# Equipes de Software

## Um guia para o desenvolvedor de software se relacionar melhor com outras pessoas

**Brian W. Fitzpatrick**
**Ben Collins-Sussman**

Novatec

Editor: Rubens Prates
Tradução: Eduardo Kraszczuk
Revisão técnica: Joel Saade
Revisão gramatical: Patrizia Zagni
Editoração eletrônica: Carolina Kuwabata

ISBN: 978-85-7522-329-1

Histórico de impressões:

Janeiro/2013 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
Email: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/nova

**Dados Internacionais de Catalogação na Publicação (CIP)**
**(Câmara Brasileira do Livro, SP, Brasil)**

Fitzpatrick, Brian W.
Equipes de software / Brian W. Fitzpatrick e Ben Collins-Sussman ; tradução Eduardo Kraszczuk. -- São Paulo : Novatec Editora ; Sebastopol, CA : O'Reilly, 2012.

Título original: Team geek : a software developer's guide to working well with others.
ISBN 978-85-7522-329-1

1. Programação (Computadores) - Aspectos psicológicos 2. Programação (Computadores) - Aspectos sociais 3. Programadores de software 4. Relações humanas 5. Trabalho em equipe I. Collins-Sussman, Ben. II. Título.

12-14052 CDD-005.1

Índices para catálogo sistemático:

1. Programação de computadores 005.1
2. Programadores de software 005.1

VC20121203

CAPÍTULO 1

# O mito do programador gênio

Uma vez que este livro é sobre os perigos sociais do desenvolvimento de software, faz sentido nos concentrarmos na única variável sobre a qual você tem controle: você.

As pessoas são inerentemente imperfeitas. Mas antes que você possa entender os bugs nos seus colegas, você precisa entender os bugs em si mesmo. Vamos lhe pedir para pensar sobre suas próprias reações, comportamentos e atitudes – e, em troca, esperamos que você obtenha alguma compreensão real sobre como se tornar um engenheiro de software mais eficiente e de sucesso. Você vai acabar gastando menos energia lidando com problemas relacionados a pessoas e mais tempo escrevendo códigos excelentes.

A ideia essencial neste capítulo é compreender que o desenvolvimento de software é um esporte coletivo. Para ter sucesso em uma equipe de engenharia, você precisa reorganizar seus comportamentos de acordo com os princípios básicos de humildade, respeito e confiança.

Antes de colocarmos o carro na frente dos bois, vamos começar observando como os programadores se comportam em geral.

## Ajude-me a esconder meu código

Nós dois temos ministrado muitas palestras em conferências sobre programação nos últimos seis anos. Já que somos parte

da equipe original que lançou o serviço open source Hospedagem de Projetos do Google em 2006, costumávamos ouvir muitas perguntas e solicitações sobre o produto. Em meados de 2008, percebemos uma tendência distinta no tipo de solicitações que estávamos recebendo:

> *Por favor, no Google Code, vocês conseguem que o Subversion oculte desvios específicos?*
>
> *Vocês podem criar projetos open source que ficam escondidos do mundo no começo, e são revelados quando estiverem prontos?*
>
> *Oi! Quero reescrever todo o meu código desde o início. Por favor, vocês podem limpar todo o histórico?*

Você consegue perceber o tema comum dessas solicitações?

O motivo principal é a *insegurança*. As pessoas têm medo que outras vejam e julguem seu trabalho em andamento. De certa forma, isso é só parte da natureza humana – ninguém gosta de ser criticado, especialmente por coisas que não estão finalizadas. Essa atitude nos deu a dica para uma tendência no desenvolvimento de software. Na verdade, a insegurança é o sintoma de um problema maior.

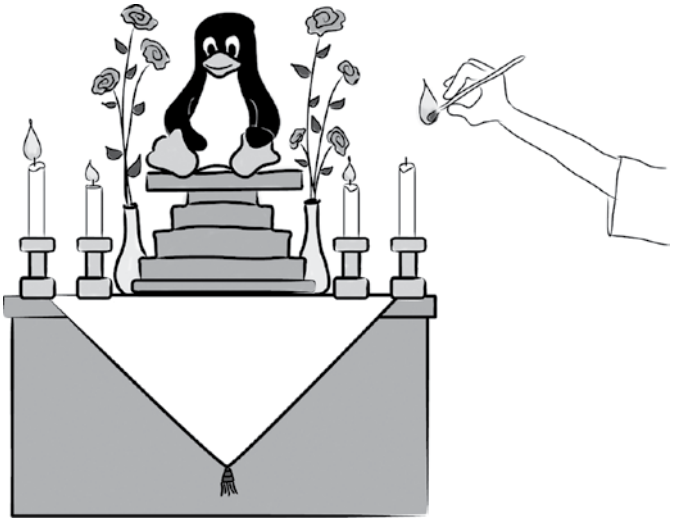
## O mito do gênio

Primeiro, vamos ser claros: não somos realmente fãs de esportes. Quando nossas esposas torcem por beisebol ou futebol americano na TV, coçamos a cabeça e ficamos imaginando o que é tão empolgante. Mesmo assim, vivemos o começo dos anos 1990 e testemunhamos a incrível série de campeonatos vencidos pelo Chicago Bulls (a propósito, esse é um time de basquete). Nós dois estávamos em Chicago durante esse período e a mídia nacional ficou anos saturada com histórias sobre esse time extraordinário.

Sobre o que ouvíamos mais na TV e nos jornais? Não sobre o time, mas sobre Michael Jordan, o superastro. Todos os

jogadores do mundo queriam *ser* ele. Víamos Jordan dar voltas ao redor dos outros jogadores. Assistimos aos comerciais de televisão em que ele aparecia. Fomos ver filmes bobos em que ele jogava basquete com personagens de desenho animado. Ele era uma estrela, e todas as crianças que praticavam arremesso ao cesto ao redor do mundo desejavam secretamente crescer e seguir seus passos.

