

# O Programador Apaixonado

Construindo uma carreira notável em desenvolvimento de software



© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

# Sumário

1	Liderar ou sangrar?	1
2	Oferta e demanda	5
3	Escrever código não é suficiente	9
4	Seja o pior	13
5	Invista em sua inteligência	17
6	Não escute seus pais	21
7	Seja generalista	27
8	Seja especialista	33
9	Não coloque todos os seus ovos num só cesto	37
10	Ame-o ou deixe-o	41
11	Aprenda a pescar	47
12	Aprenda como os negócios realmente funcionam	51
13	Encontre um mentor	55
14	Seja um mentor	59
15	Pratique, pratique, pratique	63

16	O jeito que você faz	69
17	Nos ombros dos gigantes	73
18	Automatize-se em um emprego	77
19	Agora mesmo	83
20	Leitor de mentes	87
21	Êxito diário	91
22	Lembre-se de para quem você trabalha	95
23	Esteja onde você está	99
24	Quão bom eu posso fazer um trabalho hoje?	103
25	Quanto você vale?	107
26	Uma pedrinha em um balde d'água	111
27	Aprenda a amar manutenção	115
28	Maratona de oito horas	121
29	Aprenda a falhar	125
30	Diga "Não"	129
31	Não entre em pânico	133
32	Diga, faça, mostre	137
33	Percepções e perssepisões	145
34	Guia de aventura	149
35	Eu iscrevu mto beim	153

36	Estar presente	157
37	Fale com propriedade	163
38	Mude o mundo	167
39	Faça sua voz ser ouvida	169
40	Construa sua marca	173
41	Lance seu código	177
42	Seja marcante	181
43	Fazendo o gancho	185
44	Já obsoleto	191
45	Você já perdeu seu emprego	195
46	Caminhe sem destino	197
47	Trace um roteiro	199
48	Observe o mercado	203
49	Aquele gordo no espelho	205
50	A armadilha de macaco da Índia do Sul	209
51	Evite planejamento de carreira do modelo cascata	213
52	Melhor que ontem	217
53	Seja independente	221
54	Divirta-se	225
55	Nota da editora	227

**Bibliografia**

**229**

## CAPÍTULO 1

# Liderar ou sangrar?

Se você vai investir seu dinheiro, muitas opções estão disponíveis. Você pode investilo em uma poupança, apesar de que o rendimento possivelmente não seja tão vantajoso se comparado com a inflação. Ou você pode investi-lo em títulos do governo. Novamente, você pode não ganhar tanto dinheiro, mas são apostas seguras.

Ou ainda, você pode investir em uma pequena *startup*. Você pode, por exemplo, investir algum dinheiro em troca de uma pequena parte na sociedade da empresa. Se a ideia da empresa for boa e ela puder executá-la, você pode potencialmente ganhar **bastante** dinheiro. Por outro lado, não é nem garantido que você vá recuperar seu investimento.

Esse conceito não é novo. Você começa a aprendê-lo como uma criança brincando. **Se eu correr por aqui, eu vou surpreender todo mundo e não vou ser pego.** Você é lembrado disso com bastante frequência no seu cotidiano. Você passa pelo *trade-off* risco-recompensa quando está atrasado para uma reunião e precisa escolher o melhor caminho para o trabalho. **Se o trânsito não estiver ruim, eu posso chegar lá 15 minutos mais rápido se eu for por esse caminho. Mas se tiver trânsito,**

**estou ferrado.**

O trade-off risco-recompensa é uma parte importante do processo de fazer escolhas intencionais sobre em quais tecnologias e domínios investir. Quinze anos atrás, uma escolha de baixo risco seria aprender a programar em COBOL. Claro, também havia tantos programadores COBOL com quem concorrer, que o salário médio nessa época já não era fenomenal. Você podia facilmente encontrar trabalho, mas não seria algo tão lucrativo. Baixo risco. Baixa recompensa.

Por outro lado, se àquela época você tivesse decidido investigar a nova linguagem de programação da Sun Microsystems, o Java, durante um tempo talvez fosse difícil encontrar emprego em algum lugar que estivesse usando essa linguagem. Quem iria saber se alguém realmente faria algo em Java?

Mas se você estivesse de olho em como estava a indústria naquela época, da forma como a Sun estava, você poderia ter visto algo especial no Java. Poderia ter sentido que aquilo se tornaria grande. Investir naquela tecnologia logo no início faria de você um líder em uma grande e influente tecnologia.

É claro, nesse caso você teria se dado bem. E se tivesse jogado suas cartas corretamente, seu investimento em Java teria sido muito lucrativo. Alto risco. Alta recompensa.

Agora, imagine que também há 15 anos você tenha visto uma demonstração do novo BeOS, da Be. Na época ele era incrível. Ele foi construído para tirar vantagem de múltiplos processadores. Suas capacidades de multimídia eram impressionantes. A plataforma fez bastante barulho na época e os especialistas sentiam que estava ali um novo concorrente fortíssimo no mercado de sistemas operacionais.

Com a nova plataforma, claro, viriam novas maneiras de programar, novas APIs, novos conceitos de interfaces com o usuário. Era muita coisa para se aprender, mas parecia valer bastante a pena. Você poderia ter criado o primeiro cliente de FTP ou o primeiro gerenciador de informações para o BeOS. Quando a Be lançou uma versão compatível com processadores Intel, rumores indicavam que a Apple a compraria a empresa para usar sua tecnologia na próxima geração do sistema operacional Macintosh.

A Apple não comprou a Be. E após algum tempo, ficou claro que a Be não iria conseguir nem sequer um mercado de nicho. O produto simplesmente não pegou. Muitos desenvolvedores que dominaram programação para o BeOS começaram a perceber que aquele investimento não se pagaria no longo prazo. Após algum tempo, a Be foi comprada pela Palm e o sistema operacional foi descontinuado. BeOS era um investimento arriscado, porém atrativo, que não deu retorno a longo prazo. Alto



risco. Nenhuma recompensa.

Até aqui, eu falei sobre a diferença entre escolher tecnologias que ainda são recentes e as que já estão estabelecidas. Escolher uma tecnologia estável e que já está em produção em diversos sistemas é uma escolha mais segura, no entanto, com recompensa potencialmente menor do que uma tecnologia que poucos conhecem e que ninguém sabe como colocar em produção. Mas e as tecnologias que já estão saindo de cena? Aquelas que já estão apenas esperando para serem enterradas?

Quem dará fim nessas tecnologias? Você pode pensar nos últimos poucos programadores que desenvolvem em RPG como sendo aqueles caras de cabelos grisalhos e contando as horas até a aposentadoria, enquanto os jovens desenvolvedores nunca ouviram falar de RPG. Eles estão todos aprendendo Java e .NET. É fácil pensar que a carreira dos últimos sobreviventes de uma velha e decadente tecnologia estão na mesma espiral da morte que a própria tecnologia.

Mas sistemas antigos não morrem simplesmente. Eles são substituídos. Além disso, na maioria dos casos, sistemas caseiros são substituídos em estágios. Nesses estágios, o sistema antigo precisa conversar com o novo. Alguém precisa fazer o novo falar com o antigo e vice-versa. Usualmente, as crianças rebeldes não sabem (ou não querem saber) como fazer o sistema antigo escutar. E nem os mais velhos pré-aposentadoria sabem como fazer as novidades conversarem com sua amada criatura.

*As duas pontas da curva de adoção devem provar serem lucrativas.*

—

Então, há um papel a ser preenchido pelos desenvolvedores: **o asilo da tecnologia**. Ajudar antigos sistemas a morrer confortavelmente e com dignidade é algo que não deve ser subestimado. E lógico, a maioria das pessoas irá pular do barco antes que ele afunde, seja via aposentadoria ou indo para outra tecnologia. Sendo o último que sobrou para manter os sistemas que ainda são críticos, você é o cara. É arriscado, já que, uma vez que a tecnologia já era, você será especialista em algo que não existe. No entanto, se você puder se mover rápido o suficiente, você pode procurar a próxima geração de sistemas que estão morrendo e começar novamente.

A curva de adoção tem altas nas duas pontas. Onde nessas curvas você quer estar?

## Faça algo

- 1) Faça uma lista de tecnologias recentes, médias e antigas baseada no mercado atual. Mapeie-as em uma folha da esquerda para a direita. Na esquerda, coloque as tecnologias recentes, e na direita, as que estão próximas do fim. Force a si mesmo para encontrar a maior quantidade possível de tecnologias. Seja o mais granular possível sobre onde as tecnologias estão quando comparadas umas com as outras.

Quando você tiver mapeado o máximo de tecnologias que conseguiu se lembrar, marque as tecnologias nas quais você se considera forte. Então, talvez em uma cor diferente, marque aquelas sobre as quais você possui algum conhecimento, mas não domina. Onde está a maioria das tecnologias que você marcou? Elas estão aglomeradas em alguma ponta? Estão espalhadas? Elas estão próximas às pontas em que você se interessa?

## CAPÍTULO 2

# Oferta e demanda

Quando a Web realmente começou a decolar, era possível ganhar bastante dinheiro simplesmente criando páginas HTML para empresas. Toda empresa queria uma página web e relativamente poucas pessoas sabiam como fazê-las. Empresas estavam dispostas a pagar bem para web designers “experientes”, o que naquela época significava dizer que conheciam o básico de HTML, criação de links e estrutura de sites.

Fazer HTMLs é muito simples. Difícil é fazer páginas que sejam visualmente bonitas, mas o básico é fácil de atingir. Conforme as pessoas perceberam os valores cobrados por esses web designers, mais e mais pessoas começaram a pegar livros de HTML e aprender a tecnologia por conta própria. O mercado estava quente, os salários ou valores por hora eram atraentes, e como resposta, a oferta de especialistas em HTML começou a aumentar.

Conforme o mercado foi inundado de web designers, o pessoal de web começou a ser classificado entre reais artistas, que dominavam a tecnologia e possuíam grande habilidade, e os práticos, que não tinham a mesma qualidade e competência dos reais artistas. Além disso, a concorrência acabou levando à queda dos preços. Como

um resultado dos preços mais baixos, mais empresas estavam dispostas a dar seu primeiro passo em direção à presença na internet. Eles não pagariam \$5.000 por seu primeiro site, mas sim \$500.

Claro, algumas empresas ainda estavam dispostas a pagar bastante dinheiro por um site “fantástico”. E certamente os web designers ainda conseguiam cobrar valores “fantásticos”.

Em um determinado momento, a inundação de web designers de médio e baixo custo regrediu. Web designers menos talentosos foram substituídos por usuários finais e outro pessoal de TI que não necessariamente dominava HTML. Nesse ponto, a oferta, demanda, e o preço de desenvolvimento chegaram a um equilíbrio.

Essa história sobre a evolução do web design mostra um modelo econômico de que todos nós já ouvimos falar, chamado **oferta e demanda**. Quando a maioria de nós pensa em oferta e demanda, achamos que tem a ver com o preço com que algo pode e será vendido. Se houver mais de um item à venda do que o número de pessoas querendo comprar aquele produto, então seu preço vai diminuir. Se houver mais pessoas querendo comprar o produto do que a quantidade de produtos disponíveis, o preço irá aumentar, de forma que os compradores concorram entre si.

Além de prever o preço de bens e serviços, o modelo de oferta e demanda pode prever como as mudanças de preço irão afetar os que querem comprar e vender os produtos e serviços. Geralmente há mais compradores para um produto barato do que para um mais caro.

*Você não pode competir em preço. Na verdade, você não consegue competir em preço.*

—

Por que isso é importante para nós? A tendência de fazer software *offshore* acaba de injetar uma grande oferta de pessoas de TI de baixo custo em nossa economia. Embora estejamos preocupados em perder empregos, o custo mais baixo por programador na verdade aumentou a demanda. Ao mesmo tempo, conforme a demanda aumenta, os preços diminuem. Concorrência em produtos e serviços com alta demanda acaba por movimentar também os preços. No mercado de empregos, isso significa salários. Você não pode competir em preço. Você não consegue. Então, o que fazer?

O mercado de desenvolvimento *offshore* injetou seus programadores de baixo custo em um conjunto relativamente pequeno de tecnologias. Programadores Java e .NET são baratos e existem aos montes na Índia, junto dos DBAs Oracle. Tecnologias menos populares são pouco adotadas pelas empresas de offshore. Durante a escolha

de um conjunto de tecnologias para focar em sua carreira, você deve entender os efeitos da oferta crescente e menores preços.

Como um desenvolvedor .NET, você deve se encontrar concorrendo com mais dezenas de milhares de pessoas do que, por exemplo, um desenvolvedor Python. Isso resultaria no custo médio de um desenvolvedor .NET reduzindo significativamente, possivelmente aumentando a demanda (nesse caso, criando mais empregos em .NET). Com isso, você encontraria emprego, mas que não pagaria tão bem. A oferta de programadores Python deve ser bem menor que a de .NET para suportar a demanda.

Se o mercado de trabalho em Python estivesse pagando valores mais altos por programador, as pessoas ficariam atraídas por oferecer seus serviços por esses valores, resultando em concorrência, o que levaria os preços para baixo novamente.

É tudo uma questão de balanceamento. Mas uma coisa parece certa (por enquanto). A Índia provê programadores para os mercados de TI que já estão balanceados. Você não encontra empresas de offshore indianas e famosas utilizando tecnologias não convencionais. Eles não são pioneiros. Eles geralmente não se arriscam. Eles esperam o mercado balancear para então entrarem nesse mercado com valores por programador significativamente mais baratos.

Com base nessa observação, você deve escolher competir em segmentos do mercado em que há menor demanda. Por mais contraintuitivo que possa parecer, se você está preocupado em perder seu emprego para o offshore, uma estratégia seria evitar os tipos de trabalho que as empresas de offshore estão fazendo. Elas estão fazendo trabalhos com alta demanda. Então, focar em tecnologias de nicho é uma estratégia que, embora não necessariamente torne a competição menos feroz (há menos empregos para ir atrás), muda o foco da concorrência de preço para habilidade. É isso o que você precisa. Você não pode competir em preço, mas você **pode** competir em habilidade.

Além disso, com o preço médio dos programadores de tecnologias *mainstream* em queda, a demanda irá crescer. Um aumento geral nessa demanda por programadores Java, por exemplo, pode resultar em mais empregos no seu país, não menos. Um aumento no mercado de offshore, que é barato, pode influenciar na demanda.

Isso acontece na prática. Para que o offshore funcione bem, muitas empresas percebem a necessidade de manter uma quantidade de desenvolvedores de alto nível, que definam os padrões da empresa, garantam a qualidade e provenham liderança tecnológica. Um aumento na demanda por programadores Java iria naturalmente elevar demanda nessa categoria de trabalho. Os trabalhos mais baratos podem estar

indo para o offshore, mas há mais trabalhos de alto nível, ou de elite, do que havia antes da época do offshore. Como vimos nos mercados de nicho, a competição mudaria de preço para habilidade.

*Explore mercados não balanceados.*

—

A lição mais importante que podemos levar do modelo de oferta e demanda é que com o aumento da demanda vem o aumento da concorrência de preço. Seguir esse caminho o levará a concorrer em preço com desenvolvedores offshore, pois suas habilidades irão se encaixar nos mercados balanceados em que o offshore atua. Para competir em tecnologias mainstream, você terá que enfrentar um nível mais alto. Alternativamente, você pode explorar mercados não balanceados, aonde as empresas de offshore não vão. Em ambos os casos é imprescindível entender as forças que estão em jogo e ser habilidoso e ágil o suficiente para reagir a elas.

## **Faça algo**

- 1) Pesquise a demanda por habilidades técnicas. Use sites de vagas de trabalho e carreiras para encontrar quais habilidades estão em alta e em baixa demanda. Ache os sites de empresas de offshore (ou fale com funcionários dessas empresas). Compare as habilidades usadas nessas empresas com a lista que tecnologias em alta demanda que você fez. Anote as tecnologias às quais você vê que há uma boa demanda, mas que empresas de offshore não usam.

Faça uma comparação similar entre tecnologias recentes e as habilidades pedidas por empresas de offshore. Fique de olho em ambos os conjuntos de habilidades técnicas que não são usadas pelas empresas de offshore. Quanto tempo levará para que elas preencham esse buraco? Esse tempo é a janela na qual há um mercado não balanceado.

### CAPÍTULO 3

# Escrever código não é suficiente

Não é suficiente pensar em quais tecnologias você vai investir. Afinal, a parte de tecnologia é um bem, certo? Você não vai ser capaz de sentar e simplesmente dominar uma linguagem de programação ou um sistema operacional, permitindo que as pessoas de negócios cuidem da parte de negócios. Se tudo o que eles precisavam era de um robô de código, seria fácil contratar alguém de outro país para fazer esse trabalho. Se você quer permanecer relevante, vai ter que ir fundo no domínio do negócio dentro do qual você está.

Na verdade, uma pessoa de software deve compreender um domínio não apenas bem o suficiente para desenvolver software para ele, mas também para se tornar uma de suas referências. Em uma empresa onde trabalhei, vi um excelente exemplo disso. A equipe de administração de banco de dados consistia de pessoas que realmente não estavam interessadas em tecnologia de banco de dados. Quando eu as encontrei pela primeira vez, foi um choque. **Por que essas pessoas trabalham com tecnologia?** Eu me perguntava. Em termos de habilidade técnica, eles não eram bons. Mas este time tinha algo especial. Sendo os guardiões e protetores de dados de nossa empresa, eles

conheciam o domínio do negócio melhor do que quase qualquer analista de negócios que tínhamos. Seus conhecimentos e compreensão do negócio os tornaram figuras importantes nesse mercado. Enquanto nós, geeks, estávamos olhando para eles com desdém, o “negócio” para o qual eles trabalhavam via um enorme valor neles.

Você deve pensar em sua experiência de domínio do negócio como uma parte importante do seu repertório. Se você é um músico, quando adicionar algo ao seu repertório, não significa apenas que você tocou a música uma vez. Significa que você realmente **conhece** a música. A mesma teoria deve ser aplicada à sua experiência de domínio. Por exemplo, depois de ter trabalhado em um projeto no setor de seguros de saúde, isso não garante que você entenda a diferença entre uma transação HIPAA 835 e uma HIPAA EDI 837. É esse tipo de conhecimento que diferencia dois desenvolvedores trabalhando nessa problema.

Você pode ser “apenas um programador”, mas ser capaz de falar com seus clientes do negócio na língua de seu domínio de negócio é uma habilidade única. Imagine o quanto a vida seria mais fácil se todo mundo com que você tivesse que trabalhar realmente entendesse como funciona desenvolvimento de software. Não seria necessário explicar a eles por que é uma má ideia devolver 30000 registros em uma única página em uma aplicação web ou por que eles não compartilham o endereço para seu servidor de desenvolvimento. Esta é a forma como os seus clientes de negócios se sentem em relação a você: **Imagine o quão mais fácil seria trabalhar com esses programadores, se eles entendessem o que eu estava pedindo, sem que eu tivesse que explicar tudo de forma tão burra e tão detalhista!** E, adivinhem? É a empresa que paga o seu salário.

Assim como as tecnologias se destacam, domínios podem ser escolhidos da mesma forma. Java e .NET são agora mainstream no desenvolvimento de software. Se você aprender essas linguagens, poderá concorrer a um emprego em uma das muitas empresas que utilizam estas tecnologias. O mesmo é verdade para os domínios. Ao escolher em qual indústria trabalhar, você deve ter o mesmo cuidado que teve ao decidir quais tecnologias dominar.

*Agora é hora de pensar nos domínios em que você investe seu tempo.*

—

Além da importância que se deve dar ao escolher um domínio ao montar seu portfólio, a empresa e indústria em que você escolher trabalhar se tornam um investimento significativo em si próprio. Se você ainda não tiver parado para pensar em



quais domínios deveria investir, agora é a hora. Cada dia que passa é uma oportunidade perdida. Da mesma forma que deixar suas economias em uma conta poupança de baixo rendimento, enquanto é possível conseguir taxas de juros mais altas, deixar o seu desenvolvimento em negócios parado é uma péssima escolha de investimento.

## **Faça algo**

- 1) Agende um almoço com um empresário. Converse sobre como eles trabalham. Durante a conversa, pergunte a si mesmo o que você teria que mudar ou aprender se quisesse ter o trabalho dele. Pergunte sobre as especificidades do seu trabalho. Converse sobre como tecnologia os ajudam (ou atrapalham). Pense no seu trabalho a partir da perspectiva deles.

Faça isso regularmente.

Isso pode parecer uma ideia estranha ou desconfortável. Tudo bem. Eu comecei a fazer isso há muitos anos e isso fez uma grande diferença na forma como eu entendia e me relacionava com o negócio no qual eu estava trabalhando. Eu também me senti mais confortável falando com os meus clientes, o que é um efeito colateral positivo.

- 2) Pegue uma revista especializada na indústria de sua empresa. Provavelmente você não terá nem sequer que comprá-la. A maioria das empresas tem essas revistas espalhadas em algum lugar. Comece a lê-la. Provavelmente você não irá entender tudo o que lê, mas seja persistente. Faça listas de perguntas que você pode fazer a seus gerentes ou clientes. Mesmo que suas perguntas lhe pareçam estúpidas, seus clientes vão perceber que você está tentando aprender.

Procure sites que você possa monitorar de forma regular. Tanto nos sites como nas revistas, preste atenção ao assunto das grandes notícias e dos artigos. Quais são os problemas que a indústria está tentando resolver? Qual é o novo assunto dominante? Seja o que for, converse sobre isso com seus clientes. Peça para eles lhe explicarem e darem suas opiniões. Pense em como essas tendências atuais afetarão sua empresa, sua área, sua equipe, e eventualmente, o seu trabalho.



## CAPÍTULO 4

# Seja o pior

Pat Metheny, lendário guitarrista de jazz, costuma dar um conselho para jovens músicos: “Sempre seja o pior cara da banda em que você estiver”.

Antes de iniciar minha carreira em TI, eu tocava jazz e saxofone profissionalmente. Como músico, eu tive a sorte de aprender esta lição bem cedo. Ser o pior cara da banda significa sempre tocar com pessoas que são melhores do que você.

*Seja o pior em qualquer grupo em que estiver.*

---

Agora, por que você escolheria ser a pior pessoa em uma banda? “Isso não é irritante?”, você se pergunta. Sim, é extremamente irritante, no começo. Como um jovem músico, eu me via em situações em que ficava tão óbvio que eu era o pior cara da banda, que eu tinha certeza que iria ficar de fora. Eu apareceria para o show mas nem pegaria meu sax, com medo de ser expulso da apresentação. Eu estava perto de pessoas as quais antes eu observava e em cujo nível esperava tocar, às vezes como o instrumento principal.

Sem falhar (felizmente!), algo mágico acontecia: eu me encaixava no grupo. E não me destacava como uma estrela entre os outros músicos. Mas por outro lado, eu não era, também, deixado de lado. Isto acontecia por duas razões. A primeira é que eu não era tão ruim quanto eu pensava. Vou falar disso depois.

A razão mais interessante pela qual eu me encaixava com estes caras tão bons, alguns até meus heróis, é que minha música se parecia com a deles. Eu achava que eu tinha algum tipo de super poder de me transformar em um gênio simplesmente por estar ao lado de algum deles, mas na verdade, é algo bem menos interessante que isso. Era um comportamento instintivo, programado em mim. É o mesmo fenômeno que faz com que eu fale mais difícil, use um diferente vocabulário ou hábitos gramaticais quando estou perto de pessoas que falam de forma diferente. Quando voltamos da Índia, após viver um ano e meio por lá, minha esposa, às vezes, me ouvia falando e morria de rir. “Você ouviu o que você disse?” — eu estava falando inglês indiano.

Ser o pior cara da banda trouxe o mesmo comportamento em mim como um saxofonista. Naturalmente eu tocava como os outros. O que torna esse fenômeno realmente sem glamour é que quando eu tocava em cassinos e bares com bandas não tão boas, eu tocava mal como eles. Além disso, meio que naturalmente eu via os maus hábitos das bandas sendo levados por mim para as noites nas quais eu não tocava com eles.

Com isso, eu aprendi que as pessoas podem melhorar ou piorar em habilidade, apenas levando em conta com quem elas estão trabalhando. E ficar em um grupo por muito tempo pode impactar na capacidade da pessoa.

*As pessoas ao seu redor afetam a sua performance. Escolha bem seu grupo.*

—

Mais tarde, conforme eu me mudei para a indústria de tecnologia, percebi que o hábito de procurar os melhores músicos era natural para mim como um programador. Talvez inconscientemente eu procurei trabalhar com as melhores pessoas de TI. E, não surpreendentemente, essa lição é verdade. Ser o pior cara (ou garota, é claro) na equipe tem o mesmo efeito de ser o pior cara da banda. Você acha que é inexplicavelmente mais inteligente. Você pode até mesmo falar e escrever de forma mais inteligente. Seus códigos ficam mais elegantes e você acredita ser capaz de resolver problemas difíceis com soluções cada vez mais criativas.

Vamos voltar para a primeira razão pela qual eu fui capaz de me integrar a essas bandas melhor do que eu esperava. Eu realmente não era tão ruim quanto eu pensava. Na música, é muito fácil medir se os outros músicos pensam que você é

bom. Se você é bom, eles o convidam para tocar com eles novamente. Se você não for bom, eles o evitam. É uma medida muito mais confiável do que apenas perguntar o que eles pensam, porque bons músicos não gostam de tocar com músicos ruins. Para minha surpresa, eu percebi que, em muitos desses casos, eu era telefonado por um ou mais desses músicos superiores para algum trabalho adicional ou até mesmo para começar bandas com eles.

Tentar ser o pior faz com que você pare de se vender por tão pouco. Você pode pertencer à banda A, mas sempre se coloca na banda B, pois está com medo. Reconhecer abertamente que você não é o melhor tira o medo de você ser descoberto da forma que você não gostaria. Na verdade, mesmo quando você tentar ser o pior, você não será.

## Faça algo

- 1) Encontre uma situação para você ser o pior. Você pode não se dar ao luxo de mudar de equipe ou até mesmo de empresa só porque quer trabalhar com pessoas melhores. Em vez disso, encontre um projeto para trabalhar como voluntário em que você possa trabalhar com outros desenvolvedores, que irão torná-lo melhor por osmose. Veja se há encontros de grupos de desenvolvedores em sua cidade e participe dessas reuniões. Desenvolvedores estão frequentemente procurando por projetos para ocupar seu tempo livre, praticar novas técnicas e aprimorar suas habilidades.

Se não há uma comunidade ativa perto de você, use a internet. Escolha um projeto *open source* de que você goste e cujos desenvolvedores parecem estar em um nível acima do seu. Vá até a lista de tarefas do projeto ou o histórico da lista de discussão, escolha uma funcionalidade ou uma correção de bug importante e escreva o código! Imita o estilo de código do projeto. Faça disso um jogo. Faça o seu código de forma que ele seja indistinguível do restante do projeto, de modo que até mesmo os desenvolvedores originais não saibam quem escreveu. Então, quando você estiver satisfeito com o seu trabalho, submeta suas modificações. Se for bom, vai ser aceito no projeto. Comece de novo e faça novamente. Se você tomou decisões das quais os desenvolvedores do projeto discordam, pegue o *feedback* deles, use-o para melhorar seu código e envie novamente as modificações. Se eles modificarem seu código, tome nota das alterações que fizeram. Em seu próximo *patch*, tente fazer com que ele seja aceito com menos retrabalho. Eventualmente, você vai perceber que irá se tornar alguém de confiança da equipe do

projeto. Você vai se surpreender com o que se pode aprender a partir de um conjunto remoto de desenvolvedores seniores, mesmo se você nunca teve a chance de ouvir suas vozes.

## CAPÍTULO 5

# Invista em sua inteligência

Quando escolher em que focar, pode ser tentador simplesmente olhar para as tecnologias que geram mais empregos e concentrar-se nelas. Java é grande. .NET é grande. Aprender Java tem um efeito simples, transitivo: se eu sei Java, eu posso me candidatar, e possivelmente, conseguir um trabalho onde vou escrever código Java.

Usando essa lógica, seria insensato escolher investir em um nicho de tecnologia, especialmente se você não tem intenção de tentar explorá-lo.

O TIOBE, <http://www.tiobe.com>, usa os sites de busca da internet para identificar a popularidade de linguagens de programação, com base em pessoas falando sobre as linguagens. Segundo o TIOBE, “Os números são baseados na disponibilidade mundial de engenheiros qualificados, cursos e fornecedores”. Definitivamente não é uma medida científica de popularidade, mas não deixa de ser um bom indicador.

No momento da escrita desse livro, a linguagem mais popular é o Java, seguido por C. C# está em um respeitável sexto lugar, mas com uma trajetória levemente na ascendente. ABAP está em décimo sétimo lugar e está caindo lentamente. Ruby, que

é a minha linguagem favorita — na qual eu faço praticamente todo o meu trabalho sério e aquele para o qual eu co-organizo uma conferência internacional a cada ano — está em décimo primeiro lugar. E no momento em que a primeira edição deste livro foi publicada, ela não ficou nem entre os top vinte. Ficou abaixo de ABAP!

Por que Ruby? Eu estava louco ou fui burro? Deve ser um dos dois, certo?

Em seu trabalho chamado “Grandes Hackers” — <http://paulgraham.com/gh.html>, Paul Graham deixou a comunidade de desenvolvedores irritadíssima com a afirmação de que os programadores Java não eram tão espertos quanto quem programava em Python. Ele deixou bravo um monte de programadores Java burros (eu disse isso?), fazendo com que vários deles escrevessem respostas a ele em seus sites. A reação violenta indica que ele atingiu algo. Eu estava na plateia quando seu trabalho foi apresentado pela primeira vez, na forma de um discurso. Ele causou um *flashback* em mim.

Eu estava em uma viagem de recrutamento na Índia, no meio de centenas de candidatos para dezenas de vagas, e a equipe de entrevistadores estava cansada e ficando sem tempo por conta do baixo índice de entrevistas bem sucedidas, que resultassem em contratações. Com dores de cabeça e olhos vermelhos, fizemos uma reunião tarde da noite para discutir uma mudança estratégica na forma como iríamos abordar os candidatos. Nos tínhamos que otimizar o processo para que pudéssemos entrevistar mais pessoas ou de alguma forma entrevistar pessoas melhores (ou ambos). Com o pouco que restava da minha voz depois de doze horas seguidas de tentar arrancar respostas de programadores mudos, eu pedi para adicionar Smalltalk para a lista de palavras-chave que nossos headhunters estavam usando para procurar no banco de dados de currículos. “Mas ninguém sabe Smalltalk na Índia”, gritou o diretor de recursos humanos. Esse foi o meu ponto. Ninguém sabia, e programar em Smalltalk era uma experiência diferente de programar em Java. A mudança da experiência daria aos candidatos um nível diferente de expectativas, e a natureza dinâmica de Smalltalk levaria a uma outra maneira com a qual um programador Java iria abordar um problema. Minha esperança era que esses fatores encorajassem um nível de maturidade técnica que eu não tinha visto dentre os candidatos que eu conhecia até então.

A adição de Smalltalk para a lista de requisitos rendeu um grupo de candidatos que era pequeno se comparado com nossa lista anterior. Essas pessoas eram diamantes brutos. Elas realmente aprenderam programação orientada a objetos. Estavam cientes de que o Java não era a loucura idealista como às vezes era pintada. Muitos deles **amavam** programar! **"Onde você esteve nas últimas duas semanas?"**, nós



pensávamos.

Infelizmente, a nossa capacidade de atrair esses desenvolvedores, com os salários que éramos capazes de pagar, era limitada. Eles ditavam o tom, e a maioria deles preferiu ficar onde estava ou continuou procurando um novo emprego. Apesar de não conseguirmos recrutar muitos deles, aprendemos uma lição valiosa de recrutamento: nós estávamos mais propensos a fazer propostas para candidatos com experiências diversas (e até mesmo pouco ortodoxas) do que para aqueles cuja experiência era homogênea. Minha explicação é que, ou as pessoas boas buscam a diversidade, pois elas amam aprender coisas novas, ou ser forçado em experiências e ambientes mais exóticos criava programadores mais maduros e preparados. Eu acredito que seja um pouco dos dois, mas, independentemente do motivo pelo qual ele funciona, a gente descobriu que isso funciona. Eu continuo usando esta técnica quando procuro desenvolvedores.

Então, ao invés de tentar simplesmente aparecer na minha frente quando eu estiver à procura de alguém para contratar, por que você não vai investir em tecnologias em que você raramente terá a chance de ser pago para trabalhar?

Para mim, como um gerente de contratação, a primeira razão é que isso mostra que você está interessado. Se eu sei que você aprendeu alguma coisa por causa do autodesenvolvimento e (melhor ainda) pura diversão, eu sei que você está animado e motivado sobre a sua profissão. Fico muito irritado quando pergunto às pessoas se elas já viram ou usaram algumas tecnologias não tradicionais e ouço como resposta: “não me deram a oportunidade de trabalhar com isso”. Não lhe deram oportunidade? Pra mim também não! **Eu fiz** a oportunidade para aprender.

*Não lhe deram a oportunidade...? Faça a oportunidade!*

—

Mais importante do que retratar a percepção de que se está devidamente motivado e engajado em sua área é que a exposição a essas tecnologias e metodologias não tradicionais lhe traz mais profundidade e o torna melhor, mais inteligente e mais criativo.

Se essa não é uma razão suficientemente boa, então provavelmente você está na profissão errada.

**Faça algo**

- 1) Aprenda uma nova linguagem de programação. Mas não vá de Java para C# ou de C para C++. Aprenda uma nova linguagem que o faça pensar de uma maneira diferente. Se você é um programador Java ou C#, tente aprender uma linguagem como Smalltalk ou Ruby, que não usem tipagem forte e estática. Ou, se está programando orientado a objetos por bastante tempo, tente uma linguagem funcional como Haskell ou Scheme. Não precisa se tornar um especialista. Faça código o suficiente para que você realmente sinta a diferença de programar no novo ambiente. Se ela não soar estranha o suficiente, ou você escolheu a linguagem errada ou você está aplicando sua velha forma de pensar na nova linguagem. Peça para desenvolvedores mais experientes reverem o seu código e fazerem sugestões que o tornem idiomáticamente mais correto.

## CAPÍTULO 6

# Não escute seus pais

Em nossa cultura, se existe algo que é sagrado, provavelmente são os conselhos dados por nossos pais. É visto como um dever da criança segui-los religiosamente. Livros, filmes e histórias de televisão formam um conjunto de ideias na cabeça de nossos pais. Mas para nossas carreiras no mercado de trabalho, esta ideia é errada.

Seus pais preferem que você esteja estável ao invés de conseguir uma carreira notável ao custo de correr grande risco pessoal. Mais do que quaisquer outras pessoas, eles vão lhe dar conselhos influenciados pelo medo. Conselhos influenciados pelo medo tendem ao **não perder**. Mas pensar em não perder não é o caminho para ganhar! Vencedores assumem riscos. Eles pensam sobre aonde eles querem ir, não onde o resto das coisas estão. Planejamento de carreira guiado pelo medo provavelmente o levará a algum cubículo para o resto de sua vida, em vez de ao caminho para a grandeza. Claro, é seguro, mas não é divertido.

Para as gerações anteriores, diversão não era fator decisivo quando o assunto era opções de carreira. Empregos não deveriam ser divertidos. Eles deveriam trazer comida pra casa. Diversão é o que você faz nos seus dias de folga. Diversão acontece

nas noites e fins de semana. Mas se o seu trabalho não é divertido, como temos percebido, você não o faz bem. As coisas não são diferentes agora, mas nossa compreensão cultural do que significa trabalhar mudou para melhor. Mais pessoas entendem que a paixão leva à excelência. E sem diversão, não é provável que haja qualquer paixão em um trabalho.

Outro fator de tomada de decisão sobre carreira, que provavelmente não vai de encontro com o pensamento dos seus pais, é que não há problema em mudar de emprego (na verdade, muitas vezes é preferível). Um profissional de software experiente já viu o mercado por diferentes ângulos: desenvolvimento de produto, suporte de TI, desenvolvimento de sistemas internos e trabalho para o governo. Quanto mais domínios e arquiteturas técnicas você já tenha visto e trabalhado, mais está preparado para tomar as decisões corretas em projetos mais desafiadores. Ficar em uma única empresa, trabalhando para subir na hierarquia, é um ambiente limitador para crescer como um desenvolvedor. Já foi o tempo em que as pessoas começavam sua carreira em uma empresa e ficavam lá mesmo até se aposentar. Este tipo de comportamento costumava ser sinal de dedicação. Agora é uma responsabilidade. Se você só trabalhou em um lugar e viu aquele conjunto de sistemas, muitos gerentes (inteligentes) podem ver isso como um ponto negativo contra você quando for decidir se deve contratá-lo. Eu pessoalmente prefiro contratar alguém que tenha vivenciado diversas situações de sucessos e fracassos em ambientes diferentes do que alguém que conhece apenas um jeito de fazer as coisas.

Há muitos anos, eu percebi que a minha própria carreira tinha sido muito influenciada pelos valores profissionais dos meus pais e sua geração. Eu trabalhei em uma das maiores e mais estáveis empresas do mundo e estava crescendo em um ritmo lento e constante. Mas eu estava estagnando. Eu ficava me dizendo que eu não estava catando migalhas, baseando-me no fato de que a empresa era tão grande que eu poderia executar diferentes tarefas em uma aparente infinidade de lugares. Mas no final, eu ficava no mesmo lugar fazendo o mesmo tipo de trabalho.

Eu me lembro de conversar com um amigo sobre sair da empresa, e ele disse: “É o seu destino trabalhar em grandes empresas para o resto da sua vida?” **Claro que não!** Então, eu rapidamente encontrei outro emprego e saí.

Este movimento marcou o início de um salto no meu sucesso na indústria de software. Eu vi novos domínios, trabalhei em problemas mais difíceis, e fui recompensado como nunca havia sido. Algumas vezes chegava até a ser assustador, mas quando eu decidi ser menos influenciado pelo medo e menos conservador nas decisões sobre minha carreira, ela acabou mudando e para melhor.

Assuma riscos em sua carreira. Não deixe o medo o consumir. E se você não estiver se divertindo, você não vai ser excelente.

## Como eu abri mão de 300 mil dólares na Microsoft para trabalhar full time no Github

*Por Tom Preston Werner*

Em 2007 eu estava sentado sozinho em uma mesa no Zeke Sports Bar and Grill na 3rd Street, em San Francisco. Eu normalmente não vou a bares de esportes, e muito menos a um bar de esportes em SOMA, mas aquela quinta-feira foi uma noite *"I Can Has Ruby"*. Naquela época, eu acho que *"I can has \_"* também foi um apelido razoável para dar a praticamente qualquer coisa. ICHR era um encontro de hackers Ruby que geralmente mergulhavam em sessões de fim de noite bebendo. Normalmente as noites passavam, como a minha ressaca na manhã seguinte, mas esta noite foi diferente. Esta foi a noite em que nasceu o GitHub.

Eu acho que estava sentado na mesa sozinho, porque tinha acabado de pedir um *Fat Tire* e precisava dar uma parada na socialização que estava acontecendo nas mesas longas e mal iluminadas do bar. No quinto ou sexto gole, Chris Wanstrath entrou, eu não sei dizer se na época Chris e eu éramos amigos. Nós nos encontramos em *meetups* Ruby e conferências, mas apenas casualmente, com um mútuo "Ei, eu acho seu código impressionante". Eu não sei o que me fez fazer isso, mas eu fiz um sinal para ele e disse: "Cara, olha isso". Cerca de uma semana antes, eu tinha começado a trabalhar em um projeto chamado Grit, que permitia acessar repositórios Git através de código Ruby de forma orientada a objetos. Chris era um dos poucos rubistas na época que estava começando a levar Git a sério. Ele se sentou e eu comecei a mostrar o que eu tinha. Não era muito, mas foi o suficiente para ver que tinha despertado algo em Chris. Percebendo isso, lancei um site meia-boca que funcionava como hub para programadores que queriam compartilhar seus repositórios Git. Eu até tinha um nome: GitHub. Posso estar parafraseando, mas sua resposta foi um enfático "Estou dentro! Vamos fazer isso!"

Na noite da próxima sexta-feira, 19 de outubro de 2007, às 22:24, Chris fez o primeiro commit para o repositório do GitHub, marcando o início de nossa *joint venture*.

Até então, não havia nenhum acordo sobre como as coisas iriam funcionar. Éramos apenas dois caras que decidiram hackear juntos em algo que parecia legal.

Lembra daqueles minutos incríveis do Karate Kid em que Daniel está treinando para se tornar um especialista em artes marciais? Lembra da música? Bem, você

deveria ouvir, porque estou prestes a lhe contar uma história. A música é *You are the best* do Joe Esposito.

Pelos os próximos três meses, Chris e eu passamos horas planejando e codificando o GitHub. Eu continuei com o Grit e desenhei a interface do usuário. Chris construiu a app Rails. Nós nos encontrávamos todos os sábados para tomar decisões de design e tentar pensar no nosso plano de preços. Eu me lembro de um dia muito chuvoso em que conversamos por umas boas duas horas sobre as diferentes estratégias de preços dos melhores *egg rolls* vietnamitas da cidade. Fizemos tudo isso mantendo nossos compromissos. Eu, por exemplo, trabalhava em tempo integral na Powerset como desenvolvedor de ferramentas para o Ranking e a equipe da Relevance.

Em meados de janeiro, após três meses de noites e finais de semana, lançamos um beta privado e enviamos convites para nossos amigos. Em meados de fevereiro, PJ Hyett se juntou a nós. Lançamos o site ao público em 10 de abril. Não convidamos o TechCrunch. Neste ponto, nós éramos apenas três caras de 20 e poucos anos sem um centavo sequer de investimento externo.

Eu ainda estava trabalhando em tempo integral na Powerset em 1 de julho de 2008, quando soubemos que ela tinha acabado de ser adquirida pela Microsoft por cerca de US\$ 100 milhões. Um *timing* interessante. Com a aquisição, eu estava a caminho de ser confrontado com uma escolha muito mais cedo do que eu imaginava. Eu poderia me tornar um funcionário da Microsoft ou sair e ir cuidar do GitHub em tempo integral. Aos 29 anos de idade, eu era o mais velho dos três GitHubbers e tinha acumulado uma quantidade proporcionalmente maior de dívidas e despesas mensais. Eu estava acostumado com meu estilo de vida de seis dígitos. Para confundir ainda mais, minha esposa, Theresa, estava para voltar do seu PhD, na Costa Rica. Logo eu iria voltar a ser um homem casado.

Para tornar a decisão ainda mais difícil, a oferta de emprego da Microsoft foi tentadora. Salário, mais 300000 dólares em três anos. Isso é dinheiro o suficiente para fazer qualquer pessoa pensar duas vezes sobre qualquer coisa. Então, fui confrontado com isso: um trabalho seguro na Microsoft e com muito dinheiro garantido ou um trabalho arriscado, com quantidades desconhecidas de dinheiro, como empresário. Eu sabia que as coisas com os outros Githubbers ficariam tensas se eu não saísse da Powerset. Por terem guardado algum dinheiro e se tornado freelancers há algum tempo, ambos tinham começado a se dedicar em tempo integral no GitHub. Era hora de “fazer ou morrer”. Era escolher o GitHub e investir nele ou fazer a escolha segura e desistir do GitHub para fazer dinheiro na Microsoft.

