Fonte: <http://akitaonrails.com/2011/04/19/off-topic-adendo-a-controversia-haml-sass-coffeescript-tamanho-do-documento>

Nos últimos artigos eu descrevi algumas controvérsias que rolaram pelo comunidade recentemente, primeiro a decisão do [Twitter de trocar Ruby por Java](http://akitaonrails.com/2011/04/16/twitter-muda-de-ruby-para-java-ruby-e-3x-mais-lento-que-java), depois [a decisão do Rails Core Team de adotar SASS e Coffeescript](http://akitaonrails.com/2011/04/16/a-controversia-coffeescript), finalmente [a discussão entre Rspec e Test::Unit](http://akitaonrails.com/2011/04/17/a-controversia-test-unit-vs-rspec-cucumber).

Em todos esses casos existe ainda outras variáveis a considerar. Primeiro, as discussões são de desenvolvedor para outro desenvolvedor. Na maioria dos casos vamos cair em estética, o que é basicamente questão de gosto. Alguns vão dizer que Less é melhor que Sass, quem gosta de Sass vai dizer que é melhor do que Less. Quem gosta de ERB vai dizer que não gosta de HAML, quem já se acostumou com HAML vai dizer que ERB é feio e assim por diante. E agora o ponto crucial: nenhum dos dois está nem certo nem errado.

Sei que essa resposta não ajuda mas quem disse que a vida é fácil? Quero discutir dois argumentos para essas discussões todas, a primeira mais conceitual e filosófica (vocês me conhecem) e a segunda mais comercial e de negócio.

A Explicação Filosófica

Eu já discuti sobre isso num artigo de 2006 entitulado [Evolução pela Concorrência](http://akitaonrails.com/2006/09/27/evolu%C3%A7%C3%A3o-pela-concorr%C3%AAncia). É essencialmente a mesma coisa. Entre Java e C#, por exemplo, na menor da hipóteses existirão pelo menos duas pessoas que acreditam que o seu é melhor: os criadores! E quem pode lhes dizer que eles estão errados? Nenhum dos dois precisa discutir abertamente, mas se eles se mantém nas suas criações originais indica que cada um acredita que a sua solução ainda tem mais vantagens do que a outra. Entre um Gosling e um Hejlsberg quem está certo?

Daí começam pelo menos três tipos de comparação: performance (sempre!), produtividade e suporte (quantos desenvolvedores sabem mais um ou mais o outro). Cansei de ver [comparações como esta](http://java.dzone.com/news/productivity-race-ruby-rails-v) tentando mostrar quem desenvolve mais rápido: um cara usando tecnologias Rails ou outro usando Java. E alguns usam isso como “prova” de que um argumento está certo ou errado. Isso demonstra, pra variar, que as pessoas não sabem a diferença entre **“prova”** e **“evidência”** e que não entendem o método científico, como eu já expliquei num artigo de 2008 entitulado [Método Científico vs Cargo Cult](http://akitaonrails.com/2008/12/16/off-topic-m-todo-cient-fico-vs-cargo-cult). Via de regra, duvide de qualquer um que diz ter “provas” ou que “provou” alguma coisa. Normalmente ele apenas encontrou uma “evidência favorável”, que de jeito nenhum significa “prova”.

Agora, para um estar certo, o outro precisa estar necessariamente errado? Isso me leva a outro artigo que escrevi em 2007, entitulado [Para eu ganhar, o outro precisa perder …](http://akitaonrails.com/2007/10/23/para-eu-ganhar-o-outro-precisa-perder). Isso é uma grande bobagem. Nenhum problema defender porque você gosta de alguma coisa, mas isso não significa que o outro não tenha valor ou que deveria desaparecer. Ruby não é melhor do que C, ou do que Lisp, ou do que Python, ou do que Perl. Pelo contrário, Ruby só é bom porque Lisp, C, Perl e Python vieram antes dele mostrando possibilidades que o Matz, sozinho, provavelmente não entenderia. E o Matz entende isso, justamente por isso você não o vê evangelizando Ruby às custas de denegrir os outros.

**Cuidado:** não estou fazendo apologia à ficar em cima do muro, em ser conivente, conformista, ou algo assim. Quando você já sabe que um lado é errado, não há porque ficar no meio. Releia meu artigo de 2009 entitulado [O Culto da Moral Cinzenta](http://akitaonrails.com/2009/09/08/off-topic-o-culto-da-moral-cinzenta). Uma coisa é você discutir entre comida e veneno, o meio do caminho é que o veneno sempre ganha. O que estou descrevendo são discussões do tipo, qual comida é melhor: maçã ou laranja?



