Incluindo dicas de top desenvolvedores do Google, Microsoft, Spotify, Amazon e mais

14 hábitos de desenvolvedores altamente produtivos

Zeno Rocha

14 Hábitos de Desenvolvedores Altamente Produtivos

v1.0.0

Data: 28 de Julho, 2020

Site: 14habits.com/br

Email: zeno@14habits.com

Ilustrações por Briza Bueno

Traduções por Caroline Moreschi

Copyright © 2020 por Zeno Rocha

ISBN: 978-1-7352665-1-0

14 Hábitos de Desenvolvedores Altamente Produtivos

Parte um: Princípios

Olá mundo

“Não há elevador para o sucesso, você precisa subir as escadas.” - Zig Ziglar

Comecei a desenvolver software há mais de dez anos. Desde então, construí muitos aplicativos, criei dezenas de projetos de código aberto e escrevi milhares de commits. Além disso, palestrei em mais de uma centena de conferências e tive a oportunidade de conversar com vários desenvolvedores ao longo do caminho.

Tive a sorte de estar em contato com alguns dos melhores engenheiros de software da indústria, mas também conheci muitos programadores que ainda fazem a mesma coisa há anos.

O que separa um grupo do outro? O que há de único nas pessoas que trabalham nas maiores empresas da indústria? O que há de especial nas pessoas que criam os aplicativos mais usados no mundo? Como alguns desenvolvedores podem ser tão prolíficos no trabalho e também fora de seus empregos?

Essas perguntas ficaram na minha cabeça por um longo tempo. Percebi que podia comprar os melhores teclados mecânicos, participar das conferências de tecnologia mais famosas do mundo e aprender todos os novos frameworks. Ainda assim, se eu cultivasse maus hábitos, seria impossível me tornar um desenvolvedor de primeira linha. Por isso, decidi procurar os melhores desenvolvedores que conheço e pedir dicas de como ser mais produtivo.

Este livro não oferece um caminho reto ou fórmula predefinida para o sucesso. Este livro é o resultado de uma busca. Uma busca para descobrir quais hábitos podem ser cultivados para se tornar um melhor engenheiro de software.

Metodologia

Este não é um livro convencional. Você não encontrará o mesmo formato ou estrutura que um livro comum possui. Na verdade, este livro foi projetado para ser o mais simples e objetivo possível. Você pode seguir a ordem dos capítulos ou lê-los individualmente. Tudo é independente e não requere conhecimentos anteriores.

No final de cada hábito, você encontrará uma seção marcada como **“Perguntas e Respostas”**, onde entrevisto desenvolvedores seniores e líderes de tecnologia de várias empresas para entender como eles chegaram lá. Fui atrás de gigantes da tecnologia como Google, Amazon, Microsoft e Adobe. Startups poderosas como GitHub, Spotify, Elastic, Segment, GoDaddy e Shopify. E até organizações estabelecidas, como Citibank, BlackBerry e The New York Times.

Essas pessoas vêm de todo o mundo e têm um histórico bastante diversificado. De São Francisco a Nova York. De São Paulo a Montreal. De Londres a Estocolmo. A ideia é apresentar a você não o ponto de vista de apenas uma pessoa, mas uma coleção de ideias sobre como navegar em sua carreira.

Você também encontrará seções marcadas como **“TODO”**, onde recomendo que você reflita sobre determinados tópicos e tome medidas com instruções específicas. Eu recomendo levar alguns minutos para mergulhar neles, pois isso gerará ainda mais conhecimento.

E, finalmente, você pode encontrar algumas seções marcadas como **“Bônus”**. Esses são conteúdos extras que eu preparei para você. Eles podem ser encontrados fora do livro e complementam o que você está lendo.

Por quê hábitos?

Se você já tentou perder peso, sabe como esse processo é frustrante. Você pode se exercitar o máximo que puder por três horas, mas se fizer isso apenas uma vez por semana, o efeito será nulo. O que realmente gera resultados é quando você se exercita várias vezes por semana. Então, alguns meses depois, você começará a perceber mudanças no seu corpo.

Consistência é importante e esse mesmo conceito também se aplica à sua carreira profissional. As coisas levam tempo, e a intensidade nem sempre é a resposta. Os hábitos que você decide cultivar (ou não cultivar) determinarão suas futuras oportunidades de vida. Assim como foi dito no livro *Atomic Habits*:

"Os hábitos são os juros compostos do auto-aperfeiçoamento. Da mesma forma que o dinheiro se multiplica através dos juros compostos, os efeitos de seus hábitos se multiplicam à medida que você os repete. Eles parecem fazer pouca diferença em um determinado dia e, no entanto, o impacto que eles proporcionam ao longo dos meses e dos anos podem ser enormes. É apenas quando se olha para trás dois, cinco ou talvez dez anos é que o valor dos bons hábitos e o custo dos maus se tornam surpreendentemente aparentes."

— *James Clear*

Eu recebo muitos e-mails de outros engenheiros de software perguntando qual linguagem de programação eles deveriam aprender. E embora essa seja uma pergunta muito importante, sinto que a pergunta mais valiosa seria: "Que hábitos eu preciso cultivar para ser eficaz em qualquer linguagem de programação?"

É por isso que decidi focar este livro em hábitos, e não em táticas.

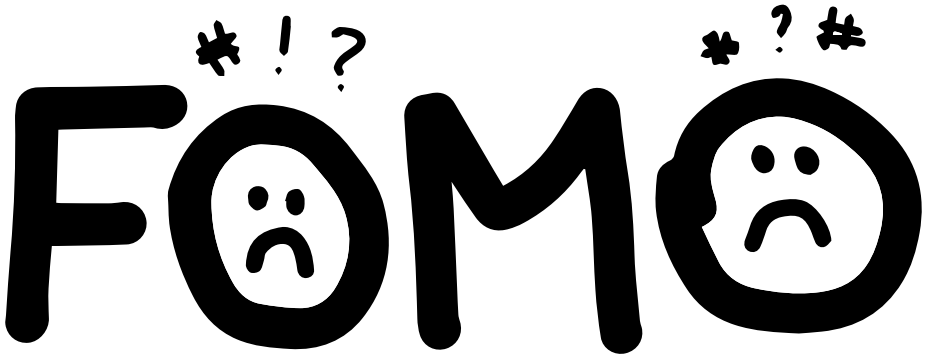
Agora vamos nessa! Você está pronto?

Parte dois: Hábitos de aprendizagem

Hábito 1: Procure os sinais

“As palavras mais antigas e mais curtas - sim e não - são aquelas que requerem mais reflexão.”- Pitágoras

Todos os dias somos bombardeados com tweets, e-mails e vídeos nos dizendo o que devemos fazer, o que devemos aprender e o que devemos focar. Estamos constantemente diante do FOMO *(the fear of missing out)* ou “o medo de estar perdendo algo”. E se estivermos perdendo nosso tempo com uma linguagem de programação errada? E se o framework mais produtivo não for o que você está usando atualmente?



Para mim, tudo começou com a escolha do sistema operacional. Na época, eu tinha uma máquina Windows, mas todo mundo me dizia que os computadores da Apple eram melhores e que eu deveria trocar. No entanto, os preços das máquinas da Apple estavam muito longe da minha realidade de compra, então conseguir um nem era uma possibilidade. Depois de alguns anos, um dos meus chefes me deu a opção de escolher entre Windows e MacOS. Escolhi o MacOS para ver o que tinha de tão incrível nele. Depois de algum tempo usando MacOS, o Ubuntu começou a se tornar muito popular e todo mundo me dizia que eu deveria mudar para o Ubuntu. Então pensei em experimentar e comecei a usá-lo. Meu ponto aqui

não é dizer qual você deve escolher, o ponto aqui é que **não existe a melhor ferramenta**.

Em vez disso, devemos praticar o JOMO *(the joy of missing out)* ou “a alegria de estar perdendo algo”, que é principalmente sobre ser feliz e se contentar com o que você já sabe. Dê a si mesmo algum crédito e reflita sobre tudo o que aprendeu até agora. Obviamente, você não deve ser complacente e parar de estudar novas tecnologias. Seria melhor se você encontrasse algum equilíbrio entre praticar suas habilidades existentes e aprender novas.

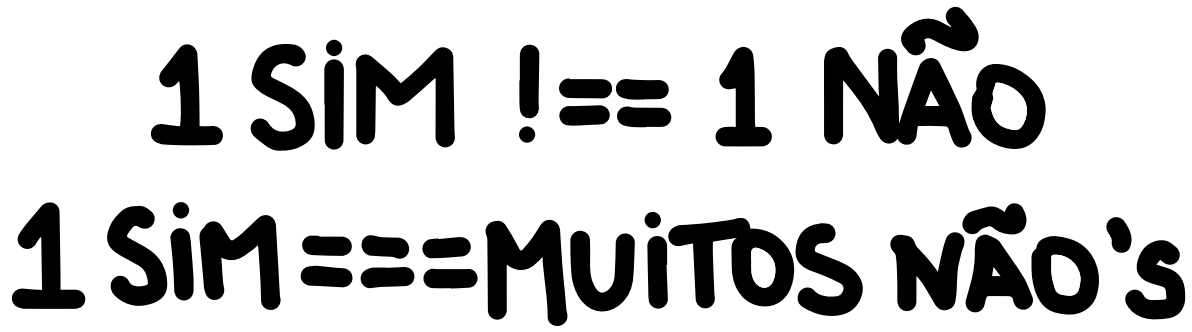
As pessoas tentarão convencê-lo de qual é o melhor sistema operacional, a melhor linguagem de programação, o melhor framework. Eles vão falar sobre todas as coisas incríveis que a ferramenta deles faz e que a sua ferramenta não faz. A realidade, porém, é que cada ferramenta é diferente e também somos diferentes como usuários. O que é melhor para você, pode não ser o melhor para mim ou para os outros.

Pense nisso como uma estação de rádio que você está tentando sintonizar. Eu sei que você provavelmente não usa mais rádios, mas fique comigo. Imagine que você gira o dial e está apenas captando ruído estático, mas depois de alguns segundos frustrantes, ele finalmente consegue captar um sinal e sintonizar uma estação. O sinal é a informação significativa na qual você realmente está interessado. O ruído estático é apenas a variação aleatória e indesejada que interfere no sinal. É por isso que a

autoconsciência é tão importante, você precisa ser capaz de identificar qual é o sinal e o que é apenas ruído.

É crucial entender que o barulho sempre estará lá. Você não precisa necessariamente abandonar as mídias sociais, cancelar as listas de e-mail e parar de assistir vídeos do YouTube. Uma desintoxicação digital pode definitivamente ajudar por um tempo, no entanto, não é uma solução a longo prazo. O que você precisa fazer é escolher o que é relevante para você neste momento da sua carreira.

Aceite o fato de que você simplesmente não consegue aprender tudo. **Lembre-se, desejos são infinitos, necessidades são limitadas**. Aceite o fato de que o novo nem sempre é o melhor. Existem pessoas que trabalham com linguagens de programação arcaicas e ainda ganham muito dinheiro. Pratique diariamente a arte sutil de dizer ‘não’. Não à nova biblioteca. Não àquela plataforma mais sofisticada. Diga mais ‘não’ para poder dizer ‘sim’ ao que realmente importa para você.



Ouça o barulho, mas preste atenção apenas aos sinais.

```
// TODO

Crie uma lista de todas as tecnologias e ferramentas que você
gostaria de aprender. Agora marque cada uma delas com uma
prioridade diferente: “Esta semana”, “Próximo mês”, “Próximo
ano”. Sempre que sentir que está perdendo alguma nova
tendência brilhante, revise a lista e reorganize a
prioridade.
```

 # Perguntas e Respostas: Como você decide qual tecnologia aprender e investir tempo?

Daniel Buchner (Microsoft):

> "No início da minha carreira, lembro de prestar muita atenção a todos os novos frameworks que apareciam na primeira página do *Hacker News*. Eu lia sobre qualquer novidade que eles adotassem e passava horas extras mexendo com o framework ou a biblioteca para ter uma noção do que era. É perfeitamente tranquilo se você tiver tempo livre suficiente para explorar e aprender, mas muitas vezes pode levar a fadiga, porque a lista de coisas novas não tem fim.
>
> Atualmente, tento me concentrar em algumas coisas importantes que conduzem minhas avaliações:
>
> 1. Quais são os requisitos técnicos indispensáveis para qualquer coisa em que estou trabalhando? Isso pode incluir itens como desempenho, UX desejado ou integração com outros sistemas.
> 2. Quão fácil será para os outros trabalharem no que estou construindo e como será para eles integrarem em quaisquer projetos em que estejam trabalhando?
> 3. O que estou fazendo está alinhado com a trajetória da web aberta, dos padrões e das especificações que acredito que permanecerão a longo prazo?

> Os três pontos acima tendem a eliminar muitas das bibliotecas, frameworks e outros utilitários que encontro. Isso me poupou tempo e permitiu que eu me concentrasse mais em entregar o que precisava, em vez de ser pego no turismo de desenvolvimento."

Addy Osmani (Google):

> "Aceite que você não pode aprender tudo, mas pode aprender o suficiente para ser produtivo. Quando noto que uma ideia, framework ou tecnologia está ganhando força, invisto uma tarde estudando para ter uma ideia de duas coisas: ‘Isso melhora minha produtividade?’ e ‘Isso melhora a experiência do usuário dos tipos de projetos que costumo criar?’ Se a resposta para qualquer uma dessas perguntas for afirmativa, eu considero gastar mais tempo aprendendo sobre o framework ou a tecnologia com mais profundidade.
>
> Como nosso tempo é limitado, há muitas tecnologias que irei achar convincentes o suficiente para ficar de olho, mas devo pensar na troca sobre aprender e investir em favor de outra coisa. Eu acho que isso é saudável. Originalmente, eu tentei (e usei) o React no primeiro ano em que foi lançado, mas agora o uso regularmente. Gosto de muitas ideias do Svelte, mas como eu já havia aprendido o React, o Preact e o Vue, decidi que era um investimento melhor de tempo subir de nível em áreas completamente diferentes com esse tempo, como a API de animações da Web.
>
> Como disse, com o tempo limitado, é bom encontrar um equilíbrio no que você escolhe aprender para manter-se produtivo. Quando você escolhe apenas algumas opções de alto impacto, deve reservar um tempo para se aprofundar na tecnologia que você escolheu. Isso pode ser imensamente útil quando você está tentando criar algo não trivial no mundo real."

Loiane Groner (Citibank):

> "Posso responder essa pergunta depois de me fazer mais algumas perguntas. Que problema a tecnologia resolve? Nós, desenvolvedores, somos pagos para resolver problemas, não somos pagos para

necessariamente escrever código. Escrever código é uma das consequências de resolver o problema do cliente com a tecnologia. Então, que problema essa nova tecnologia ou framework está tentando resolver? Como se diferencia dos frameworks/tecnologias já existentes? Preciso aprender algo novo para poder aprendê-la? Posso tirar proveito das minhas habilidades e conhecimentos existentes (o framework usa uma linguagem que eu já conheço)? E a infraestrutura? Existem provedores de nuvem que suportam a nova tecnologia ou a minha infraestrutura existente pode suportá-la? Essas são apenas algumas das perguntas que eu me perguntaria antes de aprender uma nova tecnologia.

Além das perguntas, você deve sempre aprender uma visão geral da tecnologia para saber do que se trata. Na minha humilde opinião, não existe conhecimento em excesso. Talvez você tenha um cenário na empresa em que trabalha ou um projeto pessoal que esteja tentando colocar em prática que possa usar essa nova tecnologia.

Por fim, fique de olho nas empresas que estão adotando a tecnologia e colocando projetos em produção. Esta é a parte mais crítica. Todos nós podemos codificar POCs (prova de conceitos), mas para realmente aprender todos os cenários e também o que será necessário para manter um projeto, é crucial enviá-lo à produção. Se muitas empresas estão adotando a tecnologia, talvez valha a pena mergulhar fundo e tentar aprender conceitos mais avançados sobre ela."

Netto Farah (Segment):

"Eu diria que isso é algo com o qual ainda luto hoje em dia, mas acho que minha experiência e a minha intuição têm sido boas bússolas que ajudam a me guiar em cenários como este.

Eu tento padronizar os novos frameworks com as experiências anteriores. React, por exemplo, parece bastante semelhante ao Flex em alguns aspectos, o que foi uma ideia bem testada e comprovada. O Next.js parece um pouco com o Rails no que diz respeito ao DX (developer experience), então eu também considero isso um bom sinal.

Algumas mudanças de paradigma mais drásticas, no entanto, são um pouco mais interessantes e mais difíceis de julgar apenas de relance. Para esses, costumo experimentar um pouco a tecnologia antes de investir muito tempo nelas.

Outro ponto a considerar é o quão transferível o conhecimento adquirido pode ser. Por exemplo, aprendendo Rust vs. Go: Duas ferramentas bem distintas que estão competindo no mesmo espaço de problemas. O Rust parece muito mais legal e sofisticado, mas o Go é muito provavelmente a linguagem que o ajudará a conseguir um bom emprego. Nesse caso, eu pessoalmente otimizaria aquela que pode me ajudar a alcançar o próximo passo na minha carreira."