Se você quer uma receita para uma péssima noite de sono, eu posso lhe dar. Adicione uma pedaço de “O que minha mulher acha?”, com 3.000 pedaços de Benjamin Franklin, misture com uma “cerveja a hora que você bem entender” e adicione uma cobertura de chance de independência financeira.

Eu me tornei muito bom em chegar nos meus chefes e dar a má notícia de que eu estava deixando a empresa para ir fazer alguma coisa mais legal. Eu dei a notícia para meu chefe na Powerset na data limite de responder à proposta. Eu disse que estava saindo para ir trabalhar em tempo integral no GitHub. Como qualquer grande chefe, ele estava chateado, mas entendeu. Ele não tentou me seduzir com um bônus maior ou qualquer coisa do tipo. Acho que no fundo ele sabia que eu ia sair. Talvez eu tenha até recebido mais incentivo para ficar do que os outros, por conta do risco. E vou lhe dizer, aqueles gerentes da Microsoft eram persistentes. Eles têm bônus de retenção como uma ciência — bem, exceto quando você tem um empreendedor no conjunto, a singularidade do mundo dos negócios. Tudo é desequilibrado quando você tem um deles por perto.

No final, assim como Indiana Jones nunca poderia recusar a oportunidade para procurar o Santo Graal, eu também não poderia perder a chance de trabalhar para mim mesmo em algo que eu realmente amo, não importa o quão segura outra opção fosse. Quando eu estiver velho e morrendo, eu pretendo olhar para trás em minha vida e dizer: “Uau, que aventura”, não “Uau, me senti seguro.”

*Tom Preston-Werner é co-fundador do GitHub.*

## Faça algo

- 1) Quais são seus maiores medos com relação à sua carreira? Pense sobre as últimas decisões de carreira que você tomou. Não precisam ser grandes decisões (afinal, se você está fazendo escolhas influenciadas por medo, suas decisões provavelmente não serão grandes). Por exemplo, pode ser se você aceitou alguma tarefa especial, ou se candidatou para uma promoção. Faça uma lista dessas escolhas, e para cada uma, obrigue-se a fazer uma avaliação honesta: o quanto dessa sua decisão foi conduzida pelo medo? O que você teria feito se o medo não o tivesse influenciado? Se a decisão foi de fato influenciada pelo medo, como você pode revertê-la ou encontrar uma oportunidade similar, em que possa tomar a decisão com menos medo?





## CAPÍTULO 7

# Seja generalista

Por pelo menos algumas décadas, gerentes e donos de empresas desesperados têm fingido que o desenvolvimento de software é um processo fabril. Especificações de requisitos são criadas e os arquitetos as transformam em algo de nível técnico. Designers completam a arquitetura com a documentação detalhada do projeto, que é entregue para programadores robô, que com uma mão seguram um livro ruim e com a outra escrevem código. Por fim, um investigador recebe o código completo, que não ganha o selo de aprovação a menos que cumpra as especificações originais.

Não é nenhuma surpresa que os gestores queiram que o desenvolvimento de software seja como uma fábrica. Eles “entendem” como fazer fábricas funcionarem. Temos décadas de experiência em como construir objetos físicos eficientemente. Assim, aplicando o que aprendemos de manufatura, devemos ser capazes de otimizar o processo de desenvolvimento de software para como qualquer outra indústria funciona.

Na tão falada fábrica de software, os funcionários são especialistas. Eles se sentam em seu lugar na linha de montagem, combinando componentes Java ou apa-

rando arestas de um aplicativo Visual Basic. O investigador é um testador. Os componentes vão passando e são testados e carimbados da mesma forma todo dia. Designers J2EE desenham aplicações J2EE. Programadores C++ programam em C++. O mundo é muito limpo e organizado.

Infelizmente, a analogia da fábrica não funciona. Software é no mínimo tão maleável como os requisitos de software. As coisas mudam no negócio, e os empresários sabem que o *software* é *soft* e pode ser alterado para atender a essas mudanças. Isso significa que arquitetura, design, código e os testes devem todos ser criados e revisados de uma forma mais ágil do que os processos de fabricação mais enxutos podem proporcionar.

Neste tipo de ambiente de mudanças rápidas, o flexível irá sobressair. Quando há pressão, um empresário inteligente vai até um profissional de software que consiga resolver o problema. Então, como você se torna a pessoa cujo nome é lembrado quando eles estão à procura de um super-herói para salvar o dia? A chave é ser capaz de resolver os problemas que possam surgir.

Quais são esses problemas? É isso mesmo: você não sabe. Nem eu. O que eu sei é que há problemas dos mais diversos, como de implantação, falhas de projeto críticos que precisam ser resolvidos e rapidamente reimplementados, integração de sistemas heterogêneos e geração de relatórios. Diante de um conjunto de problemas tão diversos como este, aquele investigador ficaria rapidamente para trás.

O rótulo de “conhece um pouco de tudo, mas tudo de nada” é normalmente pejorativo, o que implica que o rotulado não tem o foco para realmente mergulhar em um assunto e dominá-lo. Mas, quando sua loja online está fora do ar e você está perdendo centenas de vendas conforme o tempo passa, é o “conhece um pouco de tudo” que não só sabe como funciona o código do aplicativo, mas também pode fazer debug dos processos no servidor web rodando em UNIX, analisar a configuração do seu banco de dados para possíveis gargalos de performance, e verificar a configuração de seu roteador de rede para encontrar problemas difíceis. E, mais importante, depois de encontrar o problema, o “conhece um pouco de tudo” pode rapidamente tomar decisões de arquitetura e de design, implementar correções de código, e implantar uma correção para o sistema em produção. Neste cenário, a ideia de fábrica parece bem antiquada na melhor das hipóteses e terrivelmente falha na pior.

Outro ponto em que a fábrica de software não funciona é que, em contraste com uma linha de montagem em que o trabalho continua vindo em um fluxo constante, projetos de software são geralmente cíclicos. Não só o fluxo dos projetos são cíclicos, mas o trabalho dentro de um projeto também é. Um programador fica lá sentado na

cadeira enquanto os requisitos estão sendo especificados, arquitetados e projetados, ou o programador trabalha ao mesmo tempo em vários projetos. O problema com os programadores multitarefa é que, apesar das intenções da fábrica de software, quando o bicho pega, os programadores se baseiam bastante no contexto e na experiência para concluir seus trabalhos. Requisitos, arquitetura e documentos de design podem ser um começo, mas no final, se os programadores não entenderem o que o sistema deve fazer, eles não serão capazes de implementá-lo bem.

Claro que aqui eu não estou pegando no pé só de programadores. O mesmo é verdade em quase todas as linhas de montagem de software. Contexto importa, e ser multitarefa não funciona. Como resultado, temos um sistema de produção ineficiente. Já houve várias tentativas de resolver este problema de ineficiência, sem sair do sistema de produção de inspiração em fábricas, mas ainda não descobrimos como otimizar nossas fábricas de software a um nível aceitável.

Se você é “apenas” um programador, ou um testador, ou um designer, ou um arquiteto, você vai se encontrar ocioso ou fazendo trabalho pesado e sem importância durante vários momentos do seu projeto. Se você é “apenas” um programador J2EE, ou um programador .NET, ou um programador de sistemas UNIX, você não vai ter muito a contribuir quando o foco de um projeto ou de uma empresa de mudar, mesmo que temporariamente, da sua área de foco. Não é sobre onde você está na cadeia de valor de trabalho percebido do projeto (onde o arquiteto tem o posto mais alto da realeza). É sobre o quão útil você é.

Se seu objetivo é ser a última pessoa de pé em meio a rodadas de demissões ou terceirização de trabalho, é melhor se tornar “genericamente” útil. Se você tem medo de que o seu escritório de desenvolvimento, que antes era lotado, vire a casa de um bando de gente estranha, ajudaria se você percebesse que quando o time tem apenas algumas vagas, o “só testador” ou o “apenas codificador” não serão requisitados. Melhor, se você só quer se destacar e ser notável, envolver-se no todo é o melhor que você faz.

*Generalistas são raros...e por isso, preciosos.*

—

O caminho para se tornar um generalista é não se rotular com um papel específico ou tecnologia. Podemos nos tornar flexíveis em nossas carreiras de muitas maneiras. Para entender o que significa ser um generalista, podemos dar uma olhada em como está o cenário das carreiras de TI, em diferentes aspectos independentes.

Eu consigo pensar em cinco, mas é claro que há uma infinidade (tudo depende de como você separa os assuntos):

- Degrau na escada da carreira;
- Plataforma / OS;
- Código x dados;
- Sistemas x aplicações;
- Negócio x TI.

Essas são diferentes formas pelas quais você poderia abordar o problema de se tornar um generalista. Esta é apenas uma maneira de pensar sobre toda a sua carreira, e você provavelmente pode ter uma lista melhor para si próprio. Por agora, vamos discutir esses pontos.

Primeiro, você pode optar por ser um líder ou gerente, ou ser uma pessoa técnica. Ou, você pode se autotransformar um arquiteto em vez de ser um programador ou testador. A capacidade de ser flexível nos papéis que você pode e vai ter é um atributo cujo valor muitas pessoas não compreendem. Por exemplo, enquanto um líder forte deve evitar ao máximo se intrometer, esse novo mundo de equipes de desenvolvedores terceirizados pode se beneficiar de uma pessoa que sabe como liderar pessoas e projetos, mas também pode arregaçar as mangas e corrigir alguns bugs críticos de última hora, enquanto a equipe offshore está dormindo. O mesmo vale para um arquiteto de software que possa acelerar dramaticamente o progresso em um projeto se ele escrever um pouco de código. Quando se trata de atravessar hierarquias, não é a relutância que impede as pessoas de fazê-lo. É a capacidade. Programadores nerds não podem liderar, e os líderes e gerentes não podem escrever código. É raro encontrar alguém seja bom nos dois.

*Suas habilidades transcendem tecnologias.*

—

Outra falha imperdoável é encontrada quando falamos de plataformas e sistemas operacionais. Ser um cara UNIX que se recusa a trabalhar com Windows é cada vez mais inviável. O mesmo vale para .NET vs J2EE ou quaisquer outras plataformas de infraestrutura. O tempo vai exigir que você seja neutro com relação à plataforma no seu ambiente de trabalho. Todos nós temos nossas preferências, mas você vai

ter que deixar seus ideais em casa. Domine um e fique bom no outro. Suas habilidades devem transcender a tecnologia e plataforma. Ela é apenas uma ferramenta. Se queremos alguém que só saiba Windows, podemos contratá-lo nas Filipinas. Se quisermos alguém que realmente entenda de Windows e UNIX e pode integrá-los, provavelmente vamos procurar aqui por perto. Não seja deixado de lado por causa de algo que é essencialmente espírito de equipe.

A linha que divide o administrador de banco de dados (um papel solidificado em TI na última década) e desenvolvedor de software também é tênue. Ser um administrador de banco de dados ou DBA, significa em muitos lugares saber como usar alguma ferramenta visual de administração e como configurar um banco de dados. Você não precisa necessariamente saber muito sobre como usar o banco nas aplicações. Por outro lado, os desenvolvedores estão cada vez mais preguiçosos e ignorantes sobre como trabalhar com bancos de dados. Cada lado alimenta o outro.

O que mais me surpreendeu quando eu entrei na área de TI era que muitos programadores (talvez a maioria) não sabiam como configurar os ambientes que eles utilizavam para o desenvolvimento e para *deploy*. Eu trabalhei com desenvolvedores que não conseguiam sequer instalar um sistema operacional em um computador se você pedisse, muito menos configurar um servidor de aplicações para fazer o *deploy* das suas aplicações. É raro encontrar um desenvolvedor que realmente entende a plataforma em que está trabalhando. As aplicações são melhores por consequência, o trabalho costuma ser feito mais rápido.

Finalmente, como discutimos no capítulo 3, a barreira entre o negócio e TI deve ser derrubada já! Comece a entender como sua empresa funciona.

## Faça algo

- 1) Em um pedaço de papel ou em uma lousa, liste as dimensões em que você pode ou não estar generalizando seus conhecimentos e habilidades. Para cada dimensão, escreva sua especialidade. Por exemplo, se “sistema operacional” é uma de suas dimensões, você pode escrever Windows/.NET ao lado. Agora, à direita da sua especialidade, escreva um ou mais tópicos que você deveria aprender. Continuando com o mesmo exemplo, você pode escrever Linux/Java (ou mesmo Ruby ou Perl).

Assim que possível (**mas ainda essa semana**), pegue 30 minutos de seu tempo e comece a aprender alguma tecnologia que você listou. Não basta ler sobre ela. Se possível, faça um *hands-on*. Se é uma tecnologia web, então faça você mesmo a

instalação e configuração do servidor. Se é um assunto relacionado a negócios, escolha um de seus clientes no trabalho e chame-os para almoçar ou conversar.

## CAPÍTULO 8

# Seja especialista

“Como você escreveria um programa, em Java puro, que trave a Java Virtual Machine?” Silêncio mortal. “Oi?”

“Desculpa. Não entendi. Você pode repetir a pergunta, por favor?” A voz soava desesperada. Eu sabia que repetir a pergunta não iria ajudar em nada. Então, eu repeti a pergunta, devagar e em um tom de voz mais alto. “Como você escreveria um programa, em Java puro, que faça Java Virtual Machine parar de funcionar?”

“Uh... desculpe. Eu nunca fiz isso.”

“Eu sei que não. Que tal essa pergunta: como é que você escreveria um programa que NÃO trave a JVM?”

Eu estava procurando por programadores Java realmente bons. Para começar a entrevista, perguntei para esta pessoa (e todos os outros candidatos que eu havia entrevistado naquela semana) para avaliar a si mesmo em uma escala de um a dez. Ele disse **nove**. Estou esperando uma estrela aqui. Se esse cara se avalia de forma tão alta, por que ele não consegue pensar em algum simples truque que trave a JVM?

A falta de profundidade técnica.

*Muitos acreditam que ser especialista em algo significa que você não sabe outras coisas.*

—

Essa pessoa se dizia especialista em Java. Se você o conhecesse em uma festa e perguntasse o que ele fazia para viver, ele diria: “Eu sou um desenvolvedor Java”. Ainda assim, ele não conseguia responder a esta simples pergunta. Ele não conseguia sequer chegar a uma resposta **errada**. Depois de intensas duas semanas e meia de entrevistas em uma viagem de recrutamento, esta foi a regra, não a exceção. Milhares de especialistas Java tinham se candidatado para as vagas abertas, mas quase nenhum deles sabia explicar como *Classloader* Java trabalhava ou então dar uma visão geral de como o gerenciamento de memória funciona em uma JVM.

Claro, você não tem que saber essas coisas para escrever código sob a supervisão de outros. Mas estas pessoas deveriam ser **especialistas**.

Muitos acreditam que ser especialista em algo significa que você não sabe outras coisas. Eu poderia, por exemplo, chamar minha mãe de especialista em Windows, porque ela nunca usou o Linux ou Mac OS X. Ou, eu poderia dizer que meus parentes na zona rural do Arkansas são especialistas em música country, porque nunca ouviram nada diferente.

Imagine que você visite o seu médico da família, reclamando de um nódulo estranho no seu braço direito. O seu médico indica um especialista para fazer uma biópsia. E se esse especialista fosse uma pessoa cujas únicas credenciais como especialista eram que ele não compareceu às aulas na escola de medicina ou nunca teve experiência em residências que não eram diretamente ligadas ao procedimento específico que ele vai realizar hoje em você? Eu não quero dizer que eles foram mais fundo nos temas relacionados ao procedimento de hoje. E se ele tivesse apenas visto por cima essas assuntos e não soubesse de mais nada? “E se essa máquina ali começar a apitar durante o procedimento?”, você pode perguntar. “Ah, isso nunca aconteceu antes. Não vai acontecer desta vez. Eu não sei o que a máquina faz, mas ela nunca apita.”

Felizmente, a maioria dos desenvolvedores de software não é responsável por situações de vida ou morte. Se eles fracassarem, isso geralmente vai resultar em atrasos nos projetos ou bugs que simplesmente custarão dinheiro de seus empregadores, não vidas.

Infelizmente, a indústria de software tem movimentado um monte desses especialistas rasos, que usam o termo especialista como uma desculpa para saber só uma



coisa. Na indústria médica, um especialista é alguém com profundo conhecimento de alguma área específica. Doutores encaminham os pacientes para especialistas, porque, em certas circunstâncias, o especialista pode cuidar melhor do paciente do que um clínico geral.

Então, o que um especialista de software deve ser? Eu posso lhe dizer o que eu estava procurando por toda a parte durante a viagem de recrutamento. Era por pessoas que entendessem profundamente programação Java e ambiente de implantação. Eu queria pessoas que pudessem dizer “estive lá, fiz isso” em 80% das situações e cuja profundidade de conhecimento passasse tranquilidade para as outras 20% das situações. Eu queria alguém que, ao lidar com abstrações de alto nível, entendesse os detalhes da implementação dessas abstrações. Eu queria alguém que pudesse resolver qualquer problema de implantação que encontrássemos ou que ao menos soubesse a quem pedir ajuda se necessário.

Este é o tipo de especialista que vai sobreviver na indústria de TI. Se você é um especialista em .NET, isso não é uma desculpa para não saber nada, além de .NET. Isso significa que, se tem a ver com .NET, você é o cara. Servidores IIS travados e precisando de reboot? “Não tem problema.” Integração de código fonte com o Visual Studio .NET? “Eu lhe mostro como.” Clientes à beira de um ataque de nervos por conta de problemas de performance que ninguém sabe o que é? “Dê 30 minutos.”

Se isso não é o que ser um especialista significa para você, então eu espero que não saia falando que é um.

## Faça algo

- Você usa uma linguagem de programação que compila e roda numa máquina virtual? Se assim for, levará algum tempo para aprender sobre os detalhes de como a VM trabalha. Considerando Java, .NET e Smalltalk, muitos livros e sites são dedicados ao tema. É mais fácil aprender sobre elas do que você imagina.

Independente de a sua linguagem se basear em uma VM, leva algum tempo para estudar o que acontece quando você compila um arquivo fonte. Como é que o código que você digitou passa de texto legível para as instruções que um computador pode executar? O que significaria escrever o seu próprio compilador?

Quando você importa ou usa bibliotecas externas, de onde elas vêm? O que realmente significa importar uma biblioteca externa? Como é que o seu com-

pilador, sistema operacional, ou máquina virtual liga vários pedaços de código para formar um sistema coerente? Aprender esses pontos fará com que você fique muitos passos mais próximos de ser um especialista na sua tecnologia preferida.

- Encontre uma oportunidade — no trabalho ou fora dele — de dar uma aula sobre algum aspecto de uma tecnologia em que você gostaria de se aprofundar. Como você verá em “Seja um mentor”, capítulo 14, o ensino é uma das melhores maneiras de aprender.

## CAPÍTULO 9

# Não coloque todos os seus ovos num só cesto

Uma vez, enquanto gerenciava um time de desenvolvedores, eu perguntei a um dos meus funcionários: “O que você quer fazer com sua carreira? O que você quer ser?” Fiquei muito decepcionado com sua resposta: “Quero ser um arquiteto J2EE” Eu perguntei: “Por que não um designer Microsoft Word ou um instalador RealPlayer?”

Ele queria fazer sua carreira em torno de uma tecnologia específica, criada por uma empresa específica, da qual ele não era um empregado. E se a empresa acabar? E se sua tecnologia se tornar obsoleta? Por que você quer confiar sua carreira a uma empresa de tecnologia?

De alguma maneira, como uma indústria, nós mesmos nos enganamos ao pensar que **líder de mercado** é a mesma coisa que **padrão**. Assim, para algumas pessoas, parece inteligente fazer com que o produto de uma empresa faça parte da sua identidade. Pior ainda, alguns baseiam suas carreiras em torno de produtos que nem líderes de mercado são — pelo menos até suas carreiras falharem miseravelmente e

sua única escolha for repensar sua estratégia perdedora.

Vamos parar para lembrar que devemos pensar na nossa carreira como um negócio. Embora seja possível construir um negócio que seja um parasita de outro negócio (como empresas que constroem softwares de remoção de spyware para suprir falhas de segurança do navegador da Microsoft), como uma pessoa, isso é uma coisa extremamente arriscada de se fazer. Uma empresa, como a do exemplo do spyware, geralmente pode reagir às mudanças no mercado, como uma melhoria na segurança do navegador da Microsoft (ou a Microsoft decidir entrar no mercado de remoção de spyware), enquanto que uma pessoa não tem a mão de obra ou dinheiro sobrando, para de repente, mudar de direção na carreira.

*Uma visão centrada apenas no fornecedor é uma visão míope.*

—

O lado triste da visão centrada no fornecedor é que, normalmente, os detalhes de implementação do software de um fornecedor são um segredo. Você pode aprender bastante sobre o software de alguém, até chegar na **barreira do serviço profissional**. Essa é uma barreira artificial que a empresa impõe entre você e a solução para um problema possível, dessa forma, ela pode lucrar vendendo o suporte. Algumas vezes, essa barreira é proposital, e às vezes é um efeito colateral da tentativa da empresa de proteger sua propriedade intelectual (não compartilhando o código-fonte).

Assim, apesar de que o investimento em uma única determinada tecnologia seja quase sempre uma **má ideia**, se você precisar fazer isso, considere focar em uma opção *open source*, ao invés de uma comercial. Mesmo que você não possa ou não queira usar uma solução open source no seu trabalho, use-a para conseguir ir mais a fundo na tecnologia. Por exemplo, você pode querer se tornar um especialista em como funcionam os servidores de aplicação J2EE. Em vez de concentrar seus esforços sobre os detalhes de como configurar e implantar um servidor de aplicação comercial (afinal, qualquer um pode descobrir como ajustar as configurações em um arquivo de configuração, certo?), baixe o código fonte do JBoss ou Geronimo e reserve um tempo para si mesmo, não só para aprender a operar os servidores, mas para estudar suas partes internas.

Em pouco tempo, você vai perceber que naturalmente o seu ponto de vista vai mudar. Esse negócio de J2EE (ou seja lá o que você escolheu estudar) realmente não é nada de tão especial. Agora que você vê os detalhes da implementação, pode perceber que há padrões conceituais ali no meio. E começar a perceber que, seja com

Java ou outra linguagem ou plataforma, arquitetura distribuída é arquitetura distribuída. Sua visão aumenta e sua mente começa a abrir. Você começa a perceber que os conceitos e padrões que o seu cérebro está organizando e assimilando são muito mais escaláveis e universais do que qualquer tecnologia de uma empresa específica. “Não importa o fornecedor, eu sei como criar um sistema!”

## **Faça algo**

- 1) Tente um projeto pequeno, duas vezes. Tente uma vez em sua tecnologia padrão e, depois, da forma mais idiomática possível, faça em uma tecnologia concorrente.



## CAPÍTULO 10

# Ame-o ou deixe-o

Pode soar como uma espécie de historinha clichê feita pra despertar aquela exaltação idealista em você, mas é muito importante para deixar de falar. Se você quer ser realmente bom no seu trabalho, você tem que ser apaixonado por ele. Se você não dá a mínima pra ele, isso vai transparecer.

Quando minha esposa e eu nos mudamos para Bangalore, eu estava bem animado. Pela primeira vez na minha carreira, estava ansioso para encontrar tecnólogos com paixão por aprender. Estava esperando uma vida pós-trabalho cheia de reuniões de grupo de usuários e profundas discussões filosóficas sobre técnicas e metodologias de desenvolvimento de software. Eu estava esperando encontrar o Silicon Valley da Índia cheio de artesãos, entusiastas da grande arte de desenvolver software.

O que eu encontrei foi um **monte** de gente que estava lá só pra receber seu salário e **poucos** artistas apaixonados.

Era igual ao lugar de onde vim.

Claro, na hora eu não percebi que era igual ao lugar de onde vim. Eu tinha algumas referências dos Estados Unidos, mas sempre achei que era porque eu trabalhava

em cidades ruins ou em empresas com ambiente ruim. Sempre achei que minhas primeiras experiências na área de TI tinham sido exceções. “Deve ser assim com a maioria dos desenvolvedores”, pensava. “Eu só não encontrei o ambiente certo ainda.”

Comecei a trabalhar no departamento de TI da minha universidade através de uma recomendação do meu amigo Walter, que tinha me visto trabalhar com computadores, o suficiente para saber que eu provavelmente poderia fazê-los trabalhar melhor do que a maioria das pessoas que precisavam de ajuda na universidade. Eu não acreditava que eu podia, sem ter nenhum tipo de treinamento. Eu era apenas um tocadour de saxofone que gostava de jogar videogame. Mas Walter me candidatou em uma vaga e marcou uma entrevista para mim. Fui contratado sem nem uma única questão técnica ter sido perguntada, e era para começar imediatamente.

Quando eu apareci no trabalho, estava morrendo de medo de ser descoberto como um charlatão, o que eu realmente era. “O que é que este saxofonista está fazendo aqui com a gente, profissionais treinados?” Afinal, eu estava trabalhando com gente que tinha níveis avançados em ciência da computação. E aqui estava, tendo cursado só uma parte da faculdade de Música, tentando me encaixar como se eu soubesse alguma coisa.

Depois de alguns dias de trabalho, a verdade começou a afundar. “Essas pessoas não fazem ideia do que estão fazendo!” Na verdade, algumas pessoas estavam me observando trabalhar e **tomando notas!** Pessoas com **mestrado em ciência da computação!**

Minha primeira reação foi assumir que eu estava cercado por idiotas. Afinal, eu não tinha nenhum treinamento formal. Passei minhas noites tocando em bandas de bar e meus dias jogando videogame. Eu tinha aprendido a trabalhar com computadores só porque eu estava interessado neles. Na verdade, eu aprendi a escrever programas porque eu queria fazer meus próprios jogos de computador. Eu chegava em casa tarde depois de uma noite ensurdecidora em algum bar e ficava até o sol nascer em sites Gopher com tutoriais sobre programação. Aí eu ia dormir, acordava e continuava estudando até que a hora em que eu tinha que sair e tocar novamente. Eu parava meus estudos com meus amados jogos, comia e depois voltava para brincar com Gopher e qualquer compilador que eu conseguisse fazer funcionar.

*Trabalhe porque você não poderia **não** trabalhar.*

—

Pensando bem, eu era viciado, mas de um jeito bom. Meu desejo por criar tinha



sido despertado em grande parte da mesma forma que quando comecei a escrever música clássica ou tocar jazz. Eu era obcecado por aprender tudo o que eu pudesse. Eu não estava ali para uma nova carreira. Na verdade, muitos dos meus amigos músicos achavam isso uma irresponsável distração da minha carreira atual. Eu estava lá porque eu não poderia **não estar**.

Esta foi a diferença entre eu e os meus superestudados, mas de baixo rendimento, colegas de trabalho. Paixão.

Essas pessoas não tinham ideia de **por que** eles estavam na área de TI. Eles se encontraram por acaso em suas carreiras, porque acharam que programar podia pagar bem, porque os seus pais os incentivaram ou porque não conseguiam pensar em um curso melhor na faculdade. Infelizmente o seu desempenho no trabalho refletia isso.

Se você pensar nas biografias que você leu ou nos documentários a que assistiu sobre os grandes nomes em vários campos, esse mesmo padrão de vício e comportamento apaixonado aparece. Diz-se que o grande saxofonista de jazz, John Coltrane, praticava tanto que seus lábios chegavam a sangrar.

É claro, o talento natural tem um grande papel na habilidade. Nem todos podem ser Mozart ou Coltrane. Mas todos nós podemos dar um grande passo longe da mediocridade encontrando um trabalho pelo qual somos apaixonados.

Pode ser um domínio ou negócio que lhe interesse. Ou, por outro lado, pode ser uma tecnologia específica ou domínio de negócio que o afunde. Ou um tipo de empresa. Talvez você esteja destinado a pequenas equipes ou a times grandes. Ou a processos rígidos. Ou a processos ágeis. Qualquer que seja a combinação, leva algum tempo para encontrar a sua.

Você pode fingir por um tempo, mas a falta de paixão vai pegar você e seu trabalho.

## **Sendo um oportunista em série**

*por James Duncan Davidson*

Desde o início, eu não tive o que muitos consideram uma carreira tradicional. Pelo contrário, tem sido muito mais um caminho de seguir as oportunidades conforme elas aparecem. A primeira apareceu enquanto eu estava na escola trabalhando em uma licenciatura em arquitetura. Eu tinha decidido na idade de 15 ou 16 anos que eu queria ser um arquiteto, e passei muito tempo investindo no futuro. Mas as sementes do que realmente seria a minha carreira depois da escola foram plantadas em meu fascínio precoce com sistemas online BBS. Eu era um daqueles garotos que

amavam o modem de 300 no PC da família. Isso me levou, eventualmente, para a internet, que por sua vez me levou a Gopher e, então, World Wide Web.

A web imediatamente me fisionou. Eu construí vários sites pessoais em rápida sequência e aproveitei toda a tecnologia à minha disposição, ensinando a mim mesmo conforme necessário. Na época, eu pensei neste trabalho como experimentos em cyber-arquitetura. Parece muito grandioso e até mesmo bastante idiota agora, mas era o mundo em que nós dos primeiros tempos da web vivíamos. Estávamos tentando imaginar o que o futuro poderia trazer.

Naturalmente, o verdadeiro trabalho de construir o futuro da internet não estava acontecendo nos laboratórios de arquitetura. Ele estava acontecendo no mundo dos negócios. Em pouco tempo, e com base no que eu tinha feito com o meu site público, fui contatado por uma *startup* que estava construindo websites para os Hilton, Better Business Bureau e semelhantes. Eles tinham visto os sites eu havia construído, e, aparentemente, eu tinha o conjunto de habilidades de que eles precisavam. Ofereceram-me um trabalho com o que parecia, na época, um salário ridiculamente grande. Aceitei-o, imaginando que eu poderia aproveitar a onda por um tempo, ganhar algum dinheiro, e voltar para a escola em poucos anos.

Foi em 1995. Mal sabia eu quão longe as coisas iriam e aonde um pouco de vontade de mergulhar em algo novo iria me levar.

Enquanto eu ajudava a construir a primeira versão do website Hilton, que tinha posicionamento de reservas em tempo real, aprendi como construir sites usando uma variedade de tecnologias do lado do servidor. Em poucos meses, eu passei de aprendiz a criador de meus próprios frameworks. Olhando para trás, parece ridículo, mas naquele momento, isso era o que era necessário. Eu vi uma abertura, aceitei-a, e apostei nela por tudo que valia, reinventando a mim mesmo conforme necessário.

Uma coisa levou à outra. Em 1997, fui para a JavaSoft trabalhar aplicações servidoras, e em poucos anos, acabei encarregado da especificação Servlet. Infelizmente, era um esforço subfinanciado, e eu não tinha uma equipe para me ajudar a fazer tudo que precisava ser feito, incluindo construir uma nova implementação de referência. Mas eu não deixei isso me parar, e dei início ao desenvolvimento de uma implementação completamente nova, que eventualmente foi lançada como o Kit JavaServer Web Development. Poucos se lembram desse software. Mas a maioria dos que trabalham com Java sabe sobre a versão seguinte daquele código. É o chamado Tomcat. Ele foi lançado mundialmente via Fundação Apache com um sócio chamado Ant. A história por trás desse lançamento daria um livro. O suficiente é dizer que tudo aconteceu graças a um perfeito conjunto de oportunidades nas quais estive disposto

a apostar.

Depois de trabalhar na Sun por quatro anos e encarar uma questão do tipo “O que eu devo fazer agora?”, decidi me tornar independente. Escrevi livros para a O’Reilly, desenvolvi software para Mac, desenvolvi alguns softwares próprios que acabei não lançando. E terminei fazendo um pouco de desenvolvimento Ruby on Rails. Ser um desenvolvedor de software independente foi bom para mim e sou bastante bom nisso. Mas ao longo do caminho, um hobby que eu estava perseguindo começou a se transformar em uma própria carreira.

Além de ser um estudante de arquitetura que se tornou tecnólogo, eu era também um fotógrafo havia um bom tempo. Minha avó tinha me ensinado o básico. Meus pais me encorajaram. Como resultado, até onde posso me lembrar, eu tinha uma câmera por perto. Isso tem sido uma grande parte da minha vida. Aliás, o software não lançado que eu escrevi para mim mesmo depois de deixar a Sun era para o trabalho com fotografia.

Em 2005, dez anos depois que eu fiz uma pausa e mudei de estudante de arquitetura para desenvolvedor de software, fui chamado por meus amigos do grupo de conferências O’Reilly. Eles precisavam de alguém para documentar seus eventos e perguntaram se eu não estaria interessado em ir tirar algumas fotos. Eu aceitei, mas em vez de somente tirar algumas fotos, fui além da minha causa. Fiquei maluco e trabalhei em todas as sessões importantes, e postei imagens no Flickr para fornecer um retorno extremamente rápido. Fui convidado de novo e, nos últimos quatro anos, construí um negócio em torno disso com uma boa gama de clientes.

Enquanto escrevo isso, eu ainda faço código de tempos em tempos, e até desenvolvo um pouco para alguns clientes. Mas, na maior parte, eu sou um fotógrafo full-time nos dias de hoje. Contudo, isso pode mudar. Nunca se sabe. É difícil dizer o que o futuro trará.

O que eu sei é que sou um oportunista. Quando vejo algo interessante e excitante para mim, eu mergulho dentro disso e faço o que for preciso para obter sucesso. Geralmente, isso significa aprender habilidades e desenvolver novas capacidades. Alguns podem achar que construir novas habilidades é um entrave, mas por alguma razão, eu adoro aprender como fazer coisas novas. Afinal, novas habilidades permitem que você faça coisas novas. E eu nunca me defini por minhas habilidades. Ao contrário, sempre me defini pelo que fiz e pelo que eu quero fazer em seguida. Habilidades são apenas um caminho para chegar lá.

*James Duncan Davidson é um programador e fotógrafo.*

## Faça algo

- 1) Vá encontrar um emprego pelo qual você seja apaixonado.
- 2) Começando na próxima segunda-feira, mantenha um registro simples nas próximas duas semanas. A cada dia de trabalho, avalie seu nível de empolgação de 1 a 10. 1 significa que você preferia estar doente do que ir ao trabalho e 10 significa que você não consegue ficar na cama, pois está consumido pela ideia de finalizar a sua próxima tarefa.

Depois de duas semanas de registro, analise os resultados. Houve picos? Houve tendências? Foi tudo baixo ou tudo alto? Qual seria a sua nota média se isso fosse um teste de escola?

Para as próximas duas semanas, toda manhã planeje como você fará amanhã ser um 10. Planeje o que você vai fazer hoje para fazer com que amanhã seja um desses dias de trabalho que você mal pode esperar para começar. Cada dia, registre o nível de empolgação do dia anterior. Se depois de duas semanas as coisas não estiverem boas, talvez seja hora de pensar em uma mudança.

## CAPÍTULO 11

# Aprenda a pescar

Lao Tzu disse: “Dê a um homem um peixe, e alimente-o por um dia. Ensine um homem a pescar, e alimente-o por toda a vida”. Isso é tudo muito certo. Mas Lao Tzu não menciona a situação em que o homem não quer aprender a pescar e lhe pede outro peixe no dia seguinte. Educação exige tanto um professor como um estudante. Muitos de nós somos muitas vezes relutantes em ser um estudante.

*Não espere que lhe digam. Pergunte!*

—

Mas o que é um **peixe** na indústria de software? É o processo de utilização de uma ferramenta ou alguma parte de uma tecnologia ou uma informação específica do domínio de negócio no qual você está trabalhando. É como baixar uma *branch* específica do sistema no controle de versão, ou colocar no ar um servidor de aplicações para o desenvolvimento. Muitos de nós interpretam isso como definitivo. “Alguém pode cuidar disso para mim”, você pode pensar. O cara do build conhece o sistema de controle de versão. Você só precisa pedir a ele para definir as coisas quando você

precisar. A equipe de infraestrutura sabe como os firewalls entre você e seus clientes são configurados, por isso, se você tem uma necessidade, basta enviar um e-mail e a equipe vai cuidar disso.

Quem quer ficar à mercê de outra pessoa? Ou pior: se você estivesse contratando alguém para fazer um trabalho para você, você ia gostar que ele estivesse à mercê dos **especialistas**? Eu não. Eu ia querer contratar alguém autossuficiente.

O jeito mais óbvio para começar é aprender as ferramentas do seu mercado. Controle de versão, por exemplo, é uma ferramenta poderosa. Uma parte importante da sua função é focada em tornar os desenvolvedores mais produtivos. Não é apenas o lugar onde você coloca o seu código quando você o julga pronto, e você não deve tratá-lo como tal. É uma parte integrante do seu processo de desenvolvimento. Não deixe que uma coisa tão importante seja como um voodoo para você. Um desenvolvedor autossuficiente pode facilmente ver as diferenças entre a versão de um projeto que ele pegou e a última versão estável no repositório. Ou talvez você precise baixar a versão do último *release* e fazer uma correção de um bug. Se aparece um bug crítico no meio da noite, você não vai querer chamar alguém para pedir que lhe dêem a versão correta do código para você resolver os problemas. Isso vale para IDEs, sistemas operacionais e praticamente todas as partes de infraestrutura do seu código ou processo.

Igualmente importante é a plataforma tecnológica que você está usando. Por exemplo, você pode estar desenvolvendo aplicações usando J2EE. Você sabe que deve criar várias classes, interfaces e scripts de deploy. Mas você sabe **por quê**? Você sabe como essas coisas são usadas? Quando você inicia um container J2EE, o que realmente acontece? Você pode não ser um desenvolvedor de servidor de aplicação, mas saber como eles funcionam possibilita que você desenvolva código confiável naquela plataforma e resolva os problemas quando algo der errado.

Uma maneira particular e fácil de ser folgado é usar vários wizards que geram código para você. Isso é particularmente predominante no mundo de desenvolvimento Windows, em que, as ferramentas de desenvolvimento tornam várias tarefas realmente fáceis. O lado negativo é que muitos desenvolvedores Windows não têm ideia de como seus códigos realmente funcionam. O trabalho dos wizards continua sendo um mistério mágico. Não me leve a mal – geradores de código, quando usados corretamente, podem ser uma ferramenta muito útil. Por exemplo, são eles que traduzem C# de alto nível para bytecodes que podem rodar em .NET. Você obviamente não gostaria de ter que escrever todos esses bytecodes você mesmo. Mas, especialmente em níveis mais altos, deixar os wizards fazerem seu trabalho torna o

seu conhecimento raso e o deixa limitado ao que as ferramentas podem fazer por você.

Nós podemos facilmente ignorar o “peixe” em nosso domínio de negócio. Se está trabalhando para uma empresa de hipotecas, você pode pedir para um especialista o cálculo da taxa de juros para cada cenário que você precisar nos testes, ou poderia você mesmo aprender como calcular. Embora interações com seus clientes sejam boas, e é bom clarear os requerimentos do negócio com eles (o contrário de entender pela metade e preencher os detalhes você mesmo), imagine quão rápido você poderia avançar se realmente soubesse os detalhes do domínio em que está trabalhando. Você provavelmente não vai saber cada regra de negócio – e nem é sua função. Mas você pode, pelo menos, aprender o básico. Muitas das melhores pessoas que trabalham com software, com quem trabalhei durante anos, tornaram-se mais especialistas em seus domínios do que até alguns de seus clientes. Isso resulta em produtos melhores. Alguém que é ignorante em domínio vai deixar passarem erros bobos – erros que um conhecimento básico poderia evitar. Além disso, ele trabalharia mais lentamente (e custaria mais à empresa) do que o desenvolvedor que entende do negócio.

Para nós, desenvolvedores de software, a intenção de Lao Tzu pode ser igualmente interpretada como “Peça um peixe; coma por um dia. Peça a alguém para ensinar-lhe a pescar; coma para a vida toda”. Melhor ainda, não **peça** para ser ensinado – vá aprender você mesmo.

## Faça algo

- 1) **Como e por quê?** – tanto enquanto você está sentado lendo este livro, ou na próxima vez em que estiver no trabalho, pense sobre as partes do seu trabalho que você entende por completo. Você pode se fazer duas perguntas extremamente úteis sobre qualquer assunto para mergulhar fundo nele: **como isso funciona?** E **por que isso tem que acontecer?**

Você pode não conseguir responder as perguntas, mas o simples ato de fazê-las vai levar sua mente a um novo quadro e pode gerar um nível mais alto de consciência sobre seu ambiente de trabalho. Como o servidor IIS acaba passando requests para minhas páginas ASP.NET? Como eu devo gerar essas interfaces e scripts de deploy para minhas aplicações EJB? Como meu compilador lida com linkagem dinâmica ou estática? Como calculamos a taxa de modo diferente se um comprador mora em outro estado?

Claro, a resposta para qualquer dessas perguntas levará a uma outra potencial oportunidade de se fazer a pergunta mais uma vez. Quando você não puder mais avançar na árvore do **como** e do **porquê**, você provavelmente terá ido longe o suficiente.

- 2) **Dica** – escolha uma das mais críticas, porém negligenciadas, ferramentas da sua caixa de ferramentas e foque nela. Talvez seja seu sistema de controle de versão, talvez uma biblioteca que você use bastante mas procurou apenas superficialmente, ou talvez o editor que você usa pra programar.

Quando você escolher a ferramenta, reserve um pequeno período de tempo todos os dias para aprender **uma coisa nova** sobre ela que vai lhe tornar mais produtivo ou lhe dar mais controle sobre seu ambiente de desenvolvimento. Você pode, por exemplo, escolher dominar o GNU Bourne Again Shell, também conhecido como bash. Durante um desses momentos em que sua mente começa a fugir do que deve ser feito, em vez de ir para o Slashdot ou Facebook, você poderia procurar na internet por dicas de bash. Em um minuto ou dois, você deve encontrar **algo** útil que você não sabia sobre como usar o shell. Claro, agora que tem um novo truque, você pode mergulhar em suas entranhas com uma série de **como** e **porquês**.



## CAPÍTULO 12

# Aprenda como os negócios realmente funcionam

No capítulo anterior, nós discutimos a importância de fazer uma escolha intencional sobre o domínio de negócio no qual você trabalha. Conhecimento de domínio, sendo na melhor das hipóteses um diferencial para empregos e na pior das hipóteses um estraga-prazeres, não é algo que você deva subestimar. Antes de fazer um investimento para aprender os prós e contras de um domínio de negócios, você deveria se certificar de que está investindo na coisa certa para você e para a situação do mercado.

Mas há uma parte do conhecimento que não é nem técnica nem de domínio específico e não será ultrapassada em breve: o básico de finanças. Independentemente da sua linha de negócio, se é de manufaturas, assistência médica, sem fins lucrativos, ou uma instituição educacional, ainda se trata de **negócios**. E isso é por si só uma área de conhecimento que alguém pode – e até deve – aprender.

