

Websites Atendendo a Requisitos de Acessibilidade e Usabilidade



Vagner Figueredo de Santana

Leonelo Dell Anhol Almeida

Maria Cecília Calani Baranauskas

Prefácio por
Maria Teresa Eglér Mantoan



<http://leanpub.com/warau>

ISBN: 978-85-907039-1-4

Prefácio	8
Introdução	10
Sobre os Autores	12
O que é Acessibilidade?	14
Design Universal e Design Acessível	16
Tecnologias Assistivas	17
Princípios para o desenvolvimento Web acessível	18
Padronização de códigos HTML	20
Exemplo de atributo HTML com aspas simples:	20
Exemplo de comentário em HTML:	20
Estrutura de documentos HTML	22
Exemplo de <i>tags</i> básicas do HTML:	22
Exemplo de uso das <i>tags</i> <head> e <body>:	22
Contraexemplo de título de página:	22
Exemplo de <i>tags</i> <meta>:	23
Contraexemplo sobre atualização automática de páginas:	23
Exemplo de relação entre documentos na <i>tag</i> <link>:	23
Exemplo da <i>tag</i> <body>:	24
Âncoras e hiperlinks - A tag HTML <a>	25
Exemplo de <i>skip link</i> :	25
Mapeamentos - A tag HTML <map>	27
Listas - As tags HTML , e <dl>	28
Exemplo das <i>tags</i> <dt> e <dd>:	28
Subtópicos - As tags HTML <h1> a <h6>	30
Exemplo de aninhamento de <i>tags</i> para subtópicos:	30
Tabulação - O atributo HTML tabindex	31
Frames - A tag HTML <frame>	32
Elementos de bloco e elementos de linha	33
Agrupamento de elementos de bloco - A tag HTML <div>	34
Exemplo da <i>tag</i> <div>:	34
Agrupamento de texto - A tag HTML 	35
Exemplo da <i>tag</i> :	35
Quebras de linha - A tag HTML 	36
Exemplo da <i>tag</i> :	36
Parágrafos - A tag HTML <p>	37
Exemplo da <i>tag</i> <p>:	37
Idiomas - O atributo HTML lang	38
Exemplo do atributo <i>lang</i> :	38
Textos sobrescritos - A tag HTML <sup>	39
Exemplo da <i>tag</i> <sup>:	39
Textos subscritos - A tag HTML <sub>	40
Exemplo da <i>tag</i> <sub>:	40
Fontes - A tag HTML 	41
Exemplo da <i>tag</i> :	41
Ênfase no texto - As tags HTML e 	42
Exemplo das <i>tags</i> e :	42
Abreviações e siglas - As tags HTML <abbr> e <acronym>	43

Exemplo das <i>tags</i> <code><abbr></code> e <code><acronym></code> :	43
Exemplo:	43
Cores - Os atributos HTML <code>color</code> e <code>bgcolor</code>	44
Exemplo dos atributos <code>color</code> e <code>bgcolor</code> :	44
Réguas horizontais - A tag HTML <code><hr></code>	45
Exemplo da <i>tag</i> <code><hr></code> :	45
Textos alternativos	46
Imagens - A tag HTML <code></code>	47
Exemplo da <i>tag</i> <code></code> :	47
Elementos multimídia - A tag HTML <code><object></code>	48
Exemplo da <i>tag</i> <code><object></code> :	48
Formulários - A tag HTML <code><form></code>	49
Exemplo de URL destino de um formulário via <i>get</i> :	49
Campos de entrada de dados - A tag HTML <code><input></code>	50
Exemplo da <i>tag</i> <code><input></code> :	50
Rótulos - A tag HTML <code><label></code>	51
Exemplo da <i>tag</i> <code><label></code> :	51
Caixas de seleção - A tag HTML <code><select></code>	52
Exemplo da <i>tag</i> <code><select></code> :	52
Exemplo da <i>tag</i> <code><optgroup></code> :	52
Campos de texto - A tag HTML <code><textarea></code>	53
Exemplo da <i>tag</i> <code><textarea></code> :	53
Agrupamento de elementos de formulário - A tag HTML <code><fieldset></code>	54
Exemplo da <i>tag</i> <code><fieldset></code> :	54
Rótulos para fieldsets - A tag HTML <code><legend></code>	55
Exemplo da <i>tag</i> <code><legend></code> :	55
Tabelas - As tags HTML <code><table></code> , <code><td></code> , <code><tr></code> , <code><th></code> e <code><caption></code>	56
Exemplo da <i>tag</i> <code><table></code> :	56
Exemplo da <i>tag</i> <code><th></code> :	56
Exemplo do atributo <i>abbr</i> na <i>tag</i> <code><th></code> :	57
Exemplo da <i>tag</i> <code><caption></code> :	57
Validação de código HTML	59
Sintaxe CSS	60
Exemplo de declaração atributo-valor CSS:	60
Exemplo de agrupamento de seletores CSS:	60
Exemplo de seletor <i>class</i> em CSS:	60
Exemplo de uso do seletor <i>class</i> CSS no HTML:	60
Exemplo de uso combinado de classes CSS:	60
Exemplo de omissão do elemento HTML na classe CSS:	61
Exemplo de uso de classe CSS com omissão do elemento HTML:	61
Inclusão de código CSS	62
Exemplo de declaração interna do CSS:	62
Exemplo de declaração externa do CSS:	62
Exemplo de declaração <i>inline</i> do CSS:	62
Estrutura de documentos CSS	64
Exemplo de cálculo de pesos de seletores CSS:	64
Padronização de código CSS	65

Exemplo de declaração padronizada em CSS:	65
Contraexemplo de declaração <i>inline</i> em CSS:	65
Exemplo de declaração <i>inline</i> em CSS:	65
Contraexemplo de declaração CSS baseada na aparência:	65
Contraexemplo de declaração CSS baseada no conteúdo:	65
Exemplo de declaração CSS baseada na função do dado:.....	65
Prioridade para estilos do usuário	67
Unidades de medida relativas - Unidades em CSS: Porcentagem e em.....	68
Conteúdos gerados pelo CSS: os pseudo-elementos before e after	69
Listas ordenadas utilizando CSS	70
Exemplo de CSS para listas ordenadas com subníveis:	70
Exemplo de marcações de fim de listas ordenadas usando CSS:	70
Exemplo de marcações de fim de listas ordenadas usando CSS no HTML:.....	70
Réguas - Alterando a aparência com CSS	72
Exemplo de alteração da aparência de réguas usando CSS:	72
Exemplo de alteração da aparência de réguas usando CSS no HTML:.....	72
Bordas utilizando CSS.....	73
Exemplo de definição de bordas usando CSS:	73
Posicionamento utilizando CSS	74
Exemplo de posicionamento usando CSS:	74
Fontes utilizando CSS	75
Exemplo de definição de fontes usando CSS:	75
Efeitos em texto utilizando CSS	76
Contraexemplo de uso de letras maiúsculas no HTML:	76
Exemplo de transformação de texto para letras maiúsculas usando CSS:.....	76
Exemplo de transformação de texto para letras maiúsculas usando CSS no HTML:	76
Formatação e posicionamento de texto utilizando CSS	77
Contraexemplo de ênfase em texto usando espaços e letras maiúsculas:	77
Exemplo de ênfase em texto usando CSS:	77
Exemplo de ênfase em texto usando classe CSS no HTML:	77
Utilizando texto em vez de imagem	78
Contraexemplo de uso de imagens para representar texto:.....	78
Exemplo de texto com aparência definida no CSS:	78
Exemplo de uso de classe CSS para alterar a aparência do texto no HTML:	78
Atributos aurais com CSS	79
Exemplo de atributos aurais no CSS:	79
Cores com CSS.....	80
Exemplo de definição de cor de fundo e de texto usando CSS:	80
Validação de folhas de estilo CSS.....	81
Recomendações para uso do JavaScript	82
Inclusão de código JavaScript	83
Exemplo de inclusão de JavaScript internamente no HTML:	83
Exemplo de inclusão de documento Javascript no HTML:	83
Declaração de variáveis em JavaScript	84
Exemplo de declaração de variáveis em Javascript:	84
Contraexemplo de uso de variáveis fora de seu escopo:	84
Exemplo de uso de variáveis com o escopo correto:	84

Ordem da declaração de variáveis e métodos em JavaScript	85
Quebras de linha no código JavaScript	86
Contraexemplo de declaração em Javascript sem ponto-e-vírgula:	86
Exemplo de declaração em Javascript usando ponto-e-vírgula:	86
Cuidados com versões da linguagem JavaScript.....	87
Deteção de navegador, objetos e métodos utilizando JavaScript	88
Contraexemplo de código Javascript dependente de navegador:	88
Exemplo de teste de compatibilidade de objeto Javascript sem dependência de navegador:	88
Contraexemplo de detecção de método Javascript com uso de parênteses:	88
Exemplo de detecção de método Javascript:	88
Conversão de tipo de dados em JavaScript	90
Contraexemplo de comparação de variáveis Javascript de tipos diferentes:	90
Exemplo de conversão de tipo de variável Javascript antes de comparação:	90
Exemplo de retorno de função com tipo de dados compatível com a lógica da função:	90
Manipuladores de evento em HTML	92
Exemplo de manipuladores Javascript de eventos para diferentes dispositivos:	92
Desvios e laços condicionais em JavaScript	93
Contraexemplo de não uso de chaves para blocos condicionais e laços em Javascript:	93
Exemplo de uso de chaves para blocos condicionais e laços em Javascript:	93
Escrita de HTML via JavaScript.....	94
Contraexemplo de impressão em Javascript:	94
Exemplo de impressão em Javascript usando concatenação:	94
Aspas simples e duplas em JavaScript	95
Contraexemplo de uso de aspas em Javascript:	95
Exemplo de uso de aspas em Javascript:	95
Exemplo de uso de aspas simples em Javascript:	95
ETAGO em JavaScript	96
Contraexemplo de impressão de ETAGO em Javascript:	96
Exemplo de impressão de ETAGO em Javascript:	96
Escrevendo a palavra reservada script	97
Contraexemplo de impressão da palavra script:	97
Exemplo de impressão da palavra script:	97
Manipulação de CSS utilizando JavaScript	98
Exemplo de manipulação de CSS utilizando Javascript:	98
Navegação utilizando JavaScript.....	99
Exemplo de melhoria de navegação usando Javascript:	99
Validação de formulários utilizando JavaScript	101
Exemplo de validação de formulários usando Javascript:	101
Acesso a conteúdo e redundância utilizando JavaScript	102
Exemplo de uso da <i>tag</i> <code><noscript></code> :	102
Gráficos e animações utilizando JavaScript	103
Exemplo de animação utilizando Javascript:	103
Manipulação de HTML utilizando JavaScript	105
Exemplo de alteração de código HTML utilizando Javascript:	105
AJAX.....	106

Exemplo de busca dinâmica utilizando AJAX:	106
Manipulação de HTML utilizando JavaScript	108
Exemplo:	108
Verificação de código JavaScript	109
Reaproveitamento de Código	110
Contraexemplo de documento HTML que não propicia reaproveitamento de código:	110
Exemplo de declaração de HTML reaproveitando código para cabeçalho:	110
Contraexemplo de seletores CSS que não propiciam reaproveitamento:	111
Exemplo de declaração de seletores CSS utilizando a estrutura em cascata:	111
Contraexemplo de Javascript para reaproveitamento de código:	112
Exemplo de Javascript para reaproveitamento de código:	112
O que é Usabilidade?	113
Integração Usabilidade - Acessibilidade	115
Os 10 maiores erros em webdesign	116
Inspeção Heurística de Usabilidade	119
Heurística genéricas	120
Usabilidade na Web	122
Formulário de Inspeção Heurística na Web	126
Acessibilidade no W3C	128
Modelo de Acessibilidade Brasileiro - e-MAG	130
Recomendações de Acessibilidade para: Utilização de novas tecnologias	131
Recomendações de Acessibilidade para: Portabilidade	132
Bloqueio da navegação por teclado	133
Recomendações de Acessibilidade para: Skip Links	134
Exemplo de <i>skip link</i> :	134
Recomendações de Acessibilidade para: Breadcrumbs	135
Exemplo de <i>breadcrumb</i> :	135
Recomendações de Acessibilidade para: Teclas de Atalho	137
Exemplo de declaração de teclas de atalho utilizando o atributo <i>accesskey</i> :	137
Exemplo de CSS para indicação visual de teclas de atalho:	138
Exemplo de uso de classe CSS para indiciação visual de teclas de atalho no HTML:	138
Recomendações de Acessibilidade para: Legibilidade	139
Escrita de textos para a Web	140
Recomendações de Acessibilidade para: Contexto	142
Recomendações de Acessibilidade para: Agrupamento Espacial	143
Recomendações de Acessibilidade para: Ícones	144
Recomendações de Acessibilidade para: Redundância	145
Contraexemplo de informação fornecida somente por cores:	145
Exemplo de informação fornecida por cores e texto:	145
Recomendações de Acessibilidade para: Recursos de Áudio	147
Recomendações de Acessibilidade para: Controle do Usuário	148
Técnicas para encontrar conteúdo em um website	149
Avaliação Simplificada de Acessibilidade	150
Formulário para Avaliação Simplificada de Acessibilidade	152
Avaliação de acessibilidade utilizando ferramentas semiautomáticas	156
Referências	157

Glossário.....	160
Referências do Glossário.....	167

Prefácio

O projeto “Acesso, Permanência e Prosseguimento da Escolaridade de Nível Superior de Alunos com Deficiência: Ambientes Inclusivos”, financiado pela CAPES e desenvolvido nos anos de 2004 a 2009, na UNICAMP, adiantou-se à apresentação e implementação da educação inclusiva pela Política Nacional de Educação Especial, na Perspectiva da Educação Inclusiva (PNEEPI), apresentada pelo Ministério da Educação, em 2008. Na época, já vislumbrávamos o caminho ascendente dos estudos de alunos da educação especial, que, certamente, culminariam nos cursos universitários.

O desenvolvimento desse estudo, que reuniu as expertises de professores e pesquisadores da Faculdade de Educação (FE) e do Instituto de Computação (IC) vinculou-se inicialmente à nossa comunidade universitária, na intenção de dar suporte especializado não apenas aos estudantes público alvo da educação especial, mas aos professores, servidores, pessoal interno e externo a seus campi. Por tudo isso, o projeto foi, de fato, uma iniciativa pioneira em favor da inclusão escolar e social no Brasil.

Especulações em torno de temas relacionados ao direito de todos à educação, à singularidade humana, às identidades, acessibilidade e participação de todos à escola, à sociedade, sem discriminação e segregação estavam subjacentes à iniciativa de se propor esse projeto e de executá-lo naquele momento. Temas relacionados ao acesso à comunicação, ao ambiente físico e às bibliotecas da UNICAMP nos conduziram por caminhos inexplorados e pouco conhecidos da inclusão. Éramos muitos, assim como os produtos que conseguimos entregar após quatro anos de muito trabalho e dedicação. Um grupo de nossos pesquisadores de nossa equipe, entre os quais o autor deste livro, dedicou-se exclusivamente à acessibilidade na Web. Esse braço da pesquisa tinha em vista o direito constitucional que assegura a todos os brasileiros o acesso à informação e estava alinhado à ideia de design de produtos, equipamentos, recursos abertos a todos os usuários, com e sem deficiências. A questão era não perder de vista essa abrangência e o resultado desse trabalho está exposto neste livro.

À época, o propósito inicial desse grupo foi produzir uma ferramenta para apoiar mantenedores e desenvolvedores da UNICAMP na produção e adaptação de websites a requisitos de usabilidade e de acessibilidade da codificação à validação e avaliação. Criou-se, então, o WARAU (Websites Atendendo a Requisitos de Acessibilidade e Usabilidade). Concebido para atender a uma necessidade interna da UNICAMP, revelada pelo projeto, a ferramenta chamou atenção dos interessados na área. Tudo era muito novo. A promoção do acesso de todos, indistintamente, aos meios de comunicação, seguindo o que preconiza um design para todos foi penetrando novos espaços acadêmicos e fazendo parte de seus diferentes serviços.

Aos poucos, a ferramenta extrapolou o seu espaço de criação e é essa expansão que celebramos aqui, recomendando-a a quem se interessa e se dedica a promover a inclusão por meio de websites acessíveis a todos.

Pensar em cada um, na sua diferença, como um ser singular, alguém que não se repete e que se constitui no devir é o que se impõe a nós, que compreendemos o valor social e individual de nossas ações, intentos e produções.

Esse foi e continua sendo o propósito e o caminho dos criadores do WARAU e pelos quais merecem parabéns. Fazer uso dessa ferramenta é mais um movimento em favor de um mundo que cuida para que todos possamos ter melhores condições de vivê-lo e de usufruí-lo sem limites.

Maria Teresa Eglér Mantoan, em abril de 2020.

Introdução

O projeto Websites Atendendo a Requisitos de Acessibilidade e Usabilidade, ou WARAU, nasceu como um *website* que tinha o objetivo de ser um espaço de discussão sobre normas, diretrizes, técnicas e boas práticas para a criação de código Web acessível e usável. O nascimento do projeto foi marcado pela publicação do relatório técnico intitulado *Um processo para adequação de websites a requisitos de acessibilidade e usabilidade*, 2008 [1]. Este relatório representa uma contribuição ao Projeto "Acesso, Permanência e Prosseguimento da Escolaridade em Nível Superior de Pessoas com Deficiência: Ambientes Inclusivos" (PROESP/CAPEs), realizado na Universidade Estadual de Campinas (UNICAMP). Além disso, o WARAU também envolveu um estudo realizado com uma equipe de desenvolvimento de sistemas Web da UNICAMP em que foram identificadas dificuldades e desafios relacionados ao desenvolvimento web acessível combinando HTML, CSS, JavaScript, Acessibilidade e Usabilidade. Como forma de lidar com esses desafios, o WARAU foi criado e disponibilizado para a comunidade de mantenedores de websites. Para detalhes sobre a criação do conteúdo do WARAU e sobre o estudo realizado mencionado, veja [2].

O conteúdo do WARAU é voltado para mantenedores de *websites* que já tenham algum conhecimento de código Web (e.g., HTML, CSS, JavaScript) e que queiram aprender como construir *websites* válidos, acessíveis e usáveis, a partir de exemplos. O objetivo do website – hoje não mais mantido – era possibilitar diferentes formas de filtrar os tópicos por perfil (desenvolvedor, designer e redator) e/ou por tema (HTML, CSS, JavaScript, Acessibilidade e Usabilidade). Entretanto, mantemos neste material as referências de relevância para os temas de forma que o leitor saiba qual é a relevância de cada tópico para seu perfil. Para saber qual é o seu perfil, considere o seguinte:

- Se você é responsável pelo conteúdo do *website*, considere o perfil redator;
- Se você é responsável pela codificação da dinâmica do *website*, considere o perfil de desenvolvedor.
- Se você é responsável por garantir a qualidade da interação do usuário, assim como a acessibilidade e usabilidade do *website*, o seu perfil é *designer*.
- Por fim, se você se enquadra em mais de um dos perfis, fato muito comum em equipes pequenas, considere todos os que você se identificar.

Cada tópico do WARAU aborda três perguntas básicas:

- 1) Como fazer?
- 2) Quais ferramentas posso utilizar?
- 3) Qual resultado/benefício esperar?

Para responder a primeira questão, disponibilizamos definições, textos explicativos e exemplos. Para a segunda, nós indicamos algumas ferramentas e *websites* (geralmente *software* livre) que podem ser utilizados para aplicar o que foi discutido na primeira questão. Já para a terceira questão indicamos os principais resultados esperados com a aplicação do tópico.

Neste livro apresentamos uma maneira alternativa de navegar pelo conteúdo do WARAU, um roteiro de leitura sequencial que aborda a tríade básica de tecnologias Web, mas sempre com foco prático e sempre considerando Acessibilidade e Usabilidade.

1. Almeida, L. D. A., Santana, V. F., Baranauskas, M. C. C. (2008). [Um processo para adequação de websites a requisitos de acessibilidade e usabilidade](#). Relatório Técnico do Instituto de Computação, UNICAMP. Janeiro de 2008. Campinas-SP, Brasil.
2. Santana, V. F., Almeida, L. D. A., Baranauskas, M. C. C. (2008). [Aprendendo sobre Acessibilidade e Construção de Websites para Todos](#). Revista Brasileira de Informática na Educação, v. 16, p. 71-83, 2008.

Sobre os Autores

Vagner Figueredo de Santana¹

Pesquisador e Master Inventor na IBM Research. Docente colaborador na Universidade Federal do ABC. Doutor em Ciência da Computação pela Universidade Estadual de Campinas (UNICAMP) (2012), mestre em Ciência da Computação pela UNICAMP (2009) e bacharel em Ciência da Computação pela Universidade Presbiteriana Mackenzie (2006). Durante o doutorado estagiou no laboratório Human Interface in Information Systems, do Consiglio Nazionale delle Ricerche, em Pisa, Itália, sob orientação do prof. Fabio Paternò. Lecionou disciplinas relacionadas à Engenharia de Software como professor visitante na pós-graduação da Universidade Presbiteriana Mackenzie de 2009 a 2015. Foi webmaster da Folha Online (www.folha.com.br) de 2002 a 2007. Tem experiência na área de Ciência da Computação, com ênfase na Web e em Interação Humano-Computador, mais especificamente em avaliação de Interfaces de Usuário e análise de logs de interação com foco em acessibilidade e usabilidade.

Leonelo Dell Anhol Almeida²

Possui graduação em Bacharelado em Informática pela Universidade Estadual de Ponta Grossa (2003), mestrado em Informática pela Universidade Federal do Paraná (2006) e doutorado em Ciência da Computação pela Universidade Estadual de Campinas (2011). Atualmente é professor adjunto do magistério superior da Universidade Tecnológica Federal do Paraná. Atua no Programa de Pós-Graduação em Computação Aplicada (PPGCA, UTFPR), no nível de mestrado, e no Programa de Pós-Graduação em Tecnologia (PPGTE, UTFPR), nos níveis de mestrado e de doutorado. Tem experiência na área de Ciência da Computação, com ênfase em Interação Humano-Computador, atuando principalmente nos seguintes temas: acessibilidade, Design de Interação, estudos em Ciência, Tecnologia e Sociedade e sistemas colaborativos.

Maria Cecília Calani Baranauskas

Possui graduação em Ciência da Computação - Bacharelado pela Universidade Estadual de Campinas (1976) e em Matemática - Licenciatura pela Universidade Estadual de Campinas (1976), mestrado em Ciência da Computação pela Universidade Estadual de Campinas (1981) e doutorado em Engenharia Elétrica pela Universidade Estadual de Campinas (1993). Realizou pós-doutoramento no Semiotics Special Interest Group (SSIG) na Staffordshire University (School of Computing) em 2001 e no Applied Informatics with Semiotics (AIS) lab da University of Reading (Dept. of Computer Science) em 2002, UK. Recebeu a Cátedra Ibero-Americana UNICAMP-Santander Banespa para estudar problemas de acessibilidade em engenharia de software na Universidad Politécnica de Madrid, Espanha (2006-2007). Foi agraciada com o Diploma do Mérito Educacional "Prof. Darcy Ribeiro" em 2006 e com o ACM SIGDOC Rigo Award em 2010. É Professora Titular da Universidade Estadual de

¹ Pesquisa apoiada até março de 2009 pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) através de bolsa de mestrado. De abril de 2009 a setembro de 2012 a pesquisa foi apoiada pela FAPESP através de bolsa de doutorado (processo #2009/10186-9).

² Pesquisa apoiada pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) através de bolsa de doutorado (processo #2007/02161-0).

Campinas. Tem experiência na área de Ciência da Computação, com ênfase em Metodologia e Técnicas da Computação, atuando principalmente nos seguintes temas: interação humano-computador, semiótica organizacional, interface de usuário e design de sistemas computacionais interativos em diversos domínios (social, educacional, de trabalho).

Designer

Desenvolvedor

Redator

O que é Acessibilidade?

Segundo o Decreto nº 5.296 (DECRETO nº 5.296 DE 2 DE DEZEMBRO DE 2004, 2004), acessibilidade está relacionada a fornecer condição para utilização, com segurança e autonomia, total ou assistida, dos espaços, mobiliários e equipamentos urbanos, das edificações, dos serviços de transporte e dos dispositivos, sistemas e meios de comunicação e informação, por pessoa com deficiência ou com mobilidade reduzida. No mesmo documento, barreiras são definidas como qualquer entrave ou obstáculo que limite ou impeça o acesso, a liberdade de movimento, a circulação com segurança e a possibilidade de as pessoas se comunicarem ou terem acesso à informação (DECRETO nº 5.296 DE 2 DE DEZEMBRO DE 2004, 2004).

O programa Custe o que Custar (CQC) da rede Bandeirantes de televisão fez uma matéria em 02/06/2008 que ilustra bem como ainda há muito para ser feito sobre acessibilidade no Brasil.

Vídeo: CQC - Proteste Já! Vagas para Deficientes (parte 1 de 2). Fonte: Programa CQC da Bandeirantes. <https://www.youtube.com/watch?v=vSfo3-9-1cY>

Vídeo: CQC - Proteste Já! Vagas para Deficientes (parte 2 de 2). Fonte: Programa CQC da Bandeirantes. <https://www.youtube.com/watch?v=NqEfBYnvNBI>

Acessibilidade na *Web* significa que pessoas com diferentes tipos de deficiência também conseguem entender, navegar, interagir e contribuir com *websites*, inclusive idosos, que sofrem alterações em algumas de suas habilidades devido à idade (WAI - Web Accessibility Initiative).

Assim, podemos pensar na acessibilidade como sendo a qualidade de locais, dispositivos, meios de comunicação, sistemas e serviços, que podem ser utilizados por pessoas com diferentes necessidades e capacidades. Note que por "pessoas com diferentes necessidades" nos referimos a qualquer indivíduo, uma vez que existem inúmeras situações em que um ou mais sentidos de uma pessoa não estão voltados para a mesma tarefa. A seguir são apresentados alguns cenários em que estão envolvidas barreiras e problemas de acessibilidade que atingem pessoas com diferentes tipos de limitações:

Em torneiras que utilizam temporizador para controlar o fluxo da água, se uma torneira está mal regulada de forma que a água pare assim que o botão é liberado, um usuário que está com uma das mãos ocupadas não conseguirá utilizar a torneira. Basta que o usuário esteja com um dos braços engessado ou segurando um bebê para não conseguir utilizar a torneira em questão.