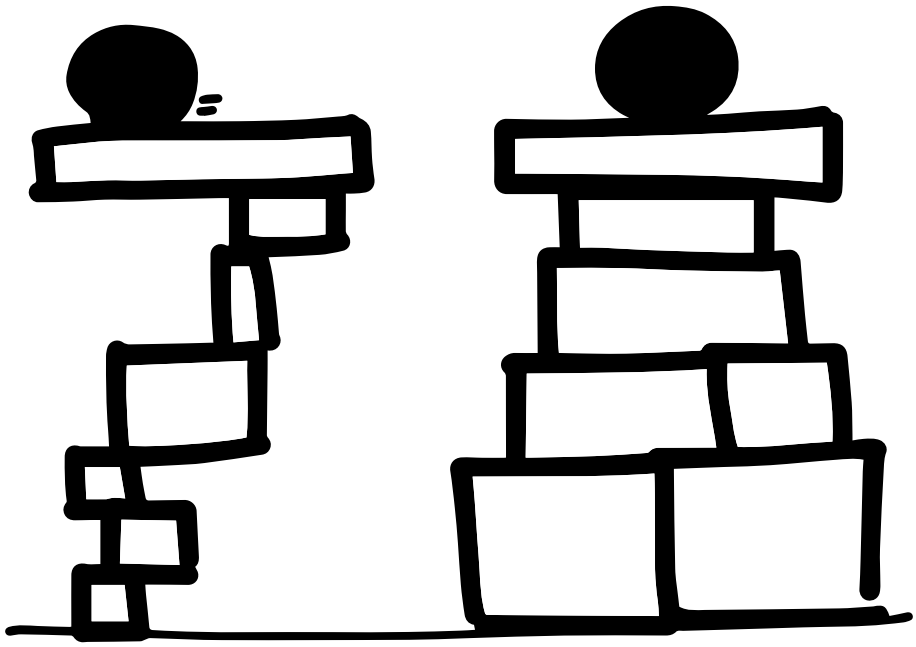
Hábito 2: Foco nos fundamentos

“Você pode praticar arremessos oito horas por dia, mas se sua técnica estiver errada, tudo o que você se tornará será muito bom em arremessar da maneira errada. Pratique os fundamentos e o nível de tudo o que você fizer vai subir.” - Michael Jordan

Em março de 2009, eu estava estudando na universidade e tive um professor, Mariano Pimentel, que nos forçou a escrever HTML/CSS no bloco de notas. De todas as aplicações que você pode escolher para escrever código, o bloco de notas é um dos piores. Naquela época, o Dreamweaver era a criança popular da escola. Ele tinha uma interface gráfica amigável, verificação de sintaxe em tempo real e sugestões de código. Por outro lado, no bloco de notas, não havia realce de sintaxe, não havia preenchimento automático, não havia visualização ao vivo, não havia absolutamente nada para me ajudar a ser mais produtivo.

Todo aquele semestre teve um grande impacto em mim. Mesmo mais de dez anos depois, ainda me lembro de todas as tags HTML e propriedades CSS, porque tive que escrever elas manualmente sem ajuda. Se eu tivesse esquecido apenas um caractere, a coisa toda não funcionaria, então eu precisava ter certeza de que entendia completamente o que estava escrevendo.

Não estou defendendo que todos devemos seguir o mesmo caminho e começar a abandonar nossos amados editores de código. A ideia crucial é que aprender os fundamentos vai te preparar para o futuro.



Se você quer se tornar um pintor, é essencial que você aprenda alguns fundamentos importantes. Entender *teoria das cores* vai te ajudar a representar diferentes significados, composições e harmonia. Entender *perspectiva* vai te ajudar a posicionar sombras e iluminação. Entender *formas* ajudará você a fazer os objetos parecerem tridimensionais, dando a eles a ilusão de volume. Entender *anatomia* ajudará você a desenhar personagens com a proporção e estrutura corporais corretas. Você não precisa necessariamente de todo esse conhecimento para fazer sua primeira pintura, mas tenho certeza de que você não pode se tornar um grande pintor profissional sem conhecer esses tópicos.

O mesmo se aplica a você. Se você decidir se tornar um ótimo desenvolvedor, é importante entender os principais conceitos, como algoritmos, lógica, redes, acessibilidade, segurança e experiência do usuário. Você não precisa necessariamente deles para criar seu primeiro aplicativo, mas conhecê-los vai te ajudar a criar os próximos dez aplicativos complexos que você criará no futuro.

/* Bônus: A comunidade de código aberto criou um roteiro de fundamentos para desenvolvedores DevOps, Front-end e Back-end. Acesse 14habits.com/br/bonus/2 para ver. */

```
// TODO

Gaste algum tempo pesquisando quais são os conceitos
fundamentais em seu campo. Pegue um pedaço de papel e divida-
o em duas colunas. No lado esquerdo, liste todo o
conhecimento que você já possui. No lado direito, liste todo
o conhecimento que você ainda precisa adquirir. Planeje um
horário dedicado do seu dia para estudar esses conceitos.
```

 # Perguntas e Respostas: Como você abordaria a aprendizagem de uma nova linguagem de programação hoje?

Lais Andrade (Google):

> "Costumo abordar novas linguagens da mesma maneira que abordaria a aprendizagem de qualquer novo framework: tento ter um entendimento básico dos principais conceitos e procurar exemplos de como outras pessoas resolvem problemas semelhantes. Isso geralmente abrange boas práticas, como você pode ver padrões de como certos frameworks são usados e, a partir daí, é fácil encontrar documentação sobre o raciocínio por trás dessas opções.
>
> Aprender uma nova linguagem geralmente me ocorre com a necessidade de aplicá-la imediatamente em um projeto. Como eu não sou conhecedora de linguagens de programação, tenho a tendência de entrar em contato com novas à medida que eu preciso delas no meu dia-a-dia. Ter revisores que estão familiarizados com a linguagem e a base de código da equipe também é muito importante para aprender mais rápido coisas novas.

> E a coisa mais importante que tento fazer é sempre entender tudo o que escrevo. Sempre que me vejo confusa entre conceitos semelhantes, ou sempre que alguém sugere um novo recurso que nunca vi antes, tento ler a documentação e entender a teoria por trás deles. Eu preciso ter confiança sobre por que essa é a melhor escolha possível."

Fabio Costa (GoDaddy, Ex-Facebook):

> “Eu aprenderia uma nova linguagem de programação escolhendo um dos meus projetos pessoais existentes e reescrevendo-os usando essa nova linguagem de programação. Na minha opinião, essa é a maneira mais eficiente de aprender uma nova linguagem, porque posso focar 100% apenas em aprender a linguagem e nada mais. Escolher um projeto completamente novo, por exemplo, pode exigir pensar em novas funcionalidades, documentação e outras coisas que tirariam meu foco de aprender a linguagem”.

Michael Lancaster (BlackBerry):

> "Como eu não tenho um diploma formal de ciência da computação, foi muito importante, desde o início, encontrar a maneira mais favorável para eu aprender novas tecnologias enquanto me adapto a elas.
>
> Tudo começa com o ABC (Always Be Coding) ou “Sempre Continue Programando”, por mais clichê que pareça, a prática leva à perfeição. Por esse motivo, ao aprender uma nova linguagem de programação, me comprometo a trabalhar nela pelo menos 1 hora por dia durante cerca de 30 dias consecutivos. Eu abro o YouTube e escolho tutoriais para criar projetos, mas escolho autores que se aprofundam mais nos fundamentos.
>
> Enquanto acompanho os tutoriais, sempre faço referência à documentação oficial, testes de unidade e projetos populares de código aberto para ajudar a estruturar minha base e hábitos o mais próximo possível daqueles já experientes na linguagem em particular."

Hábito 3: Ensinar é igual aprender

“Poder não vem do conhecimento adquirido, mas do conhecimento compartilhado.” — Bill Gates

No ano passado, fiz minha 100ª apresentação em uma conferência de desenvolvedores. Desde 2011, tive a oportunidade de visitar 16 países e 71 cidades para palestrar. Voei 1.146 kms, o que equivale a 28,6 vezes ao redor da Terra ou 3 viagens à Lua.

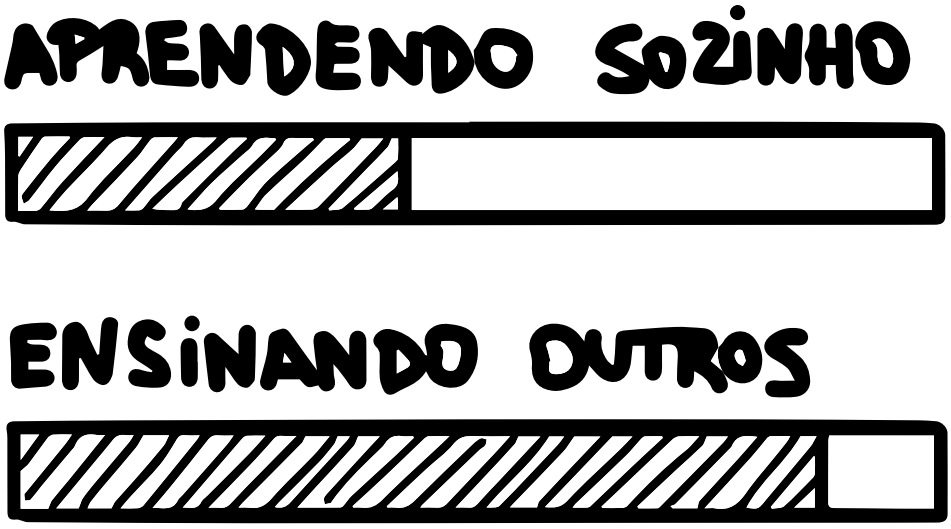
Parecem números impressionantes. Quando você os lê, pode parecer que já sei tudo. A verdade é que não. Ainda sinto muito medo antes de subir ao palco. *Toda. Santa. Vez.*

Sou introvertido. Quando era criança, minha mãe costumava dizer que eu andava pela rua olhando para o chão, porque eu era tão tímido que não conseguia olhar para o rosto das pessoas. Na escola, nunca quis apresentar nada na frente da classe. Eu era extremamente tímido. Eu gostaria de poder dizer que tudo isso foi passado, mas não é. Mesmo depois de fazer mais de cem apresentações, ainda me sinto inseguro.

Então, por que continuo me colocando nessa posição? Por que me submeter a uma palestra? Por que gravar um vídeo? Por que escrever uma postagem no blog? A resposta pode ser encontrada em uma frase simples que meu pai costumava me dizer:

“Se você realmente quer aprender algo, precisa ensiná-lo”.

O processo de digerir o conteúdo para alguém é o que faz você realmente entender. O processo de escrever como fazer algo é o que faz você memorizá-lo. O processo de ensinar é o que faz você realmente aprender.

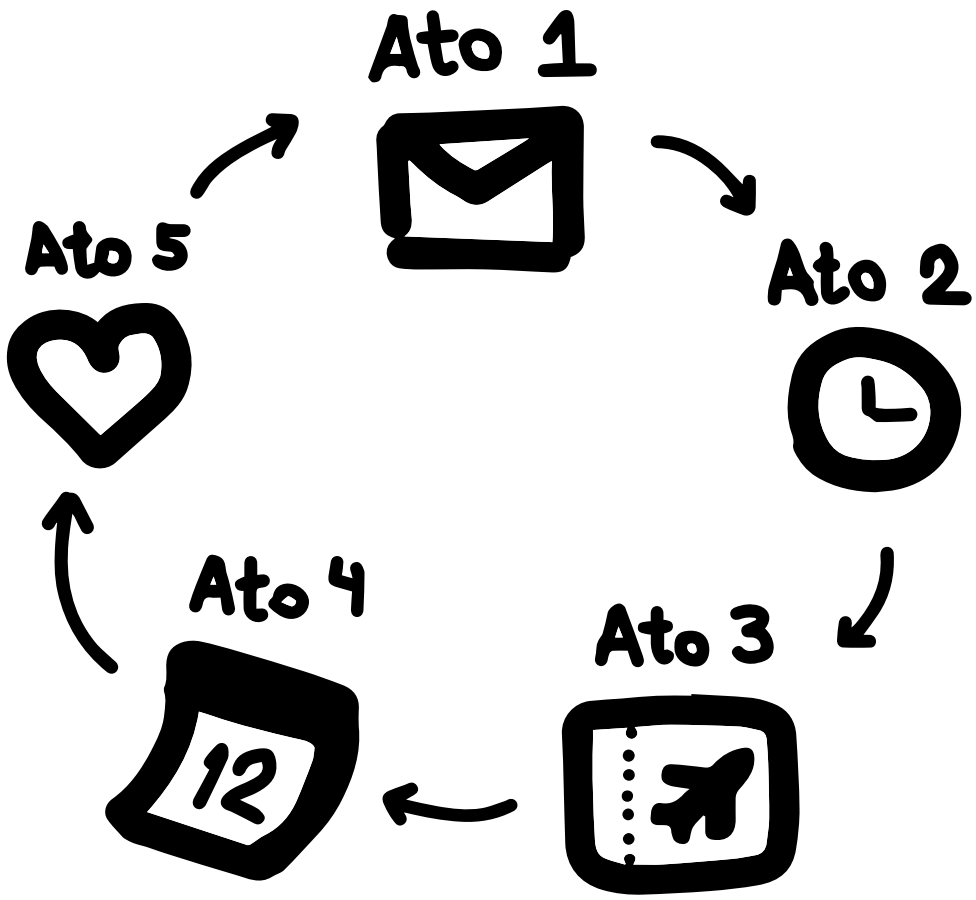


O problema é que você deve fazer tudo isso em público. Não existe escapatória; para que isso funcione, você precisa de um ser humano ouvindo do outro lado. Se você decidir fazer isso, deixa eu te dizer uma coisa: não vai ser fácil e você vai querer desistir muitas vezes.

Existem muitas maneiras de ensinar outras pessoas. Você pode começar compartilhando links nas mídias sociais, escrevendo postagens no blog e gravando transmissões de tela. Meu favorito ainda é fazer palestras.

Segundo muitos estudos, o medo número um da maioria das pessoas é falar em público. O número dois é a morte. É isso mesmo, a morte é o número dois. Você provavelmente já viu e ouviu essa frase milhares de vezes. Aposto que está escrito em todos os treinamentos de oratória do mundo.

Quando olho para todas as apresentações que dei, consigo identificar um padrão entre todas essas experiências. Eu posso até resumir em cinco atos.



Ato 1 – O convite: Tudo começa com um e-mail aleatório. Você está vivendo sua vida normal e, de repente, chega uma notificação. É de um organizador de eventos que está animado com o trabalho que você faz. Ele pede que você viaje para uma cidade diferente - talvez até para um país diferente - para compartilhar o conhecimento que possui. Você aceita o convite e segue sua vida.

Ato 2 – Cai a ficha: Algumas semanas se passam e um dia você olha o seu calendário e de repente percebe a rapidez com que a conferência está se aproximando. Você entra em pânico quando percebe que ainda não começou a trabalhar nos slides. Você sabe que precisa parar tudo para começar a trabalhar nisso, mas sempre tem algo mais importante para fazer.

Ato 3 – Surtando: De repente, é a semana da conferência. Você ainda não conseguiu colocar suas ideias no papel. Você começa a se arrepender. Você

se pergunta: *“Por que eu aceitei esse convite? Que desculpa posso dar? Sou apenas um desenvolvedor, não sou um palestrante!”* Ainda assim, você se comprometeu a ir. Agora as passagens de avião já foram compradas, a reserva do hotel foi feita, eles já anunciaram seu nome e você não pode voltar atrás. Então, o que você faz? De alguma forma, você encontra tempo, de alguma forma encontra motivação para preparar sua apresentação.

Ato 4 – A hora da verdade: Chegou o dia da sua palestra. Você viaja para o local e a ansiedade começa a tomar conta. Você está prestes a enfrentar centenas (ou talvez milhares) de pessoas. Você está com medo. Você se sente uma fraude. Você sabe que existem pessoas muito melhores por aí. Mais uma vez você sabe que não pode recuar. Então, o que você faz? Você apenas enfrenta a situação e tenta fazer o seu melhor.

Ato 5 – Vamos fazer de novo: A apresentação termina. Sua adrenalina está extremamente alta. Você está super aliviado(a) que finalmente acabou. Algumas pessoas te procuram nos corredores da conferência. Elas perguntam sobre algumas ideias que você compartilhou. Algumas perguntas realmente não fazem sentido. Outras perguntas são extremamente interessantes. Você se sente incrível. Você sente que causou impacto na vida das pessoas. Alguns meses depois, você submete uma palestra para outro evento e tudo começa de novo.

Isso pode parecer loucura, mas se você já fez uma apresentação, tenho certeza de que pode identificar os mesmos comportamentos e padrões. Ainda assim, nos submetemos a essa provação. E talvez a razão disso seja porque, no final das contas, a pessoa que tira o máximo proveito dessas experiências de ensinar não é alguém na multidão, somos nós!

```
// TODO

Encontre um evento online e submeta uma apresentação. Abra um
software de compartilhamento de tela e registre-se fazendo
alguma coisa. Crie um blog e compartilhe um artigo. Escolha
qualquer tópico que você deseja aprender e tente ensiná-lo.
```

 # Perguntas e Respostas: Você já ensinou alguma coisa em público? Como isso afetou sua vida?

Fernando Tadashi (Adobe):

> “Não tenho o hábito de palestrar para grandes públicos, mas como trabalho na área de consultoria, tenho um público que sempre exige o máximo dos meus conhecimentos técnicos. Reuniões e apresentações com os clientes me ajudam não apenas no relacionamento interpessoal, mas também no autocontrole sobre várias situações, principalmente aquelas que envolvem um estresse mais significativo.”

Loiane Groner (Citibank):

> "Faz pouco mais de dez anos que decidi retribuir à comunidade. Apresentei palestras, escrevi alguns livros, publiquei alguns vídeos no Youtube e também fui convidada a dar uma aula para um curso de extensão na universidade em que me formei. Foi uma experiência fantástica e me ensinou muito.
>
> Uma maneira de conhecer verdadeiramente um conceito ou tecnologia é tentar criar um projeto prático. Outro método muito eficaz é quando você tenta explicá-lo a outra pessoa. Isso permite que você perceba as lacunas que você tem, assim você mergulha e estuda mais sobre o tópico que está tentando ensinar. Em outras palavras, ensinar é uma ótima maneira de aprender sobre um tópico específico.