Eu me lembro de quando eu era um jovem programador indo para reuniões

da equipe, meus olhos vidrados enquanto algum líder, com quem eu nunca tinha trabalhado diretamente, mostrava um monte de números que eu acreditava serem completamente irrelevantes para mim. *Eu só quero voltar e terminar a funcionalidade que estou trabalhando*, eu choramingava para mim mesmo. Meus colegas sentavam-se juntos, parecendo uma fila de crianças contorcidas numa longa viagem de carro. Nenhum de nós entendia o que estava sendo apresentado, e nenhum de nós se importava. Nós culpávamos os gerentes incompetentes que haviam convocado a reunião, pelo que sentíamos que era uma completa perda de tempo.

*Você não pode criativamente ajudar em um negócio antes de saber como ele funciona.*

—

Olhando para trás, eu percebo como nós éramos bobos. Trabalhávamos para uma empresa, e nossa função era contribuir para ou fazer, ou economizar dinheiro para aquela empresa. Ainda assim não entendíamos o básico de como um negócio se torna lucrativo. Pior ainda, nós não pensávamos que era nosso dever saber. Nós éramos programadores e administradores de sistema. Pensávamos que nossos trabalhos eram estritamente sobre aqueles tópicos aos quais nos devotávamos. Contudo, como nós supostamente poderíamos contribuir **criativamente** para que aquele negócio fosse rentável, se nem mesmo entendíamos como isso **funcionava**?

O uso da palavra **criativamente** no parágrafo anterior é a chave. É plausível ter a visão de que somos, de fato, especialistas em TI e que é para isso que somos pagos. Dados os projetos e liderança corretos, nós deveríamos nos esforçar em tarefas que contribuíssem para a empresa. Não precisamos entender totalmente como uma empresa opera para dar-lhe valor.

Mas para **criativamente** adicionar valor, é preciso uma compreensão mais completa do ambiente de negócios em que você trabalha. No mundo dos negócios, ouvimos a frase **ponto de partida** o tempo todo. Quantos de nós realmente entendem o que é o ponto de partida e o que contribui para ele? Mais importante, quantos de nós realmente entendem como **nós** contribuímos para o ponto de partida? A organização é um centro de custo ou de lucros (você adiciona ou retira do ponto de partida)?

Conhecer a conduta financeira – e linguagem – da sua empresa vai lhe proporcionar a habilidade de fazer mudanças significativas, em vez de tentar acertar no escuro em coisas que lhe parecem intuitivamente certas.

## Faça algo

- 1) Procure um livro básico sobre negócios, e trabalhe com ele. Uma dica para encontrar um bom livro com uma visão geral é procurar por livros sobre MBA. Um desses livros que eu acho particularmente útil (e agradavelmente curto) é o *The Ten-Day MBA* [9]. Você realmente pode concluí-lo em dez dias. Ele não custa caro.
- 2) Peça a alguém para orientá-lo na área de finanças da sua empresa e expicar como funcionam as coisas (se isso for uma informação que sua empresa não se importe em compartilhar com seus funcionários).
- 3) Explique para eles de volta.



## CAPÍTULO 13

# Encontre um mentor

Uma coisa que a cultura musical do jazz realmente acertou foi a prática de mentoria. No mundo do jazz, é comum os jovens músicos procurarem os mais experientes, que os carregarão sob suas asas e passarão seus conhecimentos de jazz. Mas isso não para por aí. Esses músicos mais velhos geralmente servem como um conselheiro de carreira, e da vida toda. Em troca, os músicos jovens são devotamente fiéis, construindo uma rede de apoio de apreciadores fanáticos em torno de seus mentores.

Contatos são feitos e músicos são contratados todos os dias por meio desses relacionamentos. A sociedade do jazz criou uma cultura auto-organizada e um conjunto de costumes em torno do relacionamento com o mentor. É um sistema que funciona tão bem que você suspeitaria que ele é guiado por algum tipo de entidade organizadora.

*É OK depender de alguém. Só tenha certeza de que é a pessoa certa.*

—

No mundo profissional tradicional (e especificamente na área de TI), nós estamos menos propensos a pedir ajuda uns aos outros. Depender dos outros é fre-

quentemente visto como um sinal de fraqueza. Temos medo de admitir que não somos perfeitos. Tudo é competição. Apenas os fortes sobrevivem, e isso é tudo. Infelizmente, isso leva a um sistema de mentoria extremamente subdesenvolvido. Se tivéssemos que perguntar a um grupo de músicos de jazz: “Quem é seu mentor?”, a maioria deles teria uma resposta. Agora faça a mesma questão para programadores. Nos Estados Unidos, eles provavelmente responderiam “O quê?”.

Nem sempre foi assim por aqui. A história do oeste inclui um próspero sistema de mentoria profissional, que vem desde a Idade Média. A abordagem do artesariado ao treinamento profissional era ainda mais forte e mais formalizado que o sistema que evoluiu na cena musical. Pessoas jovens começavam suas carreiras profissionais como aprendizes de respeitáveis mestres artesãos. Elas trabalhavam em troca de um salário nominal e pelo privilégio de aprender com o mestre, cuja obrigação era treinar os discípulos para criarem coisas seguindo a mesma tradição (e qualidade) que a sua própria.

O primeiro e mais importante propósito de um mentor é ser um modelo a seguir. É difícil saber o que é possível ser feito até que você veja alguém que possa ir além dos limites que você conhece. Um modelo estabelece o padrão do que “bom” significa. Se você se imaginasse como um jogador de xadrez, por exemplo, só o fato de poder vencer as pessoas de sua família imediata pode ser muito bom. Contudo, se você jogasse contra um competidor de campeonatos, você descobriria que o xadrez é um jogo muito mais denso do que você jamais pensava. Se você fosse jogar contra um grande mestre, você teria outra revelação. Se você continuasse jogando e derrotando apenas os membros de sua família, você poderia ficar com a ideia de que você é **realmente bom** no xadrez. Sem um modelo a seguir, não há incentivo para melhorar.

Um mentor também pode dar estrutura para seu processo de aprendizagem. Como você viu no capítulo anterior, você tem uma enorme quantidade de escolhas para fazer sobre as tecnologias e domínios em que vai investir. Às vezes, muitas possibilidades podem o travar. Com certeza, é melhor estar se movendo em uma direção do que ficar sentado parado. Um mentor pode ajudar a escolher no que focar suas energias.

Quando eu comecei minha carreira, trabalhando com suporte, eu conheci um santo chamado Ken, que era um dos administradores da rede da nossa universidade. Ele veio me salvar de um grande problema com a rede NetWare do nosso campus, que estava atrapalhando os alunos que tentavam usar o laboratório de informática. Quando o incitei a me dar direções sobre como me tornar mais bem informado e autossuficiente, ele me deu uma simples dica: mergulhe nos serviços de diretório,

aprenda uma distribuição UNIX, e domine uma linguagem de script.

Ele escolheu três habilidades para eu aprender, dentre as infinitas disponíveis. E, com a confiança que essa pessoa, quem eu considerava ser um mestre, as prescreveu, eu me dispus a aprender essas três habilidades. Minha carreira desde então tem sido construída sobre o alicerce desses conhecimentos, todos os três, os quais ainda são completamente relevantes para tudo. Eu faço isso. Não é que o caminho que Ken me deu foi a resposta absolutamente certa – não há respostas absolutamente certas. O importante é que ele estreitou a longa lista de habilidades que eu **poderia** estar aprendendo, transformando-a em uma curta lista de habilidades que eu, de fato, **aprendi**.

O mentor também serve como alguém de confiança que pode observar e julgar suas decisões e seu progresso. Se você é um programador, você pode mostrar seus códigos e obter indicadores. Se você está planejando fazer uma apresentação no escritório ou em alguma reunião de um grupo de usuários locais, você pode mostrá-la de antemão para seu mentor para receber feedback. Um mentor é alguém em que você pode confiar o suficiente para perguntar “O que deveria ser diferente em mim enquanto profissional?”, porque você sabe que ele não vai apenas criticá-lo, mas sim ajudá-lo a melhorar.

Finalmente, assim como no jazz, você não apenas cria uma ligação pessoal e de responsabilidades para com seu mentor, mas o inverso também acontece. Se meu papel, em um relacionamento, é ajudar alguém, eu invisto no sucesso dessa pessoa. Estou ajudando alguém ao longo de sua carreira, por um caminho que eu acredito ser o correto. Portanto, se o caminho leva ao sucesso, o sucesso é meu também.

Isso cria incentivos por parte do mentor para que seus orientandos tenham sucesso. Tipicamente, sendo mais experiente e já bem sucedida, uma pessoa em tal papel teria o respeito de uma significativa rede de pessoas. O mentor se torna um elo que reforça a conexão entre você e sua rede. A importância desse tipo de conexão não pode ser subestimada. Afinal, a frase “Não é o que você sabe. É quem você conhece” não é um clichê sem motivo.

O grau de formalidade num relacionamento de mentor não é importante. Ninguém precisa explicitamente pedir a alguém para ser seu mentor (embora não seja definitivamente uma coisa ruim se você o fizer). Na verdade, seu mentor pode nem **saber** que está fazendo esse papel para você. O que importa é que você tenha alguém de confiança e que admire, que possa proporcionar uma orientação à sua carreira e ajudar a afiar seu ofício.

## Faça algo

- 1) **Tutorie a si mesmo** – todos nós queremos ter alguém para nos orientar, mas a realidade é que nem sempre vamos poder encontrar alguém a quem possamos dar esse papel. Eis um jeito de fazer uma automentoria.

Pense na pessoa de sua área que você mais admira. Muitos de nós já possuem uma pequena lista pronta, pegando de algum momento de nossas carreiras. Pode ser alguém com quem tenha trabalhado, ou pode ser alguém cujo trabalho é admirado. Liste os dez atributos mais importantes desse modelo a seguir. Escolha os atributos que são a razão de você tê-lo escolhido para esse papel. Tais atributos podem ser de áreas específicas de habilidade, como amplitude de tecnologia, ou a profundidade de conhecimento sobre algum domínio. Ou, eles podem ter características mais pessoais, como a habilidade de deixar os membros da equipe tranquilos, ou a de ser um palestrante envolvente.

Agora, ranqueie essas qualidades em ordem de importância, sendo que 1 é o menos importante e 10, o mais importante. Você acabou de criar e destilar uma lista de atributos que você acha admiráveis e importantes. Esses são os caminhos nos quais você deve se empenhar para emular o modelo escolhido. Mas como escolher em qual focar primeiro?

Adicione uma coluna à lista, e para cada item na lista, imagine como seu modelo iria avaliar **você** em uma escala de 1 a 10 (10 sendo o melhor). Tente realmente se colocar na mente de seu modelo e observar a si mesmo como se fosse uma terceira pessoa.

Quando você tem os atributos, o ranking, e suas próprias avaliações, em uma coluna final, subtraia sua classificação em cada fileira do nível de importância que você deu à coluna anterior. Se você ranqueou algo como 10 para o atributo mais importante de seu modelo, e sua própria avaliação foi 3, isso lhe dá uma prioridade final de valor 7. Tendo preenchido esta coluna completamente, separando em ordem descendente, você terá uma lista das 10 áreas priorizadas em que você deve melhorar.

Comece com os primeiros dois ou três itens, e formule uma lista de tarefas concretas que você pode **começar a fazer agora** para melhorar a si mesmo.



## CAPÍTULO 14

# Seja um mentor

Se você quer realmente aprender alguma coisa, tente ensiná-la para outra pessoa. Não tem jeito melhor de aperfeiçoar seu entendimento de algo do que se forçar a passá-lo para outras pessoas, para que **eles** possam entender. O simples ato de falar é um elixir conhecido para clarear a mente. Falar para bonecos e outros objetos inanimados é uma forma de resolver problemas bastante conhecida do folclore do desenvolvimento de software.

*Para descobrir se você realmente sabe algo, tente ensinar para outra pessoa.*

---

Eu vi Martin Fowler dar uma palestra em uma sala de desenvolvedores em Bangalore, na qual ele dizia que toda vez que ele realmente quer aprender algo, ele escreve sobre isso. Martin Fowler é um conhecido desenvolvedor de softwares e autor. Podemos dizer que ele é um dos mais bem conhecidos e influentes **professores** que essa indústria tem a oferecer, se considerarmos que seu papel enquanto autor é aquele de um professor remoto e mentor.

Aprendemos ensinando. É irônico, porque esperamos que um professor já saiba de tudo. Claro, eu não quero dizer que podemos aprender completamente algo ensinando outra pessoa. Mas conhecer os fatos não é o mesmo que entender suas causas e ramificações. É o tipo de conhecimento mais profundo que desenvolvemos ao ensinar a alguém.

Procuramos analogias para explicar conceitos complexos, enquanto tentamos entender porque algumas analogias que parecem funcionar, na prática não funcionam. Já outras que parecem menos efetivas, acabam sendo melhores.

Quando você ensina, você precisa responder questões nas quais você pode nunca ter pensado. Por meio do ensino, nós limpamos os cantos sujos do nosso conhecimento, que raramente são expostos.

Portanto, assim como você pode se beneficiar encontrando um mentor, você pode se beneficiar **sendo** um mentor para outra pessoa.

Mentoria traz efeitos sociais positivos também. Um grupo de mentores e seus orientandos cria uma rede social firme e poderosa. A ligação mentor-orientando é bastante forte, de modo que os laços dessa rede profissional seja mais forte do que aqueles relacionamentos passivos. Quando você está em uma relação de mentoria com alguém, você forma uma aliança um com o outro. Uma rede desse tipo é um ótimo lugar para fazer circular problemas difíceis ou procurar por trabalho.

Você também não deveria subestimar que simplesmente **faz sentir bem** ajudar as pessoas. Se você pode manter em mente os interesses de outros, terá realmente feito algo altruísta com suas habilidades. No incerto ambiente econômico de hoje, **ajudar** de fato alguém é um trabalho do qual você não pode ser demitido. E é pago com uma moeda que não desvaloriza com inflação.

Você não encontra um orientando saindo por aí e se autodeclarando um guru, mas sim sendo conhecido e disposto a compartilhar seu conhecimento pacientemente. Não se preocupe se você não é um expert absoluto em algum tópico. As chances são de que há **algo** em que você tem experiência e que o qualificaria para ajudar alguém menos experiente. Encontre esse algo e comece a ser prestativo.

Você pode, por exemplo, ter feito uma considerável quantidade de trabalhos em PHP. Você poderia ir até seu grupo local de usuários PHP e oferecer ajuda aos usuários menos experientes com seus problemas específicos. Ou, se você não tem um lugar disponível para fornecer mentoria presencial, você poderia simplesmente começar a responder perguntas em um fórum de discussões online ou em um canal IRC, ou ainda ajudar pessoas a fazer debug nos problemas de suas aplicações. Entretanto, mantenha em mente, que mentoria diz respeito a pessoas. Um relacionamento

de mentoria online nunca pode ser comparado ao que acontece entre dois seres humanos no mesmo local.

Você não precisa montar um relacionamento de mentoria formal para obter esses benefícios. Apenas comece a ajudar pessoas, e o resto virá automaticamente.

### **Faça algo**

- 1) Procure alguém para carregar sob suas asas. Você pode encontrar alguém mais jovem e menos experiente em sua empresa, talvez um estagiário. Ou, você poderia conversar com os departamentos de ciência da computação ou sistemas de informação em sua universidade local e se voluntariar para orientar um estudante da faculdade.
- 2) Encontre um fórum online e escolha um tópico. Comece a ajudar. Torne-se conhecido por seu desejo e por sua habilidade de pacientemente ajudar pessoas a aprender.



## CAPÍTULO 15

# Pratique, pratique, pratique

Quando eu estudava música, eu passava longas noites no prédio de música da minha universidade. Pelas finas paredes das salas de estudo da universidade, eu ficava imerso em algum dos sons musicais mais feios imagináveis. Não é que os músicos da minha escola eram ruins. Muito pelo contrário. Mas eles estavam praticando.

Quando você treina música, ela não deveria soar boa. Se você sempre soa bem durante as sessões de estudo, significa que você não está estendendo seus limites. É para isso que serve a prática. O mesmo é válido para os esportes. Atletas se esforçam até o limite durante os exercícios, de modo que eles possam **expandir** tais limites para as performances reais. Eles deixam a “feiura” acontecer nesse momento de prática – e não quando eles estão realmente trabalhando.

Na indústria da computação, é comum encontrar desenvolvedores que forçaram seus limites. Infelizmente, esse é geralmente um caso de um desenvolvedor sendo subqualificado para as tarefas que ele assumiu. Nossa indústria tende a praticar no emprego. Você consegue imaginar um músico profissional subindo ao palco e replicando os sons inarticulados das salas de estudo da minha universidade? Isso não

seria tolerado. Músicos são pagos para **apresentar** em público – não para praticar. Similarmente, um lutador de artes marciais ou um boxeador estressando o-se até a fadiga durante os combates não iria muito longe no esporte.

Como indústria, nós devemos criar tempo para a prática. Nós do ocidente frequentemente tomamos partido de programadores domésticos, baseados na qualidade relativamente alta do código que eles produzem versus a de times offshore. Se vamos tentar competir com base na qualidade, temos que parar de tratar nosso emprego como uma sessão de prática. Temos que **investir o tempo** em nossa profissão.

Muitos anos atrás, eu comecei a experimentar exercícios de programação modelada após minhas sessões de prática de música. A regra número um era que o software que eu estava desenvolvendo não poderia ser algo que eu desejava usar. Eu não queria limpar os cantos na pressa, por conta de um objetivo final. Então eu escrevia softwares que não eram úteis.

Eu não me preocupava com os cantos, mas estava frustrado ao descobrir que muitas das ideias que tive durante a prática não estavam funcionando. Embora eu estivesse tentando fazer tão bem quanto se fosse pra valer, na medida do possível, os designs e códigos que eu criava não eram tão elegantes quanto eu queria que fossem.

Olhando para trás agora, vejo que o estranho sentimento que eu tive com essas experiências era um **bom sinal**. Meu código não estava completamente desprovido de momentos brilhantes. Mas eu estava esticando meus músculos mentais e construindo meus pedaços de código. Assim como para tocar o saxofone, se eu sentasse para praticar e só saíssem sons bonitos, eu saberia que não estava praticando. Da mesma forma, se eu sentasse para praticar os códigos e só viessem códigos elegantes, eu estaria provavelmente em algum lugar próximo ao **centro** das minhas atuais capacidades, em vez de no limite, aonde uma boa sessão de treinamento deveria me levar.

*Pratique nos seus limites.*

—

Portanto, como você sabe o que praticar? O que estende seus limites? O assunto de como praticar poderia facilmente dar um livro só pra isso. Como um começo, eu vou pegar emprestado mais uma vez da minha experiência como um músico de jazz. Eu dividia o estudo de jazz nas seguintes categorias (simplificadas para os não músicos):

- Físico / coordenação;

- Leitura à primeira vista;
- Improvisação;

Isso deve servir como um quadro de **uma das maneiras** de como um desenvolvedor deve praticar.

**Físico / coordenação:** músicos devem praticar os aspectos técnicos de seus instrumentos. Produção de som, coordenação física (fazer os dedos se moverem com agilidade, por exemplo), velocidade e precisão. Tudo isso é importante.

O que nós, desenvolvedores de software, temos desses fundamentos de música? E quanto aos cantos sujos de sua primeira linguagem de programação que você raramente visita? Por exemplo, a linguagem que você usa suporta expressões regulares? Expressões regulares são um recurso extremamente poderoso e subutilizado de muitos ambientes de programação. A maioria dos desenvolvedores não as usa quando poderia, porque não possui o nível de habilidade necessário para serem produtivos com elas. Como resultado, um monte de código de manipulação de *strings* é criado e, portanto, exige maior manutenção.

As mesmas regras se aplicam para as APIs de sua linguagem. Se você não tiver as várias ferramentas do ambiente sob seus dedos (como dizem os músicos), é pouco provável que você as use quando elas realmente o poderiam ajudar. Tente verdadeiramente mergulhar em, por exemplo, como programação *multi-threaded* funciona na sua linguagem. Ou, e quanto às bibliotecas, APIs pra programar aplicações em rede, ou mesmo o grupo de funcionalidades disponíveis para trabalhar com coleções ou listas? A maioria das linguagens de programação modernas oferece ricas e poderosas bibliotecas em todas essas áreas, mas desenvolvedores tendem a aprender uma pequena parcela, com a qual eles podem, menos eficientemente, escrever o mesmo código que eles teriam escrito se tivessem dominado o todo o conjunto de ferramentas.

**Leitura à primeira vista:** especialmente como um músico de estúdio, a habilidade de ler e executar música quase perfeitamente na primeira tentativa é suprema para um profissional. Uma vez eu toquei saxofone em um jingle para a Blockbuster (a videolocadora). Eu toquei ambas as partes principais e secundária de saxofone alto em uma música que tinha um ritmo bem rápido. Quando a fita começou a tocar, era a primeira vez que eu via aquela música. Tocamos uma vez a parte principal e outra a parte secundária. Qualquer erro, e a banda inteira teria de fazer tudo de novo – e isso aumentaria o custo de horas no estúdio.

Para desenvolvedores, o que significaria ser capaz de ler um código ao vê-lo? Ou especificações de requisitos ou designs? Um excelente lugar para encontrar novos códigos com os quais praticar é a comunidade open-source. Você tem algum biblioteca open-source de que goste? E se você tentar adicionar uma nova funcionalidade nela? Vá procurar na lista de tarefas a fazer dessa biblioteca, e force a si mesmo a dedicar uma considerável parcela de tempo para implementar isso (ou, pelo menos, determinar o que é necessário para implementá-la).

Depois de escolher uma biblioteca, baixe seu código-fonte, e comece a explorar. Como você sabe onde procurar? Quais truques você usa para se achar em uma base de código relativamente grande? Por onde começa?

Esse é um exercício que você pode fazer com frequência e em curtos períodos de tempo. Você não necessariamente **deve** implementar a funcionalidade. Apenas a use como um ponto de partida. O verdadeiro objetivo é entender o que você está visualizando o mais rápido possível. Certifique-se de variar os softwares com os quais você trabalha. Tente diferentes tipos de software em diferentes estilos e linguagens. Anote os pontos que facilitam ou dificultam a você se encontrar no código. Quais padrões você está desenvolvendo que o ajudam a trabalhar no código? Quais rastros você deixa para si mesmo para o ajudar a navegar conforme você sobe ou desce nas camadas dessa funcionalidade?

**Improvisação:** improvisação é pegar alguma estrutura ou restrição e criar algo novo, em tempo real, sobre essa estrutura. Como programador, eu me descobri fazendo mais improvisos em momentos de estresse. “Oh não! O servidor da rede sem fio caiu, e nós estamos perdendo pedidos!” É aí que algumas das mais criativas improvisações acontecem. É aí que você faz coisas loucas, como recuperar dados perdidos manualmente resubmetendo pacotes sobre uma rede sem fio a partir de um arquivo binário. Ninguém o pediu pra fazer essas coisas, especialmente não no calor do momento. Esse tipo de habilidade de programação afiada e rápida pode ser como um poder mágico quando usada no momento certo.

Uma ótima maneira de afiar a mente e melhorar suas habilidades de improvisar códigos é praticar com restrições autoimpostas. Pegue um simples programa, e tente escrevê-lo com essas restrições. Meu exercício favorito é escrever um programa que mostra a letra da velha canção “*99 Bottles of Beer on the Wall*”. Como você poderia escrever tal programa sem atribuir variáveis? Ou, o quão curto você consegue escrever o programa, de modo que ele ainda exiba a letra corretamente? Para uma restrição adicional, quão rápido você pode escrever esse programa? Que tal praticar problemas pequenos e difíceis com um *timer*?



Essa é apenas uma perspectiva limitada sobre como praticar. Você pode pegar exemplos de qualquer área, das artes visuais até prática de religião. O que importa é encontrar **suas** próprias necessidades e certificar-se de que você não está praticando durante suas apresentações (no emprego). **Você** deve reservar um momento para praticar. É **sua** responsabilidade.

## Faça algo

- 1) *TopCoder* – <http://topcoder.com> é um renomado site de competição de programação. Você pode registrar uma conta e competir online por prêmios. Mesmo se você não se interessa em competir com outros, o TopCoder oferece uma sala de treinamento com uma grande coleção de exercícios que podem ser usados como uma excelente base para suas sessões de estudo. Vá se registrar e faça uma tentativa.
- 2) *Code Kata* – Dave Thomas, um dos Pragmatic Programmers (o editor deste livro no original), transformou a ideia de praticar código em algo... bem, pragmático. Ele criou uma série do que ele chama de *Code Kata*, que são pequenos exercícios que estimulam o pensamento que os programadores podem fazer na linguagem de sua escolha. Cada *kata* enfatiza uma técnica específica ou processo de pensamento, provocando uma flexão concreta de seus músculos mentais.

No blog <http://codekata.pragprog.com>, Dave disponibiliza gratuitamente diversos *katas*. Lá, você também encontra links para uma lista de e-mail e para as soluções de outros usuários, junto da discussão sobre como foram resolvidos.

Seu desafio: trabalhe com os *katas*. Mantenha um diário (ou talvez um blog) das suas experiências com eles. Quando terminar, escreva seu **próprio** *kata* e compartilhe.



## CAPÍTULO 16

# O jeito que você faz

“Desenvolver software” não é uma coisa, um substantivo. Do contrário, “desenvolver software” é um *termo*; é o *processo* de criar uma coisa. Quando estamos codificando, é tão importante focar no processo utilizado quanto focar no produto que está sendo desenvolvido. Se você tirar os olhos do processo, estará arriscando entregar atrasado, ou entregar o produto errado, ou mesmo não entregar. Esses resultados tendem a ser desprezados pelos clientes.

Felizmente, muita atenção tem sido dada ao processo de criar um bom software (e produtos em geral). Muito dessas técnicas foram catalogadas em um grupo de *metodologias*, que são tema de vários livros.

Infelizmente, a maioria dos desenvolvedores não se beneficia dessa boa informação. Para a maior parte das equipes, o processo é uma reflexão tardia ou algo imposto de cima. O termo *metodologia*, em suas mentes, se tornou sinônimo de papelada e longas e insignificantes reuniões. Muito frequentemente, a metodologia é algo que seus chefes impõem.

Os gerentes intuitivamente sabem, que eles precisam seguir *algum tipo* de pro-

cesso, mas geralmente não sabem sobre as opções que agora estão disponíveis. Como resultado, eles tiram a poeira dos mesmos processos que lhes foram impostos na década de 80, os envolvem com uma fita de chavões compatíveis (a fita com tons pastéis Agile é uma boa escolha nesse momento), e passam a prática para seus times. E, ao menos que alguém quebre o ciclo realmente pesquisando o que funciona e o que não, o mesmo processo acontecerá outras vezes, conforme os desenvolvedores da equipe tornam-se gerentes.

Talvez você esteja pensando que deve existir um jeito melhor de desenvolver software. E para a maioria das equipes, existe.

Se você é um programador, testador, ou designer de software, você pode pensar que o processo do desenvolvimento não é sua responsabilidade. Infelizmente, geralmente não é responsabilidade de *ninguém*. Se for designado a alguém, pode cair no buraco negro, também conhecido como “grupo de processo”. A verdade é que, para um processo de software ter alguma chance de ser implementado com sucesso, ele deve ser compreendido pelas pessoas que o estão usando – pessoas como você.

A melhor maneira de compreender e dominar esses processos é ajudar a implementá-los. Se sua empresa não possui um processo, pesquise metodologias que podem funcionar para vocês. Tenha almoços com sua equipe para discutir os atuais problemas de desenvolvimento, e sugira que adotar um processo padrão pode mitigá-los. Reúna um plano de colocar em ação o processo escolhido na sua empresa e obtenha a aprovação de todos. Então comece a implementar seu plano.

*Se você quer sentir que é dono de um processo, ajude a implementá-lo.*

—

Alternativamente, você talvez trabalhe em um ambiente em que o processo é passado de cima para baixo. No momento em que as ideias chegam ao time de desenvolvimento, as práticas foram diluídas e reinterpretadas, até o ponto em que se tornam irreconhecíveis das originais. Lembra-se da brincadeira de telefone-sem-fio?

Mais uma vez, essa é uma oportunidade de tomar a iniciativa. Pesquise a metodologia que introduziram, e ajude a interpretar o que ela realmente significa, tanto para seu time como para sua gerência. Você não poderá contestar que um processo foi imposto, então você deve fazê-lo funcionar fazendo o correto.

O mundo da metodologia pode rapidamente começar a parecer uma concha oca de chavões. Mas, como podem ser chavões compatíveis, você sempre pode aprender *alguma coisa* do estudo de um processo de software – mesmo se essa coisa é o

que *não* fazer. Se você entender bem do processo de software, poderá fazer uma argumentação mais confiável sobre como sua equipe deveria estar trabalhando.

Mesmo com a abundância de metodologias para escolher, é bem difícil que você trabalhe para uma empresa que implemente uma delas inteiramente. Tudo bem. O melhor processo a se seguir é o que torna sua equipe mais produtiva e que resulta em melhores produtos.

### **METODOLOGIAS: NÃO SÓ PARA GEEKS**

Embora o gerenciamento de projetos não seja necessariamente ligado à metodologia de desenvolvimento de software, você pode dar de cara com as técnicas de gerenciamento de projetos da sua empresa. Numerosas metodologias de gerenciamento de projetos estão em uso por toda indústria de tecnologia. Provavelmente, o mais notável é o *Project Management Book of Knowledge* (<http://www.pmi.org>), do *Project Management Institute*, com seu programa de certificação, que é bastante reconhecido.

*Six Sigma* (<http://www.isixsigma.com>) é outra metodologia de qualidade não específica de software. Dirigida por empresas como General Electric e Motorola, a abordagem da Six Sigma enfatiza a medição e a análise dos processos e produtos para conduzir à satisfação do cliente e à eficiência. Embora não sejam específicas para desenvolvimento de software, a abordagem rigorosa e metódica da Six Sigma oferece várias lições que são diretamente aplicáveis à sua função como programador.

O único modo de descobrir essa epifania reveladora é estudar as opções disponíveis, escolher as peças que fazem sentido para você e sua equipe, e continuamente refiná-las baseando-se nas suas experiências.

Em último caso, se você não pode implantar o processo, você não consegue fazer o produto. É muito mais fácil encontrar alguém que pode fazer o software funcionar do que encontrar alguém que faça o *making of* do software funcionar. Portanto, adicionar conhecimento do processo de desenvolvimento de software para seu arsenal apenas vai ajudá-lo.

### **Faça algo**

- 1) Escolha uma metodologia de desenvolvimento de software, e escolha um livro, comece a ler sites e inscreva-se em uma lista de e-mails. Olhe para a metodologia

com um olho crítico. Quais você acha que seriam seus pontos fortes e fracos? Em seguida, faça o mesmo para outra metodologia. Contraste suas vantagens e desvantagens. Como você poderia combinar suas abordagens?

## CAPÍTULO 17

# Nos ombros dos gigantes

Quando eu tocava jazz, eu ficava muito tempo ouvindo música. Eu não apenas colocava música de fundo enquanto eu lia ou dirigia. Eu realmente *ouvía* a música. Se a improvisação no jazz significa executar o que você escuta por cima dos acordes de uma música, então realmente ouvir a música é uma fonte de inspiração e conhecimento do que funciona e do que não. O que soa ótimo é apenas o que se encaixa lá.

A enorme história do jazz serve como uma incrível fonte de conhecimento, para qualquer um com a habilidade de ouvi-las. Ouvir música, portanto, não é uma atividade passiva para um músico de jazz. É um estudo. Mais ainda, a habilidade de entender o que você está ouvindo é uma proficiência que se desenvolve ao longo do tempo. Meu círculo de amigos músicos realmente faziam esse tipo de audição explicitamente. Tínhamos festas para ouvir músicas, em que um grupo de músicos geeks de jazz se sentavam e ouviam música, para depois discuti-la. Às vezes, jogávamos “*Nomeie o improvisador*”, no qual um de nós tocava uma gravação de um solo improvisado e o resto deveria adivinhar, baseado no estilo, quem havia gravado aquele

improviso.

Nós do mundo do jazz não éramos especiais, claro. Compositores de música erudita fazem a mesma coisa. Assim como romancistas e poetas. Assim como escultores e pintores. Estudar a obra dos mestre é uma parte essencial para se tornar um mestre.

Quando ouvíamos as gravações de jazz, discutíamos os artifícios musicais que os improvisadores usavam para comunicar seus pontos musicais. “Uau! Você ouviu como ele começou contornando no final da estrutura?” ou “Foi realmente estranho o jeito que ele estava tocando atrás do compasso”. Essas discussões nos ajudavam a investigar e descobrir truques que podíamos levar para tentar na nossa próxima sessão de improvisação.

*Minere os códigos existentes para ter insights.*

—

Design de software e programação têm muito em comum com as artes, guardadas as devidas particularidades. Podemos investigar uma base de código grande para encontrar padrões e truques. O movimento dos *design patterns* (veja *Design Patterns* [3]) é focado na descoberta e na documentação de soluções reusáveis para problemas comuns de desenvolvimento de software. *Design patterns* formalizaram o estudo de códigos existentes, tornando a prática acessível a um grande número de profissionais de software. Ainda assim, eles abrangem apenas um pequeno subconjunto dos tipos de técnicas que podemos aprender através da leitura de código.

Como outros programadores resolvem problemas? Como outros usam estrategicamente variáveis, funções e nomenclatura? Se eu quisesse implementar o protocolo Jabber, como eu poderia fazê-lo? Quais maneiras criativas eu poderia achar para lidar com a comunicação entre processos, entre UNIX e sistemas Windows? Esses são o tipo de questões que você pode responder através do estudo de códigos existentes.

*Use código existente para refletir sobre suas próprias capacidades.*

—

Ainda mais importante que encontrar soluções para problemas específicos é o uso de códigos de outros desenvolvedores como uma lente de aumento para inspecionar nossos próprios estilos e capacidades. Assim como escutar uma gravação de John Coltrane sempre me lembrava de onde eu me estava na escala de habilidades de um saxofonista; ler o código de um *ótimo* desenvolvedor possui um efeito humilhante. Entretanto, não se trata apenas de se sentir humilhado. Conforme você lê



o código, você encontrará coisas que nunca teria feito. Encontrará coisas que você pode nunca ter imaginado. Por quê? Em que o desenvolvedor estava pensando? Quais foram suas motivações? Você pode até aprender com códigos ruins neste tipo de exploração crítica de autoconhecimento a partir do trabalho de alguém.

O ato de aprender a partir do trabalho que veio antes de você funciona bem no mundo das artes, porque não há recursos escondidos para uma pintura ou uma peça de música. Se você pode ouvir a música ou ver a obra de arte, você pode aprender dela. Felizmente, como desenvolvedores de software, temos acesso a praticamente uma infinita série de softwares existentes no mundo *open source*.

Há tantos *open sources* disponíveis que seria impossível realmente ler todos. Definitivamente há projetos open source ruins, mas também há alguns *ótimos*. Existe código open source disponível para implementar quase qualquer tarefa que pode ser feita com software em quase todas as linguagens de programação disponíveis.

Conforme você olha para esse código com um olhar crítico, você começa a desenvolver seus próprios gostos, assim como seria com música, arte ou literatura. Diversos estilos e artifícios vão entretê-lo, impressioná-lo, deixá-lo bravo, e (eu espero) *desafiá-lo*. Se você realmente estiver procurando por eles, encontrará algum truque que o torne mais produtivo com relação a paradigmas de design, mudando completamente o jeito como você resolve problemas. Assim como nas artes, estudando e aprendendo com os hábitos dos outros, você desenvolverá seu próprio estilo distintivo de desenvolvimento de software.

Um efeito colateral positivo de leitura de código é que você vai aprender mais sobre o que já existe. Quando você tem um novo problema que precisa de solução, você deve se lembrar que “Oh, eu vi uma biblioteca que faz manipulação de MIME naqueles projetos”. Se os termos de licença estão certos, você pode economizar bastante tempo, e sua empresa, bastante dinheiro, ao se tornar mais ciente do que já está lá fora para ser usado. Você pode se surpreender ao perceber quanto dinheiro é gasto na indústria de software para reimplementar a roda (*invenção* seria uma palavra muito generosa) mais e mais vezes.

Sir Isaac Newton disse: “Se vi mais longe, foi porque eu subi nos ombros dos gigantes”. Pessoas espertas como Newton sabem que há muito o que aprender daqueles que vieram antes de nós. Seja como Newton.

## Faça algo

- 1) Escolha um projeto e o leia como um livro. Faça anotações. Esboce as coisas boas

e ruins. Escreva uma crítica e a publique. Ache pelo menos um truque ou padrão dele que você possa usar. Encontre pelo menos uma coisa ruim que você observou que acrescentará à sua lista *do que não fazer* quando estiver desenvolvendo software.

- 2) Encontre um grupo de pessoas com a mentalidade parecida com a sua e se reúna com eles uma vez por mês. A cada sessão alguém indica algum trecho de código para ser estudado, de 2 a 200 linhas. Quebre o código e discuta o que está por trás dele. Pense nas decisões que levaram a ele. Pondere sobre o código que *não* está lá.

## CAPÍTULO 18

# Automatize-se em um emprego

Um assunto constante na minha carreira tem sido o conflito entre o desejo das áreas gerenciais de TI em contratar empresas de consultoria com baixo custo (geralmente offshore) para fazer o trabalho e minha forte crença de que o desenvolvedor mais barato geralmente não leva a um custo mais baixo. Eu já tive várias conversas tensas com diretores e vice-presidentes, argumentando de forma apaixonada sobre as vantagens de contratar poucos desenvolvedores que sejam realmente bons em vez de uma legião de pessoas de baixo custo e de habilidades fracas.

Infelizmente, em geral eu era interrompido no meio da minha argumentação. O problema da minha posição não é que eu estava errado (obviamente!). É que não há uma maneira fácil de provar que estava certo. E, a partir de uma perspectiva de custo, a única evidência concreta que temos leva à conclusão de que um baixo custo por hora é, de fato, vantajoso para a empresa.

Imagine um projeto de software hipotético para algum domínio que vier à sua mente. Quantos programadores são necessários para escrever um software como este em três meses? Cinco, você diz? Seis? OK, e quanto ao mesmo projeto em dois

meses? Como você diminui um mês?

O gerenciamento padrão de TI diz que você adiciona programadores para ir mais rápido. É errado, mas é nisso que as pessoas acreditam. E se você consegue fazer um projeto simples ir mais rápido ao adicionar programadores, a conclusão desta regra é a de que mais gente significa maior produtividade. Mas há mais de uma maneira de conseguir isso. Se a meta é aumentar a taxa de desenvolvimento de software, você pode:

- Conseguir pessoas mais rápidas para fazer o trabalho;
- Conseguir **mais** pessoas para fazer o trabalho, ou;
- Automatizar o trabalho.

Uma vez que ainda não se sabe como verdadeiramente medir a produtividade de desenvolvimento de software, é difícil provar que uma pessoa é mais rápida que outra. Então, gerentes de finanças se focam no custo por hora.

Isso leva a uma simples fórmula, que assume um período de tempo fixo:

$$\text{Produtividade} = \frac{\text{Número de projetos}}{\text{Número de programadores} \times \text{Taxa horária}}$$

Em alguns ambientes, é realmente possível calcular a produtividade real de um time de software. Na maior parte, você vai encontrar medidas maleáveis e amorfas como **número de projetos** ou **número de requerimentos**, sem nenhuma possibilidade de repetir a medição dessas unidades.

Logo, a abordagem do **programador mais rápido** é muito difícil de se provar, e nós **não queremos encorajar** a abordagem de se **adicionar mais programadores baratos**. Isso nos leva a uma automação.

Eu me lembro do sensacionalismo em relação à perda de emprego nos Estados Unidos na década de 80. Naquela época, não apenas estávamos culpando outros países, mas também culpávamos as máquinas e, especificamente, computadores. Gigantescos braços robotizados estavam sendo instalados em indústrias. Eles podiam bater o trabalho dos humanos tanto em rendimento quanto em precisão, o que indica

que nem valia a pena compará-los. Todo mundo estava chateado — todo mundo, isto é, com exceção das pessoas que **criaram** os braços robotizados.

Imagine que sua empresa seja do ramo de criação de websites para pequenas empresas. Você basicamente precisa criar o mesmo site repetidas vezes, com contatos, pesquisas, carrinhos de compra, os serviços. Ou você poderia contratar um pequeno número de programadores realmente rápidos para construírem os sites para você, ou contratar vários programadores com custo baixo para fazer a coisa toda manualmente e repetitivamente, ou, em vez, criar um sistema para **gerar** os sites.

Se colocarmos alguns números (inventados) na fórmula do gerente de finanças, temos as equações mostradas na Figura 18.1.

Automação faz parte do DNA de nossa indústria. Ainda assim, por alguma razão, tendemos a não automatizar **nosso** trabalho. Como você poderia **comprovadamente** fazer softwares melhores, mais rápidos e mais baratos do que seu concorrente offshore? Faça os robôs; Automatize-se em um emprego.

#### Programadores rápidos

$$\frac{5}{3 \times \$80} = 0,02$$

#### Programadores baratos

$$\frac{5}{20 \times \$12} = 0,02$$

#### Um programador + um braço robótico

$$\frac{5}{1 \times \$80} = 0,06$$

Figura 18.1: Comparação de produtividade

## De consultor de TI para diretor administrativo

*por Vik Chadha*

Minha jornada de deixar de ser um consultor de TI na GE para ser empresário

na bCatalyst (um aceleradora de negócios com um capital de 5 milhões de dólares) não foi um caminho que eu tinha pensado como o próximo passo de minha carreira.

Então, como foi que eu fiz a transição de trabalhar para uma das 5 principais empresas no ranking da revista Fortune, com dezenas de milhares de funcionários, para trabalhar em uma empresa que orientava e investia em *startups* de tecnologia em estágio inicial? Quando eu olho para trás e tento ligar os pontos, alguns padrões importantes emergem, e eu gostaria de compartilhá-los com você, com a esperança de que você possa adaptá-los para o seu contexto.

Logo depois de terminar meu mestrado em engenharia da computação e elétrica na Virginia Tech, eu entrei na GE como um consultor. O uso comercial da internet estava começando a avançar em passos largos, e eu trabalhei em vários projetos para fazer a plataforma mais incrível e poderosa, junto de suas tecnologias internas, progredindo rapidamente começando pela equipe de finanças de TI, passando para o grupo de serviços e tecnologia, para a automação de vendas, e finalmente, para os dados de venda do grupo de depósito, trabalhando em cada equipe para desenvolver novas iniciativas. Eu adorava pesquisar, e com isso, pegar as mais novas tecnologias e aplicá-las para resolver problemas complicados.

