Guerra contra o terror

Os porques de não desenvolver para browsers antigos

"It is fatal to enter any war without the will to win it."

- General Douglas MacArthur

Introdução

Essa decisão é importante

É muito difícil tomar a decisão de não suportar browsers antigos ao produzir um grande site. Sempre há um número que se elevado a milhões de usuários o problema pode se agravar. Para uma grande empresa que tem 8 milhões de clientes, ignorar 1% destes usuários é algo realmente fora de questão. O ponto é que você não precisa ignorar estes valiosos 1%. Mas também não precisa gastar milhões nem perder muito tempo tentando resolver os problemas que estes usuários trazem para a produção. Nossa ideia aqui é dar a melhor experiência que eles podem ter sem prejudicar os outros 99%.

O número de 1% que eu citei acima é simbólico. Este 1% varia de empresa para empresa. Depende do porte, depende do mercado. Pode ser que no seu caso eu esteja falando de um mercado de 80.000 usuários, mas que 10% destes usuários utilizam um browser antigo. Quero que você entenda que você ou o seu cliente **não vai perder seus visitantes/usuários/clientes**. Entenda que esse documento traz informações importantes para que você economize tempo e dinheiro e saiba realmente qual o problema. Acredite, é mais fácil do que você pode imaginar. Mas primeiro preciso que você visualize o cenário. Se quiser pular toda essa baboseira, não faça isso. Tente tomar um tempo para ler e entender exatamente o que quero dizer aqui.

Para que serve este documento?

Este documento é recomendado para qualquer um que trabalhe com Web. Não importa se você é desenvolvedor, gerente, diretor, PMO, CIO, CEO, DPT[1], ou qualquer outra sigla desinteressante que você tenha no seu crachá, se seu mercado é o mercado de web, você precisa ler este documento ou perderá tempo e dinheiro ao desenvolver seu site.

Quero mostrar qual sua linha de corte. Essa linha de corte vai fazê-lo concentrar esforços para os problemas reais, você vai poder realocar as pessoas da sua equipe para tarefas realmente importantes e economizará dinheiro ao produzir um website (portal, sitezinho, sistema ou aplicações baseadas em web) diminuindo o tempo de manutenção, de produção e equipe.

Se você for um diretor ocupado e não quer perder tempo com este assunto, passe esse documento para um gerente de confiança e peça para que ele resolva o problema.
Se você é o gerente que recebeu esse documento e acha que tem coisa melhor pra resolver, dê para um desenvolvedor de confiança da sua equipe. Além deste documento dê a ele autonomia. Os resultados virão e provavelmente você será promovido.

Este documento explica quais os problemas de manter o suporte a browsers antigos. Mas mais do que explicar os problemas, ele dá soluções e sugestões que podem ser seguidas para diminuir custos na produção. Este documento não é mais uma campanha para que o Internet Explorer 6 seja extinto. Eu não falo aqui apenas sobre o IE6, mas também sobre qualquer browser ou meio de acesso antigo ou que traga problemas para a produção. Usarei o IE6 como exemplo durante todo o documento apenas porque ele é o caso mais comentado entre os desenvolvedores. Mas isso se aplica a browsers como Firefox 3.0, Internet Explorer antigo para mobiles e outros meios de acesso antiquados como por exemplo aqueles celulares que só conseguem ler WAP.

---

*[1] Abreviação da posição empresarial chamada Dono da P*rra toda.*

Quero dizer isso novamente para que fique claro: não vou dizer para você simplesmente deixar de dar suporte a estes meios de acesso. Você certamente perderia dinheiro e tempo se fizesse isso. Quero sugerir outro caminho, um caminho mais inteligente.

A Microsoft não quer mais continuar com o IE6

Este deve ser um motivo muito relevante para você tomar uma decisão sensata. A Microsoft lançou no dia 04/03/2011 um website chamado The IE6 Countdown[2], onde ela incentiva fortemente a migração do IE6 para uma versão mais atual, no caso o IE8. Claro que a Microsoft não está indicando a migração para qualquer um dos browsers concorrentes, mas cá entre nós, isso não importa. O que realmente importa é que a própria Microsoft está ciente dos problemas que o IE6 acarreta na produção web.

O interessante é que a Microsoft sabe que algumas empresas tem sérias restrições de upgrade de software e por isso ela divulgou alguns estudos de casos de grandes empresas que fizeram a migração, como a Dell. No site você também encontra um estudo de impacto financeiro referente a migração do IE6 para o IE8.

No mês de Fevereiro de 2011, de acordo com a Net Applications[3], o IE6 tinha 12% de marketshare no mundo todo. A Microsoft quer diminuir essa porcentagem para menos de 1% com a ajuda da campanha.

O básico

Você já deve saber para que serve o HTML. Mas se quiser se aprofundar ainda mais sobre assunto, abaixo descrevo o conceito e a teoria da utilidade do HTML. É algo bem simples de ser entendido e que mesmo você não sendo técnico conseguirá assimilar bem.
Se você quiser pular essa parte, fique à vontade, mas aconselho de verdade que você a leia em algum momento posterior.

Para que serve o HTML?

A Web foi criada para ser um ambiente onde fosse possível trocar informações livremente e que essas informações pudessem ser acessadas ao redor do planeta por qualquer pessoa ou meio de acesso. Em 1994, foi criado o W3C[4] (World Wide Web Consortium): um consórcio internacional onde são desenvolvidas os padrões para a web (Web Standards) tais como: HTML, CSS, XML, XSLT, entre outros.