Ensinar e palestrar publicamente transformaram minha carreira. Tive a oportunidade de viajar para novos lugares, conhecer novas pessoas, fazer novos amigos e conversar com pessoas que nunca imaginei que teria a chance. É importante observar que você não precisa ser um especialista em um tópico para apresentar uma palestra, você pode apresentar sobre as tecnologias que está aprendendo internamente para seus colegas de trabalho, em reuniões ou até publicar um vídeo sobre isso."

Addy Osmani (Google):

“Sim! Ensinar os outros muda você para melhor. Obriga você a dar um passo crítico para trás e perguntar: ’Eu realmente entendo esse tópico tão bem quanto acho que eu entendo? Eu poderia explicar isso para um iniciante em um minuto?”. Às vezes, quando você precisa explicar algo para outra pessoa, enfrenta o desafio de simplificar até a ideia mais central. Essa pode ser uma excelente oportunidade de forçar a melhoria do seu conhecimento. Também é uma grande habilidade para se ter na indústria de tecnologia, porque você muitas vezes transmitirá ideias para pessoas com funções diferentes ou que tenham um contexto diferente do seu.

Ensinar outras pessoas também ajuda a questionar até que ponto você deseja entender um tópico. Você pode ter amplitude em um campo, mas você tem profundidade nos tópicos em que está interessado? Faz sentido ir mais fundo? Pegue a engenharia de front-end - há muita coisa acontecendo - novas APIs da Web, JavaScript, CSS, como os navegadores funcionam, como o hardware funciona, como os frameworks funcionam etc. Mas seu nível de entendimento de cada tópico provavelmente varia. Talvez eu entenda JavaScript, mas minha profundidade não corresponderá à de um engenheiro de JavaScript baixo nível. Tudo bem. Os tópicos que quero ensinar podem não exigir que eu faça isso profundamente, mas ir além do entendimento superficial é ótimo porque ajuda você a raciocinar e tomar decisões mais adequadas.

Em última análise, sinto que uma boa maneira de medir se você é capaz de ensinar um tópico é: você entende dele bem o suficiente para

ser eficaz? Se o tópico for mostrar, uma ferramenta - você sabe quando não deve usá-la? Como você pode perceber, a maior coisa que ensinar me abriu é o valor de questionar até que ponto você sabe o que sabe e de equipá-lo com a capacidade de decidir se deseja saber mais."

Manuel De La Peña (Elastic):

"Eu ensino em público há cerca de 15 anos. Mesmo antes de me tornar um desenvolvedor profissionalmente, eu trabalhava como professor em uma escola. Nunca pensei sobre isso, mas ao escrever essas linhas, posso ver como sou mais confiante ao falar em público hoje em dia, graças a essa experiência.

Ao dar uma palestra, você deve ler, talvez codificar alguns exemplos, preparar os slides para compartilhar com o público e condensar todas as informações para torná-las compreensíveis. Durante esse processo, você está aprendendo sobre o tópico que irá apresentar. Porque dar uma palestra não significa que você é a pessoa mais experiente do mundo em algo. Para mim, isso significa que você deseja compartilhar um conhecimento com outras pessoas."

Parte três: Hábitos do dia-a-dia

Hábito 4: Seja sem graça

“Tudo o que precisa ser dito já foi dito. Mas, como ninguém estava ouvindo, tudo deve ser dito novamente.” - Austin Kleon

No mundo do software, a intensidade é sempre elogiada. Quantas vezes você ouviu alguém dizer *“Passei a noite toda programando”* e todo mundo ao redor ficou impressionado? Admiramos aqueles que se esforçam para fazer algo. Eu perdi a conta de quantas vezes dormi no escritório, mas sempre me lembro de me sentir como um super-herói depois de fazer isso. Por outro lado, quantas vezes você já ouviu alguém dizer "*Não aguento mais, estou esgotado"*? Há uma linha tênue entre intensidade e *burnout* (esgotamento profissional). Muitos de nós já cruzamos essa linha e é muito difícil voltar.

Não me leve a mal. Entendo perfeitamente quando as pessoas precisam trabalhar horas extras para cumprir um prazo muito específico ou resolver uma emergência que está afetando os usuários na produção. No entanto, quando esse tipo de comportamento se torna normal, você deve parar e prestar atenção ao que realmente está acontecendo. A intensidade é uma ferramenta que você pode usar em ocasiões especiais, mas se esse for o seu *modus operandi*, você não vai conseguir sustentar por muito tempo. Pense em quem você preferiria ter como colega de trabalho.

Quem é um designer melhor? Aquele que está constantemente tentando desenhar soluções que vão ajudar o usuário? Ou aquele que só cria coisas quando quer?

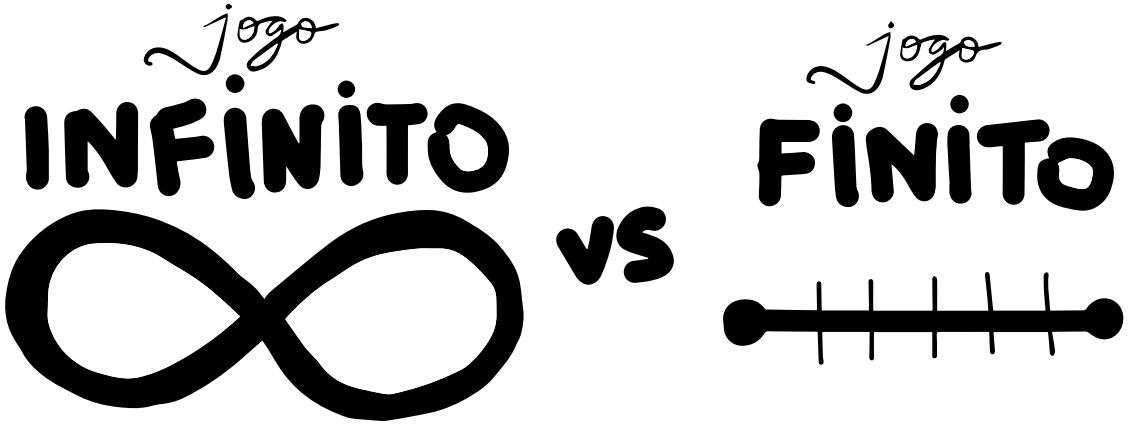
Quem é um gerente de projetos melhor? Aquele que está constantemente fazendo acompanhamento com a equipe e outras partes interessadas? Ou aquele que só fala com você quando o cliente está pressionando?

Quem é um gerente do produto melhor? Aquele que planeja, faz anotações e prioriza tarefas de maneira consistente? Ou aquele que só trabalha quando está motivado?

E, finalmente, quem é um programador melhor? Aquele que está constantemente tentando se tornar uma versão melhor de si mesmo? Ou aquele que se preocupa apenas em testar seu código, melhorar o desempenho e documentar sua solução quando é obrigado a fazer isso?

Quebrar as regras. Fazer o que quiser. Esses conceitos são todos admirados por nós. No fundo, todos nós queremos ser rebeldes. Disciplina. Consistência. Persistência. Esses conceitos não são nada sexy, mas são a chave para jogar jogos infinitos.

Em 1986, o filósofo James Carse introduziu o conceito de dois tipos de jogos em seu livro *Jogos Finitos e Infinitos: Uma Visão da Vida como Jogo e Possibilidade*.



Um jogo finito é definido como tendo jogadores conhecidos, regras fixas e objetivos preestabelecidos. Por exemplo, o futebol é um jogo finito. Onze pessoas de um time jogam contra onze pessoas de outro time. Você tem que usar os pés ou a cabeça para marcar um gol. E após 90 minutos de jogo, o time que marcou mais gols é o vencedor. Esses são os jogadores, regras e objetivos. Você não pode ter um time jogando com doze jogadores, não pode marcar gol com as mãos (a menos que seu nome seja Maradona) e não pode mudar o objetivo para ser o time vencedor aquele que fizer menos gols.

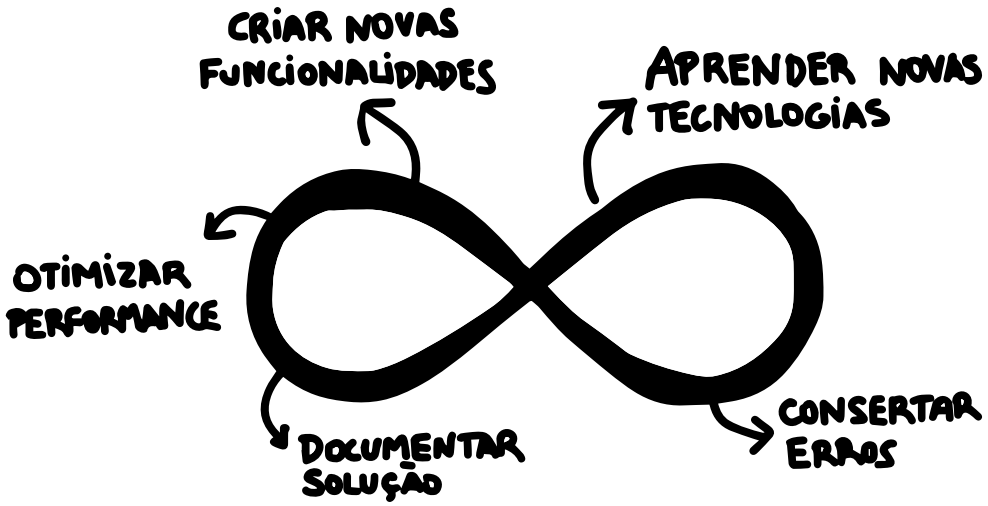
Um jogo infinito, por outro lado, é definido como aquele em que existem jogadores conhecidos e desconhecidos, as regras são mutáveis e o objetivo

não é vencer, é continuar jogando. Como não há vencedores e perdedores, o que acontece é que os jogadores abandonam o jogo quando ficam sem vontade ou sem recursos para jogar, o que os leva a serem substituídos por outros jogadores. Por exemplo, o casamento é um jogo infinito. Sempre há pelo menos dois jogadores envolvidos, apesar da família e as crianças também desempenharem um papel importante nesse jogo. Existem limites e regras que devem ser seguidas, mas elas podem mudar com o tempo dependendo de quem está jogando. Não há como "vencer" o jogo do casamento, não há linha de chegada e você não pode se declarar a melhor esposa/marido. O objetivo do casamento é apenas mantê-lo existindo de maneira saudável e respeitosa.

Em 2019, Simon Sinek pegou esses conceitos e escreveu um livro sobre eles chamado *The Infinite Game*, no qual descreve os desafios que surgem com esses tipos de jogos.

Quando há um jogador finito competindo contra outro jogador finito, o sistema é estável. E quando há um jogador infinito competindo contra outro jogador infinito, o sistema também é estável. Os problemas começam a acontecer quando há um jogador finito competindo contra um jogador infinito porque o jogador finito está jogando para ganhar e o jogador infinito está jogando para permanecer no jogo.

A programação é um jogo infinito, embora a maioria de nós aja como se fosse um jogo finito. Os jogadores são conhecidos e desconhecidos - sempre há novos programadores entrando no mercado de trabalho. As regras são mutáveis - novos paradigmas aparecem, novos padrões são inventados, novos bugs são encontrados. O objetivo é continuar jogando - você não pode vencer na programação, só pode continuar evoluindo o software para ser melhor, mais escalável e mais útil todos os dias.



Programadores que estão jogando o jogo finito, estão focados em seus bônus no final do ano ou aguardando o término daquele projeto freelancer. Enquanto isso, programadores que jogam o jogo infinito, estão menos preocupados com a intensidade e mais focados na consistência. Eles não sabem exatamente quando verão os resultados. De fato, pessoas diferentes mostrarão resultados em momentos diferentes. Ainda assim, sem dúvida, todos sabem que funcionará.

```
// TODO

Você está jogando um jogo finito ou infinito? Quanto tempo
você gasta tentando obter ganhos a curto prazo versus
investindo tempo em resultados a longo prazo?
```

 # Perguntas e Respostas: Pense no melhor programador com quem você já trabalhou. O que eles faziam constantemente que faz você pensar dessa maneira?

Luciano Sousa (Shopify):

> "Acho que aprendi muito com pessoas que sempre tiveram uma boa agenda em suas vidas. Elas estavam sempre no horário, começando e terminando o dia de trabalho na hora certa. Todas essas pessoas me mostraram que eu só posso ser bem-sucedido se organizar meu dia de maneira inteligente e evitar a procrastinação.
>
> Em geral, penso que a maioria delas teve um impacto positivo na minha vida, me incentivando a sempre tentar criar um padrão diário de trabalho simples e direto. Um padrão não tão rígido e ao mesmo tempo não tão flexível."

Berg Brandt (Amazon, Ex-Yahoo):

> "Uma das coisas que mais valorizo em qualquer profissional é a integridade. A integridade se manifesta de várias formas e os melhores desenvolvedores com quem trabalhei demonstraram integridade de uma maneira ou de outra. A mais fundamental é o foco em fazer a coisa certa pelas razões certas.
>
> Desenvolvedores de alta integridade estão conscientes do resultado e por que ele é importante. Embora possam correr, se necessário, andam de forma consistente e implacável em direção à meta. Eles proativamente chamam sua atenção se algo estiver errado ou se o seu

plano precisar de ajustes. Eles são os que assumem um projeto que não é tão glamuroso apenas porque é importante fazê-lo. Por último, mas não menos importante, são eles que estarão ao seu lado, ajudando a encontrar soluções quando as coisas derem errado."

Caio Gondim (New York Times):

"Os melhores programadores com quem trabalhei são confiáveis. E você não pode ter consistência sem confiança. Eles sabem que não vale a pena trabalhar longas horas durante a noite, porque no dia seguinte, você não ficará descansado o suficiente para pensar com clareza. Sabem que é essencial se divertir, se alimentar bem.

Eles não fazem estimativas com base em condições perfeitas. Porque sabem que é raro ter um dia inteiro sem distrações - um dia inteiro apenas de programação.

Eles sabem que existem reuniões, que as coisas podem dar errado em produção e que precisam ajudar outras pessoas a navegar no código-fonte."

Blake Williams (GitHub):

"Graças ao meu tempo de consultoria, pude trabalhar com um grande número de equipes em contextos variados ao longo dos anos. Acabei percebendo um padrão: as pessoas que considerava melhores programadores eram extremamente disciplinadas. A disciplina que esses desenvolvedores tinham parecia como um superpoder no início de minha carreira. Eles sempre tiveram tempo para escrever um ótimo código, adicionar uma cobertura abrangente de teste e explicar os detalhes minuciosos por meio de mensagens detalhadas de commit.

Os desenvolvedores disciplinados não trabalham a noite toda e se estressam em entregar a próxima tarefa. Eles investem seu tempo em fazer as coisas bem e se comunicar com clareza. Na minha experiência, essa disciplina ajuda outras pessoas da equipe a sentir menos necessidade de se apressar e permite que invistam tempo melhorando seu trabalho existente, em vez de passar imediatamente

para a próxima tarefa. Por fim, esses desenvolvedores economizam o tempo da equipe e do time, diminuindo a velocidade e adotando uma abordagem pragmática para a solução de problemas."

Hábito 5: Faça para o seu futuro eu

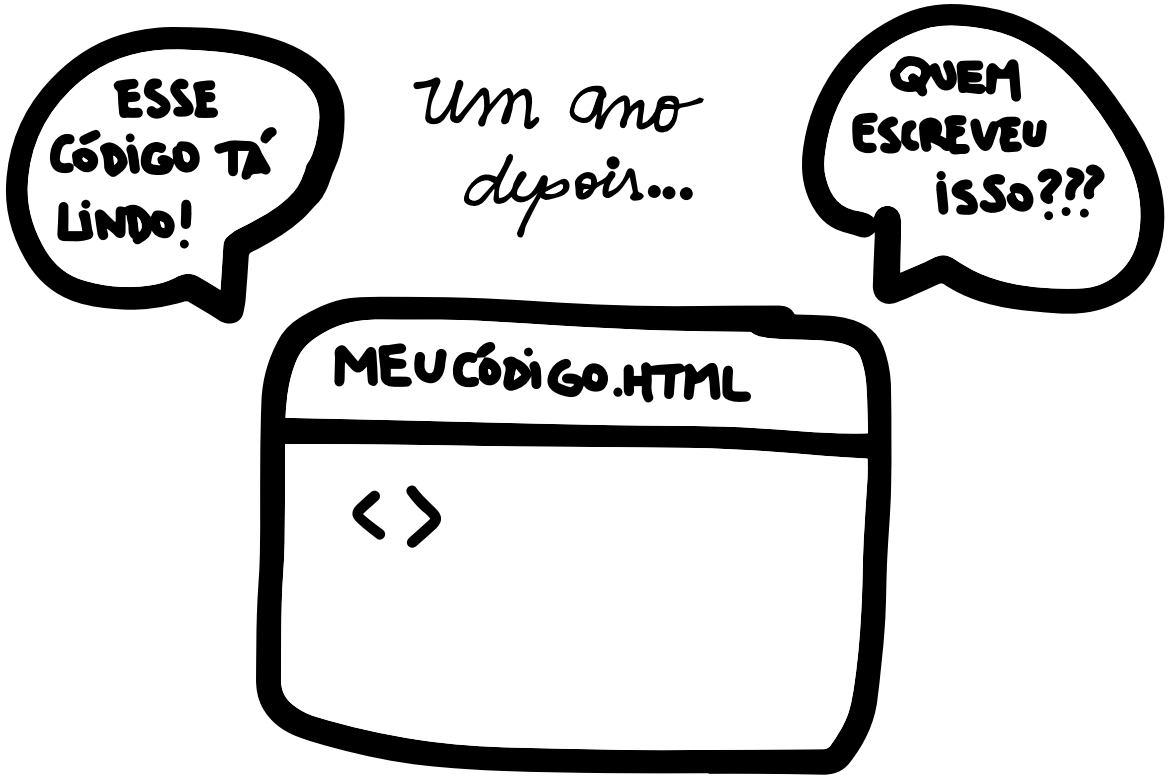
“Quando você sentir necessidade de fazer um comentário, primeiro tente reestruturar o código de modo a que qualquer comentário se torne supérfluo.” - Martin Fowler

Não gosto de julgar o código de outras pessoas. Também não estou dizendo isso para impedir que as pessoas julguem o meu código. Digo isso porque mesmo os melhores programadores do mundo escreveram códigos ruins no passado. E isso não é apenas porque eles ainda estavam aprendendo e não conheciam todas as melhores práticas. A razão pela qual todos escrevemos códigos de baixa qualidade de tempos em tempos está relacionada às circunstâncias que nos rodeiam naquela semana, dia ou horário específico.