Entretanto, viver nessa área de novidades da tecnologia nem sempre era divertido. Nós invariavelmente tínhamos problemas com tecnologias que ainda não estavam prontas para serem usadas, e gastávamos muito tempo e energia ajudando os seus donos a debugarem seus produtos. Do ponto de vista do cliente, eu aprendi que não importa quão legal a tecnologia seja, ela só terá valor se resolver um problema real. Ao longo do tempo, isso me ajudou a mudar a maneira de pensar, de ser centrado em tecnologia para ser centrado em solução. Perceber essa nova forma de pensar foi muito valioso para avaliar as startups de tecnologia em estágio inicial na bCatalyst, alguns anos depois.

Contudo, por mais que eu gostasse de trabalhar na GE, um aspecto importante estava faltando. Eu senti que, na minha função como um profissional de TI, eu estava basicamente desenvolvendo todas minhas habilidades em uma única dimensão, sem ter a oportunidade de realmente entender como as empresas funcionam, como elas fazem dinheiro, o que as tornam sustentáveis, e como elas inovam. Em vez de ficar frustrado, eu decidi tomar a iniciativa e fazer algo a respeito disso, aprendendo mais sobre negócios e empreendedorismo. Eu nunca fiz nenhum curso de negócios e sabia que o único jeito para eu aprender os detalhes do que é preciso para começar uma empresa era colocando a mão na massa (isto é, aprender por meio de tentativas e erros).

Um ex-colega de quarto e empreendedor, que também era um amigo próximo (Raj Hajela), minha esposa (Vidya), e eu elaborávamos ideias tentando descobrir onde havia necessidades insatisfeitas no mercado. Nós queríamos explorar as oportunidades de *e-commerce*, mas não queríamos vender nada que fosse uma mercadoria. Nós tínhamos um interesse real, e um conhecimento prévio em artes e gostávamos do fato de que cada obra de arte era única em sua natureza. Meu tio durante a vida toda foi um artista que lutou para se sustentar com isso. Fizemos algumas pesquisas e concluímos que este era o caso da maioria dos artistas. Então decidimos resolver esse problema criando uma plataforma que ajudasse os artistas para que pudessem publicar e promover seus trabalhos, além de manter contato com seus consumidores. Com essa missão em mente, lançamos o [Passion4Art.com](http://Passion4Art.com) e começamos o trabalho pesado de conseguir artistas para se juntarem ao nosso site e colocarem as imagens digitais de suas pinturas online. Depois que inscrevemos os primeiros 1000 artistas, e eles conseguiram montar seus próprios websites, acreditamos que estávamos dando alguma coisa de valor, e passamos a procurar por investimento externo.

Naquela época (cerca de 1999), uma empresa chamada [eMazing.com](http://eMazing.com) fornecia dicas diárias sobre uma variedade de assuntos, e nós achamos que podíamos fazer parceria com eles (trabalhando com nossos artistas e o canal de distribuição deles), para fornecer um *Art Tip of the Day* (dica de arte do dia). Um dos chefes executivos se reuniu conosco, aprovou o que tínhamos a oferecer e concordou em fazer uma tentativa.

Falamos pra ele que estávamos procurando investimento para construir nossa infraestrutura, e ele gentilmente se encarregou de enviar nosso plano de negócios para um novo acelerador de negócios na cidade, chamado [bCatalyst](http://bCatalyst).

Poucos dias depois, recebemos uma ligação de Keith Williams, o CEO da [bCatalyst](http://bCatalyst), informando-nos que eles tinham interesse em nos encontrar pessoalmente e saber mais sobre nossas intenções. Estávamos obviamente empolgados com essa reunião. Eu não percebi até muito tempo depois o quão importante era o fato de eles terem ouvido sobre nós de uma fonte confiável. A lição aqui é que se você precisar entrar em contato com um investidor de risco, trabalhe duro para conseguir uma boa indicação, já que esta é a melhor maneira de abrir portas.

Ao longo das várias reuniões que tivemos com Keith, percebemos que havia uma boa química entre nossos times, mas a bolha da internet havia recentemente estourado, e o momento não era bom para eles fazerem um investimento nessa área. Contudo, eles nos disseram que se trouxéssemos outra ideia da qual eles gostassem, não hesitariam em nos apoiar. Eu perguntei-lhes se essa era uma maneira educada de

se dizer “não” ou se eles estavam falando sério sobre trabalhar conosco. Eles nos garantiram que realmente estavam interessados.

Então eu pedi outra reunião com Keith e disse que eu estava disposto a sair da GE para trabalhar com eles em período integral nos próximos meses e explorar outras oportunidades de startup. Essa oportunidade se materializou porque eu consegui convencê-los de que eu estava disposto a colocar minha própria pele na jogada, uma vez que eu havia deixado a GE sem um caminho claro e seguro diante de mim.

Durante os próximos 12 meses, todo dia nós nos encontrávamos com equipes diferentes, selecionando suas ideias, e eu reparei em um novo padrão no conjunto de questões que perguntávamos a cada empresa.

Eu compilei essa lista e estou compartilhando as questões com você, no caso de você precisar arrecadar dinheiro de investidores de risco no futuro; veja <http://www.greaterlouisville.com/EnterpriseCORP/Forms/BusinessAssessment/>.

As habilidades que eu adquiri durante aquele ano na bCatalyst me levaram ao meu emprego atual, como diretor administrativo da Enterprise Corp. Nos últimos sete anos, eu trabalhei com mais de 100 empresas e ajudei a arrecadar um fundo de mais de 75 milhões de dólares. Essa tem sido uma experiência extremamente satisfatória, que não teria sido possível se eu não tivesse tomado a iniciativa e sido aventureiro em tentar coisas novas. Os vários *zig-zags* ao longo do caminho foram parte integral do processo. Minha esperança é de que você, leitor, use minha história para se inspirar a encontrar seu próprio caminho, aquele que vai usar suas habilidades ao máximo.

*Vik Chadha é o diretor administrativo da Enterprise Corp.*

## Faça algo

- 1) Escolha uma tarefa que você normalmente faz de forma repetitiva e escreva um gerador de código para ela. Não se preocupe com reusabilidade. Apenas garanta que seu gerador economize tempo.

Pense em uma maneira de aumentar o nível de abstração do que você está gerando.

- 2) Pesquise por *model-driven architecture* (MDA — Arquitetura orientada a modelo). Experimente algumas das ferramentas disponíveis. Procure algum lugar do seu trabalho no qual aplicar alguns **conceitos** de MDA, ou talvez o grupo completo de ferramentas. Pense em aplicar os conceitos de MDA apenas com as ferramentas que você utiliza todo dia.



## CAPÍTULO 19

# Agora mesmo

Imagine que você está em uma corrida com um prêmio de \$100000. Vence a primeira equipe que criar um software para implementar uma aplicação de contas a receber. Você e sua equipe se inscreveram para competir. O concurso vai acontecer em uma semana. Para ganhar, seu código deve ser completamente testado e implementar um conjunto mínimo de funcionalidades. Você começa sábado de manhã, e tem até segunda-feira de manhã para completar sua aplicação. A primeira equipe que concluir antes de segunda de manhã ganha a corrida. Se nenhum time terminar antes disso, o que tiver mais funcionalidades implementadas vence.

Você folheia os requisitos das funções da aplicação. Olhando para a lista de funcionalidades, você percebe que o sistema é semelhante em tamanho e escopo a vários sistemas com que você trabalhou no passado. Enquanto o objetivo acordado em sua equipe é terminar no meio do dia de domingo, por um momento você começa a se questionar. **Como é que uma aplicação de escopo similar àquelas nas quais passamos semanas trabalhando no escritório vai ser concluída em um único final de semana?**

Mas com o começo da competição você começa a codificar e percebe que sua equipe **vai** ser capaz de alcançar o objetivo. A time está em um movimento coletivo, produzindo recurso depois de recurso, consertando erros, tomando decisões em frações de segundo, e concluindo as funcionalidades. Você se sente bem. Em revisões de projeto e reuniões de status lá na sua empresa, você sonhava pegar uma pequena equipe, tirar daquele ambiente burocrático para se dedicar à criação de uma nova aplicação em tempo recorde.

Muitos de nós têm esse sonho. **Sabemos** que nossos processos nos atrasam. Não apenas isso, mas sabemos que a velocidade de nossos ambientes **nos** faz ir mais devagar.

*O que podemos fazer? Agora mesmo?*

—

A lei de Parkinson afirma que “O trabalho se expande até preencher o tempo disponível em sua completude”. O lado triste é que, mesmo quando você não quer que seja desse jeito, você pode cair na armadilha, especialmente quando existem tarefas que você não quer fazer.

No caso de uma corrida para implementar um sistema em um final de semana, você não tem tempo de deixar as tarefas de lado, então você não o faz. Você não pode atrasar uma decisão, então você não o faz. Você não pode evitar trabalho chato, e você sabe que deve fazê-lo tão rápido que nada pode ser **tão** chato.

A lei de Parkinson é uma observação empírica — e não um mandato humano inescapável. Um senso de urgência, mesmo manufaturado, é suficiente para facilmente dobrar ou triplicar sua produtividade. Tente isso, e você verá. Você pode fazer mais rápido. Você pode fazer agora. Você pode torná-lo pronto em vez de conversar sobre torná-lo pronto.

Se você tratar seus projetos como uma corrida, você chegará ao final muito mais rápido do que se você tratá-los como uma cela de prisão. Crie movimento. Seja aquele que empurra. Não fique confortável demais.

Sempre seja aquele que pergunta “Mas o que pode ser feito **agora mesmo?**”

## Faça algo

- 1) Olhe para sua escrivaninha. Examine as tarefas que estão lá por um longo tempo, os projetos que estão começando a tomar forma, ou aqueles em que você está um

pouco **travado** — talvez por questões burocráticas, talvez travados por conta de algum análise.

Encontre um que você poderia **fazer** nas horas livres do seu trabalho normal, quando você estaria provavelmente navegando pela internet, checando e-mails ou tendo um almoço por mais de uma hora. Transforme um projeto de meses em uma tarefa de menos de uma semana.



## CAPÍTULO 20

# Leitor de mentes

Eu trabalhava com um rapaz chamado Rao. Ele vinha de um estado do sul da Índia chamado Andhra Pradesh, mas estava alocado nos Estados Unidos e trabalhava com a gente. Ele era o tipo de pessoa que podia codificar qualquer coisa que você pedisse. Se você precisasse de programação de sistema de baixo nível, ele era a pessoa a quem pedir. Se você precisasse de programação de aplicações de alto nível, ele poderia fazer praticamente tudo que você pedisse.

Entretanto, o que tornou Rao verdadeiramente diferenciado foi o que ele fez **antes** de você pedir. Ele tinha essa misteriosa habilidade de prever o que você estava prestes a pedir, e o fazia mesmo antes de você pensar nisso. Era como mágica. Acredito que eu até o acusei de aplicar truques em mim, em certo ponto, mas não há como ter sido truque. Eu diria “Rao, estive pensando sobre mudar a forma como estamos encapsulando o controlador no framework da nossa aplicação. Se nós mudássemos apenas um pouco, ele poderia ser usada além da web. O que você acha?”

“Eu fiz isso no começo dessa semana”, ele diria. “Dá uma olhada lá no controle de versão.” Isso **constantemente** acontecia com Rao. Era tão frequente que a única

forma de isso ter sido uma coincidência era se Rao estivesse literalmente fazendo **qualquer coisa imaginável** com o software que a equipe mantinha.

Naquela época, eu liderava a equipe de arquitetura de aplicações da minha empresa. Entre outras coisas, nós construíamos e mantínhamos frameworks para uso nas aplicações da empresa. Meus colegas de equipe passavam muito tempo conversando sobre como queríamos ver a área de desenvolvimento de software na empresa melhorar. Também conversávamos bastante sobre o papel dos componentes de nossa infraestrutura básica nessas melhorias.

É aí que os truques de magia de Rao entravam. Ele não falava muito nessas conversas, mas ele não era nem um pouco desengajado. Ele ouvia com atenção. E, entregando seu segredo como nenhum mágico faria, ele depois me contou que o truque era que ele estava apenas fazendo coisas que eu já havia dito que queria. Eu já o havia dito de maneira tão sutil que **nem eu** percebia o que dissera.

A gente podia estar esperando o café ficar pronto e eu dizia sobre como seria ótimo ter alguma nova flexibilidade no nosso código. Se eu o dissesse com frequência ou convicção suficientes, mesmo sem colocá-lo na lista de afazeres da equipe, Rao iria preencher as lacunas entre “trabalho real” analisando a viabilidade de implementar um daquelas coisas. Se fosse fácil (e barato) de implementar, ele iria eliminar a tarefa colocando-a em funcionamento.

*O truque da leitura de mente, se feito certo, faz as pessoas dependerem de você.*

—

Leitura de mente não apenas se aplica aos seus chefes, mas também aos seus clientes. Se eles estão dando as dicas certas, você pode ser capaz de acrescentar funcionalidades que eles ou **vão** pedir ou **teriam** pedido se tivessem percebido que elas eram possíveis. Se você sempre faz o que seus clientes pedem quando eles pedem, você os satisfará. Todavia, se você fizer **mais** do que eles pedem, ou se você já tiver feito antes de eles pedirem, isso irá encantá-los, isto é, ao menos que sua habilidade de ler mentes seja defeituosa.

Esse truque de leitura de mente não é inteiramente seguro. É uma corda bamba sobre a qual você quer evitar andar, ao menos que haja uma rede segura embaixo. Os maiores riscos (alguns atenuados) são os seguintes:

- Você gasta o dinheiro da empresa fazendo trabalho que não foi pedido. E se você estiver errado? Comece pequeno. Só faça suposições do que pode ser encaixado nas frestas do seu trabalho normal, de modo que o impacto seja

pequeno ou nulo. Se você está tão disposto, dedique-se a esse trabalho extra no seu tempo livre.

- Toda vez que você adiciona código em um sistema, você corre o grande risco de torná-lo menos resistente a mudanças. Evite trabalho de leitura de mente que pode forçar o sistema a um determinado caminho arquitetônico ou limitar de alguma forma o que ele pode fazer. Quando o impacto da mudança é grande o suficiente, é necessária uma decisão de negócio. Raramente são apenas os desenvolvedores que precisam opinar em tal decisão.
- Você pode decidir mudar uma funcionalidade que seus clientes **pediram**, de uma maneira que, inesperadamente, a torne menos funcional ou desejável para o cliente. Esteja atento ao adivinhar quando se trata especificamente de interface de usuário.

Gerenciar pessoas e projetos é um trabalho desafiador. Pessoas que podem manter um projeto andando na direção certa, sem que seja dada muita orientação, possuem alto valor e são reconhecidas por seus sobrecarregados gerentes e clientes. O truque de leitura de mente, se feito certo, faz as pessoas dependerem de você – uma excelente receita para uma carreira cuja direção você pode controlar. É uma habilidade que vale ser explorada e desenvolvida.

## Faça algo

- 1) Um revisor do início deste livro, Karl Brophey, sugere que, em seu novo projeto ou para um sistema que você mantém, você comece a tomar notas sobre o que você **acha** que os usuários e chefes vão pedir. Seja criativo. Tente ver o sistema a partir do ponto de vista deles. Depois de ter uma lista das funcionalidades não tão óbvias que podem aparecer, pense sobre como você poderia implementá-las de forma mais eficiente. Pense sobre casos limites que seus clientes podem não ter em mente imediatamente.

Conforme você explora as solicitações de melhoria, acompanhe sua taxa de acerto. Quantas de suas suposições tornaram-se funcionalidades que realmente foram requisitadas? Quando suas suposições vieram à tona, você foi capaz de usar as ideias que vieram em sua sessão de *brainstorm*?





## CAPÍTULO 21

# Êxito diário

Todos nós gostamos de pensar, pela virtude de nosso conhecimento e por sermos bons em software, que naturalmente vamos dominar um assunto e virar referência. Para a minoria sortuda (e eu **realmente** pretendo usar a palavra **sorte** aqui), uma estratégia como essa vai funcionar.

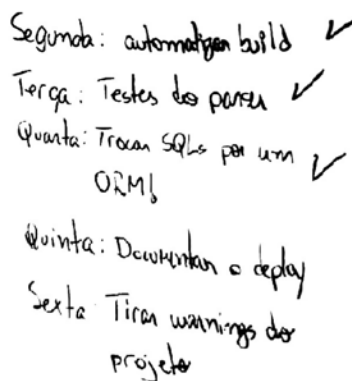
Porém, todos nós podemos tirar benefícios a partir do agendamento e do acompanhamento do nosso desempenho. O senso comum afirma que se excedermos as expectativas dos nossos chefes estaremos na lista A. Dado que exceder as expectativas é uma meta digna, surpreendentemente poucos de nós têm mecanismos para acompanhar como e quando excederam as expectativas de seus patrões.

Assim como a maioria das tarefas dignas de serem feitas, tornar-se alguém destacado é mais provável com algum trabalho específico e intencional. Quando foi a última vez que você foi além do chamado de dever? Seu chefe soube disso? Como você pode melhorar a **visibilidade** desse comportamento?

*Tenha uma realização para relatar todo dia.*

James McMurry, um colega de trabalho que também é um grande amigo (<http://semanticnoise.com>), contou-me no início de nossas carreiras sobre um sistema que ele inventou para se certificar de que estava fazendo um bom trabalho. Impressionou-me sendo extremamente perspicaz, dada sua experiência (talvez seja uma dica dada por seus pais), e eu o uso até hoje. Sem avisar seu chefe, ele começou a acompanhar seus **êxitos diários**. Seu objetivo era, cada dia, ter algum tipo de conquista marcante para relatar a seu chefe – alguma ideia em que ele pensou ou implementou, que tornaria seu departamento melhor.

Simplesmente estabelecendo um objetivo (diário, semanal, ou quando você for capaz) e acompanhando seus êxitos é possível mudar seu comportamento radicalmente. Quando você começa a procurar por conquistas marcantes, você naturalmente entra no processo de avaliar e priorizar suas próprias atividades, baseado no valor de negócio daquilo em que você pretende trabalhar.



Segunda: automatizar build ✓  
Terça: Testes dos parâmetros ✓  
Quarta: Trocar SQLs por um ORM ✓  
Quinta: Documentar o deploy  
Sexta: Tirar warnings do projeto

Figura 21.1: Uma semana de êxitos

Acompanhar seus êxitos com uma frequência razoavelmente alta vai assegurar que você não fique travado: se você deve alcançar um êxito por dia, não pode passar duas semanas elaborando a tarefa **perfeita**. O tipo de pensamento e trabalho torna-se um hábito, em vez de uma produção maior. E, assim como um desenvolvedor viciado na barra verde de uma **suíte de teste de unidade**, você começa a ficar irritado se não alcançou o êxito do dia. Não é preciso se preocupar muito em acompanhar seu progresso, porque agir nesse nível parece mais com um tique nervoso do que um conjunto de tarefas que precisam ser planejadas em um Microsoft Project.

## Faça algo

- 1) Separe meia hora de sua agenda, e sente-se com um papel e lápis em um lugar tranquilo onde você não será interrompido. Pense sobre as pequenas picuinhas dos problemas diários de sua equipe. Escreva-os. Quais são as tarefas irritantes que gastam alguns minutos do tempo da equipe todo dia, mas que ninguém tem tempo ou energia de resolver?

Em que lugar do seu atual projeto você está fazendo algo manualmente que podia ser automatizado? Escreva-o. E quanto a seu processo de *build* ou *deploy*? Há alguma coisa que você poderia limpar? Como você poderia reduzir falhas em seu build? Escreva todas essas ideias.

Dê a si mesmo sólidos vinte minutos para isso. Escreva todas suas ideias – boas ou ruins. Não se permita terminar antes dos vinte minutos. Depois de ter feito uma lista, em um novo papel, escreva seus cinco itens favoritos (mais irritantes). Na semana seguinte, na segunda-feira, pegue o primeiro item da lista, e faça algo a respeito dele. Na terça-feira, pegue o segundo item. Na quarta-feira, o terceiro, e assim por diante.



## CAPÍTULO 22

# Lembre-se de para quem você trabalha

É muito fácil dizer “Certifique-se de que seus objetivos e seu trabalho estejam alinhados com os objetivos de sua empresa”. É muito fácil dizer; mas é muito difícil fazer, especialmente quando você é um programador enterrado sob tantas camadas organizacionais. No início da minha carreira, eu trabalhava para uma grande empresa de entrega de pacotes, em uma equipe de arquitetura de desenvolvimento de software apoiando os sistemas definidos pela empresa. Ela estava tão entrelaçada com hierarquia, que eu nunca vi nada no meu trabalho diário que chegasse perto de entrega de pacotes. Lembro-me de minha equipe participando das reuniões trimestrais, sentindo-se completamente deslocados e alienados. “Qual é a conquista que estamos comemorando? O que toda essa métrica significa?”

De fato, nesse ponto de minha carreira, eu estava mais interessado em construir sistemas elegantes e hackear softwares open sources, do que mergulhar nas profundezas de um negócio de entrega de pacotes. (OK, eu admito – eu **ainda** sou mais

interessado naquelas coisas). Mas se eu realmente quisesse alinhar meus objetivos com os da empresa, eu não tenho certeza se eu saberia por onde começar.

Então está certo dizer que precisamos alinhar nosso trabalho com as metas da empresa – para tentarmos assegurar que estamos impactando o ponto de partida de tudo aquilo. Contudo, que a verdade seja dita, muitos de nós simplesmente não enxerga como fazer isso do lugar onde estão. Não podemos ver a floresta por causa das árvores.

Talvez isso não seja nossa culpa. Talvez estejamos cobrando muito de nós mesmos. Talvez a ideia de tentar impactar diretamente a empresa seja equivalente a tentar ferver o oceano. Portanto, precisamos ter uma visão mais compartimentada, dividindo o negócio em poças fervíveis.

A poça mais óbvia com a qual começar é sua própria equipe. É provavelmente pequena e suficientemente focada, de modo que você pode conceitualmente se envolver em torno dela. Você muito provavelmente entende os problemas que sua equipe enfrenta. Você sabe no que ela está focada em melhorar, seja na produtividade, no rendimento, redução de erro, ou qualquer outra coisa. Se você não tem certeza, há um lugar óbvio para se descobrir: seu chefe.

Em última análise, em um ambiente bem estruturado, os objetivos de seu chefe são os mesmos de sua equipe. Resolva o problema de seu chefe e terá resolvido um problema para a equipe. Adicionalmente, se seu chefe possui a mesma abordagem que você, os problemas que você está resolvendo para ele, na verdade, são os problemas do chefe dele. E assim por diante, até que isso chega no nível mais alto da empresa – o CEO, os acionistas, ou até mesmo os consumidores.

Fazendo sua pequena parte, você estará contribuindo para a realização das metas de sua empresa. Isso pode dar-lhe um senso de propósito. Isso dá significado ao seu trabalho.

Alguns podem resistir a essa estratégia. “Eu não vou fazer esse serviço para ele”, ou, “Ela vai levar crédito pelo meu trabalho!”

Bom, sim. Mais ou menos. É assim que funciona. O papel de um bom chefe não é, como Lister e DeMarco dizem em *Peopleware* [2], “ser um jogador reserva do time”, sabendo fazer todo o trabalho da equipe. O papel de um bom chefe é colocar prioridades para a equipe, certificar-se de que ela tem o que precisa para realizar o trabalho, e fazer o que for preciso para manter a equipe motivada e produtiva, por fim, fazer o que deve ser feito. Um trabalho bem feito pela equipe é um trabalho bem feito pelo chefe.

*O sucesso de seu chefe é o seu sucesso.*

—

Se a função do chefe é saber e estabelecer prioridades, mas não pessoalmente **realizar** o trabalho, logo, sua função é fazer todo o trabalho. Você não está fazendo o serviço de seu gerente. Você está fazendo seu serviço.

Se você realmente está preocupado com quem recebe os créditos, lembre-se que é seu **chefe** que detém as chaves de sua carreira (em sua atual empresa, pelo menos). Na maior parte das empresas, é o chefe direto quem influencia a avaliação das performances, ações salariais, bônus e promoções. Portanto, o crédito que você busca fica depositado com seu gerente.

Lembre-se de para quem você trabalha. Você não vai somente se alinhar com as necessidades do negócio, mas alinhar o negócio com **suas** necessidades. Se você vai **dominar** a execução de sua função, isso vai assegurá-lo de executar as coisas certas.

## Faça algo

- 1) Agende uma reunião com seu chefe. Esse compromisso é para você entender as metas de seu chefe para a equipe para o próximo mês, trimestre, e ano. Pergunte como você pode fazer a diferença. Após a reunião, examine como seu trabalho diário se alinha aos objetivos de sua equipe. Faça com que eles sejam um filtro de tudo o que você faz. Priorize seu trabalho de acordo com esses objetivos.





## CAPÍTULO 23

# Esteja onde você está

Como chefe, posso dizer que a coisa mais frustrante é lidar com um funcionário que está sempre almejando o próximo degrau da escada. Você conhece o tipo: você não consegue sentar com ele no almoço sem que ele traga o assunto de quem recebeu qual promoção. Ele sempre tem algum tipo de fofoca do escritório para espalhar, e ele parece se apegar à política corporativa como se fosse o enredo de uma novela na qual ele seja viciado. Ele reclama sobre a incompetência da gerência, e com muito desgosto completa suas tarefas, sabendo muito bem que ele podia exercer a função de chefe melhor do que eles. Eles só são incompetentes para entender seu potencial.

Ele pensa que muitas tarefas estão abaixo dele. Ele as evita quando possível e as faz a contragosto (e lentamente), quando não. Ele escolhe trabalhos que acha, mesmo subconscientemente, que estão de acordo com seu nível e que podem levá-lo à próxima promoção.

*Seja ambicioso, mas não demonstre.*

—

A coisa triste sobre esse cara é que, pelo fato de ele estar vivendo em seu próximo emprego, ele geralmente está fazendo um trabalho medíocre na sua função atual. É como aparar a grama para mim. Eu detesto aparar a grama. Isso me dá coceiras e me faz suar. Pior de tudo, isso faz com que eu não faça algo que eu **preferia** estar fazendo. Você pode contratar pessoas para apararem sua grama. Eu fui uma dessas pessoas. Isso foi há muito tempo, e agora eu progredi. Então, quando eu **preciso** aparar a grama, o que eu faço? Eu me apresso. Faço um serviço desleixado. Passo o tempo todo pensando em como acabar logo com isso para que eu possa me dedicar às coisas que eu preferia fazer. Resumindo, eu faço um péssimo serviço ao aparar a grama.

Ainda bem que, no meu exemplo de aparar a grama, ninguém está me observando e me avaliando (apesar de que minha esposa ficou tão irritada que eu não sou mais o responsável pela grama na nossa casa). O problema é meu se a grama não está ótima quando concluí o trabalho. Ninguém está me segurando para ser “apenas um cortador de grama” por causa da minha performance no jardim. No caso de um emprego de TI, o mesmo comportamento pode provocar uma catástrofe em uma carreira. Voltando ao nosso amigo dos parágrafos anteriores, como você acha que a administração o verá? Eles verão que estavam errados ao negligenciar seu talento brilhante e decidirão promovê-lo? Eles darão grandes aumentos para fazê-lo feliz?

Claro que não. Ele é um funcionário ridículo com uma atitude ruim. **E daí** se ele tem alto potencial! Por enquanto, ele não o está mostrando. A empresa não faz dinheiro por causa de potencial. Acionistas não mantêm seus investimentos se o potencial não é efetivado. Além disso, sua atitude faz seus chefes quererem parar de investir **nele**.

Então, esse é o ponto de vista de um gerente. Agora, claro, eu não estou completamente isento de culpa aqui. Eu fui essa pessoa por algum tempo. Também não é muito bom desse lado da rua. Você passa todo seu tempo desejando algo. Desejo é o oposto de satisfação. Você acorda de manhã e tem que ir para “aquela droga de emprego” onde ninguém entende seu potencial. Com ressentimento, você labuta em seu serviço, repassando estratégias para subir de cargo. Você fantasia sobre o que **você** faria na última situação em que seu chefe errou as coisas – como você lidaria com isso diferentemente. Você adia viver enquanto você está trabalhando até que você possa fazer **do seu jeito**, na posição que você merece.

Aqui está um segredo: essa sensação nunca vai passar. Se e quando você finalmente conseguir a grande promoção com a qual você tem sonhado, você rapidamente vai se cansar e perceber que não é **este** trabalho que você queria – é o **pró-**

**ximo.** O ciclo começa de novo. Eu ainda não alcancei o topo, mas eu tenho a forte intuição de que se houvesse tal posição, e eu a alcançasse, eu olharia para cima e perceberia que estava perseguindo um fantasma. Que frustrante desperdício de vida profissional.

Mas nós não deveríamos ter ambições? Haveria uma Microsoft ou uma General Electric se grandes empreendedores não tivessem ambicionado e tido metas?

Claro que deveríamos. Eu não estou defendendo uma visão apática. É bom ter objetivos, e é bom querer ter sucesso. Mas pense no cara negativo e ressentido que eu descrevi no começo desse capítulo. Você acha que ele vai ser a pessoa que tem sucesso? Parece retrógrado, mas focar-se no presente vai levá-lo mais longe em direção aos seus objetivos, do que manter sua mente focada no objetivo em si.

Você vai descobrir que isso é muito pragmático. Focar no presente vai permitir-lhe aproveitar as pequenas vitórias do dia a dia de trabalho: a sensação de trabalho bem feito, a sensação de ser considerado um expert em um problema crítico de negócio, a sensação de ser um membro integral de uma equipe bem sucedida.

Isso é o que você vai perder se estiver sempre com a cabeça nas nuvens. Você estará sempre esperando o **grandioso**, enquanto ignora as pequenas coisas que acontecem todo dia, que fazem valer a pena aparecer em seu serviço.

Não apenas **você** vai se sentir bem, mas aqueles ao seu redor também vão. Seus colegas de trabalho, chefes e clientes vão senti-lo. Isso vai transparecer no seu trabalho. Por mais contraintuitivo que pareça, deixar de lado seu desejo de ter sucesso resultará em uma aprimorada habilidade de ser bem sucedido.

Você é próximo aos seus clientes. Você é próximo aos líderes com poder de decisão que vão modelar sua carreira a curto prazo e, possivelmente, a longo prazo. Desenvolvedores na Índia ou nas Filipinas não possuem essa vantagem, mas **você** tem. Portanto, **esteja** onde você **está**.

## Faça algo

- 1) Coloque as metas de sua carreira de lado por uma semana. Escreva seus objetivos para seu emprego **atual**. Em vez de pensar sobre aonde você quer ir em seguida, pense sobre o que você quer ter alcançado quando você terminar o trabalho no qual está agora. O que você pode ter **produzido** nesse emprego que terá sido ótimo? Crie um plano que seja ambos estratégico e tático. Passe a semana implementando essas táticas em apoio às metas a longo prazo de “terminar” esse trabalho.

Durante almoços e intervalos com seus colegas, foque a conversa nesses objetivos. Quanto tempo vai passar até que você conquiste tudo o que sente que precisa em sua função atual? Como você saberá que terá concluído? Planeje a próxima semana e repita.

## CAPÍTULO 24

# Quão bom eu posso fazer um trabalho hoje?

É recompensador realizar um bom trabalho e ser reconhecido. Embora a maioria de nós saiba disso intuitivamente, somos extremamente seletivos sobre onde e quando realmente vamos além do nosso trabalho para nos sobressairmos. Nós podemos nos dedicar cegamente ao design do Novo Grande Projeto do departamento de marketing, ou nós rapidamente mergulhamos em um problema para salvar o dia em alguma grande catástrofe, porque nossos cérebros estão programados para entender tais momentos como oportunidades de exibir nosso notório material. Podemos até mesmo fazer o trabalho no meio da noite, com um nível de foco de detalhe que normalmente nos levaria às lágrimas. Uma situação terrível pode frequentemente fazer surgir o melhor em nós.

Eu já deixei essa sensação inebriante de exaltação manter-me acordado e efetivamente trabalhando em algumas das mais cansativas falhas de sistema e prazos não cumpridos. Por que é que, quando não há grandes pressões, nós geralmente não so-

mos capazes desse tipo de comportamento de altruísmo e de frenesi ultraprodutivo? Quão bem nós atuaríamos se pudéssemos tratar as tarefas mais desinteressantes e chatas com o mesmo desejo fervoroso de fazê-las direito?

*Quão mais divertido você pode tornar seu trabalho?*

—

A última questão fica melhor se a reescrevermos. Quão mais **divertido** seria seu trabalho se você pudesse tratar as tarefas mais chatas com o mesmo desejo fervoroso de fazê-las direito? Quando temos mais diversão, fazemos um melhor trabalho. Então, quando não temos interesse em uma tarefa, ficamos entediados e, como resultado, nosso trabalho também sofre.

Como transformar um trabalho entediante em um mais divertido? A resposta a essa pergunta pode ser mais aparente se você invertê-la. Por que o trabalho entediante é entediante? Por que ele **ainda** não é divertido? Qual a diferença entre trabalho que você aprecia e trabalho que você abomina?

Para muitos de nós, técnicos, o trabalho entediante é assim por dois motivos primários. O trabalho que amamos nos permite flexionar nossos músculos criativos. Desenvolvimento de software é um ato criativo, e muitos de nós somos atraídos para isso por essa razão. O trabalho que não amamos é raramente o que consideramos de natureza criativa. Pense sobre isso por um momento. Pense sobre o que está em sua lista de tarefas para a próxima semana. As tarefas que você adoraria pular provavelmente não dão muito espaço para a imaginação. Elas são apenas tarefas a fazer, que você gostaria de poder delegar a outra pessoa.

A segunda razão pela qual tarefas chatas são chatas, e relacionada à primeira, é que as tarefas chatas não são desafiadoras. Nós amamos mergulhar em um problema difícil e resolver coisas que outros não conseguiram. É o mesmo sentimento que motiva membros de nossa espécie a recreativamente arriscar suas vidas escalando montanhas e saltando de bungee jump. Amamos fazer coisas para provar que somos capazes. Tarefas chatas normalmente não estimulam o cérebro. Fazê-las é tão desafiador quanto colocar o lixo para fora de casa.

Então como é possível ainda usarmos nossa criatividade e desafiar a nós mesmos enquanto tendemos às sobras mundanas do nosso dia de trabalho (que provavelmente tomam mais de 80% do tempo da maioria de nós)?

E se você tentasse fazer as coisas chatas **perfeitamente**? Digamos que, por exemplo, você odeia teste de unidade. Você ama programar, mas fica entediado em ter de escrever código de teste automatizado. E se você se esforçar para fazer seus testes

perfeitos? Como isso mudaria seu comportamento? O que **perfeito** significa em relação a teste de unidade? Provavelmente terá algo a ver com cobertura de teste. Isso significaria que você testou 100% da funcionalidade do seu código. Testes de unidade perfeitos também são limpos e de fácil manutenção, e não dependem de muitos fatores externos que podem ser difíceis de replicar em outro computador. Eles deviam ser executáveis diretamente depois de um *checkout* no controle de versão. E, é claro, todos os testes deviam passar.

Isso está começando a parecer difícil; uma cobertura de teste de 100% é quase impossível. E o negócio de dissociação de testes de modo que eles possam rodar sem dependências externas apresenta vários desafios. Aliás, você provavelmente terá que modificar seu código para tornar isso possível. Mas, se você pudesse fazê-lo, os testes seriam incríveis.

Eu não sei quanto a você, mas isso me parece até divertido. De fato, você pode aplicar o mesmo pensamento para a maioria das tarefas que cruzam seu caminho. Tente isso amanhã.

Olhe para seu trabalho e pergunte a si mesmo: “Quão bom eu posso fazer isso hoje?”. Você descobrirá que você gostará mais de seu emprego e ele gostará de você.

## Faça algo

- 1) **Torne isso visível** — Transforme aquelas tarefas chatas em uma competição com seus colegas de trabalho. Veja quem pode fazê-las melhor. Não gosta de escrever testes de unidade? Imprima o número de *asserts* dos seus testes todo dia, e o pendure em sua parede. Mantenha uma tabela de pontos para a equipe inteira. Compita pelo direito de se gabar (ou mesmo por prêmios). Ao final, faça com que o vencedor ganhe um prêmio, por exemplo, ter os testes feitos por outra pessoa da equipe por uma semana.





## CAPÍTULO 25

# Quanto você vale?

Você já parou para analisar exatamente quanto você custa para a empresa na qual trabalha? Quero dizer, você sabe o seu salário. Esta parte é fácil. Mas e quanto a benefícios, despesas de gerenciamento, e todas as outras coisas que não necessariamente aparecem em seu holerite?

É fácil simplesmente **querer mais**. Infelizmente, isso parece ser uma tendência básica humana, de fato. Você recebe um aumento e isso faz-lhe sentir bem por um tempo, mas a partir de então você começa a pensar no próximo. “Se eu pudesse fazer apenas 10% mais, eu poderia comprar aquele novo...”. Todos nós fizemos isso. Em algum momento, o número verdadeiro se torna abstrato. Não se trata mais de 10% a mais por mês. Trata-se de fazer qualquer que seja o número atual subir mais. Se não recebemos um aumento de salário satisfatório em um ano, tornamo-nos insatisfeitos com nosso trabalho e nossa empresa. “Por que eles não gostam de mim?”

Quanto você realmente custa? Como eu já mencionei, é obviamente mais do que seu salário básico. Por uma questão de discussão, vamos fazer a estimativa com duas vezes seu salário. Então, se você ganha 5 mil por mês, a empresa realmente gasta em

torno de 10 mil mantendo você empregado.

Isso foi fácil. Agora a parte difícil: quanto valor você produziu no ano passado? Qual foi seu impacto **positivo** no resultado final da empresa? Já sabemos que você custa à empresa (em nosso cenário imaginário) aproximadamente 10 mil por mês. O que você deu em troca? Quanto dinheiro você fez a empresa economizar? Quanto mais contribuiu na receita dela?

Esse número é maior do que o dobro do seu salário?

É um exercício complicado de fazer, porque geralmente é difícil relatar cada aspecto de seu trabalho no resultado final da empresa. Pode até parecer uma pergunta sem sentido para você. “Como eu vou saber? Eu sou apenas um programador!”. Este, é claro, é o ponto. Você trabalha para um negócio e, ao menos que você forneça algum tipo de valor real, você é um desperdício de dinheiro. É fácil cair na armadilha de pensar que aumentos de salário são um direito. Analogamente, uma empresa tem o direito de cobrar mais por seus produtos todo ano. Mas, por sua vez, os consumidores têm o direito de **não comprar aquele produto** se o preço não os atrair.

Agora que você começou a pensar sobre quanto você custa versus quanto você entrega, quanto você acha que você precisa entregar para ser considerado um investimento de valor à empresa? Conversamos sobre o cenário do dobro de seu salário, mas isso é o suficiente? Se você entregar valor totalizando o dobro de seu salário, a empresa ficou no zero a zero. Este é um bom jeito de gastar dinheiro?

Como um ponto de referência, pense sobre a taxa de juros de uma conta poupança de um cliente comum. Não é lá grande coisa, certo? Ainda assim, é definitivamente melhor que zero. Dadas as opções, você colocaria as economias de um ano em uma conta poupança que produz 0% ou 3%? Entregar apenas o seu salário como valor não é interessante do ponto de vista de uma empresa, assim como uma conta poupança de 0% é para você. Eles se comprometeram a 10 mil por mês, e você não está nem mesmo entregando valor suficiente para acompanhar a taxa de inflação da economia. Um empate neste caso é, na verdade, ainda uma perda.

Lembro de quando comecei a pensar dessa maneira. Isso me deixou paranoico no começo. Um mês se passava, e eu ficava pensando: “O que eu entreguei este mês?”. Então, eu comecei a ficar tão granular como as semanas e dias. “Fui valioso hoje?”

*Pergunte-se: “Fui valioso hoje?”*

—

Você pode tornar isso concreto. Apenas quanto valor **você adiciona**? Converse

com seu chefe sobre a melhor forma de quantificá-lo. O puro fato de você **querer** quantificá-lo será levado como uma coisa boa. Como você poderia criativamente economizar o dinheiro da empresa? Como você poderia tornar sua equipe de desenvolvimento mais eficiente? Ou e quanto aos usuários finais de seu software? Você se surpreenderá com quantas oportunidades você pode identificar se começar a fazer essas perguntas. Agora, comece a implementar algumas delas. Mantenha essa imagem em sua cabeça: **o dobro do meu salário**. Não pare até que você tenha superado aquele número para o ano.

## Faça algo

- 1) Quando empresas fazem investimentos, eles tentam se assegurar que estão usando seu dinheiro da melhor maneira possível. Simplesmente calcular o retorno de um investimento (eu coloco 100 e obtenho 120) não é o suficiente para tomar uma decisão inteligente. Dentre outros fatores, empresas têm que considerar inflação, custo de oportunidade e riscos. Especificamente não-intuitivo para aqueles que não frequentaram faculdade de economia ou administração, está o conceito de **valor do dinheiro no tempo**. Com o risco de estar simplificando demais, é algo assim: um dólar hoje vale mais que um dólar no ano que vem, porque um dólar hoje pode ser usado para **gerar mais dólares**.

A maioria das empresas estabelece uma barra da **taxa de retorno**, abaixo da qual um investimento não será feito. Investimentos devem render uma porcentagem acordada em um período de tempo acordado, ou eles não são feitos. Este número é chamado de **taxa mínima**.

Descubra qual a taxa mínima de sua empresa, e aplique isso a seu salário. Você é um bom investimento?



## CAPÍTULO 26

# Uma pedrinha em um balde d'água

O que aconteceria se você se levantasse e saísse de seu escritório para nunca mais voltar? Conheço vários programadores que sentem um certo prazer ao imaginar essa cena. Você apenas se levanta, dirige-se à sala de seu chefe, e comunica sua demissão. **Vou mostrar para eles por que eles precisam de mim!** É quase um sonho para ajudá-lo a atravessar os dias realmente ruins, mas obviamente não é uma atitude muito inteligente para se ter com você o tempo todo.

Além disso, não é verdade. Pessoas deixam empresas todos os dias. Muitas delas são despedidas. Muitas escolhem se demitir. Algumas ainda tentam viver suas ilusões e saem sem aviso prévio. Mas em alguns casos, as empresas que eles deixam, na verdade, sentem um impacto significativo depois de suas saídas. Na maioria dos casos, mesmo em posições críticas, o efeito é surpreendentemente baixo. Sua presença no trabalho é, para a empresa, como uma pedrinha em um balde d'água. É claro que o nível da água é mais alto como resultado. Você faz as coisas. Você faz sua parte. Mas, se você tirar a pedrinha do balde e se afastar para olhar a água, você não consegue realmente ver a diferença.

Não estou tentando deixar você deprimido. Todos nós precisamos sentir que nossas contribuições significam alguma coisa. E elas significam. Mas passamos tanto tempo sendo um **eu**, que podemos facilmente esquecer que todos os outros são um **eu** também. Todos os funcionários de sua empresa andam por aí, um ser sensível e autônomo, presos a esta coisa chamada **ego**, que é a única janela pela qual eles veem seus empregos. Pense assim: se você se demitisse amanhã, a diferença seria (em média) não mais nem menos impactante que se algum de seus colegas saísse.