Os programadores têm esse mesmo instinto – encontrar ídolos e adorá-los. Linus Torvalds, Richard Stallman, Bill Gates – todos heróis que mudaram o mundo com feitos heroicos. Linus escreveu o Linux sozinho, certo?



*Cuidado com o instinto natural de divinizar as coisas.*

Na verdade, Linus escreveu somente o começo da prova de conceito de um kernel semelhante ao Unix e mostrou-o a uma lista de remetentes via email. Essa foi uma tarefa pequena e, definitivamente, uma realização impressionante, mas tratou-se só da ponta do iceberg. O Linux é centenas de vezes maior do que isso e foi desenvolvido por centenas de pessoas inteligentes. A realização real de Linus foi *liderar* essas

pessoas e coordenar seu trabalho. O Linux é o resultado do seu trabalho coletivo (o Unix foi escrito por um pequeno grupo de pessoas inteligentes no Bell Labs, e não inteiramente por Ken Thompson e Dennis Richie).

Do mesmo modo, Stallman escreveu pessoalmente todos os softwares da Free Software Foundation? Ele escreveu a primeira geração de Emacs. Contudo, centenas de outras pessoas foram responsáveis pelo Bash, pelo conjunto de ferramentas GCC e por todo o restante dos softwares que rodam no Linux. Steve Jobs liderou a equipe que criou o Macintosh, e embora Bill Gates seja conhecido por escrever um interpretador BASIC para os primeiros computadores pessoais, sua maior realização foi construir uma empresa de sucesso com base no MS-DOS. Ainda assim, todos se tornaram líderes e símbolos das suas realizações coletivas.

E quanto a Michael Jordan?

É a mesma coisa. Nós o divinizamos, mas o fato é que ele não venceu sozinho todos os jogos de basquete. Sua verdadeira genialidade estava no modo como ele trabalhava *com* sua equipe. O treinador da equipe, Phil Jackson, era extremamente inteligente – suas técnicas de treino são lendárias: ele reconheceu que um jogador sozinho nunca vence um campeonato e, então, montou um "equipe dos sonhos" ao redor de Michael Jordan. A equipe era uma máquina bem azeitada – no mínimo tão impressionante quanto o próprio Michael.

Então, por que divinizamos repetidamente o indivíduo nessas histórias? Por que as pessoas compram produtos endossados por celebridades? Por que queremos comprar o vestido de Michelle Obama ou os sapatos de Michael Jordan?

A celebridade é responsável por grande parte disso. Os humanos têm um instinto natural para identificar líderes e modelos de comportamento, divinizá-los e tentar imitá-los. Todos precisamos de heróis para nos inspirar e o mundo da programação

também tem seus heróis. O fenômeno da "celebridade técnica" quase se transformou em uma mitologia. Todos queremos escrever algo que mude o mundo como o Linux ou projetar a próxima brilhante linguagem de programação.

Lá no fundo, desejamos secretamente ser gênios. A fantasia geek definitiva é descobrir um incrível conceito novo. Você vai para sua Batcaverna durante semanas ou meses, para implementar melhor sua ideia. Então, você "solta" seu software no mundo, chocando todos com sua genialidade. Seus colegas ficam pasmos com sua inteligência. As pessoas fazem fila para usar seu software. Fama e fortuna vêm naturalmente.

Mas espere: vamos fazer uma verificação da realidade. Provavelmente você não é gênio.

Sem ofensa, é claro – temos certeza de que você é muito inteligente. Contudo, você entende como são raros casos reais de genialidade? É claro que você escreve código e esta é uma habilidade complicada que provavelmente o coloca um pouco acima da média da população. Mas mesmo que você seja um gênio, *isso não é suficiente*. Gênios também cometem erros e ter ideias brilhantes e habilidades em programação não garante que seu software será um sucesso. O que vai alavancar ou destruir sua carreira é o quão bem você colabora com os outros.

Na verdade, o mito do gênio é só outro aspecto da nossa insegurança. A maioria dos programadores tem medo de compartilhar trabalho que eles tenham acabado de começar, porque isso significa que seus colegas irão ver seus erros e saber que o autor do código *não é um gênio*. Para citar outro programador do blog de Ben:

> *Sei que fico muito inseguro com pessoas olhando antes que algo esteja pronto. Sinto-me como se elas fossem me julgar seriamente e me achar um idiota.*

Esse é um sentimento extremamente comum entre os programadores, e a reação natural é se esconder em uma caverna e

trabalhar, trabalhar, trabalhar. Ninguém vai ver seus erros. Você ainda terá uma chance de exibir sua obra-prima quando ela estiver pronta. Esconde-a até que tudo esteja perfeito.

Outra motivação comum para manter suas cartas perto do peito é o medo de que outro programador possa pegar sua ideia e fugir com ela antes de você ter chance de trabalhar melhor nela. Ao manter a ideia em segredo, você a controla.

Sabemos o que você provavelmente está pensando agora: e daí? As pessoas não deveriam trabalhar como quisessem?

Na verdade, não. Nesse caso, afirmamos que você está trabalhando de modo errado, e isso é importante. Eis o porquê.

## Esconder é considerado prejudicial

Se você passar todo o tempo trabalhando sozinho, estará aumentando o risco de fracasso e prejudicando seu potencial de crescimento.

Antes de mais nada, como você sabe se está no caminho certo?