Imagine tentar escrever um código simples de entender, bem documentado, rápido e modular, enquanto você está passando por muita ansiedade na sua vida. É muito difícil ser eficiente como programador quando você enfrenta muita pressão do seu chefe e tem prazos loucos para cumprir. Ainda assim, muitos de nós passamos por essas experiências pelo menos uma vez em nossas vidas profissionais. Para aqueles que ainda estão sendo tratados assim todos os dias, deixa eu te dizer uma coisa: existem lugares melhores para se trabalhar por aí.

Agora vamos supor que você não esteja em um ambiente de trabalho tóxico. Vamos fingir que você está trabalhando em um projeto paralelo, e não há absolutamente nenhuma pressão externa para terminar este software. Você abre o seu editor de código e começa a programar. As ideias começam a fluir e tudo parece funcionar. Após horas de codificação, você analisa todos os arquivos que escreveu e se sente orgulhoso do seu trabalho. Tudo faz sentido; é fácil de ler, fácil de manter, fácil de evoluir.

De repente, há uma mudança de prioridade e você abandona essa ideia. Você se envolve em outros projetos, começa a usar linguagens diferentes, tenta novos frameworks. Um ano depois, você decide continuar o que começou, abre o projeto novamente e é irreconhecível!



A maioria de nós já passou por isso. E a causa principal é que você está escrevendo um código para o seu eu atual. Em vez disso, você precisa escrever para o seu eu futuro. Seu eu atual tem todo o contexto necessário para entender esse bloco de código, enquanto o seu eu futuro estará envolvido com outras coisas e provavelmente nem se lembrará do que essa função X significa.

Não tente se mostrar, não tente codificar algo para fazer você se sentir mais inteligente, não precisa mostrar todos os novos truques que acabou de aprender. Basta escrever um código legível. Pense em manutenção e use nomes significativos para suas classes, funções e variáveis. Da próxima vez que começar a desenvolver, faça a si mesmo esta pergunta: *“O futuro eu entenderá a intenção deste código?”*

/* Bônus: criei uma lista dos meus 5 livros favoritos sobre arquitetura de software e boas práticas. Acesse 14habits.com/br/bonus/5 para receber. */

```
// TODO

Abra um projeto atual em que você esteja trabalhando. Existe
alguma refatoração que você possa fazer para facilitar a vida
do seu eu futuro?
```

 # Perguntas e Respostas: Quais são as coisas que você faz hoje para se ajudar no futuro?

Daniel Buchner (Microsoft):

> "Obviamente, todo mundo tenta criar algo que tornará seu projeto atual um sucesso, mas para ajudar a garantir que o futuro do seu trabalho seja mais previsível, com melhores resultados a longo prazo, uma habilidade crítica a ser desenvolvida é a capacidade de olhar em volta. Provavelmente, existem muitos caminhos que você pode seguir e que levarão ao sucesso do seu projeto atual, mas escolher aquele que configura a próxima versão e as cinco seguintes para o sucesso, é mais desafiador.
>
> Ao pensar sobre isso, tenho o costume de observar tendências maiores do setor, a progressão de onde acredito que a tecnologia estará em 3 a 5 anos e quais padrões emergentes parecem sólidos o suficiente para apostar. Você nem sempre encontrará algo novo para incorporar a cada projeto que trabalha, mas quando faz uma pequena modificação no início de um projeto pode mudar significativamente o fim dele."

Silvio Gustavo (Spotify):

"Eu tento ter empatia com qualquer um que venha a mexer neste código no futuro, incluindo o meu futuro eu. Esse código é legível para desenvolvedores com qualquer nível de experiência? Com esse pensamento em mente, busco:

1. Escrever um código simples com nomes significativos de variáveis/métodos/classes. Não assuma que outras pessoas (incluindo você) entenderão nomes abreviados que não significam muito ou alguns termos específicos que podem mudar com o tempo.
2. Ter bons testes automatizados. Isso fornecerá documentação adicional sobre o seu código e também ajudará qualquer pessoa que precise trabalhar nele no futuro.
3. Usar a ferramenta de controle de versão como documentação. A medida que o projeto evolui, são feitas alterações e correções de bugs o tempo todo. No futuro, quando se tornar um código legado, ninguém conseguirá entender as decisões e alterações feitas naquele momento, se não estiverem documentadas. O histórico de commits e pull requests são uma boa ferramenta para explicar os porquês."

Lais Andrade (Google):

"Uma das melhores coisas para manter a base de código à prova do futuro é ter bons padrões e que eles sejam seguidos pela equipe. Isso vai além do estilo, vai até as ferramentas preexistentes. Reinventar a roda é algo que a maioria de nós é facilmente levado a fazer, mas uma pesquisa rápida pode reduzir a quantidade de novo código adicionado, código que exigiria manutenção futura, além de ajudar a evitar erros conhecidos e que já foram resolvidos por boas bibliotecas.

O segundo ponto-chave é muito famoso, mas surpreendentemente difícil de pôr em prática: a otimização precoce é a raiz de todo mal. É fácil escrever uma única linha sem comentário que faça exatamente o que precisamos, mas é muito difícil consertar essa linha algumas semanas, imagina meses ou anos no futuro. As revisões em pares são extremamente úteis para evitar essas armadilhas: sempre que você precisar de muito contexto para entender exatamente o que uma

linha/método/componente está fazendo, você deve se afastar e pedir aos programadores para adicionar mais variáveis intermediárias com nomes significativos ou adicionar alguns comentários explicando por que isso foi feito dessa maneira, ou mesmo para dividi-lo em componentes menores e mais simples.

O terceiro e último aspecto é sempre testar tudo o que você escreve. Teste de unidade, teste de integração, teste de desempenho, tudo isso. A imposição de uma política simples de ‘nenhum código novo sem testes de unidade’, por exemplo, pode fazer maravilhas. Definitivamente, você pode notar a diferença entre as bases de código que a aplicam e as que não o aplicam. Se os testes são feitos como um aspecto fundamental do projeto e recebem a prioridade certa, é fácil para a equipe poupar alguns ciclos trabalhando em melhores ferramentas de teste (a qualidade do código de teste é tão importante quanto a qualidade do código de produção) e infraestrutura (como integração contínua etc.). Nenhum código deve ser adicionado sem testes de unidade, nenhum bug corrigido sem um teste de regressão. Esses pequenos passos terão um enorme impacto na qualidade geral do projeto."

Hábito 6: Seu 9 às 5 não é suficiente

“Não espere estar motivado todos os dias para chegar lá e fazer as coisas acontecerem. Você não estará. Não conte com motivação. Conte com a disciplina.” - Jocko Willink

Nos Estados Unidos, existe um termo chamado “9 às 5”, que significa um trabalho tradicional de 8 horas por dia, 40 horas por semana. Se estendermos essa definição, temos um “programador de 9 às 5”, isso significaria uma pessoa que possui um trabalho técnico, mas depois que é concluído, eles deixam o trabalho lá e não o leva para casa.

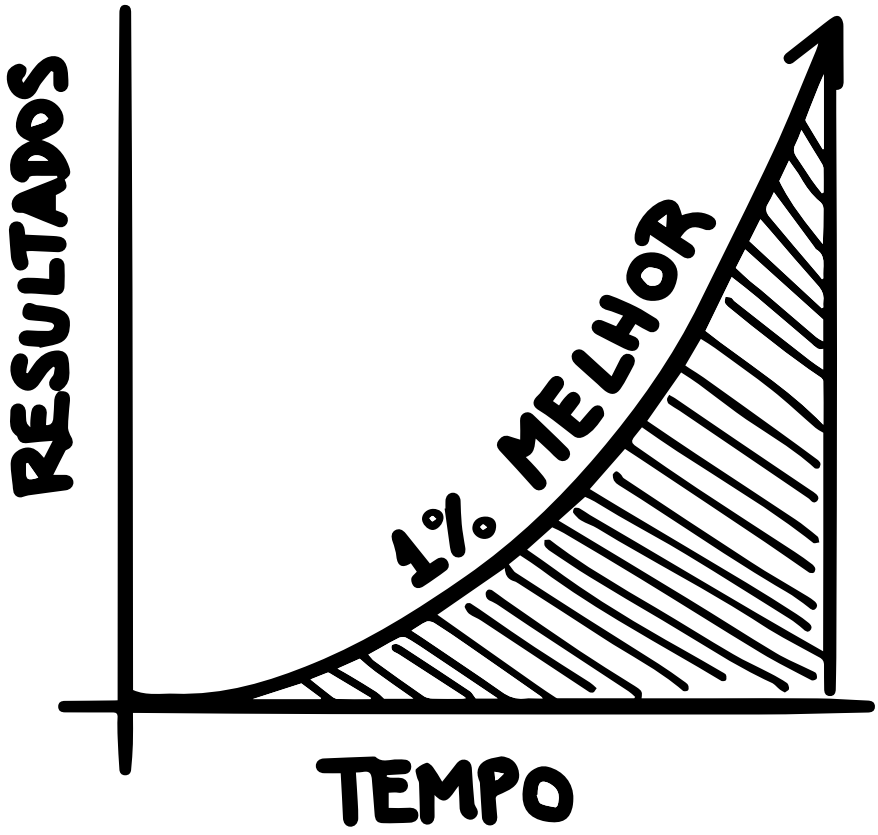
Durante minha carreira, conheci muitos programadores de 9 às 5. Alguns deles eram brilhantes, focados e sabiam exatamente o que estavam fazendo. Outros simplesmente não gostavam muito de programação, essa era apenas uma maneira de ganhar dinheiro, e eles tinham outros hobbies que eram mais importantes.

Eu não acho que haja algo errado nisso. Cada um de nós tem a liberdade de usar seu próprio tempo livre da maneira que desejar. No entanto, acredito realmente que, se você deseja ter uma carreira de destaque em tecnologia, vai precisar investir horas extras nas suas habilidades.

Alguém poderia argumentar que não esperamos que os policiais persigam criminosos no seu tempo livre, ou que os bombeiros apaguem incêndios extras. Então, por que devemos esperar que programadores usem seu tempo livre para aprender algo novo? A resposta está na natureza do nosso trabalho.

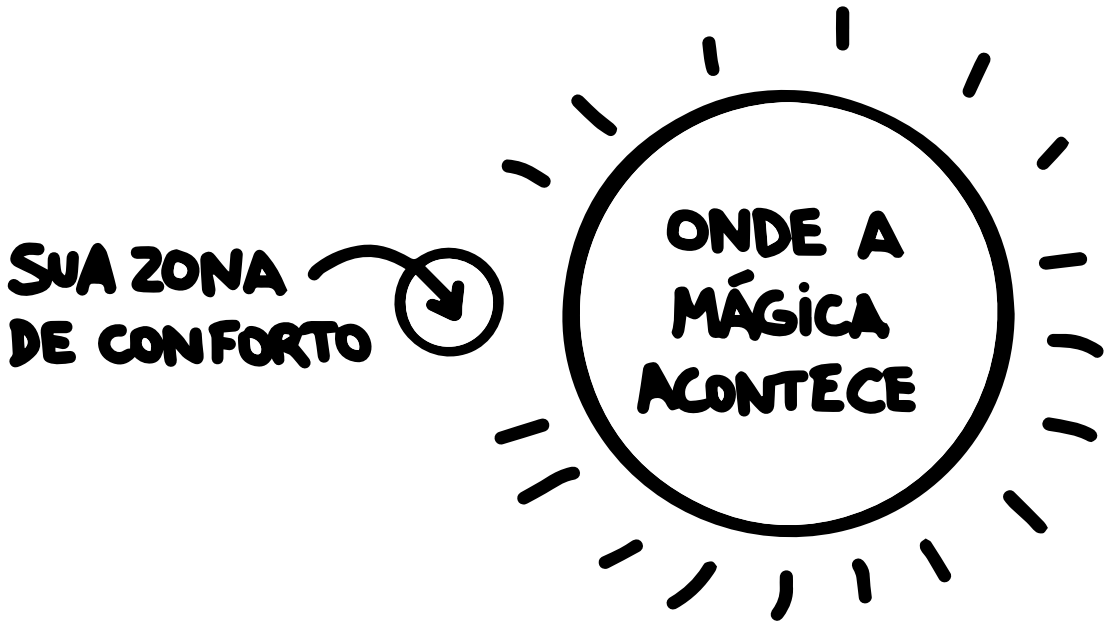
Nosso trabalho normal provavelmente não é suficiente para mostrar todo o nosso potencial. O que geralmente acontece é que estamos vinculados às restrições tecnológicas da empresa em que trabalhamos. Temos um conjunto determinado de linguagens, frameworks e estilo de código que precisamos seguir. Às vezes, essas tecnologias mudam e temos a oportunidade de aprender as novas no trabalho, mas se você está apenas

começando sua carreira, provavelmente vai precisar expandir seu horizonte com outras tecnologias no seu tempo livre.



Não estou sugerindo que as pessoas percam horas de sono para estudar uma nova tecnologia ou passar o final de semana inteiro programando. Definitivamente, não estou dizendo que você deve sacrificar suas outras responsabilidades como marido/esposa ou pai/mãe na busca de escrever um código melhor. Afinal, no final de sua vida, quando você olhar para trás, não quer que seu placar fique assim:

- Desenvolvedor de software: 9/10
- Marido/Esposa: 2/10
- Pai/Mãe: 4/10



O que estou tentando dizer é o seguinte:

- Você pode assistir Netflix e ainda assim ser produtivo.
- Você pode jogar seu PlayStation e ainda trabalhar em um projeto de código aberto.
- Você pode ter um tempo de qualidade com seu marido/esposa e ainda ler um livro.

Não deixe que as pessoas pensem que você não pode relaxar e fazer as coisas. Você tem tempo mais que suficiente.

```
// TODO

Pense nas habilidades que você acha que poderiam ser
melhoradas. Você pode planejar um tempo extra para
desenvolvê-las? Até mesmo 10 minutos por dia podem fazer a
diferença.
```

 # Perguntas e Respostas: Como você gasta seu tempo livre? Como você equilibra o trabalho com sua vida pessoal?

Silvio Gustavo (Spotify):

> "Eu costumava passar muito do meu tempo livre me aprofundando no desenvolvimento de software, especialmente em como o Android funciona internamente, pesquisando melhores práticas de desenvolvimento e em como melhorar a arquitetura do código. Era como um hobby para mim.
>
> Em um determinado momento, mesmo que eu gostasse, às vezes eu me sentia mais estressado ao combinar isso com o aumento da carga de trabalho das 9 às 5 horas. Decidi então desacelerar e usar meu tempo livre para fazer outras coisas, como jogar, aprender novos hobbies e me exercitar. Hoje em dia, tenho um equilíbrio muito melhor entre vida profissional e particular e ainda estou aprendendo coisas novas."

Fernando Tadashi (Adobe):

> “Nos últimos anos, mudei meus hábitos de aprendizagem devido à mudança que estou fazendo na minha carreira. Sempre gostei da área técnica e de programação. E, é algo que me anima saber os detalhes de como certos softwares funcionam e o controle que você tem sobre eles. Ainda dedico meu tempo a saber quais tecnologias estão surgindo e que ainda são comumente usadas, e às vezes eu me aprofundo nisso para aprender algo que posso usar e que pode me ajudar diariamente.”

Fabio Costa (GoDaddy, Ex-Facebook):

> "Nos fins de semana, costumo ficar muito mais com minha família e amigos. Isso geralmente pode envolver uma festa de aniversário ou apenas um encontro na casa de um amigo ou na nossa própria casa. Durante a semana, isso realmente depende. Colocamos nossos filhos para dormir muito cedo, então depois disso eu:
>
> - Fico com minha esposa, conversando sobre vários assuntos e fazendo algo especial para beber e/ou comer.
> - Se estou animado com um projeto pessoal, trabalho nos problemas em aberto.
> - Se estou animado com o trabalho, também posso trabalhar horas extras.
>
> Não sinto que a carga de trabalho no meu trabalho atual exija que eu trabalhe além do horário normal. A verdade é que, como a maioria dos desenvolvedores por aí, eu realmente amo o que faço, por isso, quando estou animado com um projeto, posso trabalhar horas extras, mas isso não afeta minha qualidade de vida, porque não fui forçado fazer isso."