Em escritórios em que é permitido que se escute música com a utilização de fone de ouvido, um telefone sem indicativo visual de que está tocando não chamará a atenção do seu usuário e consequentemente não atingirá seu objetivo. Note que o usuário não precisa ter nenhum problema no aparelho auditivo para fazer uso da redundância obtida com a utilização do recurso visual.

Na computação a redundância também é um recurso bastante útil para a garantia da acessibilidade. Um exemplo da relevância da redundância é a ferramenta de comunicação instantânea do webmail Gmail (Google Mail). Nele, notificações são informadas através de efeitos visuais e auditivos.

Vídeo: The Fountain. Fonte: aniBoom. <https://www.youtube.com/watch?v=FX9g5a-KwHQ>

Por fim, o apoio à busca de acessibilidade não exclui nenhum dos usuários e estende o conceito de usabilidade como um todo. E, seguindo a definição de barreiras do Decreto nº 5.296 (DECRETO nº 5.296 DE 2 DE DEZEMBRO DE 2004, 2004), uma forma de tornar algo mais acessível é identificar e eliminar quaisquer barreiras existentes entre os usuários e o que eles querem utilizar.

Vídeo: Acessibilidade - Siga esta ideia. Fonte: CONADE - Conselho Nacional dos Direitos das Pessoas Portadoras de Deficiência. https://www.youtube.com/watch?v=IXPg04_Evw

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•WAI
Resultados esperados	•Definição de conceitos

Design Universal e Design Acessível

O *Design Universal* é um conceito bastante polêmico, dada a sua definição inicial, que é "o *design* de produtos e ambientes para serem usáveis por todas as pessoas, na maior extensão possível, sem a necessidade de adaptação ou *design* especializado." (The Center for Universal Design: The Principles of Universal Design, Version 2.0, 1997), tradução de (Baranauskas, 2007). No entanto, é necessário salientar que criar produtos segundo os princípios do *Design Universal* não implica em, necessariamente, um produto único para todos. Na maioria das vezes, o objetivo é o de respeitar, o máximo possível, os princípios do *Design Universal*, que são 7 no total (a descrição completa dos princípios pode ser encontrada em Princípios do Design Universal):

- Uso equitativo;
- Flexibilidade de uso;
- Simples e intuitivo;
- Informação perceptível;
- Tolerância ao erro;
- Baixo esforço físico;
- Tamanho e espaço para aproximação e uso.

Um exemplo de um produto que atende princípios do *Design Universal* é o de portas automáticas que se abrem quando alguém se aproxima. Este é um produto que funciona independentemente se a pessoa está caminhando, em uma cadeira de rodas, se é deficiente visual, etc. Um contraexemplo para isso são as entradas de estabelecimentos que possuem uma escada e uma passarela ao lado. Essa forma de *design* faz uso de produtos diferentes para usuários com habilidades distintas.

Já o *Design Acessível* é um subconjunto do *Design Universal* e, conseqüentemente, possui um escopo mais restrito. Assim, "... o *Design Acessível* foca nos princípios que estendem o processo de *design* padrão para aquelas pessoas com algum tipo de limitação". (Vanderheinden, G. C., 1992, tradução de (Baranauskas, 2007))

Tema	• Acessibilidade
Relevância para designers	• Alta
Relevância para desenvolvedores	• Alta
Relevância para redatores	• Alta
Ferramental utilizado	• Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	• Avaliação e definição de conceitos

Tecnologias Assistivas

Tecnologias assistivas, sob o escopo da informática, são todos os artefatos que auxiliam de alguma forma as pessoas com algum tipo de necessidade, seja ela física, ambiental, entre outros. Os artefatos voltados às tecnologias assistivas podem ser tanto *hardware* (e.g., impressora Braille, linhas Braille, apontadores) como *software* (e.g., leitores de telas, ampliadores de telas, navegadores textuais, barras de acessibilidade com ajustes de tamanho de texto e contraste).

No entanto, ao contrário do que é comumente praticado, o uso de tecnologias assistivas nem sempre é suficiente para promover a acessibilidade. Um exemplo disso ocorre quando uma pessoa utilizando um leitor de telas carrega uma página *Web* que não está de acordo com as recomendações de acessibilidade para *websites*. Dessa forma ela possivelmente não conseguirá acessar os elementos da página de forma eficiente.

Exemplos de tecnologias assistivas de hardware são:

Impressora Braille, Linha Braille e Teclado para ser usado apenas com uma mão.

Exemplos de tecnologias assistivas de software são:

Leitores de telas: *NVDA* (NVDA), *JAWS* (JAWS), *WebAnywhere* (WebAnywhere)

Ampliador de telas.

Barra de acessibilidade do *website*.

Headmouse. *Software* para mexer o ponteiro do *mouse* a partir de movimentos da cabeça.

Versão em alto contraste.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•NVDA, Jaws, WebAnywhere, DOSVOX e Lynx
Resultados esperados	•Definição de conceitos

Princípios para o desenvolvimento Web acessível

Um bom processo de desenvolvimento *Web* não deve se restringir somente a validações de código. O conteúdo do WARAU é norteado por princípios oriundos de experiências (profissionais e acadêmicas) e que acreditamos serem necessários para a criação e manutenção de *websites* acessíveis e usáveis. Estes princípios envolvem desde a estruturação do *website* e padronização de código até o uso de ferramentas de validação e integração de tecnologias. São eles:

1. **Definir padrão de codificação** - A seleção criteriosa de padrão de nomenclatura de elementos, variáveis e versões de linguagem contribui para a legibilidade do código. Para a equipe de desenvolvimento facilita manutenção e inclusão de novos recursos.
 - Padronização de códigos HTML;
 - Padronização de código CSS;
 - Inclusão de código JavaScript;
2. **Estruturar páginas e *websites* prezando o reaproveitamento de código** - Todo o código *Web*, seja ele documento HTML, folha de estilo CSS ou programa em JavaScript, deve ser escrito de tal forma que possa ser reaproveitado em diferentes áreas do *website* e, conseqüentemente, contribua para a manutenção e redução de consumo de recursos de tempo e financeiro.
 - Reaproveitamento de Código;
3. **Prezar pela semântica no código** - O conteúdo de *websites* deve ser escrito considerando os elementos semânticos disponíveis nas linguagens de marcação (e.g., títulos, parágrafos, tabelas, abreviações).
 - Subtópicos - As tags HTML <h1> a <h6>;
 - Idiomas - O atributo HTML lang;
 - Ênfase no texto - As tags HTML e ;
4. **Aplicar padrões e diretrizes de tecnologias e conceitos** - Linguagens *Web* contam com padrões e recomendações que, quando conhecidos por mantenedores, solucionam grande parte dos problemas comumente encontrados em *websites*. Com isso, possibilitam sua maior compatibilidade com os diversos dispositivos, navegadores, sistemas operacionais e outras aplicações utilizadas. Além disso, quando consideradas diretrizes de acessibilidade e usabilidade, é esperado um ganho em relação à capacidade de o *website* atender às necessidades específicas de cada usuário (e.g., navegação via teclado, sem recursos sonoros).
 - Os 10 maiores erros em webdesign;
 - Inspeção Heurística de Usabilidade;
5. **Não se restringir a padrões, diretrizes de tecnologias nem conceitos** - Apesar de trazerem melhorias, padrões e diretrizes não são soluções suficientes para a garantia de qualidade de um *website*. Para tanto é necessário considerar as condições e restrições de uso específicas de cada *website* e levar em consideração o referencial teórico da área em questão.
 - Recomendações de Acessibilidade para: Skip Links;
 - Recomendações de Acessibilidade para: Breadcrumbs;
6. **Considerar a diversidade de usuários** - Ao contrário do que geralmente é adotado por mantenedores de *websites*, desenvolver *websites* para o "usuário médio" não é garantia de ampla aceitação de *websites*. Portanto, o conhecimento da diversidade

de usuários pode ser fator determinante para o sucesso de um *website*. Tal conhecimento complementa e, por vezes, redireciona diretrizes e padrões.

- Atributos aurais com CSS;
 - Navegação utilizando JavaScript;
 - Recomendações de Acessibilidade para: Teclas de Atalho;
7. **Considerar diferentes formas de apresentação de páginas Web (dispositivos e configurações)** - *Websites* não são documentos estáticos e, portanto, seriam melhor construídos se fossem considerados como construções flexíveis e a diferentes dispositivos, tamanho de *display* e preferências de visualização de usuários.
- Unidades de medida relativas - Unidades em CSS;
8. **Avaliação e validação** - Devido à característica dinâmica de *websites*, mesmo quando mantenedores conhecem e empregam os padrões e recomendações, a tarefa de manter um *website* atendendo completamente a essas recomendações exige um monitoramento constante. Esse monitoramento pode ser obtido por meio de ferramentas automatizadas de validação de código ou por meio de avaliação manual.
- Validação de código HTML;
 - Validação de folhas de estilo CSS;
 - Verificação de código JavaScript;
 - Formulário de Inspeção Heurística na Web;
 - Avaliação Simplificada de Acessibilidade;
9. **Integrar tecnologias e conceitos durante todo o desenvolvimento** - Um dos grandes problemas no desenvolvimento de *websites* é a lacuna entre as recomendações técnicas e os conceitos relacionados. Um exemplo disso é a recomendação de acessibilidade sobre o fornecimento de texto alternativo a imagens. Apesar de proverem texto alternativo e, portanto, seguirem a diretriz em questão, mantenedores falham na escolha de qual informação deveria estar presente nesse texto e comumente não sabem quais são os usuários que se beneficiam desse recurso. Para tanto é necessária uma abordagem integrada que permita a compreensão não somente das regras de desenvolvimento, mas também das necessidades e dos benefícios gerados por sua aplicação.
- Integração Usabilidade – Acessibilidade.

Tema	•HTML, CSS, JavaScript, Acessibilidade e Usabilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•WARAU
Resultados esperados	•Definição de conceitos e Técnica

Padronização de códigos HTML

Como em qualquer linguagem de programação, sugerimos que todos que escrevem códigos HTML para um dado *website* sigam o mesmo estilo de codificação. Com isso, o HTML ganha legibilidade e todos ficam familiarizados com os códigos feitos por qualquer um dos membros da equipe. A seguir discutiremos a padronização de três importantes componentes de códigos HTML: *tags*, atributos e comentários.

Apesar das tags HTML não serem sensíveis à escrita em letras maiúsculas ou minúsculas, sugerimos que todas as *tags* sejam escritas com letras minúsculas. Esta é uma das recomendações do W3C para HTML e é obrigatória para XHTML, o que torna uma possível migração de HTML para XHTML muito mais fácil. Padronização de código facilita bastante a manutenção.

Basicamente existem dois tipos de *tags* HTML: as vazias e as não vazias. *Tags* vazias não contam com *tag* de fechamento (e.g., `
`) e *tags* não vazias sempre devem utilizar *tags* de abertura e fechamento (e.g., `` e ``).

Atributos fornecem informações adicionais sobre uma *tag*. Eles sempre aparecem na *tag* de abertura para *tags* não vazias e sempre são formados por pares do tipo `nome="valor"`. Por exemplo, `align="left"`, que define um alinhamento à esquerda. Sugerimos que os atributos sejam escritos em letras minúsculas pelo mesmo motivo comentado em relação às *tags*.

Tanto por compatibilidade quanto por legibilidade do código sugerimos que todos atributos estejam entre aspas duplas. Aspas simples também são permitidas e podem ser utilizadas quando há necessidade de utilizar aspas dentro de um atributo.

Exemplo de atributo HTML com aspas simples:

```
alt="Anderson 'Spyder' Silva"
```

Comentários são utilizados para demarcar código que não será mostrado nos navegadores. Procure utilizar comentários somente onde for necessário tornar algo mais claro para a equipe que mantém o *website*. Evite utilizar comentários para inutilizar trechos de código obsoleto, pois isto pode causar erros.

Exemplo de comentário em HTML:

```
<p>
Texto mostrado
<!-- Texto não mostrado -->
</p>
```

Código renderizado

Texto mostrado

Note que no HTML 4.01 é possível encerrar comentários com -- e o sinal de >, não necessariamente concatenados. Comentários fechados desta forma podem dificultar a descoberta de erros e, conseqüentemente, dificultar a manutenção de código. Sugerimos fechar comentários sempre com -->, pois assim seu código ficará válido com versões mais recentes de especificações, por exemplo, XHTML e HTML 5.

Tema	•HTML
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•W3C HTML 5 Specification, W3C HTML 4.01 Specification, W3C XHTML 1.0 Specification, Notepad++ e Bluefish
Resultados esperados	•Padronização de código

Estrutura de documentos HTML

Um arquivo HTML possui as seguintes *tags* básicas: `<html>`, `<head>` e `<body>`. Elas possuem *tag* de abertura e fechamento e cada uma delas deve aparecer somente uma vez por página.

A *tag* `<html>` deve estar após a declaração DOCTYPE e a *tag* de `</html>` deve ser a última *tag* do código HTML. Para a *tag* `<html>` pode ser definido o atributo `lang`, que define o idioma a que o conteúdo se refere.

Exemplo de *tags* básicas do HTML:

```
<!DOCTYPE html>
<html lang="pt-br">
...
</html>
```

A *tag* `<head>` deve estar após a *tag* `<html>` e a *tag* `</head>` deve aparecer antes da *tag* `<body>`.

Exemplo de uso das *tags* `<head>` e `<body>`:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
...
</head>
<body>
...
</body>
</html>
```

Na *tag* `<head>` são inseridas *tags* utilizadas para indexação do documento HTML (e.g., `<title>`, `<meta>`) e relacionamento com documentos externos (e.g., `<link>`).

A *tag* `<title>` é utilizada para definir o título da página, que é um dos principais conteúdos indexados pelas ferramentas de buscas e é o primeiro elemento processado por leitores de tela, contextualizando o usuário sobre o que a página trata.

Procure redigir o conteúdo de forma concisa e sem pontuações que procurem reproduzir detalhes gráficos, também conhecidos como ASCII *art*, pois nestes casos o leitor de telas vai "falar" todas pontuações que estiverem presentes no título.

Contraexemplo de título de página:

```
<title>_.:^ Teste ^:._</title>
```

As *tags* `<meta>` fornecem importantes informações sobre o documento para navegadores, ferramentas de busca, servidores de *proxy*, leitores de tela, entre outros.

Exemplo de tags <meta>:

```
<meta name="author" content="Nome do autor">
<meta name="keywords" content="HTML, Acessibilidade">
<meta name="description" content="Descrição utilizada por ferramentas de busca">
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="expires" content="Thu, 01 Jan 1970 00:00:00 GMT">
<meta http-equiv="cache-control" content="no-store">
```

Além destes exemplos, as tags <meta> podem ser utilizadas para atualizar a página automaticamente e redirecionar o usuário para outra página. No entanto, estas funcionalidades devem ser utilizadas com cuidado, pois essas manipulações devem ser feitas somente se o usuário puder interrompê-las facilmente quando julgar necessário. Caso contrário, podem criar barreiras de acessibilidade e problemas de usabilidade. Para usuários de leitores de tela uma atualização inesperada pode tirar a pessoa da área da página que está utilizando, redirecionando-a para o início da navegação da página. Para verificar este problema experimente navegar utilizando um leitor de telas.

Contraexemplo sobre atualização automática de páginas:

```
<meta http-equiv="refresh" content="60">
<meta http-equiv="refresh" content="1; http://www.unicamp.br/">
```

A tag *link* deve estar entre as tags <head> e </head>. Com ela é possível definir a relação entre documentos, como indicar a relação de uma página com uma folha de estilo, ou definir para quais tipos de dispositivos uma determinada página será utilizada.

O atributo *rel* é o que especifica a relação entre uma página e o documento referenciado.

O atributo *media* é utilizado para especificar quais tipos de dispositivos a página em questão será utilizada. Alguns dos valores possíveis são: *screen*, *tty*, *tv*, *projection*, *handheld*, *print*, *braille*, *aural* ou *all*.

Exemplo de relação entre documentos na tag <link>:

```
<link rel="stylesheet" type="text/css" href="estilo.css"
media="screen, tty, print, braille, aural">
```

A última das tags básicas da estrutura de documentos HTML é a tag <body>, que deve estar após a tag </head>, e a tag </body>, que deve estar antes da tag </html>. Na tag <body> é incluído todo o conteúdo do documento HTML que será exibido no navegador.

Exemplo da tag <body>:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
...
</head>
<body>
...
</body>
</html>
```

Tema	•HTML
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•W3C HTML 4.01 Specification, W3C XHTML 1.0 Specification, Notepad++ e Bluefish
Resultados esperados	•Estruturação de documentos

Designer

Desenvolvedor

Redator

Âncoras e hiperlinks - A tag HTML <a>

As tags <a> e são utilizadas para definir um *hiperlink*. Note que o texto (ou rótulo) do *hiperlink* deve identificar claramente o local para onde o usuário será enviado. Não utilize palavras vagas nem repita o mesmo conteúdo para dois *hiperlinks* distintos. Toda âncora deve ter conteúdo textual ou gráfico associado de forma que todos os usuários possam acessá-la.

Os atributos principais desta tag são: *href*, *target*, *accesskey* e *title*. O atributo *href* indica qual é o destino do *hiperlink*, que pode ser uma página ou uma âncora. Para criar uma âncora basta nomear uma região da página usando o atributo *id*. Para referenciar uma âncora basta colocar o nome da âncora após o sinal #, dentro do atributo *href*.

Um uso comum dessas âncoras para tornar páginas mais acessíveis aos usuários de leitores de tela é nomear uma região do conteúdo principal e incluir um *hiperlink* no início da página que remeta o usuário diretamente para esse conteúdo, evitando que tenha que passar por cada um dos *links* até alcançar o conteúdo principal. Este mecanismo é chamado de *skip link*.

Exemplo de *skip link*:

```
<a href="#conteudo">Ir para conteúdo principal</a>
...
<a id="conteudo">Início do conteúdo</a>
```

Código renderizado

[Ir para conteúdo principal](#)

...

Início do conteúdo

O atributo *accesskey* indica um atalho pelo qual o usuário poderá acessar um *hiperlink* apenas combinando uma tecla especial (e.g., ALT) com o caractere associado a este atributo, sem necessitar do *mouse*. Note que o uso deste atributo deve ser consistente e único para cada *hiperlink* ou função do *website*.

O atributo *title* é diferente da tag <title>. Aqui o atributo indica ao usuário o que há na página/âncora que o *hiperlink* remete, já a tag <title> se refere ao título da página.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Navegação

Designer

Desenvolvedor

Redator

Mapeamentos - A tag HTML <map>

A tag `<map>` é utilizada para definir diferentes *links* para diferentes regiões de uma mesma imagem.

Evite utilizá-la, pois alguns leitores de tela (e.g., Virtual Vision) não conseguem ler o conteúdo do atributo *alt* dos *hiperlinks* dentro de um mapeamento, o que dificulta a navegação para quem utiliza leitores de tela.

No entanto, se utilizar algum tipo de mapeamento, forneça também os mesmos *hiperlinks* em formato textual.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Imagens e animações

Listas - As tags HTML , e <dl>

HTML permite a criação de três tipos básicos de listas: (listas ordenadas), (listas não ordenadas) e <dl> (listas de termos/definições).

As tags e demarcam uma lista ordenada de itens, ou seja, os marcadores dos itens seguem uma ordem, como 1, 2, 3, etc.

As tags e demarcam uma lista não ordenada de itens, ou seja, apenas por tópicos não numerados.

As tags e são utilizadas para demarcar um item de uma lista, seja ela ordenada ou não.

Sempre que for aplicável prefira listas ordenadas, pois estas auxiliam usuários com deficiência visual a navegarem entre os itens.

Estes elementos podem ser aninhados, ou seja, um pode estar dentro do outro. Quando isto ocorre os navegadores apresentam as listas de itens de forma indentada. No entanto, estas marcações devem ser utilizadas somente para listas. Se o intuito for exclusivamente a indentação, deve-se utilizar outro tipo de formatação via CSS.

As tags <dl> e </dl> demarcam uma lista de definições de termos. Dentro delas, cada termo é delimitado por <dt> e </dt>; cada definição por <dd> e </dd>.

Exemplo das tags <dt> e <dd>:

```
<dl>
<dt>Usabilidade</dt>
<dd>É a capacidade de um produto qualquer ser utilizado por seus usuários
de maneira que eles atinjam seus objetivos com eficiência e
satisfação.</dd>
</dl>
```

Código renderizado

Usabilidade

É a capacidade de um produto qualquer ser utilizado por seus usuários de maneira que eles atinjam seus objetivos com eficiência e satisfação.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Navegação e Legibilidade

Designer

Desenvolvedor

Redator

Subtópicos - As tags HTML <h1> a <h6>

Utilize as *tags* <h1>, <h2>, <h3>, <h4>, <h5> e <h6> para estruturar uma página por meio do particionamento do conteúdo em subtópicos. O <h1> deve ser utilizado para os tópicos mais gerais e a partir de cada subtópico deve-se utilizar o número seguinte, por exemplo, <h2>, depois <h3>, e assim sucessivamente até o <h6>.

É importante que os elementos sejam aninhados corretamente de acordo com sua estrutura.

Exemplo de aninhamento de *tags* para subtópicos:

```
<h1>Título</h1>
<h2>Subtítulo</h2>
<h3>Intertítulo</h3>
<h3>Intertítulo</h3>
```

Note que estas *tags* não devem ser utilizadas com o intuito formatar algum conteúdo textual, elas devem indicar uma hierarquia no documento. Dê preferência sempre à formatação via CSS.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Navegação e Legibilidade

Tabulação - O atributo HTML tabindex

Note que a ordem padrão da tabulação é a que os elementos aparecem no código HTML.

O atributo *tabindex* controla a ordem em que o cursor percorrerá os campos de um formulário ao se utilizar a tecla TAB. Este atributo pode ser utilizado em *hiperlinks*, elementos de formulários, entre outros. Uma boa utilização do *tabindex* é em elementos dinamicamente adicionados na página. Dessa maneira é possível alterar a ordem da tabulação para que ela faça mais sentido para quem estiver navegando pelo teclado.

Além da tabulação em HTML, também é necessário verificar se elementos gerados em outras linguagens, como Flash ou Silverlight, oferecem a tabulação em uma ordem que faça sentido ao usuário.

Tema	•Html
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Navegação e Formulários

Frames - A tag HTML <frame>

Tags <frame> são utilizadas juntamente com *tags* <frameset> para dividir o espaço útil da tela em regiões que podem conter páginas HTML diferentes, como navegação e conteúdo.

Recomendamos que este recurso não seja utilizado, pois ele pode tornar a navegação através de um leitor de tela significativamente mais difícil, pois o usuário precisa navegar entre todos os *frames* para conferir o que foi alterado em cada uma das regiões.

Complementarmente, os *frames* possuem baixa compatibilidade entre os diferentes navegadores e comumente causam problemas de usabilidade. Folhas de estilo CSS são uma boa alternativa aos *frames* e eliminam muitos desses problemas.

Tema	•HTML
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Navegação e Codificação

Elementos de bloco e elementos de linha

Há duas formas básicas de diferenciar elementos de bloco (e.g., `<p>`, `<h1>`, `<form>`, `<table>`) de elementos de linha (e.g., ``, ``, `<input>`):

- **Conteúdo** - elementos de bloco podem conter outros elementos de bloco e elementos de linha. Já os elementos de linha somente podem conter outros elementos de linha;
- **Formatação** - elementos de bloco começam em novas linhas, o que não acontece com elementos de linha.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•W3C HTML 4.01 Specification
Resultados esperados	•Definição de conceitos

Agrupamento de elementos de bloco - A tag HTML <div>

A tag <div> delimita elementos de bloco, ou seja, é um elemento de bloco utilizado para encapsular elementos de bloco ou conjuntos de texto com algum propósito especial.

Exemplo da tag <div>:

```
<div class="citacoes">
<p>
<span class="citacao">
O personagem diz <span lang="it">"Buon Giorno Principessa"</span>
</span>
</p>
</div>
```

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Estruturação de documentos e Legibilidade

Agrupamento de texto - A tag HTML

A tag é um elemento de linha utilizado para encapsular um conjunto de texto com algum propósito especial. Por exemplo, para aplicar um determinado estilo em CSS ou para delimitar um conteúdo em uma língua diferente do resto do texto.

Exemplo da tag :

```
<p>  
<span class="citacao">  
O personagem diz <span lang="it">"Buon Giorno Principessa"  
</span>  
</span>  
</p>
```

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Estruturação de documentos e Legibilidade

Quebras de linha - A tag HTML

A tag
 define uma quebra de linha. Ela não possui tag de fechamento.

Exemplo da tag
:

Linha 1
Linha 2
Linha 3

Código renderizado

Linha 1

Linha 2

Linha 3

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Parágrafos - A tag HTML <p>

Utilize as *tags* <p> e </p> para delimitar parágrafos. Use
 somente quando se referir a quebras de linha dentro do mesmo parágrafo.

Para facilitar a leitura, o espaçamento entre linhas de parágrafos deve ser pelo menos de 1,5. E o espaçamento entre parágrafos deve ser de pelo menos de 1,5 o tamanho do espaçamento entre linhas. O alinhamento dos parágrafos **não** deve ser justificado.

Exemplo da *tag* <p>:

```
<p>Parágrafo 1. Linha 1<br>Parágrafo 1. Linha 2</p>  
<p>Parágrafo 2.</p>
```

Código renderizado

Parágrafo 1. Linha 1
Parágrafo 1. Linha 2

Parágrafo 2.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Idiomas - O atributo HTML lang

Para qualquer mudança de idioma no conteúdo textual utilize o atributo *lang* para indicar qual é o idioma do trecho em questão.