Uma vez trabalhei para um CIO que era um dos mais poderosos em uma das maiores empresas do mundo. Ele e sua equipe (da qual eu fazia parte) recebiam todos os prêmios e estabeleciam todos os padrões de TI da empresa. Esse era um cara que, obviamente, tinha descoberto algum tipo de elixir mágico e o estava distribuindo em almoços e jantares que ele oferecia.

Um dos poucos conselhos que eu já recebi deste CIO — e eu o ouvi várias vezes — é que você nunca deveria ficar muito confortável. Ele declarou que, todo dia ao acordar, ele intencional e explicitamente lembrava-se que ele podia ser tirado de seu pedestal a qualquer dia. **Hoje pode ser o dia**, ele dizia.

Seus funcionários olhavam para ele, incrédulos. Não. Hoje não pode ser o dia. As coisas estão indo tão bem. Tem muita coisa acontecendo para você.

*Tenha cuidado para não se cegar com o próprio sucesso.*

—

Este era o ponto. Humildade não é apenas algo que desenvolvemos para alegar que somos mais espirituais. Ela também nos permite ver nossas ações mais claramente. O que nosso CIO estava nos ensinando era que **quanto mais bem sucedido você é**, é mais provável que você cometa um erro fatal. Quando várias coisas estão acontecendo para você, é menos provável que você questione seu próprio julgamento. Quando a maneira como você sempre fez sempre funcionou, é menos provável que você reconheça que um jeito novo pode funcionar melhor. Você se torna arrogante, e com isso desenvolve pontos cegos. Quanto mais insubstituível você **pensa** que é, mais substituível você é (e você se torna menos desejável).

Sentir-se insubstituível é um mau sinal, especialmente sendo um desenvolvedor de software. Se você não pode ser substituído, isso provavelmente significa que você realiza tarefas de tal modo que os outros não conseguem fazer. Mesmo que todos nós gostaríamos de ser considerados algum tipo de gênio especial, poucos desenvolvedores de softwares são tão inigualáveis que, de fato, **deveriam** ser insubstituíveis.

Já ouvi vários programadores meio que brincando sobre criar uma “segurança de emprego” com código não manutenível. E vi programadores realmente tentarem fazer isso, por incrível que pareça. Em todos os casos, essas pessoas viraram **alvos**. Claro, era assustador para a empresa finalmente demiti-los. Tentar ser insubstituível é uma estratégia defensiva que cria um relacionamento hostil com seu empregador (e com seus colegas) onde antes não existia.

Usando essa mesma lógica, tentar ser **substituível** devia criar um relacionamento de trabalho não hostil. Todos nós somos substituíveis. Aqueles de nós que abraçam essa ideia e mesmo trabalham em direção a isso, na verdade, diferenciam-se e, não intuitivamente melhoram suas próprias chances.

## Faça algo

- 1) Faça um inventário do código que você escreveu ou que você mantém e todas as tarefas que você realiza. Anote qualquer coisa em que a equipe é completamente dependente de você. Talvez você seja o único que entende totalmente do processo de desenvolvimento de sua aplicação. Ou há uma seção do código que você escreveu que é especialmente difícil para o resto da equipe entender.

Cada um desses itens vai para sua lista de tarefas. Documentar, automatizar, ou refatorar cada pedaço de código ou tarefa, de modo que eles possam ser facilmente compreendidos por qualquer um de sua equipe. Faça isso até que você complete a lista original. Compartilhe proativamente esses documentos com sua equipe e com seu líder. Certifique-se de que os documentos sejam guardados em algum lugar para que permaneçam de fácil acesso à equipe. Repita esse exercício periodicamente.





## CAPÍTULO 27

# Aprenda a amar manutenção

Muitos anos atrás, eu estava engajado em fazer um centro de desenvolvimento de software para 250 pessoas, do zero. Começamos com um prédio vazio e fomos designados a contratar e montar toda a equipe de desenvolvimento. Nesse processo, a gente se deparou com um desafio inesperado. Todos queriam fazer novos sistemas. Ninguém queria manter sistemas antigos. Queríamos criar um novo ambiente com uma nova cultura energizada, então tínhamos de prestar atenção ao que nossos novos funcionários **queriam**, se iríamos começar no caminho certo.

Todo mundo gosta de criar. É quando sentimos que nos é dada a oportunidade de realmente deixar nossa marca em um trabalho. Sentir que nós somos seu dono. Expressarmo-nos através de nossa criação. Nós também tendemos a acreditar que os projetos novos são os mais visíveis para nossas empresas. As pessoas que constroem a nova geração são as que devem receber o mérito, certo? Eu sabia que essa atitude prevalecia entre os programadores com quem eu havia trabalhado anteriormente. Mas, lidando com mais de duzentos desenvolvedores, vi isso levado a um extremo que eu nunca esperava.

Embora desenvolvedores de software sejam tipicamente criativos, pessoas amantes de liberdade, a “sociedade” de programadores é surpreendentemente como a de castas. Programadores querem ser designers, que querem ser arquitetos, e assim por diante. Dar manutenção não proporciona nenhuma *badge* nem um papel maior, como um arquiteto, para contar para seus pais e colegas de faculdade.

Então, os fatores de motivação são a habilidade de ser criativo e a chance de dar passos em direção a uma promoção. O engraçado é que trabalhar em projetos novos **não** necessariamente é o melhor lugar para fazer ambos.

Dar manutenção geralmente significa viver em um mundo cheio de sistemas velhos e deteriorados, e usuários finais insistentes. Uma vez que o software é tido como feito, os departamentos de TI geralmente se focam em reduzir o custo de manutenção desses sistemas, de modo que possam buscar a maneira mais barata possível para mantê-los funcionando.

Isso geralmente se soma a pouquíssimos recursos destinados a cuidar dos sistemas e a investimentos insignificantes de dinheiro direcionado a rejuvenescê-los.

Trabalhar em um projeto novo, por outro lado, é onde você começa com um mundo novo, limpo e verde. Em uma empresa bem administrada, cada projeto contribui para fazer, ou economizar, dinheiro, assim os projetos são geralmente financiados de modo suficiente para serem concluídos (embora experiências possam variar aqui). Não há um campo minado de código antigo obrigando os programadores a andarem cuidadosamente nas pontas dos pés para poderem desenvolver funcionalidades corretamente, com menos empecilhos do que se estivessem trabalhando em um sistema legado. Em suma, as circunstâncias nos projetos novos são geralmente muito mais ideais.

Se eu lhe der mil reais e pedir que me traga uma xícara de café, ficarei muito triste se você voltar com menos mil reais e nenhuma xícara de café. Eu ficarei triste até se você me trouxer um café realmente bom, mas demorar duas horas. Se eu não lhe der dinheiro e pedir uma xícara de café, ficarei extremamente grato se você efetivamente voltar com um café, e serei compreensivo se não. Trabalhar em um projeto novo é como o primeiro cenário. Manutenção é como o segundo.

Quando não temos as restrições do código legado ruim e de falta de fundos, nossos gerentes e clientes legitimamente podem esperar mais de nós. E, em projetos novos, há uma melhoria de negócio esperada. Se nós não entregarmos, nós falhamos. Uma vez que nossas empresas estão contando com essas melhorias, eles vão frequentemente apertar as rédeas naquilo que for criado, como, e para quando. De repente, nosso *playground* de criatividade começa a parecer mais uma operação

militar — qualquer movimento nosso é ditado de cima.

*Manutenção pode ser um lugar de liberdade e criatividade.*

—

Mas no lado da manutenção, tudo o que é esperam que façamos é manter o software rodando sem problemas e com o mínimo custo possível. Ninguém espera nada chamativo da equipe de manutenção. Tipicamente, se tudo estiver indo certo, os clientes ficarão alheios a seu trabalho. Consertar erros, implementar pequenas solicitações de funcionalidades, e mantê-las rodando. Isso é tudo que você precisa fazer.

E se um erro aparece e exige que se refaça uma parte da aplicação? Isso é tudo parte do conserto de erros, certo? O design deve estar antigo e embolorado, e janelas quebradas devem estar espalhadas pelo sistema.

### **TEORIA DAS JANELAS QUEBRADAS**

Para conhecer a teoria das janelas quebradas, você pode ler o livro *The Pragmatic Programmer* [7].

Aí está uma oportunidade de colocar suas refatorações em teste. Quão elegante esse sistema pode ser? Quanto mais rápido você pode consertar ou aprimorar esta seção na próxima vez devido à refatoração que você está fazendo agora?

Enquanto você continuar a mantê-lo rodando e respondendo às solicitações dos usuários em tempo hábil, o modo de manutenção é um lugar de liberdade e criatividade. Você é líder de projeto, arquiteto, designer, programador e testador. Você pode modelar suas habilidades criativas da forma de que gosta, e é você quem vai suportar os sucessos ou as falhas do sistema.

Quando você está fazendo a manutenção de um sistema, você também pode planejar melhorias mais visíveis. Seu sistema web de 3 anos de idade pode não levar vantagem de algumas das novas interfaces impertinentes de usuário disponíveis para os browsers modernos. Se você pode trabalhar entre manter o sistema rodando e consertar os erros, você poderia visivelmente aprimorar a experiência do usuário com o sistema. Adicionar algumas funcionalidades bem colocadas que seus clientes não estavam esperando não é tão diferente do que surpreender sua esposa com flores, ou, enquanto criança, limpar a casa quando seus pais estão fazendo compras. E, sem a burocracia de um projeto em pleno desenvolvimento, você se surpreenderá com

o quanto você pode se encaixar nesses pequenas frestas. Seus clientes também se surpreenderão.

Uma vantagem oculta de dar manutenção é que, diferente do ambiente contratual de muitos dos projetos atuais, o programador que dá manutenção geralmente tem a oportunidade de interagir diretamente com seus clientes. Isso significa que mais pessoas saberão quem você é, e você terá a chance de construir uma base mais larga de defensores em seu negócio. Isso também o coloca em um lugar especial para verdadeiramente aprender o funcionamento interno do seu negócio.

Se você é responsável por uma aplicação em sua totalidade, sempre trabalhando com seus usuários finais através dos problemas e questões, há mais chances de que, mesmo sem muito esforço, você entenda o que a aplicação realmente faz tão bem quanto muitos de seus usuários. Regras de negócio são implementadas em uma lógica que mesmo pessoas de negócio não conseguem entender. Já vi muitas situações em que só os programadores que entendiam como um processo específico funcionava em uma empresa. Ninguém mais tinha exposição direta ao código daquela lógica.

A grande ironia em relação à separação de projeto versus manutenção é que trabalhar em projeto novo é manutenção. Assim que sua equipe de projeto escreveu a primeira linha do código, cada funcionalidade adicional está sendo enxertada em uma base de código vivo. Claro, o código deve ser o mais limpo ou deve ser menor do que se você estivesse trabalhando em uma aplicação legada, mas o ato básico é o mesmo. Novas funcionalidades são adicionadas e erros são consertados no código existente. Quem sabe como fazer isso melhor e mais rápido do que alguém que realmente adotou o ato de dar manutenção e estabeleceu como missão aprender a fazer isso direito?

## Faça algo

- 1) **Meça, melhore, meça** — Para a aplicação ou o código mais crítico que você mantém, faça uma lista de fatores **mensuráveis** que representam a qualidade da aplicação. Isso pode ser o tempo de resposta, número e exceções não tratadas que são lançadas durante o processamento, ou tempo de atividade da aplicação. Ou, se você cuida do suporte, não avalie diretamente a qualidade da **aplicação**. O tempo que você leva para responder ao chamado é uma parte importante da experiência do usuário com a aplicação.

Escolha os mais importantes desses atributos mensuráveis, e comece a medi-los.

Depois que você tiver uma boa base, estabeleça um objetivo realista e melhore a performance da sua aplicação (ou sua própria performance) para alcançar esta meta. Depois de ter feito uma melhoria, meça mais uma vez para verificar que você realmente fez a melhoria desejada. Se sim, compartilhe isso com sua equipe e seus clientes.

Escolha uma outra métrica e faça isso mais uma vez. Depois da primeira, você descobrirá que isso se torna divertido, como um jogo. Aprimorar coisas de modo mensurável como este se torna viciante.



## CAPÍTULO 28

# Maratona de oito horas

Uma das diversas fontes de controvérsia a respeito do movimento *Extreme Programming* é sua declaração inicial de que os membros da equipe não deveriam trabalhar mais que quarenta horas por semana. Esse tipo de conversa realmente contraria gerentes escravocratas que querem espremer o máximo de produtividade possível de suas equipes. Isso pode chatear até mesmo os programadores. O número de horas de trabalho contínuo se torna parte da masculinidade do desenvolvedor, como quantas cervejas alguém pode virar em uma cervejada.

Bob Martin (<http://www.objectmentor.com>) , um dos grandes nomes da comunidade de Extreme Programming, inverteu essa sentença de tal forma que a tornou muito mais tolerável por ambas as partes, enquanto ainda se mantinha fiel ao intento original de Kent Beck. Martin renomeou **quarenta horas de trabalho semanais** para “maratona de oito horas”. A ideia é que você deveria trabalhar com tanto afinco que não seria possível continuar por mais tempo do que oito horas.

Antes de você mergulhar na maratona, por que a ênfase em manter o número de horas baixo de qualquer maneira? Este capítulo é sobre terminar coisas. Não

deveríamos estar conversando sobre trabalhar por **mais** horas?

Quando se trata de trabalho, menos realmente pode ser mais. O primeiro motivo citado pelos Extreme Programmers é que, quando estamos cansados, não conseguimos pensar tão efetivamente quanto quando estamos descansados. Quando estamos esgotados, não somos tão criativos, e a qualidade do nosso trabalho se reduz dramaticamente. Começamos a cometer erros estúpidos que acabam nos custando tempo e dinheiro.

*Projetos são maratonas, não corridas de velocidade.*

—

A maioria dos projetos dura bastante tempo. Você não consegue manter o ritmo de um arranque e terminar uma maratona. Embora sua produtividade a curto prazo vá aumentar significativamente conforme conta as horas, a longo prazo você vai decair tão gravemente que o tempo de recuperação será mais longo do que os ganhos de produtividade que você aproveitou durante suas semanas de oito horas.

Você também pode pensar em seu tempo da mesma maneira que você pensa de seu dinheiro. Quando eu era um adolescente, trabalhava meio período por salário mínimo. Eu teria sido muito feliz se tivesse a quantidade de dinheiro que eu **gasto** agora. Eu tenho tanto mais dinheiro agora do que quando adolescente, que eu tendo a ser menos consciente com como eu gasto cada dólar. De alguma forma, eu conseguia sobreviver naquele tempo. Eu tinha um lugar para viver, um carro para dirigir, e comida para comer.

Eu tenho as mesmas coisas hoje. E eu não tenho um estilo de vida particularmente extravagante agora. Aparentemente, quando dinheiro era escasso, eu dava um jeito de ser mais eficiente com o que eu ganhava. E o resultado final era essencialmente o mesmo.

Nós tratamos os recursos escassos como sendo mais valiosos, e fazemos uso mais eficiente deles. Somando-se à questão de dinheiro, podemos aplicar isso para nosso tempo. Pense no quarto dia da sua última semana de setenta horas de trabalho. Sem dúvidas, você estava realizando fazendo um baita esforço. Mas a partir do quarto dia, você começa a afrouxar. **São 10h30 da manhã, e eu sei que estarei aqui por quatro horas depois que todos os outros vão para casa. Quer saber, acho que vou dar uma olhada nas últimas notícias de tecnologia.**

Quando você tem muito tempo para trabalhar, seu tempo de trabalho se reduz significativamente em valores percebíveis. Se você tem setenta horas disponíveis,



cada hora é menos preciosa para você do que quando você tem quarenta horas disponíveis.

Quando o valor do dólar sofre uma inflação, são necessários mais dólares para comprar a mesma coisa. Quando o valor da **hora** é deflacionado, você precisa de mais horas para **fazer** coisas. A maratona de oito horas de Bob Martin estabelece uma limitação e lhe oferece uma estratégia para lidar com essa restrição. Você vai trabalhar e pensa: **Eu só tenho oito horas! Vai, vai, vai!** Com barreiras estreitas nos horários de início e fim, você naturalmente começa a organizar seu tempo mais eficazmente. Você pode começar com um conjunto de tarefas que precisam ser concluídas no dia, organizá-las em ordem de prioridade, e começar a finalizá-las uma de cada vez.

A maratona de oito horas cria um ambiente que se parece com aquele final de semana ultraprodutivo que você possivelmente teve na época da faculdade, estudando para uma prova de uma matéria que você não estava nem aí ou deixando de lado um trabalho para a última hora por pura preguiça. A diferença é que este é um **acúmulo** limitado. Esses momentos em que você corre atrás são extremamente produtivos, porque o tempo se torna escasso e, portanto, bastante valioso. A maratona de oito horas é um método de estar sobrecarregado frequentemente, sem ter que ficar acordado a noite inteira à base de *Ritalina* e bebendo *Coca Cola*.

Como trabalhadores mentais, mesmo se não estamos na frente de um computador ou no escritório, podemos estar trabalhando. Você pode estar trabalhando enquanto dirige para jantar com sua esposa ou enquanto você assiste a um filme. Seus deveres ficam seguindo-o e incomodando-o.

Meu trabalho geralmente me incomoda quando eu não prestei atenção suficiente a ele. Posso ter deixado alguma tarefa escapar ou deixado-as acumular, não cuidado delas. É esse o momento que o trabalho me persegue até em casa e me cutuca quando tento relaxar. Se seu trabalho se intensifica todo dia, você vai descobrir que ele não o persegue até em casa. Não somente você está deliberadamente se impedindo de trabalhar horas-extras, mas sua mente vai realmente **permitir** que você pare de fazer horas-extras.

Faça o orçamento de seu trabalho cuidadosamente. Trabalhe menos, e você realizará mais. Trabalho é sempre mais bem-vindo quando você se deu um tempo longe dele.

## Faça algo

- 1) Certifique-se de que você durma bem hoje. Amanhã, tome o café da manhã e, então, comece a trabalhar em um horário exato (de preferência, um pouco mais cedo que o usual). Trabalhe intensivamente por quatro horas. Tire uma hora para almoçar. Então, trabalhe por mais quatro horas tão intensamente que você fique absolutamente exausto e não consiga fazer mais. Vá para casa, relaxe, e divirta-se.

## CAPÍTULO 29

# Aprenda a falhar

Como programadores, sabemos que, quanto mais cedo no processo de desenvolvimento conseguimos descobrir falhas de software, mais robusto ele vai ficar. Testes de unidade nos ajudam a desentocar erros estranhos o mais cedo possível. Se encontrarmos erros bizarros em nosso código, e se isso acontecer cedo, ficaremos felizes. Embora eles signifiquem uma pequena falha em nosso papel como desenvolvedores – afinal cometemos uma falha de programação – descobri-los cedo e frequentemente é um bom sinal da saúde do software que você vai ter.

Somos ensinados a permitir que nossos erros de programação sejam grandes e bagunçados desde cedo. Você quer saber quais eles são e em qual momento eles acontecem, para que você faça as correções necessárias. Quando você está escrevendo código, você não desvia do seu caminho para esconder as pequenas falhas de software que estão destinadas a aparecer durante o desenvolvimento. Esta é a forma de o código conversar com você. Aquelas pequenas falhas são parte do processo de fortalecimento do software.

As pequenas falhas que encontramos também nos ensinam que tipo de falhas

esperar. Se você nunca andou em um campo minado antes, você pode não saber as protuberâncias ou sujeiras nas quais **não** pisar. Se seu software não tem dado erro regularmente, você pode não saber onde os problemas podem estar. E além disso, você pode até programar muito defensivamente, colocando diversas verificações, mas seria o mesmo que se estivesse programando às cegas.

Além disso, claro, é importante programar defensivamente. A qualidade do software é realmente colocada à prova quando as coisas dão errado. É inevitável que **alguma coisa** vai acontecer para um cenário que você não percebeu. *Seg-faults* e telas azuis em produção mostram que os programadores não fizeram um bom trabalho de lidar com as situações que eles não puderam prever.

Os mesmos princípios se aplicam ao emprego. Alguém que faz trabalhos manuais é realmente colocado à prova quando surgem os erros. Aprender a lidar com os erros é uma habilidade altamente valiosa e difícil de se ensinar. Como músico improvisador de jazz, aprendi que toda nota errada é no máximo um tom distante de uma nota certa. O que torna os improvisos ruins é quando o improvisador não sabe o que fazer quando a nota errada aparece. Com uma banda de um lado e o público do outro, o som de uma nota errada é suficiente para travar um amador. Até os mestres tocam notas erradas. Mas eles se recuperam de tal forma que o ouvinte não consegue notar se a coisa toda não foi intencional.

Todos nós vamos cometer erros estúpidos no trabalho. Faz parte do ser humano. Cometemos erros que fazem os clientes verem *stack traces*. Nós nos penitenciamos quando cometemos erros graves de design. Ou, pior ainda, dizemos as coisas erradas para as pessoas da nossa equipe, chefes e clientes. Traçamos maus compromissos ou fazemos afirmações falsas sobre o que somos capazes de fazer. Ou acidentalmente damos um mau conselho à equipe sobre como resolver um problema técnico, desperdiçando horas de seu tempo.

Porque todos nós cometemos erros, e também sabemos que todo mundo comete erros. Logo, com razão, não julgamos uns aos outros nos erros que cometemos. Julgamo-nos sobre como lidamos com esses erros inevitáveis.

Seja um erro técnico, de comunicação, ou de gerência de projeto, as seguintes regras podem ser aplicadas:

- Levante o assunto logo que souber. Não tente esconder. Em desenvolvimento de software e testes, erros descobertos antes são um problema menor do que erros descobertos tarde. O mais cedo que você expuser o que tiver feito, menos negativo o impacto deve ser.

- Leve a culpa. Não tente procurar por um bode expiatório mesmo se você encontrar um que sirva. Mesmo se a culpa não for completamente sua, assuma a responsabilidade e continue em frente. O objetivo é passar desse ponto o mais rápido possível. Um problema precisa de uma resolução. Prolongá-lo discutindo de quem é a culpa apenas atrasa a solução.
- Ofereça uma solução. Se você não possui uma, ofereça um plano de ataque para descobrir uma solução. Fale em termos concretos e prazos previsíveis. Se você tiver conduzido sua equipe a um canto, dê prazos para quando você retornará com a avaliação do esforço necessário para corrigir o problema. Metas concretas e alcançáveis, mesmo se pequenas e imateriais, são importantes nesse estágio. Não apenas elas mantêm as coisas em funcionamento, de mau para bom, mas também ajudam a reconstruir a credibilidade no processo.
- Peça ajuda. Mesmo se você for o único culpado de um problema, não deixe seu orgulho piorar a situação, recusando ajuda em uma solução. Os membros de sua equipe, chefes e clientes vão olhar para você de um ângulo mais positivo se você conseguir manter uma atitude saudável e deixar seu ego de lado enquanto a equipe ajuda a encontrar a saída. Muito frequentemente, sentimos um senso de responsabilidade que nos faz suportar orgulhosamente um fardo pesado demais, e acabamos travados até que alguém intervenha.

Pense na última vez em que você teve um problema em um restaurante. Talvez você esperou demais até o **prato errado** finalmente chegar à sua mesa. Pense sobre como o garçom reagiu à sua reclamação.

*Momentos de estresse oferecem as melhores oportunidades de construir fidelidade.*

—

A reação errada seria se o garçom desse desculpas ou culpasse os cozinheiros. A reação errada seria o garçom sair andando e reenviar o pedido, e ficar fora do seu campo de visão por um tempo, enquanto você continua lá com fome e pensando quando é que sua comida vai chegar. Claro, a reação **realmente** errada seria o garçom chegar com um prato que ele já sabe que é o errado, esperando que você não repare ou não reclame.

A diferença entre como uma empresa nos trata quando houve um erro pode ser a última palavra em construir fidelidade (ou destruir). Um erro com o qual for bem lidado pode tornar-nos consumidores mais fiéis do que seríamos se nunca tivéssemos

experienciado o problema de serviço. Lembre-se disso com **seus** clientes quando cometer erros no trabalho.

## CAPÍTULO 30

# Diga “Não”

O jeito mais fácil de não cumprir com seus compromissos é se comprometer com o que você sabe que não conseguirá cumprir. Eu sei que, evidentemente, isso soa óbvio, mas nós fazemos isso todo dia. Estamos à vista de todos e não queremos desapontar nossos líderes, então aceitamos o compromisso de realizar trabalhos impossíveis em prazos inviáveis.

*Dizer “sim” para evitar desapontamento é apenas mentir.*

—

Dizer “sim” é um hábito viciante e destrutivo. É um mau hábito mascarado como bom. Mas existe uma grande diferença entre uma atitude “posso fazer” e a deturpação da capacidade de alguém. Esta última causa problemas não apenas para você mas para as pessoas para quem você está fazendo promessas. Se eu sou seu gerente e pergunto se você pode reescrever o modo como rastreamos remessas no sistema da empresa até o fim do mês, é provável que eu tenha perguntado especificamente sobre o fim do mês por alguma razão. Alguém provavelmente perguntou **a mim** se

isso poderia ser feito até lá. Ou deve haver outra mudança crítica de negócios que estamos tentando fazer que depende do sistema. Então, munido de sua garantia de que você pode conseguir nesse prazo, eu saio e falo aos meus clientes que isso será feito.

Dizer “sim” desse jeito é tão bom quanto mentir. Não estou dizendo que é malicioso. Nós mentimos para nós mesmos tanto quanto para aquelas pessoas com quem nos comprometemos. Afinal, dizer “não” nos faz sentir mal. Somos programados a sempre querer obter sucesso. E, dizer que nós **não conseguimos** fazer alguma coisa dá a sensação de que falhamos.

O que nós, humanos, não conseguimos absorver é que “sim” não é sempre a resposta certa. E “não” é raramente a resposta errada. Eu digo absorver, porque eu acho que todos nós **sabemos** que isso é verdade. Afinal, nenhum de nós quer receber falsos compromissos.

A inabilidade de dizer “não” é uma parte comum da cultura indiana. Empresas sem experiências em terceirizações *offshore* quase sempre vão em direção a isso. Você aprende com o tempo a farejar incertezas e faz as perguntas certas. Conversas do tipo “mais um dia para isso estar pronto” naturalmente o treinam a sondar mais profundamente. E isso não é apenas parte da cultura de TI. Quando morei em Bangalore, deixei de ir ao trabalho e fiquei em casa não menos do que cinco vezes, à espera de uma instalação do modem a cabo que nunca aconteceu. Nas primeiras três vezes, a empresa nem possuía os aparelhos necessários para a instalação quando fizeram o agendamento. Mas eles não queriam me desapontar. Eu disse que esperava ter o modem instalado até a próxima semana, então eles me prometeram que a instalação seria feita, sabendo muito bem que isso não ia ser possível na próxima semana.

Embora a intenção fosse boa, as ramificações foram negativas. Eu eventualmente fui desagradável com os instaladores e até mesmo fiz com que viessem à minha casa para fazer a instalação em um feriado. Não confiei na promessa de que o fariam “amanhã, depois do feriado”. O fato de terem falhado repetidamente com seus compromettimentos destruiu qualquer possibilidade que eu tinha de confiar neles. Além disso, eu fui desenvolvendo uma atitude de hostilidade em relação a eles.

Por outro lado, o que acontece quando lhe pedem que faça uma tarefa crítica e você diz que não pode? Como gerente tanto de equipes *onshore* como de *offshore*, posso lhe dizer que “não” se tornou uma fonte de alívio para mim. Se um membro da equipe tem a coragem de dizer “não” quando esta é a verdade, então eu sei que quando eles dizem “sim”, eles realmente querem dizer isso. Um comprometimento



de uma pessoa assim será mais creditável e terá mais peso. Se eles realmente alcançarem seus objetivos com os quais se comprometeram, não vou questioná-los quando disserem que **não podem** alcançar outro.

Se alguém sempre diz “sim”, ele é ou incrivelmente talentoso ou está mentindo. Este último é geralmente o caso.

“Eu não sei” também é uma ótima coisa a dizer quando apropriado. Essa pode ser uma resposta para se você consegue cumprir tal prazo e precisa de tempo para pesquisar a tarefa antes de estabelecer qualquer compromisso. Ou você pode ser questionado sobre como uma tecnologia funciona ou como alguma parte do código do seu projeto é implementada. Assim como no caso dos comprometimentos, não saber a resposta de algo dá a sensação de uma pequena falha. Mas seus colegas de trabalho e seus chefes terão mais fé em você quando você alegar saber alguma coisa. Você notará que, quando encontrar um guru real em determinada área, ele nunca terá medo de admitir quando não sabe algo. “Eu não sei” não é uma frase para gente insegura.

A mesma coragem também pode aparecer convenientemente quando estiver lidando com decisões vindas de cima. Quantas vezes você viu uma decisão de tecnologia ditada por um chefe que fez os membros da equipe sentarem ao redor da mesa silenciosamente olhando para seus sapatos e esperando uma chance para escapar da sala de reunião, para poderem reclamar uns com os outros? Gerentes são geralmente o alvo do fenômeno **A roupa nova do rei** (*Emperor's New Clothes*). Todo mundo sabe que uma decisão é ruim, mas todos estão com medo de se pronunciar. Entretanto, eu não contrato meus funcionários para serem robôs. São os que se pronunciam e oferecem uma sugestão melhor que se tornam meus acessórios confiáveis.

Não vá muito além com o jogo do “não”. Atitudes de “poder fazer” ainda são bem-vindas, e é bom ter metas ambiciosas. Se você não tem certeza se consegue fazer alguma coisa, mas quer tentar, diga isso. “Isso será um desafio, mas eu gostaria de tentar” é uma resposta maravilhosa. Às vezes, é claro, a resposta é simplesmente “sim”.

Tenha coragem o suficiente para ser honesto.

## Faça algo

1) Karl Brophey, um crítico, sugere manter uma lista de tudo o que você se comprometeu:

- O que foi pedido com uma data limite?

- Com o que você se comprometeu?
- Se você está atrasado, registre tanto o que você fez e o que lhe mandaram fazer.
- Registre quando você cumpriu o dever.

Examine isso diariamente. Comunique quando você falhará assim que você perceber. Examine isso mensalmente — qual sua média de sucesso? Com qual frequência você está correto?

## Não entre em pânico

Eu comecei minha carreira de programador por causa de vídeo games. Desde a época dos cartuchos com meu Commodore 64, eu fui conquistado pelas experiências imersivas e interativas. Eu costumava ter vergonha de admitir isso, mas agora percebo que não é nada para se envergonhar. Para mim, jogos de computador fizeram do ambiente *on-screen* (do sistema operacional, eu acho) um ambiente em que eu me sentia confortável e disposto.

Meu jogo favorito de todos os tempos era **Doom**, da ID Software. Eu era apaixonado pelas modos de *one-on-one*, *jogador vs. jogador* e combates mortais. Os jogadores se conectavam via modem ou conexão serial, e lutavam entre si em um pequeno e acelerado ambiente. Eu fiquei bom nos combates mortais do Doom. Eu dizia que isso devia ser o que eu melhor sabia fazer da minha vida. O jogo de combate mortal é surpreendentemente complexo. É tanto técnico como psicológico — como uma mistura frenética de xadrez e esgrima.

Assim como para a maioria das habilidades, uma ótima maneira de se tornar bom é assistir os mestres. Lá na minha época de Doom, havia um mestre que possuía

o pseudônimo irônico de “Noskill” (Sem habilidades, em inglês). Noskill era, de fato, o campeão de Doom. Pessoas dos Estados Unidos e Canadá pagavam tarifas telefônicas de longa distância para tentar a sorte contra ele. Esses combates eram gravados. Eu assistia a todos.

Não levou muito tempo para eu aprender seu segredo. Claro, no geral ele era bom no jogo, mas havia uma chave óbvia para seu sucesso: ele nunca entrava em pânico. Doom era o tipo de jogo em que uma partida podia terminar em, literalmente, segundos depois de começar. Era realmente rápido. Eu me lembro do meu primeiro jogo de combate mortal. Começa, morre, começa, morre, começa, morre. Quando eu finalmente consegui ficar vivo por mais do que dois segundos, peguei-me correndo sem rumo, mal sendo capaz de acompanhar onde eu estava.

Mas Noskill nunca agia dessa forma. Não importava quão difícil era a situação, você podia notar ao assistir às gravações que ele estava sempre relaxado e sempre pensando no que fazer em seguida. Ele sempre aparentava estar ciente de como seu contexto atual se encaixava no formato geral da partida.

*Heróis nunca entram em pânico.*

—

Agora, se você pensar em outros jogos, em particular esportes, você vai reconhecer que todos os melhores jogadores compartilham dessa qualidade. Aliás, até mesmo os personagens que admiramos nos livros, na televisão e nos filmes compartilham dessa qualidade. Heróis nunca entram em pânico. Eles são sempre as pessoas que podem ter uma bomba nuclear caindo sobre sua cidade, ou ter um acidente de avião, e conseguem organizar um grupo, ajudar os sobreviventes, ser mais esperto que o inimigo, ou pelo menos, apenas não se desabar em lágrimas.

Isso se estende à vida real também. Apesar do meu melhor planejamento, minha vida profissional tem sido uma corrente de emergências e desastres. Projetos são concluídos bem tarde. Softwares dão problemas, custando dinheiro e credibilidade dos meus chefes. Eu digo a coisa errada para o vice-presidente errado e ganho um inimigo político. Na maioria das vezes, essas coisas vêm em ondas todas juntas, nunca uma por vez.

Em meus piores momentos, eu entro em pânico. Eu travo e, na melhor das hipóteses, consigo pensar taticamente. Reajo a cada pequeno movimento sem a clareza de considerar o grande cenário.

Mas olhando para trás para, literalmente, **cada** desastre, nenhum teve um impacto duradouro e notável em mim ou em minha carreira. Portanto, por mais que

eu tenha ficado em pânico, depressivo ou chateado nessas situações aparentemente desastrosas, nenhuma foi **realmente** um verdadeiro desastre.

O que o pânico trouxe para mim? Qual foi a vantagem de reagir negativamente para cada uma dessas situações? Nada. O que o pânico realmente me deu foi uma inabilidade de agir no meu melhor nas vezes em que eu realmente **precisava** dar o meu melhor.

Agora eu tenho que admitir que não entrar em pânico em situações estressantes é mais fácil de se dizer do que fazer. É quase como dizer para alguém: “simplesmente seja feliz”. É claro, é um bom conselho, mas como se faz isso? Como você evita entrar em pânico quando as coisas parecem estar desabando? Para responder esta questão, talvez ajude pensar um pouco sobre por que nós **entramos** em pânico.

Nós entramos em pânico porque perdemos perspectiva. Quando algo tem dado errado, é difícil não focar toda nossa atenção no problema. De certa maneira, esse é um bom jeito de resolver problemas. Infelizmente, isso também faz o problema, não importa quão pequeno seja, aparentar ser mais importante do que realmente é. E com o problema aumentando, e os níveis de estresse lá em cima, nossos cérebros se desligam.

Quem é o pior usuário de computador que você conhece? Para mim, é provavelmente um dos meus sogros (eu sei quem, mas sou esperto o suficiente para não dar nomes aqui). Imagine aquela pessoa sentada com seus computadores tentando terminar um projeto quando uma mensagem de erro começa a aparecer com qualquer coisa que eles tentam fazer. Todos nós já vimos isso acontecer. Usuários sem experiência ficam rapidamente frustrados e se desesperam. Então começam a freneticamente clicar e arrastar coisas pela tela, ignorando as mensagens que potencialmente os poderiam ajudar conforme elas continuam aparecendo repetidamente. Eles eventualmente ficam tão afobados que precisam pedir ajuda, mas geralmente não antes de bagunçar uma ou duas coisas adicionais no computador.

Não me leve a mal, mas eu quero que você imagine essa situação com alguém que você conheça, que seja apropriado para o papel principal, e quero que você ria disso. Esse tipo de comportamento é realmente ridículo, não? É cômico.

Mas, por mais genuinamente engraçado que isso seja, o que nós acabamos de imaginar foi um cenário da vida real em que uma pessoa, trabalhando fora de sua zona de conforto, encontra um problema e entra em pânico. Não é diferente da forma como eu reagi quando projetos se atrasaram ou eu acidentalmente causei um bug no sistema, ou desapontei um cliente. É apenas um contexto diferente.

Então, aqui está como eu estou aprendendo a não entrar em pânico. Quando

algo ruim acontece, e eu começo a sentir aquela sensação estressante de afundar que me levaria ao pânico, eu me comparo àquele frustrado usuário de computador inexperiente, e rio de mim mesmo. Eu analiso a situação pela perspectiva da terceira pessoa, como se eu estivesse ajudando aquele membro da família que está frustrado com seu desastre. Problemas aparentemente difíceis tornam-se repentinamente mais fáceis. Situações aparentemente ruins de repente não são tão ruins. E eu sempre encontro a solução simplesmente olhando na minha cara, da mesma forma que a caixa de diálogo de erro geralmente lhe diz exatamente o que você deve fazer. Se você apenas tiver a presença de espírito para ler a mensagem de erro, o problema pode ser solucionado.

## Faça algo

- 1) Mantenha um diário do pânico. A chave para capturar o pânico antes que ele aconteça é desenvolver uma consciência intensificada em tempo real de suas percepções e emoções conforme elas acontecem. Eu tirei a sorte grande ao aprender a fazer isso analisando minhas reações a situações depois do fato. Eu não sou esperto o suficiente para manter uma linha constante de análise dos meus pensamentos conforme eles acontecem, mas eu descobri que se eu praticar a análise “offline”, eu melhoro cada vez mais para realizá-la em tempo real.

Dizer que você se empenhará mais em analisar suas reações e realmente fazer isso são duas coisas bastante diferentes. Manter um diário vai ajudá-lo a adicionar estrutura ao processo. Todo dia, em um horário específico (use um calendário como um lembrete!), abra um arquivo de texto e registre qualquer situação que o levou a sentir pânico, mesmo se foi pouco. Uma vez por semana, veja a lista da semana passada e mantenha um estoque dos últimos impactos de cada situação que induziram ao pânico. A situação valeu o pânico? Qual teria sido a reação mais produtiva para a situação? O que um herói, em um filme dramático sobre sua vida, teria feito em vez de entrar em pânico?

Depois de alguma prática, você descobrirá que a análise começa a acontecer **enquanto** o pânico emerge. Conforme você racionalmente explora as razões do seu pânico em tempo real, você descobrirá que o pânico, por si próprio, vai para o banco de trás, e eventualmente se dissipa.

## CAPÍTULO 32

# Diga, faça, mostre

O jeito mais fácil de nunca tornar algo pronto é nunca se comprometer com nada. Se você não tem um prazo, não há nenhuma pressão ou incentivo suficiente para concluir algo. Isso é ainda mais verdadeiro quando esse **algo** que deve ser feito não é 100% excitante.

O instinto de, até mesmo, um mau gerente geralmente diz que é importante planejar. Para alguns desenvolvedores, a invocação da palavra **planejamento** é motivo de alarme. Reuniões intermináveis com seus chefes criando e analisando gráficos do Microsoft Project, que ninguém entende ou usa, são uma causa válida de alarme. Então, os tecnólogos geralmente tentam compensar, como rebelião contra o excesso de planejamento, agindo conforme suas próprias decisões, em vez das que foram estabelecidas no plano.

Planejamento não é um remédio tão ruim, daqueles que precisamos tampar o nariz para engolir. Ele pode ser uma experiência libertadora. Quando você tem muito a fazer, um plano pode fazer a diferença entre um começo de dia de trabalho confuso e ambíguo, e a confiança de uma mente limpa ao atacar as tarefas.

Planos não precisam ser grandes e prolongados. Uma lista em um documento de texto ou um e-mail é perfeitamente suficiente. Planos não precisam cobrir um longo espaço de tempo. A possibilidade de começar o dia e responder a questão “O que você vai fazer hoje?” é um ótimo primeiro passo. Conheço muitas pessoas cujos dias começam tão agitados que eles provavelmente falhariam nesse teste. Um bom primeiro passo seria achar tempo nessa tarde e listar tudo que você quer concluir no próximo dia de trabalho, e organizar em uma lista de prioridades. Tente ser realista quanto ao que pode fazer em um dia, embora seja provável que você erre e se sobrecarregue.

Você pode ser tão detalhista ou vago quanto quiser com seu plano de um dia. Eu tive um colega de quarto na faculdade chamado Chris que acordava todas as manhãs e, mesmo com o risco de se atrasar para sua primeira aula, planejava meticulosamente seu dia, especialmente com foco em seu horário de praticar piano (ele estava cursando piano e jazz). Sua agenda já era bastante rígida com a seleção de aulas que ele tinha que frequentar. Mesmo assim, Chris realmente fazia um plano sobre como ele poderia usar aqueles quinze minutos **entre** as aulas para encaixar sua rotina de treinos que podiam ser feitos rapidamente.

Muitas de suas aulas eram no mesmo prédio, de modo que era comum ter bastante tempo de lazer entre elas para socializar ou pegar alguma bebida nas máquinas. Nesse tempo, Chris preferia se dedicar às escalas ou a treinar sua audição, enquanto o resto de nós ficava sentado esperando a próxima aula começar. Ele até mesmo dividia seu horário em múltiplos segmentos de três a cinco minutos, para poder treinar mais de um exercício em um dado período de dez minutos. O Chris acabou se tornando um dos músicos mais respeitados em nossa cidade. Claro que o talento natural tinha algo a ver com isso, mas desde então eu tenho a crença de que ele planejou e executou seu caminho para a elite musical.

Então, você fez seu planejamento. Pode não ser tão detalhado como o do Chris, mas é o suficiente para responder a pergunta de o que você vai fazer com seu dia. Quando você for trabalhar amanhã, pegue a lista e comece o primeiro item. Trabalhe pela lista até o almoço, e depois continue de onde parou e tente concluí-la.

À medida que você completa cada item da lista, marque “FEITO”. Use letras maiúsculas. Esteja feliz. Ao fim do dia, olhe para sua lista de coisas FEITAS e sinta que você conquistou alguma coisa. Não apenas você sabia o que ia fazer nesse dia, mas agora você sabe que o **fez**.

Se você não concluiu tudo, não se preocupe. Você sabia que não estaria certo em relação a quanto caberia em um dia de qualquer forma. Apenas mova os itens



incompletos de hoje (se ainda forem relevantes) para a lista de amanhã, e comece o processo de novo. É um processo estimulante. É rítmico. Isso permite que você divida seus dias e semanas em séries de pequenas vitórias, cada uma incentivando-lhe para a outra. Você descobrirá que isso não apenas lhe dá visibilidade para o que está conquistando, mas lhe dará, na verdade, **mais feitos** do que se você não estivesse olhando as coisas tão de perto.

