por qualquer membro da equipe de desenvolvimento; ou,
   - Privados: acessíveis apenas para o usuário que declarou o comentário.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)
(Potluri et al., 2018)

CATEGORIA 5: *Skimming* de Código

Diretriz 5.1 Dobra de Código

Descrição
Informar quando um trecho de código está dobrado (oculto) e pode ser expandido.

Critérios de sucesso
Critério de sucesso 5.1.1: O usuário é informado quando um trecho de código está dobrado (oculto) e pode ser expandido.
Critério de sucesso 5.1.2: O usuário pode expandir um trecho de código que estiver dobrado (oculto).
Critério de sucesso 5.1.3: O usuário pode ocultar (dobrar) um trecho de código.

Por que fazer?
O recurso de dobra de código permite ocultar e exibir seletivamente seções de um arquivo de código-fonte, permitindo que os programadores tenham uma visão melhor das instruções de código no entorno da região colapsada. Esse recurso não é acessível aos leitores de tela e os usuários com deficiência visual podem não ser capazes de identificar áreas ocultas no código ou não ter clareza sobre o uso desse recurso.

Como fazer?
1. Utilizar alertas sonoros para notificar o usuário imediatamente sobre a presença de um código oculto (dobrado) quando o cursor do teclado estiver posicionado sobre a linha correspondente a ele.
2. Fornecer atalhos de teclado para que o usuário possa dobrar (ocultar) ou expandir um trecho de código.

Fonte
(Petrausch & Loitsch, 2017)
(Potluri et al., 2018)

Diretriz 5.2 Visão geral do código

Descrição
Fornecer visão geral mais ampla da base de código.

Critérios de sucesso
Critério de sucesso 5.2.1: É possível obter a quantidade total de linhas de um arquivo de código-fonte.

Critério de sucesso 5.2.2: É possível obter uma lista de todos os arquivos de código-fonte de um projeto de software.

Critério de sucesso 5.2.3: É possível obter uma lista de todas as funções e métodos de um arquivo de código-fonte.

Critério de sucesso 5.2.4: É possível navegar entre os arquivos de código-fonte em um projeto de software.

Critério de sucesso 5.2.5: É possível navegar entre as funções e métodos de um arquivo de código-fonte.

Por que fazer?
Programadores com deficiência visual têm dificuldade para obter uma visão geral de alto nível de seu código porque os leitores de tela os restringem a ler o código linha por linha, em sequência. Ao contrário de usuários com visão, que podem obter uma visão geral da estrutura do código rolando rapidamente para cima e para baixo em uma página, os leitores de tela forçam os usuários com deficiência visual a ler um documento inteiro.

Como fazer?

1. Organizar a estrutura de um projeto de software e do código-fonte por meio de uma árvore de navegação hierárquica, listando todos os arquivos de código-fonte de um projeto, bem como todas as funções e métodos existentes em um arquivo de código-fonte. Deve ser possível navegar até o componente de código desejado pressionando a tecla ENTER.
2. No caso de código-fonte hospedado na internet, pode-se usar *tags* HTML para determinar informações estruturais, onde o tipo de *tag* de título (nível de cabeçalho) anexada a um elemento depende de sua localização relativa no código-fonte.
3. Por meio de atalhos do teclado, o usuário pode solicitar que o leitor de telas informe a quantidade total de linhas de um arquivo de código-fonte.

IMPORTANTE: As orientações presentes nessa Diretriz precisam ser consistentes com as recomendações feitas na Diretriz 1.1 (Semântica) e na Diretriz 2.2 (Variáveis e Constantes).

Fonte
(Mealin & Murphy-Hill, 2012)
(Potluri et al., 2018)

CATEGORIA 6: Interface Gráfica de Usuário (GUI[3])

Diretriz 6.1: Construção de Interface Gráfica

Descrição
Fornecer mecanismos para facilitar a construção de interfaces que utilizam componentes visuais.

Critérios de sucesso
Critério de sucesso 6.1.1: O usuário pode informar ao IDE à disposição que deseja posicionar os elementos na tela.
Critério de sucesso 6.1.2: As dimensões dos elementos gráficos são atualizadas automaticamente, a partir de configurações informadas pelo usuário.

Por que fazer?
Desenvolvedores com visão podem usar ferramentas automatizadas fornecidas pelas IDEs para arrastar e soltar os componentes de uma interface gráfica e criar um *layout* rapidamente. Como as interações do mouse não são acessíveis para pessoas com deficiência visual, normalmente eles precisam escrever todo o código para construir uma interface do usuário, necessitando de um tempo maior para realizar essa tarefa.

Como fazer?