Com isso, leitores de tela que suportam diferentes linguagens lerão o trecho de acordo com o idioma indicado e não de acordo com o idioma padrão ou o que foi definido para todo o documento na tag `<html>`.

Exemplo do atributo *lang*:

```
<html lang="pt-br">
...
<p>O personagem diz <span lang="it">"Buon Giorno Principessa"</span></p>
```

Outra funcionalidade deste atributo é que usuários que lerem o conteúdo em Braille contarão com as marcações adequadas e abreviações para cada linguagem.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Textos sobrescritos - A tag HTML <sup>

As tags ^e são utilizadas para demarcar conteúdo sobrescrito, como índices de nota de rodapé e potências matemáticas.

Exemplo da tag <sup>:

```
<p>2<sup>2</sup></p> <br>2<sup>3<sup>4</sup></sup></sup></p>
```

Código renderizado

2²
2^{3⁴}

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Textos subscritos - A tag HTML

As tags _e são utilizadas para fazer com que elementos fiquem subscritos como, por exemplo, o número de átomos em uma molécula de um elemento químico.

Exemplo da tag <sub>:

```
<p>Óxido de ouro - Au<sub>2</sub>O<sub>3</sub></p>
```

Código renderizado

Óxido de ouro - Au₂O₃

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Fontes - A tag HTML

A tag é pouco utilizada atualmente e não pertence ao conjunto de *tags* do XHTML nem do HTML 5, portanto seu uso deve ser evitado. Recomendamos a configuração de fontes utilizando CSS. No entanto, por ser uma *tag* presente em muitos *websites* apresentamos alguns pontos relativos à acessibilidade que podem ser levados em consideração quando sua remoção não for viável.

Utilizando valores inteiros para o atributo *size* fazemos referência a tamanhos pré-definidos. No entanto, quando utilizamos valores inteiros acompanhados dos sinais de + ou - os tamanhos são relacionados às configurações do navegador do usuário.

Portanto sugerimos que ao utilizar a tag os tamanhos sejam sempre especificados de forma relativa às configurações do usuário, pois quando um usuário configura uma fonte de tamanho maior, todo o *website* se adequa aos novos parâmetros.

Exemplo da tag :

```
<font size="+1">...</font>
```

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Codificação

Ênfase no texto - As tags HTML e

As tags e são utilizadas para dar ênfase e estruturar o conteúdo de um documento.

Evite a utilização das tags (negrito) e <i> (itálico), pois elas implicam apenas no destaque visual. As tags e foram projetadas para indicar ênfase e, portanto, podem ser renderizadas de várias formas, inclusive via leitor de telas.

Exemplo das tags e :

```
<p><strong>A grande produção</strong> de <em>adrenalina</em>.</p>
....
<p><strong>Não</strong> use <em>clique aqui</em>.</p>
```

Código renderizado

A grande produção de *adrenalina*.

...

Não use *clique aqui*.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Abreviações e siglas - As tags HTML <abbr> e <acronym>

As tags <abbr> e <acronym> são utilizadas para demarcar abreviações e siglas, respectivamente. O atributo `title` é usado para apresentar a forma completa do termo reduzido. No HTML 5 a tag <abbr> deve ser usada em qualquer um dos casos, pois <acronym> caiu em desuso.

Exemplo das tags <abbr> e <acronym>:

```
<p><abbr title="Universidade Estadual de Campinas">Unicamp</abbr></p>
<p><acronym title="Código de Endereçamento Postal">CEP</acronym></p>
```

Código renderizado

Unicamp

CEP

Outra utilização da tag <abbr> é para tornar conteúdos do tipo ASCII art acessíveis.

Exemplo:

```
<abbr title="smiley">:-)</abbr>
```

Código renderizado

:-)

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Média
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Cores - Os atributos HTML color e bgcolor

O atributo *color* é utilizado para definir a cor de texto e o atributo *bgcolor* é utilizado para definir a cor de fundo de várias *tags* como `<body>`, `<table>`, `<td>`, entre outras. Não utilize como valor destes atributos os nomes das cores (e.g., *red*, *black*, *white*, entre outros), pois é uma definição ultrapassada e não é utilizada nos padrões mais recentes.

O valor destes atributos deve ser em hexadecimal, ou seja, utilizando 16 valores (de 0 a 9 e a, b, c, d, e, f).

Recomendamos a utilização de propriedades CSS para a definição de cores, pois facilita a manutenção e ajuda a deixar o design consistente.

Exemplo dos atributos *color* e *bgcolor*:

```
<table bgcolor="#0000bb">
<tr>
<td>coluna 1</td>
<td><font color="#00ff00">coluna 2</font></td>
</tr>
</table>
```

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Réguas horizontais - A tag HTML <hr>

A tag <hr> não possui tag de fechamento. Ela define uma régua horizontal e é utilizada para auxiliar na distinção de blocos de conteúdo.

Exemplo da tag <hr>:

```
<p>Texto do primeiro bloco.</p>
<hr>
<p>Texto do segundo bloco.</p>
```

Código renderizado

Texto do primeiro bloco.

Texto do segundo bloco.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Baixa
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Textos alternativos

Textos alternativos devem ser fornecidos para cada elemento da página que não for baseado em texto, por exemplo: imagens, vídeos, animações e áudio.

Se o elemento for um vídeo, pode ser fornecida uma descrição textual geral, legenda dos diálogos e uma audiodescrição do conteúdo do vídeo.

Se o elemento for uma imagem então deve ser fornecido um texto alternativo para o conteúdo da imagem.

Se o elemento for um CAPTCHA - para verificação se a página está sendo acessada por um ser humano e não uma máquina - então o usuário deve ter disponível não somente a verificação por imagens, mas também por outro recurso que use um mecanismo perceptual diferente, tal como áudio.

Se o elemento for somente para decoração ou estiver invisível na página - por exemplo: bordas e fundos de tela -, então não deve ser criado texto alternativo.

Tema	•HTML e Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Notepad++, Bluefish e WAI
Resultados esperados	•Legibilidade

Imagens - A tag HTML

A tag é utilizada para adicionar imagens ao documento HTML. Ela não possui tag de fechamento.

Um dos principais atributos da tag , no que diz respeito à acessibilidade, é o atributo *alt*, para o texto alternativo. Seu conteúdo deve refletir o que está apresentado na imagem ou na ação associada à imagem, ou seja, ser um texto equivalente ao conteúdo da imagem.

Exemplo da tag :

```
  
<a href="servicos.html"></a>
```

O conteúdo do atributo *alt* é utilizado em vários cenários. Entre eles: quando as imagens estão desabilitadas, quando o usuário utiliza leitor de telas ou quando o usuário utiliza um navegador baseado em texto (e.g., Lynx).

Para que um código HTML seja válido todas as imagens devem contar com o atributo *alt*, assim, para imagens decorativas ou estruturais este atributo deve ser vazio (i.e., *alt=""*).

Não recomendamos a utilização de imagens para representar textos com efeitos gráficos. Em vez disso, recomendamos a utilização atributos CSS.

Por fim, não utilize imagens animadas a não ser que o usuário tenha uma forma de interromper o movimento.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++, Bluefish e Lynx
Resultados esperados	•Legibilidade e Imagens e animações

Elementos multimídia - A tag HTML <object>

A tag <object> é usada para incluir elementos de multimídia em uma página HTML, por exemplo, animações em Flash, vídeos, etc.

Ao utilizar estes elementos multimídia deve-se oferecer conteúdo equivalente em texto e/ou áudio, isto é, texto alternativo. Isso pode ser alcançado adicionando uma ou mais das alternativas:

- descrição geral na tag <object>, conforme exemplo abaixo;
- legenda sincronizada com a animação ou vídeo;
- audiodescrição da animação ou vídeo;
- interpretação em linguagem de sinais, no caso brasileiro em LIBRAS, sincronizada com a animação, vídeo ou áudio.

Outro aspecto muito importante é a relação de elementos que piscam na tela. Nenhum elemento deve piscar mais que três vezes por segundo, caso contrário pode trazer riscos à saúde das pessoas, como provocar ataques epiléticos. Um exemplo do potencial destrutivo dessa falha é o que aconteceu em dezembro de 1997 no Japão (Fonte: Flávio Calazans). Ao assistir um episódio do desenho Pokemon, no qual havia diversas luzes piscando, mais de 700 pessoas (sendo a maioria crianças) foram parar no pronto-socorro com sintomas como: ataques convulsivos, vômitos, hemorragias, olhos injetados, vertigens e desmaios. No dia seguinte, mais de 12.000 estudantes faltaram às aulas (somente em Tóquio), a maioria decorrente de problemas relacionados ao desenho.

Exemplo da tag <object>:

```
<object id="ex22_slides" data="apresentacao.swf">
Apresentação de slides sobre utilização de HTML válido utilizando aspectos
de usabilidade.
...
</object>
```

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Imagens e animações

Formulários - A tag HTML <form>

As tags `<form>` e `</form>` são utilizadas para delimitar quais campos fazem parte de um formulário. Seus principais atributos são: *action*, *method* e *name*.

O atributo *action* indica qual é a página que receberá os dados do formulário.

O atributo *method* pode ter os valores *get* ou *post*. No *get* os dados do formulário vão juntamente com a URL. Garanta que a URL resultante não tenha comprimento maior que 255 caracteres, caso contrário, navegadores e servidores de *proxy* mais antigos podem não suportar essas URLs (Fielding et al., 1997). Uma alternativa interessante é utilizar *cookies*, pois podem ser recuperados via JavaScript (*client-side*) ou via linguagem de programação *server-side*. A seguir um exemplo de uma URL de destino recebendo dados de um formulário via *get*.

Exemplo de URL destino de um formulário via *get*:

```
https://www.exemplo.com.br/cadastro.php?nome=Carlos&Idade=33
```

No *post* os dados são enviados juntamente com a requisição e o limite é configurável no servidor *Web*.

Note que se o formulário utilizar dados que não são caracteres (e.g., *upload* de arquivos) então o atributo *method* do formulário deve ser *post*.

O atributo *name* é utilizado para nomear o formulário para que ele possa ser referenciado dentro da página. Lembre-se que a especificação da linguagem XHTML 1.0 *Strict* não conta com o atributo *name* para a tag `<form>`.

Tema	•HTML
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification e Notepad++
Resultados esperados	•Estruturação de documentos e Formulários

Campos de entrada de dados - A tag HTML <input>

A tag `<input>` é utilizada para definir um campo de entrada de dados em um formulário. Por meio do seu atributo `type` é possível escolher qual é o tipo de elemento a ser utilizado. Os principais valores para o atributo `type` são: `text`, `checkbox`, `radio` e `submit`.

Além do atributo `type` deve-se utilizar os atributos `id`, `name` e `value`.

O atributo `id` serve para identificar o elemento e, entre outras coisas, possibilitar o uso da tag `<label>`.

O atributo `name` é utilizado para referenciar o elemento no formulário via programação no cliente ou no servidor. O atributo `value` indica qual é o valor inicial do campo.

Quando for necessário indicar que um campo está selecionado, use o atributo `checked` com valor `checked`.

Exemplo da tag `<input>`:

```
<label for="ex24_idade">Idade:</label>
<input type="text" id="ex24_idade" name="campo_idade" value="18"/>
<input type="submit" id="ex24_enviar" name="botao_enviar" value="Enviar"/>
```

Código renderizado

Idade:

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Formulários

Rótulos - A tag HTML <label>

Sempre utilize a tag <label> para associar um rótulo ao seu respectivo campo de um formulário.

A associação é feita por meio dos atributos *id* da tag <input> e *for* da tag <label>.

Para destacar um elemento como obrigatório inclua a marcação dentro das tags <label>.

Note que quando usa um rótulo, ele se torna um elemento clicável. E quando clicado, o foco é aplicado no campo de formulário a que fez referência.

Exemplo da tag <label>:

```
<label for="ex25_nome">* Nome:</label>
<input type="text" name="nome" id="ex25_nome" value="João"/>
```

Código renderizado

* Nome:

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Formulários

Caixas de seleção - A tag HTML <select>

A tag <label> permite a criação de uma caixa de seleção de itens. Para permitir a seleção de vários itens use o atributo *multiple*. O número de itens exibidos simultaneamente na caixa é definido pelo atributo *size*.

Para adicionar um item à caixa use a tag <option>. Para indicar que um item da caixa de seleção está selecionado use o atributo *selected* com valor *selected*.

Exemplo da tag <select>:

```
<label for="ex26_personagens">Personagens</label>
<select multiple="multiple" size="3" id="ex26_personagens">
<option value="1">He-man</option>
<option value="2">Teela</option>
<option value="3">Mentor</option>
<option value="4">Feiticeira</option>
<option value="5">Gorpo</option>
</select>
```

A tag <optgroup> deve ser utilizada para agrupar tags <option>. Ela permite reunir elementos, estruturá-los e facilitar sua identificação.

Exemplo da tag <optgroup>:

```
<label for="ex26_herois">Heróis/heróínas</label>
<select multiple="multiple" size="3" id="ex26_herois">
<optgroup label="Heróis">
<option value="1">He-man</option>
<option value="5">Gorpo</option>
<option value="3">Mentor</option>
</optgroup>
<optgroup label="Heroínas">
<option value="2">Teela</option>
<option value="4">Feiticeira</option>
</optgroup>
</select>
```

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Formulários

Campos de texto - A tag HTML <textarea>

As tags <textarea> e </textarea> delimitam uma área de entrada de texto extenso. Os atributos *rows* e *cols* definem o número de linhas e colunas, respectivamente.

Exemplo da tag <textarea>:

```
<label for="ex27_texto">Texto</label>
<textarea rows="5" cols="15" id="ex27_texto">
A tag textarea permite a criação de campos de entrada de texto de várias
linhas.
</textarea>
```

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Formulários

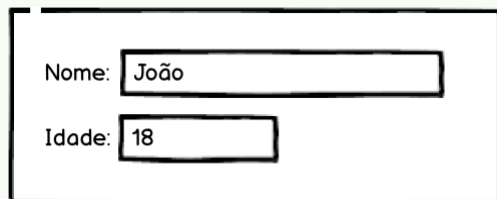
Agrupamento de elementos de formulário - A tag HTML <fieldset>

A tag <fieldset> deve ser utilizada para agrupar elementos de formulários correlacionados.

Exemplo da tag <fieldset>:

```
<fieldset>
<label for="ex28_nome">Nome:</label>
<input type="text" name="nome" id="ex28_nome" value="João" />
<label for="ex28_idade">Idade:</label>
<input type="text" name="idade" id="ex28_idade" value="18" />
</fieldset>
```

Código renderizado



Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Formulários

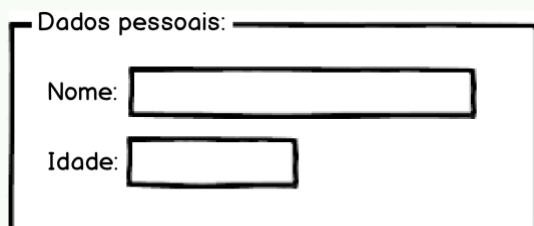
Rótulos para fieldsets - A tag HTML <legend>

A tag <legend> é utilizada para atribuir um rótulo a um <fieldset>.

Exemplo da tag <legend>:

```
<fieldset>
<legend>Dados pessoais</legend>
<label for="ex29_nome">Nome:</label>
<input type="text" name="nome" id="ex29_nome" value="" />
<label for="ex29_idade">Idade:</label>
<input type="text" name="idade" id="ex29_idade" value="" />
</fieldset>
```

Código renderizado



Dados pessoais:

Nome:

Idade:

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Formulários

Tabelas - As tags HTML <table>, <td>, <tr>, <th> e <caption>

Procure utilizar tabelas somente para dados tabulares. Nesses casos, sempre faça uso do atributo *summary* contendo uma descrição da tabela, pois assim usuários que utilizam leitores de tela poderão saber do que se trata sem terem que ler a tabela célula a célula.

Evite utilizar tabelas para dados não tabulares, por exemplo, somente para posicionar elementos. Mas, se utilizar para esta finalidade, não coloque nenhuma informação sobre o conteúdo nem estrutura da tabela (e.g., *summary*, <caption>, <th>, etc.). Faça também com que as tabelas sejam linearizáveis, ou seja, ao ignorar toda a informação de linhas e colunas e renderizar todo o conteúdo como se fosse uma série de parágrafos o conteúdo deve ser facilmente entendido.

A tag <td> é utilizada para demarcar o conteúdo de uma célula de tabela.

A tag <tr> é utilizada para demarcar o conteúdo de uma linha de tabela.

Exemplo da tag <table>:

```
<table summary="Esta tabela mostra a evolução da cotação do dólar turismo
nos últimos 12 meses, mês a mês.">
<tr>
<td>R$ 3,97</td><td>R$ 4,02</td>
</tr>
</table>
```

Código renderizado

R\$ 3,97	R\$ 4,02
----------	----------

A tag <th> é utilizada para demarcar o conteúdo do cabeçalho de uma coluna.

O conteúdo do atributo *id* da tag <th> é utilizado pelas células das linhas seguintes para referenciar seu cabeçalho. Com isto quando um leitor de telas processa cada célula ele pode repetir o conteúdo da célula referenciada pelo *id* e assim contextualizar a informação lida.

Exemplo da tag <th>:

```
<table summary="Esta tabela mostra a evolução da cotação do dólar turismo
nos últimos 12 meses, mês a mês.">
<tr>
<th id="ex31a_col_jun">Junho</th>
<th id="ex31a_col_jul">Julho</th>
</tr>
<tr>
<td headers="ex31a_col_jun">R$ 3,97</td>
<td headers="ex31a_col_jul">R$ 4,02</td>
</tr>
</table>
```

Código renderizado

Junho	Julho
R\$ 3,97	R\$ 4,02

Complementarmente, o atributo *abbr* pode ser utilizado nos casos em que o conteúdo da tag `<th>` for extenso, o que tornaria o processamento de uma tabela por um leitor de telas muito cansativo para o usuário.

*Exemplo do atributo **abbr** na tag `<th>`:*

```
<table summary="Esta tabela mostra a evolução da cotação do dólar turismo
nos últimos 12 meses, mês a mês.">
<tr>
<th id="ex31_col_totaljun" abbr="Média Junho">Média da cotação do dólar
pelo Banco Central no mês de junho.</th>
<th id="ex31_col_totaljul" abbr="Média Julho">Média da cotação do dólar
pelo Banco Central no mês de julho.</th>
</tr>
<tr>
<td headers="ex31_col_totaljun">R$ 3,97</td>
<td headers="ex31_col_totaljul">R$ 4,02</td>
</tr>
</table>
```

Código renderizado

Média da cotação do dólar pelo Banco Central no mês de junho.	Média da cotação do dólar pelo Banco Central no mês de julho.
R\$ 3,97	R\$ 4,02

As tags `<caption>` e `</caption>` são utilizadas para definir uma legenda de uma tabela.

Exemplo da tag `<caption>`:

```
<table summary="Esta tabela mostra a evolução da cotação do dólar turismo
nos últimos 2 meses.">
<caption>Cotação do dólar turismo</caption>
<tr>
<th id="ex31b_col_jun">Junho</th>
<th id="ex31b_col_jul">Julho</th>
</tr>
<tr>
<td headers="ex31b_col_jun">R$ 3,97</td>
<td headers="ex31b_col_jul">R$ 4,02</td>
</tr>
</table>
```

Código renderizado

Cotação do dólar turismo	
Junho	Julho
R\$ 3,97	R\$ 4,02

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•W3C HTML 4.01 Specification, Notepad++ e Bluefish
Resultados esperados	•Tabelas

Designer

Desenvolvedor

Redator

Validação de código HTML

Para validar páginas HTML, o W3C disponibiliza o Markup Validation Service (<http://validator.w3.org/>) que é uma ferramenta *online*, onde basta informar a URL, fazer *upload* do arquivo ou informar o código diretamente, para receber uma relação de possíveis erros, segundo a versão do HTML que esteja sendo utilizada na página.

Tema	•HTML
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramenta utilizada	•W3C Markup Validation Service
Resultados esperados	•Avaliação

Sintaxe CSS

Sintaxe básica do CSS - A sintaxe do CSS é composta de dois itens principais: o seletor que é a referência ao elemento que receberá o estilo, e a lista dos atributos que é composta de pares atributo-valor separados por ponto e vírgula e delimitadas por chaves. Tem assim a seguinte forma:

```
seletor {atributo: valor; [atributo: valor;]}
```

Exemplo de declaração atributo-valor CSS:

```
body {background-color: #00ff00;}  
p {font-family: "sans serif";}
```

* Note o uso das aspas para nomes compostos

Agrupamento - O CSS permite que você agrupe seletores que tenham propriedades em comum.

Exemplo de agrupamento de seletores CSS:

```
h1, h2, h3, h4, h5, h6 {color: #00ff00; margin-bottom: 10px;}
```

Seletor class - Permite que você defina mais de um estilo para um mesmo elemento HTML. A seguir exemplo para duas definições para o elemento parágrafo.

No CSS:

Exemplo de seletor *class* em CSS:

```
p.citacao {text-align: right;}  
p.paragrafoComum {text-align: left;}  
p.destaque {background-color: #00ff00;}
```

No HTML:

Exemplo de uso do seletor *class* CSS no HTML:

```
<p class="paragrafoComum">Parágrafo comum.</p>  
<p class="citacao">Parágrafo alinhado à direita.</p>
```

Também é possível combinar duas ou mais classes.

Exemplo de uso combinado de classes CSS:

```
<p class="citacao destaque">  
Parágrafo alinhado à direita com fundo verde.  
</p>
```

Outro recurso é o de omitir o nome do elemento HTML na classe. Dessa forma é possível utilizar a classe para qualquer elemento que possua as propriedades utilizadas.

No CSS:

Exemplo de omissão do elemento HTML na classe CSS:

```
.destaque {background-color: #00ff00;}
```

No HTML:

Exemplo de uso de classe CSS com omissão do elemento HTML:

```
<p class="destaque">Parágrafo com fundo verde.</p>  
<div class="destaque">Bloco com com fundo verde.</div>
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools CSS2 Reference
Resultados esperados	•Padronização de código e Estruturação de documentos

Inclusão de código CSS

CSS permite que você defina a forma de exibição dos elementos de marcação de arquivos HTML, XHTML, entre outros. Normalmente são criados arquivos .css que podem ser reutilizados e, com isso, você pode economizar muito trabalho. Além disso, o código da página que utiliza CSS fica mais claro, pois os elementos de decoração ficam separados do conteúdo.

Declaração interna do CSS - Recomendado quando o estilo for único para o documento HTML. Use comentários HTML para manter compatibilidade com navegadores antigos. Este recurso faz com que os navegadores antigos que não suportam CSS não processem a folha de estilo nem apresentem seu conteúdo como sendo texto.

Exemplo de declaração interna do CSS:

```
...
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<style type="text/css"><!--
hr {color: #000000;}
p {margin-left: 20em;}
body {background-image: url("images/back40.gif");}
--></style>
</head>
...
```

Declaração externa do CSS - É a forma mais recomendada, pois permite o melhor reuso e clareza do documento HTML.

Exemplo de declaração externa do CSS:

```
...
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<link rel="stylesheet" type="text/css" href="estilos.css">
</head>
...
```

Note que em casos como este a folha de estilo não necessita estar dentro dos comentários CSS.

Declaração inline - Somente recomendado quando o estilo é único para o elemento HTML.

Exemplo de declaração *inline* do CSS:

```
...
<p style="color: #000000; margin-left: 20em;">
Este é um parágrafo
</p>
...
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools CSS2 Reference
Resultados esperados	•Padronização de código, Estruturação de documentos e Compatibilidade de código

Designer

Desenvolvedor

Redator

Estrutura de documentos CSS

Caso sejam feitas várias definições de estilo sobre um mesmo elemento, essas definições serão *cascadeadas* formando um só elemento. Por isso, a combinação dos arquivos .css e de suas definições deve ser feita de forma muito cuidadosa. O cascadeamento segue a seguinte ordem (note que o maior número tem maior prioridade):

1. Padrão do navegador;
2. *Style sheet* externo;
3. *Style sheet* interno (entre as *tags* <head> e </head>);
4. *Inline style* (no atributo *style* de uma *tags*).

Dentro de cada nível da cascata, o estilo será aplicado com base no nível de especificidade do seletor CSS. A representação a seguir mostra as especificidades dos seletores e os pesos atribuídos. Quanto maior for o peso calculado maior será a especificidade do seletor.