O HTML é baseado no conceito de Hipertexto. Hipertexto são conjuntos de elementos – ou nós – ligados por conexões. Você pode chamar este nós de LINKS. Estes links podem ser elementos como palavras, imagens, vídeos, audio, documentos etc. Eles não estão conectados linearmente como se fossem textos de um livro, onde um assunto é ligado ao outro seguidamente. A conexão feita em um hipertexto é algo imprevisto que permite a comunicação de dados, organizando conhecimentos e guardando informações relacionadas.
Há muita história por trás disso. Se estiver mais interessado do que eu imagino leia sobre Memex[5], criado por Vannevar Bush[6].

Para distribuir informação de uma maneira global, é necessário haver uma linguagem que seja entendida universalmente por diversos meios de acesso. O HTML se propõe a ser esta linguagem.
Desenvolvido originalmente por Tim Berners-Lee o HTML ganhou popularidade quando o Mosaic - browser desenvolvido por Marc Andreessen na década de 1990 - ganhou força. A partir daí, desenvolvedores e fabricantes de browsers utilizaram o HTML como base, compartilhando as mesmas convenções.

---

[2] http://theie6countdown.com/
[3] http://www.netapplications.com/
[4] http://www.w3.org/Consortium/
[5] http://en.wikipedia.org/wiki/Memex
[6] http://en.wikipedia.org/wiki/Vannevar_Bush

O começo e a interoperabilidade

Entre 1993 e 1995, o HTML ganhou as versões HTML+, HTML2.0 e HTML3.0, onde foram propostas diversas mudanças para enriquecer as possibilidades da linguagem. Contudo, até aqui o HTML ainda não era tratado como um padrão. Apenas em 1997, o grupo de trabalho do W3C responsável por manter o padrão do código, trabalhou na versão 3.2 da linguagem, fazendo com que ela fosse tratada como prática comum. Você pode ver as mudanças em http://migre.me/3ZhIF.

Desde o começo o HTML foi criado para ser uma linguagem independente de plataformas, browsers e outros meios de acesso. Interoperabilidade significa menos custo. Você cria apenas um código HTML e este código pode ser lido por diversos meios, ao invés de versões diferentes para diversos dispositivos. Dessa forma, evitou-se que a Web fosse desenvolvida em uma base proprietária, com formatos incompatíveis e limitada.

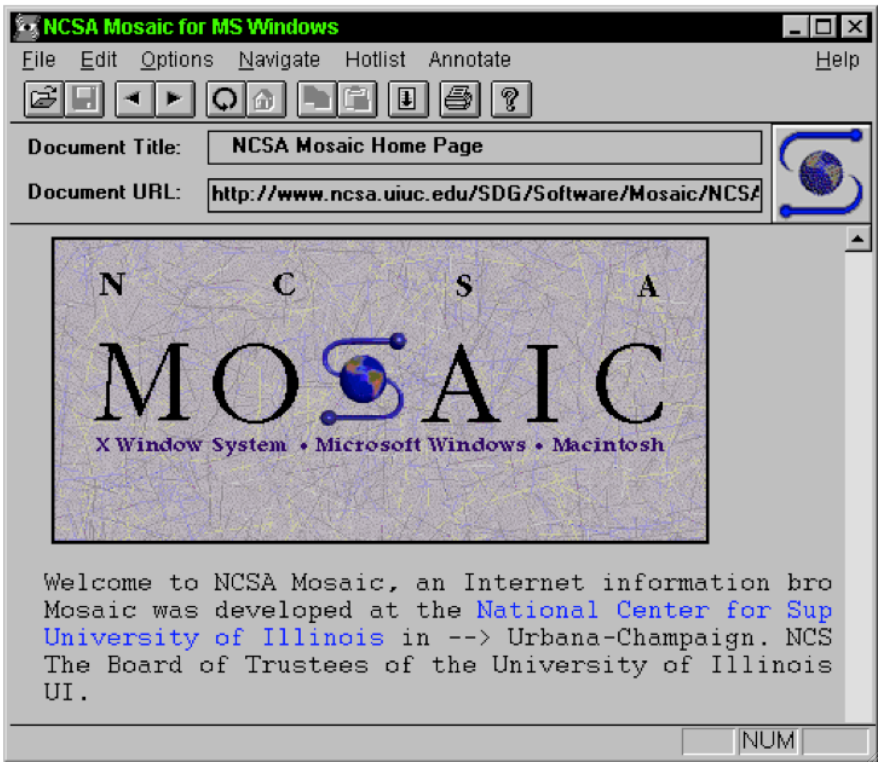
Por isso o HTML foi desenvolvido para que essa barreira fosse ultrapassada, fazendo com que a informação publicada por meio deste código fosse acessível por dispositivos e outros meios com características diferentes, não importando o tamanho da tela, resolução, variação de cor. Dispositivos próprios para deficientes visuais e auditivos ou dispositivos móveis e portáteis. O HTML deve ser entendido universalmente, dando a possibilidade para a reutilização dessa informação de acordo com as limitações de cada meio de acesso.

Introdução a Guerra dos Browsers

Não quero me alongar muito nessa história que você já deve saber de trás para frente e também não é o foco aqui. Mas para contextualizar nossos cenários é muito bom.

O browser que popularizou a web foi o Mosaic. Entenda que o Mosaic não foi o primeiro browser criado. Mas foi ele que deixou a web popular entre as pessoas comuns.
Antes disso, o Tim Berners-Lee em um computador NeXT, criou o primeiro browser chamado WorldWideWeb[7], lançando apenas para um grupo de estudantes no CERN em Março de 1991. Depois dali criou-se alguns outros browsers que foram mais populares em alguns nichos e universidades.

O Mosaic[8] foi criado pelo Marc Andreessen e Eric Bina.
A partir daí todos os browsers, até mesmo os atuais, foram inspirados pelo Mosaic. Até hoje os browsers carregam algumas características de uso e elementos herdados do Mosaic.