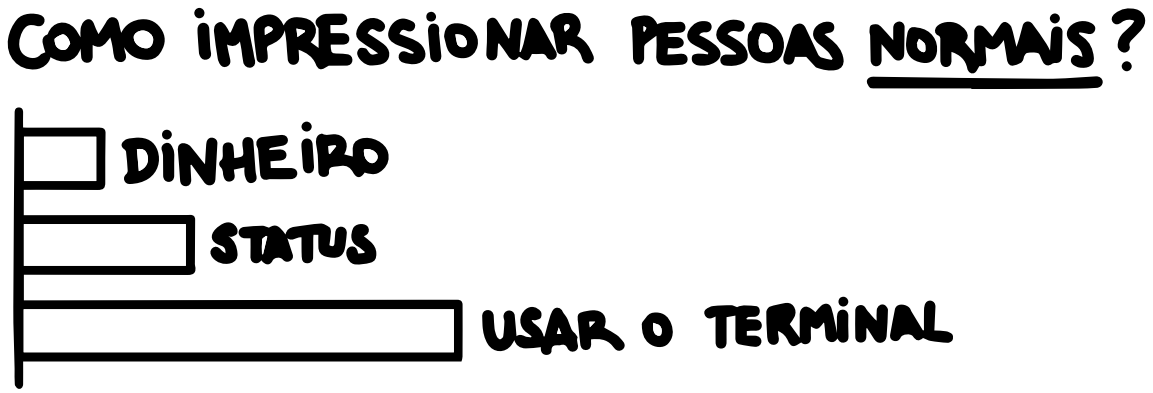
Parte quatro: Hábitos de carreira

Hábito 7: Domine o lado sombrio da força

“Nomeado deve ser o seu medo, antes de bani-lo, você pode.” — Yoda

No começo, a programação era um campo extremamente seletivo. Somente universidades de ponta possuíam currículos que abordavam tópicos de engenharia de software. Somente famílias privilegiadas podiam pagar tal educação para seus filhos. Somente “super-nerds” conseguiam programar e arranjar um emprego. Em outras palavras, a barreira de entrada era muito alta.

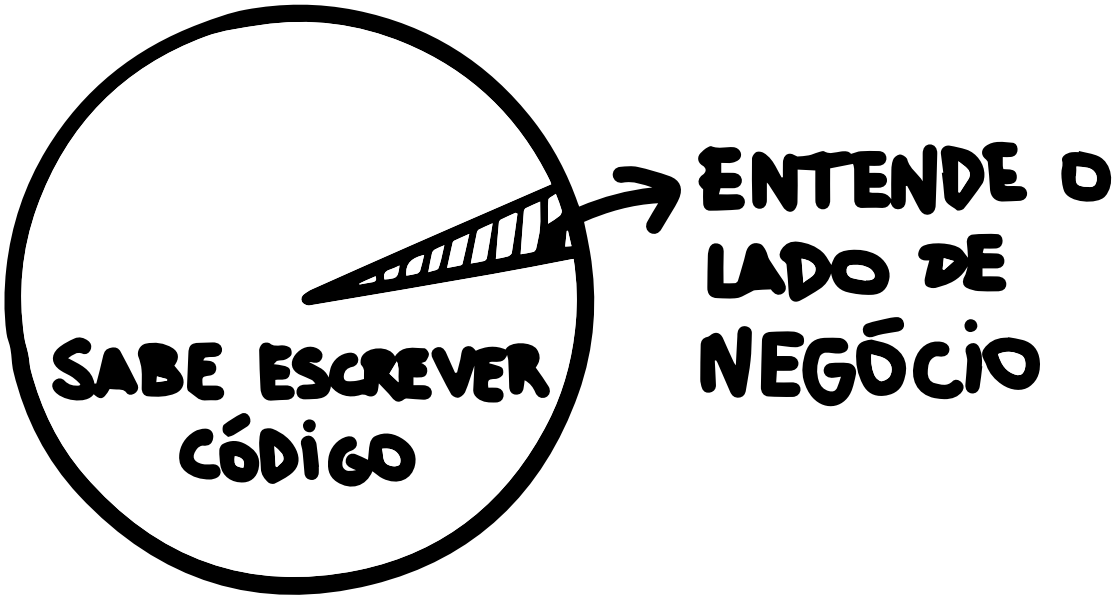
Computadores, internet e smartphones não são mais o futuro, eles são uma parte normal do nosso dia-a-dia. Do oeste ao leste, do mais novo ao mais velho, todo mundo tem algum tipo de dispositivo hoje em dia. Software está em toda parte e isso significa que programadores também estão em toda parte. De acordo com um estudo da Evans Data Corporation (EDC), a partir de 2019, já existiam 26 milhões de desenvolvedores de software em todo o mundo. E adivinha, esse número só continua crescendo a cada minuto.



Atualmente, qualquer pessoa pode se inscrever em um curso e aprender a programar em apenas algumas semanas ou meses. Você não precisa de um diploma ou certificado. Existem milhares de livros e conteúdos online que

podem ensinar você a se tornar um desenvolvedor. Implementar código não é mais a parte mais difícil. Ainda assim, uma coisa extraordinariamente rara são desenvolvedores que conhecem o lado de negócio.

Se você pensar bem, desenvolvedores são essencialmente tradutores. Nosso trabalho é traduzir os requisitos do usuário em funcionalidades. Em algumas empresas, esses requisitos do usuário são comunicados para gente com um alto nível de detalhe. Existem até analistas de negócios (BAs) que detalham todas as histórias de usuários e nos entregam uma enorme lista de critérios de aceitação. Em um mundo perfeito, iríamos apenas sentar em nossas cadeiras e escreveríamos o código, mas nem sempre é esse o caso. Obter informações de alguém pode ser muito complicado, pois todos temos uma compreensão diferente do sistema. É fácil se esconder atrás de arquiteturas complexas e jargões técnicos, porém quebrar paredes que dividem e se comunicar com essas pessoas pode ser um diferencial muito bom.



Durante muito tempo, vi clientes e vendedores como seres perversos. Esses personagens insistentes e motivados por dinheiro não se importavam com a qualidade do meu código e isso sempre me fazia não gostar deles. Esse tipo de atitude me fez focar apenas no código e esquecer todo o resto. Para mim,

programação era o *Lado Luminoso* e os negócios, o *Lado Sombrio*. No entanto, com o tempo, comecei a reconhecer o valor deles e comecei a perceber que conhecer o lado de negócio poderia me ajudar com as minhas atividades diárias.

1. **Economize tempo:** Mesmo que você gaste dezenas de horas planejando antes de começar a trabalhar em uma funcionalidade específica, você sempre descobrirá novas implicações e casos extremos que afetam sua implementação. Quanto melhor você entender o negócio, melhor será para resolver esses problemas. Quando você encontra essa exceção à regra, é mais provável que você já saiba a resposta e assim evite ter que agendar uma reunião com o especialista para encontrar uma solução.

2. **Evite código complexo:** Sempre queremos criar o código mais abstrato, flexível, extensível e reutilizável possível. Queremos tanto que às vezes desenvolvemos uma aplicação excessivamente complexa. Frequentemente, essas implementações nunca serão estendidas ou reutilizadas, gerando um cemitério de códigos. Há certas coisas que nunca serão estendidas, porque esse simplesmente não é o foco da aplicação. Quando você possui experiência de negócio, você está em uma posição muito melhor para determinar qual parte da base de código precisa de mais atenção do que outras.

3. **Priorize melhor:** Se tivermos um prazo para a próxima semana, a implementação será muito diferente da prevista para amanhã. Essa é apenas a realidade do trabalho. Com um melhor conhecimento dos negócios, é mais fácil priorizar as micro-decisões que você precisa tomar ao programar. Você pode prever em que parte é mais importante gastar tempo. Você pode se colocar no lugar do usuário e sentir sua dor. Além disso, quando você souber quais são as funcionalidades de negócio mais críticas, provavelmente escreverá um código de alta qualidade nessa área, o que impedirá a refatoração futura.

Cada indústria é diferente, então não existe uma receita de bolo pronta para seguir quando se trata de conhecer melhor o lado de negócio. No entanto, minha recomendação é começar com o vocabulário. Preste atenção às palavras e termos específicos usados pelas pessoas de negócios. Se você

imitar a mesma terminologia, a comunicação com eles será mais eficiente. Depois de começar a ler e consumir mais conteúdo sobre seu setor, você naturalmente desenvolverá mais conhecimento da indústria.

Lembre-se, uma pessoa que sabe programar é poderosa, uma pessoa que sabe programar *e* sabe como os negócios funcionam é imparável.

```
// TODO

Elabore uma lista dos termos mais comuns usados no seu
ambiente de trabalho. Converse com seus colegas para entender
suas áreas. Como é o funil de vendas? Quais nichos de
marketing estão sendo visados? Quais são as perguntas mais
comuns do suporte ao cliente? Qual é a diferença entre o seu
produto e o dos concorrentes?
```

 # Perguntas e Respostas: Você tenta entender o lado de negócio das coisas antes de programar alguma coisa?

Michael Lancaster (BlackBerry):

> "Depende da complexidade do trabalho e do tempo, porque muitas vezes podemos realizar o trabalho muito bem entendendo pouco ou nada do negócio.
>
> Dito isso, quanto mais soubermos sobre o negócio, mais valiosos seremos para a equipe, pois isso nos permitirá usar nosso conhecimento técnico e comercial para dar uma contribuição maior."

Caio Gondim (New York Times):

> "Vou responder a essa pergunta contando uma história.
>
> João trabalha em um banco como líder técnico. Sua equipe foi designada para criar um novo sistema usado por quase todos os funcionários do banco.
>
> Em uma reunião com os usuários, eles pediram uma maneira de exportar todos os dados que haviam sido salvos anteriormente no Excel para o novo sistema. No entanto, o sistema não possui um recurso para importar diretamente. Seria necessário implementar um programa que leia todas as entradas nos arquivos do Excel e as

converta em chamadas para o novo sistema. João estimou que isso levaria um mês.

Marisa foi a escolhida para fazer o projeto. Apesar da complexidade da tarefa, ela pode concluí-la em três semanas (uma semana a menos do que o esperado).

João fica satisfeito com o resultado e diz: ‘Finalmente, poderemos importar dados da planilha para o novo sistema. Nossos usuários estavam ansiosos por isso!’.

Marisa fica surpresa ao saber que foi nisso que ela trabalhou, e diz: ‘Poxa, João. Nós não precisávamos fazer todo esse trabalho. Poderíamos simplesmente exportar do Excel para CSV e usar o recurso de importação de CSV. Nossos usuários poderiam ter feito isso sozinhos’.

O “problema XY” é quando perguntamos como implementar a solução, quando deveríamos estar falando sobre o problema em si. João concluiu que era necessário implementar o recurso X e perguntou à sua equipe como implementar esse recurso. Se ele tivesse falado sobre o problema que estava tentando resolver, a equipe responderia que esse recurso não é necessário."

Luciano Sousa (Shopify):

“Eu sempre tento entender o lado de negócio do que estou desenvolvendo. Essa é a melhor maneira de garantir que estou entregando algo útil ao cliente. É um processo difícil, mas fico feliz que a maioria das empresas que trabalhei sempre tentaram promover esse tipo de interação entre P&D e clientes. Sem essas interações, tenho certeza de que nem a qualidade do que estava fazendo nem a qualidade do projeto em geral seriam tão eficazes quanto às expectativas do cliente.”

Hábito 8: Projetos paralelos

> *“Fracasso e invenção são gêmeos inseparáveis. Para inventar, você precisa experimentar, e se você sabe com antecedência que isso vai funcionar, então não é um experimento.” — Scott Galloway*

Quando pensamos nos produtos de maior sucesso hoje, pensamos imediatamente nessas startups ou unicórnios com muito apoio financeiro e uma enorme equipe por trás delas. Mas você sabia que o Twitter, o Craigslist e o Slack começaram como projetos paralelos?

Algumas pessoas iniciam projetos paralelos para criar um novo produto e, eventualmente, construir uma empresa de sucesso. No entanto, existem muitas outras razões pelas quais trabalhar em projetos paralelos pode ser útil. E você não precisa ter espírito empreendedor ou desistir do seu trabalho em tempo integral para fazer.

Se você está apenas começando sua carreira, projetos paralelos podem ser uma tremenda oportunidade para aumentar seu portfólio, criar um currículo e mostrar suas habilidades. Você pode ler muitos livros sobre qualquer assunto em particular, mas não há nada como “sujar as mãos” e projetos paralelos podem ser essa caixa de areia para você.

Se você já está estabelecido em sua carreira, projetos paralelos ainda podem ser muito úteis. Eles permitem que você experimente e aprenda novas habilidades que seu trabalho do dia-a-dia pode não oferecer. Além disso, não há pressão, horários ou especificações impostas por ninguém, o que significa que você pode usar toda a sua criatividade e se divertir fazendo isso.

Ok, digamos que você deseja criar um projeto paralelo. O próximo passo é decidir o que você deve construir? Que ideias vale a pena buscar?

Esta é uma lista de perguntas que me faço antes de iniciar um novo projeto paralelo.

1. Um projeto paralelo significa que você vai precisar renunciar seu tempo pessoal para trabalhar nele, portanto, a coisa mais importante a se perguntar é: *"Eu realmente gosto deste assunto, campo, tipo de trabalho?"*

Se a resposta for "não", há chance de que você não terá energia para gastar horas extras trabalhando nele.

2. Leva tempo para um projeto paralelo ganhar tração, então a próxima coisa que você deve se perguntar é: *"Estou disposto(a) a passar pelo menos cinco anos trabalhando nessa ideia?"*

Se a resposta for "não", é muito provável que você perca a motivação e acabe desistindo do projeto antes que ele decole. No entanto, se seu objetivo é apenas experimentar, você não precisa se preocupar com isso.

3. Ter uma ideia é uma coisa, ter a capacidade de executá-la é outra, portanto, você deve se perguntar: *"Posso executar essa ideia*

completamente sozinho?”

Se a resposta for “não”, considere aprender uma nova habilidade ou convidar um amigo para preencher as lacunas.

4. Dizer “sim” a uma ideia significa dizer “não” a várias outras ideias, então pergunte a si mesmo: *“Essa ideia em particular é melhor do que outras que eu tive no passado? Existe alguma outra ideia que poderia usar melhor o meu tempo?”*

Se a resposta for “não”, repita esse ciclo com a outra ideia.

5. Entender para quem você está construindo esta solução é vital. Se você não conhece seu público, é improvável que você entenda as necessidades desse público, então, pergunte a si mesmo(a): *“Tenho esse problema pessoalmente ou estou resolvendo para outra pessoa?”*

Se a resposta for “não”, considere entrar em contato com pessoas que realmente experimentam a dor que você está tentando resolver. Você pode se surpreender com o que elas têm a dizer.

6. E a última pergunta é: *“Por que estou animado com essa ideia agora?”*

É crucial entender sua motivação. Você quer aprender uma nova tecnologia? Você quer fazer mais dinheiro? Você está se distraindo dos outros problemas? Seja honesto com você mesmo(a).

Mais importante do que escolher sua ideia é escolher o escopo do seu projeto. Muitas vezes, projetos paralelos não vêem a luz do dia.


RIP
MELHOR
IDÉIA
DA VIDA
RIP
IDÉIA
milionária
7
RIP
TODA
IDÉIA

/* Bônus: Eu listei 7 ideias para começar um projeto paralelo, incluindo exemplos da vida real de meus próprios projetos pessoais. Vá para 14habits.com/br/bonus/8 para pegar. */

```
// TODO

Nos próximos dias, preste atenção aos aplicativos que você
mais usa. Falta alguma coisa neles? Você seria capaz de criar
uma versão melhor disso? Tenha um bloco de notas dedicado
para todas as suas ideias de projetos paralelos e leve as
notas para todos os lugares com você. Quando estiver pronto,
escolha uma dessas ideias e tente.
```

 # Perguntas e Respostas: Você tem o hábito de criar projetos paralelos? Qual é o impacto em sua carreira?

Addy Osmani (Google):

> "Os projetos paralelos sempre foram uma ótima saída criativa para resolver problemas pequenos e divertidos de maneira menos estressante. Adoro o trabalho que faço como parte do meu trabalho diário, mas às vezes pode ser complexo (mesmo sendo legal) e levar vários meses para ser terminado. Por outro lado, os projetos paralelos geralmente são rápidos, podem ser descartados ou, na melhor das hipóteses, podem levar a um pequeno projeto de código aberto que você pode compartilhar com outras pessoas. Atualmente, existem sites inteiros dedicados a ajudar você a começar projetos paralelos, então agora seria um ótimo momento para experimentar um.
>
> O impacto que os projetos paralelos tiveram na minha carreira é substancial. Eles me deram oportunidades para criar bibliotecas e ferramentas que agora são usadas por outras pessoas. Por meio do código aberto, conheci muitos amigos (incluindo o Zeno!), ampliei meu conhecimento revisando pull requests e a minha admiração

aumentou pelas boas práticas de engenharia que podem ser diferentes daquelas que normalmente uso no trabalho.

Também é muito legal trabalhar em projetos paralelos que outras pessoas já haviam tentado antes. Alguns dos meus projetos paralelos foram apenas eu trabalhando em um repositório privado tentando reimplementar uma ideia apenas para o meu próprio conhecimento. Enquanto você estiver se divertindo, construindo coisas legais e aprendendo algo com a experiência, vale a pena."

Michael Lancaster (BlackBerry):

"A criação de projetos paralelos me ajudou em uma variedade de coisas, como adquirir um melhor conhecimento das tecnologias que eu já estava usando no meu trabalho do dia a dia, me familiarizar com tecnologias às quais, de outra forma, eu não estaria exposto no trabalho, e aprender muito mais sobre mim e como ser produtivo.

Além de tudo isso, ao trabalhar em projetos paralelos, podemos nos divertir muito e ter a chance de sermos muito criativos, o que nos forçará a explorar e trabalhar em outras áreas que talvez não tivéssemos a mesma chance de outra maneira."