Exemplo de cálculo de pesos de seletores CSS:

```
strong { color: #ff0000; } /* id=0 classe=0 tag=1 -> peso = 1 */
p strong { color: #00ff00; } /* id=0 classe=0 tag=2 -> peso = 2 */
div p strong { color: #0000ff; } /* id=0 classe=0 tag=3 -> peso = 3 */
strong.fantasia { color: #000000; } /* id=0 classe=1 tag=1 -> peso = 11 */
#cabecalho { color: #336699; } /* id=1 classe=0 tag=0 -> peso = 100 */
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Estruturação de documentos

Padronização de código CSS

Padronização facilita a identificação de erros e ajuda na manutenção de código bem escrito. Dessa forma sugerimos que as definições das propriedades sejam separadas e terminadas por ponto e vírgula.

Exemplo de declaração padronizada em CSS:

```
div.content { padding: 5%; margin: 5%; }
```

Em estilo *in-line*, sugerimos que todas as definições sejam terminadas com ponto e vírgula. Isso ajuda na manutenção e inclusão de novas propriedades, além de facilitar uma eventual migração de estilos *in-line* para um arquivo CSS melhor estruturado.

Contraexemplo de declaração *inline* em CSS:

```
<td style="font-weight: bold">
```

Exemplo de declaração *inline* em CSS:

```
<td style="font-weight: bold;">
```

Ao nomear seletores nunca se deve associar o nome à aparência do elemento. Tente sempre identificar o seletor a partir da função ou do dado a que ele se refere.

Contraexemplo de declaração CSS baseada na aparência:

```
<td class="corVermelha">01/01/2007</td>
```

Contraexemplo de declaração CSS baseada no conteúdo:

```
<td class="texto1">01/01/2007</td>
```

Exemplo de declaração CSS baseada na função do dado:

```
<td class="data">01/01/2007</td>
```

Por fim, devido a problemas de compatibilidade na interpretação de CSS entre diferentes navegadores, especialmente para os mais antigos, sugerimos que atalhos (i.e., definição de vários atributos para uma propriedade) sejam usados de maneira cautelosa, sempre verificando se são compatíveis com os navegadores usados. Exemplos de atalhos que funcionam em navegadores mais antigos são: *margin*, *padding* e *border*.

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Padronização de código, Compatibilidade de código e Codificação

Designer

Desenvolvedor

Redator

Prioridade para estilos do usuário

O operador *!important* é usado em CSS para forçar a aplicação de um determinado estilo. No entanto, ele descarta toda a estrutura e pesos de seletores aplicados. Dessa forma, não use *!important*, pois ele é um indicativo que há problemas na estruturação do código CSS, uma vez que foi necessário forçar a aplicação de um determinado estilo.

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Validation Service, Notepad++ e Bluefish
Resultados esperados	•Padronização de código e Estruturação de documentos

Unidades de medida relativas - Unidades em CSS: Porcentagem e em

Utilize sempre unidades de medida relativas para que a apresentação da página possa se adequar à saída, da melhor forma possível. As unidades de medida que sugerimos são porcentagens e *em*. Note que a unidade de medida do CSS e a *tag* `` do HTML são duas coisas diferentes. No CSS a unidade *em* é utilizada para referenciar a letra *M* do tamanho da fonte que estiver configurada no dispositivo do usuário. No HTML, a *tag* `` é utilizada para dar ênfase a conteúdos textuais.

O principal objetivo de utilizar unidades de medida relativas é fazer com que a formatação possa se adequar às configurações e ao dispositivo do usuário, para *displays* maiores e menores que o projetado inicialmente. Lembre-se que atualmente há pessoas que acessam *websites* desde celulares com poucas polegadas a TVs com mais de 40 polegadas.

Unidades de medida fixa (e.g., px) devem ser usadas somente quando o elemento a ser definido contiver algo com medida fixa (e.g., imagens não vetoriais).

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Codificação

Conteúdos gerados pelo CSS: os pseudo-elementos *before* e *after*

A geração de conteúdo utilizando CSS é uma característica do CSS nível 2, o que indica que deve ser utilizada com cautela e acompanhada de testes com os navegadores mais utilizados pelo público alvo. A geração de conteúdo possibilita adicionar informações sobre o contexto de elementos que são destacados por atributos visuais como borda, imagens ou cores de fundo.

Além dos pseudo-elementos *before* e *after*, o CSS nível 2 conta com as propriedades *cue*, *cue-before* e *cue-after*, que possibilitam que sons sejam tocados para auxiliar na identificação de conteúdo. Outras formas mais compatíveis de fornecer contexto e aumentar a acessibilidade de uma página *Web* são: a partir da utilização dos atributos *display* e *visibility* para esconder elementos de contexto que são redundâncias para informações visuais, ou tornar um texto desses pequeno e de cor igual a cor de fundo. Conforme comentado anteriormente, verifique sempre a compatibilidade das propriedades e elementos que deseja utilizar com os navegadores utilizados no seu *website*. A referência sugerida é Referência de CSS do W3 Schools.

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Validation Service, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Codificação

Listas ordenadas utilizando CSS

Sugerimos a utilização de listas ordenadas (*tag* ``) em vez das listas não ordenadas (*tag* ``). No entanto, ao utilizar listas numeradas aninhadas surge um problema de contextualização, pois o número de cada item representa apenas sua posição em relação aos outros que estão no mesmo nível. Para uma pessoa que somente ouve o conteúdo de uma lista dessas pode se confundir com a numeração porque diferentes itens podem estar sob o mesmo índice.

Assim, uma maneira de resolver este problema seria adicionar um número para cada nível de profundidade e assim obter itens da forma 1, 1.2, 1.2.3, etc. Mas não há uma maneira fácil de incluir esta informação em HTML, e é aí que entra a utilização de CSS.

Exemplo de CSS para listas ordenadas com subníveis:

```
ul, ol { counter-reset: item; }
li { display: block; }
li:before { content: counters(item, ".") ; counter-increment: item; }
```

Outra forma de melhorar a acessibilidade de uma lista é incluir marcações que auxiliem na navegação entre seus itens. Assim, além da numeração comentada anteriormente é possível incluir uma marcação para indicar o final de cada seção.

No CSS:

Exemplo de marcações de fim de listas ordenadas usando CSS:

```
span.fimDeLista { display: none; }
```

No HTML:

Exemplo de marcações de fim de listas ordenadas usando CSS no HTML:

```
<ol>
<li>
  Padronização
  <ol>
    <li>Marcação</li>
    <li>Folha de Estilo</li>
    <li>Scripts <span class="fimDeLista">Fim das linguagens</span></li>
  </ol>
</li>
<li>Usabilidade</li>
<li>Acessibilidade <span class="fimDeLista">Fim da lista</span></li>
</ol>
```

Código renderizado

1. Padronização
 1. Marcação
 2. Folha de Estilo
 3. Scripts
2. Usabilidade
3. Acessibilidade

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Navegação e Legibilidade

Designer

Desenvolvedor

Redator

Réguas - Alterando a aparência com CSS

No HTML as réguas horizontais (<hr>) são indicadas para separar e delimitar blocos de conteúdo. Com a utilização de CSS elas se transformam em um elemento muito versátil para a separação de conteúdo. Propriedades de CSS como borda, estilos de borda, cor, margem e imagem de fundo, fazem com que sua aparência possa ser adaptada de várias formas.

No CSS:

Exemplo de alteração da aparência de réguas usando CSS:

```
hr.reguaDeInstrucoes {  
    margin: 1em 0 1em 0;  
    height: 1em;  
    border: none;  
    background-image: url("estrela.gif");  
    background-repeat: no-repeat;  
    background-position: center center;  
}  
hr:before { content: attr(title); }
```

No HTML:

Exemplo de alteração da aparência de réguas usando CSS no HTML:

```
<hr title="Fim das instruções" class="reguaDeInstrucoes">
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Validation Service, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Bordas utilizando CSS

Através das propriedades oferecidas pelo CSS é possível atribuir vários tipos de formatação que sem sua utilização seriam possíveis apenas com imagens.

Além de possibilitar alterações na cor, tamanho e estilo, é possível editar cada um dos quatro lados dos elementos de bloco (e.g., `<table>`, `<div>`, `<p>`, entre outros).

Exemplo de definição de bordas usando CSS:

```
div {  
  border-top: 0.2em dotted #000000;  
  border-right: 0.2em dotted #cccccc;  
  border-bottom: 0.3em double #000000;  
  border-left: 0.3em double #cccccc;  
}
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Posicionamento utilizando CSS

Uma das tarefas mais difíceis, senão a mais difícil, é fazer com que os elementos de uma página *Web* fiquem posicionados corretamente em diferentes navegadores. Existem algumas diferenças muito pequenas de implementação de certas propriedades que podem tornar a implementação 100% consistente um grande desafio. No entanto, a utilização de CSS para posicionamento é a mais adequada, pois centraliza todas as configurações da aparência do *website*. Deve-se buscar um alto nível de consistência entre os diferentes navegadores utilizados pelos usuários do *website*. Note que o alto nível comentado deve ser definido pela equipe de desenvolvimento e manutenção das páginas.

Através das propriedades *float*, *position*, *top*, *right*, *bottom* e *left* é possível alterar o posicionamento de quase todos os elementos. No entanto, esta flexibilidade deve ser utilizada com cautela, pois se o navegador não suportar CSS, o documento deve seguir uma ordem lógica, sem prejudicar sua leitura. É importante frisar que quando nos referimos a navegadores que não suportam CSS estamos nos referindo, da maneira mais ampla possível, a qualquer aplicação que possa receber conteúdo *Web*.

As propriedades indicadas para definir a distância entre os elementos de bloco são: *margin*, *margin-top*, *margin-right*, *margin-bottom* e *margin-left*.

E as propriedades para definir a distância entre a borda dos elementos e seu conteúdo são: *padding*, *padding-top*, *padding-right*, *padding-bottom* e *padding-left*.

Assim, para definir o CSS de uma tabela com margem superior e inferior de 0.5em e células com espaçamento de 0.1em entre o conteúdo e a borda teríamos o seguinte CSS:

Exemplo de posicionamento usando CSS:

```
table { margin-top: 0.5em; margin-bottom: 0.5em; }
table tr td { padding: 0.1em; }
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Validation Service, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Fontes utilizando CSS

Sempre utilize uma família de fonte genérica, pois elas estão presentes na grande maioria dos navegadores.

Exemplo de definição de fontes usando CSS:

```
body { font-family: verdana, sans-serif; }
```

A utilização do atalho para a manipulação do estilo da fonte (`font: bold 12px arial;`) deve ser usada com cautela, pois, assim como outras definições, deve ter sua compatibilidade testada antes. Em navegadores antigos a propriedade *text-transform* não funciona quando usada dentro de atalho. No entanto, esta é uma situação bastante específica. Como regra geral mantenha seu código válido e faça com que seus usuários não sejam prejudicados por problemas de compatibilidade dos navegadores.

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade, Compatibilidade de código e Codificação

Efeitos em texto utilizando CSS

Para utilizar letras maiúsculas ou minúsculas para dar ênfase ou para padronizar *layout* utilize a propriedade *text-transform*. Note como é feita a separação entre os dados e sua apresentação.

Contraexemplo de uso de letras maiúsculas no HTML:

```
<strong>DESTAQUES DO EVENTO</strong>
```

No CSS:

Exemplo de transformação de texto para letras maiúsculas usando CSS:

```
strong.destaque { text-transform: uppercase; }
```

No HTML:

Exemplo de transformação de texto para letras maiúsculas usando CSS no HTML:

```
<strong class="destaque">Destaques do evento</strong>
```

Por fim, imagine um *website* milhares ou milhões de páginas que utilizam algumas informações em letras maiúsculas. Agora como seria a alteração dessas informações para minúsculas se as páginas não utilizassem CSS?

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Validation Service, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Formatação e posicionamento de texto utilizando CSS

Utilize CSS para alcançar a formatação que deseja. Não utilize espaços nem pontuações para enfatizar trechos de texto, pois provavelmente uma pessoa que utilizar um leitor de telas não receberá a mesma ênfase e, talvez, fique até difícil de entender o conteúdo. As propriedades *word-spacing* e *letter-spacing* controlam o espaçamento entre as palavras e entre as letras, respectivamente.

Contraexemplo de ênfase em texto usando espaços e letras maiúsculas:

A T E N Ç Ã O

No CSS:

Exemplo de ênfase em texto usando CSS:

```
em.mensagem { letter-spacing: 150%; text-transform: uppercase; }
```

No HTML:

Exemplo de ênfase em texto usando classe CSS no HTML:

```
<em class="mensagem">Atenção</em>
```

Por fim, em vez de utilizar a tag `<center>` para controlar o alinhamento de texto, use a respectiva propriedade de CSS da seguinte forma: `text-align: center`.

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Legibilidade

Utilizando texto em vez de imagem

Não utilize imagens que contenham apenas texto. O CSS fornece várias propriedades para alteração da aparência de textos e utilizar apenas imagens para representá-los restringe o acesso para usuários que utilizam *display* em Braille, leitores de tela e outras tecnologias assistivas que se baseiam no conteúdo textual.

Contraexemplo de uso de imagens para representar texto:

```
<p>
<a href="pagina_2.html">

</a>
</p>
```

No CSS:

Exemplo de texto com aparência definida no CSS:

```
a.paginacao {
    font-family: verdana, sans-serif;
    font-weight: bold;
    letter-spacing: 150%;
}
```

No HTML:

Exemplo de uso de classe CSS para alterar a aparência do texto no HTML:

```
<p><a href="pagina_2.html" class="paginacao">Próxima página</a></p>
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Validation Service, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Codificação

Atributos aurais com CSS

As propriedades relacionadas aos atributos aurais fazem parte do CSS nível 2 e, conforme outras propriedades que podem ter baixa compatibilidade, deve ser verificada entre os navegadores que seus usuários utilizam. No entanto, essas propriedades são promissoras no que diz respeito ao aumento da usabilidade e da acessibilidade, pois com elas é possível formatar a forma de como o conteúdo textual é falado por leitores de tela como alterar volume, incluir ícones sonoros, alterar o *pitch* da voz, se um conteúdo deve ser lido ou soletrado, entre outros. Para verificar outras propriedades veja referência em Referência de CSS2 Aural do W3 Schools.

Exemplo de atributos aurais no CSS:

```
h1, h2, h3, h4, h5, h6 { volume: 130%; voice-family: male; }
```

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Specification, Notepad++ e Bluefish
Resultados esperados	•Audibilidade

Cores com CSS

Sugerimos que defina uma cor de fundo para todos os elementos da página. Uma forma bem estruturada de controlar essa propriedade de atribuir a cor desejada ao elemento principal da página (e.g., *body*) e para outros elementos contidos na página fazer com que essa cor seja herdada ou que os elementos contidos tenham o fundo transparente. Assim, quando for necessária uma alteração, a cor de fundo está concentrada em uma declaração e a alteração fica mais fácil. Note que o contraste entre a cor de fundo e de texto deve ser suficiente para facilitar a leitura. Para validar se a sua escolha de cores oferece contraste adequado você pode utilizar ferramentas como o Luminosity Colour Contrast Ratio Analyser.

Exemplo de definição de cor de fundo e de texto usando CSS:

```
p.destaque { color: #000000; background-color: #ffffff; }  
p.destaque p.titulo { color: #663399; background-color: transparent; }
```

A redundância é um dos pontos-chaves no desenvolvimento de páginas acessíveis. Note que o termo redundância é usado no sentido de que a informação está disponível para a máxima extensão possível da população. Assim, nunca utilize somente cores para identificar elementos ou contextualizar quaisquer informações. Uma forma simples de testar o contraste de sua página *Web* é imprimir a página em preto e branco, incluindo imagens e cores de fundo.

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on, W3C CSS Validation Service, Notepad++ e Bluefish
Resultados esperados	•Compatibilidade de código e Técnica

Validação de folhas de estilo CSS

Para validar folhas de estilo CSS, o W3C disponibiliza o CSS Validation Service (<http://jigsaw.w3.org/css-validator/>) que é uma ferramenta *online*, em que basta informar a URL, fazer *upload* do arquivo ou informar o código diretamente, para receber uma relação de possíveis erros, segundo a versão do CSS que esteja sendo utilizada na página.

Tema	•CSS
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramenta utilizada	•W3C CSS Validation Service
Resultados esperados	•Avaliação

Recomendações para uso do JavaScript

JavaScript deve ser usado para oferecer **funcionalidades adicionais** às já existentes e necessárias ao funcionamento da página *Web*. Dessa forma, caso o navegador do cliente não ofereça suporte a *scripts* a funcionalidade da página não será afetada.

O JavaScript permite que você informe a versão com que você deseja que seu código seja interpretado e também permite que você identifique a versão do navegador do cliente. No entanto, escrever códigos para versões restritas do JavaScript não é uma opção muito boa (apesar de amplamente adotada). O que sugerimos é a padronização e adoção de boas práticas de codificação que tornem o código JavaScript menos sensível a versões dos navegadores e versões da própria linguagem.

A utilização de *scripts* também levanta questões relacionadas à acessibilidade. Existe a percepção que páginas que utilizam *scripts* não são acessíveis e que é impossível desenvolver páginas acessíveis usando *scripts*. Ambas as afirmações estão incorretas e, neste site, são apresentadas formas de criar páginas acessíveis utilizando *scripts*. O problema principal está na maneira como *scripts* são usados como única forma de adicionar determinadas funcionalidades.

JavaScript deve ser usado de maneira redundante e oferecer funcionalidades adicionais. Ele não deve ser a única tecnologia usada para oferecer funcionalidades básicas de um *website*.

Tema	•JavaScript
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Compatibilidade de código e Codificação

Inclusão de código JavaScript

Os códigos em JavaScript utilizados em uma página podem ser internos ou externos. Para utilização de código interno sugerimos que a estrutura utilizada seja a seguinte:

Exemplo de inclusão de JavaScript internamente no HTML:

```
<script type="text/javascript"><!--  
código JavaScript  
//--></script>
```

Priorize a declaração de códigos JavaScript no início dos documentos HTML, dentro da *tag* `<head>`. Com isso, você melhora a legibilidade do código e evita erros de escopo de declaração de variáveis e ordem de métodos.

Note que os comandos `<!--` e `-->` localizados após as tags *script* de abertura e fechamento, respectivamente, são usados para que código JavaScript não seja mostrado por navegadores que não suportam *scripts*. Esta solução está caindo em desuso com a evolução dos navegadores, mesmo assim é sempre bom conhecer alternativas para problemas já conhecidos.

A inserção de *scripts* externos pode ser feita da seguinte forma:

Exemplo de inclusão de documento Javascript no HTML:

```
<script type="text/javascript" src="codigo.js"> </script>
```

Note que arquivos externos não precisam iniciar com os comandos utilizados nos *scripts* internos, pois o arquivo externo só será interpretado se o navegador suportar JavaScript.

Tema	•JavaScript
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código, Estruturação de documentos e Compatibilidade de código

Declaração de variáveis em JavaScript

Todas as variáveis devem ser declaradas, com os tipos apropriados para o algoritmo, não importando se são globais ou não.

Exemplo de declaração de variáveis em Javascript:

```
var x = new Number( 1 );
```

Outro ponto a ser lembrado é que as variáveis devem ser utilizadas dentro do escopo em que foram criadas.

Contraexemplo de uso de variáveis fora de seu escopo:

```
function Valida(pa,pb) {  
    ...  
    if (pa) {  
        ...  
        var soma = new Number( 0 );  
        soma = pb;  
    } else {  
        soma = 1;  
    }  
    ...  
};
```

Exemplo de uso de variáveis com o escopo correto:

```
function Valida(pa,pb) {  
    var soma = new Number( 0 );  
    ...  
    if (pa) {  
        ...  
        soma = pb;  
    } else {  
        soma = 1;  
    }  
    ...  
};
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código e Compatibilidade de código

Ordem da declaração de variáveis e métodos em JavaScript

A declaração de métodos e variáveis deve ser ordenada de forma que eles sejam utilizados somente após terem sido declarados. Uma boa prática é manter o código JavaScript na *tag* `<head>`.

Quando não respeitada a ordem de declaração e uso de variáveis e métodos você fica sujeito a problemas de compatibilidade entre versões de JavaScript e diferentes formas de interpretação do código realizadas pelos navegadores.

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código, Estruturação de documentos e Codificação

Quebras de linha no código JavaScript

De acordo com a especificação da linguagem as quebras de linhas são suficientes para separar expressões, mas devido às diferenças existentes entre as implementações de JavaScript *client-side* deve-se, sempre, utilizar ponto-e-vírgula ao final de todas expressões.

Contraexemplo de declaração em Javascript sem ponto-e-vírgula:

```
<script type="text/javascript"><!--  
var x = new Number( 1 )  
x = 2  
//--></script>
```

Exemplo de declaração em Javascript usando ponto-e-vírgula:

```
<script type="text/javascript"><!--  
var x = new Number( 1 );  
x = 2;  
//--></script>
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código e Compatibilidade de código

Cuidados com versões da linguagem JavaScript

Não utilize versão da linguagem na declaração do *script*. Diferentes implementações de JavaScript podem utilizar objetos diferentes na mesma versão, e se isso não for verificado pode resultar em erros. O ideal é verificar se o suporte a um objeto existe e, só depois disso, utilizá-lo no *script*. Assim, o código se torna independente da versão e funciona corretamente em futuras versões, desde que suportem o objeto utilizado. Se não se sentir seguro em verificações deste tipo ou se preferir não ter que fazê-la, pense na possibilidade de utilizar bibliotecas de *script* que já façam esse tipo de verificação para você.

Tema	•JavaScript
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código e Compatibilidade de código

Detecção de navegador, objetos e métodos utilizando JavaScript

Sugerimos que evite fazer verificações relacionadas ao navegador utilizado pelo usuário, uma vez que restringem a execução do código JavaScript a um grupo muito específico, além de não levar em conta usuários de navegadores que serão lançados. Lembre-se que quanto mais tempo seu código "sobreviver" menos manutenção ele necessitará.

Contraexemplo de código Javascript dependente de navegador:

```
<script type="text/javascript"><!--  
var browserName = navigator.appName ;  
if ( browserName == "Firefox" ){  
...  
}  
//--></script>  
...
```

Em vez disso, para garantir que o código será executado corretamente, sugerimos utilizar a detecção de objetos e métodos dessa forma:

Exemplo de teste de compatibilidade de objeto Javascript sem dependência de navegador:

```
<script type="text/javascript"><!--  
if ( document.getElementById ){  
    // O objeto getElementById pode ser utilizado  
}  
else{  
    // O objeto não pode ser utilizado  
}  
//--></script>
```

Além da detecção de objetos sugerimos a utilização da detecção de métodos. Note que para detecção de métodos não se deve utilizar parênteses. Usar parênteses indica a utilização do método e não a verificação de sua existência.

Contraexemplo de detecção de método Javascript com uso de parênteses:

```
<script type="text/javascript"><!--  
// Verifica se é possível colocar o foco na janela,  
// assumindo que o método pode ser utilizado  
if ( window.focus() ){  
...  
}  
//--></script>
```

Exemplo de detecção de método Javascript:

```
<script type="text/javascript"><!--  
// Verifica se existe suporte ao método focus  
if ( window.focus ){  
...  
}  
//--></script>
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código e Compatibilidade de código

Designer

Desenvolvedor

Redator

Conversão de tipo de dados em JavaScript

Operadores de comparação estritos (`===` ou `!==`) não devem ser utilizados. Sugerimos que, como boa prática de programação, a conversão de variáveis de tipos diferentes deve ser feita pelo programador. Com isso fica claro no código JavaScript quais são os tipos utilizados.

Contraexemplo de comparação de variáveis Javascript de tipos diferentes:

```
<script type="text/javascript"><!--
var x = new String( "1" ) ;
...
y = new Number ( 1 ) ;
if ( x == y ){
...
}
//--></script>
```

Exemplo de conversão de tipo de variável Javascript antes de comparação:

```
<script type="text/javascript"><!--
var x = new String( "1" ) ;
...
y = new Number ( 1 ) ;
if ( new Number ( x ) == y ){
...
}
//--></script>
```

As implementações da versão 1.2 do Javascript não convertem os tipos antes de uma comparação, já as implementações da versão 1.3 do Javascript possuem o mesmo comportamento que a versão 1.1, ou seja, convertem tipos diferentes em tipos compatíveis antes de uma comparação. Assim, sugerimos que a conversão seja feita pelo programador para que assim o código seja mantido sem erros, independente da versão do navegador utilizada pelo cliente. Para verificar versões de JavaScript utilizadas em algumas versões do *Internet Explorer* e do *Netscape Navigator* acesse JavaScript Guidelines and Best Practices (<https://www.irt.org/articles/js169/#2.7>)

Faça com que o tipo do valor retornado pelas funções JavaScript seja adequado para a lógica do programa. Assim, as operações aplicadas a esses valores podem ser feitas diretamente, o que torna o programa mais legível e mais estruturado.

Exemplo de retorno de função com tipo de dados compatível com a lógica da função:

```
function getLinha(pval){
    var linha=new String();
    linha = pval;
    ...
    return new Number( linha ) ;
}
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-one W3Schools JavaScript Reference
Resultados esperados	•Estruturação de documentos e Compatibilidade de código

Designer

Desenvolvedor

Redator

Manipuladores de evento em HTML

A partir da versão 4.0 o HTML possui recursos para disparar eventos por meio do navegador. Um exemplo seria o evento *onkeypress* que é executado toda vez que o usuário pressiona e libera uma tecla. Quando você estiver utilizando manipuladores de eventos lembre-se de incluir manipulares para diferentes dispositivos. Assim, se houver uma função disparada a partir apenas do *onclick*, então se deve disponibilizar a mesma funcionalidade a partir do *onkeypress*, e vice-versa.

Exemplo de manipuladores Javascript de eventos para diferentes dispositivos:

```
<script type="text/javascript">!--
function selecionou(porigem) {
    if (porigem == "M") {
        alert("Selecionou o link pelo mouse.");
    } else if (porigem == "T") {
        alert("Selecionou o link pelo teclado.");
    }
};
//--></script>
...
<a title="Clique para testar a redundância de manipuladores" href="#"
onclick="selecionou('M')" onkeypress="selecionou('T');">Clique para testar
a redundância de manipuladores</a>
...
```

Tema	•HTML, JavaScript e Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e Lynx
Resultados esperados	•Compatibilidade de código e Codificação

Desvios e laços condicionais em JavaScript

De acordo com a especificação da linguagem, as chaves não são necessárias para laços e condicionais simples, ou seja, contendo apenas uma expressão. No entanto sugerimos a utilização de chaves mesmo nesses casos, pois desta forma se obtém um código mais legível e mais portátil.

Contraexemplo de não uso de chaves para blocos condicionais e laços em Javascript:

```
<script type="text/javascript"><!--  
if ( ... )  
    // apenas uma expressão  
//--></script>
```

Exemplo de uso de chaves para blocos condicionais e laços em Javascript:

```
<script type="text/javascript"><!--  
if ( ... ){  
    // uma ou mais expressões;  
}  
//--></script>
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código, Estruturação de documentos e Compatibilidade de código

Escrita de HTML via JavaScript

No lugar de executar vários comandos de impressão (e.g., `document.write`), o que pode tornar o *script* muito lento, concatene a saída em uma variável e por fim envie o conteúdo da variável de uma única vez.