Imagine que você é um entusiasta de um projeto de bicicletas e um dia tem uma ideia brilhante para um modo completamente novo de projetar um câmbio. Você compra as peças e passa semanas enfurnado na garagem tentando construir um protótipo. Quando seu vizinho – também um entusiasta das bicicletas – pergunta o que você está fazendo, você decide não falar sobre isso. Você não quer que ninguém saiba sobre seu projeto até que ele esteja absolutamente perfeito. Mais alguns meses se passam e você continua tendo problemas em fazer seu protótipo funcionar corretamente. Mas, já que você está trabalhando em segredo, é impossível pedir conselhos aos amigos com vocação mecânica.

Então, um dia, seu vizinho tira a bicicleta da garagem com um novo e radical mecanismo de câmbio. Ele estava construindo algo muito similar à sua invenção, mas com a ajuda de alguns

amigos da loja de bicicletas. Nesse ponto, você está irritado. Você mostra a ele seu trabalho. Ele diz que seu projeto tem algumas falhas simples – falhas que poderiam ter sido consertadas na primeira semana se você tivesse mostrado a ele.

*Trabalhar isolado muitas vezes leva ao desapontamento.*

Há algumas lições para aprender aqui. Se você mantiver sua grande ideia escondida do mundo e se recusar a mostrar qualquer coisa a qualquer um até que a implementação esteja pronta, você estará fazendo uma aposta muito grande. É fácil cometer erros fundamentais de projeto no início. Você se arrisca a reinventar a roda.[1] Você também abandona os benefícios da colaboração: percebeu como seu vizinho avançou mais rápido ao trabalhar com outras pessoas? É por isso que as pessoas mergulham os dedos do pé na água antes de entrar na parte funda: você precisa ter certeza de que está trabalhando na coisa certa, que a está fazendo corretamente e que ela nunca foi feita antes. A probabilidade de dar um passo errado é alta no início. Quanto mais feedback você receber no começo, mais você reduzirá esse risco.[2]

1 Literalmente, se você for, de fato, um projetista de bicicletas.

2 Devemos observar que às vezes é perigoso receber feedback muito cedo no processo. Vamos abordar esse assunto em um capítulo posterior.

Lembre-se do consagrado mantra: "Falhe no começo, falhe rápido, falhe com frequência". Vamos discutir a importância do fracasso em detalhes mais adiante no livro.

Compartilhar no início não é só sobre impedir erros pessoais e ter suas ideias vetadas. Também é importante fortalecer o que chamamos o *fator ônibus* do seu projeto.

*Fator ônibus* (substantivo): o número de pessoas que precisam ser atropeladas por um ônibus antes que seu projeto esteja completamente condenado.

*Qual é o fator ônibus da sua equipe?*

Quanto conhecimento e know-how estão contidos em seu projeto? Se você for a única pessoa que entende como funciona o código do protótipo, isso poderá conferir uma boa segurança a seu emprego, mas também significará que o projeto estará perdido se você for atropelado por um ônibus. Entretanto, se você estiver trabalhando com um amigo, você dobrou o fator ônibus. Se você tiver uma pequena equipe fazendo junta os projetos e protótipos, a situação estará ainda melhor – o projeto não vai acabar quando um membro da equipe sair. Lembre-se: os membros da equipe podem não ser literalmente atropelados por um ônibus, mas outros eventos

imprevisíveis da vida ainda podem acontecer. Alguém pode se casar, mudar-se, deixar a empresa ou precisar cuidar de um parente doente. Você precisa proteger o futuro de um projeto gerenciando o fator ônibus.

Além do fator ônibus, existe o problema do passo geral do progresso. É fácil esquecer que trabalhar sozinho é muitas vezes um trabalho árduo, muito mais lento do que as pessoas querem admitir. O quanto você aprende quando trabalha sozinho? Com que velocidade você progride? A web é um excelente local para obter opiniões e informações, mas não substitui a experiência humana real. Trabalhar com outras pessoas aumenta diretamente o conhecimento coletivo por trás do esforço. Quando você fica preso em algo absurdo, quanto tempo perde para sair dessa situação? Pense sobre como a experiência seria diferente se você tivesse alguns colegas para olhar seu trabalho e dizer a você – instantaneamente – o que você errou e como superar o problema. É exatamente por isso que as equipes se sentam juntas (ou fazem programação em pares) nas empresas de engenharia de software: muitas vezes você precisa de um segundo par de olhos.

Eis outra analogia: pense sobre como você trabalha com seu compilador. Quando se senta e escreve um grande trecho de software, você passa dias ou semanas escrevendo código e, então, quando acha que tudo está acabado e completamente perfeito, aperta o botão "compilar" pela primeira vez? É claro que não. Você pode imaginar que tipo de desastre iria acontecer se tentasse compilar 50 mil novas linhas de código? Como programadores, trabalhamos melhor em laços *estreitos* de feedback: escreva uma nova função e compile. Adicione um teste e compile. Refaça algum código e compile. Corrigimos os erros de digitação e bugs o mais rápido possível depois de gerar o código. Queremos o compilador do nosso lado para cada pequeno passo, ajudando-nos. Alguns ambientes podem até compilar nosso código *conforme digitamos*. É assim

que mantemos a alta qualidade dos códigos e garantimos que nosso software está evoluindo corretamente bit por bit.

O mesmo tipo de laço de feedback rápido é necessário não só no nível do código, mas também no nível do projeto como um todo. Projetos ambiciosos evoluem rapidamente e têm que se adaptar a ambientes em mudança conforme avançam. Os projetos encontram obstáculos imprevisíveis, obstáculos políticos ou simplesmente descobrem que as coisas não estão indo como planejado. As solicitações mudam inesperadamente. Como você consegue esse laço de feedback para saber o instante em que seus planos ou projetos precisam mudar? Resposta: trabalhando em equipe. Eric Raymond muitas vezes é citado por ter dito: "Muitos olhos tornam todos os bugs superficiais", mas uma versão melhor seria: "Muitos olhos garantem que seu projeto permaneça relevante e nos trilhos". Pessoas que trabalham em cavernas acordam para descobrir que, embora sua visão original possa estar completa, o mundo mudou e tornou o produto irrelevante.