Blake Williams (GitHub):

"Adoro projetos paralelos. Eles me permitem expressar minha criatividade e investir em mim mesmo, me incentivam a aprender novas habilidades para alcançar os objetivos que estabeleci. Ao longo dos anos, os projetos paralelos me levaram a melhorar meu CSS, aprender React, melhorar o design e muito mais. Essas habilidades foram fundamentais para ajudar no crescimento da minha carreira e abriram portas para mim que, de outra forma, estariam fechadas.

Além de ajudar minha carreira, também me dá a chance de criar e retribuir à comunidade através de contribuições de código aberto. O código aberto me levou a conhecer e trabalhar com pessoas realmente incríveis. Sem projetos paralelos, acho que não estaria onde estou hoje."

Hábito 9: Mario ou Sonic?

“Como a dor é uma constante universal na vida, as oportunidades para crescer a partir dessa dor são constantes na vida. Tudo o que precisamos é entender que não devemos anestesiar e nem ignorar essa dor. Tudo o que importa é enfrentar e encontrar o valor e o significado da dor.” - Mark Manson

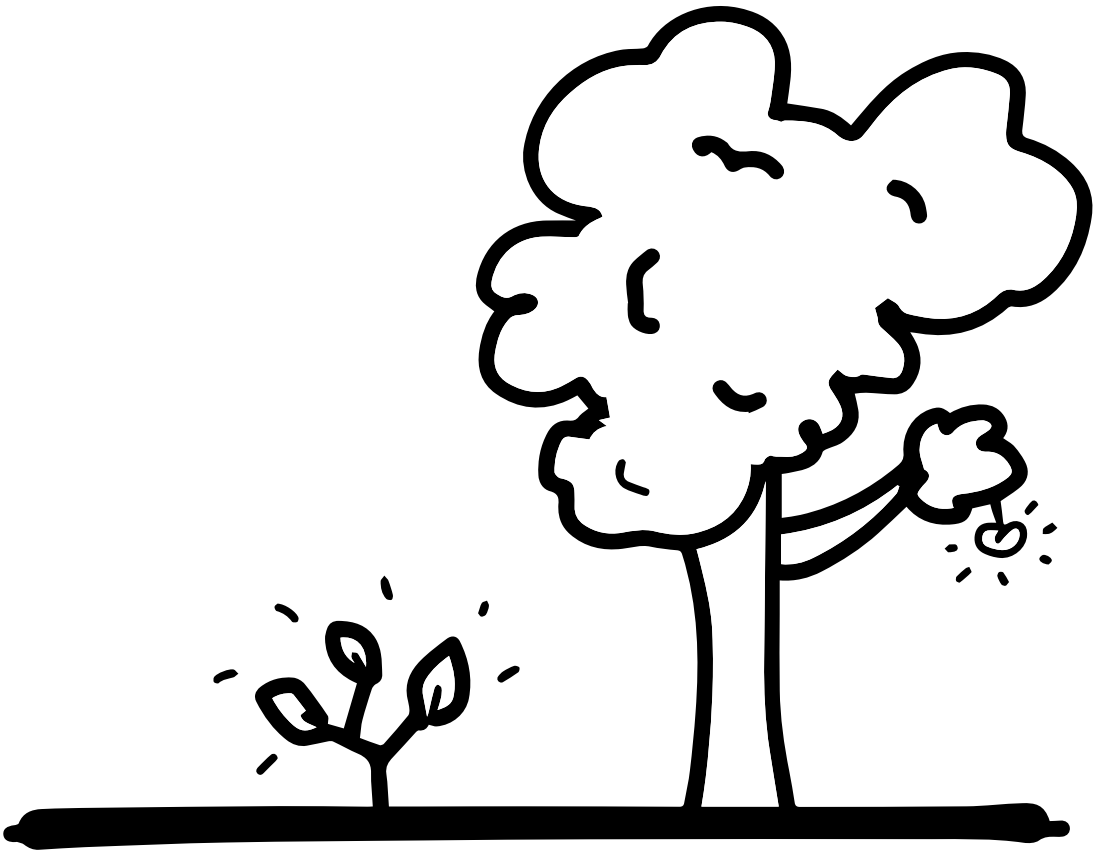
Um dos jogos de festa mais legais que você pode jogar se chama *“Quem venceria em uma luta.”* Se você não está familiarizado com o jogo, a ideia básica é elaborar cenários malucos e discutir quem é mais poderoso. Por exemplo: quem venceria em uma briga entre Batman e Homem de Ferro? Dumbledore e Gandalf? Darth Vader e Thanos? Você entendeu a ideia.

Existem dois personagens fictícios que estavam muito presentes na minha infância e que eu sempre pensei que deveriam ter uma batalha épica. No canto vermelho, imagine Mario, o encanador icônico que pode saltar super alto e sempre protege o Reino dos Cogumelos do tirano Bowser. No canto azul, imagine Sonic, o Ouriço, que pode correr em velocidades incríveis, atravessar robôs como nada e que protege o mundo do vilão Dr. Eggman. Agora, vamos imaginar eles como engenheiros de software e, em vez de brigarem, eles estão apenas vivendo suas vidas e progredindo em suas carreiras.

Mario está sempre pulando de um lugar para o outro. Se ele fosse um programador, seria uma dessas pessoas que permanece no emprego por seis meses e depois se muda para outro. Mario é confiante e forte, mas na maioria das vezes ele está apenas evitando situações perigosas. Você pode pensar naquele amigo que é muito inteligente, mas que por algum motivo tem medo de ir para a entrevista de emprego.

Por outro lado, Sonic está sempre disposto a enfrentar os maiores desafios que pode encontrar. Se ele fosse um programador, estaria sempre procurando problemas complexos para resolver e as tecnologias mais avançadas para trabalhar. Sonic é rápido, ele pode aprender coisas novas, se adaptar a novos desafios e não fica para trás.

No início de sua carreira, não há problema em mudar de emprego com frequência. Você provavelmente precisa trabalhar em muitos lugares diferentes para ter novas experiências. No entanto, há um efeito colateral nisso. Quanto menos tempo você gasta em um projeto, mais superficial você será como profissional. Por outro lado, quanto mais tempo você gasta em um projeto, mais oportunidades terá para ter um impacto a longo prazo.



Ambos são incríveis e, no final das contas, ambos têm o potencial de salvar o mundo. Mas se você tivesse que escolher entre esses dois, não seja como Mario, seja como o Sonic.

```
// TODO

Pense nas três coisas que você mais gosta no seu emprego
atual. Agora pense nas três coisas que você mais odeia no seu
emprego atual. Há algo que você possa fazer para transformar
essas partes ruins em partes boas? Essas mudanças estão fora
do seu controle ou não?
```

 # Perguntas e Respostas: Qual foi o emprego mais longo que você teve? Por que você ficou lá por tanto tempo?

Berg Brandt (Amazon, Ex-Yahoo):

> "Trabalhei no Yahoo por quase 10 anos em 3 países diferentes - Brasil, Canadá e Estados Unidos - em diversas funções e departamentos. Meu primeiro projeto foi uma reformulação das páginas iniciais do Yahoo para a América Latina e o Canadá. Este trabalho me deu a oportunidade de interagir com várias pessoas no Canadá, o que acabou gerando uma oferta para eu ir trabalhar lá no final de 2007. A partir daí, quase 2 anos depois, em setembro de 2009, acabei vindo para Santa Mônica, onde passei a maior parte do meu tempo.
>
> Em Santa Mônica, tive a oportunidade de ser um dos fundadores de uma plataforma de publicação para centenas de sites de mídia e entretenimento do Yahoo nos EUA e internacionalmente. Eu aprendi e cresci muito com esse projeto. Dada a minha experiência na plataforma, me tornei líder de tecnologia e depois mudei para gerente. Depois de alguns anos nesse projeto, me ofereceram a oportunidade de liderar o redesenho do Yahoo Screen (plataforma de vídeo do Yahoo) e mudei para o departamento de Vídeo, onde permaneci até sair em 2016.
>
> Yahoo transformou minha vida profundamente e sou extremamente grato por isso. Fiquei tanto tempo porque havia muito o que fazer, muito a aprender e meu trabalho foi impactante. Marty Cagan, do

> Grupo de Produtos do Vale do Silício, tem um ótimo artigo sobre ‘Missionários vs. Mercenários’. Eu sou muito missionário. Acredito que você tenha que ficar em algum lugar por um certo período de tempo para causar impacto e se expor às oportunidades que levarão sua carreira ao próximo nível. Penso em todos os trabalhos em termos de investimento e retorno, e não se trata apenas de dinheiro, mas de retorno do investimento para a sua carreira como um todo."

Netto Farah (Segment):

> "Estou na Segment há 3,5 anos, e é o maior tempo que já fiquei em uma empresa de tecnologia. A principal coisa que me mantém aqui é o quanto estou empolgado com a trajetória da empresa, seguido por uma autonomia realmente boa (que também veio com muitas responsabilidades).
>
> Acho que não há um número mágico ou resposta correta para esta pergunta. Costumo tentar otimizar a felicidade e me considero muito privilegiado por poder mudar de emprego facilmente, se quiser. Trabalhar de 40 a 50 horas por semana é um investimento de tempo bastante significativo em sua vida, por isso é muito importante que você esteja satisfeito com seu emprego atual, independentemente do salário ou quanto tempo está na escada da sua carreira."

Manuel De La Peña (Elastic):

> "Trabalhei na Liferay por quase 8 anos. Entrei no verão de 2011 e saí na primavera de 2019. Entrei como desenvolvedor Java com alguma experiência na tecnologia usada por esta empresa e saí como engenheiro de software com um perfil totalmente diferente. Tive a chance de trabalhar com muitas tecnologias, muitas vezes nem relacionadas ao produto. Nuvem, virtualização, automação, linguagens de programação, padrões de design. Posso dizer que metade da minha experiência vem do tempo que trabalhei lá.
>
> No entanto, a tecnologia não foi a razão pela qual fiquei tanto tempo. Fiquei lá por quase 8 anos por causa dos valores da empresa, que eu ainda amo e que genuinamente são relacionados aos meus interesses

pessoais. Por exemplo, trabalhando para suas comunidades mais próximas e retribuindo a eles para que possam crescer com você. Além disso, a filosofia de código aberto - contribuindo com outros projetos abertos para ajudá-los a melhorar. Essa cultura de compartilhamento é importante e altamente motivadora para mim como ser humano.

E, finalmente, eu também fiquei lá por causa das pessoas. Eles me disseram na entrevista que ‘preferimos contratar pessoas legais (de bom comportamento) que possam aprender o trabalho do que contratar pessoas inteligentes que não se comportam bem’ (tradução direta do espanhol). E acho que esse é um ponto justo. Se você contrata um astro do rock que produz muito, mas não consegue trabalhar como parte de uma equipe, ninguém pode tocar seu código, ou coisas assim, mas o que acontece quando essa pessoa sai? E os problemas que essa pessoa produz valem a pena em comparação com a produtividade? O processo de recrutamento foi ótimo porque eles contrataram muitas pessoas legais e, ao mesmo tempo, inteligentes, posso dizer honestamente que conheci alguns engenheiros incríveis lá: pessoas muito motivadas, muito qualificadas e muito agradáveis."

Parte cinco: Hábitos de equipe

Hábito 10: Ouvir ativamente

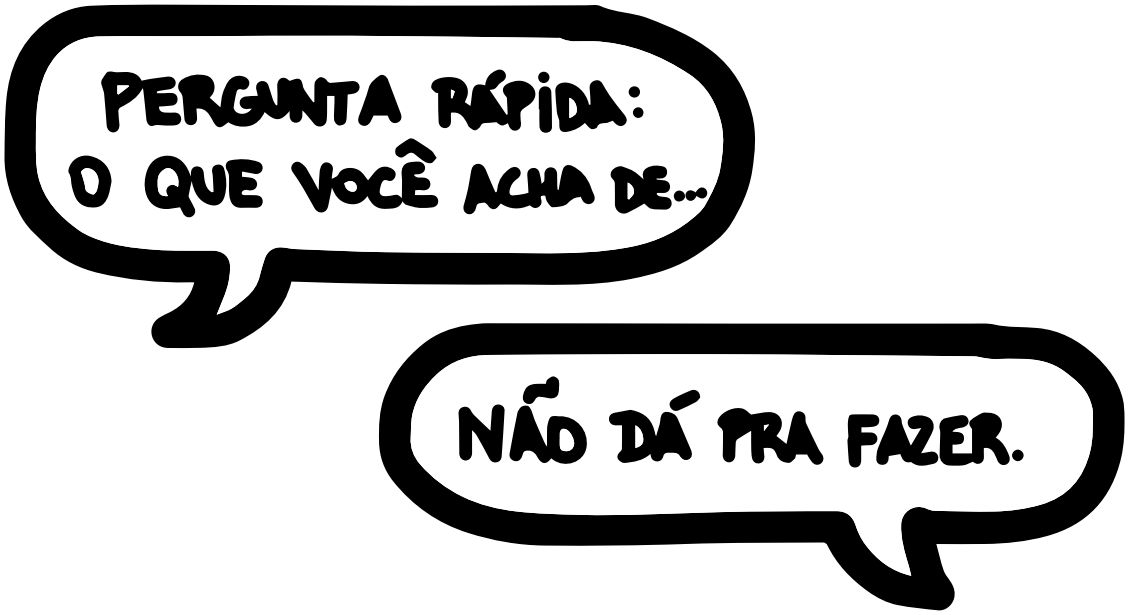
“Sábios falam porque têm algo a dizer. Tolos porque têm que dizer alguma coisa.” - Platão

Imagine uma empresa inteira composta apenas por engenheiros de software. Imagine todas as decisões sendo tomadas com base em métricas. Imagine um lugar sem subjetividade, onde tudo é resolvido com puro pensamento racional e analítico. Parece um ambiente perfeito, no entanto, não é assim que as empresas funcionam, e não é assim que os seres humanos funcionam.

Uma empresa é composta por uma variedade de profissionais, como designers, gerentes de projeto, vendedores, executivos e outros profissionais. Além disso, essas pessoas são de diferentes idades, têm diferentes origens, vêm de diferentes culturas. Às vezes, as decisões são tomadas com números e outras com sentimentos. Os problemas são criados e resolvidos, com ou sem você. E mesmo se você for o fundador da empresa, haverá muitas coisas sobre as quais você simplesmente não terá controle ou visibilidade.

Uma empresa não é um logotipo, não é um prédio, é apenas uma aglomeração de pessoas e, se você quiser se tornar eficaz lá, precisará se comunicar bem, mesmo se for introvertido.

Conversas geralmente são bidirecionais, mas isso não significa que você deva falar o tempo todo apenas por falar. Um equívoco que muitas pessoas cometem é acreditar que precisamos **ouvir para responder**, enquanto, na realidade, precisamos **ouvir para entender**. Compreender alguém é muito mais importante do que responder à alguém. Qualquer um consegue responder, poucos conseguem entender.



Conversas podem ser informativas, divertidas e às vezes combativas. Ao ter uma “hierarquia mais alta” em uma conversa difícil, é muito fácil mostrar falta de paciência, ser agressivo e querer mostrar poder e autoridade. Por outro lado, ao ter uma “hierarquia mais baixa” em conversas difíceis, é muito fácil parar de prestar atenção ou colocar a culpa em outra pessoa. Goste ou não, todos nós temos que lidar com conflitos e esses são os momentos em que você pode mostrar o seu melhor ou o pior.

Independentemente da conversa que estiver tendo, pratique o hábito de ouvir ativamente. Não apenas ouvir para responder, quero dizer **realmente ouvir**, realmente tentar entender o que a outra pessoa está tentando transmitir.

Se você é um desenvolvedor front-end, como está o relacionamento com o designer da sua equipe? O que acontece quando ele(a) propõe algo que parece totalmente louco e extremamente difícil de implementar? Você assume uma posição reativa e imediatamente tenta rejeitar a proposta?

Se você é um desenvolvedor back-end, como está o relacionamento com o gerente de projetos da sua equipe? O que acontece quando você perde um

prazo e é pressionado a fazer algo? Você fica na defensiva e começa a reclamar?

Se você é arquiteto de software, como está o relacionamento com os desenvolvedores juniores de sua equipe? O que acontece se eles implementarem algo completamente diferente do que você imaginou? Você tem empatia por seus problemas e desafios?

Lembre-se das palavras do grande comediante Robin Williams:

> “Todo mundo que você conhece está travando uma batalha que você não conhece. Seja gentil. Sempre.”

```
// TODO

Na sua próxima reunião, escolha praticar suas habilidades de
escuta. Em vez de ser o primeiro a dizer algo, espere até que
todos compartilhem suas ideias e seja o último a falar. Isso
dará a todos os outros a sensação de serem ouvidos, e você
terá o benefício de ouvir as ideias de todos antes de
compartilhar suas próprias opiniões.
```

 # Perguntas e Respostas: Como você se comunica com outros membros da equipe?

Manuel De La Peña (Elastic):

> "Eu trabalho completamente remoto há mais de 3 anos e parcialmente remoto desde 2011, estou convencido de que a única maneira eficiente de se comunicar em trabalhos remotos é fornecer o máximo de contexto possível. Para uma empresa que é 100% distribuída (remoto + assíncrono), existem não apenas engenheiros, mas também RH, vendas e consultores trabalhando remotamente, portanto a comunicação é essencial.
>
> Além disso, somos desenvolvedores, por isso estamos mais acostumados a nos comunicar com código. Por esse motivo, um pull request ou ticket deve fornecer o contexto para o revisor continuar em seu próprio fuso horário, que pode ser uma diferença de 9 horas em relação ao desenvolvedor que enviou o pull request. Criar templates para tickets ou pull requests geralmente são necessários para mim. Esses templates devem abranger os itens que podem fazer com que o remetente pense cuidadosamente sobre o que está enviando.
>
> Finalmente, é super importante para mim falar sobre código, nunca sobre a pessoa, porque você não sabe o que está acontecendo do outro lado da tela. Talvez uma criança precise de muitos cuidados por algum motivo médico, ou talvez o desenvolvedor simplesmente tenha tido um

dia ruim … quem sabe! Portanto, por esse motivo, é vital ser sempre educado e gentil. Isso é algo que tento seguir como mantra. E me sinto super sortudo por ter encontrado isso na maioria das equipes em que trabalhei."