[7] http://en.wikipedia.org/wiki/WorldWideWeb
[8] http://en.wikipedia.org/wiki/Mosaic_(web_browser)

Depois que James H. Clark investiu no projeto o Mosaic virou Netscape. E é aqui que começa nossa guerra. O Netscape então virou o browser mais popular da web e era atualizado com frequencia trazendo novas features para seus usuários. O primeiro Internet Explorer foi lançado no Windows 95 Plus.

A partir daí os dois browsers começaram a briga por usuários. Era uma briga ferrenha onde cada um dos browsers tentava se adiantar e lançar uma versão melhor que a outra. Até aí, ótimo. Concorrência sempre é bom! O problema é que o W3C neste tempo ainda esta muito imatura. Os padrões que eles sugeriam não eram reconhecidos. E como eu disse, eram apenas sugestões, recomendações de uso. O W3C nunca pode impor o que os desenvolvedores devem ou não seguir e a mesma coisa se aplica os fabricantes de browsers.

Alguns dos pontos mais importantes para o desenvolvimento de websites ainda não estavam claros o que levou a Netscape e a Microsoft criarem códigos proprietários. Quem não se lembra da tag LAYER, BLINK e MARQUEE?

Outro problema foi que a popularidade da criação de websites com tabelas ganhou força. Isso dificultou muito a vida de todos nós até hoje e atrasou a adoção dos padrões pelos desenvolvedores e pelos fabricantes de browsers. O trabalho com as tabelas era insano. Levava-se muito tempo para fabricar um website com um design elaborado. Algumas features interessantes de Javascript e CSS não funcionavam uniformemente nos browsers o que levou os desenvolvedores a criarem sempre duas versões do website. Era comum que ao entrar nos websites você escolhia qual dos browsers você estava usando para que você fosse redirecioando para o website correto.

Toda essa confusão durou algum tempo até que um grupo de desenvolvedores resolveram fazer uma revolução. Eles formaram um movimento chamado WaSP - Web Standards Project[9]. Estes caras são os responsáveis por fazermos websites com padrões web hoje. Eles convenceram todo mundo a desenvolverem websites utilizando melhores práticas e de acordo com o que o W3C falava.

Sem dúvida estamos vivendo um cenário muito melhor do que o cenário anterior. Isso tende a melhorar com o tempo com toda a concientização de seguir os padrões web, tanto dos desenvolvedores quanto dos fabricantes de browsers. A cada ciclo completado, ou seja, a cada browser antigo que se extingue, novas portas se abrem e o trabalho com web fica mais interessante. O trabalho não diminui, mas com certeza é mais facilitado.

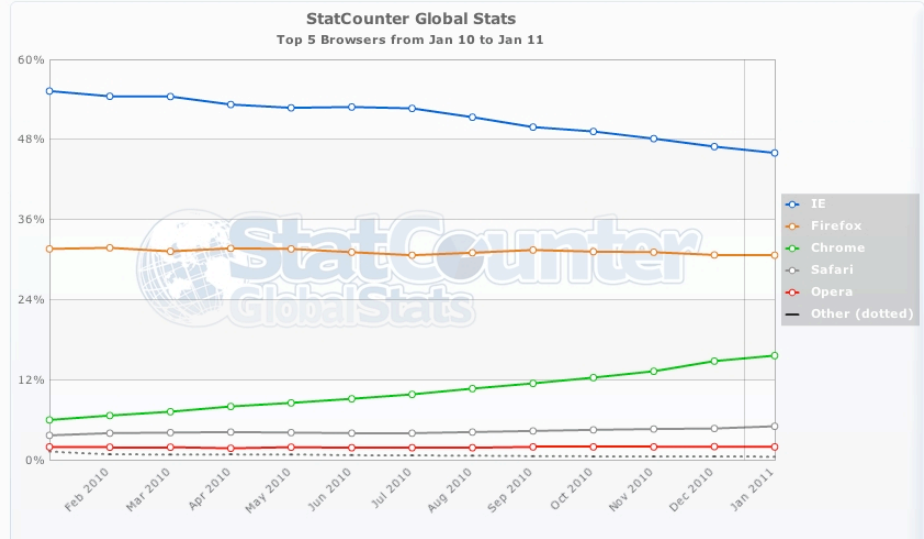
[9] http://www.webstandards.org/

O mercado de browsers

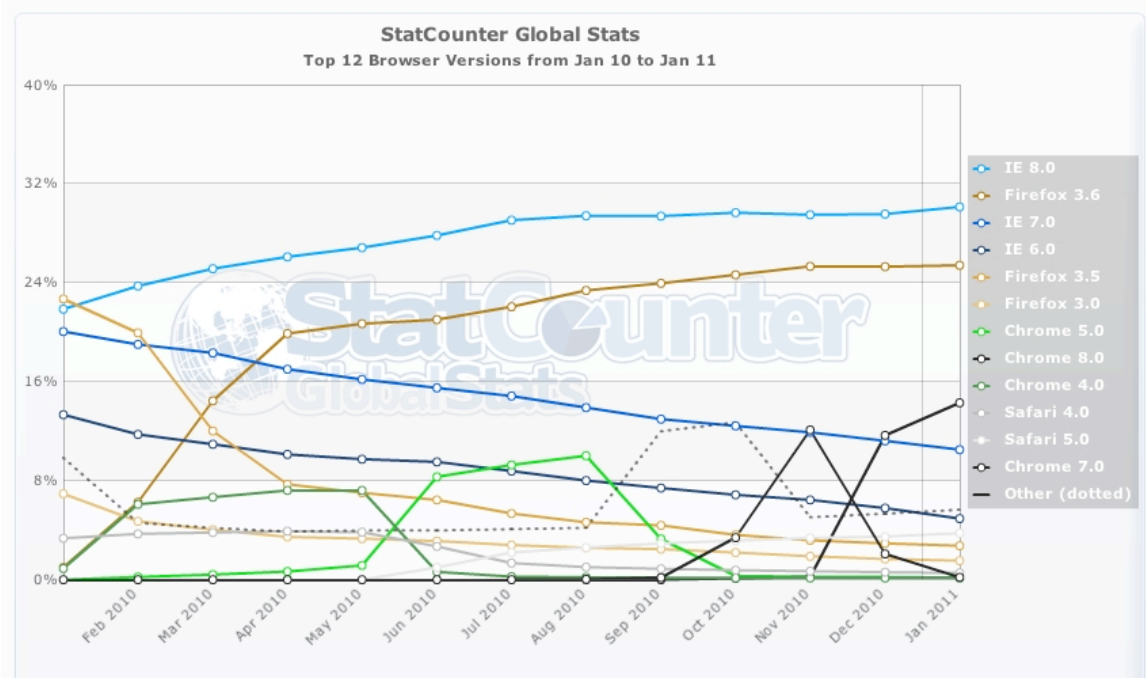
Atualmente - até o dia em que escrevi este capítulo (Janeiro de 2011) - existem 5 principais browsers. As versões listadas são as atuais até a data de quando este documento foi escrito:

1. Chrome 9
2. Firefox 3.6 (já é possível baixar o Beta 4)
3. Internet Explorer 8 (já é possível baixar o Beta 9)
4. Opera 11
5. Safari 5

Todos eles tem quase 100% de suporte para o CSS 2.1 e parte do CSS 3, pelo menos a parte que facilita o desenvolvimento. Apenas os browsers que tem motor Webkit (Chrome e Safari) tem suporte a CSS3 Animation ou 3D.
Abaixo segue um gráfico de Janeiro de 2010 até Janeiro de 2011 que mostra o marketshare dos browsers listados acima. Esta é uma visão Global. Iremos analisar mais pra frente o mercado especificamente do Brasil.



Veja que o Internet Explorer até hoje lidera o topo com uma pequena baixa. Desde quando o Firefox surgiu o Internet Explorer tem perdido vagarosamente seus pontos de ano em ano. Com Chrome e Safari isso melhorou um pouco, mas ainda assim a grande maioria utiliza o Internet Explorer como seu browser padrão. Temos que entender que o IE tem tantos adeptos por conta do monopolio da Microsoft. Mas o Internet Explorer 8 e até mesmo a versão 7 em alguns casos não apresentam muitos problemas de renderização ou compatibilidade como o Internet Explorer 6 apresenta. Abaixo vemos o mesmo gráfico colocado acima, mas agora com a versão dos browsers descritas para que tenhamos uma visão mais interessante do mercado.



Veja que o IE6 tem algo em torno de 5% de marketshare no mercado de browsers. O IE7 tem em torno de 10% e o IE8, liderando o trio com 30%.

O problema de se ter várias versões de um mesmo browsers é que em cada versão os fabricantes implementam muitas atualizações que podem quebrar a compatibilidade de alguns sites. Isso acontece em atualizações pontuais, como por exemplo os problemas que acontecem com o Firefox entre as versões 3.0 e 3.5/6. Há muitas diferenças entre Firefox 3.0 e versões posteriores. A versão 3.0 representa globalmente 1.53% dos usuários. Felizmente os usuários de Firefox entendem que o browser tem atualizações de tempos em tempos, facilitando a reciclagem de versões.

Tudo é mais chato

Desenvolver para browsers antigos é complicado. Muito complicado. Onde você acha que levaria 10 minutos, pode levar 1 hora. Onde você acha que seriam duas linhas de código, na verdade são 100. Não estou exagerando. Você usará mais de tudo para desenvolver para browsers antigos: mais pessoas, mais tempo, mais código. Isso gerará mais problemas de manutenção, mais investimentos para contratar mais pessoas, mais trabalho com monitoramento de problemas e soluções.

Desenvolver para browser antigos leva tempo. É necessário criar um planejamento estratégico sobre quais serão as medidas para manter o projeto durante e após a produção. É no começo que os gestores devem tomar a decisão de nivelar por cima e manter a sanidade da equipe no nível saudável ou nivelar por baixo, fazendo com que browsers de 10 anos atrás sejam tratados da mesma forma que os browsers atuais.

Mais código, mais manutenção

Lembre-se: quanto mais código escrito, mais cuidado é preciso. Não estou dizendo que você deve ficar apavorado se um dos seus arquivos de CSS ficar com 4000 linhas de código. Isso é inevitável dependendo do tamanho do projeto. O problema é se essas 4000 linhas não estiverem comentadas ao longo do trabalho. Se a nomenclatura e a organização dos elementos não for clara o suficiente para que um grupo de pessoas entenda e o código não fique travado no conhecimento de apenas um desenvolvedor.

Há diversos motivos para que seu código fique gigante. Manter a compatibilidade com browsers antigos é um destes motivos. Na verdade é o motivo onde não há nada o que fazer para manter menos linhas de código. Quando um desenvolvedor está iniciando, é normal ele escrever mais código. Isso se resolve com prática e estudo. Quando temos um site com muitas animações ou transições de elementos, é normal que o código fique grande, mas isso pode ser resolvido tirando uma ou outra transição.
Quando o problema e compatibilidade com browsers antigos, não é possível diminuir a quantidade de código. Você fatalmente vai precisar manter uma ou duas folhas de estilo CSS separadas para poder manter sob controle as diferenças que variam entre quebras de layout ou diferenças de pixels na posição dos botões. Abaixo falo mais sobre isso.