## Engenheiros e escritórios

Vinte anos atrás, a sabedoria convencional dizia que para um engenheiro ser produtivo, ele precisava ter seu próprio escritório com uma porta que pudesse ser fechada. Supostamente, esse era o único modo de ele ter grandes períodos ininterruptos para se concentrar profundamente em escrever páginas de código.

Para a maioria dos engenheiros,[3] achamos que ter um escritório privado é não só desnecessário, mas também perigoso. Hoje, o software é escrito por equipes, não por indivíduos, e uma conexão com grande largura de banda prontamente disponível com o restante da sua equipe é ainda mais valiosa que sua conexão com a Internet. Você pode ter todo o tempo do mundo, sem interrupção, mas se o estiver usando para trabalhar no *item errado*, estará perdendo tempo. Entre no escritório de qualquer empresa de alta

3 Entretanto, reconhecemos que os introvertidos provavelmente precisam de mais espaço, silêncio e tempo a sós do que a maioria das pessoas e podem se beneficiar de um ambiente mais silencioso, se não do seu próprio escritório.

tecnologia que esteja crescendo rapidamente e tenha sido iniciada no século XXI e você encontrará engenheiros aglomerados em baias compartilhadas (também conhecidas como "cocheiras") ou em áreas com mesas compartilhadas, mas raramente vai encontrá-los trancados em escritórios, longe uns dos outros.

É claro que você ainda precisa de um modo de filtrar o ruído e as interrupções, e é por esse motivo que a maioria das equipes que já vimos desenvolveu uma maneira de avisar que estão ocupadas e que você deve evitar interrupções. Costumávamos trabalhar em uma equipe com um protocolo de interrupções verbal: se você quisesse falar, precisava dizer "ponto de interrupção, *Mary*", em que "*Mary*" era o nome da pessoa com quem você queria falar. Se "Mary" estivesse em um ponto onde ela pudesse parar, ela giraria sua cadeira e ouviria. Se ela estivesse muito ocupada, simplesmente diria "ack" e você continuaria a trabalhar até que ela tivesse terminado.

Outras equipes distribuem fones de ouvido para os engenheiros, que cancelam o ruído e tornam mais fácil lidar com o ruído na área – na verdade, em muitas empresas o simples ato de usar fones de ouvido é um sinal comum que significa "não perturbe a menos que seja realmente importante". Outras equipes têm sinais ou animais de pelúcia que os membros da equipe colocam sobre seus monitores para indicar que eles devem ser interrompidos somente em caso de emergência.

Não nos entenda mal – ainda achamos que os engenheiros precisam de tempo sem interrupção para se concentrar em escrever código, mas também ainda mais de uma conexão com grande largura de banda com sua equipe.

---

Então, tudo se resume a isso: *trabalhar sozinho é inerentemente mais arriscado do que trabalhar com outras pessoas*. Embora você possa ter medo de alguém roubar sua ideia ou achar que você é burro, você deve ter mais medo ainda de desperdiçar muito do seu tempo trabalhando no item errado.

Infelizmente, esse problema de "esconder as ideias" não se restringe somente à engenharia de software – é um problema presente em todos as áreas. Por exemplo, a ciência profissional deveria tratar da troca livre e aberta de informações. Mas a

desesperada necessidade de "publicar ou perecer" e competir por bolsas tem o efeito exatamente oposto. Os grandes pensadores não compartilham ideias. Eles se agarram obsessivamente a elas, realizam sua pesquisa privativamente, ocultam todos os erros ao longo do caminho e, então, publicam um artigo que faz parecer que todo o processo foi fácil e óbvio. E os resultados muitas vezes são desastrosos: eles acidentalmente duplicam o trabalho de outra pessoa, ou cometem logo no início um erro não detectado, ou produzem algo que era interessante no passado, mas que hoje é visto como inútil. O tempo e o esforço desperdiçados são trágicos.

Não se torne outra estatística!

## É tudo sobre a equipe

Então, vamos dar um passo atrás e reunir essas ideias.

O ponto que estamos martelando é que no mundo da programação, os artesãos solitários são extremamente raros – e mesmo quando eles existem, não realizam feitos sobre-humanos em um vácuo. Suas realizações que mudam o mundo são quase sempre resultado de uma faísca de inspiração seguida por um heroico esforço em equipe.

Criar uma *equipe* de superastros é o objetivo real, mas extremamente difícil. As melhores equipes fazem uso brilhante dos seus superastros, mas o todo é sempre maior que a soma das partes.

Vamos colocar essa ideia em palavras mais simples:

> *O desenvolvimento de software é um esporte coletivo.*

Esse pode ser um conceito difícil a princípio, já que contradiz diretamente nossa fantasia interna do gênio programador. Tente entoá-la como um mantra.

*Lembre-se de que o desenvolvimento de software é um esporte coletivo.*

Não é suficiente ser brilhante quando você está sozinho na sua toca de hacker. Você não vai mudar o mundo ou deleitar milhões de usuários de computador se escondendo e preparando sua invenção secreta. Você precisa trabalhar com outras pessoas, compartilhar sua visão, dividir o trabalho, aprender com os outros e criar uma equipe brilhante.

Considere isso: quantos softwares amplamente usados e de sucesso você pode contar que foram realmente escritos por uma *única* pessoa? (algumas pessoas podem dizer "LaTeX", mas este dificilmente é "amplamente usado", a menos que você considere que o número de pessoas que escrevem artigos científicos é uma parte estatisticamente significativa de todos os usuários de computador!).