Fernando Tadashi (Adobe):

"Aprendi que a comunicação é uma arte e precisa ser praticada todos os dias, pois nos proporciona a oportunidade de falar, nos ensina a ouvir e a criar empatia.

Meu dia a dia é repleto de reuniões, e-mails e mensagens por meio de ferramentas de bate-papo, e eu uso a técnica ‘Zero Inbox’ e ‘Bullet Journal’ para organizar minhas atividades com meus contatos."

Berg Brandt (Amazon, Ex-Yahoo):

"Ao se comunicar com as pessoas como um líder, comunicação eficaz não é necessariamente um processo eficiente. Eficiência é fazer mais com menos, eficácia é fazer com sucesso. Para transmitir uma mensagem com sucesso, você quase sempre precisa repeti-la. Algumas vezes. O contexto também desempenha um papel importante. Quanto menos o público se identifica com um contexto, mais ineficiente é a comunicação eficaz.

Comunicação eficaz desempenha um papel imenso na formação de equipes. Como líder, especialmente ao montar uma equipe do zero ou contratar novos profissionais, você precisa se certificar de que está sempre preenchendo a lacuna de contexto na comunicação. As pessoas vêm de diferentes origens, trazem diferentes experiências e habilidades para o time e até falam diferentes idiomas nativamente (como eu, português, em um ambiente em inglês). É por isso que, quando alguém se junta à nossa equipe, considero que uma equipe completamente nova se formou. É por isso que nos esforçamos bastante para preencher a lacuna de contexto e atualizar o novo membro.

Em suma, comunicação eficiente exige muito esforço, mas é uma habilidade essencial para um líder (gerentes e qualquer pessoa em

posição de liderança). Comunicação eficiente, mas não eficaz, é muito melhor que o oposto e vale a pena. Obviamente, é tarefa de um líder preencher a lacuna entre eficácia e aumentar a eficiência ao longo do tempo."

Hábito 11: Não subestime

“Só vai levar 5 minutos” — Todo desenvolvedor do planeta

Como desenvolvedores, somos pagos para resolver problemas. Toda tarefa é um problema completamente novo. Mesmo que você tenha trabalhado exatamente na mesma tarefa no passado, sempre há detalhes envolvidos em cada tarefa. Talvez você precise trabalhar com um membro de uma equipe diferente, talvez o processo de desenvolvimento tenha mudado, talvez a arquitetura do software não seja a mesma. Seja qual for o caso, não existe a possibilidade de resolver o mesmo problema adotando a mesma solução.

Além disso, todas as tarefas estão associadas a um desafio de negócio. Alguns não afetam tanto os usuários e eles ainda podem usar o software sem muitos problemas. Enquanto que outros desafios de negócios sejam extremamente sensíveis e podem estar afetando os usuários em produção nesse exato momento.

Não importa o quão experiente você seja, não importa qual seja o seu cargo, não importa para qual empresa você trabalha, mais cedo ou mais tarde alguém irá te perguntar: *“Quanto tempo esta tarefa vai demorar?”* Goste ou não, chefes ou clientes precisam de prazos e você precisará responder alguma coisa a eles.

Se for uma tarefa trivial, provavelmente diremos que levará uma hora, mas três horas depois ainda estaremos trabalhando nisso. Quando achamos que é uma tarefa maior, podemos dizer que levará três horas, mas então o dia inteiro se foi e de repente ainda estamos terminando alguns detalhes finais. Por que isso acontece? Por que ainda somos tão ruins em estimativas de software?

Há muitas razões para isso, vamos ver se alguma delas lhe parece familiar.

1. **Queremos impressionar os outros:** Cada um de nós quer ser reconhecido pelo nosso trabalho. Queremos nos sentir importantes, queremos nos sentir valiosos, queremos nos sentir superiores. Então, quando chega a hora de fazer uma estimativa, dizemos que vai levar um curto período de tempo para impressionar os outros. Dessa forma, esperamos que eles nos vejam de maneira diferente, que nos vejam como alguém que pode terminar as coisas mais rapidamente do que outros.

2. **Esquecemos que não se trata apenas de código:** Quando recebemos uma tarefa, a primeira coisa que queremos fazer é ir diretamente para o código. Isso é normal, o desenvolvimento é definitivamente a parte mais divertida do trabalho, mas devemos lembrar de todo o ciclo de

vida de desenvolvimento de software. Ainda precisamos de tempo para compilar, criar testes de unidade, documentar as correções, resolver conflitos, responder os comentários, participar das reuniões diárias etc.

3. **Não nos concentramos em uma coisa:** Estamos frequentemente trabalhando em vários projetos ao mesmo tempo, esperando que uma pessoa termine a parte dela, para que possamos continuar com a nossa parte. Como resultado, estamos constantemente sendo puxados para dentro e fora do contexto. Há um custo cognitivo real associado à constante necessidade de reorientar sua tarefa atual. Em outras palavras, leva tempo para *"focar."*

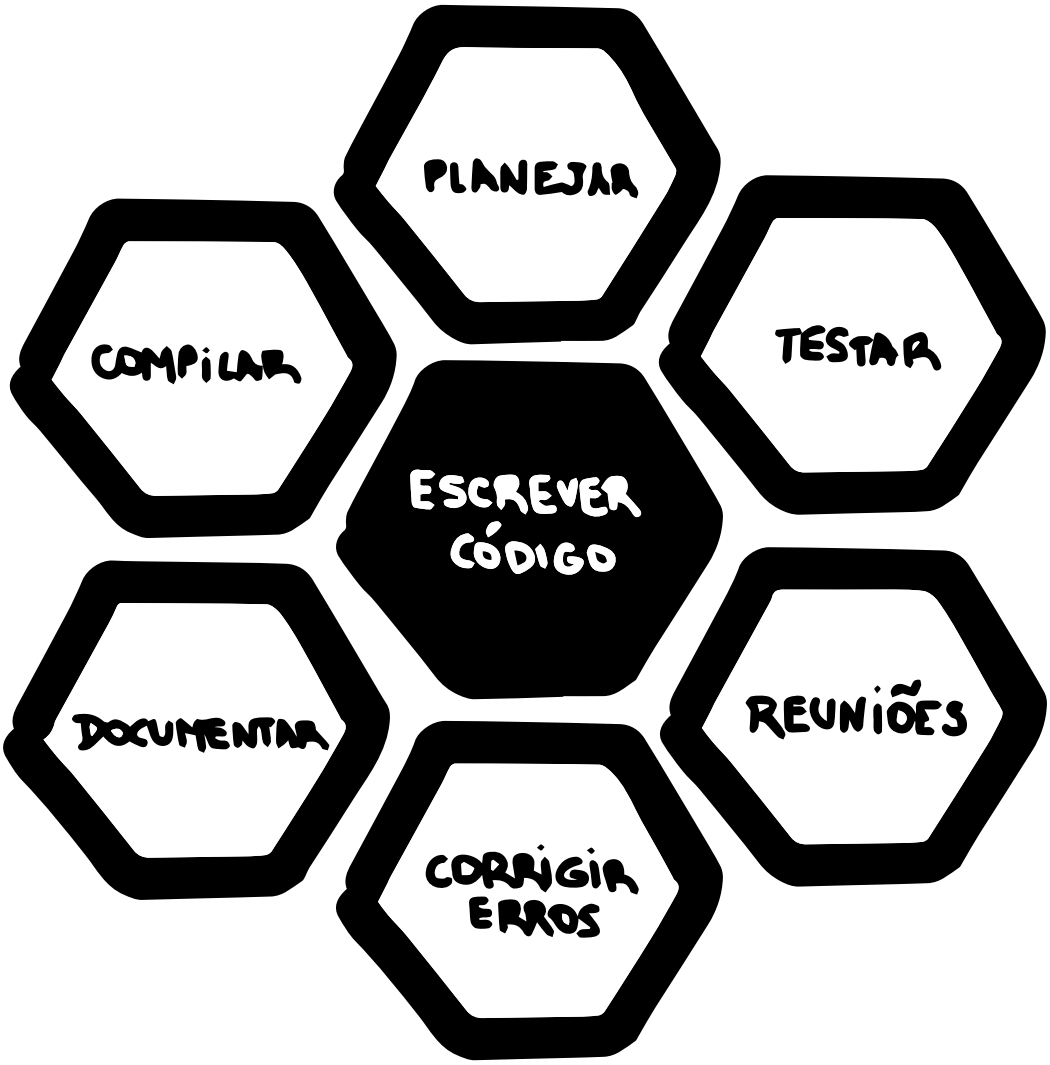
4. **Acreditamos que todos são iguais:** Se você é o líder da equipe, é possível que esteja dando estimativas não apenas para você, mas para o resto de sua equipe. É comum pensar em termos de tempo médio que levaria para concluir uma tarefa. No entanto, devemos entender que estimativas são estritamente individuais. O tempo que leva para o Jim concluir a tarefa X é diferente do tempo que o Dwight pode levar para concluir a mesma tarefa X.

5. **Não conseguimos lidar com a pressão:** Às vezes o problema não está relacionado diretamente com a gente. Clientes externos, gerentes de projeto, líderes de produtos ou mesmo o CTO podem estar enfrentando uma pressão extremamente alta de outros e podem estar transferindo a mesma pressão para você. Eles precisam fazer algo o mais rápido possível e é difícil mostrar a eles que não será tão fácil assim.

Na nossa sede de impressionar os outros, na nossa incapacidade de dizer "não" aos outros, na nossa falta de planejamento para atividades não relacionadas à codificação, acabamos subestimando as tarefas, perdendo prazos e perdendo a confiança. Então, qual é a solução? Como podemos melhorar nas estimativas?

Primeiro, devemos admitir que é impossível ser 100% preciso com estimativas. Problemas inesperados vão ocorrer, bugs ocultos sempre aparecerão, membros da equipe deixarão a empresa. Também devemos entender que nunca teremos a imagem completa diante de nós. Os requisitos de negócios não serão totalmente documentados, os critérios de

aceitação nem sempre estarão completos e novas informações vão aparecer no meio do processo de desenvolvimento.



Agora você pode estar se perguntando: *se existem tantas variáveis a serem consideradas e é impossível fornecer estimativas precisas, por que tentar?* A resposta é: você se aproximará cada vez mais dela.

No livro *Estimativa de software: Desmistificando a arte negra* de Steve McConnell, ele recomenda:

- Separar tarefas grandes em tarefas menores, melhora a precisão das estimativas. Divida estimativas em tarefas que exigirão não mais de 2

dias de esforço.
- Estimativa em escalas: pior, mais provável e melhor caso para a tarefa.
- Para o desenvolvimento inicial com uma nova linguagem/ferramenta, em comparação com o desenvolvimento usando uma linguagem/ferramenta familiar, adicione um aumento de 20% a 40% no esforço.
- Documente e comunique as suposições em sua estimativa.
- As melhores técnicas de estimativa para pequenos projetos tendem a ser “de baixo para cima”, com base em estimativas criadas por pessoas que realmente farão o trabalho.

Independentemente das técnicas de estimativa que você usar (*T-shirt sizing*, *Story points* com escala Fibonacci ou *Poker planning*), trate as discussões de estimativa como solução de problemas e não como negociação. **Reconheça que você e as partes interessadas no projeto estão do mesmo lado da mesa.**

Também é importante saber que programadores experientes não têm necessariamente experiência em fornecer estimativas. Um desenvolvedor que não esteja envolvido no processo de estimativa simplesmente não será bom em estimativa. Além disso, se o tempo real gasto em uma tarefa não for medido e comparado com a estimativa, não haverá feedback para aprender.

```
// TODO

Da próxima vez que alguém solicitar uma estimativa, pegue um
Post-it e anote sua resposta. Quando estiver pronto para
iniciar a tarefa, fique de olho no relógio e veja quanto
tempo leva para concluir. Depois que a tarefa estiver
concluída, anote quantas horas realmente levou para terminar.
Repita o mesmo para a próxima tarefa.
```

 # Perguntas e Respostas: Como você faz a estimativa de tarefas?

Blake Williams (GitHub):

> “Acho que a melhor estimativa é não ter estimativa. Às vezes, dizer ‘não’ não é uma opção. Quando não posso dizer não, faço o que presumo que qualquer outro engenheiro faça nessa situação, adivinhar. Adivinhar não significa que você deve gritar YOLO e chutar um período de tempo aleatório, mas use suas experiências passadas e seu melhor julgamento para fazer um palpite mais assertivo possível. Quando tiver essa estimativa, multiplique-a por 2. Vi muitas pessoas irritadas quando o software é entregue com atraso, mas nunca vi alguém bravo com a entrega antecipada de software. Se você viver de ‘prometer menos e entregar mais’ quando se trata de estimativas de software, estará em boa companhia.”

Caio Gondim (New York Times):

> "Imagine que você comprou um novo teclado mecânico online. A hora prevista de chegada é em 4 dias. Mesmo que você desejasse que ele chegasse amanhã, 4 dias estão ok. Talvez eles não o tenham no armazém principal ou o produto seja fornecido do exterior. Portanto, 4 dias não é ruim.
>
> E então, em 2 dias, você volta do trabalho, entra em sua casa e … boom! Seu pacote está lá. Era para chegar dois dias depois. Você

estava mentalmente preparado para esperar mais. Mas é tão bom ter recebido ele antes hora!

Agora imagine essa outra situação - você compra o mesmo teclado mecânico e o tempo para ser entregue é para amanhã. Você conta para os seus amigos e faz planos de fazer um unboxing com eles.

É o fim do dia e seu pacote não chegou. A loja online diz que houve um atraso, mas deve chegar amanhã. E de fato chega no dia seguinte. Você fez todos esses planos, você tinha suas expectativas super altas, você está feliz porque o pacote chegou, mas não tanto.

No final, o tempo de entrega para as duas situações foi o mesmo: 2 dias. No entanto, em qual deles o cliente ficou mais feliz? Penso que o cliente na situação anterior ficou muito mais satisfeito.

Deveríamos fazer o mesmo na estimativa de nossas tarefas. Superestimar e entregar antes do prazo. Essa é a melhor maneira de gerenciar expectativas e se prevenir contra os imprevistos que apareçam no meio do caminho."

Luciano Sousa (Shopify):

“Estimar uma tarefa não é algo que me motive, mas como desenvolvedor, todos nos perguntam sobre isso o tempo todo. Quando tenho algum conhecimento sobre a tarefa, sempre tento compará-la com outros problemas similares que lidei no passado. É claro que este não é um processo 100% confiável, e sempre tento explicar ao meu chefe que tudo o que queremos ‘estimar’ nem sempre é confiável.”

Hábito 12: Especialista vs. Generalista

“O próximo Bill Gates não criará um sistema operacional. O próximo Larry Page ou Sergey Brin não criará um mecanismo de busca. E o próximo Mark Zuckerberg não criará uma rede social. Se você está copiando esses caras, você não está aprendendo com eles”. - Peter Thiel

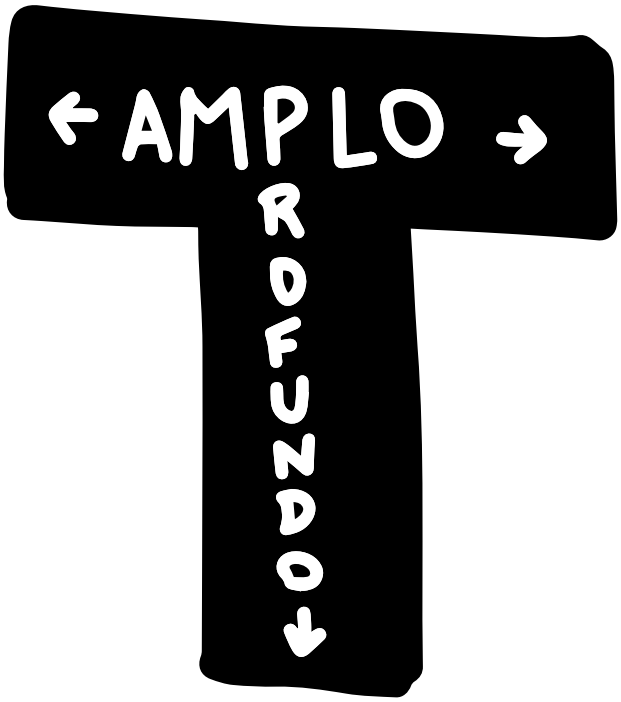
Quando você inicia sua carreira como engenheiro de software, há muito conhecimento técnico que você precisa absorver em um período relativamente curto. Como em qualquer carreira, você provavelmente começará como estagiário ou junior. Alguns meses depois, à medida que você for se familiarizando com o funcionamento das coisas, irá entender como escrever código e descobrirá que a linguagem de programação não é mais um grande problema.

Eventualmente, você fará a transição para uma posição de nível pleno. Este é o período em que você se sentirá mais confortável. Você vai começar a experimentar técnicas sofisticadas e tudo vai parecer fluir bem. Alguns anos depois, você pode ser promovido a uma posição sênior. Nesse momento, você entende como as coisas funcionam em um nível mais profundo, sabe como construir uma arquitetura escalável e poderá orientar novos desenvolvedores na equipe.