Mantendo duas versões

Em meados de 1996 os desenvolvedores eram obrigados a manter duas versões do mesmo site para IE e Netscape. Naquele tempo os dois navegadores não se baseavam no W3C para renderizar o HTML e o CSS da forma correta. Na verdade a culpa não era deles. O W3C ainda estava no começo e eles não tinham de verdade os padrões definidos dessas linguagens. O que chamávamos de padrões na verdade eram rascunhos. Logo, o que os browsers não encontravam instruções sobre como fazer a implementação, eles criavam suas próprias regras, gerando códigos proprietários. Resultado: o que o desenvolvedor fazia para IE não funcionava no Netscape e vice-versa. Obviamente os browsers aproveitavam essa situação para minar o campo do concorrente prejudicando a adoção do browser na comunidade.

Quando produzimos um site nivelando a compatibilidade por baixo, podemos gerar os mesmos problemas que cenário do IE e Netscape, por isso precisamos de um plano para a manutenção. Você precisará manter este site em ordem, fazendo com que os elementos fiquem no lugar para manter a fidelidade visual entre os diferentes browsers. É normal manter uma equipe ou um representante, dependendo do tamanho do projeto, para monitorar e manter o site para os browsers mais antigos. Esse desenvolvedor terá apenas uma tarefa:

manter a fidelidade visual e o comportamento dos elementos nos diversos browsers. E é aqui que começa o problema.

É normal separarmos um ou dois arquivos de CSS para corrigir bugs nos browsers antigos. Além de se preocupar com o código integral, feito para funcionar em browsers novos, precisamos lembrar de corrigir e manter outra versão do CSS e Javascript para os browsers como o IE6. Qualquer alteração é necessário reproduzir - muitas vezes utilizando hacks - nos arquivos que cuidam exclusivamente para estes browsers. Em vez de fazer manutenção em um código, você precisa fazer em dois. Não estamos falando aqui apenas de CSS, muitas vezes é necessário manter códigos de Javascript que afetam apenas os browser antigos em arquivos separados para que não comprometa o resto dos browser.

Mais imagens, menos velocidade

Fato: até a versão 8 do Internet Explorer não há suporte a bordas arredondadas. Para os outros browsers, utilizamos apenas uma linha de código. Não é necessário criar imagens e elementos extras. Para o Internet Explorer, na melhor das hipóteses, precisamos criar dois novos elementos e duas imagens extras. Aí você fala: "Você tem preguiça de criar duas imagens e digitar duas linhas a mais de código?" Não é preguiça. Para mim é só um pouco mais de trabalho. O problema são as implicações disso. Para manter o controle terei que fazer esses elementos via javascript, isso significa mais processamento do Browser e consequentemente mais processamento de máquina. Terei que baixar duas imagens a mais no servidor, o que significa mais duas requisições de imagens. Se houver outros tipos de botões diferentes no site, serão outras imagens a serem baixadas, requisitadas e montadas via Javascript para formar o botão.
Há casos onde o uso de CMSs como o Vignette dificultam o uso de Javascript em elementos de uma forma mais genérica. Então o código que colocaríamos via JS para facilitar as coisas deverá ser colocado manualmente em cada elemento que precisa de bordas arredondadas.

O ponto aqui não é criar mais imagens ou fazer mais linhas de código. Isso é importante evitarmos. O problema é que você vai concentrar mais ou menos 30% do seu esforço de trabalho para tentar manter a compatibilidade visual para um browser que tem um tempo de vida restante muito curto. Não vale a pena concentrar o tempo de um desenvolvedor para cuidar deste browser. Os usuários deste browser nem vão perceber a borda arredondada dos botões. Para eles isso é detalhe. Eles estão interessados se o site estará ou não funcionando. Este é o ponto principal e é o que eu protejo aqui. Você pode matar alguns elementos visuais do site, mas ele ainda sim precisa funcionar perfeitamente em qualquer lugar.

Mais investimento... muito mais.

Já vi sites serem vendidos por R$500 e outros por milhões. Obviamente o valor depende muito da complexidade do projeto e do cliente. Se você acha mesmo que vale a pena ter uma, duas ou mais pessoas focadas em manter a compatibilidade com browsers antigos, prepare-se, isso vai sair caro e seu projeto pode atrasar. É o pior dos mundos. Ninguém quer ter um projeto mais caro e que tenha mais riscos de atraso. Para que você realize melhor o que quero dizer: suponha que o valor de hora/homem seja de R$100. Para implementar HTML/CSS/Javascript da home de um site, nos meus bons tempos, eu levaria algo em torno de 4 horas... 5 se o layout for complicado. Isso nos dá R$500. Se for contar com a implementação do IE6, isso pode dobrar facilmente porque terei que criar um CSS apenas para este browser. Não poderei utilizar PNG porque ele não tem suporte a este tipo de imagem. Posso utilizar um plugin de Javascript para tentar fazer funcionar o PNG, mas isso deixaria o site um pouco mais lerdo e não há nenhum plugin que não dê algum tipo de problema de conflito ou defeito na hora de renderizar que valha o trabalho da implementação. Sem contar que se eu usar estes plugins, terei que listar todos os meus elementos ou imagens que são PNG para que o script funcione sem problemas aparentes. Se eu decidir não utilizar um script deste, terei que criar uma imagem em GIF para cada uma das imagens que são PNG. Como o GIF não tem canal alpha de opacidade,

terei que me manter atento para que as imagens que tenham fundo com gradiente fiquem na posição correta para que não haja diferenças de cores entre o background e o GIF.

Estamos falando apenas do recorte de imagens, sem contar as diferenças de layout e os bugs que preciso prever para que os elementos não quebrem a diagramação do site.
Se colocarmos na ponta do lápis isso pode levar o dobro do tempo de desenvolvimento. Sem contar o tempo de manutenção posterior ou algum planejamento de escalabilidade para o futuro. No nosso exemplo o valor do sitezinho de R$500 virou algo em torno de R$1000 ou R$1500. Isso tudo gasta muito tempo. E se tempo é dinheiro, gasta-se muito dinheiro. E esse caso não justifica o investimento. O seu dinheiro, quer dizer, o dinheiro do seu cliente foi pro beleléu.