Uma vez estabelecido um ritmo de planos e ataques, você está pronto para começar a pensar em termos de semanas e até mesmo meses. É claro, quanto mais comprido for o espaço de tempo para o qual você está planejando, maior o nível que seu planejamento deve ter. Pense nos planos diários e semanais como planos de batalhas táticas, com planos de trinta, sessenta, e noventa dias focando em objetivos mais estratégicos que você quer alcançar.

O próprio ato de pensar sobre o que você quer alcançar em noventa dias é algo não natural desenvolvedores de software.

Nós somos pessoas táticas. Forçar-se a imaginar um estado final para seu sistema, para o processo de sua equipe, ou de sua carreira depois de noventa dias vai trazer coisas à tona que você nunca esperaria. O campo, visto de cima, mostra-nos várias coisas diferentes do que a visão do chão. É difícil no começo, mas mantenha-se firme a isso. Como todas as boas habilidades, isso se torna mais fácil com a prática, e os benefícios serão visíveis tanto para você como para aqueles que trabalham com você (mesmo se eles não sabem o que você está fazendo).

*Relatórios de status podem ajudá-lo a se vender e mostrar resultados.*

—

Você deveria começar a comunicar seus planos para seus chefes. O melhor momento para isso é depois de ter executado pelo menos um ciclo do plano. E — este é um ponto importante — comece a fazer isso antes de eles lhe pedirem. Nenhum chefe em sã consciência ficaria chateado ao receber um e-mail semanal **sucinto** de um funcionário, estabelecendo o que foi feito na semana anterior e o que está planejado para a próxima. Receber esse tipo de mensagem regular não solicitada é a paz de todo chefe.

Comece comunicando-se semana por semana. Quando você estiver confortável com esse procedimento, comece a trabalhar em planos de trinta, sessenta e noventa dias.

A longo prazo, concentre-se no progresso mais alto nível e impactante que você deseja aplicar em seus projetos ou nos sistemas que você mantém. Sempre estabeleça

esses planejamentos como propostas para seu chefe, e peça por feedback. Com o tempo, essas tentativas de antecipação vão requerer menos puxões de orelha dos seus chefes, conforme você aprende quais itens geralmente não precisam ser editados e quais são assuntos que precisam de mais atenção.

O fator mais crítico para se ter em mente com tudo que vai em um planejamento é que tudo deve ser contabilizado depois. Cada item deve estar ou visivelmente alcançado, atrasado, removido, ou substituído. Nenhum item pode estar fora dessas categorias. Se itens aparecem em um plano, e nunca são mencionados de novo, as pessoas vão parar de confiar em seus planos, e você estará indo contra a eficácia do planejamento. Mesmo se o resultado é **ruim**, você também deve comunicá-lo. Todos nós cometemos erros. O jeito de se diferenciar é tornar públicos seus erros e inabilidades, e pedir ajuda para resolvê-los. Seguir consistentemente as tarefas de um plano vai gerar a merecida impressão de que nenhum trabalho importante está se perdendo no meio.

Mantenha esse procedimento e de repente, aos olhos de seu chefe, você expôs seu lado estratégico. Criar e executar planos demonstra que você não é apenas um robô que digita códigos, mas que você é um **líder**. É esse tipo de produtividade independente que as empresas precisam conforme elas reduzem a sobrecarga.

Um benefício final de comunicar seus planos é que seus compromettimentos ganham mais credibilidade. Se você disser que você vai fazer algo, e realmente o faz e o mostra pronto, você desenvolve a reputação de ser um **cumpridor**. Com credibilidade vem a influência. Imagine que você queira introduzir um novo processo, tal como uma prática de desenvolvimento ágil, em sua empresa, ou então você quer trazer uma nova tecnologia. Com sua habilidade comprovada de estabelecer e alcançar compromissos, vão conceder-lhe mais liberdade para tentar coisas novas.

Em nosso centro de software de Bangalore, tínhamos uma equipe que trabalhou em turnos da noite por mais de um ano. Dos sete membros da equipe, dois estavam sempre no turno da noite. Eles revezavam semanalmente, de modo que toda terceira ou quarta semana, cada membro mudaria para o horário das 19h às 3h da manhã. A equipe estava cada vez mais frustrada e esgotada. Mas a equipe estava atuando em um papel de apoio crucial, e os clientes internos que ficavam nos Estados Unidos estavam convictos de que não conseguiriam continuar sem a ajuda em tempo real, ao vivo, do grupo de Bangalore.

Então, a equipe formulou um plano de ataque. Eles olharam para seus vários processos de suporte e suas medidas associadas, e montaram um plano para tanto voltar para o turno de um dia **quanto para** fazer melhorias significantes para seus

clientes, simultaneamente. Como líder de operações ativas no centro de software, eu os ajudei a sofisticar o plano e estive presente (como apoio moral) na proposta formal que eles fizeram ao chefe dos Estados Unidos.

Eles sabiam que isso seria um assunto delicado para o chefe, que teria que responder aos seus clientes americanos pessoalmente.

Naturalmente houve muita trepidação entre os membros da equipe conforme a reunião começou. Entretanto, o chefe da equipe ficou tão impressionado e feliz que imediatamente assinou a proposta, e a equipe pôde colocar seu plano em ação. Dentro de poucas semanas, a loucura tinha acabado e todo mundo estava de volta para o turno do dia.

A solidez desse plano para, não apenas lidar com a mudança na carga horária, mas também para as melhorias estratégicas na performance da equipe, inspirou grande confiança em seus chefes e, eventualmente, em seus clientes. O gerente da equipe usou o plano quando comunicou a mudança a seus clientes. E a equipe prosseguiu assim. Dentro de meses, estavam funcionando dentro de um novo nível de eficiência. Desde então eles conquistaram tanta credibilidade e confiança que receberam mais e mais propriedade e independência em seus trabalhos.

A equipe usava seu plano como uma resposta completa a um problema. Eles se dirigiam ao seu chefe não com reclamações, mas com propostas de soluções.

Seus líderes querem que você tenha independência e propriedade. Fazer, executar, e comunicar seus planos irá ajudá-lo a obter ambos.

## Falhar e copiar

*por Patrick Collison*

Larry Wall escreveu que as características de um ótimo programador são a preguiça, a impaciência e a arrogância. Eu não sei se isso é inato ou se pode ser adquirido com o tempo. De qualquer forma, não é fácil usar essa informação para se tornar um programador melhor. Então, em vez de olhar as características, devíamos olhar para as atividades que vão ajudar a trazer melhorias.

Se eu tivesse que escolher duas, diria falhar e copiar.

Eu falho mais do que a maioria dos programadores que conheço. Certamente, a maior parte dos meus projetos falha. Dentro do meu diretório `Projects` na minha máquina, eu tenho um monte de ideias interessantes negligenciadas, cada uma com a mesma probabilidade de dar certo quanto a de um peixe fugir do seu aquário. Assim como em famílias, projetos bem sucedidos são parecidos, enquanto cada projeto mal sucedido falha de um jeito próprio.

E apesar de ser quase um clichê dizer que a falência de uma empresa dá uma ótima experiência, eu não ouço a mesma ideia com relação à programação.

(Eu sou bom nos dois, todavia, tive empresas que faliram também.)

Falência comercial tende a construir um tipo de experiência bem direta. Você aprende a importância de guardar dinheiro, ou você se torna mais determinado. Mas, em se tratando de programação, não é muito a experiência de falir que é valiosa e que importa, mas sim o conhecimento adquirido enquanto se trabalha nos projetos que podem falhar.

Quando comecei a programar, muito de meu tempo era gasto em tentativas falhas de escrever todos os tipos de coisas incríveis: sistemas operacionais, sistemas de arquivos, máquinas virtuais, reimplementação de protocolo de rede, interpretadores e compiladores JIT. A maioria deles nunca funcionou direito, e aqueles que funcionaram eram mesmo assim muito ruins. É claro, mesmo ignorando os aspectos técnicos, a maior parte estava condenada desde o princípio. Eu não tenho certeza de qual fração de 1% é a taxa de sucesso para se criar um novo sistema operacional, mas é pequena.

Ainda assim, para mim, esses projetos são a forma mais gostosa da programação. São problemas fundamentais de engenharia de software. É tudo questão de um perde e ganha entre espaço, velocidade, confiabilidade e complexidade, sem ter um código cheio de bugs em vista.

Esses são os tipos de problemas nos quais você pode ficar enfiado durante meses e mesmo assim não conseguir sair com alguma coisa que funcione — como aconteceu comigo.

Não sei exatamente por quê, mas as pessoas que estão aprendendo programação hoje não parecem muito ter esse tipo de experiência.

Eu acho que isso pode ser, ao menos parcialmente, devido à ascensão de sistemas web. Poucos dias atrás, uma pessoa da Hacker News perguntou se ainda existia alguém interessado em escrever software *client-side*. É um exagero, mas não está tão longe da verdade. E ei, software para web é mesmo bem legal.

Do ponto de vista de um programador, contudo, essa mudança tem uma desvantagem. Apps web raramente envolvem duros desafios técnicos, até que você precise enfrentar os problemas de compatibilidade do Internet Explorer.

Em outras palavras, a barreira de entrada para o fracasso é maior. Você tem que ser bem sucedido antes.

Portanto, especialmente dado este movimento em direção aos sistemas web, eu penso que é importante procurar por projetos sujeitos a falhas.

E quanto a copiar? Para se tornar um melhor programador, qualquer pessoa lhe dirá que você deve ler códigos bons. Mesmo que eles presumidamente não queiram dizer isso literalmente (isso seria realmente chato), “ler” permanece sendo, no fundo, uma ideia errada: é passivo. Em vez disso, eu acho que você deve ativamente copiar, sem dó e sem ter vergonha disso.

Isso se aplica a muitas coisas, é claro. Hunter S. Thompson não apenas lia bons livros; ele redigitava Hemingway e Fitzgerald. E os manuscritos mais velhos conhecidos do Bach eram transcrições que Bach fazia de trabalhos de outros organistas. Um dos casos mais famosos, talvez, seja o de Gates pegar programas em latas de lixo em Harvard.

Não é muito difícil entender como isso ajuda. Copiar constrói memória muscular. Você sente a nuance e a forma do original — o tipo de detalhe que seria perdido em uma leitura rápida.

No caso de código, também há um benefício menos óbvio — mas significativo. Copiar permite ir mais fundo com projetos que possam vir a falhar. Isso pode ser uma transcrição franca da, digamos, implementação de uma *hash-table* (o que tornou meu primeiro interpretador “menos pior”) ou um design que apenas inspirou e se moldou ao original (como, digamos, o Linux foi pelo Minix).

Na melhor das hipóteses, isso leva a um tipo de ciclo virtuoso de falhar e copiar, que se torna um autoaprimoramento preguiçoso.

Você lida com algo duro, tropeça em algum problema insuperável, copia a solução de outra pessoa e, ei, agora você sabe o que fazer, o que quer que seja.

Nesse roubo irrestrito, à medida que você sinceramente absorve várias técnicas, você descobre como usá-las de uma nova maneira. Eu não sei ao certo o que Picasso quis dizer com “Bons artistas copiam, enquanto ótimos artistas roubam”. Talvez ele estava apenas sendo intencionalmente perverso, mas o significado inicial é o que eu sempre presumi.

Programação é cheia de ideias estranhas. Usar nomes mais curtos e menos descritivos frequentemente produz código que é mais legível no geral. As linguagens mais poderosas geralmente têm, de longe, menos conceitos do que as menos poderosas. E falhar e copiar pode ser o melhor jeito de produzir um trabalho bem sucedido e original.

*Patrick Collison é um aluno do MIT*



## CAPÍTULO 33

# Percepções e perssepisões

É confortável bancar o idealista e fingir que você não liga para o que as outras pessoas pensam de você. Você não pode se deixar acreditar nisso. Você **deve** se preocupar com o que as outras pessoas pensam de você. Percepção é realidade. Supere isso.

Você provavelmente conhece o clichê da velha questão filosófica, “Se uma árvore cai no meio da floresta mas ninguém está lá para ouvi-la cair, ela fez um som?” A correta resposta para isso é: “Quem se importa?”

Quero dizer, a queda provavelmente produziu um som. Não é uma resposta muito interessante em um nível metafísico, mas ela provavelmente o fez. Mas, se ninguém a ouviu cair, o fato de ela ter produzido um som não importa.

A mesma coisa vale para seu trabalho. Se você arrasa e ninguém está lá para ver, será que você arrasou mesmo? Quem se importa? Ninguém.

Na subcultura burocrática de TI, na Índia, eu estava abismado com como eles realmente não conseguiam **entender** essa simples verdade. Quase todo mundo com quem lidei lá não entendia por que devia ser importante que seus chefes, por exemplo, soubessem o que ele estava fazendo. Se **você** soubesse que você estava melho-

rando cada vez mais, isso se refletiria nos comentários de sua performance, em sua avaliação e em seu salário. Eles tinham se enganado ao pensar que a forma como os outros os viam era, de alguma forma, subserviente à **verdade** seja lá qual fosse.

Essa coisa da verdade... o que é isso? O que é bom e o que é ruim em um sentido absoluto?

A resposta é que não há um bom ou ruim absoluto, ao menos não em termos de julgar quem é melhor em um trabalho que exige criatividade e conhecimento. Como você define o que faz uma música ser boa? E quanto a um bom quadro? Você pode ter suas próprias definições, mas eu duvido que eu concordaria com elas. São subjetivas.

*Avaliações de performance nunca são objetivas.*

—

Os péssimos departamentos de recursos humanos de péssimas empresas giram suas engrenagens buscando modos objetivos de mensurar as pessoas que contratam. Às vezes, eles nem implementam sistemas de avaliação “objetivos”. Todos os membros da minha equipe na Índia achavam que **eles** deveriam ser medidos dessa forma.

Não existe maneira de medir objetivamente a qualidade de um trabalhador ligado ao conhecimento, e também não há como medir objetivamente a qualidade de seu trabalho. Vá em frente. Discorde. Agora pense em seu argumento por um instante. Percebe as falhas?

Então, se a medida de sua eficácia em sua empresa (ou indústria, ou mercado de trabalho, o que for) é subjetiva, o que isso significa? Que você sempre será avaliado baseado na **percepção** que outra pessoa tem de você. O potencial de ser promovido ou de ter um aumento no salário — mesmo a decisão de se você deve continuar na folha de pagamento — depende completamente da percepção dos outros.

Subjetividade, baseada em gosto **pessoal**, implica que você não pode contar que dois pareceres serão o mesmo. Pessoas diferentes se impressionam com diferentes fatores. Alguns podem gostar de uma estrutura rígida, enquanto outros preferem mais solta, com liberdade de criação. Alguns podem preferir se comunicar via e-mail e outros, pessoalmente, ou por telefone. Alguns chefes podem gostar que seus funcionários sejam mais agressivos, enquanto outros preferem que ajam como subordinados. Você diz “biscoito” — eu digo “bolacha”.

Não se trata apenas de preferência pessoal. As pessoas interagem com você com diferentes funções e relacionamentos, e constroem suas percepções com base nas ca-



racterísticas mais importantes para fazer aquele relacionamento **em particular** funcionar bem. Se eu sou gerente de um projeto, a qualidade de seu código fonte é muito menos importante para mim do que a qualidade de nossa comunicação. Se eu sou um colega de programação, sua habilidade bruta e sua criatividade dirigem a percepção que tenho de você mais que, por exemplo, seu acompanhamento. Mas se eu sou seu chefe, a habilidade crua é fundamentalmente insignificante para mim, a não ser que você realmente **faça** algo com isso.

Nós culturalmente nos treinamos para **perceber** que lidar com a percepção é, de certa forma, uma atividade suja e vergonhosa. Mas, como você pode ver, controlar a percepção é apenas prático. Quando você explicitamente nota os fatores que dirigem as percepções que outras pessoas têm de você, você descobre com mais firmeza como deixar os clientes felizes. Você não vai impressionar seu cliente com suas habilidades de programação orientada a objeto. Você pode ser um gênio em design, mas se você não consegue se comunicar efetivamente e você não consegue completar seu trabalho a tempo, seus clientes pensarão que você é ruim. Não é culpa deles. Você é ruim.

Percepções realmente importam. Elas o mantêm empregado (ou desempregado). Elas o levam à promoção ou o deixam preso no mesmo cargo por anos. Elas lhe dão aumento ou diminuição em seu salário. Quanto mais cedo você se superar e aprender a lidar com percepção, mais cedo você estará no caminho certo.

## Faça algo

- 1) Percepções são conduzidas por diferentes fatores, dependendo de qual o público alvo. Sua mãe não se importa muito sobre quão bem você consegue programar sistemas orientados a objetos, mas seus colegas de equipe sim.

Entender o que é importante em cada relacionamento é uma parte crucial para formar percepções verossímeis naqueles com quem você interage. Pense nos diferentes tipos de relacionamento que você geralmente tem com as pessoas do escritório. Por exemplo, você provavelmente tem colegas que fazem o mesmo tipo de serviço que você. Você também provavelmente tem um chefe direto, e talvez você tenha um ou mais clientes e um gerente de projeto.

Pegue esses grupos diferentes (ou quaisquer que se aplicarem, dada a estrutura de seu ambiente de trabalho), e os liste. Próximo a cada um, escreva quais dos seus atributos mais provavelmente vão conduzir a percepção daquele grupo sobre você. Aqui está um exemplo:

Grupo	Condutores de percepção
Colegas de equipe	Habilidades técnicas, sociais e trabalho em equipe
Chefe	Espírito de liderança, foco no cliente, habilidades de comunicação, acompanhamento, trabalho em equipe
Clientes	Foco no cliente, habilidades de comunicação, acompanhamento
Gerente do projeto	Habilidades de comunicação, acompanhamento, produtividade, habilidades técnicas

Pense um pouco sobre sua própria lista. Como você poderia mudar seu comportamento com o resultado dessa lista? De que maneira você já vem ajustando seu foco conforme interage com cada grupo? E em que sentido você **não** tem ajustado apropriadamente seu comportamento?

## CAPÍTULO 34

# Guia de aventura

Com o risco de afirmar o óbvio, o aspecto mais importante para colocar sua palavra para fora em seu ambiente de trabalho é sua habilidade de se comunicar. Já se foram os dias do hacker despenteado inclinado sobre seu terminal e escrevendo códigos pela luz de seu monitor nas entranhas mais profundas do servidor. O grunhido monossilábico ocasional não vai funcionar.

Mesmo que a proposta seja perturbadora, ponha-se na mente de um chefe ou de um cliente (eu usarei apenas a palavra **cliente** neste capítulo para me referir a ambos).

Eles são responsáveis por algo tão gravemente importante que, em último caso, precisam confiar em algum cara assustador de TI para a implementar algo. Eles fazem o que podem para ajudar a movimentar as coisas, mas no fim das contas eles estão à mercê desses programadores. Além disso, eles não têm ideia de como controlá-los ou mesmo como se comunicar inteligentemente sobre o que é que estão fazendo. Nessa situação, qual seria o atributo mais importante que eles estariam procurando em um membro da equipe? Eu aposto o preço deste livro que não era se eles tinham

memorizado os últimos *design patterns* ou quantas linguagens de programação eles sabiam.

*Eles estavam procurando por alguém que os deixassem confortáveis quanto ao projeto no qual estão trabalhando.*

*Seus clientes têm medo de você.*

—

Esses gerentes e clientes estavam conversando sobre ter um pequeno segredinho: eles têm **medo** de você. E com uma boa razão. Você é esperto. Você fala uma linguagem oculta que eles não entendem. Você os faz sentir estúpidos com seus comentários sarcásticos (nos quais você talvez nem teve a intenção de ser sarcástico). E seu trabalho é frequentemente o último e mais importante pedágio entre a concepção de um projeto e seu nascimento.

Sua função é ser o guia de aventura de seu cliente, através dos implacáveis terrenos do mundo da tecnologia da informação. Você vai deixar seus clientes confortáveis enquanto os guia em um lugar não familiar. Você os fará suspirar e os levará onde eles querem ir, enquanto evita as partes obscuras e decadentes que você encontrou no passado.

Pessoas que não programam são, na média, tão inteligente quanto os programadores. (Isso quer dizer que a maioria deles não é muito inteligente, mas poucos realmente são). Há grandes chances de seu cliente ser tão esperto quanto você, mas não saber como programar. Está tudo bem. Você provavelmente não sabe fazer o que ele, por sua vez, faz em seu dia a dia. Por isso há dois de vocês, e ambos estão sendo pagos para aparecer no trabalho.

Eu mencionei um pouco sobre inteligência porque as pessoas da computação, com muita frequência, presumem que quem não sabe como operar um computador não é inteligente. Dizer isso tão explicitamente assim soa idiota, mas é a verdade de todos os preconceitos. Contudo, essa sensação é tão arraigada em tantos de nós que nem mesmo sabemos quando a sentimos. Tentar controlar isso explicitamente não funciona.

Meu conselho é reverter o relacionamento. Em vez de se achar um gênio da computação, descendente do céu dos computadores para salvar seu pobre cliente do purgatório, vire a mesa ao contrário. Se você está, por exemplo, trabalhando em uma seguradora, pense em seus clientes como um especialista no assunto de seguros, dos quais **você** tem o que aprender para fazer **seu** serviço.

Tendo dito isso, você precisa estar ciente de que seus clientes podem precisar atenuar os tópicos técnicos um pouco quando vocês estiverem discutindo sobre assuntos relacionados a software. Há um balanço delicado entre muito tecnólogo e muito ignorante.

“Por que toda essa tarefa sobre como tratar seus clientes? Eu achei que nós íamos conversar sobre como fazer propaganda de mim mesmo.” Se você trabalha em uma típica empresa de TI, muito do orçamento que o mantém empregado vem de área de negócio.

Quando estão sendo tomadas decisões sobre promoções e *staff*, o melhor advogado que você pode ter é um cliente que não quer ficar sem você. Do outro lado, imagine o impacto de um cliente que pensa que você é condescendente. Seu cliente representa a necessidade do negócio, e você é pago para fornecer o suprimento disso. Não se esqueça disso.

## Faça algo

- 1) **Olhe para si mesmo** — você é aquele programador mau humorado e ogro que todo mundo teme? Você tem certeza? Eles têm medo de lhe dizer?

Vá ao seu e-mail e procure exemplos de e-mails que você enviou a colegas de trabalho menos técnicos, chefes e clientes. Conforme você lê, tente ver as coisas a partir do ponto de vista do destinatário. Se já passou algum tempo desde que você enviou as mensagens, você poderá se ver como um observador na terceira pessoa.

Melhor ainda, mostre os e-mails para sua mãe. Conte para ela que alguém com quem você trabalha vai enviar as mensagens para um cliente, e pergunte a ela como essas mensagens fariam-na sentir.

- 2) **Pule a cerca** — encontre uma oportunidade para ser jogado em uma situação em que você **não** é especialista e, portanto, depende daqueles que são.

Se você tem dois pés esquerdos, imagine-se em um time de futebol. Se você tem dois polegares esquerdos, imagine que você é parte da Equipe Nacional de Tricô. Como você gostaria que seus colegas de equipe se comunicassem com você?



## CAPÍTULO 35

# Eu iscrevu mto beim

Os dias do programador ogro e monossilábico acabaram. Se as empresas querem ter dificuldades em se comunicar com seus programadores, eles os enviarão para um outro continente e em um fuso horário diferente e se comunicarão apenas via e-mail e telefone.

Portanto, comunicação é importante. Na lista de tarefas que você tem que fazer para permanecer empregado, isso pode parecer meio artificial, bobo ou trivial. Talvez você se sinta como se tivesse voltado às aulas do ensino médio. Tudo bem. Desta vez, você pode realmente prestar atenção.

Vamos passar pela mais chata primeiro: gramática e ortografia são importantes. Você provavelmente tem um diploma em um assunto avançado como engenharia ou ciência da computação, e aqui estou ensinando-o a soletrar. **Coragem!**

Mas, pelo menos nos Estados Unidos, temos um problema.

De acordo com uma reportagem da Comissão Nacional de Escrita (<http://www.writingcommission.org/report.html>) , mais da metade das empresas entrevistadas levam em consideração as habilidades de escrita quando estão tomando decisões

tanto para contratar como para promover. 40% dos entrevistados do setor de serviços disseram que um terço ou menos de suas contratações correspondia às habilidades de escrita desejadas.

Quando você realmente dá um passo para trás e olha para o grande cenário, habilidades de escrita são necessárias e estão em falta.

Como você sabe, a força de trabalho mundial está se distribuindo globalmente. Conforme a tendência continua, vai chegar uma hora — para alguns, o momento é agora! — quando **a maior parte** da comunicação no local de trabalho será feita em forma escrita, seja por mensagem instantânea ou e-mail.

Você escreverá **muito**. Se parte tão grande do seu trabalho envolverá escrever, é melhor você ser bom nisso. Mais do que nunca, as percepções sobre você serão formadas com base em sua habilidade de escrita. Você pode ser um ótimo programador, mas se você não consegue se expressar em palavras, você não será tão eficaz em uma equipe distribuída.

A habilidade de escrever cria tanto uma percepção superficial de você, como possibilita uma real compreensão de como sua mente funciona. Se você não consegue organizar seus pensamentos em sua língua materna de modo que os outros consigam claramente os entender, como nós podemos esperar que você consiga fazer isso em uma linguagem de programação? A habilidade de formular uma ideia e guiar o leitor por um processo do pensamento até uma conclusão lógica não é muito diferente da habilidade de criar um design limpo e uma implementação de sistema que futuros mantenedores poderão entender.

Não se trata de ser julgado. Se você possui membros de sua equipe em fusos horários diferentes em locais distantes, escrever pode ser o único jeito que você tem de explicar como você programou algo, ou em que seus colegas devem trabalhar.

*Você é o que você consegue **explicar**.*

—

Comunicação, especialmente por escrito, é o gargalo pelo qual todas as suas maravilhosas ideias devem passar. Você é o que você consegue **explicar**.

## Faça algo

- 1) Comece a manter um diário de desenvolvimento. Escreva um pouco em cada dia, explicando em que você tem trabalhado, justificando suas decisões de programação e examinando difíceis decisões técnicas e profissionais. Mesmo que



você seja seu primeiro (ou único — depende de você) público, preste atenção na qualidade de sua escrita e em sua habilidade de se expressar claramente. Ocasionalmente, releia registros antigos e os critique. Ajuste seus novos registros com base no que você gostou ou desgostou nos anteriores. Não apenas sua escrita vai melhorar, mas você pode também usar esse diário como uma forma de fortalecer seu entendimento das decisões que você tomou, e como um lugar ao qual se referir quando precisar entender como e por que você fez tal coisa.

- 2) Aprenda a digitar. Se você ainda não é um datilógrafo, faça um curso ou baixe algum programa para ensiná-lo. É mais provável que você se sinta confortável e natural em sua escrita se você está confortável com o jeito como você o faz. É claro, com todas as coisas que você terá que escrever, você pode já economizar tempo aprendendo a digitar rapidamente.



## CAPÍTULO 36

# Estar presente

Você possui a vantagem de estar frente a frente com seus líderes e seus clientes. Não desperdice a oportunidade.

Quando eu morava em Bangalore, como diretor técnico (*CTO — Chief Technical Officer*) do nosso centro de desenvolvimento de software, eu tive a desagradável experiência de relatar ao gerente sobre quem eu não apenas desgostava (e que não gostava de mim), mas quem estava nos Estados Unidos. Tínhamos conversas tensas no telefone, tarde da noite ou logo pela manhã, que se tornavam cada vez mais frustrantes por causa de ruídos de fundo ou porque a linha caía. Eu escrevia e-mails longos e descritivos tentando ajudar a encurtar a distância e a diferença de fuso horário, mas era apenas ignorado. E se eu reclamasse sobre ser ignorado, eu seria criticado por escrever e-mails longos. Parecia uma situação sem saída.

Minha empresa, na época, tinha um processo anual de revisão de performance, no qual os chefes listavam o desempenho de seus funcionários e suas (assim chamadas) necessidades de desenvolvimento. No topo da minha lista de necessidades de desenvolvimento estava algo chamado **presença**.

Mas presença, nesse contexto, é uma palavra de cunho empresarial para descrever uma característica bem confusa sobre liderança. Trata-se da qualidade imensurável de ter sua presença marcada — principalmente em situações face a face. Isso também inclui a qualidade igualmente imensurável de levar a si mesmo como um líder.

Quando eu ficava sentado conversando sobre minha revisão de performance (ao telefone) com minha amada chefe, eu colocava o telefone no mudo quando ela dizia “presença”. Eu não queria que ela me ouvisse rindo. Eu ficava imaginando se ela podia ver a minha cara de metade careta, metade sorriso, que eu não conseguia tirar do rosto pelo resto da conversa. Ambos, ela e eu, sabíamos que o problema **real** era presença na forma mais comum do termo: eu apenas não estava nos Estados Unidos com todo mundo.

A maioria de nós que só queria compartilhar o que pensávamos, não gostava dessa chefe. Ela fez pouca coisa para impor respeito, então isso já era esperado. O padrão que apareceu foi que os únicos funcionários que tinham um relacionamento realmente **negativo** com ela eram os que não estavam geograficamente no mesmo lugar que ela.

Aqueles em outros países, tais como Índia, Hungria e a Grã Bretanha (em ordem decrescente) tinham relacionamentos tensos com ela, já que estávamos não apenas separados fisicamente, mas tínhamos horários, infraestrutura, cultura e limites por causa do idioma.

Parecia que, mesmo para as pessoas dos Estados Unidos que faziam tudo o que podiam para evitar essa chefe, a proximidade física e as conversas frente a frente ocasionais eram todo o necessário para deixá-la confortável. E, é claro, o fenômeno de “o que os olhos não veem, o coração não sente” era rapidamente validado quando eu colocava meus pés na Índia.

Além de apenas contar uma história sobre uma gerente ruim, você pode aprender algo mais dessa experiência. A proximidade física é uma vantagem no lugar de trabalho.

Pense sobre a última vez que um parente ou um amigo, que não era da área da computação, pediu-lhe ajuda com um problema do computador. Você tenta ajudar pelo telefone, e se eles não estão entendendo, você fica cada vez mais agitado. **Se você pudesse mostrar para eles...** Em contraste, a comunicação cara a cara é incrivelmente eficaz. Você pode ouvir o outro lado mais claramente. Você pode usar de expressões visuais para se fazer entender, seja com gestos ou desenhos ou lousas. E todos nós expressamos **implicitamente** bastante conteúdo em nossas expressões

faciais, mesmo sem perceber isso conscientemente.

Nós não apenas vemos mais produtividade e uma comunicação mais aprimorada por meio de interação cara a cara, mas também formamos laços pessoais mais firmes. Leva muito mais tempo criar algo que você chamaria de amizade se você nunca encontra a pessoa frente a frente. Quinze anos atrás, isso era inédito para nós. Hoje em dia, com a ubiquidade da internet, é menos comum do que amizades tradicionais cara a cara. Por muitas das mesmas razões pelas quais trabalhamos com menos eficiência via telefone, e-mail e chat, nós também somos menos eficientes em construir um relacionamento dessa forma. Adicione a isso o desconforto da artificialidade de um e-mail ou de uma conversa em chat (algo de que a próxima geração provavelmente não vai se lembrar), e, na maioria dos casos, o relacionamento erguido em um ambiente de trabalho remoto vai permanecer estritamente centrado em cumprir tarefas.

Um forte relacionamento de equipe, com comunicação eficaz e em banda larga, favorece na rapidez da entrega do software. Na maior parte dos ambientes, decisões importantes sobre o projeto são feitas pessoalmente, nos intervalos para o café, e conversas paralelas. Essas são observações bastante óbvias, e a vantagem que a pessoa tem em participar disso também é bastante óbvio. O que não é tão óbvio — especialmente para nós, geeks — é a importância de ser **visto**.

Eu **nunca** entrei em um banco. Eu faço todas as tarefas bancárias ou online ou em caixas eletrônicos. Meus avós são diferentes. Eles fazem tudo o que precisam **dentro do banco**, conversando com **pessoas reais**. Eles nem gostam de conversar sobre negócios por telefone. Não é confortável para eles. Eles também conhecem as pessoas do armazém que frequentam. Eles vão e voltam várias vezes e conversam na saída. Eles não pensam em mudar de armazém (ou de banco), porque essa escolha é muito mais do que algo de peso pragmático de conveniência e custo. É pessoal.

Enquanto não existirem robôs ou programas de computador para tratarem de nossas avaliações de performance, todos os negócios continuarão sendo pessoais. Nós, pessoas, gostamos de interagir com outras pessoas pessoalmente. Alguns de nós, de qualquer maneira.

O modo de trabalho natural de uma pessoa da computação é se enfiar em um cubículo, colocar seus fones de ouvido, e entrar “no clima” até a hora de comer. Douglas Coupland, em seu livro *Microserfs* [1], conta a agradável história de uma equipe que tinha que comprar comidas achatadas para poder passar debaixo da porta do escritório de um programador em uma missão. Esse tipo de isolamento focado se tornou parte da cultura e do folclore da indústria de software.

Infelizmente para a sua carreira, isso é ruim para o mercado. Se você está trancado em um escritório, acessível apenas por telefone (se você atender) ou e-mail, e talvez até trabalhando todas as horas da madrugada e dormindo tarde, como consequência, não existe diferença entre você estar no mesmo local que seus chefes e clientes, ou estar no *offshore*. Você está perdendo uma grande oportunidade de se tornar uma figura memorável em sua empresa. Lembre-se, você precisa tornar as coisas **pessoais**, e para isso, você precisa lembrar da tendência humana natural de querer trabalhar com outros seres humanos — não por mensagem de voz, e-mail, ou mensagens instantâneas, mas pessoas reais.

*Aprenda sobre seus colegas.*

—

No atual ambiente distribuído, você pode pensar que, embora esteja no mesmo país que as pessoas com quem você trabalha, vocês não estão no mesmo estado ou na mesma cidade. Viagens regulares para ter reuniões pessoalmente são ótimas nessas situações se isso for prático para você e sua empresa. Mas a melhor coisa que você pode fazer é pegar o telefone e ligar para seus chefes e colegas. Não use microfones, e não dependa de reuniões agendadas. Você precisa tentar simular o tipo casual das conversas dos intervalos para o café que você teria se morasse ou trabalhasse no mesmo lugar; portanto, separe um tempo para conversas (aparentemente) espontâneas. Na ocasião, espere a oportunidade de tornar a conversa pessoal. Deixe que o “Como vai hoje?” siga para o “O que você geralmente faz nos finais de semana?” Tente aprender de verdade sobre as **pessoas** com quem você trabalha. Não apenas isso o firma mais em sua empresa, como também é uma forma mais enriquecedora de viver.

## **Faça algo**

- 1) Na próxima semana, pegue um dia para se forçar (com razão) a não mandar nenhum e-mail. Toda vez em que você iria, normalmente, enviar um e-mail, ou telefone para a pessoa ou, melhor ainda, vá até o escritório e converse com ela pessoalmente.
- 2) Faça uma lista de colegas, chefes e clientes com quem você não conversa o suficiente. Coloque compromissos recorrentes em sua agenda para ligar e falar com

eles (ou pelo telefone, ou pessoalmente). Torne as conversas curtas mas significantes. Use-as para falar de coisas relacionadas ao trabalho e também para simplesmente estabelecer um contato humano.





## CAPÍTULO 37

# Fale com propriedade

Todos os meus jovens sobrinhos usam o computador regularmente. Eles são, relativamente falando, bons entendedores de computador. Eles se comunicam com os amigos pelo mundo afora. Sentem-se completamente confortáveis com mensagens instantâneas, e-mail, baixar coisas da internet, e, é claro, tornarem-se públicos e todas as coisas que você faria se fosse um aluno do ensino médio.

Mas se eu fosse me exibir para eles, dizendo que meu novo computador tem um disco rígido de 10.000 RPM Serial ATA, eles podem, na melhor das hipóteses, tentar adolescentemente fingir um entusiasmo. Eles também ficariam pouco interessados se eu lhes dissesse que tenho vários gigabytes de RAM e um GPU mais rápido do que os CPUs nos sistemas que usei apenas cinco anos atrás.

Contudo, se eu lhes dissesse que eles podem rodar o mais novo jogo de tiros em primeira pessoa, em alta resolução sem que a aparência visual do jogo fique pipocando, eles se sentariam e me escutariam.

Gigahertz e rotações por minuto não são interessantes para a maioria dos meninos de 14 anos. Jogos de computador, sim.

Pessoas de negócio não estão tão interessados em gigahertz e RPM, tampouco. Eles gostam quando suas aplicações são rápidas, porque não têm que esperar enquanto estão no telefone com um cliente, ou enquanto tentam fechar as contas do trimestre.

Mas eles não ligam para quantos pedidos por segundo o servidor de sua nova aplicação customizada pode atender.

Os negócios e aqueles que lidam com ele estão interessados nos **resultados**. Portanto, promover suas realizações em qualquer outra língua que não a linguagem de negócio é ineficaz.

*Promova suas realizações na linguagem do seu negócio.*

—

Você não faria uma propaganda de um produto destinado ao público americano em alemão. Uma empresa de refrigerantes não tentaria vender uma bebida baseando-se na quantidade de corante nº8 que ela contém. O senso comum ensina que, para vender um produto para um determinado público alvo, você deve se dirigir a essa audiência em uma linguagem que eles podem tanto entender como se referirem a ela.

Como um desenvolvedor de softwares, isso significa esboçar suas realizações dentro do contexto da empresa para a qual você trabalha. Claro que você **o fez**, mas o que **foi** isso? Por que isso importava? Por que isso foi tão chamado de realização, e não apenas uma perda de tempo?

Meu palpite é que se fosse para você pensar nas conquistas do último mês, você talvez não consiga articular a resposta de **por que** elas eram tarefas úteis para se fazer. Obviamente, mandaram-lhe realizá-las, mas quais benefícios elas agregaram ao negócio?

Na General Electric, há uma lenda urbana de que o CEO original Jack Welch gostava de entrar em um elevador de um dos prédios da GE com um funcionário aleatório que tenha subido no elevador com ele. Ele se dirigia ao já amedrontado subalterno com a pergunta “Em que você está trabalhando?”, para então perguntar (e é aqui que está a alfinetada) “Qual é o benefício disso?” A moral da história é que você deve sempre ter um **discurso de elevador** pronto, só para garantir.

O que você diria se seu CEO lhe fizesse a mesma pergunta, do nada? Mesmo tendo alguns minutos para se preparar, será que você conseguiria explicar o benefício das tarefas que você está realizando ou que recentemente concluiu? Você conseguiria

se expressar em palavras de modo que um alto executivo totalmente não técnico pudesse não apenas entender, mas também apreciar?

### **Faça algo**

- 1) Faça uma lista de suas últimas realizações. Escreva o benefício de negócio para cada uma. Se há conquistas para as quais você **não consegue** escrever um benefício, pergunte ao seu chefe ou a um conhecido confiável como eles enquadrariam o benefício.
- 2) Faça seu discurso de elevador e decore-o.



## CAPÍTULO 38

# Mude o mundo

A pior coisa que alguém de seu trabalho pode perguntar sobre você é “O que ele/ela faz?” Ter de fazer essa pergunta significa que eles não sabem o que você já **fez**.

É triste, mas eu não sei o que fazia a maioria das pessoas com quem trabalhei em uma grande empresa de TI. Simplesmente não se pensa assim. Eles vão ao trabalho, cumprem seus compromissos e voltam para casa. Não há nenhum impacto marcante deixado por eles, que não seja o pedaço de código, documentos e e-mails que deixaram no histórico.

Isso é o que acontece quando você vai trabalhar sem uma **missão**. Você apenas fica sentado, esperando que lhe digam o que fazer. E, quando você realiza a tarefa mandada, as únicas pessoas que saberão o que você fez são aquelas que o requisitaram. Tudo bem se você quiser trabalhar seguindo a lógica de varejo ou mesmo se você quiser ser um programador *offshore*.

*Tenha uma missão. Assegure-se de que os outros saibam disso.*

Porém, se você quer ser um desenvolvedor de software em um país de alto custo, você precisa ir ao trabalho com uma missão. Você precisa fazer a mudança, mas não mudar a si mesmo ou seu trabalho (que é dado). Você precisa efetuar mudanças visíveis através de sua equipe, da organização, da empresa.

A mudança pode ser pequena. Você pode ter a chave do teste de unidade, ditando práticas de testes para todos os programadores de sua empresa. Ou pode ser algo maior, com a introdução radical de uma nova tecnologia, que levará a sistemas mais baratos e feitos mais rápido.

Você faz essas coisas porque você é internamente **guiado** para fazê-las. Você não pode se afastar e assistir às pessoas fazerem coisas erradas. Você sabe que tudo poderia ser melhor, então você **deve** mudá-las.

É claro, se você quer mudar o mundo, está ao mesmo tempo fadado a deixar algumas pessoas bravas. Tudo bem, contanto que suas intenções estejam certas. Não seja um idiota a respeito disso, mas também não precisa andar na ponta dos pés se fazendo de conservador sempre.

Se você **realmente** acabar eliminando algumas pessoas, poderá, pelo menos, se sentir confortável pelo fato de que não vão mais perguntar “O que você faz?”

Se você não sabe qual é sua cruzada, provavelmente não possui uma. Se ainda não está **ativamente** tentando deixar sua marca, você provavelmente não está conseguindo.

## Faça algo

- 1) Catalogue as jornadas que você pessoalmente testemunhou em seu ambiente de trabalho. Pense nas pessoas com quem você trabalhou, que agiam como se tivessem uma missão. Pense nas pessoas mais focadas e eficazes de sua empresa. Quais eram suas **missões**?

Você pode pensar em alguma delas que tenha sido inapropriada? Qual o limite entre foco e obsessão? Você já viu alguém passar desse limite?

## CAPÍTULO 39

# Faça sua voz ser ouvida

As ideias que exploramos até aqui foram razoavelmente conservadoras e focadas em ser reconhecido pelo que você faz em seu trabalho. Se você quer ser percebido, promovido, ou mesmo ficar empregado em sua empresa atual, os tópicos nos quais tocamos serão críticos.

*Mas que chato!*

O mundo está mudando. Se você quer garantir sua passagem, é preciso pensar mais alto do que os trabalhadores de TI do ano passado. Enquanto progredir de um nível 23 para o 24 na programação possa **realmente** ser seu objetivo de carreira a curto prazo, sendo um programador hoje, você deve pensar além da próxima promoção ou além até do seu emprego atual.

Observe-se mais do alto. Não se defina como um programador de uma empresa específica — afinal, é provável que você não fique no mesmo lugar para sempre — mas como um membro participante de uma companhia. Você é um artesão ou um artista. Você tem algo a compartilhar além do aplicativo de relatório de despesas que você está desenvolvendo para o departamento de recursos humanos, ou os erros que

you corrigiu no sistema de testes de sua empresa.

Empresas querem contratar especialistas. Enquanto um currículo com uma sólida lista de projetos é um bom modo de demonstrar experiência, nada é melhor, em uma entrevista de emprego, do que o entrevistador já ter ouvido falar de você. É especialmente bom se eles já ouvirem falar de você porque leram seus artigos ou livros, ou assistiram a uma palestra sua em um congresso. **Você** não ia querer contratar a pessoa que “escreveu o livro” sobre a tecnologia ou metodologia que você está tentando implantar?