Eu não sou psicólogo portanto não vou dizer que a seguinte conclusão é psicologicamente comprovada, mas o que eu consigo [inferir](http://en.wikipedia.org/wiki/Inference) baseado em inúmeras discussões desse tipo a respeito é que todos eles se parecem com a milenar questão do **tamanho do seu documento**. Todo macho-alfa parece ter uma necessidade fisiológica de “medir” para se auto-afirmar: *“ah, o meu é maior”* – e então ele suspira em silêncio – *“ufa!”*. E vou lhes dizer mais: eu sou japonês, então vocês imaginam desde minha infância quanto já não tive que ouvir as piadas infames de quem se acha engraçado: *“ah, sabe aquela piada do eletricista japonês?”*

Quem quer sempre medir e quem sempre acha que tem que comparar pra descobrir qual é o maior denota aquilo que a maioria das mulheres já sabe: *“esse homem é muito inseguro, praticamente uma criança.”* Ou seja, são pessoas que tem pouca fundação e pouca segurança em si mesmo e precisa se agarrar em resquícios. Resumindo: discussões de *“o meu é maior que o seu, e por isso o seu é necessariamente inferior, pior e desnecessário”* é discussão de quem tem insegurança em si mesmo.

É por isso que tantas pessoas tem tanta dificuldade em aprender coisas novas ou diferentes: preconceito e insegurança. E às vezes entender que mesmo que você goste infinitamente de maçã, comer uma laranja de vez em quando não tem nenhum prejuízo. Ninguém está falando para trocar a comida por veneno, ou ficar no meio do muro e misturar veneno na comida.

E como sempre, eu também entro nesse tipo de discussão. Às vezes é simplesmente divertido. E também é uma reação natural. Não imaginem que por causa desse “sermão”, quer dizer que eu nunca faço.

A Explicação Comercial

Eu sou um consultor, sempre fui e mesmo quando era funcionário CLT com cargo e tudo, nunca deixei de pensar como consultor. Um consultor tem como tarefa ser um **provedor de solução**. Agora, uma solução é, por definição, uma resposta a um problema ou uma forma ou processo para se chegar à resolução de um problema.

Claro, para o problema *“quero um site”* existem centenas de maneiras diferentes de fazer. Um consultor precisa avaliar o passado, o presente e até o possível futuro de uma solução, as circunstâncias internas e externas. Tem muito mais do que simplesmente escolher tecnologias e fazer código.

E aqui entra meu adendo especificamente ao caso de HAML, SASS, CoffeeScript ou qualquer outro tipo de pré-processador que serve de intermediário para gerar as estruturas originais como HTML,CSS e Javascript, no caso.

A argumentação é muito simples: se um determinado aplicativo for ser mantido por você mesmo, o desenvolvedor escolhendo tais ferramentas, ou então será uma aplicação que dificilmente terá qualquer modificação (veja aqui, você especulando sobre o futuro!), ou então será uma aplicação que vai morrer rápido (por exemplo, hotsite de alguma campanha de marketing), nesse caso você praticamente tem passe livre para usar quaisquer ferramentas e métodos para desenvolver – salvo, claro, que existam requerimentos ou motivos muito fortes para limitar isso (digamos, política da empresa do cliente).

Agora, se já sabemos que o produto será desenvolvido por nós e depois será entregue ao cliente porque ele quer dar manutenção ele mesmo, ou porque ele prefere que outra empresa dê manutenção ou qualquer caso desses, agora temos um problema. Escolher usar no back-end Ruby, Python, Java, C#, cada um tem prós e contras e sempre há bons argumentos (que não exigem denegrir as outras alternativas) que facilmente convencem um potencial cliente – e sempre temos que levar em consideração os requerimentos do projeto e do cliente, claro. Já o front-end, coisa que costuma ser fluida e volátil, que precisa acompanhar a demanda dos usuários finais e do mercado, e que muitas vezes tem manutenção separada do back-end, não deveria ser tão complicado de mexer.



Quando já era ASP, PHP, que tinha HTML misturado com código no meio, era ruim, mas a maioria dos desenvolvedores de interface já entende o que ele pode ou não pode fazer. Mas agora, se ele contratar um outro terceiro, e lhe mostrar um site onde o front-end é todo feito numa estrutura que ele nunca viu antes (e tire daqui a discussão se ele deveria ou não saber), é claro que ele vai reclamar, surtar, xingar. Especialmente se, como nós, ele também fechou um projeto de tempo fechado com o cliente e recebe um material todo em HAML e SASS. Nós podemos argumentar que todo mundo deveria ser capaz de aprender e tudo mais. Mas não diga isso a alguém que acabou de fechar um projeto de custo fechado e recebe uma surpresa.