1. Dividir a tela em três linhas (superior, intermediária e inferior) e três colunas (esquerda, central e direita), permitindo que o usuário escolha onde deseja posicionar os componentes gráficos nas diferentes áreas da tela, sem precisar medir ou usar tentativa e erro.
2. Disponibilizar gerenciadores de *layout*, ou seja, ferramentas que agrupam e organizam automaticamente os componentes da interface gráfica de acordo com restrições especificadas pelo desenvolvedor.
3. Oferecer ao usuário a capacidade de estabelecer critérios temáticos para a interface e disponibilizar paletas de cores que permitam que o usuário defina as combinações de cores que seriam adotadas pelos elementos adicionados à interface.

[3] *Graphical User Interface*

Fonte
(Kearney-Volpe & Hurst, 2021)
(Pandey et al., 2022)
(Siegfried, 2006)
(Zen, da Costa & Tavares, 2023)

Diretriz 6.2 Validação da interface gráfica

Descrição
Realizar validação automatizada da interface gráfica gerada a partir de um código-fonte.

Critérios de sucesso
Critério de sucesso 6.2.1: É possível verificar se a interface gráfica gerada atende aos critérios de acessibilidade especificados pela WCAG[4].

Critério de sucesso 6.2.2: São fornecidas informações espaciais como localização, altura e largura dos elementos da interface gráfica.

Critério de sucesso 6.2.3: É possível verificar a consistência relacionada ao espaçamento entre os elementos, bem como à utilização de fontes e cores.

Por que fazer?
Pessoas com deficiência visual têm dificuldades para compreender e verificar se a saída gráfica gerada um código-fonte atende às necessidades e especificações desejadas. Essa dificuldade pode forçar os desenvolvedores com deficiência visual a solicitar auxílio de colegas ou procurar soluções alternativas para este fim. Além disso, a realização de tarefas que envolvem o desenvolvimento de interfaces de usuário faz com que muitos desenvolvedores com deficiência visual sintam que não são capazes de realizá-las com a mesma eficiência ou confiança que os desenvolvedores com visão. Por consequência, essas atividades são evitadas por pessoas com deficiência visual, que optam por se concentrarem no aspecto lógico do programa.

Como fazer?
1. Oferecer validadores de acessibilidade automatizados para verificar se a interface gráfica segue as recomendações estabelecidas na versão mais atual da WCAG.
2. Fornecer informações espaciais como localização, altura e largura dos elementos da interface gráfica utilizando unidades de medida selecionadas pelo usuário (pixels, centímetros ou milímetros, por exemplo).
3. Verificar a consistência entre o espaçamento horizontal e vertical entre elementos no mesmo nível.
4. Verificar se há variações significativas nas fontes utilizadas e se são aplicadas no máximo três tipos de fontes diferentes em uma página.
5. Verificar se elementos gráficos de mesmo tipo compartilham a mesma cor.
6. Emitir uma lista de violações de projeto encontradas que seja acessível aos leitores de tela.

Fonte
(ATAG, 2015)
(Baker, Bennett & Ladner, 2019)
(Huff et al., 2020)
(Mealin & Murphy-Hill, 2012)
(Pandey et al., 2022)
(Petrausch & Loitsch, 2017)
(Potluri et al., 2019)

[4] *Web Content Accessibility Guidelines*

(Siegfried, 2006)
(Zen, da Costa & Tavares, 2023)

CATEGORIA 7: Sobrecarga Auditiva

Diretriz 7.1 Alertas sonoros

Descrição
Possibilitar personalizar as configurações dos alertas sonoros utilizados.

Critérios de sucesso
Critério de sucesso 7.1.1: É possível configurar o volume dos alertas sonoros.

Critério de sucesso 7.1.2: É possível ativar ou desativar os alertas sonoros.

Critério de sucesso 7.1.3: É possível alterar o tipo de alerta sonoro associado a uma determinada funcionalidade.

Critério de sucesso 7.1.4: É dada preferência para alertas sonoros não verbais (*earcons*[5]) ao invés de alertas sonoros de fala (*spearcons*[6]).

Por que fazer?
Desenvolvedores com deficiência visual podem experimentar estresse decorrente de sobrecarga auditiva ao receber muitas informações transmitidas pelo canal de áudio de uma interface. Isso acontece porque além da leitura dos elementos da interface e código-fonte, os IDEs utilizam outros alertas sonoros (*earcons* e *spearcons*) para repassar informações aos usuários.

Como fazer?
1. A utilização de tons curtos, claros e distinguíveis é preferível para oferecer *feedback* adicional, como erros e avisos. Especialmente quando é necessário que esses alertas sejam emitidos ao mesmo tempo que é realizada a leitura de código-fonte, comentários ou entradas.
2. Deve-se implementar mecanismos que permitam ao usuário escolher entre alertas sonoros baseados em fala (*spearcons*) e não fala (*earcons*), bem como o tipo de som que será emitido.
3. Deve-se implementar mecanismos que permitam ao usuário controlar o volume do áudio, independentemente do nível global de volume do sistema operacional.
4. Deve-se implementar mecanismos que permitam ao usuário habilitar/desabilitar os alertas sonoros associados a alguma funcionalidade.