Contraexemplo de impressão em Javascript:

```
<script type="text/javascript"><!--
document.write( "<table>" ) ;
document.write( "<tr><td>" ) ;
document.write( "teste" ) ;
document.write( "<\/td><\/tr>" ) ;
document.write( "<\/table>" ) ;
//--><\/script>
```

Exemplo de impressão em Javascript usando concatenação:

```
<script type="text/javascript"><!--
var html_code = "<table>"
html_code += "<tr><td>" ;
html_code += "teste" ;
html_code += "<\/td><\/tr>" ;
html_code += "<\/table>" ;
document.write( html_code ) ;
//--><\/script>
```

Tema	•HTML e JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código e Codificação

Aspas simples e duplas em JavaScript

Conteúdo de texto deve estar sempre entre aspas duplas (") e todas ocorrências delas no conteúdo devem ser escapadas com a barra invertida (\).

Contraexemplo de uso de aspas em Javascript:

```
<script type="text/javascript"><!--  
document.write( 'Conteúdo de texto. "Teste", teste.' ) ;  
//--></script>
```

Exemplo de uso de aspas em Javascript:

```
<script type="text/javascript"><!--  
document.write( "Conteúdo de texto. \"Teste\", teste." ) ;  
//--></script>
```

Aspa simples ou apóstrofe (') deve ser utilizada somente quando o código JavaScript estiver dentro de um atributo de uma tag HTML.

Exemplo de uso de aspas simples em Javascript:

```
...<a href="." onClick="alert( function( 'conteúdo string' ) ) ;">...
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código

ETAGO em JavaScript

Deve-se escapar a barra de qualquer ETAGO (</) que estiver sendo escrito pelo JavaScript. Isso evita que o *parser* identifique um ETAGO de outra *tag* como sendo a de *script*.

Contraexemplo de impressão de ETAGO em Javascript:

```
<script type="text/javascript"><!--  
document.write( "<p>Teste</p>" ) ;  
//--></script>
```

Exemplo de impressão de ETAGO em Javascript:

```
<script type="text/javascript"><!--  
document.write( "<p>Teste<\/p>" ) ;  
//--></script>
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on
Resultados esperados	•Padronização de código e Compatibilidade de código

Escrevendo a palavra reservada script

Quando a palavra *script* for processada dentro de código JavaScript ela deve ser dividida e concatenada, para que o *parser* não a identifique como sendo a *tag script* que encerra o código JavaScript.

Contraexemplo de impressão da palavra script:

```
...
<script type="text/javascript"><!--
document.write( "<script language=\"javascript\" type=\"text/javascript\"
src=\"codigo.js\"></script> " ) ;
//--></script>
...
```

Exemplo de impressão da palavra script:

```
...
<script type="text/javascript"><!--
document.write( "<scr" + "<ipt language=\"javascript\"
type=\"text/javascript\" src=\"codigo.js\"></scr" + "<ipt> " ) ;
//--></script>
...
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código, Compatibilidade de código e Codificação

Manipulação de CSS utilizando JavaScript

A combinação de JavaScript, CSS e HTML é muito utilizada em *websites*. Esta combinação possibilita que programas em JavaScript alterem a aparência (CSS), dados e/ou estrutura (HTML) de páginas *Web*.

Para manipular folhas de estilo via JavaScript basta alterar as propriedades do objeto *style* dos elementos ou alterar o atributo *class*.

Exemplo de manipulação de CSS utilizando Javascript:

```
...
<script type="text/javascript"><!--
var name_element = document.getElementById( "name" ) ;
name_element.style.backgroundColor = "#00ff99" ;
name_element.style.borderColor = "#000000" ;
//--></script>
...
```

Tema	•CSS e JavaScript
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código

Navegação utilizando JavaScript

Programas em JavaScript são muito utilizados para manipular menus e sub-menus em navegações de *websites*, mas utilizações cuidadosas desse recurso são menos comuns. Se o conteúdo utilizado nesses menus só está disponível para usuários que utilizam JavaScript surge um problema de acessibilidade, já que parte do conteúdo não será acessível. Sendo assim, um *script* que controla elementos relacionados à navegação deve adicionar funcionalidades em vez de resultar em barreiras para parte dos usuários.

Algumas das formas de se utilizar JavaScript em navegações de *websites* são a utilização de recursos minimizar/maximizar ou mostrar/esconder. Estas estratégias dizem respeito à disponibilização de elementos de interface de usuário que possibilitam que os usuários possam filtrar a apresentação visual dos itens de navegação. Com funcionalidades de minimizar/maximizar ou mostrar/esconder o usuário pode, por exemplo, verificar os itens acerca de uma sessão presente na navegação ou acessar um item de interesse dentro de uma lista simplificada com muitos elementos.

Exemplo de melhoria de navegação usando Javascript:

```
...
<ol id="ex415_navegacao">
<li id="ex415_graduacao">
<a href="#" onclick="open_close( new Array( 'ex415_graduacao_menu' ) );
return false;">Graduação</a>
<ol id="ex415_graduacao_menu">
<li><a href="#">Cursos</a></li>
<li><a href="#">Institutos</a></li>
</ol>
</li>
<li id="ex415_pos_graduacao"><a href="#" onclick="open_close( new Array(
'ex415_pos_graduacao_menu' ) ); return false;">Pós-graduação</a>
<ol id="ex415_pos_graduacao_menu">
<li><a href="#">Mestrado profissional</a></li>
<li><a href="#">Mestrado acadêmico</a></li>
<li><a href="#">Doutorado</a></li>
</ol>
</li>
</ol>
<script type="text/javascript"><!--
function open_close( id_array ){
    if ( !document.getElementById ){
        return ;
    }
    for ( i = 0 ; i < id_array.length ; i++ ){
        if ( document.getElementById( id_array[i] ) ){
            if ( document.getElementById( id_array[i] ).style.display == "none"
){
                document.getElementById( id_array[i] ).style.display = "block" ;
            }
            else{
                document.getElementById( id_array[i] ).style.display = "none" ;
            }
        }
    }
}
open_close( new Array( "ex415_graduacao_menu", "ex415_pos_graduacao_menu" ) )
```

```
) ;  
//--></script>  
...
```

Código renderizado

...

1. Graduação
 1. Cursos
 2. Institutos
2. Pós-graduação
 1. Mestrado profissional
 2. Mestrado acadêmico
 3. Doutorado

...

Tema	•JavaScript
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Navegação, Codificação e Técnica

Validação de formulários utilizando JavaScript

Uma das funções mais interessantes do uso de JavaScript é a de validação de formulários sem a necessidade de envio de informações para o servidor. No entanto, é necessário lembrar que a validação também deve estar presente no servidor.

No exemplo a seguir temos o exemplo de validação de um *input* que deve conter somente números inteiros. A validação é feita com uma expressão regular que diz que o valor deve ser obrigatoriamente composto por números.

Exemplo de validação de formulários usando Javascript:

```
...
<script type="text/javascript">!--
var expressao = /^\\d+$/;
function verificarInteiro( val ){
    if( expressao.test( val ) ){
        alert( "OK" );
    }
    else{
        if( val != null && val != "" ){
            alert( "Não é um inteiro válido" );
        }
    }
}
//--></script>
<form id="frmInteiro" action="#"
onsubmit="verificarInteiro(this.txtInteiro.value); return false;">
<div>
<label for="txtInteiro">Valor:</label>
<input type="text" size="10" id="txtInteiro" name="txtInteiro">
<input type="submit" value="Validar">
</div>
</form>
...
```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Formulários e Codificação

Acesso a conteúdo e redundância utilizando JavaScript

A execução de *scripts* pode ser desabilitada por várias razões tais como segurança, limitações do navegador ou por opção do usuário (comum em usuários com deficiência visual). Portanto, é interessante que sejam fornecidas outras formas de acesso ao conteúdo que seria apresentado pelo código JavaScript. Para tanto pode ser utilizada a tag `<noscript>` que é executada sempre que *scripts client-side* estejam desabilitados.

Exemplo de uso da tag `<noscript>`:

```
...
<script type="text/javascript"><!--
document.write("Texto escrito via javascript!");
//--></script>
<noscript>
<p>Texto alternativo</p>
</noscript>
...
```

Tema	•JavaScript
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Compatibilidade de código e Codificação

Gráficos e animações utilizando JavaScript

JavaScript também pode ser utilizado para produzir efeitos como animações. Para ilustrar essa funcionalidade apresentamos o código de um *ticker*. Uma característica desse código é que ele permite a visualização das mensagens de forma estática quando o JavaScript não está disponível, sem fazer uso do elemento `<noscript>`. O intuito é mostrar que, apesar de ser uma boa opção para garantir a redundância de funcionalidades, o elemento `<noscript>` não é o único recurso disponível.

Exemplo de animação utilizando Javascript:

```
...
<div id="ex418_caption">
<p>Universidade Estadual de Campinas</p>
<p>Websites Atendendo a Requisitos de Acessibilidade e Usabilidade</p>
<p>Experimente desabilitar o JavaScript do seu navegador</p>
</div>
<script type="text/javascript"><!--
var message_counter = 0 ;
var char_counter = 0 ;
var messages = new Array() ;
//Torna o objeto invisível
function init( obj_id ){
    if ( !document.getElementById ){
        return ;
    }
    if ( document.getElementById( obj_id ) ){
        document.getElementById( obj_id ).style.display = "none" ;
    }
}
//Carrega as mensagens que serão apresentadas no ticker
function get_messages( noscript_container ){
    if ( !document.getElementById ){
        return ;
    }
    var obj = document.getElementById( noscript_container ) ;
    var p_array = obj.getElementsByTagName( "p" ) ;
    for ( var i = 0 ; i < p_array.length ; i++ ){
        messages.push( p_array[i].innerHTML ) ;
    }
    obj.innerHTML = "" ;
    obj.style.display = "block" ;
}
//Mostra as mensagens carregadas no ticker
function play_caption(){
    if ( !document.getElementById ){
        return ;
    }
    //Limpa o ticker caso esteja começando uma nova mensagem
    if ( char_counter == 0 ){
        document.getElementById( "ex418_caption" ).innerHTML = "" ;
    }
    //Mostra o próximo caracter da mensagem
    if ( document.getElementById( "ex418_caption" ) ){
        if ( char_counter < messages[ message_counter ].length ){
            document.getElementById( "ex418_caption" ).innerHTML +=
messages[message_counter].charAt( char_counter++ ) ;
            setTimeout( 'play_caption()', 150 ) ;
        }
    }
}
```

```

    }
    else{
        char_counter = 0 ;
        message_counter++ ;
        //Intervalo antes de iniciar uma nova mensagem
        if ( message_counter < messages.length ){
            setTimeout( 'play_caption()', 2000 ) ;
        }
        //Intervalo antes de iniciar a primeira mensagem
        else{
            message_counter = 0 ;
            setTimeout( 'play_caption()', 2000 ) ;
        }
    }
}
}
// Chamada às funções
init( 'ex418_caption' ) ;
get_messages( 'ex418_caption' ) ;
setTimeout( "play_caption()", 1000 ) ;
//--></script>
...

```

Tema	•JavaScript
Relevância para designers	•Baixa
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Imagens e animações e Codificação

Manipulação de HTML utilizando JavaScript

Para alterar conteúdo de *tags* HTML, a maneira mais simples é utilizar o identificador da *tags* para acessar o respectivo objeto. Após acessar o objeto basta utilizar o método `innerHTML` para acessar e manipular o conteúdo da *tags*.

Exemplo de alteração de código HTML utilizando Javascript:

```
...  
<script type="text/javascript">!--  
var name_element = document.getElementById( "name" ) ;  
name_element.innerHTML = name_element.innerHTML + "_ " ;  
//--></script>  
...
```

Tema	•HTML e JavaScript
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código

AJAX

Ao contrário do que alguns pensam AJAX não é uma linguagem. AJAX é uma técnica que usa JavaScript para trocar informações em XML entre a página que você está vendo e um servidor, assincronamente. Depois disso fica mais fácil entender o acrônimo AJAX que vem do inglês *Asynchronous JavaScript And XML*. Isso mesmo, JavaScript Assíncrono E XML.

AJAX é comumente usado para evitar que páginas completas ou informações não atualizadas sejam baixadas do servidor e carregadas no cliente. No entanto, note que isto é apenas um exemplo e muito mais pode ser feito com esta técnica.

Imagine que você baixou e está vendo no seu computador uma página com uma lista com 100 mensagens e segundos depois chega uma nova mensagem no servidor de onde você baixou estas mensagens. Sem AJAX, provavelmente, você teria que atualizar toda a página e baixar todas as 101 mensagens para poder visualizar a nova. Com AJAX é possível, por exemplo, baixar somente a mensagem mais nova. Assim, em vez de carregar toda a página e todas as 101 mensagens, será necessário apenas baixar uma mensagem. Muito mais rápido!

A utilização de XML como linguagem de marcação para os dados trocados entre o cliente e o servidor não é por acaso. XML é uma linguagem de marcação muito portátil e permite troca de informações entre as mais diversas tecnologias e plataformas. Mesmo assim, outras opções como JSON (*JavaScript Object Notation*) ou mesmo texto são possíveis. Para quem leva o acrônimo ao pé-da-letra estes dois casos resultariam em AJAJ e AJAT, respectivamente. Trocando o "X" de XML por "J" de JSON e "T" de Text.

O exemplo a seguir usa o objeto XMLHttpRequest do JavaScript para fazer conexões HTTP com o servidor. A sequência de *trys* e *catchs* é usada, pois alguns navegadores tratam o XMLHttpRequest de maneira diferente. O formato usado para troca de informações é HTML para deixar o exemplo mais simples.

Exemplo de busca dinâmica utilizando AJAX:

```
<script type="text/javascript"><!--
function getSuggestions(q) {
    var xmlhttp;
    try { xmlhttp=new XMLHttpRequest(); } catch (e) {
        try { xmlhttp=new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) {
            try { xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); } catch (e) {
                return false;
            }
        }
    }
    xmlhttp.onreadystatechange = function() {
        if( xmlhttp.readyState == 4 ){
            try{
                document.getElementById("ex421_topicSearchResults").innerHTML =
xmlhttp.responseText ;
            }
            catch( e ) { }
        }
    } ;
    xmlhttp.open( "GET", "sites/default/files/examples/ajax-suggest.php?q=" + q
```

```

, true ) ;
xmlHttp.send( null ) ;
}
//--></script>
<form action="/files/examples/ajax-suggest.php">
<fieldset id="ex421_busca">
<label for="ex421_input">Procure por um tópico</label>
<input id="ex421_input" type="text" onkeyup="getSuggestions(this.value);"
name="q" /></fieldset>
<div id="ex421_topicSearchResults">
</div>
</form>

```

Neste exemplo, os três elementos principais são: um formulário HTML, que contém o `<input>` para o usuário buscar conteúdo e o `<div>` onde serão apresentados os resultados da busca; uma função JavaScript *getSuggestions*, que efetua a conexão assíncrona com o servidor a cada vez que um caractere é inserido no `<input>`; um programa PHP (`/sites/default/files/examples/ajax-suggest.php`) no lado do servidor, que procura os títulos que contêm o texto passado pela variável "q" e devolve uma lista em HTML.

O trecho que usa o manipulador de evento *onreadystatechange* define o que deve acontecer quando o estado for igual a 4, ou seja, quando a resposta do servidor tiver sido concluída. Neste caso, o conteúdo retornado (*responseText*) é inserido diretamente no elemento `<div>` dos resultados.

Tema	•JavaScript
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on
Resultados esperados	•Técnica

Manipulação de HTML utilizando JavaScript

Para alterar conteúdo de *tags* HTML, a maneira mais simples é utilizar o identificador da *tags* para acessar o respectivo objeto. Após acessar o objeto basta utilizar o método `innerHTML` para acessar e manipular o conteúdo da *tags*.

Exemplo:

```
...  
<script type="text/javascript">!--  
var name_element = document.getElementById( "name" ) ;  
name_element.innerHTML = name_element.innerHTML + "_ " ;  
//--></script>  
...
```

Tema	•HTML e JavaScript
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Web Developer Add-on e W3Schools JavaScript Reference
Resultados esperados	•Padronização de código

Verificação de código JavaScript

Para validar código JavaScript, pode-se utilizar o JSLint (<http://www.jslint.com/>) que é uma ferramenta *online*, em que basta informar o código diretamente, para receber uma relação de possíveis erros e recomendações segundo boas práticas.

Tema	•JavaScript
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramenta utilizada	•JSLint The JavaScript Verifier
Resultados esperados	•Verificação

Reaproveitamento de Código

Websites são aplicações extremamente dinâmicas, exigindo um grande volume de atualizações. Além disso, grande parte das equipes de desenvolvimento *Web* é formada por pequenos grupos (2 ou 3 pessoas). Portanto, o reaproveitamento de código é essencial neste cenário.

As linguagens de programação, em geral, oferecem recursos para facilitar o reaproveitamento de código (e.g., comandos *import*, *include*). A seguir apresentamos algumas recomendações para o reaproveitamento de código.

Em HTML:

- Separar elementos que aparecem em diversas telas (e.g., cabeçalho, rodapé, menu) em documentos separados.

Contraexemplo de documento HTML que não propicia reaproveitamento de código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="pt-br">
<head>
<meta name="author" content="João">
<meta name="keywords" content="HTML, Acessibilidade">
<meta name="description" content="Página de teste">
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="expires" content="Thu, 01 Jan 1970 00:00:00 GMT">
<meta http-equiv="cache-control" content="no-store">
<title>Página inicial</title>
</head>
<body>
...
</body>
</html>
```

O exemplo a seguir usa o comando *include* de *Server Side Include* para incluir um arquivo de cabeçalho.

Exemplo de declaração de HTML reaproveitando código para cabeçalho:

```
[Arquivo cabecalho.inc]
<meta name="author" content="João">
<meta name="keywords" content="HTML, Acessibilidade">
<meta name="description" content="Página de teste">
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="expires" content="Thu, 01 Jan 1970 00:00:00 GMT">
<meta http-equiv="cache-control" content="no-store">
```

```
[Arquivo index.html]
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="pt-br">
<head>
<!--#include virtual="/cabecalho.inc" -->
<title>Página inicial</title>
</head>
<body>
...
</body>
</html>
```

Em CSS:

- Nomear seletores de acordo com sua função (ver Padronização de código CSS);
- Usar a estrutura em cascata do CSS para reaproveitar seletores (ver Estrutura de documentos CSS).

Contraexemplo de seletores CSS que não propiciam reaproveitamento:

```
...
p.exemplo { font-family: verdana, sans-serif; font-weight: bold;
margin: 0.2em 0 0.2em 0; padding: 0.2em; color: #00ff00; }
p.contraExemplo { font-family: verdana, sans-serif; font-weight: bold;
margin: 0.2em 0 0.2em 0; padding: 0.2em; color: #ff0000; }
...
```

Exemplo de declaração de seletores CSS utilizando a estrutura em cascata:

```
...
p { font-family: verdana, sans-serif; font-weight: bold;
margin: 0.2em 0 0.2em 0; padding: 0.2em; }
p.exemplo { color: #00ff00; }
p.contraExemplo { color: #ff0000; }
...
```

Em JavaScript:

- Separar arquivos javascript dos documentos HTML, evitando a duplicação de código;
- Dividir o código em funções menores e parametrizadas para serem utilizadas em vários locais.

Contraexemplo de Javascript para reaproveitamento de código:

```
...
<script type="text/javascript">!--
function isLoginFormfilled(){
  if( document.getElementById("email").value == "" ||
document.getElementById("password").value == "" ){
    return false ;
  }
  else{
    return true;
  }
}
//--></script>
...
```

Exemplo de Javascript para reaproveitamento de código:

```
...
<script type="text/javascript">!--
function isFormFilled( formObject ){
  for( var i = 0 ; i < formObject.elements.length ; i++ ){
    if( formObject.elements[i].value == "" ){
      return false ;
    }
  }
  return true ;
}
//--></script>
...
```

Tema	•HTML, CSS e JavaScript
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Notepad++ e Bluefish
Resultados esperados	•Padronização de código e Estruturação de documentos

O que é Usabilidade?

Resumidamente, um produto tem boa usabilidade se pode ser utilizado por seus usuários de forma que eles atinjam seus objetivos com eficiência e satisfação. No entanto, o termo é padronizado e também pode ser definido a partir da relação de múltiplos componentes. Nielsen (Nielsen, 1993) define usabilidade a partir de cinco atributos, são eles:

- **Facilidade de aprendizagem** - O sistema precisa ser fácil de aprender de forma que o usuário possa rapidamente começar a interagir. Segundo Nielsen, é o mais importante atributo de usabilidade, pois está relacionado à primeira experiência que qualquer usuário tem com um sistema. Este fator é avaliado em função do tempo que o usuário leva para se tornar experiente na execução de suas tarefas;
- **Eficiência** - O sistema precisa ser eficiente no uso, de forma que, uma vez aprendido, o usuário tenha um nível elevado de produtividade. Portanto, eficiência refere-se a usuários experientes, após certo tempo de uso;
- **Facilidade de relembrar** - A forma de utilização do sistema precisa ser fácil de relembrar. Assim, quando o usuário retornar depois de certo tempo, saberá como usá-lo sem ter que aprender novamente como utilizá-lo. Aumentar a facilidade de aprendizagem também torna a interface mais fácil de ser lembrada, mas tipicamente usuários que retornam a um sistema são diferentes dos usuários principiantes;
- **Erros** - Erro é definido como uma ação que não leva ao resultado esperado. O sistema precisa ter uma pequena taxa de erros, ou seja, o usuário não deve ser levado a cometer muitos erros durante o seu uso. Se ele errar, deve conseguir retornar a um estado livre de erros, sem perder qualquer coisa que tenha feito. Erros em que o usuário perde trabalho ou que ele não percebe que ocorreu um erro são exemplos de casos que não podem ocorrer;
- **Satisfação subjetiva** - Os usuários devem gostar do sistema e devem ficar satisfeitos ao usá-lo. A satisfação é muito relevante quando se considera sistemas usados fora do ambiente de trabalho, tais como jogos, sistemas domésticos em geral, etc., uma vez que estes não têm caráter obrigatório como o que ocorre em sistemas adotados no ambiente de trabalho.

Antes de apresentar a definição utilizada na norma ISO 9241-11 (International Standards Organization (ISO), 1998) é necessário apresentar alguns elementos que ela utiliza. São eles:

- **Usuário** é quem interage com o produto;
- **Contexto de uso** envolve os usuários, tarefas, equipamentos (hardware, software e materiais), ambiente físico e social em que o produto é usado;
- **Eficácia** é dada pela relação dos usuários que atingiram seus objetivos;
- **Eficiência** é relação da eficácia com a quantidade dos recursos gastos;
- **Satisfação** é estabelecida pelo conforto e aceitabilidade do produto por parte dos usuários. Ela pode ser calculada por meio de métodos subjetivos e/ou objetivos.

A partir destes elementos a atual abordagem de usabilidade foi definida como a *"capacidade de um produto ser usado por usuários específicos para atingir objetivos"*

específicos com eficácia, eficiência e satisfação em um contexto específico de uso" (International Standards Organization (ISO), 1998).

Tema	•Usabilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•Useit. Usability 101: Introduction to Usability
Resultados esperados	•Definição de conceitos

Designer

Desenvolvedor

Redator

Integração Usabilidade - Acessibilidade

A integração de acessibilidade e usabilidade deve ser buscada, pois a falta da combinação desses conceitos pode resultar em diferentes barreiras para os usuários. Um *website* pode ser fácil de usar para quem não possui nenhum tipo de limitação e não contar com requisitos de acessibilidade, o que impede que parte de seu público-alvo o utilize. Complementarmente, um *website* que implementa requisitos de acessibilidade pode ter baixa usabilidade se seus usuários, com algum tipo de limitação ou não, encontram dificuldades durante sua utilização.

Além da integração de acessibilidade e usabilidade, as heurísticas de usabilidade também endereçam alguns aspectos relacionados à acessibilidade, por exemplo, visibilidade do status do sistema, compatibilidade do sistema com o mundo real, reconhecimento ao invés de relembração.