Durante o tempo em que eu era saxofonista, eu tocava bastante nos clubes perto da rua Memphis's Beale. Conforme eu comecei a me entrar na área da computação, eu percebi várias coincidências entre o modo como você precisa promover seu nome na indústria música e na computação. Quando eu era um músico tentando achar emprego, as seguintes propriedades eram válidas:

- (Este é o mais importante) O melhor saxofonista nem sempre é contratado.
- Com quem você tocou é, pelo menos, um dado tão importante como quão bem você toca; músicos são **bons por associação**.
- Às vezes, os melhores músicos são deixados para trás porque todo mundo presume que eles não estarão disponíveis, porque estão muito intimidados para perguntar.
- Música funciona via efeito de rede. Se sua rede social e musical termina antes de alcançar alguém, não é provável que você seja requisitado para tocar com aquela pessoa, até que algum intermediário faça a conexão.

A indústria da computação é do mesmo jeito. Não existe um sistema objetivo para avaliar e empregar desenvolvedores de software. Ser bom é importante, mas não o leva até o fim. Nosso meio, como o musical, é uma grande, complexa rede de pessoas conectadas com outras. Quanto mais ligações você tem, maiores são suas chances de se conectar com o emprego perfeito ou de despontar em uma carreira. Limitar-se à empresa para a qual trabalha estabelece sérios limites no número de conexões que você pode formar.

Qual a melhor maneira, senão fazer publicações e falar em público, para que seu nome seja propagado e sua voz se faça ouvida? Então, como você deixa de ser um programador Zé Ninguém para ser um autor publicado e um palestrante público? Comece na web.



Primeiro, leia blogs. Aprenda mais sobre a organização dos blogs e procure se tornar um membro. Se você não sabe o que ler, pense em alguns autores dos seus livros técnicos favoritos e pesquise na web. Você provavelmente descobrirá que eles têm um blog. Assine o *feed* e também os das pessoas às quais eles linkam. Com o tempo, sua lista de feeds vai crescer ao passo que você lê e encontra links de outros blogs.

Agora abra seu próprio blog. Há vários serviços gratuitos disponíveis para se criar um blog. É extremamente simples. Comece escrevendo (e linkando) histórias que você acha interessante. Conforme escreve, você descobrirá que o universo dos blogs é por si só uma rede social — um microcosmo da rede de carreira que você está começando a construir. Seus pensamentos vão eventualmente aparecer nos feeds agregados e de outros que curtem suas postagens, que escreverão sobre isso e compartilharão suas ideias.

O blog é um campo de treinamento. Escreva na internet como se você estivesse escrevendo em uma coluna de sua revista preferida. Pratique a arte de escrever. Suas habilidades vão crescer, e você vai ganhar mais confiança.

Seus posts também vão dar exemplos de trabalho que você pode usar no próximo passo. Agora que você está escrevendo em seu próprio ambiente, você pode também enviar seus tópicos para comunidades, revistas ou até livros. Com um portfólio de sua habilidade de escrita disponível na internet, você terá bastantes exemplos de materiais para incluir em um artigo ou em uma proposta de livro. Publique, e sua rede crescerá. Quanto mais textos, você terá mais oportunidades de escrever. E tudo isso leva à oportunidade de palestrar em conferências.

Assim como nós facilmente começamos na internet nossos empreendimentos de escrita, você pode começar sua carreira de palestrante em seus encontros de desenvolvedores locais. Se você é alguém de .NET, prepare uma apresentação para o grupo local de desenvolvimento da Microsoft. Se você é um programador Linux, converse com o grupo de usuários de Linux. A prática leva à perfeição, no que tange aos discursos. Assegure-se de dedicar muita reflexão quando estiver preparando sua palestra. Não seja superficial. Mesmo que você esteja palestrando para apenas um pequeno grupo de pessoas em sua cidade natal, é lá onde você vive e trabalha. Um trabalho **realmente** bem feito, eventualmente vai ser recompensado. Você descobrirá que, se você der a quantidade certa de atenção, localmente, essas pequenas palestras não serão diferentes das conferências nos congressos das grandes indústrias. Estas são, obviamente, o próximo passo lógico.

Com todas essas formas de levar seu nome e sua voz para fora, a dica mais crucial

de todas é começar tão logo quanto você achar que está pronto. A maioria das pessoas se vende mais barato do que poderia.

Você **tem** algo a ensinar. Você nunca vai se sentir 100% pronto, então talvez você deva começar agora.

## **Faça algo**

- 1) Se você ainda não tem um blog, crie um agora mesmo. Vá a um dos vários sites de hospedagem grátis e monte um.

Agora crie um arquivo de texto em seu computador. Nele, faça uma lista de possíveis tópicos para seu blog. Esses serão futuros artigos que você irá escrever. Não se limite a ideias épicas. Mire em ideias sobre as quais você pode escrever em dez a vinte minutos. Pare quando a lista tiver dez itens (ao menos que você esteja inspirado — continue).

Salve o arquivo mas deixe-o aberto. Se você reiniciar, abra novamente o arquivo. Você tem três semanas. Cada dia, escolha um item da lista e escreva um artigo. Não pense muito sobre isso. Apenas escreva e publique. Nos artigos, coloque links para outros blogs com artigos relacionados. Conforme você lê a lista para escolher o tópico do dia, sinta-se à vontade para acrescentar ideias no arquivo. Depois dessas três semanas, escolha seus dois artigos favoritos e envie-os a sites novos moderados por usuários, como o *Digg* e o *Reddit*.

Se você ainda tem ideias em sua lista, continue a escrever.

## CAPÍTULO 40

# Construa sua marca

Construção de marca tem duas partes: realmente fazer sua marca, de modo que as pessoas a reconheçam, e ter certeza de que aquela marca é associada a traços positivos. Reconhecimento e respeito.

Hoje, quando vemos uma suástica, pensamos no Hitler e na Alemanha nazista. Do ponto de vista de construção de marca, aquilo foi muito bom para os nazistas. Eles conquistaram a primeira parte de construção de marca: conscientização. Mas qualquer um que seja mentalmente sã tem uma associação extremamente negativa com qualquer coisa relacionada ao holocausto. Assim, os nazistas, em última análise, falharam miseravelmente na parte de fazer associações positivas. De fato, Hitler roubou a suástica dos hindus, cometendo o crime que todas as empresas sérias lutam para não cometer. Para os hindus, que reivindicam a suástica original (ou *swasti*), é um símbolo auspicioso de bem estar. Mas agora, pelo ocidente esse ícone espiritual foi difamado. Muito reconhecimento e pouco respeito.

No lado oposto, está Charlie Wood. Ele é um cantor e compositor incrível, além de tocar o órgão B3 da Hammond em Memphis, Tennessee. Ele toca cinco noites por

semana em um clube na rua Beale. Todo mundo que o conhece ou ouviu falar dele sabe como ele é brilhante. Todos o admiram. Ele é o mais talentoso que você pode ser em se tratando de música R&B.

Mas relativamente ninguém sabe quem ele é. Nenhum reconhecimento e muito respeito.

O que **você** quer é tanto reconhecimento quanto respeito. Seu nome é sua marca.

*Seu nome é sua marca.*

—

Toda esta parte do livro é sobre como conseguir ambos. Aqui mesmo neste parágrafo, o que você deve entender é que a combinação das duas coisas é um patrimônio que vale a pena construir e proteger. Diferente de um grande e assustado departamento de marketing corporativo, que processa crianças da faculdade por sites que se apropriam indevidamente de alguma imagem ou frase, você não precisa gastar muito tempo protegendo sua marca de **outras** pessoas. A força mais potencialmente destrutiva para sua marca é você mesmo.

Não enfraqueça aquilo que você representa. Tenha cuidado com os lugares onde seu nome aparece. Não realize projetos ruins ou não envie péssimos e-mails para grupos grandes (nem faça posts ruins no blog para que toda a internet leia). Não seja um idiota. Ninguém gosta de um idiota, mesmo se eles, de alguma forma, **merecem** ser um idiota.

*O Google nunca esquece.*

—

Sobretudo, lembre-se de que as coisas que você escolheu fazer e às quais se associou, terão um impacto duradouro no significado que seu nome tem para as pessoas. Agora que muitas de nossas interações se dão via internet ou fóruns públicos, sites e listas de e-mails, muitas de nossas ações são de registro público, armazenadas em cache, indexadas e pesquisáveis — para sempre.

Você pode esquecer, mas o Google não.

Proteja sua marca com toda sua força. Defenda-a de si mesmo. É tudo que você tem.

## Faça algo

- 1) **Pesquise seu nome no Google** — digite seu nome no Google entre aspas. Veja as primeiras quatro páginas de resultados (são realmente quatro páginas de resultados?). O que alguém poderia deduzir de você depois de seguir apenas os links dessas quatro páginas? Você está bem representado nelas? Essa imagem que essas páginas constroem lhe agrada?

Faça a mesma pesquisa de novo, mas desta vez preste atenção a conversas de fóruns e listas de e-mails. Você é um idiota?



## CAPÍTULO 41

# Lance seu código

Imagine como seria mais fácil arranjar um emprego se houvesse empresas que já se utilizassem de um software que você escreveu. Você poderia dizer “Oh, vocês estão usando Nifty ++?” Eu posso ajudá-los — fui eu que inventei”. Isso seria diferente. Entrevistadores e gerentes de contratação se lembrariam de você. É isso que você quer.

Apenas uma década atrás, isso parecia uma boa ideia, mas não havia muitas oportunidades para se fazer isso. Você tinha que ter trabalhado para um fornecedor de software comercial primeiro, e suas credenciais seriam relacionadas aos produtos que você ajudou a desenvolver enquanto trabalhava para eles. Mas as coisas mudaram. Você não tem mais que trabalhar para Os Grandes para desenvolver um software popular, com marca própria.

Agora há uma outra saída: *open source*. Software open source atingiu o *mainstream*. Conforme empresas de TI começam novos projetos, o velho debate mudou de *construir vs. comprar* para *construir vs. comprar vs. baixar*. Senão aplicações inteiras, frameworks variando de pequenas bibliotecas a containers de aplicações estão

sendo lançados sob licença open source e estão, de fato, se tornando padrões.

E as pessoas que estão desenvolvendo este software, na maior parte, são pessoas como você. São pessoas sentadas em suas casas de tarde e aos fins de semana, labutando em softwares como um trabalho apaixonado. É claro, existem alguns recursos vindos de empresas para criar e dar suporte aos produtos open source. Mas a maioria dos desenvolvimentos open source é feita por programadores independentes, como um hobby.

*Qualquer um pode usar Rails. Poucos podem se dizer **contribuinte** do Rails.*

—

Embora muitos desses desenvolvedores estejam apenas se divertindo e se expressando, há alguns incentivos reais aí. Eles estão subindo na cadeia social de uma comunidade. Estão construindo nome para si mesmos. Estão construindo uma reputação na indústria. Eles talvez não o estejam fazendo de propósito, mas estão fazendo **marketing** de si próprios no processo.

Além de construir um nome para si mesmo, contribuir com software open source mostra que você é apaixonado pela sua área. Mesmo se uma empresa não usou ou ouviu falar de seu software, o fato de você tê-lo criado e lançado já é um diferencial. Pense desta forma: se você estivesse querendo contratar um desenvolvedor, você preferiria alguém que se dedica apenas em suas horas de trabalho e depois vai para casa e assiste à TV? Ou você iria preferir alguém que tem tanta paixão pelo que faz que se supera e até faz horas extras programando em finais de semana?

Contribuições open source **demonstram** habilidade. Se você está produzindo código real e contribuindo para um projeto real, isso é muito melhor em seu currículo do que apenas **dizer** que você conhece a tecnologia. Qualquer um pode escrever Rails ou Nant em seu currículo. Muitos poucos podem escrever **contribuinte do Rails** ou **commiter do Nant**.

Liderar um projeto open source pode demonstrar muito mais do que competência técnica. São necessárias habilidades em liderança, gerenciamento de *releases*, documentação, e suporte a produtos e comunidades para promover uma comunidade em torno de seus esforços. E se você fizer **essas** coisas com sucesso — em seu tempo livre, como um hobby — você será incrivelmente diferente da maioria de pessoas competindo pelos mesmos cargos. A maior parte das empresas não pode **pagar** seus programadores para fazer todas essas coisas e bem feitas. A maioria não consegue nem pagar seus desenvolvedores para fazer **algumas** dessas coisas bem. Mostrar que



you não apenas pode fazê-las, mas que você se preocupa o suficiente para fazê-las gratuitamente, demonstra um surpreendente poder de iniciativa.

Se você cria algo realmente útil, você pode até terminar famoso. Você pode ser famoso em um pequeno campo técnico — talvez famoso entre desenvolvedores Rails, por exemplo. Ou, se você tiver sorte, poderia ser **realmente** famoso mesmo fora da comunidade geek como Linus Torvalds ou... bem, como Linus. Mesmo se você não é tão famoso, lançar seu código vai, definitivamente, torná-lo **mais** famoso. Se fama significa que muitas pessoas sabem quem você é, então ter mais uma pessoa que saiba de você o torna mais famoso. E a comunidade open source é uma rede **mundial** de pessoas que, pesquisando códigos na internet, podem encontrar seu software, instalá-lo e usá-lo. Ao fazer isso, eles o conhecerão, e, conforme seu software se espalha, seu nome e sua reputação também crescerão. É disso que se trata o marketing. É o que você quer.

## Faça algo

- 1) Stuart Halloway (<http://thinkrelevance.com>) tem um workshop feito em congressos que chama de “Refactotum”. Se você tiver a chance de participar, eu realmente o recomendo, mas a essência é o seguinte:

Pegue uma parte de um software open source com testes de unidade. Rode os testes por um analisador de cobertura de código. Encontre a parte menos testada do sistema e escreva testes para melhorar a cobertura daquele código. Códigos não testados geralmente são códigos não testáveis. Refatore para torná-lo mais testável. Envie um *patch* com a sua alteração.

O legal é que isso é mensurável e pode ser feito rapidamente. Não há desculpas para não tentar.



## CAPÍTULO 42

# Seja marcante

O currículo tradicional de marketing se refere aos quatro P: produto, preço, promoção e praça (*product, price, promotion and placement*). A ideia é que, se você cobrir todas essas quatro categorias, você terá um plano de marketing completo, sendo que as quatro recebem o mesmo peso.

Mas qual o objetivo do marketing? Seu propósito é criar conexão entre os produtores e consumidores de um produto ou um serviço. Esta ligação começa com a consciência sobre um produto. O mecanismo tradicional de criar consciência é via promoções, incluindo propagandas, postagens, e seminários educacionais.

Recentemente, o mundo do marketing tem voltado sua atenção para o que é coloquialmente chamado de marketing **viral**. Isso acontece quando a ideia é marcante o suficiente para os consumidores a espalharem uns aos outros. Ela propaga como um vírus, sendo que cada consumidor infectado potencialmente pode infectar muitos outros.

Marketing viral não é preferido apenas porque é caro enviar postagens ou comprar um tempo de propaganda na televisão. Sim, porque os consumidores confiam

mais em seus amigos do que em um comercial da TV ou um spam. Eles são mais propensos a comprar algo sobre o que ouviram falar de seus colegas de trabalho do que algo que viram em um anúncio no meio do jornal de domingo.

Em seu livro *Purple Cow* [4] o mestre em marketing Seth Godin faz a óbvia análise de que a melhor maneira de fazer um consumidor notar um produto é fazê-lo marcante. Godin vai mais longe e afirma que os quatro P são obsoletos, e que os consumidores se tornaram entorpecidos por essa estratégia de marketing em massa. A única maneira de se destacar em uma multidão, na verdade, é ser brilhante.

Então, aqui é onde os leitores cínicos começam a aplaudir. Todo o falatório de marketing que você pode fazer não é nada comparado ao poder de um conjunto de capacidades marcantes. Antes de você começar a dizer “Eu te disse”, nós provavelmente devíamos conversar sobre a definição de **marcante**.

Marcante definitivamente não significa o mesmo que “bom”. Geralmente, produtos que são marcantes **são** bons. Mas produtos que são bons são raramente marcantes. Ser marcante significa que algo merece sua atenção. Você não vai se tornar um marcante desenvolvedor de software simplesmente por ser melhor do que outros programadores que você conhece.

Ser melhor do que alguém não é algo suficiente para resultar no aumento da sua reputação. Se alguém perguntar, pode ser que você tenha um conjunto de indicações brilhante, mas ser marcante significaria que as pessoas conversaram sobre você **antes** que lhes perguntassem.

Para ser marcante, você deve ser significativamente diferente daqueles ao seu redor. Muitas das estratégias de marketing pessoal discutidas nesse capítulo são montadas focadas na **notoriedade**.

Lançar softwares open source bem sucedidos, escrever livros e artigos, e dar palestras em conferências podem contribuir para aumentar sua **notoriedade**.

*Faça demonstrações ou morra!*

—

Se você reler o último parágrafo, embora não seja uma lista exaustiva, você notará que cada um dos itens que eu incluí como sendo potencialmente marcante envolve **fazer algo**. Você pode ser o mais esperto ou o mais rápido, mas apenas **ser** não é bom o suficiente. Você precisa **estar fazendo**.

Godin usa a expressão **vaca roxa** (*purple cow*) para nos lembrar do que é necessário para ser marcante. Não é a **melhor vaca**, ou a **vaca mais leiteira**, ou a **vaca**

**mais bonita.** Uma **vaca roxa** iria se destacar em um meio de vacas melhores, mais leiteiras e mais bonitas. Seria sobre a vaca roxa que vocêalaria depois de ver esse grupo.

O que você poderia fazer que tornaria a si mesmo e suas realizações como a vaca roxa? Não apenas seja mestre em um assunto — escreva um livro sobre isso. Escreva geradores de código que tornem o que costumava ser um processo de uma semana para um de cinco minutos. Em vez de ser respeitado entre seus colegas de trabalho, torne-se a autoridade mais reconhecida de sua cidade no que diz respeito à tecnologia na qual você se foca. Faça algo previamente impensável em seu próximo projeto.

Não deixe que você seja **apenas** o melhor em um grupo. Seja o cara e faça coisas sobre as quais as pessoas **não** podem deixar de falar.

## Faça algo

- 1) Comece pequeno, mas tente fazer **algo** marcante em seu atual projeto ou trabalho. Uma das maneiras de experimentar isso é tentar alcançar uma produtividade marcante. Agendas de projeto normalmente são sobrecarregadas. Encontre algo que todos pensam que precisará de uma semana para ser feito e faça em um dia. Trabalhe horas extras para isso se você achar necessário. Você não precisa fazer disso um hábito, mas sim, levar isso como um experimento. Faça o trabalho em um período de tempo marcante. Veja se as pessoas “notam”. Se não, por que não? Se sim, de que forma? Afine as variáveis e tente de novo.



## CAPÍTULO 43

# Fazendo o gancho

Quando eu era um jovem saxofonista no Arkansas, as pessoas me perguntavam “Oh, você conhece o Chris?” — eu não o conhecia. Aparentemente, ele era o **outro** adolescente na cidade que era um sério aspirante a músico de jazz. Então, quando as pessoas me viam, eles faziam a óbvia conexão, esperando que fôssemos camaradas em nossa caminhada jazzística colegial.

No verão, eu tive a oportunidade de ver a Count Basie Jazz Orchestra tocar em um show ao ar livre em um anfiteatro nas margens do rio Arkansas. Com uma combinação de bom humor e uma coragem descomunal, acabei conseguindo conversar com os músicos no backstage, antes do show. Eu nunca fui uma pessoa muito conversadora, então isso foi realmente uma vitória. Eu fiquei nos fundos conversando com um dos saxofonistas da orquestra, quando outra jovem criança chegou e começou a conversar. Depois de cinco ou dez minutos, a banda começou, deixando-nos desacompanhados. “Você é o Chris/Chad?” — nós dissemos simultaneamente.

Nos anos que vieram, eu passei bastante tempo com Chris. Ele, eu logo aprendi, tinha uma estranha aptidão para se associar aos melhores músicos da cidade. Ele era

apenas um menino da escola. Porém, ele já fazia shows, como substituto do pianista de jazz mais respeitado do Little Rock. Chris era muito bom — especialmente para sua idade — mas ele não era **tão** bom.

Não me levou muito para entender o que estava acontecendo. Nós saíamos, várias vezes por semana, a clubes onde se tocava jazz. Sempre era, de alguma forma, uma experiência desconfortável para alguém introvertido como eu, mas, cronometricamente, quando a banda fazia um intervalo, Chris interrompia o assunto e me deixava para conversar com os músicos. Ele era como um robô. Eu preciso admitir, era um pouco enjoativo assisti-lo. Era tão previsível. Ele não estava incomodando os pobres músicos? Eles estavam fazendo um intervalo, pelo amor de Deus. Eles não queriam conversar com essa criança! Tendo sido deixado de lado, ou eu tinha que segui-lo ou ficar sentado sozinho esperando. Ocasionalmente, quando eu não tinha muita energia, escolhia o segundo. Entretanto, na maioria das vezes eu o seguia e tentava fingir que me encaixava.

Frequentemente, até demais para a minha surpresa, os músicos pareciam gostar de conversar com Chris — e mesmo comigo. Ele era muito intrometido e sempre ficava pedindo para se sentar com a banda, por mais inapropriado que me parecesse. Ele também pedia aos músicos por aulas, o que significa que ele ia a suas casas para ouvir música e conversar sobre jazz. Algumas vezes eu era arrastado junto, com a mesma sensação que eu tinha nos intervalos dos shows — que eu estava incomodando.

Mas obviamente era eu que estava confuso com o relacionamento que Chris estava desenvolvendo com aqueles músicos. Ele estava tornando o sonho real, tocando com músicos realmente bons. Foi ele quem me conduziu aos melhores músicos da cidade. Sendo que a única diferença entre nós era que ele era extrovertido.

Com o passar dos anos, sua estratégia de “ser o pior”, aliada à habilidade de descaradamente se forçar para os outros, fez com que ele se tornasse um incrível pianista. De fato, ele achou sua fresta para tocar com músicos de jazz realmente famosos. Eu, por outro lado, ainda era o mesmo cara que ele conheceu. Ele me empurrou para alguns desses shows de perfil “mais elevado”, mas era sempre ele que dava o empurrão — nunca o inverso.

Desde então, tenho visto o mesmo padrão nas pessoas que conheci em música clássica, na Comunidade Americana de Budismo Tibetano, desenvolvimento de software, e mesmo nos confins de um simples escritório. Chris chamava isso de “fazer o gancho”, o que o tornava ainda mais repulsivo para mim. Mas o resumo da história é este: as pessoas realmente boas não se incomodam se você quiser conhecê-las.



As pessoas gostam de ser apreciadas e gostam de conversar sobre os assuntos pelos quais são apaixonados. O fato de elas serem o profissional, ou o guru, ou o líder, ou o autor renomado não muda o fato de que são humanos e gostam de interagir com outros humanos.

*O medo nos separa dos profissionais.*

—

Falando por mim mesmo (e extrapolando a partir daí), a barreira mais séria entre nós, mortais, e as pessoas que admiramos é nosso próprio medo. Associar-se a pessoas espertas e bem conectadas, que podem ensinar-lhe coisas ou achar um trabalho para você, é possivelmente a melhor forma de se melhorar, mas muito de nós têm medo de tentar. Fazer parte de uma comunidade profissional coesa e bem integrada é como músicos, artistas e outras pessoas da arte se tornaram fortes e evoluíram por anos. Os gurus são os pontos de partida nas redes sociais e profissionais. Tudo o que é preciso para estabelecer a conexão é um pouco menos de humildade.

É claro que você não precisa começar a conversar aleatoriamente. Obviamente você quer procurar aqueles com quem você tem algo em comum. Talvez você leia um artigo que o influenciou bastante. Você pode mostrar ao autor o trabalho que você fez como resultado, e pedir feedback. Ou talvez você possa criar uma interface de software para um sistema que alguém criou. Este é um ótimo e legítimo jeito de estabelecer a conexão.

Obviamente, você pode fazer isso tanto on-line como pessoalmente. Uma conexão duradoura é uma conexão duradoura. Os heróis do mundo do software estão espalhados pelo mundo. O mesmo vale para a indústria musical, embora você não possa confiar que todos os músicos sejam conectáveis por e-mail. Então, o mundo da música tende a formar aglomerados profissionais locais, enquanto os desenvolvedores de software têm a vantagem de serem facilmente alcançáveis, onde quer que você esteja. Isso torna fácil não apenas encontrar os gurus de sua cidade, como também qualquer guru. Algumas das mentes mais influentes do desenvolvimento de software estão prontamente acessíveis por e-mail e, até mesmo, por chat em tempo real.

A história que me fez escrever **este livro**, na verdade, começou com um e-mail sobre uma biblioteca Ruby para um de seus editores, seguido de muitas conversas via chat on-line. Embora eu estava tímido ao mandar o e-mail inicial, aparentemente isso não irritou Dave, e aqui está você, lendo minhas palavras. Obrigado, Chris.

## Não podemos simplesmente...?

*por Stephen Akers*

Qualquer pessoa que já passou bastante tempo em seu trabalho sabe que existe uma luta entre tecnologia de informação (TI) e o negócio (aqueles que não são TI). A raiz deste conflito é quase sempre um mal-entendido, falha de comunicação e expectativas mal gerenciadas. Esses assuntos são sublinhados quase diariamente por várias frases usadas por ambos os lados.

Para TI, dentre as expressões mais odiadas estão: “Não podemos simplesmente...?” Geralmente ocorre algo como: Não podemos simplesmente terceirizar este serviço? Não podemos simplesmente colocar mais desenvolvedores no projeto? Não podemos simplesmente fazer o que fizemos da última vez? Não podemos simplesmente tornar a aplicação mais rápida? Não podemos simplesmente criar um novo banco de dados?

O problema é que muitas das pessoas de TI, quando ouvem essa frase, se focam na palavra “simplesmente”. Isso os faz sentir como se as pessoas de negócio considerassem suas sugestões como óbvias, triviais, e facilmente alcançáveis. Qualquer falha na implementação iria, por conseguinte, indicar que aqueles de TI foram incapazes de completar a mais simples das tarefas e deviam ser substituídos.

Como resultado, uma resposta comum a esses pedidos é frequentemente “não”. As pessoas de TI querem se assegurar de que as de negócio percebam que suas propostas não são apenas complexas e difíceis de executar, como são, de fato, uma ideia ruim. Eis o conflito. Em última análise, o que acontece é que as pessoas de negócio se distanciam, com a impressão de que TI sempre diz “não”, enquanto as de TI ficam com a ideia de que as pessoas de negócio não sabem o que está sendo feito.

Isso é exatamente o que eu costumava pensar. Na minha opinião, o que o negócio realmente precisava era de alguém em sua equipe que sabia o que eles estavam fazendo. Este é o motivo pelo qual, em algum momento de minha carreira, eu resolvi deixar TI e partir para os negócios. Eu realmente esperava que todos os meus projetos tivessem grande sucesso porque eu sabia como fazer as coisas do jeito certo.

É engraçado como nossos planos nem sempre resultam naquilo que esperávamos. Embora eu tenha alcançado sucesso na equipe de negócios, não era assim que eu queria marcar este ponto.

Ao contrário disso, eu descobri que ainda havia muito o que aprender. Por exemplo:

- 1) Existem fatores comerciais reais que agem como limitadores em praticamente

todos os projetos. Essas limitações vão, às vezes, requerer que você implemente não menos que a solução técnica perfeita.

- 2) *Timelines* propostas pelas pessoas de negócio geralmente não são tão arbitrárias quanto parecem. Em muitas vezes, o prazo de entrega de uma solução pode ter um efeito cascata no projeto ou mesmo na performance da empresa.

Uma vez que eu aprendi essas lições, eu percebi que as pessoas de TI têm focado na parte errada da expressão “Não podemos simplesmente...” A palavra implícita operante nessa sentença, na verdade, é o sujeito **nós**. Esta palavra implica que o negócio está considerando a área de TI como uma parte crítica de sua equipe. Estão procurando o setor de TI por ajuda para formular uma solução que resultaria em sucesso para toda a empresa.

Portanto, a próxima vez que você ouvir essa frase, tente combater a resposta automática “não”. Mantenha o foco na palavra “nós” e diga, com confiança, “sim, **nós** podemos colocar mais programadores no projeto, mas isso não seria uma boa ideia, porque...” Mas não pare por aí. Não é suficiente apenas explicar seu posicionamento. É preciso mergulhar mais fundo para descobrir sob quais limites comerciais o negócio está operando. Com o tempo, isso lhe dará conhecimento sobre o domínio de negócio e oferecerá um melhor ponto de vista sobre os problemas que precisam ser resolvidos. A combinação disso com seu conhecimento técnico o fará progredir de alguém que sempre diz “não” para um parceiro sem o qual o negócio não poderia sobreviver.

*Stephen Akers é vice-presidente de TI na Genscape, Inc. [/box]*

## Faça algo

- 1) Escolha um de seus softwares favoritos e mande um e-mail para seu criador. Comece agradecendo-o pelo software. Então, faça uma sugestão, uma pergunta, ou faça outra tentativa de estabelecer um vínculo humano com ele. Solicite resposta. Se o software é gratuito ou open source, ofereça-se para ajudar de alguma forma.
- 2) Pense em algum local para você, quem você admire ou de quem gostaria de aprender algo. Procure uma situação em que você possa encontrar essa pessoa (uma reunião de um grupo de usuários ou palestras são ótimas oportunidades), e dê um jeito de começar uma conversa, mesmo se isso o deixar desconfortável — **especialmente** se você estiver desconfortável.



## CAPÍTULO 44

# Já obsoleto

Muitos de nós somos guiados pela a indústria de TI porque as coisas estão sempre mudando. É um ambiente de trabalho fresco e excitante. Sempre há algo novo a aprender. Por outro lado, contudo, está o desanimador fato de que nossos investimentos duramente conquistados no meio tecnológico depreciam-se mais rápido do que um carro novo. A novidade de hoje é a tranqueira obsoleta de amanhã com vida útil limitada.

*Suas brilhantes novas habilidades já são obsoletas.*

—

Em *Leading the Revolution* [5], Gary Hamel fala sobre como os líderes incumbentes de qualquer indústria se tornam complacentes e, com isso, desenvolvem pontos cegos. Quanto mais bem sucedido for seu negócio, é mais provável que você fique confortável com seu modelo de negócio, tornando-se incrivelmente vulnerável para aqueles que vêm atrás de você com uma nova ideia radical — mesmo que seja estúpida — que pode fazer com que seu maravilhoso e vitorioso modelo de negócio

pareça um suéter desgastado e velho largado em uma discoteca. O mesmo pode ser dito sobre escolhas de tecnologia. Se você domina a grande tecnologia do momento, como Java EE ou .NET, até o momento de publicação deste livro, talvez você esteja se sentindo extremamente confortável. É um lugar lucrativo, certo? Cada site de vagas de emprego, ou cada seção de classificados no jornal, serve como uma afirmação em sua decisão.

Cuidado. Sucesso gera arrogância, que gera complacência. Uma onda como J2EE pode dar a impressão de que nunca vai acabar. Contudo, todas as ondas ou se dissipam, ou encontram a costa eventualmente. Muito conforto por muito tempo pode deixá-lo indefeso, imaginando o que você faria em um mundo sem J2EE.

Dito isso, as pessoas têm anunciado a morte de COBOL por décadas. Cada novo incumbente é chamado de “o COBOL do século 21”, ou alguma variação disso. Nos dias de hoje, esse rótulo é aplicado a Java. Embora eu deteste encostar, ver, ou estar perto de código COBOL, chamar Java de COBOL do século 21 é um grande elogio. Mesmo que alguns de nós adoraríamos vê-lo sumir, COBOL está aqui, e tem operado por um **longo** tempo. Programadores COBOL têm trabalhado com essa linguagem durante suas carreiras inteiras. Isso realmente diz algo na montanha russa da indústria. É difícil dizer se o mesmo tipo de investimento funcionaria na economia de hoje.

A história do COBOL é uma exceção — não a regra. Poucas tecnologias fornecem plataforma para emprego tão duradoura. A mensagem aqui não é que você viva apenas com seu conhecimento em uma tecnologia mainstream. Isso seria irresponsável. Eu **vou dizer** que quanto mais mainstream for seu conhecimento, maior risco você terá de ser deixado na tecnologia da idade da pedra.

Todos nós já ouvimos as extrapolações da lei de Moore, que diz que o poder de processamento duplica a cada 18 meses. Se os números estão corretos ou não, é fácil perceber que a tecnologia ainda está avançando com a mesma proporção do que fazia em 1965, quando Gordon Moore, da Intel, fez essa asserção. E com esses avanços no poder do hardware vêm os possíveis progressos do que fazer com softwares.

O poder de processamento **duplica**. Com a tecnologia progredindo tão rapidamente, há muita coisa acontecendo para qualquer pessoa acompanhar. Mesmo que suas habilidades sejam completamente atuais, se você não está praticamente no processo de aprender a Próxima Grande Coisa, é quase tarde demais. Você pode estar à frente da curva na onda atual e atrás da próxima. *Timing* se torna muito importante em um ambiente como este.

Você deve começar a perceber que, mesmo se você está na crista da onda atual, você provavelmente já está atrás da próxima. Se timing é tudo, comece a pensar à frente com seus estudos. O que vai ser possível em dois anos, que ainda não é agora? E se um espaço de disco fosse tão barato que fosse praticamente gratuito? E se os processadores fossem duas vezes mais rápidos? O que não precisaríamos nos preocupar em otimizar? Como esses avanços mudam o que vai estourar no futuro?

Sim, é uma aposta. Porém, é um jogo que você vai **definitivamente** perder se não jogar. O pior dos casos é se você aprendeu algo enriquecedor que não é diretamente aplicável em seu emprego em dois anos. Então ainda é melhor se você estiver olhando adiante e apostando nisso. O melhor caso é se você se mantiver à frente da curva e puder continuar a ser um expert em tecnologias de ponta.

Olhar adiante e ser explícito sobre o desenvolvimento de suas habilidades pode fazer a diferença entre ser cego ou visionário.

## Faça algo

- 1) Separe um momento semanal para investigar o que virá como vanguarda. Encontre pelo menos duas horas por semana para pesquisar tecnologias e começar a desenvolver suas habilidades nelas. Ponha a mão na massa com essas novas tecnologias. Construa aplicações simples. Faça protótipos com a nova tecnologia nos trechos difíceis de seus projetos que usam a tecnologia atual para entender quais são as diferenças e o que as novas tecnologias permitem.

Coloque isso na sua agenda. Não deixe de fazer.





## CAPÍTULO 45

# Você já perdeu seu emprego

O emprego para o qual você foi contratado já não existe mais. Você pode ainda estar pensando em um salário. Você pode ainda estar agregando valor. Você pode ainda estar deixando seu chefe extasiado e feliz. Mas você já perdeu seu emprego.

A coisa mais certa é que tudo está mudando. A economia está em constante movimento. Empregos se tornam *offshore* ou *onshore*. Os negócios estão tentando se adaptar. As coisas não alcançaram um ponto estável em nossa indústria. Ela está como um adolescente estranho passando pela puberdade. Estranho, feio, e diferente ano após ano — dia após dia.

Portanto, se você foi contratado para ser um programador, não pense em si mesmo como um programador. Pense em si mesmo como talvez não mais um programador. Continue executando suas funções, mas não fique muito confortável. Não tente se acomodar com a **identidade** de um programador. Ou um designer. Ou um testador.

De fato, não é mais seguro (como se antes fosse) identificar-se muito proximamente com o emprego para o qual foi contratado. Se as coisas ao seu redor estão

mudando, e o contexto no qual trabalha está em constante movimento, agarrar-se à sua função cria uma dissonância não saudável que contamina seu trabalho. Você pode se pegar querendo ser um programador, mas fazendo o trabalho de um gerente de projetos. E fazendo isso mal.

*Você não é seu emprego.*

—

De volta a antes de perder seu emprego, você pode ter tido planos. Você pode ter imaginado seu progresso através dos ranks de sua empresa. Você cumpria suas horas como um designer e assumia o papel de arquiteto quando a recompensa era devida. Você podia ver a progressão inteira de um arquiteto até um analista, até líder de equipe, através da cadeia de gestão.

Porém, você já perdeu seu emprego, e seus planos mudaram. Eles vão continuar mudando. A cada dia. É bom ter ambições, mas não se dedique cegamente a um futuro longo e imaginário. Você não pode se dar ao luxo de ter uma visão de túnel, onde só se vê algo tão distante no futuro. Se você busca atingir algum alvo em movimento, você não mira propriamente no alvo. Você mira no lugar ao qual o alvo deve ir. O caminho daqui até lá não é mais uma linha reta. É um arco, na melhor das hipóteses, mas é mais provável que seja um rabisco amórfico.

## **Faça algo**

- 1) Se você é um programador, tente por um dia ou dois fazer seu trabalho como se você fosse um testador ou um gerente de projeto. Quais são os vários papéis que você poderia ter dia após dia, que você nunca explicitamente considerou? Faça uma lista e tente experimentá-los. Passe um dia em cada. Talvez você nem mude os resultados do seu projeto, mas você verá seu trabalho de um jeito diferente.

## CAPÍTULO 46

# Caminhe sem destino

Um dos maiores problemas da América é que ela é uma sociedade guiada por metas. Somos uma nação de pessoas que sempre estão focadas no **resultado** de um processo, seja o processo de aprendizado, a carreira de alguém, ou mesmo dirigindo com seu carro. Somos tão focados no resultado que esquecemos de olhar para o cenário como um todo.

Se você pensar sobre isso, o foco no resultado é logicamente o inverso daquilo com que nós deveríamos estar perdendo nosso tempo. Você tipicamente passa todo seu tempo **fazendo** coisas, e pouco tempo realmente atingindo metas. Por exemplo, quando você está desenvolvendo software, o processo durante o desenvolvimento é onde você passa todo seu tempo, e não no final do projeto, quando tudo está pronto.

Isso também é válido para sua carreira. O alimento real de sua carreira não são promoções e aumentos de salário. É o tempo que você gasta trabalhando em direção a esses avanços. Ou, mais importante, é o tempo que você passa trabalhando **independentemente** dos avanços.

Se esse é o núcleo da sua vida profissional — o verdadeiro trabalho — então você

já chegou ao seu destino. O seu pensamento focado no destino e focado nos objetivos apenas o guia de uma meta à outra. Não existe um fim lógico. O que muitos de nós não conseguem entender é que **o caminho** é o fim.

De volta ao exemplo do desenvolvimento de software, é fácil se envolver demais com a entrega do código que você está criando. Seu cliente precisa de uma aplicação web, e você se foca em terminar tal projeto. Mas uma aplicação viva nunca está “concluída”. Um lançamento conduz ao outro. Foco demais no produto final nos distrai da verdadeira entrega: o desenvolvimento sustentável de uma nova entidade.

*Concentre-se em fazer, não no que está sendo feito.*

—

Focar no final faz com que você esqueça de fazer o processo direito. E processos ruins criam produtos ruins. O produto deve ter seus requisitos mínimos, mas seu interior será feio. Você se otimizou para a meta a curto prazo — e não para o futuro inevitável e contínuo do desenvolvimento do produto.

Não apenas processos ruins fazem produtos ruins, mas produtos ruins fazem processos ruins. Uma vez que você tem um desses produtos que são bagunçados em seu interior, seus processos se adaptam a isso. As **janelas quebradas** de seu produto levam a janelas quebradas em seu processo. É um ciclo vicioso.

Portanto, em vez de constantemente se perguntar “Já estamos lá? Já estamos lá?” perceba que a única resposta saudável é “sim”. É como você cruza o caminho que é importante, e não o destino.

## Faça algo

- 1) Em seu livro *The Miracle of Mindfulness* [6], Thich Naht Hanh apresenta uma sugestão: a próxima vez que você tiver que lavar a louça, não o faça para tê-las limpas. Tente apreciar a experiência de lavar a louça. Não se concentre em terminar. Foque-se no ato de lavá-las em si.

Lavar a louça é uma tarefa mundana que quase ninguém saboreia. Desenvolvedores de software fazem algo semelhante em um dia normal, como lidar com a pressão do tempo e relatórios de gastos, por exemplo. A próxima vez que você tiver que fazer uma tarefa como essa, veja se há um jeito de focar na tarefa enquanto você a realiza, em vez de ansiosamente se apressar para concluí-la.

## CAPÍTULO 47

# Trace um roteiro

Quando você está em “modo manutenção”, é fácil pegar um ritmo e simplesmente continuar a ser como você é. Como um desenvolvedor de software, você sabe que isso acontece pela sua experiência com sistemas. Se você mantém uma aplicação ou uma biblioteca que outros desenvolvedores usam, ela vai estagnar no modo de correção de bug (ou pior) ao menos que você tenha um sólido roteiro. Você pode fazer as melhorias ocasionais devido a pedidos dos usuários ou por necessidade própria, mas o código eventualmente vai atingir um estado estável e mudará em uma taxa exponencialmente mais lenta, porque você o considera pronto.

Contudo, uma aplicação viva nunca está concluída, ao menos que esteja no caminho para a aposentadoria. O mesmo é válido para sua carreira. A não ser que você esteja procurando sair do meio, você precisa de um roteiro. Se a Microsoft tivesse considerado o Windows 3.1 pronto, todos nós estaríamos usando Macs agora. Se os desenvolvedores Apache tivessem considerado seu servidor pronto quando fizeram o 1.0, eles não estariam esmagadoramente liderando o mercado agora.

O seu roteiro pessoal para seu produto é o que você usa para dizer se você pro-

grediu ou não. Quando você vai para o mesmo escritório dia após dia, para trabalhar em muitas das mesmas coisas, o cenário ao seu redor não muda. Você precisa colocar indicadores que você possa ver a distância, de modo que possa saber se realmente saiu do lugar. As funcionalidades do seu produto são esses indicadores.

Ao menos que você realmente pegue isso e faça um plano, você não conseguirá ver além do próximo pontinho no horizonte. Nos capítulos 2 e 3, você descobriu como escolher o caminho de sua carreira e como investir em sua vida profissional. Embora eu tenha focado no que parecia ser uma escolha de uma única vez sobre aquilo em que investir, cada escolha deveria ser parte de algo maior. Pensar em cada novo conhecimento ou habilidade como equivalentes a uma simples característica de uma aplicação traz muito bem esse conceito. Uma aplicação com uma característica não é muito uma aplicação.