Se protegendo contra o inevitável

Existe algumas caminhos que podemos tomar para que o inevitável fique mais brando. Caso você decida suportar browsers antigos no seu projeto estes caminhos poderão te ajudar a se manter são sob o controle. Alguns caminhos podem ser duvidosos mas com certeza serão elementos cruciais para salvar sua equipe de algum problema imediato.

Comentários Condicionais

Os Comentários Condicionais são blocos de comentários em HTML criados especificamente para o Internet Explorer. Eles servem para separar um determinado código para que apenas o Internet Explorer consiga entender e renderizar. Isso é útil porque de longe o Internet Explorer é o que mais causa problemas de diagramação.

Os Comentários Condicionais podem ser chaveados para que uma determinada versão do IE. Eu sugiro que se você utilizar os comentários, você tente fazer isso para apenas a versão mais antiga do IE, que no caso é a 6. Se mesmo assim você tiver problemas com outras versões do IE, a minha sugestão é manter um arquivo de CSS só para IE6 e outro para as outras versões. Isso também serve para Javascripts que serão apenas utilizados para suprir necessidades de compatibilidade dos IEs, como por exemplo scripts de PNG ou scripts que prometem melhorar a compatibilidade dos IEs com o CSS.

CSS Hacks

Os CSS Hacks são códigos de CSS que aproveitam um bug de interpretação de um determinado browser para fazer com que este mesmo browser entenda um determinado código CSS. Não é uma boa prática, mas muitas vezes não é possível evitá-la.
De ponto de vista de sintaxe, o CSS Hack é errado. Veja um exemplo: quando você coloca um underline antes da propriedade do CSS, como por exemplo _width: 200px; essa propriedade só é reconhecida pelo IE6. Isso acontece porque a maneira é errado escrever a propriedade do CSS com um underline na frente, por isso os outros browsers ignoram esse código, e então você tem a vantagem de poder escrever uma propriedade de CSS que será interpretado apenas no IE6. Se alguma coisa no layout estiver errada no IE6, eu posso resolver assim.

O problema é que isso não pode ser comum no código de CSS. Aí então é preferível que você utilize comentários condicionais, separando um arquivo de CSS só para o IE6 para evitar o uso de Hacks. Abaixo segue três exemplos de hacks para os IEs:

Versão do IE	***Exemplo de Hack***
6	_width: 200px;
7	*width: 200px;
8	width: 200px\0/;

Lembre-se: use com moderação.

CSS Hacks é uma praga se não for utilizado com inteligência e em lugares muito pontuais. Não vale a pena você fazer Comentários Condicionais se você for apenas para acertar um erro de alguns linhas, nesse caso o CSS Hack se encaixa muito bem. Mas só para casos pequenos desse tipo.

Javascript pode ajudar

O Javascript tem ganhado muita popularidade entre os programadores. É uma linguagem muito versátil e está presente desde o início da web. Ela fecha o trio das três camadas onde o HTML cuida da semântica e exibição da informação, o CSS cuida do visual e apresentação dessa informação nos diversos meios de acesso e o Javascript controla o comportamento dos elementos manipulando suas características via CSS.

Com os frameworks como JQuery é fácil controlar o comportamento dos elementos. Para os desenvolvedores client-side saber pelo um framework de Javascript é essencial. Se você é programador você precisa aprender a linguagem. Frameworks de Javascript servem para facilitar o trabalho de manipulação de comportamento dos elementos.

O Javascript pode ajudar muito na criação e estruturação de um website. Normalmente, quando queremos manter a compatibilidade e visual fiel em todos os browsers, é necessário criar alguns elementos extras, por exemplo, se quisermos fazer um box com largura e altura flexivel com os quatro cantos arredondados, precisamos inserir no HTML 4 elementtos que não tem significado semântico nenhum. Estes elementos não trarão informação nenhuma para o usuário final. Serão apenas úteis para criar cantos arredondados. Logo, não faz sentido colocarmos esse código direto no HTML porque pode causar problemas de acessibilidade e prejudicar a indexação dos buscadores. Por isso colocamos estes elementos dinamicamente via Javascript. Por isso o Javascript é um aliado forte para o desenvolvedor client-side.

Além disso existem outros plugins que podem ajudar na adequação visual do site. Infelizmente quanto mais específico o plugin, pior é o seu comportamento nos browsers antigos. Eu sugiro sempre tentar criar scripts para utilidades simples. Sem inventar muito.

Não customize elementos restritos

Chamo de elementos restritos objetos como Combo box (ou selectbox), checkbox e radio box. Eu não sei quem falou para os designers de website que eles poderiam fazer uma arte bacana no photoshop para estes elementos... A customização destes elementos é um problema. Não há maneira fácil hoje me dia para fazermos isso. O único lugar que permite um nível interessante de customização destes elementos via CSS é no Mobile Safari presente no iPhone, iPod e iPad. Fora isso a personalização é muito restrita.

Conseguimos customizar campos de formulários e textareas e mesmo assim com bastante custo e bugs nos Internet Explorers.
Sempre que estiver envolvido com o início do projeto, procure estes elementos customizados e tente falar com o responsável da equipe para manter o visual padrão.

Outro problema para customizar estes elementos é a acessibilidade a usabilidade. Os usuários estão acostumados com os elementos do sistema. Customizá-los pode fazer com que o usuário se perca ou fique em dúvida sobre o elemento.