Vamos repetir o conceito de esporte coletivo inúmeras vezes ao longo do livro. Equipes de alto desempenho valem ouro e são a verdadeira chave para o sucesso. Você deve buscar essa experiência sempre que puder. Este livro é sobre isso.

## Os três pilares

Então, o argumento a favor de trabalhar em equipes foi defendido. Se o trabalho em equipe é o melhor caminho para produzir um excelente software, como se forma (ou se encontra) uma equipe excelente?

Isso não é tão simples. Para atingir o nirvana da colaboração, você precisa primeiro aprender e abraçar o que chamamos de "três pilares" das habilidades sociais. Esses três princípios não se destinam somente a lubrificar as engrenagens dos relacionamentos. São a base na qual todas as interações e colaborações saudáveis se pautam.

*Humildade*

Você não é o centro do universo. Não é onisciente nem infalível. Está aberto à melhoria.

*Respeito*

Você se importa realmente com as pessoas com quem trabalha. Trata-as como seres humanos e aprecia suas habilidades e realizações.

*Confiança*

Você acredita que os demais são competentes e irão fazer a coisa certa, e está de acordo em deixá-los dirigir quando for apropriado.[4]

Referimo-nos a esses princípios reunidos como HRT (*humility* – humildade, *respect* – respeito, *trust* – confiança). Pronunciamos essa sigla como *heart* (coração) e não *hurt* (machucar), porque se trata de *diminuir* a dor, e não de ferir as pessoas. De fato, nossa tese principal se baseia diretamente nesses pilares:

> *Quase todos os conflitos sociais podem ser, no final, rastreados até chegar à falta de humildade, respeito ou confiança.*

4 Essa será uma tarefa muito difícil se você ficou mal visto no passado ao ter delegado funções a pessoas incompetentes.

Pode soar implausível a princípio, mas dê uma chance. Pense em alguma situação social desagradável ou desconfortável nesse momento da sua vida. No nível mais básico, estão todos sendo apropriadamente humildes? As pessoas estão realmente respeitando umas às outras? Existe confiança mútua?

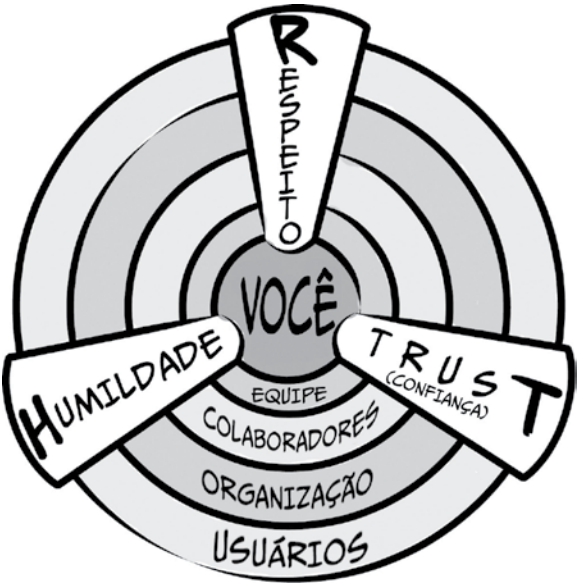
Acreditamos que esses princípios são tão importantes que estruturamos este livro em torno deles.

Este livro começa com você: fazer você abraçar o HRT e internalizá-lo de verdade, o que significa colocá-lo no centro das suas interações. O primeiro capítulo aborda esse assunto. A partir daí, criamos círculos de influência em expansão constante.

No capítulo 2, discutimos o desafio de formar uma equipe baseada nos três pilares. Criar uma cultura de equipe é a próxima etapa essencial para o sucesso – esse é o "time dos sonhos" de que falamos antes.

Examinamos, então, as pessoas que interagem diariamente com sua equipe, mas que podem não fazer parte da cultura básica da equipe. Podem ser colegas de outras equipes ou só voluntários que oferecem ajuda para seu projeto. Muitos deles não só desconsideram o HRT, mas podem ser simplesmente venenosos! Aprender a defender sua equipe deles é a primeira prioridade. Entretanto, remover suas presas e absorvê-los na sua cultura deve ser o objetivo final. É um modo excelente de expandir uma equipe.

A maioria das equipes trabalha em grandes empresas e esse ambiente pode muitas vezes ser um obstáculo tão grande quanto as pessoas venenosas. Aprender a enfrentar esses obstáculos organizacionais pode ser a diferença entre lançar um produto e ter esse mesmo produto cancelado.




*Abrace o HRT para atingir o nirvana da colaboração.*

Finalmente, consideramos os usuários do seu software. Às vezes, esquecemos que eles existem e são a vida do seu projeto. Sem usuários, seu software não tem objetivo. Os mesmos princípios de HRT que florescem na sua equipe podem e devem ser aplicados ao modo como você interage com seus usuários, e os benefícios colhidos são imensos.

Vamos parar por um momento.

Quando você escolheu este livro, provavelmente não estava pensando que estava se inscrevendo em algum tipo de grupo de suporte semanal. Simpatizamos com isso. Lidar com problemas sociais pode ser difícil. As pessoas são complicadas, imprevisíveis e muitas vezes é irritante interagir com elas. Em vez de despender energia ao analisar situações sociais e fazer movimentos estratégicos, é tentador ignorar todo o esforço. É muito mais fácil conviver com um previsível compilador, não é? Por que se importar com a parte social?