Além do mais, uma aplicação com um grupo de características que não sejam coesas vai confundir seus usuário. **Seria isso um caderno de endereços ou uma aplicação de chat? É um jogo ou um browser?** Um roteiro pessoal para seu produto pode não apenas ajudá-lo a seguir seu caminho, constantemente evoluindo, mas também lhe dar uma visão geral do que você tem a oferecer. Ele pode mostrar que nenhuma característica única se mantém. Cada novo investimento é parte de um todo maior. Algumas funcionam fabulosamente bem juntas, outras requerem um grande salto mental para os potenciais empregadores. **Seria ele um administrador de sistema ou um designer gráfico? Seria ela uma arquiteta de aplicação ou um guru de automação de QA?**

Embora seja definitivamente ok aprender diversas coisas — pois isso expande seu pensamento — também é uma boa ideia pensar sobre a história que seu conjunto de habilidades conta. Sem um roteiro, sua história pode parecer mais com um romance de Jack Kerouac do que com um coeso grupo de competências logicamente relacionadas. Sem um roteiro, você pode **realmente** até se perder.

## Faça algo

- 1) Antes de colocar no mapa o lugar para o qual você quer ir, ele pode ser um roteiro informativo e encorajador sobre onde você **esteve**. Pegue um tempo para explicitamente escrever a linha do tempo de sua carreira. Mostre onde você começou e quais eram suas habilidades e empregos em cada fase. Note os pontos onde você fez melhorias incrementais e onde você pareceu realizar grandes saltos. Observe a duração média necessária para fazer um avanço considerável. Você pode esta-

belecer metas mais objetivas para si se tiver uma imagem clara do histórico de se progresso. Uma vez que você criou esse mapa histórico, mantenha-o atualizado. É uma ótima maneira de refletir seu progresso conforme você move em direção ao seus objetivos recém-estabelecidos.





## CAPÍTULO 48

# Observe o mercado

Você seria tolo se investisse seu dinheiro em uma ação volátil e depois ignorá-la. Mesmo se você fez muita pesquisa e fez uma escolha proposital sobre **em quem** investir, o mercado é incerto. Você não pode simplesmente atirar e se esquecer, quando se trata de investimentos. Mesmo se o valor de uma ação está aumentando agora, isso não significa que ele não vai começar a afundar amanhã.

Além disso, você pode estar perdendo uma oportunidade. Talvez você pense que seja um porto seguro estar rendendo um retorno de 10% no ano. Isso parece realmente um bom negócio, se o resto do mercado não está de repente fazendo muito melhor que 10%. Seu principal investimento de hoje, mesmo se continuar a agir, pode não ser muito impressionante se comparado com o que será possível amanhã.

Na medida em que as condições do mercado mudam, não prestar atenção pode resultar em perda de dinheiro ou perda de dinheiro que se **poderia** ganhar.

O mesmo é válido para seu investimento em conhecimento. Java é a escolha conservadora dos dias de hoje. O que poderia mudar, que poria em questão tal verdade? Como você poderia saber se isso mudou?

E se, por exemplo, a Oracle começasse a dar sinais de falência? No passado, a Sun Microsystems, que mantinha o Java, perdeu sua posição dominante, e Java não era um padrão aberto. Embora hoje seja open source, é ditado e desenvolvido pela Oracle. Em qualquer momento, uma moribunda Oracle pode tentar subitamente fazer de sua linguagem e de sua máquina virtual um centro lucrativo. Isso pode fragmentar a linguagem Java com mudanças incompatíveis, gerando um pânico enorme.

Com sua cabeça focada no código do seu monitor, talvez você nem ouça falar de algo como isso antes de ser tarde demais. Você pode se ver no mercado com uma habilidade com valor subitamente diminuto. Essa é uma situação hipotética improvável, mas algo parecido pode acontecer.

O mais provável é que, você estando confortável em seu emprego atual, com suas competências atuais, você pode se tornar um ignorante feliz conforme a Nova Grande Coisa desponta. Vinte anos atrás, você teria ficado surpreso ao descobrir o quão grandes linguagens orientadas a objeto com *Gargabe Collection* se tornariam. Porém, definitivamente havia sinais, se você os estivesse observando. Daqui a 10 anos, quem sabe qual será a nova grande revolução?

Você deve manter seus olhos e ouvidos abertos. Observe as novidades tecnológicas, tanto pelo viés empresarial como pelo lado técnico puro, à procura de descobertas que podem provocar uma onda. Como Tim O'Reilly (<http://tim.oreilly.com/>), da O'Reilly and Associates, diz, observe os **geeks alfa**. Os geeks alfa são aqueles supernerds que estão sempre no mais alto topo do mais alto cume, ao menos em seus hobbies. A afirmação de Tim, que eu tenho desde então observado, é que se você consegue achar essas pessoas e ver em que eles estão mergulhados, você pode ter um vislumbre do que potencialmente pode ser grande em um ou dois anos. É estranho como isso funciona bem.

*Observe os geeks alfa.*

—

Independentemente da escolha que você fizer, é preciso estar atento ao fato de que, no setor de tecnologia, o que é um bom investimento hoje eventualmente **não** será mais um bom investimento. E, se você prestar atenção ao estado de espírito do mercado, isso raramente vai pegá-lo de surpresa. Você não quer esse tipo de surpresa.

## Faça algo

- 1) Passe o próximo ano tentando se tornar um dos geeks alfa. Ou, ao menos, **faça o gancho** com um.

## CAPÍTULO 49

# Aquele gordo no espelho

Infelizmente, estou acima do peso. Já faz tempo. Porém, quando eu morava na Índia, eu perdi **muito** peso. Parte disso foi por causa da dieta. Parte, pelos exercícios. Mas a maior parte foi por ficar doente. Depois que voltei aos Estados Unidos, eu lentamente ganhei peso de novo. Era uma coisa desapontadora, à qual reagi me matriculando em uma academia e contratando um personal trainer. O peso voltou a diminuir.

Passei por diversas dessas oscilações. O que é fascinante disso tudo é que eu não consigo realmente notar quando estou ganhando ou perdendo peso. A única forma de eu saber é se alguém me disser ou se minhas roupas de repente pararem de caber. Minha esposa me vê todo dia, de modo que ela também não consegue notar — e, nos Estados Unidos, as pessoas geralmente não falam nada se você **engordar**. Na Índia, sim.

Eu não reparo porque me vejo muito frequentemente. Se você é constantemente exposto a algo, é difícil notar que ele está mudando, ao menos que a mudança aconteça rapidamente. Se você sentar e observar uma flor desabrochar, vai passar bastante tempo até que você note que algo aconteceu. Entretanto, se você sair e voltar em dois

dias, verá algo muito notável diferente de quando você saiu.

Você perceberá o mesmo fenômeno com sua carreira. Na verdade, você **não** notará. Este é o problema. Você pode olhar para si mesmo pelo espelho metafórico todo dia e não ver nenhum traço de mudança. Você parece tão bem ajustado quanto antes. Você parece tão competitivo quanto antes. Suas habilidades parecem ser tão atualizadas quanto antes.

Então, de repente, um dia seu emprego (ou seu negócio) não se encaixa mais com você. Parece apenas desconfortável no início, mas você já alcançou um ponto crítico, no qual você precisa ou agir rapidamente ou comprar um novo par de calças (metafóricas).

Quando se trata de flutuações de peso corporal, você tem uma balança, então é razoavelmente fácil mensurar seu progresso (ou falta dele, no meu caso). Infelizmente, não há balança para medir seu poder de marketing ou suas habilidades de programador. Se houvesse, poderíamos colocá-la em uma tabela e autogerar seus salários. Já que isso não existe, você terá que criar uma para si mesmo.

Um jeito fácil de medir seu progresso é usar pessoas confiáveis. Um mentor ou um colega próximo podem ajudar a dar um olhar mais objetivo sobre onde você está. Você pode discutir suas habilidades de desenvolvedor de software, líder de projeto, comunicador, membro de equipe, ou qualquer outra faceta do pacote total que o faz quem você é. Na GE, esse é um processo chamado de “revisão 360”, que formaliza a ideia e encoraja os funcionários a procurar por feedback de seus parceiros, seus chefes e clientes internos. Esse processo é uma ótima forma de obter um número de diferentes perspectivas de si mesmo como um funcionário.

*Desenvolvedor, reveja-te a ti mesmo.*

—

A coisa mais importante para se trazer à tona, conforme você passa por um processo como este (seja sozinho ou com ajuda), é onde estão seus pontos cegos. Você não **precisa** consertar todos eles. Você apenas deve saber onde eles estão. Sem ser específico, você será cego aos seus pontos cegos. É aí que as coisas ruins acontecem e o pegam desprevenido. Coisas ruins acontecem, então é melhor saber que elas estão vindo.

Mesmo se você tivesse uma **balança mágica de valores** na qual você pudesse se medir, ela não lhe faria bem a não ser que você realmente a usasse. Agende suas revisões. Você não vai parar para refletir, ao menos que você torne o tempo de reflexão

explícito. Dizer “Não esquecer de pedir por feedback” não é uma mensagem suficientemente forte. Se você possui um calendário que tenha lembretes em pop-up, marque compromissos consigo mesmo para autoavaliação. Primeiro, determine seu sistema de medida, e então, coloque-o na agenda. Se não for uma parte intrínseca de sua vida profissional, você não o fará.

Se sua empresa já realiza tais processos, não os trate como algo nonsense de RH. Leve-os a sério, e **faça** algo de bom sair disso. Eles podem ter sido implementados de um jeito pobre no seu trabalho, mas a motivação (pelo menos o que **costumava ser** motivação) para eles ainda continua de pé.

Finalmente, quando você tiver seu próprio sistema, e tiver agendado tempo para aplicá-lo, **guarde os resultados por escrito**. Mantenha sua avaliação em algum lugar próximo. Reveja-o e revise-o com frequência. Ligar o processo de autoavaliação a um registro físico o tornará concreto.

Não deixe que coisas antigas o prendam como um par de calças apertadas.

## Aja nisso!

### 1) Faça uma revisão 360.

- Faça uma lista de pessoas confiáveis, com as quais se sente confortável em pedir feedback. Essa lista deve, preferencialmente, conter representantes de seus colegas, clientes e chefes (e subordinados, se você tiver algum).
- Faça outra lista de aproximadamente 10 características que você acredita que sejam medidas importantes suas como um profissional.
- Converta essa lista em um questionário. Nele, peça aos participantes para lhe darem nota em cada aspecto. Inclua também a questão “O que eu devia ter perguntado?”
- Distribua o questionário para uma lista de pessoas de sua confiança. Peça aos seus avaliadores que façam críticas construtivas. O que você precisa é de feedback honesto — e não puxa-saquismo.

### 2) Depois de receber as respostas, leia-as e compile uma lista de ações que você tomará como resultado. Se você fez as perguntas certas para as pessoas certas, você **terá** alguns itens acionáveis. Compartilhe o resultado de seu questionário com seus avaliadores — não as respostas, mas as mudanças resultantes que você planeja fazer. Lembre-se de agradecer. Repita o processo ocasionalmente.

- 3) Comece a manter um diário. Pode ser um blog, como discutimos em [39](#), ou um diário pessoal. Escreva sobre o que você está fazendo, o que está aprendendo e suas opiniões sobre o negócio.

Depois que você estiver mantendo o diário por algum tempo, releia registros antigos. Você ainda concorda? Eles parecem errados? O quanto você mudou?

## CAPÍTULO 50

# A armadilha de macaco da Índia do Sul

Em *Zen and the Art of Motorcycle Maintenance* [8], Robert Pirsig conta uma história sobre como as pessoas no sul da Índia costumavam capturar macacos. Eu não sei se isso é verdade, mas nos ensina uma lição útil, portanto vou parafraseá-lo.

As pessoas na Índia do Sul, tendo sido importunadas por macacos por vários anos, desenvolveram um engenhoso jeito de capturá-los. Eles cavavam um buraco longo e estreito no chão, e usavam um objeto igualmente longo para alargar o fundo do buraco. Então, jogavam arroz na parte mais larga do fundo.

Macacos gostam de comer. De fato, essa é uma grande parte do que os torna tais pestes. Eles pulam nos carros ou arriscam correr no meio de multidões para roubar comida de suas mãos. As pessoas da Índia do Sul estão cientes disso. (Acredite-me, é surpreendentemente perturbador estar sentado, tranquilo, e de repente aparecer um macaco safado para roubar algo seu.)

De acordo com Pirsig, os macacos apareciam, descobriam o arroz e esticavam

seus braços pelo buraco. Suas mãos alcançavam o fundo. Eles pegavam vorazmente o máximo de arroz possível, fechando as mãos em punho. Assim fechadas, elas cabiam na parte mais larga do buraco, mas o resto da abertura era estreita demais para que os macacos puxassem seus punhos cheios de arroz de volta. Eles ficavam presos.

É claro, eles podiam simplesmente largar a comida e sair livres.

Mas macacos dão grande valor à comida. Aliás, dão tanto valor que não conseguem se forçar a desistir. Eles vão apertar o arroz até que ele saia do chão ou morrerão tentando tirá-lo de lá. Este último era o que geralmente acontecia.

Pirsig conta essa história para ilustrar um conceito que ele chama de **rigidez de valor**. É o que acontece quando você acredita tão fortemente no valor de alguma coisa que você não consegue mais questioná-lo objetivamente. Os macacos valorizavam tanto o arroz que, quando forçados a escolher entre o arroz em cativeiro e a morte, não conseguiam ver que deixar a comida era a coisa certa a se fazer naquele momento. A história faz os macacos parecerem muito estúpidos, mas a maioria de nós tem nossos próprios equivalentes ao arroz.

Se lhe perguntassem se é uma boa ideia ajudar a alimentar crianças famintas em países em desenvolvimento, você provavelmente diria “sim”, sem pensar. Se alguém tentasse argumentar com você, você o acharia louco. **Este** é um exemplo de rigidez de valor. Você acredita nessa coisa tão fortemente que não consegue imaginar **não** acreditar nisso. Claramente, nem todos os valores que consideramos rígidos são ruins. Para a maior parte das pessoas, religião (ou a falta dela) também é um conjunto de crenças pessoais e de valores inabaláveis.

Porém, nem todos os valores rígidos são bons. Além disso, muitas vezes algo que é bom em uma determinada circunstância não é boa em outra.

*Valores rígidos o tornam frágil.*

—

Por exemplo, é fácil se desligar das escolhas tecnológicas. Especialmente quando a tecnologia que escolhemos é *underground*. Nós amamos tanto tecnologia e colocamos um valor tão alto quando a defendemos para ser adotada, que vemos cada oportunidade como uma batalha em que vale a pena lutar — mesmo quando estamos defendendo o que claramente é a escolha errada. Um exemplo que encontro (e pelo qual provavelmente me sinto culpado) é a base de fãs superzelosa de Linux. Muitos dos usuários Linux colocariam Linux no desktop de qualquer recepcionista, assistente, e vice-presidente da corporação sem levar em consideração que, em termos de usabilidade, as ferramentas simplesmente não são tão compatíveis com muitos



dos softwares comerciais disponíveis para um sistema operacional comercial. Você fica com cara de bobo e faz seus clientes infelizes quando você dá o software certo para as pessoas erradas.

É difícil perceber que você está emagrecendo porque você se vê todo dia. Rigidez de valor funciona do mesmo modo. Desde que vivemos todo dia na mesma carreira, é fácil desenvolver rigidez de valor em nossas escolhas de carreira. Sabemos o que funcionou, e continuamos fazendo isso. Ou, talvez você sempre quis ser promovido à gerência, então você continua se esforçando para este objetivo, independente de quanto você gosta de **apenas programar**.

Também é possível que a tecnologia de sua escolha se torne obsoleta, deixando-lhe subitamente sem uma base na qual se apoiar. Como um sapo em uma panela lentamente aquecendo, você pode de repente se ver em uma situação ruim. Muitos de nós, na metade dos anos 90, só pensávamos em Novell NetWare quando se tratava de disponibilizar arquivo e imprimir serviços na empresa através de uma rede. A Novell estava muito além de seu tempo com seu produto de serviços de diretório, e muitos de nós “do conhecimento” éramos quase arrogantes na crítica às tecnologias concorrentes. O produto da Novell estava desfrutando de uma saudável liderança no mercado, e era difícil imaginar a maré mudando.

Nenhum evento sozinho tornou óbvio que a Novell estava perdendo para a Microsoft. A Microsoft nunca fez o lançamento mágico de um Active Directory, que nos fizesse dizer “Uau, já era NetWare!” Mas a NetWare lentamente foi de inovação para tecnologia legada. Para muitos administradores da NetWare, a água estava fervendo antes de eles sequer perceberem que a panela estava morna.

Se é esse o caminho que sua carreira está tomando, ou as tecnologias que você defende e nas quais investe, tome cuidado com as armadilhas de macaco. Aquelas escolhas originalmente intencionais podem se tornar o último punhado de arroz que você está segurando antes de sua carreira apanhar até a morte.

## Faça algo

- 1) **Encontre suas armadilhas de macaco** — quais são **suas** presunções rígidas? Quais são os valores que guiam suas ações diárias sem você conscientemente saber?

Faça uma tabela com duas colunas: “Carreira” e “Tecnologia”. Embaixo de cada uma, liste os valores que você considera como verdades inabaláveis. Por exemplo, sob “Carreira”, o que você **sempre** considerou como sendo um de seus pontos

fortes? Ou fraquezas? Qual é o **objetivo** de sua carreira (“Eu quero ser CEO!”)? Quais são as maneiras certas de alcançar sua meta?

Na coluna “Tecnologia”, liste o que você mais valoriza sobre as tecnologias nas quais escolheu investir. Quais são os atributos mais importantes de uma tecnologia que deviam ser considerados ao fazer uma escolha? Como você faz um sistema de medida? Qual o ambiente mais produtivo para desenvolvimento de software? Quais são as melhores e piores plataformas, em geral?

Quando você tiver sua lista e você considerá-la razoavelmente completa, vá item por item e mentalmente inverta cada afirmação. E se o oposto de cada asserção fosse verdadeiro? Permita-se honestamente desafiar cada uma.

Essa é uma lista de suas vulnerabilidades.

- 2) **Conheça seu inimigo** — pegue a tecnologia que você mais odeia, e faça um projeto usando-a. Desenvolvedores tendem a separar-se entre os campos concorrentes. As pessoas de .NET odeiam J2EE, e as pessoas de J2EE odeiam .NET. Pessoas de UNIX odeiam Windows, e vice-versa.

Escolha um projeto fácil e tente fazer uma **ótima** aplicação usando a tecnologia que você odeia. Se você é uma pessoa de Java, mostre àqueles de .NET como um desenvolvedor **de verdade** usa a plataforma! Na melhor das hipóteses, você aprenderá que a tecnologia que você odeia não é tão ruim e que, aliás, é possível desenvolver bom código com ela. Você também terá uma (certamente subdesenvolvida) nova habilidade que você talvez precise usar no futuro. No pior dos casos, o exercício será uma sessão de prática para você, e você terá mais recursos para seus argumentos.

# Evite planejamento de carreira do modelo cascata

No começo deste milênio, formou-se uma rebelião inicialmente pequena na indústria de software. Um grupo de especialistas em desenvolvimento de software começou a perceber que, entre eles, estava se formando uma tendência tanto em projetos de software que estavam falindo ou tendo sucesso. Em um ambiente industrial, em que mais projetos estão falindo do que tendo êxito, eles acreditavam que tinham descoberto uma forma de fazer melhor. O grupo se autodenominou Agile Alliance.

A indústria, naquela época, se levou a acreditar que o único jeito de desenvolver projetos de software era seguir um processo rigoroso, cautelosamente planejado do topo ao fim. Analistas definiam requisitos em longos documentos, enquanto arquitetos enviavam suas criações para os designers, que criavam designs detalhados. O produto disso era passado a desenvolvedores que o transformavam em código, em alguma linguagem de programação. Finalmente, após meses — às vezes anos de esforço —, o código era integrado e entregue a um grupo de teste, que o certificaria

para implementação.

Às vezes, alguma variante desse processo funcionava. Se todo mundo soubesse cada detalhe necessário no começo de um projeto, esse tipo de planejamento e rigor podia vir a entregar um software bem pensando e com qualidade garantida. Mas na maior parte das vezes, as pessoas não sabem cada detalhe do que eles esperam de um grande projeto. Quanto maior e mais complexo for um projeto, é menos provável que seja possível imaginar cada aspecto com detalhes suficientes para se criar uma especificação. Esse tipo de processo é um **processo em cascata**, e essa expressão é equivalente a processos ruins praticamente de forma universal nos dias de hoje.

Então, como os membros da Agile Alliance perceberam, seguir um processo rígido como a maioria das organizações estava fazendo na época, resultava em software bem testado e minuciosamente documentado, o que não era o que os usuários queriam. A rebelião foi criar uma família de metodologias ágeis. Eram processos de desenvolvimento de software que caminhavam em direção à mudança fácil. Menos tempo era gasto fazendo o planejamento e o design. Software é maleável, então mudá-lo **pode** ser barato. As metodologias ágeis pressupunham as mudanças como uma parte constante do desenvolvimento, e adaptavam-se para torná-las o mais barato e fácil possível.

Tudo isso parece óbvio agora. Mas naquela época, a adoção de processos ágeis era um motivo de conflito e debate. Em tese, a ideia de planejar com detalhe e rigor parece obviamente correta. Mas na prática, isso não funciona.

No início da minha própria conversão às metodologias ágeis (especialmente *Extreme Programming*), passei a ver tudo através das lentes do desenvolvimento ágil. As forças e as motivações em jogo vieram a ser mais gerais do que para apenas desenvolvimento de software. Toda vez que eu tinha que resolver um problema complexo, eu percebia que uma abordagem iterativa e receptiva a mudanças era um jeito menos estressante e mais efetivo para mim.

De alguma forma, contudo, levou-me bastante tempo para perceber que o projeto mais complexo com que já tive de lidar — o mais estressante e mais crítico — era minha carreira. Eu tinha projetado minha carreira do começo ao fim como um projeto de software com modelo em cascata. E os mesmos problemas que ocorriam em projetos de software estavam começando a acontecer comigo e com minha carreira.

Eu estava na trilha para ser um vice-presidente bem sucedido, ou um diretor de informática. Estava indo muito bem nesse caminho. Eu havia progredido rapidamente de um programador novato para um arquiteto de software, até gerente e diretor, e podia facilmente me ver continuando na cadeia. Mas, por mais bem suce-

dido que eu fosse, comecei a sentir que estava fazendo um trabalho de que eu não gostava. De fato, quanto mais bem sucedido que eu era, era menos provável que eu estivesse em um emprego de que gostasse.

O que eu estava fazendo comigo era a mesma coisa que processos rígido fazia com seus clientes. Eu estava fazendo um trabalho **excelente** de entregar a mim mesmo uma carreira que **eu não queria**.

Não era intuitivo para mim no começo, mas a solução para tal problema era simplesmente **mudar** de carreira. Isso pode ter vários significados. Para mim, significava voltar à tecnologia crua que me deixara tão excitado com a indústria de TI em primeira instância. Para outras pessoas, o significado foi mudar de administração de sistema para desenvolvimento de software, ou mudar de um campo não relacionado à computação para programação, ou mesmo largar toda a profissão e fazer qualquer outra coisa que ama.

Assim como em desenvolvimento de software, o custo da mudança não precisa ser alto. É claro, pode ser difícil mudar de testador de software para advocacia. Mas mudar seu caminho de gerência para programação, ou vice-versa, não é difícil. Também é encontrar uma nova empresa para a qual trabalhar. Ou mudar de cidade.

E ao contrário de, digamos, construir um arranha-céu, mudar sua carreira não requer que você jogue fora tudo que você já fez. Eu fiquei programando em Ruby até agora, mas minha experiência como gerente, ou a criação de uma operação de desenvolvimento offshore, são constantemente relevantes e prestativas no que eu faço. Meus empregadores e clientes entendem isso e tiram proveito.

É importante perceber que mudanças não são apenas possíveis em sua carreira, como são **necessárias**. Como um desenvolvedor de software, você nunca gostaria de desenvolver algo que seu cliente não quer. Metodologias ágeis ajudam a prevenir isso. O mesmo é válido para sua carreira. Determine grandes objetivos, mas faça correções constantes ao longo do caminho. Aprenda com a experiência, e modifique as metas conforme você avança. Em última instância, um cliente feliz é o que todos queremos (especialmente quando, na medida em que planejamos nossas carreiras, somos os nossos próprios clientes) — não um requerimento completo.



## CAPÍTULO 52

# Melhor que ontem

Consertar um erro é (geralmente) fácil. Algo está quebrado, porque alguém relatou. Se você consegue reproduzir o erro, então consertar um erro significa corrigir qualquer mau funcionamento que o causou e verificar que ele não é mais reproduzível. Se ao menos todos os problemas fossem tão simples!

Mas nem todo problema ou desafio é tão discreto. A maioria dos desafios importantes na vida se manifestam como grandes e intransponíveis gotas amorfas de fracasso em potencial. Isso é verdade em desenvolvimento de software, gerenciamento de carreira, e até estilo de vida e saúde.

Um sistema complexo e cheio de erros precisa ser revisto. Sua carreira está estagnando a cada minuto. Você está passivamente deixando que seu estilo de vida de programador sedentário baseado em uma escrivaninha transforme seu corpo em uma desordem. Todos esses problemas são muito maiores e mais difíceis de serem **simplesmente consertados** do que um bug. Eles são complexos, difíceis de medir, e compostos de múltiplas soluções pequenas e diferentes, algumas das quais vão fracassar!

Devido a essa complexidade, nós facilmente nos desmotivamos para as grandes questões e, em vez disso, viramos nossa atenção a coisas que são mais fáceis de se medir e mais rápidas de se consertar. Eis porque procrastinamos. E procrastinação gera culpa, o que faz com que nos sintamos mal e, portanto, procrastinemos um pouco mais.

Como eu mencionei em 49, eu tenho lutado para me manter em forma desde que me posso me lembrar. De fato, quando você está miseravelmente fora de forma, “Simplesmente entre em forma” não é um conceito que você pode sequer compreender, muito menos fazer algo concreto a respeito disso. Para dificultar, se você fizer algo para melhorá-lo, você não vai conseguir notar imediatamente, ou mesmo passada uma semana, que algo mudou. Aliás, pode ser que você passe **todo o dia** treinando para entrar em forma, e uma semana depois você não tenha mais nada a oferecer.

Esse é o tipo de desmotivador que pode pular na sua frente e o derrotar antes mesmo de você começar.

Eu tenho trabalhado seriamente nesse problema. Ir à academia quase diariamente, alimentar-me melhor. Mas mesmo quando estou levando o programa a sério, é difícil ver os resultados.

Enquanto eu estava chafurdado em minha desmotivação uma tarde dessas, meu amigo Erik Kastner postou uma mensagem no Twitter, com o seguinte texto:

*Ajude-me a entrar em forma... pergunte-me uma vez por dia: “Hoje foi melhor que ontem?” (nutrição / exercício) - hoje: SIM!*

Quando li isso, percebi que esse era o ingresso para entrar em forma. Eu o reconheci dos grandes problemas que eu resolvi **com sucesso** em minha vida. O segredo é focar em fazer o que quer que você esteja tentando ficar **melhor hoje do que era ontem**. É isso. É fácil. E, assim como Erik estava, é possível sentir entusiasmo quando se dá passos reais e tangíveis em direção a uma meta distante.

Eu também estive recentemente trabalhando em uma das aplicações Ruby on Rails mais complexas e feias que já vi. Minha empresa a herdou de outro desenvolvedor como um projeto de consultoria. Havia poucas características chave que precisavam ser implementadas e uma enorme quantidade de bugs e questões de performance para serem corrigidas. Quando abrimos o capô para realizar essas mudanças, descobrimos uma gigantesca bagunça. A empresa estava com restrições de tempo e dinheiro, então não tínhamos o luxo de começar do zero, mesmo que aquilo sendo o clássico tipo de código que te dá vontade de jogar fora.

Portanto, fizemos pequenas correções após pequenas correções, levando muito



mais tempo para terminar cada uma do que era esperado. Quando começamos, parecia que a monstruosidade da base do código nunca ia se dissipar. Trabalhar naquela aplicação era cansativo e chato. Mas com o tempo, as correções se tornaram mais rápidas, e a performance antes inaceitável da aplicação havia melhorado. Isso aconteceu porque nós tomamos a decisão de tornar a base do código cada dia melhor do que era no dia anterior. Algumas vezes, isso significava refatorar um longo método em vários menores e mais organizados. Às vezes, significava remover hierarquias de herança que nunca pertenceram ao modelo de objetos. Outras vezes, simplesmente significava consertar um teste de unidade bastante quebrado.

Porém, desde que fizemos essas mudanças de forma incremental, elas vieram “de graça”. Refatorar um método é algo que você poderia fazer enquanto você estava tomando uma xícara de café ou conversando com um colega sobre as últimas notícias. E fazer uma pequena melhoria é motivador. Você pode claramente ver a diferença nessa **pequena coisa** que você consertou, tão logo quanto a mudança foi feita.

Mas talvez você não consiga ver uma diferença notável no **total** com cada mudança incremental. Quando você está tentando se tornar mais respeitado em seu lugar de trabalho, ou ser mais saudável, as melhorias individuais que você realiza cada dia frequentemente não apontam diretamente para resultados tangíveis. Essa é, como vimos antes, a razão pela qual grandes metas como essas se tornam tão desmotivantes. Portanto, para a maioria dos objetivos grandes e difíceis pelos quais você está lutando, é importante pensar não tanto em se aproximar à meta cada dia, mas sim pensar em **sair-se melhor** em seus esforços do que ontem. Eu não posso, por exemplo, garantir que serei menos gordo hoje do que ontem, mas eu **posso** controlar se eu estou fazendo mais hoje para perder peso. E se eu o fizer, tenho o direito de me sentir bem com o que fiz. Essas melhorias consistentes e mensuráveis em minhas **ações** me liberta do ciclo de culpa e procrastinação que geralmente derrota muitos de nós quando tentamos fazer as Coisas Mais Importantes.

Você também precisa se sentir feliz com **pequenas** quantidades de “melhor”. Escrever um teste a mais do que você fez ontem é suficiente para aproximá-lo mais ao objetivo de “ser melhor em teste de unidade”. Se você está começando do zero, um teste adicional por dia é uma taxa sustentável, e quando chegar o momento em que você não consegue mais fazer melhor que ontem, você descobrirá que você já se tornou “melhor em teste de unidade” e que não precisa mais continuar fazendo as mesmas melhorias. Se, por outro lado, você decidiu partir do zero em direção a 50 testes no primeiro dia de seu plano, o primeiro dia será difícil e o segundo provavelmente não acontecerá. Portanto, faça suas melhorias pequenas e incrementais, mas,

sobretudo, **diariamente**. Pequenas mudanças também diminuem o preço da falha. Se você perder um dia, você terá uma nova base para amanhã.

Uma das melhores coisas sobre essa simples máxima é que você pode aplicá-la para metas muito táticas, tais como terminar um projeto ou melhorar um trecho do programa.

Como você agiu melhor hoje para melhorar sua carreira do que você fez ontem? Faça mais um contato, submeta um *patch* a um projeto open source, escreva um post bem pensado e o publique em seu blog. Ajude mais uma pessoa em um fórum técnico de sua área, do que você fez ontem. Se todo dia você fizer algo um pouco melhor do que ontem para melhorar a si mesmo, você descobrirá que o que era uma distância oceânica para construir uma carreira notável se tornou mais palpável.

## Faça algo

- 1) Faça uma lista de melhorias complexas e difíceis que você gostaria de realizar; podem ser tanto de âmbito pessoal como profissional. Está tudo bem se você tem uma lista razoavelmente longa. Agora, para cada item da lista, pense sobre o que você poderia fazer hoje para melhorar ou fazer aquele item melhor que ontem. Amanhã, olhe para a lista de novo. Ontem foi melhor do que o dia anterior? Como você pode fazer hoje melhor? Faça o mesmo no próximo dia. Coloque isso em seu calendário. Use dois minutos pensando sobre isso cada manhã.

## CAPÍTULO 53

# Seja independente

Em momentos de estresse, eu geralmente olho para os meus dias em uma grande empresa. Eu estava tão encaixadinho no meu próprio escritório, ou cubículo, e em uma espessa e macia camada da hierarquia de gerência. Era uma piada para nós ali, mas em uma grande empresa, uma pessoa esperta pode permanecer sem necessariamente **concluir nada**. Na maioria dos casos, se um projeto não se tornar pronto, havia gente suficiente dividindo a culpa em camadas suficientes, sendo difícil descobrir onde as coisas deram errado. E isso é apenas para as falhas. Se as coisas demorassem mais do que deveriam, a complexidade da organização obscurecia as razões a um ponto em que ninguém realmente tinha a noção de quanto tempo qualquer projeto deveria levar para ficar pronto.

Então, em um dia em que você não está disposto a realmente pisar no acelerador, uma grande empresa lhe concede a oportunidade de sentar-se e, digamos, navegar na internet por um tempo. Ou ir para casa mais cedo. Ou tirar um dia de licença. Apesar de todas as reclamações que fiz durante minha vida sobre empresas grandes, definitivamente havia suas vantagens.

O problema é que o manto de segurança de uma hierarquia corporativa diminui sua velocidade. Se você consegue se esconder por trás do escudo da mediocridade que a maioria das divisões corporativas impunham, não há muito incentivo para exceder. Mesmo aqueles de nós que são geralmente bem intencionados são tentados pelo oásis paradisíaco do Youtube ou sua coleção favorita de *web comics*. Se você estiver procurando por um, tente <http://toothpastefordinner.com>. Ri durante horas.

Nesse sentido, uma grande empresa se torna um lugar maravilhoso para ir e semi-aposentar-se se você está cansado. Mas se você está lutando para ser notável (o que você está!), uma grande empresa é um lugar difícil para entrar no ritmo certo, do mesmo jeito que uma padaria é um lugar ruim para tentar eliminar seus pneuzinhos.

A solução? Seja independente!

Você possui um conjunto de habilidades. Você as afiou. Você sabe o que você vale. Tornar-se um alguém independente é um dos testes derradeiros. Você não tem burocracia por trás da qual se esconder. Você é diretamente encarregado das pessoas pagando as contas. A ideia de que você está fornecendo um serviço se torna diretamente aparente em tudo o que você faz. Não há equipe com quem dividir a culpa quando você faz coisas erradas. É somente você, seus conhecimentos e sua habilidade de executar.

Tornar-se independente também o força a aprender como fazer marketing de si mesmo e, ao mesmo tempo, testar suas escolhas de domínio e tecnologia em que focar. Você não pode depender de ser encontrado pelos clientes quando você se torna independente, no sentido como era em uma grande empresa, quando o trabalho o encontrava. Você precisa sair e **encontrar** seus clientes. Uma vez que você os achou, você precisa convencê-los de que você é digno de seu pagamento.

Você também precisa decidir quanto você quer cobrar. O que você faz custa 50 por hora? Ou custa 250?

Como você irá pagar suas contas? Como você justificará a quantia que você vale?  
**Você realmente vale o tanto que você achava que valia?**

Ser independente é difícil. Coloca todas suas habilidades enquanto profissional em teste. Você **talvez** não esteja pronto para isso ainda. A boa notícia é que você não precisa percorrer o caminho inteiro. Considere isso como um projeto de desenvolvimento pessoal, e coloque-se no mercado em seu tempo livre. Determine uma meta de ser contratado e conclua-a com um cliente satisfeito. Trabalhe nisso à noite e aos finais de semana (mas por favor, não trabalhe em seu cubículo, em seu trabalho fixo!). Você aprenderá muita coisa sem perder sua rede de segurança. Na pior das hi-

póteses, você trabalhará demais por algumas semanas, falhará em um projeto, e será mandado de volta para sua confortável cadeira com uma nova sensação de apreciar seu emprego. No melhor dos casos, você será amplamente bem sucedido, amará o trabalho, e se colocará em um novo caminho em direção a uma satisfatória carreira e à recompensa financeira.

O revisor Sammy Larbi sugere outra alternativa para ser independente. Se você atualmente trabalha para uma grande empresa, considere juntar-se a uma pequena. Se você trabalha para uma empresa consagrada, tente uma *startup*. Em uma pequena *startup*, você pode tirar o melhor dos dois mundos: um emprego de tempo integral com um salário e o desafio de ser colocado contra os problemas de seu negócio, sem filtro.

## Curiosidade é um ponto forte

*por Mike Clark*

Meus pais falariam que eu era uma criança curiosa. Fazia várias perguntas, lia tudo que eu tinha em mãos, e aprendia como as coisas funcionam desmontando-as. Como se vê, isso não foi apenas uma fase — eu nunca deixei de ter uma curiosidade insaciável. É fácil negligenciar, mas acredito que curiosidade pode ser um ponto forte. Às vezes, só é necessário um pouco de prática para desenvolvê-la.

Olhando para trás, consigo identificar vários eventos que mudaram minha carreira, que ocorreram principalmente porque eu segui uma curiosidade. Aqui ofereço os seguintes exemplos, na esperança de que eles o encorajem a escutar quando a curiosidade chama:

Eu nunca descobri que viraria um programador. Sempre fui fascinado por aviões e naves espaciais, então entrar para o programa de engenharia espacial da Embry-Riddle Aeronautical University parecia a escolha lógica. Depois de um ano olhando por aí, entretanto, descobri que a galera do departamento de Ciência da Computação estava se divertindo muito mais. Como parte do novo programa de graduação, eles estavam aplicando ciência da computação em problemas relacionados à aviação. No ensino médio eu havia me tornado curioso sobre computadores, mas nunca realmente tinha considerado programação como uma carreira. Então, comecei a sair com os geeks da computação para ver o que eles faziam. Em pouco tempo, eu troquei os programas de graduação. Essa mudança por si só acabou sendo uma das minhas melhores decisões. Os cursos ainda eram desafiadores, mas eu amava cada minuto. Minha curiosidade inicial em programação rapidamente se tornou uma paixão que me fez me inscrever em um estágio da NASA e começar minha carreira em software

com um grande salto. E até o dia de hoje, eu nunca subestimo a recompensa em potencial de descobrir em que os companheiros geeks estão trabalhando por diversão.

Toda vez que me sinto confortável, sei que é hora de tentar algo novo. Após muitos anos escrevendo software embarcado na indústria aeroespacial, eu estava confortável (o que, para mim, sempre está associado a tédio) com C e C++. Naquela época, programação web acordou minha curiosidade, principalmente porque era radicalmente diferente de programação de sistemas embarcados. Infelizmente, o projeto do meu emprego não tinha acesso a web (era um daqueles projetos ultrasecretos), então, resolvi passar minhas noites e finais de semana aprendendo a escrever software para web. Esse interesse lateral eventualmente se tornou uma oportunidade de trabalhar em um novo projeto, usando Java. Acabei construindo aplicações web baseadas em Java para muitos mais projetos... e empregadores. Minha curiosidade sobre desenvolvimento web era o catalisador para diversificar minhas habilidades, o que acabou sendo uma boa mudança de carreira.

Eu aprendi Ruby on Rails por um capricho. Ruby era uma linguagem divertida que me fazia pensar diferente sobre programação. Rails fazia o mesmo para as aplicações web. Eu não tinha nenhum cliente naquela época que estava contratando trabalho em Ruby on Rails, mas isso não importava. Eu era curioso, e simplesmente não conseguia evitar. Eu peguei as horas menos úteis e usei para me dedicar ao Ruby on Rails. Mal sabia eu que no começo de 2005 eu receberia a oportunidade de construir uma das primeiras aplicações comerciais Rails e seria convidado por Dave Thomas para ajudá-lo em seu livro de Rails. Minha curiosidade sobre outra nova tecnologia começou outra curva de sucesso em minha carreira.

Minha curiosidade vai além da tecnologia; aspectos de negócio são igualmente interessantes para mim. Isso me levou a me aventurar por conta própria como um consultor independente e a começar uma empresa de treinamento (*The Pragmatic Studio*). Minha curiosidade no funcionamento de uma pequena empresa me deu a oportunidade de aprender um monte de novas habilidades: vendas, marketing, suporte ao cliente e assim por diante. Ver o grande cenário me ajudou a me tornar um programador melhor.

Então, onde está **sua** verdadeira curiosidade? Tente seguir seus interesses por um tempo, e veja o que acontece. Você pode se surpreender!

*Mike Clark é um programador/consultor independente*

## CAPÍTULO 54

# Divirta-se

*Mas eu vos digo que, quando trabalhais, realizais parte do sonho mais longínquo da terra, desempenhando assim uma missão que vos foi designada quando esse sonho nasceu. E, apegando-vos ao trabalho, estareis na verdade amando a vida. E quem ama a vida através do trabalho, partilha do segredo mais íntimo da vida.*

– Kalil Gibran, O profeta

Se você alcançou o ponto de ser um desenvolvedor de software com o luxo de realmente pensar sobre qual caminho você quer que sua carreira tome, parabéns! Você pode se considerar bastante sortudo. Existem muitas culturas nas quais poder decidir o que você faz como profissão é um enorme privilégio que poucas pessoas disfrutam. Como um programador, é provável que você não precise se preocupar em como pagar por um lugar para viver ou como comprar comida.

Você poderia ter escolhido dentre inúmeras carreiras, mas esta é excitante. É criativa. Requer pensamento profundo e lhe traz como recompensa a sensação de conseguir fazer algo que a maioria das pessoas que você encontra todo dia não imagina que seja possível. Nós nos preocupamos com progredir para o próximo nível,

causar um impacto, ou ganhar respeito dos nossos colegas na indústria, mas se você realmente parar para pensar nisso, nós nos saímos muito bem.

Desenvolvimento de software é tanto desafiador **como** recompensador. É criativo como uma forma de arte, mas (diferentemente da arte) fornece valor concreto e mensurável.

Desenvolvimento de software é divertido!

Por fim, a coisa mais importante que eu aprendi nessa jornada que minha carreira em programação tem sido é que não é o que você faz como profissão, ou o que você **tem**, que importa. É como você escolhe aceitar essas coisas. É interno. Satisfação, como nossa escolha de carreira, é algo que devia ser buscado depois e **decidido intencionalmente**.



CAPÍTULO 55

## Nota da editora

A tradução deste livro foi feita por Adriano Almeida e Vivian Matsui. Em caso de dúvidas ou sugestões, você pode entrar em contato com os editores através dos e-mails [adriano.almeida@casadocodigo.com.br](mailto:adriano.almeida@casadocodigo.com.br) e [vivian.matsui@casadocodigo.com.br](mailto:vivian.matsui@casadocodigo.com.br).



# Referências Bibliográficas

- [1] Douglas Coupland. *Microserfs*. Regan Books, 1996.
- [2] Tom Demarco and Timothy Lister. *Peopleware: Productive Projects and Teams*. Dorset House, 1999.
- [3] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [4] Seth Godin. *Purple Cow: Transform Your Business by Being Remarkable*. Portfolio, 2003.
- [5] Gary Hamel. *Leading the Revolution: How to Thrive in Turbulent Times by Making Innovation a Way of Life*. 2002.
- [6] Thich Nhat Hanh. *The Miracle of Mindfulness*. Beacon Press, 1999.
- [7] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, 2000.
- [8] Robert M. Pirsig. *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*. Perennial Classics, 2000.
- [9] Steven A. Silbiger. *The Ten-Day MBA: A Step-By-step Guide To Mastering The Skills Taught In America's Top Business Schools*. Quill, 1999.