Pior, o cliente pode retornar a você exigindo que você dê suporte já que agora você limitou as escolhas que ele teria de contratação de fornecedores. Fornecedores podem usar isso contra você de propósito, dependendo do caso. Por isso mesmo eu sempre defino no contrato o que eu vou usar e o que vou entregar.

Não é o fim do mundo. Se fosse assim estaríamos todos escrevendo CGI em C até hoje. É arriscando e experimentando que passamos a usar Perl, depois PHP, ou ASP, depois Java, depois C# até chegarmos ao Ruby, por causa do Rails, hoje em dia. Mas isso não acontece do dia pra noite e não acontece sendo intransigente.

**Cuidado:** novamente, não estou fazendo apologia à baixa qualidade, ou defendendo quem não quer aprender coisas novas. Não é uma questão do tipo: devo fazer testes ou não? devo fazer código macarrônico ou não? Muito pelo contrário, numa conversa de desenvolvedor para desenvolvedor, eu sempre vou defender coisas novas, evoluções e tudo mais. Mas quando a conversa é sobre clientes e projetos que não são meus, os critérios são mais amplos, incluindo oportunidade de novos negócios, e o próprio timing para se oferecer certas coisas ou não.



Por exemplo, com o movimento de colocar o CoffeeScript dentro do Rails, automaticamente o CoffeeScript ganha uma exposição a um mercado maior que ele não tinha e talvez nem tivesse antes. Com isso mais e mais pessoas até de fora do Rails vão começar a adotá-lo e possivelmente um nicho considerável de desenvolvedores já estejam acostumados e muitos clientes até peçam para usar CoffeeScript de livre e espontânea vontade.

É um dilema que eu, como evangelizador de Ruby por exemplo, preciso considerar todos os dias: quando é bom dar uma forçadinha para que o cliente use Rails ou quando é melhor considerar que o custo-benefício não compensa ou que o requerimento é incompatível com Rails e recomendar que ele use Wordpress ou outra coisa?

Eu já fiz isso, várias vezes, em vez de recomendar um projeto em Rails eu já recomendei – e escrevi propostas e dei suporte de pré-vendas – em projetos de PHP, de Java, de .NET. E não faço cara feia.



Uma história que eu sempre uso é que uma vez eu fui mandado para um cliente, para dar manutenção pontual num pequeno sistema que era feito em ASP. Isso acho que era 2003, o site – em bom francês – era uma porcaria. E não era por causa do ASP, mas porque o código em si era horrível, todo bugado, altamente acoplado e misturado, sem organização nenhuma, lotado de arquivos asp no mesmo diretório que sequer eram usados, coisas do tipo “config.asp”, “config2.asp”, “config\_novo.asp”, “config\_old.asp” (e o correto era o “config\_old.asp”, pasmem! ) e assim sucessivamente. Eu poderia xingar, poderia querer mudar tudo, poderia querer reescrever ou então simplesmente decidir não fazer.

Em vez disso eu tinha acho que 4 dias ou algo assim. Eu olhei, entendi como o fluxo que eu precisava mexer funcionava, considerei o tempo limitado e o que poderia ser feito nesse período, fiz o essencial que cabia e que atendia o que o cliente pediu, e entreguei. O cliente ficou muito satisfeito, no final eu até corrigi um pouco mais do que ele esperava – na verdade ele estava com baixas esperanças porque outros foram e não conseguiram consertar. E meses depois fui chamado de novo para outro projeto, desta vez de SAP com Java, onde gerenciei uma equipe maior e fizemos um projeto que durou quase 2 anos. Lição aprendida: não é fechando a porta que se consegue as coisas e nem é sendo intransigente e imediatista. E a lição principal: não importa se por baixo você fez um algoritmo digno de Donald Knuth, se o cliente não ficou satisfeito, seu sistema é uma droga, por definição.



E como eu sempre digo: não estou dizendo que não é para discutir. Muito pelo contrário, é bom que existam discussões, questionamentos, críticas, pois sem isso não há evolução e nenhum lado ganha. Estou só avisando que – a menos que você tenha certeza que do outro lado é veneno, e saiba demonstrar isso de forma objetiva – então é melhor se segurar um pouco ou a discussão só tem um lugar para acabar: em um tentando medir o documento do outro.

Se lembrem de outra coisa: eu linkei artigos de 2006, 2007, 2008 e 2009 lá em cima, onde já discuti muito do que falei aqui. Vocês acham que se desde 2006 minha posição fosse: *“Use Ruby porque o resto é uma porcaria”* será que eu conseguiria fazer o que fiz até agora?