Aqui está uma citação de uma famosa palestra de Richard Hamming:[5]

> *Ao ter trabalho de contar piadas para as secretárias e ser um pouco amigável, consigo uma enorme ajuda delas. Por exemplo, uma vez, por algum motivo idiota, todos os serviços de reprodução na Murray Hill estavam ocupados. Não me pergunte como, mas estavam. Eu queria que um trabalho fosse feito. Minha secretária ligou para alguém na Holmdel, entrou no carro da empresa, fez uma viagem de uma hora, conseguiu a cópia e voltou. Essa foi sua retribuição pelas vezes que me esforcei em alegrá-la, contar piadas e ser amigável. Deu-lhe um pouco de trabalho extra que mais tarde teve um retorno para mim. Ao perceber que precisa usar o sistema e estudar como fazer o sistema executar seu trabalho, você aprende como adaptar o sistema aos seus desejos.*

A moral é esta: não subestime o poder de participar do jogo social. Não se trata de enganar ou manipular as pessoas. Trata-se de criar relacionamentos para que certas ações sejam realizadas e os relacionamentos *sempre* sobrevivam aos projetos.

## HRT na prática

Todo esse discurso sobre humildade, respeito e confiança soa como material para um sermão. Vamos descer das nuvens e pensar sobre como aplicar essas ideias em situações na vida real. Estamos procurando sugestões práticas, então vamos examinar uma lista de comportamentos específicos e exemplos com os quais você possa começar. Muitos deles podem soar óbvios a princípio, mas quando começar a pensar neles, perceberá com que frequência você (e seus colegas) são culpados de *não* os seguir.

5 "You and your research". Disponível em: *http://www.cs.virginia.edu/~robins/YouAndYourResearch.pdf*

## Esqueça o ego

Tudo bem, este é um modo simples de dizer a alguém que não é *humilde* o suficiente para parar de ter "atitude". Ninguém quer trabalhar com alguém que consistentemente se comporta como se fosse a pessoa mais importante na sala. Mesmo se você souber que é a pessoa mais inteligente na discussão, não esfregue isso na cara dos outros. Por exemplo, você sempre sente que precisa ter a primeira ou a última palavra em qualquer assunto? Sente que precisa comentar todos os detalhes de uma proposta ou discussão? Ou conhece alguém que faz isso?

Note que "ser humilde" não é o mesmo que dizer que se deve ser um capacho: não há nada de errado com a autoconfiança. Só não pareça um "sabe tudo". Ainda melhor, em vez disso, pense sobre ter um ego "coletivo". Em vez de se preocupar em ser pessoalmente incrível, tente construir um senso de realização e orgulho da equipe. A Apache Software Foundation tem uma longa história de criar comunidades ao redor de projetos de software. Tais comunidades têm identidades incrivelmente fortes e rejeitam pessoas que se preocupam mais em se promover.

> *O ego se manifesta de muitas maneiras, e muitas vezes pode ficar no caminho da sua produtividade e atrasá-lo. Eis outra excelente história da palestra de Hamming que ilustra esse ponto perfeitamente:*
>
> *"John Tukey quase sempre se vestia bastante casualmente. Ele entrava em um escritório importante e levava muito tempo antes que a outra pessoa percebesse que ele era um homem de status elevado e que era melhor ouvi-lo. Por muito tempo, John precisou superar esse tipo de hostilidade. Foi um esforço perdido! Eu não disse que você deveria se conformar. Disse: 'A aparência de conformidade leva você muito longe'. Se escolher afirmar seu ego de quantos modos quiser, dizer 'vou fazer do meu jeito', você pagará um pequeno preço fixo ao longo da sua carreira profissional. E isso, ao longo de toda uma vida, soma uma quantidade enorme de problemas desnecessários. [...] Ao perceber que precisa usar o sistema e estudar como fazer o sistema executar seu*

*trabalho, você aprende como adaptar o sistema aos seus desejos. Ou você pode lutar contra ele, como uma pequena guerra não declarada, pelo restante da vida".*

## Aprenda a fazer e a lidar com críticas

Joe começou em um novo emprego como programador. Depois da primeira semana, começou a realmente se aprofundar na base do código. Já que se importava com o que acontecia, começou a questionar delicadamente os outros membros da equipe sobre suas contribuições. Enviou revisões simples de código por email, perguntando educadamente sobre suposições de projeto ou destacando pontos onde a lógica poderia ser melhorada. Após algumas semanas, foi chamado ao escritório do seu diretor. "Qual é o problema?", perguntou Joe. "Fiz algo errado?" O diretor parecia preocupado: "Tivemos muitas reclamações sobre seu comportamento, Joe. Aparentemente, você tem sido realmente desagradável com seus colegas, criticando-os a torto e a direito. Eles estão irritados. Você precisa ser mais moderado". Joe estava completamente atônito. Em uma cultura forte, baseada em HRT, suas revisões de código seriam bem-vindas e apreciadas por seus colegas. Entretanto, nesse caso, Joe deveria ter sido mais sensível à insegurança generalizada da equipe e deveria ter usado meios mais sutis de introduzir as revisões do código na cultura.

Críticas quase nunca são pessoais em um ambiente de engenharia de software – geralmente são só parte do processo de criar um produto melhor. O truque é garantir que você (e as pessoas ao seu redor) entenda as diferenças entre críticas construtivas sobre o trabalho criativo de alguém e simples ataques contra seu caráter. A segunda é inútil – é mesquinha e quase impossível agir com base nela. A primeira é sempre útil, fornece orientação sobre como melhorar e o mais importante: é *respeitosa*. A pessoa que faz a crítica construtiva se importa realmente com a outra pessoa e quer que ela, pessoalmente,

ou seu trabalho, melhore. Aprenda a respeitar seus colegas e faça críticas construtivas com educação. Se você realmente respeitar alguém, será motivado a escolher palavras polidas e úteis – uma habilidade adquirida com muita prática.