Tema	•Acessibilidade e Usabilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•Useit. Usability 101: Introduction to Usability e WAI
Resultados esperados	•Definição de conceitos

Os 10 maiores erros em webdesign

A seguir serão apresentados os 10 maiores erros em *webdesign* adaptados da lista de Nielsen (Nielsen, 2007):

1. **Ferramentas de busca pouco flexíveis** - Ferramentas de busca pouco flexíveis são as que não possibilitam que usuários acessem resultados relevantes porque variantes da palavra buscada são indexadas independentemente. Exemplos dessas variantes são: plural, palavras com hífen, pequenos erros de digitação, formas diferentes de escrever um mesmo termo (e.g., detectar e detetar), etc;
2. **Conteúdo em arquivos PDF** - Ao se deparar com arquivos PDF (*Portable Document Format*), os usuários enfrentam vários tipos de barreiras (e.g., comandos e atalhos diferentes, interface com novos elementos, tamanho de fonte), seja em um PDF embutido ou em uma nova janela. O fluxo de uso é afetado devido às diferenças existentes entre páginas *Web* e arquivos PDF. Adicionalmente, são elementos que focam mídias diferentes e contam com formas de navegação diferentes. Arquivos PDF são indicados para disponibilizar documentos, normalmente extensos, que dificilmente seriam lidos *on-line*;
3. **Links visitados sem destaque** - *Links* visitados com destaque auxiliam na indicação do que já foi visto, evitando que o usuário navegue novamente, sem querer, por páginas já visitadas. *Links* são a base para navegação na *Web*, ou seja, fornecer apoio para demarcar páginas visitadas auxilia a navegação como um todo. Por fim, o destaque dado aos *links* visitados deve ser consistente e deve ser diferente o bastante para que os usuários notem a diferença entre os estados dos *links*. Por exemplo, se você utiliza *links* na cor azul, seus *links* visitados podem utilizar um azul mais saturado ou roxo;
4. **Texto pouco legível** - A leitura nos monitores de computador é mais cansativa que no papel, portanto, o redator de um *website* deve se atentar para o fato de que a escrita para *Web* tem características próprias. O conteúdo textual deve ser sucinto e objetivo, além de deixar claro sua importância e a hierarquia dos elementos da página. Algumas formas de se obter essa hierarquia são:
 - Utilização de intertítulos;
 - Listas;
 - Palavras-chave em destaque;
 - Parágrafos curtos;
 - Utilizar a idéia da pirâmide invertida, passando a idéia principal do texto no início para só em seguida apresentar toda sua evolução, ou seja, de forma não linear;
 - Estilo de escrita simples, sem termos rebuscados ou relacionados a marketing;
5. **Tamanhos de fonte absolutos** - Ao utilizar tamanhos de fonte absolutos são impostas barreiras para o usuário que deseja aumentar ou reduzir o tamanho de fonte. De maneira geral deve-se projetar *websites* de forma a respeitar as preferências do usuário;
6. **Títulos de páginas com pouca indicação do conteúdo** - Busca é a maneira mais importante dos usuários descobrirem *websites*. Também é uma forma dos usuários encontrarem o que procuram dentro de um *website*. O título das páginas *Web* (i.e., o

conteúdo dentro das tags `<title>` e `</title>`) é a principal ferramenta para atrair novos visitantes a partir de resultados em ferramentas de busca. Este conteúdo é comumente utilizado para nomear *bookmarks*. Evite títulos demasiadamente longos e que utilizem informação pouco relevante para identificar a página (e.g., "Bem-vindo ao..." ou "O maior *website* de..."). Por fim, títulos das páginas também auxiliam na definição de contexto para usuários que estão utilizando várias janelas ao mesmo tempo;

7. **Qualquer coisa que pareça com publicidade** - Visão seletiva é muito poderosa e usuários de *websites* aprendem a prestar menos atenção aos anúncios publicitários que possam atrapalhá-los durante a navegação. A partir disto, usuários também ignoram elementos que pareçam com publicidade. Ou seja, é melhor evitar a utilização de qualquer elemento que se pareça com publicidade. Seguem algumas regras para se evitar este efeito:
 - Cegueira de *banner* - usuários dificilmente fixam o olhar em algo que pareça com um *banner*, seja pelo formato ou posicionamento na página;
 - Evitar animações - usuários tendem a ignorar áreas com textos em movimento ou animações não significativas;
 - Fechamento de *pop-ups* - indica que se deve evitar uso de *pop-ups*, pois alguns usuários fecham *pop-ups* antes mesmo deles estarem completamente renderizados;
8. **Violando convenções de design** - Consistência é um dos mais poderosos princípios de *design*: quando coisas sempre se comportam da mesma forma, usuários não se preocupam com o que acontecerá. Em vez disso, eles sabem o que irá acontecer com base em suas experiências anteriores. Quanto mais as expectativas dos usuários estiverem corretas, mais eles sentirão que comandam o sistema. Quanto mais o inverso ocorre, mais eles se sentirão inseguros. Segundo Nielsen, "*usuários passam a maior parte do tempo em outros sites*", ou seja, suas expectativas se basearão no que é comumente feito na maioria dos outros *websites*;
9. **Abrindo novas janelas do navegador** - *Links* que abrem em novas janelas são usados comumente com o intuito de evitar que um usuário vá para outro *website* e não retorne. No entanto, esta prática acaba poluindo a tela do usuário com várias janelas. Outro resultado ainda mais grave é que o botão "Voltar" da nova janela fica desabilitado. Assim, se um usuário não notar que uma nova janela foi aberta ele não conseguirá retornar para a página de origem;
10. **Falta de respostas às questões dos usuários** - Usuários da *Web* são altamente orientados à conclusão de tarefas. Eles visitam *websites* porque há algo que querem alcançar. A falha mais grave de um *website* é não fornecer informações que os usuários estão procurando. Às vezes a informação que o usuário procura não está na página ou está escondida em textos pouco legíveis, o que é tão crítico quanto não estar lá, uma vez que os usuários não têm tempo para ler todo conteúdo. Contraexemplos são: não fornecer o preço de produtos ou serviços, não fornecer taxa de juros, não fornecer preço total a prazo, etc.

Tema	•Usabilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•Useit. Usability 101: Introduction to Usability
Resultados esperados	•Técnica

Designer

Desenvolvedor

Redator

Inspeção Heurística de Usabilidade

Muitas das ferramentas e metodologias para avaliação de *websites* são consideradas inaplicáveis em pequenas equipes, seja por custo, complexidade ou falta de tempo. No entanto, a inspeção heurística é um método de baixo custo, tanto financeiro, pessoal e temporal e, portanto, é uma ferramenta importante no processo de criação de *websites*.

O que é a Inspeção Heurística? - A inspeção heurística consiste na compreensão de um pequeno conjunto de heurísticas que servirão para avaliar aplicações.

Como realizar uma inspeção heurística? - Para realizar uma inspeção, basta um pequeno número de avaliadores que deverão inicialmente verificar, de forma individual, a validade do conjunto de heurísticas previamente definido. Para tanto pode ser utilizado o Formulário de Inspeção Heurística. O resultado obtido por diferentes avaliadores é reunido em uma lista onde todas as observações são ordenadas de acordo com os respectivos níveis de severidade. Por fim, é possível que a equipe delibere quais ações serão executadas com maior prioridade.

Como acoplar isso no ciclo de desenvolvimento da equipe? - Como a duração de uma inspeção heurística é pequena, ela pode ser executada em diversas fases do desenvolvimento de aplicações, tais como, prototipação e testes. Além disso, as heurísticas tendem a se tornar princípios comuns aos avaliadores durante o processo de desenvolvimento e, portanto, com o passar do tempo e com a experiência eles agregam as heurísticas ao conjunto de premissas a serem seguidas naturalmente durante o *design*.

Tema	•Usabilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Formulário de Inspeção Heurística de Usabilidade na Web
Resultados esperados	•Avaliação e Técnica

Heurística genéricas

O conjunto de heurísticas genéricas foi proposto inicialmente por Nielsen, 1994. Aqui será utilizada a versão revisada das heurísticas contida em (Rocha e Baranauskas, 2003).

1. **Visibilidade do status do sistema** - Sistema precisa manter os usuários informados sobre o que está acontecendo, fornecendo um *feedback* adequado dentro de um tempo razoável;
2. **Compatibilidade do sistema com o mundo real** - Sistema precisa falar a linguagem do usuário, com palavras, frases e conceitos familiares ao usuário, ao invés de termos orientados ao sistema. Seguir convenções do mundo real, fazendo com que a informação apareça numa ordem natural e lógica;
3. **Controle do usuário e liberdade** - Usuários frequentemente escolhem por engano funções do sistema e precisam ter claras saídas de emergência para sair do estado indesejado sem ter que percorrer um extenso diálogo. Prover funções *undo* e *redo*;
4. **Consistência e padrões** - Usuários não precisam adivinhar que diferentes palavras, situações ou ações significam a mesma coisa. Seguir convenções de plataforma computacional;
5. **Prevenção de erros** - Melhor que uma boa mensagem de erro é um *design* cuidadoso o qual previne o erro antes dele acontecer;
6. **Reconhecimento ao invés de lembrança** - Tornar objetos, ações e opções visíveis. O usuário não deve ter que lembrar informação de uma para outra parte do diálogo. Instruções para uso do sistema devem estar visíveis e facilmente recuperáveis quando necessário;
7. **Flexibilidade e eficiência de uso** - Usuários novatos se tornam peritos com o uso. Prover aceleradores de forma a aumentar a velocidade da interação. Permitir a usuários experientes "cortar caminho" em ações frequentes;
8. **Estética e design minimalista** - Diálogos não devem conter informação irrelevante ou raramente necessária. Qualquer unidade de informação extra no diálogo irá competir com unidades relevantes de informação e diminuir sua visibilidade relativa;
9. **Ajudar os usuários a reconhecer, diagnosticar e corrigir erros** - Mensagens de erro devem ser expressas em linguagem clara (sem códigos) indicando precisamente o problema e construtivamente sugerindo uma solução;
10. **Help e documentação** - Embora seja melhor um sistema que possa ser usado sem documentação, é necessário prover *help* e documentação. Essas informações devem ser fáceis de encontrar, focalizadas na tarefa do usuário e não muito extensas.

Tema	•Usabilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Formulário de Inspeção Heurística de Usabilidade na Web
Resultados esperados	•Definição de conceitos

Designer

Desenvolvedor

Redator

Usabilidade na Web

As heurísticas genéricas podem ser especializadas para atender determinada classe de aplicações (e.g., sistemas de informações geográficas, controle aéreo, *websites*). Neste tópico serão apresentadas as principais recomendações de usabilidade elaboradas especificamente para *websites*, adaptado de (Rutter, 2004).

Tabela de heurísticas na Web - Visibilidade do status do sistema		
Heurística Genérica - 1. Visibilidade do status do sistema		
Heurística para Web	Ideal	Incompleto
Tempo de carga	As páginas normalmente carregam rapidamente (menos de 10 segundos) em uma conexão de 54kbps, devido a gráficos pequenos, boa compreensão de sons e gráficos e divisão apropriada do conteúdo.	Páginas normalmente demoram mais que 15 segundos para carregar devido a grandes gráficos, sons, etc.
Tabela de heurísticas na Web - Compatibilidade do sistema com o mundo real		
Heurística Genérica - 2. Compatibilidade do sistema com o mundo real		
Heurística para Web	Ideal	Incompleto
Conteúdo	O <i>website</i> tem propósito e tema bem definido que é mantido por todo o <i>website</i> .	O <i>website</i> não possui propósito e tema bem definido.
Precisão do Conteúdo	Toda a informação fornecida pelo designer no <i>website</i> é precisa e todos os requisitos da tarefa foram alcançados.	Há várias imprecisões no conteúdo fornecido pelo designer ou vários dos requisitos não foram alcançados.
Sons	Música, clips de áudio e sons são conscientemente editados e usados somente se eles auxiliam o leitor na compreensão do conteúdo ou tornam o <i>website</i> mais acessível para pessoas com deficiência visual.	Música, clips de áudio e sons são usados randomicamente ou atrapalham o <i>website</i> .
Imagens	Todas as imagens, especialmente as usadas para navegação, tem uma tag ALT que descreve a imagem e seu link. Então as pessoas com deficiência visual podem utilizar bem o <i>website</i> .	As necessidades de pessoas com deficiência visual são ignoradas.
Conhecimento do público-alvo	O designer conhece o público-alvo do <i>website</i> e desenvolve o material visando a atender da melhor forma às necessidades desse público.	O designer não leva em consideração o público-alvo do <i>website</i> .

Tabela de heurísticas na Web - Controle do usuário e liberdade		
Heurística Genérica - 3. Controle do usuário e liberdade		
Heurística para Web	Ideal	Incompleto
Controle sobre ações	O usuário explicitamente solicita a execução das ações e tem o controle de quando executá-las.	O <i>website</i> executa ações sem que estas tenham sido solicitadas pelo usuário ou o usuário não tem liberdade para escolher quando executar as ações.
Tabela de heurísticas na Web - Consistência e padrões		
Heurística Genérica - 4. Consistência e padrões		
Heurística para Web	Ideal	Incompleto
Links	Todos os links apontam para <i>website</i> de alta qualidade e atualizados.	Menos de 3/4 dos links apontam para <i>website</i> de alta qualidade e atualizados.
Escrita e gramática	Não há erros na escrita, pontuação ou gramática no <i>website</i> .	Há mais que 5 erros na escrita, pontuação ou gramática no <i>website</i> .
Navegação	Links para navegação são claramente nomeados, consistentemente localizados, permitem que o leitor se mova de uma página para páginas relacionadas e levam o usuário para onde ele espera ir. O usuário não se perde.	Alguns links não levam o usuário para os <i>websites</i> descritos. O usuário normalmente se sente perdido.
Tabela de heurísticas na Web - Prevenção de erros		
Heurística Genérica - 5. Prevenção de erros		
Heurística para Web	Ideal	Incompleto
Compatibilidade	O <i>website</i> foi testado em diferentes sistemas operacionais e diferentes versões de navegadores.	O <i>website</i> não foi testado ou somente suporta um navegador ou um sistema operacional.
Tabela de heurísticas na Web - Reconhecimento ao invés de relembração		
Heurística Genérica - 6. Reconhecimento ao invés de relembração		
Heurística para Web	Ideal	Incompleto
Identificação de elementos de interação	O usuário percebe quais são os elementos do <i>website</i> que ele pode interagir (e.g., links e botões)	O usuário precisa lembrar quais os elementos do <i>website</i> que ele pode interagir.

Tabela de heurísticas na Web - Flexibilidade e eficiência de uso		
Heurística Genérica -7. Flexibilidade e eficiência de uso		
Heurística para Web	Ideal	Incompleto
Modos de interação	O <i>website</i> permite que o usuário interaja de mais de um modo (e.g., passo a passo, atalhos), de forma a se adaptar da melhor forma às preferências do usuário.	O <i>website</i> oferece somente um modo de interação.
Tabela de heurísticas na Web - Estética e design minimalista		
Heurística Genérica - 8. Estética e design minimalista		
Heurística para Web	Ideal	Incompleto
Background	<i>Background</i> é muito atrativo, consistente entre as páginas, incrementa o tema ou propósito do <i>website</i> e não atrapalha a legibilidade.	<i>Background</i> atrapalha a legibilidade do <i>website</i> .
Cor	Cores de background, fontes, links formam uma paleta de cores agradável, não atrapalham o conteúdo e são consistentes entre as páginas do <i>website</i> .	Cores de <i>background</i> , fontes, links tornam o conteúdo difícil de compreender ou distrai o leitor.
Gráficos	Gráficos são relacionados ao tema/propósito do <i>website</i> , são conscientemente editados, de alta qualidade e aumentam o interesse e compreensão do leitor.	Gráficos são escolhidos randomicamente, de baixa qualidade ou distraem o usuário.
Layout	O <i>website</i> tem um layout atrativo e usável. É fácil de localizar todos os elementos importantes. Espaço em branco, elementos gráficos e alinhamento são utilizados para organizar o material.	As páginas possuem uma aparência desordenada ou confusa. É frequentemente difícil localizar elementos importantes.
Fontes	Fontes são consistentes, fáceis de ler e os tamanhos variam conforme os títulos e texto. O uso de estilos de fonte (e.g., itálico, negrito, sublinhado) é consistente e melhora a legibilidade.	Uma grande variedade de fontes, estilos e tamanhos de fontes são utilizados.

Tabela de heurísticas na Web - Ajudar os usuários a reconhecer, diagnosticar e corrigir erros

Heurística Genérica - 9. Ajudar os usuários a reconhecer, diagnosticar e corrigir erros

Heurística para Web	Ideal	Incompleto
Mensagens de erros compreensíveis	Como o comportamento de websites é bastante sensível a navegadores, sistemas operacionais e configurações do usuário. O website apresenta mensagens de erro que tratam as mensagens geradas externamente apresentando: descrição do erro, origem do erro e procedimento para solucioná-lo.	As mensagens de erro são apresentadas da mesma forma que são geradas. Isso torna sua compreensão restrita a especialistas da área.

Tabela de heurísticas na Web - Help e documentação

Heurística Genérica - 10. Help e documentação

Heurística para Web	Ideal	Incompleto
Meta tags	As meta <i>tags</i> utilizadas descrevem muito bem a compreensão do material incluído na página. Descrição, título, autor, palavras-chave, classificação, tipo de recurso, entre outros, refletem precisamente o conteúdo das páginas.	As meta <i>tags</i> estão faltando. O conteúdo da página não é refletido em nenhuma meta tag.
Copyright	Todo material utilizado de outras fontes é devidamente referenciado e citado.	Materiais de fontes externas não são propriamente documentados ou o material foi utilizado sem permissão da fonte.
Informações para Contato	Cada página contém informação de autoria, data de publicação/última data de edição.	Várias páginas não contém informação de autoria, data de publicação/última data de edição.

Tema	•Usabilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Formulário de Inspeção Heurística de Usabilidade na Web
Resultados esperados	•Definição de conceitos

Formulário de Inspeção Heurística na Web

Avaliador: _____

Descrição da URL: _____

URL: _____

Data: ____ / ____ / ____

Níveis de Severidade:

- 0 - não concordo que seja um problema de usabilidade
- 1 - problema cosmético - corrigir se houver tempo extra
- 2 - problema pequeno - baixa prioridade na correção
- 3 - problema grave - alta prioridade na correção
- 4 - problema catastrófico - correção obrigatória para entrega do produto

Formulário de Inspeção Heurística na Web		
Heurística Genérica		
Heurística para Web	Avaliação	Severidade
1. Visibilidade do status do sistema		
Tempo de Carga		
2. Compatibilidade do sistema com o mundo real		
Conteúdo		
Precisão do Conteúdo		
Sons		
Imagens		
Conhecimento do público-alvo		
3. Controle do usuário e liberdade		
Controle sobre ações		
4. Consistência e padrões		
Links		
Escrita e gramática		
Navegação		
5. Prevenção de erros		
Compatibilidade		
6. Reconhecimento ao invés de relembração		
Identificação de elementos de interação		
7. Flexibilidade e eficiência de uso		
Modos de interação		
8. Estética e design minimalista		
Background		
Cor		
Fontes		
Gráficos		
Layout		
9. Ajudar os usuários a reconhecer, diagnosticar e corrigir erros		
Mensagens de erros compreensíveis		
10. Help e documentação		
Meta tags		
Copyright		
Informações para Contato		

Designer

Desenvolvedor

Redator

Tema	• Usabilidade
Relevância para designers	• Média
Relevância para desenvolvedores	• Alta
Relevância para redatores	• Baixa
Ferramental utilizado	• Firefox Web Developer Add-on e Lynx
Resultados esperados	• Avaliação e Técnica

Acessibilidade no W3C

Uma das mais relevantes iniciativas para promover a acessibilidade na *Web* é o *WAI* (*WAI* - *Web Accessibility Initiative*), promovido pelo *World Wide Web Consortium* (*W3C*) e que visa o desenvolvimento de diretrizes e recursos que contribuem para tornar a *Web* acessível. O *WAI* concentra seus esforços em três focos, tal como descrito em (Chisholm e Henry, 2005):

- Navegadores *Web*, *players* multimídia e tecnologias assistivas que permitem uma experiência completamente usável e acessível. Oferece o conjunto de diretrizes *UAAG 1.0* (*UAAG* - *User Agent Accessibility Guidelines 1.0*) e o *UAAG 2.0* (*UAAG* - *User Agent Accessibility Guidelines 2.0*), ainda em estágio de desenvolvimento;
- Ferramentas de Autoria de conteúdos *Web* e ambientes de desenvolvimento que produzem conteúdo *Web* acessível e têm interfaces acessíveis. Oferece o conjunto de diretrizes *ATAG 1.0* (*ATAG* - *Authoring Tool Accessibility Guidelines 1.0*);
- Conteúdo *Web* concebido para ser acessível. Oferece o conjunto de diretrizes *WCAG 1.0* (*WCAG* - *Web Content Accessibility Guidelines 1.0*) e o *WCAG 2.0* (*WCAG* - *Web Content Accessibility Guidelines 2.0*), ainda em estágio de desenvolvimento.

O *WCAG* organiza suas diretrizes em torno de 4 princípios. São eles:

1. **Percebível** - Informação e componentes de interface de usuário devem ser apresentáveis aos usuários de forma de eles possam percebê-los. Isto significa que usuários devem ser capazes de perceber a informação que está sendo apresentada (ela não pode ser invisível a todos os sentidos dos usuários);
2. **Operável** - Componentes e navegação da interface de usuário devem ser operáveis. Isto significa que usuários devem ser capazes de operar a interface (a interface não pode requerer interação que o usuário não possa executar);
3. **Compreensível** - Informação e a operação da interface de usuário devem ser compreensíveis. Isto significa que usuários devem ser capazes de entender a informação assim como a operação da interface de usuário (o conteúdo ou operação não podem estar além da compreensão dos usuários);
4. **Robusto** - Conteúdo deve ser suficientemente robusto de forma que possa ser interpretado confiavelmente por uma vasta variedade de agentes de usuários, incluindo tecnologias assistivas. Isto significa que usuários devem ser capazes de acessar o conteúdo conforme as tecnologias evoluem (conforme tecnologias e agentes de usuário evoluem, o conteúdo deve permanecer acessível).

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•WAI
Resultados esperados	•Definição de conceitos e Técnica

Designer

Desenvolvedor

Redator

Modelo de Acessibilidade Brasileiro - e-MAG

Como mencionado no tópico “O que é Acessibilidade?”, por meio do *decreto* (DECRETO nº 5.296 DE 2 DE DEZEMBRO DE 2004, 2004), o Governo brasileiro estabelece a obrigatoriedade de oferecer acessibilidade em *websites* governamentais. Para apoiar a criação/manutenção da acessibilidade o Governo criou o *e-MAG* (e-MAG - Modelo de Acessibilidade de Governo Eletrônico), que estabelece diretrizes para este fim.

O Governo também oferece o (ASES - Avaliador e Simulador de Acessibilidade de Sítios) que permite a validação de *websites* segundo as diretrizes do e-MAG e a simulação de restrições, tais como, as visuais. Outra ferramenta que oferece validação segundo as diretrizes do e-MAG é o (DaSilva).

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•DaSilva e ASES
Resultados esperados	•Definição de conceitos e Técnica

Recomendações de Acessibilidade para: Utilização de novas tecnologias

Ao utilizar novas tecnologias deve-se projetar as páginas *Web* de forma que os usuários que não contam com esses novos recursos não sejam prejudicados e consigam utilizar e acessar o conteúdo e os serviços normalmente. A principal maneira para garantir a utilização de seu *website* em diversos dispositivos é o atendimento aos padrões e recomendações de tecnologias *Web*. Nesse cenário, a W3C é a principal organização para a definição de padrões.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	•Avaliação, Compatibilidade de código e Técnica

Recomendações de Acessibilidade para: Portabilidade

Quando se pensa em tornar uma navegação acessível deve-se ter em mente que todos os componentes gráficos e o conteúdo textual serão disponibilizados em outros tipos de mídia (e.g., áudio, terminais de texto). Assim, toda a navegação estruturada a partir de elementos gráficos e da organização dos componentes de uma página *Web* deve possibilitar que o percurso dos usuários utilizando outros tipos de mídia ou dispositivos seja igualmente eficiente. Note que um dos maiores desafios está em possibilitar que o percurso possível na tela do computador, que é bidimensional, seja feito em mídias que contam com apenas uma dimensão (e.g., áudio).

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish, NVDA, JAWS, WebAnywhere, DOSVOX, Lynx e WAI
Resultados esperados	•Navegação, Compatibilidade de código e Técnica

Bloqueio da navegação por teclado

Usuários devem poder navegar por todos os elementos do *website* usando somente o teclado. Além disso, a navegação deve ser organizada de maneira que faça sentido em relação aos blocos de informação. Algumas técnicas simples podem aumentar significativamente a eficiência da navegação por teclado, por exemplo:

- *Skip links*;
- *Breadcrumbs*;
- Atalhos.

Tema	•HTML, JavaScript e Acessibilidade
Relevância para designers	•Média
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Média
Ferramental utilizado	•WAI
Resultados esperados	•Navegação

Recomendações de Acessibilidade para: Skip Links

Boa parte dos *layouts* de *websites* possuem um cabeçalho, ocupando a parte superior da página, um menu, ocupando a lateral esquerda, e o conteúdo, localizado no centro da página. Visualmente, esse formato é adequado, pois permite que o conteúdo principal da página fique na região central. No entanto, para usuários que utilizam leitores de telas ou navegam via teclado, cria-se uma barreira. Os leitores de telas serializam o conteúdo do topo para baixo e da esquerda para a direita. Dessa forma, seguindo o *layout* mais comum, o conteúdo principal somente seria acessado após a leitura de todo o cabeçalho e dos itens do menu.

Para tentar solucionar este problema são utilizados os *skip links*, que são *links* para regiões da página (e.g., um *skip link* "Ir para o conteúdo", no cabeçalho da página). Assim, usuários de leitores de telas ou que naveguem via teclado podem "pular" trechos da página que não lhe interessam, tal como é feito pelos demais usuários. *Skip links* podem ser feitos de diversas maneiras. Uma forma comum é usar um *link* visível no início da região que pode ser "pulada".

Exemplo de *skip link*:

```
...
<body>
<a href="#conteudo">Ir para o conteúdo</a>
...
<h1><a name="conteudo" id="conteudo">Título principal</a></h1>
<p>Texto do conteúdo</p>
...
```

Para evitar que o *skip link* afete a estética da página, muitos *designers* criam *skip links* invisíveis (isso pode ser feito utilizando CSS). Esta solução resolve o problema para usuários de leitores de telas. Entretanto, ela não atende usuários que navegam na *Web* via teclado e que não usam leitores de tela.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	•Navegação e Técnica

Recomendações de Acessibilidade para: Breadcrumbs

Breadcrumbs são um texto de uma única linha para mostrar a localização de uma página Web na hierarquia do website (Nielsen, 2007)

Nielsen aponta razões para utilizar *breadcrumbs*:

- *Breadcrumbs* mostram às pessoas sua localização corrente relativa aos conceitos de nível mais alto, auxiliando-os a entender onde estão em relação ao resto do *website*;
- *Breadcrumbs* oferecem acesso em um clique para níveis mais altos do *website* e, portanto, resgatam usuários que "caem de para-quedas" em destinos muito específicos, mas inapropriados através de pesquisas ou *links* profundos;
- *Breadcrumbs* nunca causam problemas em testes de usuários: pessoas podem ignorar este pequeno elemento de *design*, mas elas nunca interpretam incorretamente o caminho do *breadcrumbs* ou têm problemas operando-os;
- *Breadcrumbs* ocupam pouco espaço na página.

A implementação dos *breadcrumbs* é geralmente feita da mesma forma, em uma linha horizontal que (Nielsen, 2007):

- parte do nível mais alto para o mais baixo, um por vez;
- começa com a *homepage* e termina com a página atual;
- possui um *link* textual simples para cada nível (exceto para a página corrente);
- possui um separador de um caractere entre os níveis (geralmente ">").

Além do sinal de maior (>) existem outras opções como a barra (/), dois pontos (:) e imagens de setas. Ainda não existe um padrão para o separador, mas o recomendado é a utilização do sinal de maior (Nielsen, 1999).