Fonte
(Albusays, Ludi & Huenerfauth, 2017)
(Petrausch & Loitsch, 2017)
(Zen, da Costa & Tavares, 2023)

[5] Mensagens de áudio não-verbais que são usadas na interface computador/usuário para fornecer informações ao usuário sobre algum objeto, operação ou interação do computador.
[6] Instruções faladas, comprimidas e aceleradas.

CATEGORIA 8: Leitura de Código-fonte

Diretriz 8.1 Idioma

Descrição
Alternar automaticamente entre as línguas faladas pelo leitor de telas.

Critérios de sucesso
Critério de sucesso 8.1.1: A pronúncia do código-fonte é realizada em língua inglesa.

Critério de sucesso 8.1.2: A pronúncia dos rótulos e demais componentes da interface é realizada na língua da interface em uso.

Critério de sucesso 8.1.3: As mensagens de erro podem ser traduzidas para a língua da interface em uso.

Por que fazer?
A pronúncia de palavras-reservadas das linguagens de programação e das mensagens de erro emitidas (me geral, em língua inglesa), realizada pelos leitores de tela, nem sempre é correta, o que pode atrapalhar o entendimento do conteúdo transmitido. Isso ocorre porque, normalmente, o leitor de telas está configurado para leitura na língua da interface em uso (no caso de estudantes brasileiros, a língua da interface em uso geralmente é a língua portuguesa) e não consegue realizar uma leitura do código-fonte que faça sentido para o ouvinte.

Como fazer?
1. Identificar, automaticamente, a posição de foco do cursor do teclado (código-fonte, mensagens de erro ou componentes da interface) e alternar entre as línguas pronunciadas pelo leitor de telas. Palavras reservadas da linguagem de programação e mensagens de erro devem ser lidas em língua inglesa. Componentes da IDE, nomes de variáveis e métodos devem ser lidos na língua da interface em uso.
2. Deve-se fornecer atalhos que permitam traduzir as mensagens de erro para a língua da interface em uso.

Fonte
(Sampath, Merrick & Macvean, 2021)
(Zen, da Costa & Tavares, 2023)

Diretriz 8.2 Leitura contextual

Descrição
Ler o código-fonte de maneira que faça sentido para o ouvinte.

Critérios de sucesso
Critério de sucesso 8.2.1: A leitura do código-fonte fornece informações que facilitam a compreensão das instruções.

Critério de sucesso 8.2.2: As mensagens de erro são clara e facilmente compreendidas.

Por que fazer?
Ao realizar a leitura de um código-fonte, o leitor de telas verbaliza um toquem de cada vez, como se o código fosse um texto, e o programa é dividido em nada mais do que uma série de caracteres, palavras e linhas. Como a sintaxe correta é imperativa na ciência da computação, até mesmo erros intermitentes podem confundir os estudantes. Além disso, as mensagens de erro podem conter muitos jargões (por exemplo,

expressões regulares, siglas específicas do domínio etc.) que podem ser difíceis de serem verbalizadas por um leitor de tela, o que pode dificultar a compreensão dessas mensagens.

Como fazer?

1. Ao efetuar a leitura do código-fonte, o leitor de telas pode utilizar alertas sonoros para:
   - sinalizar o início ou o término de um bloco de código, geralmente representado por chaves;
   - notificar a declaração ou invocação de um método, normalmente envolvendo parênteses; ou,
   - indicar a declaração e/ou acesso aos valores armazenados em um vetor, comumente realizado através do uso de colchetes.

   Esses caracteres não alfanuméricos podem ser omitidos quando o usuário optar por ler o código-fonte e serão pronunciados ou destacados quando ocorrer um erro de sintaxe em tempo real ou quando o usuário solicitar uma lista dos erros de sintaxe identificados no código-fonte.
2. Fornecer mensagens de erro mais detalhadas, no intuito de facilitar a compreensão das informações que estão sendo transmitidas.

Fonte

(Armaly, Rodeghero & McMillan, 2018)
(Baker, Bennett & Ladner, 2019)
(Sampath, Merrick & Macvean, 2021)
(Schanzer, Bahram & Krishnamurthi, 2019)
(Zen, da Costa & Tavares, 2023)

4. REFERÊNCIAS BIBLIOGRÁFICAS

ALBUSAYS, Khaled; LUDI, Stephanie. *Eliciting programming challenges faced by developers with visual impairments: exploratory study. In: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering*. 2016. p. 82-85.

ALBUSAYS, Khaled; LUDI, Stephanie; HUENERFAUTH, Matt. *Interviews and observation of blind software developers at work to understand code navigation challenges. In: Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. 2017. p. 91-100.