Do outro lado da conversa, você precisa aprender também a aceitar críticas. Isso significa não só ser *humilde* em relação às suas habilidades, mas também *confiar* que a outra pessoa tem seus interesses (e os do seu projeto!) em mente e não acha de verdade que você é um idiota. Programação é uma habilidade como qualquer outra, que melhora com a prática. Se um colega mostrasse como melhorar seu malabarismo, você veria isso como um ataque ao seu caráter e valor como ser humano? Esperamos que não. Do mesmo modo, *sua autoestima não deveria estar ligada ao código que escreve*. Repetindo: você não é seu código. Repita isso de novo. *Você não é seu código*. Você precisa não só acreditar em si mesmo, mas também fazer seus colegas acreditarem.


*Não iguale sua autoestima à qualidade do seu código.*

Por exemplo, se você tiver um colaborador possivelmente inseguro, eis o que não dizer: "Puxa, você errou feio no fluxo

de controle daquele método. Você deveria usar o padrão de código xyzzy como todo mundo". Esse feedback está repleto de antipadrões: você está dizendo a alguém que ele está "errado" (como se o mundo fosse em preto e branco!), exigindo que ele mude alguma coisa e acusando-o de criar algo que vai contra o que todo o mundo está fazendo (fazendo-o se sentir estúpido). A resposta será bastante emocional, vinda de alguém na defensiva.

Um modo melhor de dizer a mesma coisa seria: "Ei, estou confuso com o fluxo de controle nessa seção aqui. Imagino se o padrão de código xyzzy poderia tornar isso mais claro e fácil de manter?". Observe como você está sendo humilde ao fazer a questão ser sobre você e não sobre o colaborador. Ele não está errado. Você está só tendo problemas em entender o código. A sugestão é oferecida simplesmente como um meio de esclarecer a situação a você e talvez ajudar os objetivos de sustentabilidade a longo termo do projeto. Você não está exigindo nada – está dando ao colaborador a capacidade de rejeitar pacificamente a sugestão. A discussão permanece no campo do próprio código e não trata do valor da habilidade de codificação de ninguém.

## Falhe rápido, aprenda, repita

Existe uma conhecida (e clichê) lenda urbana no mundo dos negócios sobre um gerente que comete um erro e perde impressionantes US$ 10 milhões. Abatido, ele vai para o escritório no dia seguinte e começa a empacotar suas coisas, quando recebe o inevitável recado: "O CEO quer ver você no seu escritório". Ele marcha para o escritório do CEO e, silenciosamente, coloca uma folha de papel na mesa dele.

– O que é isso? – pergunta o CEO.

– Minha demissão – diz o executivo. – Suponho que você me chamou para me demitir.

– Demiti-lo? – responde o CEO, incrédulo. – Por que iria demiti-lo? Acabei de gastar US$ 10 milhões para treinar você! [6]

É uma história extrema, é claro, mas o CEO nessa história entende que demitir o executivo não reverterá a perda de US$10 milhões e irá aumentá-la pela perda de um valioso executivo que, você pode ter certeza, não vai cometer esse tipo de erro novamente.

Nós dois trabalhamos no Google e um dos nossos lemas favoritos é: "O fracasso é uma opção". É amplamente reconhecido que se você não falha de vez em quando, não está sendo inovador ou assumindo riscos o suficiente. O fracasso é visto como uma oportunidade de ouro para aprender e melhorar para a rodada seguinte. De fato, muitas vezes se cita Thomas Edison: "Se encontrei 10 mil meios de algo não funcionar, não fracassei. Não fico desmotivado, porque cada tentativa errada descartada é outro passo à frente".

O Google muitas vezes segue o conceito de "não se esconder em uma caverna até que esteja perfeito" (que discutimos anteriormente): assim que algo é vagamente utilizável, é liberado na forma bruta para o público. É disso que se tratava o Google Labs. Torna-se aparente muito rapidamente onde estão os sucessos e fracassos, e então se espera que a equipe de programadores aprenda, repita e libere uma nova versão o mais rápido possível. O lado negativo é que o Google ocasionalmente é ridicularizado por ter ferramentas como o Gmail em "beta" por mais de quatro anos. O lado positivo é a capacidade de manobrar e se adaptar rapidamente, produzindo um produto incrível em um período muito curto de tempo. Tudo o que é necessário é um pouco de humildade – está tudo bem mostrar um software imperfeito para os usuários – e um pouco da confiança dos usuários que realmente valorizam seus esforços e estão ansiosos por ver melhorias rápidas.

6 Uma dúzia de variações desta lenda pode ser encontrada na web, atribuídas a diferentes famosos gerentes.

A chave para aprender com os seus erros é documentar seus fracassos. Escreva post-mortem[7], como eles são muitas vezes chamados no nosso negócio. Tenha um cuidado extra para garantir que o documento post-mortem não é só uma lista inútil de desculpas ou pretextos – esse não é o objetivo. Um post-mortem correto deve sempre conter uma explicação do que foi aprendido e o que vai mudar como resultado da experiência de aprendizado. Então, tenha certeza de colocá-lo em um local fácil de encontrar e execute de verdade as mudanças propostas. Lembre-se de que documentar corretamente os fracassos também torna mais fácil para outras pessoas (presentes e futuras) saber o que aconteceu e evitar repetir a história. Não apague suas pegadas – ilumine-as como guias para aqueles que vierem depois de você!

Um bom post-mortem deve incluir o seguinte:

- um breve resumo;
- uma linha do tempo do evento, desde a descoberta, passando pela investigação, até a resolução;
- a causa primária do evento;
- avaliação do impacto e danos;
- um conjunto de ações para corrigir imediatamente o problema;
- um conjunto de ações para evitar que o evento ocorra novamente;
- lições aprendidas.