Exemplo de *breadcrumb*:

```
...  
<a href="/acessibilidade/">Acessibilidade</a> >  
<a href="/acessibilidade/web/">Acessibilidade na Web</a> >  
<a href="/acessibilidade/web/navegação/">Navegação</a> > Breadcrumbs  
...
```

Código renderizado

```
...  
Acessibilidade > Acessibilidade na Web > Navegação > Breadcrumbs  
...
```

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	•Navegação e Técnica

Designer

Desenvolvedor

Redator

Recomendações de Acessibilidade para: Teclas de Atalho

Outra questão que surge com a serialização de conteúdo é: como chegar de maneira eficiente (direta) até *links* visitados freqüentemente, sem ter que navegar por outras partes da página? Quando usuários estão familiarizados com certa página *Web*, eles já sabem quais *links* devem visitar para obter os serviços que usa com freqüência. Assim, uma forma de possibilitar esse acesso direto é com a utilização de teclas de atalho. A implementação deste recurso é feita pela utilização do atributo *accesskey*, que deve conter um caractere que diz respeito à tecla de acesso desejada. Para utilizá-la, basta que o usuário combine a tecla de acesso (referente ao atributo *accesskey*) com a tecla especial utilizada pelo seu navegador (e.g., ALT).

Exemplo de declaração de teclas de atalho utilizando o atributo *accesskey*:

```
...  
<a href="#conteudo" accesskey="c">Ir para o conteúdo</a>  
...  
<label for="ex808a_busca" accesskey="b">Buscar</label>  
<input type="text" name="busca" id="ex808a_busca" value="">  
...
```

Apesar de ser um recurso interessante no que diz respeito à acessibilidade, uma vez que é uma solução para pessoas que utilizam leitores de telas ou que só podem utilizar o teclado, e à usabilidade, pois usuários experientes podem utilizar atalhos para acessar o que desejam de maneira mais eficiente, as teclas de atalho sofrem com problemas de compatibilidade e padronização devido à variedade de como este recurso é implementado em diferentes navegadores. A seguir é apresentado como utilizar as teclas de atalho em diferentes navegadores:

- Internet Explorer 5+ (Windows): Pressionar ALT + tecla de acesso, e depois ENTER;
- Firefox, Mozilla e Netscape 7+ (Windows): Pressionar ALT + tecla de acesso;
- Firefox, Mozilla e Netscape 7+ (Mac OS X): Pressionar CTRL + tecla de acesso;
- Safari e Omniweb (Mac OS X): Pressionar CTRL + tecla de acesso;
- Konqueror (Linux): Pressionar e soltar CTRL, e depois pressionar a tecla de acesso;
- Internet Explorer 4 (Windows): Pressionar ALT + tecla de acesso, e depois ENTER;
- Internet Explorer 5+ (Mac): Pressionar CTRL + tecla de acesso.

Um dos motivos desta solução ser pouco utilizada é que os usuários não sabem quais são as teclas de acesso utilizadas na página até as descobrirem (Robertson, 2003). Uma solução interessante para este problema apresentada por Robertson (2003) faz uso de indicações sublinhadas nos elementos que contam com teclas de atalho, forma semelhante à utilizada em alguns sistemas operacionais. (Para mais detalhes sobre a utilização de outros recursos de CSS combinados com teclas de acesso veja Utilização de CSS com teclas de acesso).

No CSS:

Exemplo de CSS para indicação visual de teclas de atalho:

```
...  
span.shortcut { text-decoration: underline; font-weight: bold; }  
...
```

No HTML:

Exemplo de uso de classe CSS para indicação visual de teclas de atalho no HTML:

```
...  
<a href="#conteudo" accesskey="c">Ir para o <span  
class="shortcut">c</span>onteúdo</a>  
...  
<label for="ex808b_busca" accesskey="b">  
<span class="shortcut">B</span>usca:</label>  
<input type="text" name="busca" id="ex808b_busca" value="">  
...
```

Tema	•CSS e Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	•Navegação e Técnica

Recomendações de Acessibilidade para: Legibilidade

Segundo Nielsen, *"todo o resto - design, velocidade, conteúdo - falha quando os usuários não conseguem ler o texto"* (Nielsen, 2000). Páginas Web bem projetadas e com redação adequada para a mídia, obtém boa legibilidade de forma natural. No entanto, Nielsen (2000) apresenta algumas regras básicas para que *websites* obtenham boa legibilidade, são elas:

- Utilizar cores com alto contraste entre texto e fundo;
- Utilizar fundos de cores lisas ou padrões de fundo extremamente sutis;
- Utilizar tamanho padrão de texto suficientemente grande para que as pessoas possam ler o texto facilmente, mesmo que não tenham visão perfeita;
- Utilizar texto imóvel. Mover ou piscar dificulta a leitura;
- Evitar o uso de maiúsculas para texto, pois é mais difícil para o olho reconhecer a forma das palavras e os caracteres na aparência uniforme causada pelo texto em maiúsculas.

Quando um *website* conta com uma estrutura hierárquica bem definida (incluindo código válido), boa legibilidade, escrita voltada para Web e uso adequado dos atributos aurais de CSS, é possível fazer com que seu entendimento, via áudio, seja equiparado ao obtido com elementos visuais. Note que esta qualidade não beneficia apenas usuários com deficiência visual, pois, por exemplo, com a popularização de conexão com a Internet em automóveis, será possível que um motorista escute o conteúdo de um *website* enquanto dirige seu carro e está com a visão ocupada.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++ e Bluefish
Resultados esperados	•Legibilidade e Técnica

Escrita de textos para a Web

Um dos principais erros cometidos ao criar conteúdos para *websites* é o de ignorar o conhecimento sobre o domínio do *website* e o nível de letramento dos usuários. O WCAG, no critério de sucesso 3.1.5 "*Reading Level*", define que textos não devem requerer habilidade de leitura mais avançada que o nível inferior secundário de educação ou deve fornecer uma apresentação alternativa da informação.

Para entender a relação do que seria o "nível inferior secundário de educação" no contexto brasileiro, podemos consultar a definição fornecida pela UNESCO: "... o período de dois ou três anos de educação que começa depois da conclusão de seis anos de escola e termina nove anos depois do início da educação primária". Apesar de internacionalmente aceito, ela pode não refletir a realidade em países em desenvolvimento (e.g., Índia e Brasil).

Se considerarmos a Pesquisa Nacional por Amostragem de Domicílios realizada em 2007 pelo IBGE, a média de escolaridade da população brasileira entre 15 e 59 anos é de 7,8 anos. Esta informação inicialmente indica uma adequação com os dados da UNESCO. No entanto, quando consideramos a população de 60 anos ou mais, a média cai para somente 3,8 anos.

É necessário considerarmos também a qualidade dos anos de escolaridade. Para isso vamos dar uma olhada no INAF, que investiga a habilidade da população para ler e compreender textos e representações gráficas. Em contraste com os dados providos pelo IBGE, o INAF de 2009, indica que quase 30% da população brasileira entre 15 e 59 anos eram funcionalmente iletrados, ou seja, essas pessoas não eram capazes de realizar tarefas simples envolvendo leitura de palavras e frases, ainda que algumas delas sejam capazes de ler números, como telefones e preços, ou encontrar informações específicas em textos curtos.

Ainda, observamos que nível de letramento varia significativamente de acordo com a faixa etária. Atualmente, o letramento é mais baixo entre os idosos. Dessa maneira, a escolha de texto e termos utilizados deve ser realizada de maneira cuidadosa para ser adequada ao público esperado.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Baixa
Relevância para redatores	•Alta
Ferramental utilizado	•WAI
Resultados esperados	•Legibilidade

Recomendações de Acessibilidade para: Contexto

É um desafio manter o usuário sempre informado sobre onde ele está e, consequentemente, seu contexto acerca da página *Web* que está utilizando. Vários elementos visuais informam ao usuário seu contexto (e.g., o *breadcrumb*, hierarquia de títulos e subtítulos, organização em abas).

No entanto, como auxiliar usuários que utilizam ampliadores ou leitores de telas, por exemplo, a serem informados de seu contexto na página *Web*?

Websites bem estruturados são meios de auxiliar todos os usuários neste aspecto. Mas, ao elaborar uma nova página *Web*, deve-se sempre ter em mente que sua utilização pode ocorrer em diferentes dispositivos e mediados por diferentes tipos de tecnologia assistivas. Segundo Nielsen (Nielsen, 2000), a navegação precisa ajudar os usuários a responder as três questões fundamentais da navegação: **Onde estou? Onde estive? Aonde posso ir?**

Exemplos da importância da definição de contexto:

- Quando um usuário acessa uma página *Web* enquanto executa outras tarefas. Ao interromper a navegação nesse *website* por vários minutos e retornar, deve ser relativamente fácil para que ele responda às três questões fundamentais da navegação;
- Quando um usuário que utiliza ampliador de telas acessa uma determinada área de uma lista de itens (com vários elementos e diferentes níveis aninhados), deve haver indicações (e.g., indentação, numeração dos itens), para que o usuário seja informado do contexto da área visualizada;
- Quando um usuário que utiliza um leitor de telas acessa diretamente, via teclas de atalho, um trecho do conteúdo. Neste caso, a página deve conter indicações de qual é o contexto do trecho acessado.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	•Navegação, Avaliação e Técnica

Recomendações de Acessibilidade para: Agrupamento Espacial

A tolerância de usuários de páginas *Web* é baixa, dada a variedade de opções existentes. Portanto, um *website* deve disponibilizar de forma clara as suas funcionalidades sem que para isso, o usuário tenha que ler todo o conteúdo da página. Uma opção bastante simples e eficaz é o agrupamento espacial de itens que possuem alguma forma de relacionamento.

Agrupar itens que possuem alguma afinidade é uma atitude natural do *designer*. Porém, a escolha do agrupamento e dos itens que o compõe, por vezes, não fica clara para os usuários do *website*. Para auxiliar a escolha do agrupamento vamos analisar as Leis de Gestalt (Gestalt Laws) relacionadas a aspectos visuais, que são: proximidade (i.e., objetos próximos formam um grupo), fecho (i.e., uma linha fechada, ou quase, forma um padrão), similaridade (i.e., similaridade de propriedades como cor, forma, textura, formam um grupo), continuidade (i.e., conjunto de objetos formam uma entidade), *common fate* (i.e., objetos movendo-se na mesma direção pertencem ao mesmo grupo. Movimentos singulares na mesma direção são vistos como movimento do todo) e *connectedness* (i.e., objetos conectados entre si, "tocando", pertencem ao mesmo grupo).

Cada uma das Leis de Gestalt pode ser facilmente traduzida para elementos de uma página HTML. Exemplos seriam:

- a lei de proximidade pode ser atendida com um elemento *fieldset*;
- por meio de CSS, poderia ser atribuída cor de fundo e contorno para identificar elementos de menu.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Média
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++ e Bluefish
Resultados esperados	•Avaliação, Definição de conceitos e Técnica

Recomendações de Acessibilidade para: Ícones

Pessoas não letradas ou com baixo letramento têm dificuldade para interpretar textos longos. Portanto, a utilização de ícones para representar textos é uma opção bastante eficaz. Porém, a seleção do ícone a ser utilizado não é uma tarefa trivial e exige mais do que simples bom senso.

Sempre que for possível, deve-se utilizar ícones que representem o objeto no mundo real (e.g., uma imagem de impressora para representar a função imprimir). No entanto, muitas das funcionalidades em uma página *Web* não dispõem de uma referência direta no mundo real (e.g., *reload* de uma página). Nesse caso, a primeira medida é a de verificar se existe alguma convenção para representar a funcionalidade e, somente em caso contrário deve-se optar por uma representação arbitrária.

Signos arbitrários devem ser utilizados com moderação, dado que não possuem referência para a funcionalidade que representam. Quando o seu uso for necessário, deve-se verificar se uma representação simples pode substituí-lo e realizar testes com usuários potenciais, considerando a facilidade de os usuários lembrarem a função do signo em utilizações futuras.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Design e Avaliação de Interfaces Humano-Computador
Resultados esperados	•Imagens e animações, Definição de conceitos e Técnica

Recomendações de Acessibilidade para: Redundância

Como nem todos os usuários podem enxergar as cores do *website*, seja por deficiência ou pela qualidade do monitor, é necessário que cuidados especiais sejam tomados na seleção da paleta de cores de um *website*. A primeira medida a ser tomada é a de que toda informação oferecida por cores também esteja disponível sem cores.

Contraexemplo de informação fornecida somente por cores:

```
<table width="50%" cellpadding="5" cellspacing="0" border="1" style="font-weight: bold;">
<tr style="background-color: #cccccc;">
<td>Com cores</td>
<td>Sem cores</td>
</tr>
<tr><td style="color: #0000ff">10</td><td>10</td></tr>
<tr><td style="color: #ff0000">5</td><td>5</td></tr>
<tr><td style="color: #0000ff">50</td><td>50</td></tr>
</table>
```

Código renderizado

Com cores	Sem cores
10	10
5	5
50	50

Exemplo de informação fornecida por cores e texto:

```
<table width="50%" cellpadding="5" cellspacing="0" border="1" style="font-weight: bold;">
<tr style="background-color: #cccccc;">
<td>Com cores</td>
<td>Sem cores</td>
</tr>
<tr><td style="color: #0000ff">10</td><td>10</td></tr>
<tr><td style="color: #ff0000">-5</td><td>-5</td></tr>
<tr><td style="color: #0000ff">50</td><td>50</td></tr>
</table>
```

Código renderizado

Com cores	Sem cores
10	10
-5	-5
50	50

Para auxiliar a escolha de uma combinação de cores que não crie barreiras para pessoas com deficiência existem várias ferramentas de apoio. Duas das ferramentas interessantes são o *Color Laboratory* (Color Laboratory), que permite a visualização da paleta de cores conforme o tipo de deficiência no reconhecimento de cores, monitor e sistema operacional do usuário; e o *Colorblind Web Page Filter* (Colorblind Web Page Filter), que permite que o usuário informe uma URL qualquer e a visualize utilizando as cores que um usuário com determinado tipo de deficiência enxergaria.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	•Imagens e animações, Definição de conceitos e Técnica

Recomendações de Acessibilidade para: Recursos de Áudio

A redundância nas formas de apresentação é uma palavra-chave quando se pensa em acessibilidade. Além da estruturação das diversas mídias, existem formas de utilizar recursos de áudio para aumentar a acessibilidade e usabilidade de *websites*. O áudio oferece um canal de saída complementar ao visual e pode ser utilizado para oferecer ajuda aos usuários (Nielsen, 2000).

Usuários com pouca experiência com computadores tendem a considerar *websites* que possuem auxílio via voz mais fáceis de usar, pois basta seguir as instruções e completar a tarefa em questão, o que auxilia a transpor as barreiras relacionadas à baixa familiaridade dos usuários com sistemas computacionais.

Além de recursos de áudio que utilizam voz, existem efeitos sonoros não-vocais que auxiliam os usuários a identificarem e memorizarem eventos ocorridos no sistema e serão comentados a seguir:

- **Earcons** - são padrões de som utilizados para representar um item ou evento. Geralmente são sons abstratos que não estão relacionados diretamente com o item ou evento que representam no mundo real, portanto os usuários devem aprendê-los;
- **Ícones auditivos** - também são sons utilizados para representar um item ou evento. No entanto, possuem relacionamento com o que representam, podendo não ser necessário o aprendizado.

Exemplos de utilização desse tipo de recurso por parte dos navegadores ocorrem quando é disparada a função *alert* de JavaScript ou quando um *download* é concluído. Note que sempre deve haver meios para que o usuário possa desativar estes recursos.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Alta
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++ e Bluefish
Resultados esperados	• Audibilidade, Definição de conceitos e Técnica

Recomendações de Acessibilidade para: Controle do Usuário

É possível adicionar diversas funcionalidades a uma página *Web* como incluir áudio, animações, vídeo com um intérprete de LIBRAS, entre outros. No entanto, ao adicionar um recurso é possível que ele interfira de alguma maneira na eficiência do *website* para um determinado grupo de usuários. Assim sendo, deve-se sempre disponibilizar um mecanismo para que usuários desabilitem esse tipo de recursos, evitando que *designers* tomem decisões no lugar dos usuários.

Um exemplo de recurso que necessita deste controle é o *refresh* automático. Este recurso é utilizado na tentativa de fazer com que os usuários visualizem sempre o conteúdo mais recente. No entanto, imagine um usuário que utiliza somente o teclado ou leitor de telas e acaba de conseguir chegar a um *link* de seu interesse, no meio da página. E, pouco antes de acessar o *link* que descobriu, a página é recarregada automaticamente e remete o usuário para o início da página.

Mas, onde está o problema? A resposta é que o usuário não possui controle sobre o recurso, neste caso, o *refresh* automático. Note que para resolver este caso bastaria fornecer uma maneira para que o usuário habilitasse e desabilitasse o *refresh* automático, de acordo com sua vontade.

Outros exemplos em que isso ocorre são as mídias temporizadas, como vídeos, áudios e animações. Para esse tipo de mídia é interessante que usuários possam controlar o fluxo, por meio de funcionalidades como: tocar, pausar e parar. Ainda, podem existir certos componentes que requerem controle de texto, como duração de sessões, tempo para preenchimento de um formulário, etc. Para esses casos, usuários devem poder: estender ou desligar a opção (exceto em casos onde isso possa comprometer a segurança do usuário ou do *website*).

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, Notepad++, Bluefish e WAI
Resultados esperados	•Avaliação e Técnica

Técnicas para encontrar conteúdo em um website

A quantidade de informações disponíveis nos *websites* cresce rapidamente. No entanto, nem sempre o *website* fornece recursos para que os usuários encontrem a informação desejada de maneira eficiente. Como resultado, é comum utilizarmos ferramentas de busca como Google e Bing mesmo quando sabemos em qual *website* está a informação.

Para facilitar a busca por informações, *websites* devem oferecer diversas maneiras para que elas sejam executadas. Exemplos são:

- Menus. Apresentam o conteúdo de acordo com uma categorização dos itens;
- Listas de conteúdo. É interessante prover diversas maneiras de ordenar o conteúdo;
- Mapa do *website*. Geralmente apresentado como uma lista hierárquica de itens;
- Busca. Usuários da Web nem sempre estão dispostos a entender a estrutura de *websites* e de seus menus, assim a busca tem o potencial de retornar o conteúdo desejado em somente um passo. Mecanismos de busca robustos não são de fácil implementação. No entanto, existem provedores de serviço de busca que permitem indexar seu conteúdo e obter melhores resultados do que em uma implementação local e.g. Google Custom Search Engine.

Tema	•HTML e Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•WAI
Resultados esperados	•Navegação

Avaliação Simplificada de Acessibilidade

Da mesma forma que é realizada a Inspeção de Usabilidade, a Avaliação Simplificada de Acessibilidade consiste de um conjunto básico de heurísticas que devem ser testadas nas páginas do *website*. A Avaliação Simplificada de Acessibilidade é dividida em duas seções. Na primeira, a avaliação é realizada de maneira manual, utilizando o Formulário de Avaliação Simplificada de Acessibilidade, assim o avaliador deve validar cada heurística com base na observação e interação com o *website*. A avaliação manual deve ser realizada utilizando navegadores gráficos (e.g., *Internet Explorer*, *Mozilla Firefox*) e navegadores textuais (e.g., *Lynx*). Já na segunda seção, é utilizada uma ferramenta semiautomática de avaliação, que deverá retornar todas as possíveis falhas de acessibilidade do *website*.

Em 2006, Christian Heilmann (Heilmann, 2006a)(Heilmann, 2006b) publicou um artigo intitulado "*Seven Accessibility Mistakes*" que reflete, segundo a percepção do autor, os erros mais comuns em acessibilidade encontrados na *Web*.

Erro 1: Confiar em produtos sem testá-los

Várias ferramentas de edição e *frameworks* promovem a criação de *websites* padronizados. Muitas dessas ferramentas simplesmente fecham *tags*, convertem elementos em letra minúscula e reforçam a necessidade do atributo *alt* em imagens. Apesar de não ser errado, esses recursos não são suficientes.

Erro 2: Assumir muita responsabilidade

Um *website* não é uma aplicação estática. A maioria dos *websites* está em constante crescimento e conta com a colaboração de várias pessoas com diferentes níveis de conhecimento. Portanto, assumir a responsabilidade total sobre a acessibilidade do *website* pode não ser uma boa opção. Conscientizar os demais mantenedores dos principais problemas de acessibilidade é uma opção barata e, além disso, distribui a carga de trabalho para garantir a acessibilidade do *website*.

Erro 3: Planejar somente o pior cenário

Designers conscientes das questões de acessibilidade tendem a desenvolver soluções fortemente focadas em usuários com algum tipo de deficiência. No entanto, *websites* são feitos, em sua maioria, para atender um grande espectro de usuários e, portanto, eles devem ser eficientes e agradáveis para todos.

Erro 4: Compartilhar problemas com o visitante

Questões como segurança, privacidade, *spam* e roubo de dados não devem afetar a interação do visitante com o *website*. Muitos *designers* implementam maneiras de inibir esses problemas usando *scripts* JavaScript e CSS (e.g., desabilitar o botão direito do mouse). No entanto, tais recursos somente afetam usuários com baixo conhecimento em computação. Aqueles que realmente pretendem causar danos ao *website* irão, cedo ou

tarde, burlar tais *scripts*. Portanto, a garantia de segurança do *website* deve ser implementada de maneira a não interferir na interação do usuário com o *website*.

Erro 5: Tentar resolver problemas fora da sua área de conhecimento

Algumas funções de acessibilidade (e.g., redimensionamento de fonte) são implementadas por meio de *scripts client-side*. Essa forma de implementação, além de ser fortemente dependente da configuração local do usuário muitas vezes é feita de forma deficiente. Dessa maneira, recomenda-se a utilização moderada desses recursos, aproveitando os que já são disponibilizados pelo navegador ou pelo sistema operacional.

Erro 6: Esconder ou sobrescrever melhorias de acessibilidade/usabilidade

A maioria dos mecanismos utilizados para melhorar a acessibilidade de um *website* não gera um grande impacto visual (e.g., atributo *alt*, *skip links*). No entanto, alguns *designers* escondem estes mecanismos (e.g., *skip links* invisíveis). Elementos de acessibilidade não são feitos exclusivamente para usuários de leitores de telas. Portanto, eles devem ficar acessíveis a todos que possam desejar utilizá-los.

Erro 7: Oferecer suporte ao seu cliente e não aos clientes do seu cliente

Diferente do que acontecia no início da popularização da Internet, atualmente clientes de *websites* possuem preferências e dão opiniões sobre a construção do *website*. Entretanto, ainda é bastante difícil mostrar ao cliente as vantagens de um *website* com *design* minimalista, acessível e justificar gastos com a aplicação de métodos de *design*, avaliação e testes de compatibilidade. Clientes ainda supervalorizam uma apresentação mais sofisticada, mesmo que ela cause limitações a alguns usuários. Apesar de que o *designer* deve atender às solicitações do cliente, ele deve se colocar na posição de especialista na área e estabelecer alguns limites e requisitos que devem ser atendidos na construção do *website*.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Formulário para Avaliação Simplificada de Acessibilidade
Resultados esperados	•Avaliação

Formulário para Avaliação Simplificada de Acessibilidade

(baseado do curso MO622 - Fatores Humanos em Sistemas Computacionais,
Prof.a M. Cecília C. Baranauskas, Segundo Semestre de 2007. Instituto de Computação - UNICAMP)

Avaliador: _____

Descrição da URL: _____

URL: _____

Data: _____ / _____ / _____

Níveis de Severidade:

- 0 - não concordo que seja um problema de usabilidade
- 1 - problema cosmético - corrigir se houver tempo extra
- 2 - problema pequeno - baixa prioridade na correção
- 3 - problema grave - alta prioridade na correção
- 4 - problema catastrófico - correção obrigatória para entrega do produto

Formulário de Avaliação Simplificada de Acessibilidade de Interface - Avaliação Manual: Uso de Navegadores Gráficos e Textuais		
Heurística	Problemas e exemplos	Severidade
a) Ao utilizar um navegador gráfico (e.g., Internet Explorer, Firefox)		
1. Desabilitar imagens - Verificar se são disponibilizados textos alternativos apropriados (mais informações em Textos alternativos) <ul style="list-style-type: none">• No Chrome 13: usar o complemento <i>web developer</i> ou ir em Opções > Configurações avançadas > Configurações de conteúdo > Imagens > Não mostrar nenhuma imagem.• No Firefox 6: usar o complemento <i>web developer</i> ou ir em Opções > Conteúdo > Carregar imagens automaticamente.• No Internet Explorer 9: pressionar F12 para mostrar o painel <i>Developer Tools</i> > Imagens > Desabilitar Imagens ou ir em Opções da Internet > Avançado > Multimídia > Mostrar imagens		
2. Desabilitar som - Verificar se o conteúdo sonoro continua disponível por meio de equivalentes textuais		

Designer

Desenvolvedor

Redator

<p>3. Variar o tamanho da fonte (usando controles do navegador) - Verificar se o tamanho da fonte varia na tela de forma adequada e se a página continua usável com grandes tamanhos de fonte</p> <ul style="list-style-type: none"> • Tecla Control (CTRL) e usar a rolagem do mouse ou do <i>touchpad</i> ou; • Tecla Control (CTRL) e pressionar as teclas "+" ou "-", para aumentar ou reduzir o tamanho da fonte; • Tecla Control (CTRL) combinada com a tecla 0 (zero) volta ao tamanho original da fonte. 		
<p>4. Variar resoluções de tela</p> <ul style="list-style-type: none"> • Usando o complemento <i>web developer</i> no Firefox 6 ou no Chrome 13: ir em redimensionar > escolher uma das resoluções disponíveis ou definir uma outra; • No Internet Explorer 9: pressionar F12 para mostrar o painel <i>Developer Tools</i> > ir em Ferramentas > Redimensionar > escolher uma das resoluções disponíveis ou definir uma outra. 		
<p>5. Redimensionar a janela da aplicação para tamanhos menores que o máximo - Verificar se a rolagem horizontal não é exigida</p>		
<p>6. Verificar se o contraste é adequado</p> <ul style="list-style-type: none"> • Imprimir a página em escala de cinza (ou em preto e branco) • Se você preferir uma abordagem mais ecológica e prática, uma sugestão é o Colorblind Filter 		

<p>7. Acessar links e formulários da página por meio da tecla TAB, sem auxílio do mouse - Verificar se todos os links são acessíveis e estão bem descritos, bem como se os controles dos formulários são acessíveis. Mais informações:</p> <ul style="list-style-type: none"> • Bloqueio da navegação por teclado; • Técnicas para encontrar conteúdo em um website. 		
b) Ao utilizar um navegador textual (e.g., Lynx)		
<p>1. Verificar se as informações disponibilizadas são equivalentes àquelas oferecidas pelo navegador gráfico</p>		
<p>2. Verificar se a informação apresentada faz sentido se apresentada de forma linear</p>		

Formulário de Avaliação Simplificada de Acessibilidade de Interface - Avaliação semiautomática (utilizar uma das ferramentas semiautomáticas de avaliação de acessibilidade)		
Heurística/Diretriz	Problemas e exemplos	Severidade
Comentários dos resultados		

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, DaSilva, Magenta, ATRC Accessibility Checker, Lynx e WAI
Resultados esperados	•Avaliação e Técnica

Designer

Desenvolvedor

Redator

Avaliação de acessibilidade utilizando ferramentas semiautomáticas

Ferramentas semiautomáticas de avaliação de acessibilidade na *Web* geralmente são aplicações que, a partir de uma URL dada, validam um conjunto de diretrizes e oferecem *feedback* sobre o problema encontrado e qual a solução para ele. Essas aplicações normalmente são disponibilizadas em *websites* e não necessitam a instalação de nenhum *software* localmente. Algumas das principais ferramentas disponíveis atualmente são:

- DaSilva que utiliza o conjunto de diretrizes do W3C WCAG;
- Magenta que utiliza o conjunto de diretrizes do Stanca Act;
- ATRC Accessibility Checker que utiliza o conjunto de diretrizes do EARL;
- WAVE;
- Avaliador de Acessibilidade Funcional FAE.