Mire os motores de renderização e não os browsers

Os browsers são muitos. Você tem o Firefox, Safari, Opera, Chrome, Internet Explorer.
Se formos extrapolar as versões, só de Internet Explorer você precisa se preocupar hoje com 3: IE6, IE7 e IE8. Futuramente, se continuar do jeito que está IE9. O Firefox quer ter até o final de 2011 a versão 7 do seu browser.

Há também os browsers para mobiles: Browser do Android, Mobile Safari do iPhone, o browser do BlackBerry e também dos celulares Nokia. Para quase todas essas plataformas tem o Firefox e alguns outros browsers que prometem uma vida melhor.

Realmente são muitos browsers. Eu não gosto de focar nos browsers em si, prefiro me focar nos motores de renderização. Os browsers tem um motor (engine) que leem o HTML/CSS/Javascript e montam essa informação na tela do usuário. Segue abaixo uma listagem destes motores:

Browser	***Motor***
Firefox, Mozilla, Flock	Gecko
Opera	Presto
Internet Explorer	Trident
Safari, Chrome, Mobile Safari e browser para Android, novos browsers da BlackBerry e alguns outros da Nokia	Webkit

Veja que listamos diversos navegadores envolvidos em apenas 4 motores de renderização. Normalmente quando vários browsers são feitos com um mesmo engine, eles tem comportamento praticamento iguais. Ou seja, 90% das coisas que você fizer com CSS e ver no Safari para Desktop, vai funcionar no iPad, iPhone, iPod, Blackberry (só os novos) e alguns Nokia. Com uma pancada, você resolveu vários problemas. Isso é interessante já que a tendência é sempre aumentar o número de browsers. Quanto mais um número de browsers se aglomerar de baixo de uma mesma engine, melhor para nós.
Abaixo segue algum comentário sobre cada um dos engines mais populares.

Gecko

Motor com código aberto. É utilizado nas aplicações da Mozilla: SeaMonkey, Camino, Firefox, Thunderbird etc.
Gecko é um motor herdado do antigo Netscape, baseado no Mosaic.
Depois da Guerra dos Browsers, a Netscape doou o motor de renderização para a comunidade, que culminou na criação da Mozilla.

Presto

Motor proprietário da Opera Software. A Opera é uma das empresas que mais inovam no mercado de browsers. Embora eles tenham tecnicamente um dos melhores browsers para desktops, a versão mobile é a mais utilizada. Eles tem duas versões: Opera Mobile, para smartphones e Opera Mini, para celulares mais básicos. A Opera também está muito presente em outros mercados fora da web.

Webkit

Motor com código aberto, é utilizado hoje em aplicações como Safari, Safari Mobile e Chrome, browser do Google. A BlackBerry[10] já anunciou que vai fazer seus browsers com Webkit.
É o mais novo motor de renderização do mercado. Foi criado pela Apple, baseando-se no motor de renderização KHTML, que estava só presente em browsers para Linux, como o Konqueror. Aproveitando

[10] http://migre.me/3ZpvQ

que o KHTML é um sistema OpenSource, a Apple modificou todo o seu código, fazendo melhorias e aperfeiçoando-o para criar seu browser o Safari.
A Apple fez várias outras modificações posteriores em cima desta primeira versão. Deu o nome de Webkit, e hoje, conduz o desenvolvimento dessa plataforma.

Trident

É o motor proprietário da Microsoft. É utilizado em aplicações como Outlook e claro, no Internet Explorer. Eles estão criando um novo motor, que foi utilizado no Internet Explorer 8 e será usado em versões posteriores.
Embora o Trident fora o primeiro a suportar completamente o CSS 1.0, atualmente ele é o motor de renderização mais atrasado. A Microsoft vem fazendo um bom trabalho para tentar recuperar essa má fama, mas mesmo assim, os outros motores do mercado estão muito além.

Num futuro próximo a diferença entre um ou outro browser não será apenas a capacidade do seu engine. Todos estão caminhando para serem complacentes com os padrões e isso um dia deixará de ser problema. Bom para gente, bom para web.

Entenda que sempre haverá aqueles que não utilizam browsers modernos. Eles tem os motivos deles e estes motivos podem variar. A pessoa pode não entender nada de tecnologia. Ou pode trabalhar em uma grande empresa onde a atualização de software é feita de tempos em tempos e a atualização autonoma não e permitida. Ou por que ela não tem dinheiro para fazer o upgrade do seu computador e utiliza uma versão antiga do sistema operacional. Há um entrave tecnologico aí. A atualização do browser vai acontecer vagarosamente para os grupos citados acima. Não temos controle sobre isso. Mesmo assim não precisamos nos preocupar em entregar para um browser antigo 100% de compatibilidade. Isso é impraticavel. Mas não podemos largá-los na mão, deixando-os com um site quebrado, que não funciona como deveria, isso seria desrespeito.
A melhor saída é entregar a melhor experiência que estes usuários podem ter sem prejudicar o trabalho da equipe de produção.
Os usuários crísticos não vão ligar se você entregar o site para eles com bordas quadradas em vez de arredondadas. Para o público com browsers modernos, as bordas arredondadas serão bem-vindas, mas para o usuário que utiliza um browser que não reconhece a propriedade de CSS que cria as bordas arredondadas automaticamente, entregaremos bordas quadradas.
O site vai ficar um pouco diferente do que o planejado pelo designer, mas valerá a pena. As bordas quadradas será entregue para uma pequena parcela dos usuários e você economizará tempo e dinheiro. Estou dando o exemplo da borda arredondada porque é um problema simples e recorrente em quase todo projeto web.

Abaixo mostro algumas saídas para que você consiga garantir a melhor experiência para todos os usuários de acordo com as limitações de cada software.