## Deixe tempo para o aprendizado

Cindy era uma superestrela – uma engenheira de software que havia realmente dominado sua área de especialização. Ela foi promovida à líder técnica, suas responsabilidades aumentaram e ela se mostrou à altura do desafio. Pouco tempo depois,

7 N.T.: Pós-morte, do latim.

ela se tornou mentora de todos ao seu redor e ensinava-lhes o caminho das pedras. Ela começou a ministrar palestras sobre sua área e logo passou a liderar diversas equipes. Ela adorava ser a "especialista" o tempo todo. Ainda assim, começou a ficar entediada. Em algum ponto do caminho, ela parou de aprender coisas novas. A novidade de ser a especialista com mais conhecimento e experiência começou a se desgastar. Apesar de todos os sinais externos de maestria e sucesso, algo estava faltando. Um dia, ela foi para o trabalho e percebeu que seu campo de atuação escolhido não era mais tão relevante. As pessoas haviam migrado para outros tópicos de interesse. Onde ela havia errado?

Vamos ver: é divertido ser a pessoa com mais conhecimento na sala e tornar-se mentor de outras pessoas pode ser incrivelmente recompensador. O problema é que uma vez que você atinge o topo de sua equipe, você para de aprender. E quando você para de aprender, você fica entediado ou, acidentalmente, torna-se obsoleto. É realmente fácil ficar viciado em ser líder. Mas somente ao abandonar um pouco do ego, você vai mudar de direção e se expor a situações novas. Novamente, trata-se de ser mais humilde e estar disposto a aprender tanto quanto a ensinar. Saia da sua área de conforto de vez em quando. Encontre um aquário com peixes maiores que você e enfrente quaisquer desafios que lhe ofereçam. Você será muito mais feliz a longo prazo!

## Aprenda a ser paciente

Anos atrás, Fitz começou a escrever uma ferramenta para converter repositórios CVS para o Subversion (e, depois, para o Git), e, em razão dos caprichos do RCS e CVS, ele continuou a desenterrar bugs bizarros com arquivos RCS inválidos que o CVS devoraria alegremente. Já que seu velho amigo e colega Karl conhecia intimamente CVS e RCS, ambos decidiram que deveriam trabalhar juntos para corrigir aqueles bugs.

Um problema surgiu quando eles começaram a programar em dupla: Fitz era um engenheiro "de baixo para cima", que estava feliz em mergulhar na lama e cavar sua saída tentando várias coisas diferentes rapidamente, contornando os detalhes. Entretanto, Karl era um engenheiro "de cima para baixo", que queria entender toda a situação e mergulhar na implementação de praticamente cada método na pilha de chamada antes de prosseguir e atacar o bug. Isso resultou em alguns conflitos interpessoais épicos, discórdias e em ocasional discussão. Foi preciso um esforço hercúleo, foco e muito HRT para Fitz e Karl completarem a tarefa. No final, o HRT não só ajudou a salvar o projeto, mas também a amizade deles.

## Esteja aberto à influência

Quanto mais aberto estiver à influência, mais você será capaz de influenciar. Quanto mais vulnerável for, mais forte você parecerá. Essas declarações parecem contradições bizarras. Mas qualquer um pode se lembrar de alguém com quem tenha trabalhado e que seja incrivelmente teimoso. Não importa o quanto as pessoas tentem persuadi-lo, ele bate o pé ainda mais. O que acaba acontecendo com esses membros de equipe? De acordo com nossa experiência, eles acabam sendo "contornados", como um obstáculo que todos têm como garantido. As pessoas param de ouvir suas opiniões ou objeções. Você certamente não quer que isso aconteça a você, então tenha essa ideia em mente: está tudo bem deixar alguém fazer você mudar de ideia. Escolha suas batalhas com cuidado. Lembre-se de que para ser ouvido de verdade, primeiro você precisa aprender a ouvir os outros. No caso de ser influenciado, o ouvir deve ocorrer antes de você firmar uma posição ou declarar com firmeza que decidiu algo – se estiver o tempo todo mudando de ideia, as pessoas vão achar que você é indeciso.

Sobre vulnerabilidade, isso também parece um pouco estranho, a princípio. Se alguém admitisse ser ignorante sobre o tópico em questão ou não soubesse como resolver um problema, que tipo de credibilidade teria em um grupo? Vulnerabilidade é uma exibição de fraqueza e isso destrói a confiança, certo?

Não é verdade. Admitir que você cometeu um erro ou que simplesmente está fora do seu elemento é um modo de melhorar seu status a longo prazo. Na verdade, isso engloba todo o HRT: é uma demonstração de humildade, trata-se de aceitar responsabilidade, é um sinal de que você confia na opinião dos outros, e, em troca, as pessoas acabam respeitando sua honestidade e força. Às vezes, o melhor que você pode fazer é dizer: "Não sei".


*Honestidade e humildade não são kriptonita.*

Considere os políticos profissionais. Eles são notórios por nunca admitirem seus erros ou ignorância, mesmo quando é óbvio que estão errados ou não conhecem um assunto. Por esse motivo, a maioria das pessoas não acredita em uma palavra do que eles dizem. Esse comportamento ocorre principalmente porque os políticos estão sob ataque constante de seus oponentes. Entretanto, quando você estiver escrevendo software, é desnecessário viver em estado constante de defesa – seus colegas são colaboradores, não concorrentes.

## Próximos passos

Se você chegou aqui, está no caminho certo para dominar a arte de trabalhar em equipe. Você precisa começar examinando e meditando sobre seus próprios comportamentos. Uma vez que tenha incorporado essas estratégias à sua vida diária, irá descobrir que a colaboração vai se tornar muito mais natural e sua produtividade como engenheiro vai começar a aumentar perceptivelmente.

As mudanças importantes começam com você e então se espalham para os demais. No próximo capítulo, vamos falar sobre como criar uma cultura de HRT na sua equipe imediata.