Além dos exemplos acima, é possível encontrar uma [lista extensa de ferramentas no website da W3C \(https://www.w3.org/WAI/ER/tools/\)](https://www.w3.org/WAI/ER/tools/).

Ferramentas de avaliação semiautomática é um recurso poderoso para tornar *websites* acessíveis. No entanto, elas não são suficientes para garantir que um *website* seja acessível. Para tanto, existem diretrizes e recomendações que não se restringem à codificação válida. Em “Avaliação Simplificada de Acessibilidade”, é apresentado um método de avaliação manual que pode ser bastante útil para identificar os problemas de acessibilidade que não se restringem às diretrizes operacionais.

Tema	•Acessibilidade
Relevância para designers	•Alta
Relevância para desenvolvedores	•Alta
Relevância para redatores	•Baixa
Ferramental utilizado	•Firefox Accessibility Extension, DaSilva, Magenta, ATRC Accessibility Checker e WAI
Resultados esperados	•Avaliação

Referências

1. Baranauskas, M. C. C., (2007)
Design Universal e Design Acessível Notas de Aula do curso MO622 - Fatores Humanos em Sistemas Computacionais. Universidade Estadual de Campinas, Instituto de Computação.
2. Baranauskas, M. C. C., (2007)
Mecanismos da Percepção Humana Notas de Aula do curso MO622 - Fatores Humanos em Sistemas Computacionais. Universidade Estadual de Campinas, Instituto de Computação.
3. Calazans, F. (2011)
Biomidiologia: Pokemon
http://www.calazans.ppg.br/c_pok.htm
4. Chisholm, W. A., Henry, S. L., (2005)
Interdependent components of web accessibility In Proceedings of the 2005 international Cross-Disciplinary Workshop on Web Accessibility (W4a), vol. 88, pp. 31-37. ACM.
5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T., (1997)
RFC 2068-Hypertext Transfer Protocol-HTTP/1.1. IETF-Network Working Group.
<http://www.ietf.org/rfc/rfc2068.txt>
6. Heilmann, C., (2006a)
Seven Accessibility Mistakes (part 1)
http://www.digital-web.com/articles/seven_accessibility_mistakes_part_1/
7. Heilmann, C., (2006b)
Seven Accessibility Mistakes (part 2)
http://www.digital-web.com:80/articles/seven_accessibility_mistakes_part_2/
8. International Standards Organization (ISO)., (1998)
Ergonomic requirements for office work with display terminals (VDTs). Part 11: Guidance on usability Genève.
9. Melo, A. M., Baranauskas, M. C. C., (2005)
Design e Avaliação de Tecnologia Web-acessível. XXV Congresso da Sociedade Brasileira da Computação. pg. 1500-1544.
<http://www.sbc.org.br/bibliotecadigital/download.php?paper=131>
10. Nielsen, J., (2007)
Breadcrumb Navigation Increasingly Useful.
www.useit.com/alertbox/breadcrumbs.html
11. Nielsen, J., (2000)
Projetando Websites Rio de Janeiro: Elsevier.
12. Nielsen, J., (2007)
Top Ten Mistakes in Web Design.
<http://www.useit.com/alertbox/9605.html>
13. Nielsen, J., (1993)
Usability Engineering Academic Press, Cambridge, MA.
14. Nielsen, J., (1999)
When Bad Design Elements Become the Standard.
<http://www.useit.com/alertbox/991114.html>

15. Robertson, S., (2003)
Accesskeys: Unlocking Hidden Navigation.
<http://www.alistapart.com/articles/accesskeys/>
16. Rocha, H. V. da, Baranauskas, M. C. C., (2003)
Design e Avaliação de Interfaces Humano-Computador Campinas, NIED/UNICAMP.
17. Rutter, J. P., (2004)
Web Heuristic Evaluation III Web Conference. State College, Pennsylvania.
18. Silva, M. S., (2005)
Componentes essenciais para Acessibilidade à Web.
<http://mauor.com/w3c/wcacomponents.html>
Tradução do texto "Essential Components of Web Accessibility", Web Accessibility Initiative (W3C)
19. Acesso Digital. <http://acessodigital.net>
20. ASES - Avaliador e Simulador de Acessibilidade de Sítios.
<http://www.governoeletronico.gov.br/acoes-e-projetos/e-MAG/ases-avaliador-e-simulador-de-acessibilidade-sitios>
21. ATAG - Authoring Tool Accessibility Guidelines 1.0.
<http://www.w3.org:80/TR/ATAG10/>
22. Centrice.com - Will the browser apply the rule(s)?
<http://centrice.com/ref/css/filters/>
23. Client-Side JavaScript Reference.
<http://docs.sun.com:80/source/816-6408-10/whatsnew.htm>
24. Core JavaScript 1.5 Reference:Operators:Comparison Operators
http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Operators:Comparison_Operators
25. Client-Side JavaScript Reference - Event Handlers.
<http://docs.sun.com:80/source/816-6408-10/handlers.htm>
26. Client-Side Scripting and HTML. <http://www.w3.org:80/TR/WD-script-970314>
27. Color Laboratory. <http://colorlab.wickline.org/colorblind/colorlab/>
28. Colorblind Web Page Filter. <http://colorfilter.wickline.org/>
29. CSS Techniques for Web Content Accessibility Guidelines 1.0.
<http://www.w3.org/TR/WCAG10-CSS-TECHS/>
30. DaSilva. <http://www.dasilva.org.br/>
31. DECRETO nº 5.296 DE 2 DE DEZEMBRO DE 2004.
www.planalto.gov.br/ccivil_03/_ato2004-2006/2004/decreto/d5296.htm
32. e-MAG - Modelo de Acessibilidade de Governo Eletrônico.
<http://www.governoeletronico.gov.br:80/acoes-e-projetos/e-MAG>
33. Gestalt Laws.
http://www.sapdesignguild.org/resources/optical_illusions/gestalt_laws.html
34. Google Mail. <http://gmail.google.com>
35. HTML 4.01 Specification. <http://www.w3.org/TR/html401/>
36. HTML Techniques for Web Content Accessibility Guidelines 1.0.
<http://www.w3.org/TR/WCAG10-HTML-TECHS>
37. IRT - JavaScript Guidelines and Best Practices. <http://www.irt.org/articles/js169/>
38. JAWS. http://www.freedomscientific.com/fs_downloads/jaws.asp
39. JSLint: The JavaScript Verifier. <http://www.jshint.com/lint.html>

40. Juicy Studio - Luminosity Colour Contrast Ratio Analyser.
<http://juicystudio.com/services/luminositycontrastratio.php>
41. Lomak - Light Operated Mouse and Keyboard.
<http://www.lomakkeyboard.com/index.html>
42. Lynx. <http://lynx.isc.org/>
43. Micropower - Dicas de acessibilidade - Regras para criação de páginas acessíveis pelo Virtual Vision. <http://www.micropower.com.br/v3/pt/acessibilidade/dicas.asp>
44. NVDA. <http://www.nvda-project.org/>
45. Quirksmode - JavaScript: General introduction.
<http://www.quirksmode.org/js/intro.html>
46. Quirksmode - JavaScript: Object detection.
<http://www.quirksmode.org/js/support.html>
47. Todos Nós - UNICAMP acessível. <http://www.todosnos.unicamp.br/>
48. The Center for Universal Design: The Principles of Universal Design, Version 2.0.
Raleigh, NC:North Carolina State University. (1997)
http://www.design.ncsu.edu/cud/about_ud/udprincipleshtmlformat.html#top
49. UAAG - User Agent Accessibility Guidelines 1.0. <http://www.w3.org/TR/UAAG10/>
50. UAAG - User Agent Accessibility Guidelines 2.0. <http://www.w3.org/TR/UAAG20/>
51. W3C Recommendation - Cascading Style Sheets, level 1 - Cascading order.
<http://www.w3.org/TR/REC-CSS1#cascading-order>
52. W3Schools - CSS 2 Reference. http://www.w3schools.com/css/css_reference.asp
53. W3Schools - CSS Tutorial. <http://www.w3schools.com/css/>
54. W3Schools - HTML Tutorial. <http://www.w3schools.com/html/>
55. W3Schools - JavaScript Event Reference.
http://www.w3schools.com/jsref/jsref_events.asp
56. WAI - Web Accessibility Initiative. <http://www.w3.org/WAI/>
57. WCAG - Web Content Accessibility Guidelines 1.0. <http://www.w3.org/TR/WCAG10/>
58. WCAG - Web Content Accessibility Guidelines 2.0. <http://www.w3.org/TR/WCAG20/>
59. WebAIM - Web Accessibility In Mind: "Skip Navigation" Links.
<http://www.webaim.org/techniques/skipnav/>
60. WebAnywhere. <http://webanywhere.cs.washington.edu/>

Glossário

abreviação

Omissão de parte de uma ou mais palavras para formar uma nova palavra reduzida. Por exemplo: Unicamp é abreviação de Universidade Estadual de Campinas.

WARAU: Abreviações e siglas - as tags HTML <abbr> e <acronym>

acessibilidade

Segundo a legislação brasileira sobre acessibilidade, é condição para utilização, com segurança e autonomia, total ou assistida, dos espaços, mobiliários e equipamentos urbanos, das edificações, dos serviços de transporte e dos dispositivos, sistemas e meios de comunicação e informação, por pessoa com deficiência ou com mobilidade reduzida. Entretanto, acessibilidade diz respeito à facilidade de acesso, por qualquer pessoa, aos ambientes físicos, aos bens e serviços, às pessoas, à informação.

WARAU: O que é Acessibilidade?

acessibilidade na Web

Promoção de acesso indiscriminado à *world wide Web*, considerando diferenças entre usuários, tecnologias de acesso e contexto de uso. Segundo Hull (2006), qualquer pessoa usando qualquer tecnologia de navegação na *Web* deveria estar apta a visitar qualquer *site*, obter a informação que ele oferece e interagir com o *site*.

WARAU: Acessibilidade na Web

acrônimo

Utilização de iniciais de palavras para formar um único termo. Por exemplo: CEP é um acrônimo para Código de Endereçamento Postal. Também conhecido como sigla.

WARAU: Abreviações e siglas - as tags HTML <abbr> e <acronym>.

agente de usuário

Qualquer *software* que recupera e apresenta conteúdo *Web* para seus usuários. Ex. navegadores, *players* de mídia, *plug-ins* e outros programas - tecnologias assistivas - que auxiliam a recuperar, a apresentar e a interagir com o conteúdo *Web*.

AJAX

Acrônimo em inglês para *Asynchronous JavaScript And XML*, que é uma técnica de transmissão assíncrona de dados em XML utilizando a linguagem JavaScript. O formato dos dados transmitidos pode ser diferente (e.g., texto ou JSON), resultando em outros acrônimos (e.g., AJAT, AJAJ). Note que AJAX não é uma linguagem.

WARAU: AJAX.

Ampliador de telas

Artefato utilizado para ampliar uma parte da tela do computador. É comumente utilizado por pessoas com baixa acuidade visual e pode ser um *software* ou *hardware*.

WARAU: Tecnologias Assistivas.

ATAG

Acrônimo em inglês para *Authoring Tools Accessibility Guidelines*. São diretrizes para que ferramentas de autoria de código *Web* ofereçam os recursos necessários para a criação de código acessível (conforme o WCAG), definidas por um subgrupo do WAI/W3C.

avaliação de acessibilidade Web

Diz respeito à observação da acessibilidade em páginas e *websites*. Pode combinar uma série de técnicas como a verificação da codificação usada em páginas *Web* com auxílio de ferramentas automáticas; uso de navegadores e tecnologias assistivas em diferentes configurações de acesso; uso de ferramentas semiautomáticas de avaliação de acessibilidade; verificação por pessoas experientes na avaliação de acessibilidade, apoiadas por pontos de verificação ou critérios de sucesso; avaliação com auxílio de diferentes usuários em diferentes situações de acesso (tradicionalmente via testes de usabilidade); e revisão da linguagem utilizada nas páginas.

audiodescrição

Uma audiodescrição consiste da apresentação, por meio de áudio, de elementos visuais importantes que não representados nos diálogos, por exemplo: ambiente, expressões faciais, roupas.

Braille

Código de leitura tátil e de escrita, usado por pessoas cegas. Usa conjuntos de seis pontos em relevo dispostos em duas colunas, possibilita a formação símbolos usados em literatura nos diversos idiomas, na simbologia matemática e científica, na música e na informática.

browser

Ver navegador.

CSS

Acrônimo em inglês para *Cascading Styles Sheet*, que é a definição da aparência de um arquivo que utiliza linguagem de marcação (e.g., HTML, XML). A tradução do Acrônimo resulta em Folhas de Estilo em Cascata, em que cascata se refere à ordem de prioridade que deve ser utilizada pelo navegador ao aplicar as formas de apresentação a uma dada página.

design acessível

Foca em princípios que estendem o processo de design padrão para pessoas com algum tipo de limitação.

WARAU: Design Universal e Design Acessível.

design inclusivo

Ver design para todos.

design para todos

Abordagem europeia ao design universal.

design universal

Design de produtos e ambientes que sejam usáveis por todas as pessoas, na maior extensão possível, sem a necessidade de adaptação ou design especializado.

WARAU: Design Universal e Design Acessível

diretriz

Ver *guideline*.

e-MAG

Modelo de Acessibilidade de Governo Eletrônico, desenvolvido pelo Governo Brasileiro, que oferece um conjunto de recomendações para o desenvolvimento de *sites* governamentais.

Flash

Tecnologia que viabiliza adicionar animação e interatividade em páginas *Web*. Pode ser usada para publicar conteúdos em diferentes mídias: gráficos, textos, vídeo, áudio, etc. Embora seja comum restrição quanto ao uso dessa tecnologia para o desenvolvimento de *sites*, o problema não está na tecnologia em si, mas na forma com tem sido usada: o conteúdo em *flash* raramente é desenvolvido para incluir diferentes estratégias de interação, o que o torna inacessível de alguma maneira.

ferramenta automática de verificação de código

Inspecionam o código de páginas *Web* a partir de uma determinada especificação (ex. HTML, CSS, etc), validando o código ou indicando problemas a serem corrigidos.

WARAU: Validação de código HTML

WARAU: Validação de folhas de estilo CSS

ferramenta de autoria

Software que oferece um conjunto de recursos para a criação de conteúdos, sejam eles, texto, imagem, código fonte, ou outro qualquer.

ferramenta semiautomática de avaliação de acessibilidade

Auxilia na verificação de páginas *Web*, comparando-as com pontos de verificação (ou critérios de sucesso) de diretrizes de acessibilidade. Embora seja de grande valor e praticamente indispensáveis à avaliação de acessibilidade de uma página na *Web*, indicando erros e possíveis problemas de acessibilidade agrupados em níveis de prioridades, e oferecendo orientações, algumas questões ainda precisam de avaliação por pessoas.

WARAU: Avaliação de acessibilidade utilizando ferramentas semiautomáticas

folha de estilo

Ver CSS.

guideline

Orienta o *designer* na tomada de decisões consistentes através dos elementos que constituem o produto (ex. conteúdo, apresentação, funções, etc.). Devem ser entendidas e aplicadas de forma contextualizada (ROCHA e BARANAUSKAS, 2003).

HTML

Acrônimo em inglês para *HyperText Markup Language*, que é uma linguagem de marcação para páginas *Web*.

WARAU: Estrutura de documentos HTML

heurística

Regra geral que objetiva descrever propriedades comuns de interfaces usáveis (ROCHA e BARANAUSKAS, 2003).

WARAU: Inspeção Heurística de Usabilidade, Heurísticas Genéricas.

IBGE

Acrônimo para Instituto Brasileiro de Geografia e Estatística.

INAF

Acrônimo para Indicador Alfabetismo Funcional, mantido pelo Instituto Paulo Montenegro.

inclusão

Segundo Maria Teresa Eglér Mantoan, "é a nossa capacidade de entender e reconhecer o outro e, assim, ter o privilégio de conviver e compartilhar com pessoas diferentes de nós".

independência de dispositivo

Fato de que algum artefato de *software* poder ser utilizado por usuários com diferentes dispositivos.

WARAU: Manipuladores de evento em HTML

JavaScript

Linguagem de programação interpretada e multi-plataforma mantida pela Netscape. Ela pode ser utilizada no lado do servidor e no lado do usuário. No entanto, é mais utilizada em aplicações no lado do cliente. Ela é baseada no padrão ECMA-262, que define a linguagem ECMAScript.

WARAU: Inclusão de código JavaScript

JSON

Acrônimo em inglês para *JavaScript Object Notation*, que é um formato de troca de dados baseado em um subconjunto da linguagem de programação JavaScript. Alguns de seus pontos positivos estão: ser leve, ser de fácil leitura para humanos e ser de fácil geração e interpretação por sistemas computacionais.

leitor de telas

Tecnologia assistiva que tem a finalidade de ler o conteúdo textual que é exibido no computador e, por meio de síntese de voz ou *displays* em Braille, apresenta o conteúdo lido.

WARAU: Tecnologias Assistivas

LIBRAS

Acrônimo para Língua Brasileira de Sinais.

linearização do conteúdo

Refere-se a ordem do conteúdo quando toda a formatação é removida. Trata-se de um aspecto importante no *design* de *websites*, especialmente para a leitura do conteúdo por usuários de leitores de telas.

linguagem de marcação

Linguagem que define termos e uma sintaxe específica para demarcar conteúdo textual com o uso de *tags* (ou marcadores).

navegador

Programa utilizado para navegar na *Web*. Também conhecido como *browser*.

PDF

Acrônimo em inglês para *Portable Document Format*, que é um formato ou tipo de documento criado para troca de documentos, independentes de *software*, *hardware* ou sistema operacional. Documentos neste formato também devem ser publicados com acessibilidade em mente.

recomendação

Ver *guideline*.

redundância

Na Teoria da Comunicação diz respeito ao excesso ou desperdício de sinais ou de signos na transmissão da mensagem, que serve, contudo, para neutralizar os efeitos do ruído no canal de comunicação. É estratégia bastante comum na promoção da acessibilidade.

WARAU: Recomendações de Acessibilidade para: Redundância

robustez

Em acessibilidade na *Web*, diz respeito à compatibilidade do conteúdo com tecnologias correntes e futuras, incluindo tecnologias assistivas.

sigla

Ver acrônimo.

síntese de voz

Reprodução artificial da voz humana.

tag

É uma marca utilizada em linguagens de marcação para diferenciar conteúdo de marcação. Por exemplo: <tag>Conteúdo textual</tag>.

tecnologia assistiva

No escopo da informática são todos os artefatos que auxiliam de alguma forma as pessoas com algum tipo de necessidade, seja ela física, ambiental, etc. As tecnologias assistivas podem ser tanto *hardware* (e.g., impressora Braille, linhas Braille, apontadores) como *software* (e.g., leitores de telas, ampliadores de telas, navegadores textuais, barras de acessibilidade com ajustes de tamanho de texto e contraste).

WARAU: Tecnologias Assistivas.

texto alternativo

Atributo utilizado em arquivos de marcação para representar funções de imagens e gráficos através de conteúdo textual. Dessa forma, em contextos que não utilizam as imagens (e.g., usuários de leitores de telas ou de navegadores textuais) as funções das imagens não são perdidas.

WARAU: Imagens - A tag HTML

UAAG

Acrônimo em inglês para *User Agent Accessibility Guidelines*. São diretrizes para tornar acessíveis navegadores, *players* de mídia, tecnologias assistivas, entre outros, definidas por um subgrupo do WAI/W3C.

UNESCO

Acrônimo em inglês para *United Nations Educational, Scientific and Cultural Organization*.

usabilidade

Resumidamente, um produto tem boa usabilidade se pode ser utilizado por seus usuários de forma que eles atinjam seus objetivos com eficácia, eficiência e satisfação.

WARAU: O que é usabilidade?

W3C

Abreviação em inglês para *World Wide Web Consortium*, que é um consórcio internacional que desenvolve padrões (*Web standards*) e recomendações para manutenção da *Web*, de modo a manter suas tecnologias compatíveis umas com as outras (independentes de plataforma e interoperáveis).

WAI

Acrônimo em inglês para *Web Accessibility Initiative*. É um grupo do W3C que investiga soluções para acessibilidade na *Web*. O WAI tem subgrupos que investigam navegadores, *players* de mídia, ferramentas assistivas (UAAG), ferramentas de autoria de código *Web* (ATAG) e código *Web* (WCAG).

WCAG

Acrônimo em inglês para *Web Content Accessibility Guidelines*, que corresponde ao conjunto de diretrizes de acessibilidade para desenvolvimento de conteúdo *Web*, definido por um subgrupo do WAI/W3C.

webdesign

Todo o processo de desenvolvimento de sistemas *Web*, desde sua concepção até sua publicação.

Web standards

Oferecem uma base comum para o desenvolvimento de tecnologia *Web* independente de plataforma e interoperável. Buscam simplificar a manutenção das páginas *Web* e a sua indexação por mecanismos de buscas, entre outros benefícios. São exemplos as especificações do W3C para a linguagem HTML e para folhas de estilo CSS, o padrão ECMA-262 que define a linguagem ECMAScript.

XML

Acrônimo em inglês para *eXtensible Markup Language*, que é uma linguagem de marcação extensível, ou seja, permite a utilização de *tags* customizadas.

Referências do Glossário

1. Braille Virtual - Universidade de São Paulo. Disponível em: <<http://www.braillevirtual.fe.usp.br/>>. Acesso em: 15 de setembro de 2008.
2. ECMA International - Standard ECMA-262 Disponível em: <<http://www.ecma-international.org/publications/standards/Ecma-262.htm>>. Acesso em: 15 de setembro de 2008.
3. HULL, L. Accessibility: It's not just for disabilities any more. Interactions, New York, v. 11, n. 2, 36-41, Mar./Abr. 2004.
4. JSON. Disponível em: <<http://www.json.org/>>. Acesso em: 15 de setembro de 2008.
5. MELO, A. M. Design inclusivo de sistemas de informação na Web. 2007. xxiv, 339 p. (Doutorado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2007. Disponível em: <<http://libdigi.unicamp.br/document/?code=vtls000438900>>. Acesso em: 15 de setembro de 2008.
6. MÍDIA e Deficiência. Brasília: Andi, 2003. 184 p. (Diversidade). Disponível em: <http://www.andi.org.br/pdfs/Midia_e_deficiencia.pdf>. Acesso em: 17 de setembro de 2008.
7. Netscape - Core JavaScript Reference 1.5: Core JavaScript Reference. Disponível em: <<http://devedge-temp.mozilla.org/library/manuals/2000/javascript/1.5/reference/>>. Acesso em: 15 de setembro de 2008.
8. PUPO, D. T.; MELO, A. M.; PÉREZ FERRÉS, S. (Org.) Acessibilidade: discurso e prática no cotidiano das bibliotecas. Campinas: Unicamp/Biblioteca Central Cesar Lattes, 2006. 91 p. Disponível em: <http://styx.nied.unicamp.br:8080/todosnos/artigos-cientificos/livro_acessibilidade_bibliotecas.pdf>. Acesso em: 18 de setembro de 2008.
9. ROCHA, H. V.; BARANAUSKAS, M. C. Design e Avaliação de Interfaces Humano-Computador. Campinas: Nied-Unicamp, 2003. 244 p. Disponível em: <http://www.nied.unicamp.br/download_livro.html>. Acesso em: 17 de setembro de 2008.
10. WebAIM - Web Accessibility in Mind. Creating Accessible Flash Content. Disponível em: <<http://www.webaim.org/techniques/flash/>>. Acesso em: 3 de outubro de 2008.
11. W3C - World Wide Web Consortium. Web Accessibility Guidelines 1.0 - Appendix B. 1999. Disponível em: <<http://www.w3.org/TR/WCAG10/#glossary>>. Acesso em: 15 de setembro de 2008.
12. W3C - World Wide Web Consortium. Web Accessibility Guidelines 2.0 - Appendix A. 2008. Disponível em: <<http://www.w3.org/TR/WCAG20/#glossary>>. Acesso em: 15 de setembro de 2008.