Gracefull Degradation e Progressive Enhancement

Gracefull Degradation[11] é uma maneira de manter um visual homogêneo entre seus visitantes. É muito fácil entender a técnica: você oferecerá o melhor suporte visual possível para os browsers antigos. Suponha que o designer fez um botão com bordas arredondadas e sombra. Estes botões podem ser de vários tamanhos dependendo do tamanho da palavra e etc. No IE6 você não conseguirá criar um botão assim apenas com CSS. Logo, alguns fatores ficarão de fora como a sombra e a borda arredondada. O botão ainda vai existir, ainda vi funcionar, só não vai ficar bonitão como quando visto no Safari, Firefox e etc...

Alguns desenvolvedores discutem até hoje se essa técnica é realmente válida. Muitos acham que isso é privar demais o usuário, obrigando-o a fazer um upgrade no browser, que as vezes, como já dito no começo do capítulo, pode ser impossível em alguns cenários. Outros pela praticidade técnica: você faz seu CSS ou JQuery funcionar perfeitamente em browsers modernos e espera que em browsers antigos ignorem simplesmente os códigos que não entendem. Mas e os códigos que eles entendem pela metade?
Outro problema do Gracefull Degradation é que o usuário de browsers antigos baixam códigos que seus browsers nunca usarão. Ou seja, de qualquer maneira, mesmo não vendo a sombra do botão, ele vai baixar o código que faz a sombra funcionar em outros browsers. Ele não vê a sombra porque o browser dele não entende o código de sombra, por isso ignora. Este é um dos motivos para que o Progressive Enhancement ganha força em detrimento do Gracefull Degradation.

[11] http://en.wikipedia.org/wiki/Graceful_degradation

O Progressive Enhacement[12] ou simplesmente PE faz o sentido contrário. Em vez de você colocar todas as coisas lindas que você quer e depois pensa em como retirar isso caso dê defeito nos browsers antigos, você primeiro faz funcionar de maneira muito simples nos browsers antigos e depois começa a detalhar tanto o visual quanto o funcional para que os browsers modernos funcionem melhor.

O Alysson Franklin, um dos editores do Tableless.com.br detalhou essas duas técnicas em um artigo muito bem escrito chamado **Bem-vindo a Xangri-lá** dividido em 2 partes:
Parte 1 - http://www.tableless.com.br/bem-vindo-a-xangrila-parte-1
Parte 2 - http://www.tableless.com.br/bem-vindo-a-xangrila-parte-2

O Gracefull Degradation não é invalidade pelo Progressive Enhancement. Os dois andam juntos e o Progressive Enhancement é totalmente inspirado no Gracefull Degradation. Arrisco dizer que ele e o Gracefull Degradation aprimorado.

Prefixos dos browsers

Muitas das características do CSS, principalmente da sua versão 3 ainda estão em fase de adquação e testes, portanto elas não foram implementadas definitivamente em alguns browsers e para evitar conflitos e também conseguir informações de feedback sobre o funcionamento dessas propriedades para fazer futuras adaptações ou correções. Outro ponto importante, é que algum destes fabricantes de browsers podem querer suportar uma determinada propriedade que ainda não faz parte do core do CSS mas que poderia ser muito útil para você utilizar em lugares específicos hoje, com muita cautela, claro.

A tabela abaixo nos mostra os prefixos dos principais browsers do mercado. Durante toda minha experiência com Web, utilizei muitas vezes os prefixos do Firefox, Safari e Opera, quase nunca utilizei o prefixo para Internet Explorer. Talvez, com a vinda do IE9 passemos a utilizar mais o prefixo da Microsoft para implementar propriedades que eles estejam planejando incluir nas versões posteriores do IE, mas não tenho tantas esperanças. A Microsoft precisa se esforçar muito para recuperar a confiança dos desenvolvedores.

Browser	*Prefixo*
Opera	-o-
Safari	-webkit-
Firefox	-moz-
Chrome	-chrome-
Internet Explorer	-ms-

Tenha em mente que os prefixos não são Hacks de CSS. Nem se comparam.
Com os prefixos você está ajudando os fabricantes e o W3C a entenderem melhor novas propriedades.
Concordo com você que o código não fica muito bonito de se ver e que pode causar muito transtorno quando mal organizado. Diferentemente dos css-hacks os prefixos fazem parte dos padrões web. Os css-hacks exploram uma falha/bug do browser criar um código que apenas aquele browser leia ou ignore. Normalmente para isso usamos a sintaxe do CSS de forma errada como: w\idth:200px; ou _width:200px;.

---

[12] http://www.alistapart.com/articles/understandingprogressiveenhancement/

Fazendo isso estamos colocando o projeto em risco futuramente. Os browsers passarão a ignorar esse código e se seu CSS estiver baseado nele, provavelmente algo vai quebrar. Eu costumo sugerir esse tipo de hack apenas para versões muito antigas do IE como a versão 6, que é uma versão que no cenário atual oferece muitos problemas de compatibilidade.

Conclusão

Realmente a ideia deste documento não é degradar nenhum tipo de browser ou bater de frente com equipes que optam em manter a compatibilidade nivelando por baixo. Não é esse o intuito. Quero que você desenvolva projetos mais elaborados. Quero que você não gaste tempo/dinheiro a toa se você pode ter resultados melhores se mudar um pouco o foco do desenvolvimento.

Lembre-se de que na web você não pensa no passado, mas no futuro.

Links e fontes importantes

http://www.alistapart.com/articles/beyonddoctype
http://www.zeldman.com/2009/05/21/a-new-answer-to-the-ie6-question/
http://en.wikipedia.org/wiki/List_of_web_browsers
http://en.wikipedia.org/wiki/History_of_the_World_Wide_Web
http://www.livinginternet.com/w/wi_browse.htm
http://en.wikipedia.org/wiki/Mosaic_(web_browser)
http://www.livinginternet.com/w/wi_mosaic.htm