ALOTAIBI, Hind; S. AL-KHALIFA, Hend; ALSAEED, Duaa. *Teaching programming to students with vision impairment: Impact of tactile teaching strategies on student's achievements and perceptions. Sustainability*, v. 12, n. 13, p. 5320, 2020.

ARMALY, Ameer; RODEGHERO, Paige; MCMILLAN, Collin. *Audiohighlight: Code skimming for blind programmers. In: 2018 IEEE International Conference on Software Maintenance and Evolution* (ICSME). IEEE, 2018. p. 206-216.

BAKER, Catherine M.; BENNETT, Cynthia L.; LADNER, Richard E. *Educational experiences of blind programmers. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* 2019. p. 759-765.

DA SILVA, Cláudia Ferreira; FERREIRA, Simone Bacellar Leal; RAMOS, João Felipe Moreira. *WhatsApp accessibility from the perspective of visually impaired people. In: Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems*. 2016. p. 1-10.

GERALDO, Rafael José. Um auxílio à navegação acessível na web para usuários cegos. 2016. Tese de Doutorado. Universidade de São Paulo.

HUFF, Earl W. et al. *Examining the work experience of programmers with visual impairments. In: 2020 IEEE international conference on software maintenance and evolution* (ICSME). IEEE, 2020. p. 707-711.

HUTCHINSON, Joe; METATLA, Oussama. *An initial investigation into non-visual code structure overview through speech, non-speech and spearcons. In: Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018. p. 1-6.

ISO/IEC (2018). ISO/IEC 9241. Ergonomia da interação humano-sistema. Parte 171: Orientaçõees sobre acessibilidade de software.

KEARNEY-VOLPE, Claire; HURST, Amy. *Accessible web development: Opportunities to improve the education and practice of web development with a screen reader. ACM Transactions on Accessible Computing* (TACCESS), v. 14, n. 2, p. 1-32, 2021.

MEALIN, Sean; MURPHY-HILL, Emerson. *An exploratory study of blind software developers. In: 2012 ieee symposium on visual languages and human-centric computing* (VL/HCC). IEEE, 2012. p. 71-74.

MOUNTAPMBEME, Aboubakar; OKAFOR, Obianuju; LUDI, Stephanie. *Addressing Accessibility Barriers in Programming for People with Visual Impairments: A Literature Review. ACM Transactions on Accessible Computing* (TACCESS), v. 15, n. 1, p. 1-26, 2022.

PANDEY, Maulishree et al. *Accessibility of UI Frameworks and Libraries for Programmers with Visual Impairments. In: 2022 IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC). IEEE, 2022. p. 1-10.

PETRAUSCH, Vanessa; LOITSCH, Claudia. *Accessibility analysis of the eclipse ide for users with visual impairment. In: Harnessing the Power of Technology to Improve Lives*. IOS Press, 2017. p. 922-929.

POTLURI, Venkatesh et al. *A multi-modal approach for blind and visually impaired developers to edit webpage designs. In: Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility.* 2019. p. 612-614.

POTLURI, Venkatesh et al. *Codetalk: Improving programming environment accessibility for visually impaired developers. In: Proceedings of the 2018 chi conference on human factors in computing systems*. 2018. p. 1-11.

SAMPATH, Harini; MERRICK, Alice; MACVEAN, Andrew. *Accessibility of command line interfaces. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 2021. p. 1-10.

SCHANZER, Emmanuel; BAHRAM, Sina; KRISHNAMURTHI, *Shriram. Accessible AST-based programming for visually-impaired programmers. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019. p. 773-779.

SIEGFRIED, Robert M. *Visual programming and the blind: the challenge and the opportunity*. ACM SIGCSE Bulletin, v. 38, n. 1, p. 275-278, 2006.

SMITH, Ann C. et al. *Nonvisual tool for navigating hierarchical structures. ACM SIGACCESS Accessibility and Computing*, n. 77-78, p. 133-139, 2003.

STEFIK, Andreas et al. Sodbeans. In: 2009 IEEE 17th *International Conference on Program Comprehension*. IEEE, 2009. p. 293-294.

W3C, W. W. W. C. (2018). Diretrizes de acessibilidade para conteúdo web (WCAG) 2.1. Disponível em: https://www.w3c.br/traducoes/wcag/wcag21-pt-BR/.

W3C, World Wide Web Consortium (2015). *Authoring Tool Accessibility Guidelines* (ATAG) 2.0. Disponível em: https://www.w3.org/TR/ATAG20/.

Zen, Eliana, da Costa, Viníciius K., and Tavares, Tatiana A. *Understanding the Accessibility Barriers Faced by Learners with Visual Impairments in Computer Programming*. In: XXII Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais, 2023.