Depois de se tornar um sênior, surge um novo dilema: que caminho você deve seguir?

- Tornar-se um especialista, alguém ciente de todos os detalhes de um determinado assunto?
- Ou tornar-se um generalista, alguém capaz de abordar uma variedade de assuntos diferentes?

Essa é uma pergunta comum que muitos desenvolvedores se fazem. Assim como Neo, no filme *Matrix*, pode parecer que você só tem dois caminhos para escolher. Como você sabe qual caminho/pílula é melhor? Você toma a pílula vermelha? Ou a pílula azul? Você deve ser amplo? Ou você deve se aprofundar em um único assunto?



A resposta vai depender de uma autoanálise. Você precisa conhecer a si mesmo, olhar para dentro e pensar quais tipos de tarefas o motivam mais durante o seu dia de trabalho. Dê uma olhada nesta lista e veja com quem você mais se identifica.

Especialista - Prós

- Você pode conseguir um salário muito bom em um campo específico.
- Você normalmente vai ser reconhecido como uma autoridade técnica em determinado assunto.
- Você precisa se manter atualizado(a) em apenas uma plataforma/linguagem.
- As empresas estão sempre procurando pessoas especializadas em um campo específico.

Especialista - Contras

- Pode ser difícil encontrar uma posição se sua especialização for muito limitada.

- Se a tecnologia escolhida correr o risco de se tornar ultrapassada, é possível que você tenha que começar de baixo novamente e isso vai ser difícil.
- É possível que o seu conhecimento seja muito dependente da empresa, o que dificultará a aplicação das mesmas habilidades em outro lugar.
- Se houver necessidade urgente de usar outra tecnologia, provavelmente você vai demorar mais para concluir uma determinada tarefa.

Especialista - Oportunidades de Trabalho

- Cargos sênior, especialmente em grandes empresas.
- Projetos de pesquisa em universidades.
- Freelancer em um campo específico.

Agora, vamos dar uma olhada em como essas listas se comparam às mesmas de um generalista.

Generalista - Prós

- Você está acostumado a aprender novas tecnologias rapidamente.
- Existe uma grande variedade de oportunidades que você pode buscar em diferentes setores.
- Se você precisar de um novo emprego, é mais flexível e fácil de fazer a transição.
- Se você precisar mudar o contexto para um tipo diferente de tarefa, há chance de encontrar uma solução mais rápido.

Generalista - Contras

- É difícil se manter atualizado em várias linguagens e tecnologias diferentes.
- É possível que você demore muito para resolver problemas complexos e específicos da tecnologia.
- É difícil alcançar uma posição de liderança se você estiver mudando de campo constantemente.
- Embora seja mais fácil encontrar uma nova posição, pode ser difícil mostrar que você é o melhor candidato.

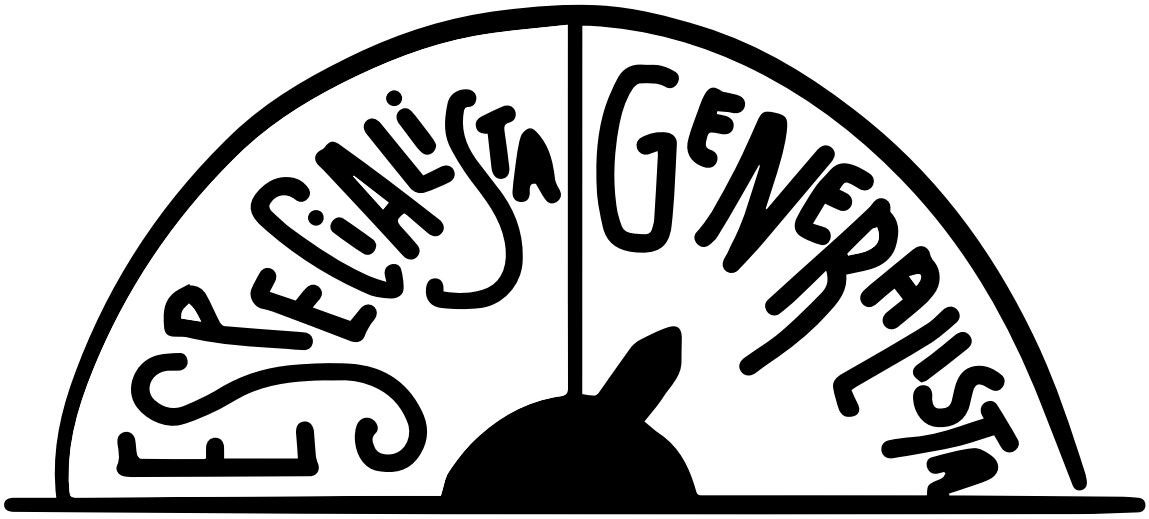
Generalista - Oportunidades de Trabalho

- Startups/Empresas em estágio inicial.
- Oportunidades de consultoria.
- Começar seu próprio negócio.

Espero que essas listas ajudem a guiar seu processo de decisão.

A única coisa que notei ao longo dos anos é que os líderes tendem a ser generalistas. Eles podem mudar de curso mais facilmente e são mais flexíveis. Um bom exemplo é Elon Musk, que construiu quatro empresas em quatro campos separados (software, energia, transporte e aeroespacial). Isso não significa que ele não precisou ser especialista em algo primeiro. Significa apenas que ele foi capaz de adaptar o conhecimento que possui a uma ampla variedade de situações.

No final das contas, esses não são estados permanentes. Às vezes, em sua carreira, você se concentra como um louco e se aprofunda em um determinado tópico, mas, outras vezes, se amplia e resolve problemas que nunca viu antes.



Meu conselho é criar o hábito de se perguntar: *"Como posso ajudar melhor minha equipe?*"

Se isso significa aprender algo completamente novo, que assim seja. Se isso significa não codificar, que assim seja. Quando o sucesso individual não é uma prioridade, você pode ir muito além do que imagina.

```
// TODO

Examine seu dia de trabalho. Examine sua semana de trabalho.
Quais são as partes que mais te deixam animado? Que tipo de
trabalho você prefere fazer? Examine sua empresa. Examine sua
equipe. No que eles estão com mais dificuldade? Existe alguma
coisa com a qual você possa ajudar?
```

 # Perguntas e Respostas: Você se considera um especialista ou generalista? Qual é o melhor na sua opinião?

Daniel Buchner (Microsoft):

> "A maioria das pessoas já ouviu essa frase clássica antes: ‘Devo me especializar em um nicho e me tornar um especialista, ou me esforçar para ser um generalista que se concentra em aprender habilidades mais genéricas.’ Costumo discordar disso e pensar em uma abordagem diferente: Aprenda a aprender de uma maneira que permita que você se aprofunde sobre um novo assunto o mais rápido possível e se torne funcionalmente capaz na área que você deseja o mais rápido possível. Isso não significa que você precisa entender tudo sobre um tópico da mesma maneira que alguém que trabalha há anos, isso significa que você precisa entender os pontos mais críticos que orientam as decisões e vê-los da maneira que um veterano os vê. Em termos de habilidades aplicadas, tento garantir que eu possa me ‘sustentar’ se for solicitado a produzir algo em uma área de tecnologia que estou aprendendo.
>
> Para fazer isso, procure mentores que estejam dispostos a responder suas perguntas e supere a sensação de que suas perguntas são ingênuas - algumas delas provavelmente são, mas um bom mentor fará você se sentir à vontade para fazer perguntas. Frequentemente, peça por revisões e, quando receber um feedback, se concentre em absorver aquilo, e não apenas em aprender a resposta específica para um problema isolado. Faça essas coisas com a humildade de um aluno, e

você poderá se tornar o melhor de um generalista e especialista na área em que está trabalhando."

Silvio Gustavo (Spotify):

"Eu sempre quis me tornar um desenvolvedor mobile, então comecei a aprender Android e tive a chance de trabalhar com ele. Estudei muito, segui as melhores pessoas da área e me especializei profundamente. Acho que foi uma boa estratégia e isso possibilitou que eu trabalhasse como engenheiro Android em uma grande empresa, algo que sempre sonhei. Acredito que engenheiros mais especializados têm mais chances em certas empresas, pois possuem um profundo conhecimento para resolver problemas em algumas áreas específicas.

No final, acredito que ser um especialista é uma coisa boa, mas ainda mais importante é ter a capacidade de trazer todo esse conhecimento e experiência quando você precisar aprender e trabalhar com novas tecnologias."

Lais Andrade (Google):

"Não tenho tanta certeza de qual deles eu seria. No passado, desenvolvia somente o lado do servidor, depois passei a trabalhar em um aplicativo Android e atualmente estou trabalhando em frameworks Android. O ambiente mudou drasticamente entre esses projetos, assim como as tecnologias e linguagens de programação que eu usei, mas havia uma coisa em comum, que era meu foco: o design e o desenvolvimento de APIs. Mesmo quando eu trabalhava no desenvolvimento de um aplicativo, eu ainda focava nas partes estruturais de baixo nível do aplicativo: criando componentes, widgets e bibliotecas para serem usados por toda a equipe no desenvolvimento e criação de novas funcionalidades.

Eu foco em entender o que me motiva na engenharia de software e tento investir nessa área para continuar aprendendo e melhorando minhas habilidades. Não significa necessariamente trabalhar especificamente com as mesmas tecnologias ou linguagens. Significa encontrar o que me faz sentir melhor sobre o meu trabalho, que me

interessa e me motiva, e depois usar isso para me tornar o que mais admiro."

Parte seis: Hábitos de vida

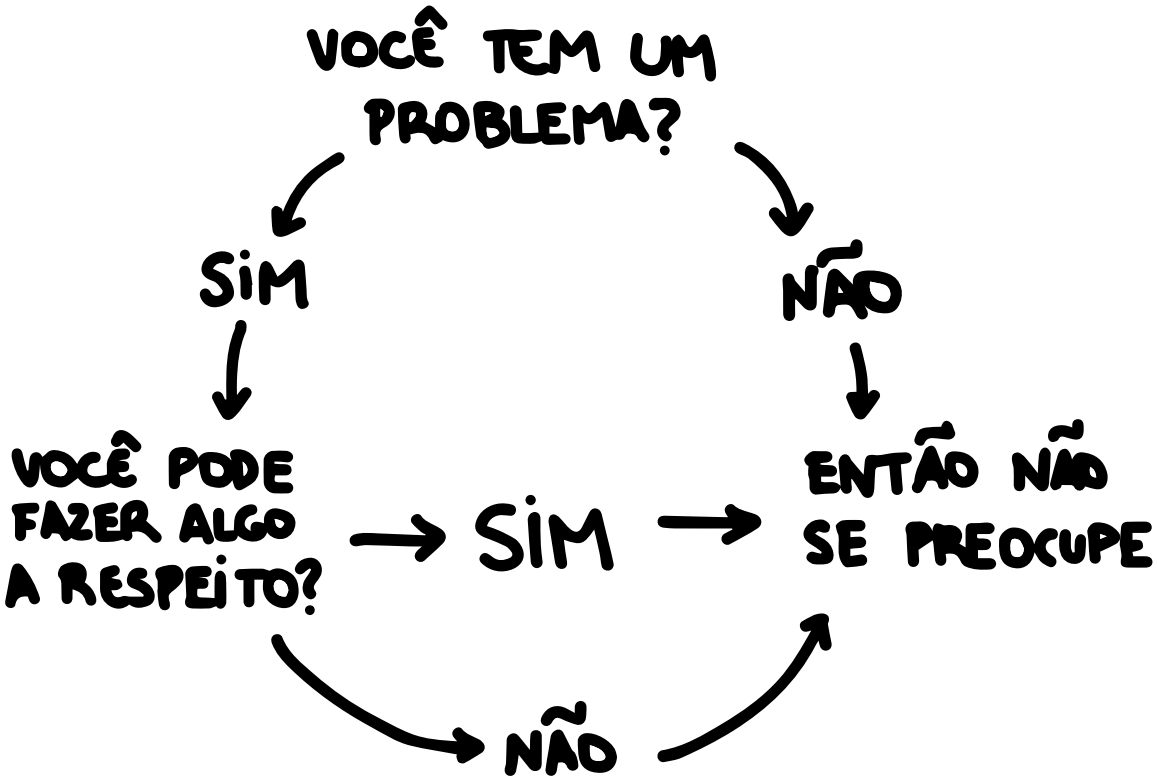
Hábito 13: Controle suas variáveis

“A única coisa sobre a qual você às vezes tem controle é a sua perspectiva. Você não tem controle sobre sua situação. Mas você tem a escolha sobre como vê-la.” — Chris Pine

Enquanto escrevia este livro, o mundo estava passando pela maior epidemia que já enfrentou. O COVID-19, ou Coronavírus, é diferente de tudo o que já passamos em nossas vidas. Milhares de pessoas estão morrendo, tudo o que vemos nas notícias é alarmante e a economia alcançou seus números mais baixos em muitos anos.

Devo admitir que fiquei extremamente ansioso com toda essa situação. Ver pessoas usando máscaras em todos os lugares, olhar as prateleiras vazias nos supermercados, todos trabalhando em casa sem qualquer previsão de quando as coisas voltariam ao normal.

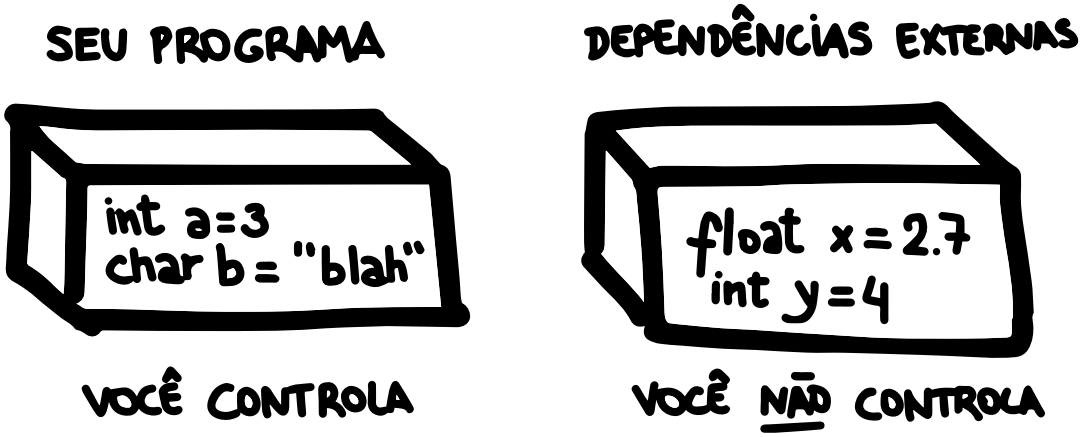
Levei alguns dias para perceber que não havia motivo para entrar em pânico. Sim, havia certas coisas que eu poderia fazer como indivíduo para ajudar a impedir que o vírus se espalhasse ainda mais, como lavar as mãos com mais frequência, evitar aglomerações públicas e cancelar viagens aéreas. No entanto, não posso controlar o mercado de ações, não posso controlar a política e definitivamente não posso controlar o vírus. A única coisa que eu sabia que podia controlar é como reagiria a toda essa situação. E foi aí que comecei a escrever este livro.



Embora este seja um evento sem precedentes em nossas vidas, a história está aqui para nos ensinar lições sobre o que fazer quando esses eventos acontecem. Não devemos esquecer que não é a primeira vez que uma epidemia atinge o mundo. Vamos considerar Isaac Newton, por exemplo, o grande físico e matemático que desenvolveu os princípios da física moderna. Ele tinha 23 anos quando a Grande Praga de Londres aconteceu. Esta foi uma grande epidemia da peste bubônica que ocorreu na Inglaterra e matou cerca de 100.000 pessoas.

Em 1665, Newton era apenas mais um estudante universitário no Trinity College, Cambridge. Conforme descrito no livro *Isaac Newton*, de Gale Christianson, quando a praga ameaçou Cambridge, Newton fugiu para a fazenda da família. Durante esse período, ele formulou sua teoria da gravidade, uma nova teoria de luz e cálculo. Newton retornou a Cambridge em 1667 com suas teorias em mãos. Dentro de seis meses, ganhou uma bolsa acadêmica. Dois anos depois, ele se tornou professor.

Como engenheiros de software, estamos acostumados a escrever código, criar funções e lidar com diferentes variáveis. Essas variáveis podem ser feitas de diferentes tipos, como números inteiros, seqüências de caracteres, booleanos, etc. Na computação, temos o poder de alterar os valores dessas variáveis a qualquer momento. Na vida, existem trilhões de variáveis que são constantemente criadas e alteradas por outras pessoas. Não temos controle absolutamente nenhum sobre essas variáveis, mas ainda assim elas afetam nossas vidas. A maneira como reagimos a essas mudanças é o que determina nosso caráter.



Variáveis que você **pode** controlar:

- Seus pensamentos
- Quem são seus amigos
- O que você come e bebe
- Como você gasta seu dinheiro
- O que você faz com o seu tempo
- Como você trata seu corpo
- Quanto você aprecia as coisas que você já tem

Variáveis que você **não pode** controlar:

- O clima
- A economia
- A saúde pública

- Como as pessoas tratam você
- O que as pessoas pensam de você
- O que as pessoas gostam ou não
- O que aconteceu no passado

Pare de perder tempo com variáveis que estão fora de seu controle.
Concentre-se nas variáveis que você pode controlar.

```
// TODO

Quais são as variáveis com as quais você está preocupado
agora? Quantas delas estão fora do seu controle?
```

 # Perguntas e Respostas: Como você lida com eventos externos que afetam sua vida no dia-a-dia?

Fabio Costa (GoDaddy, Ex-Facebook):

> “Acho que é impossível não mencionar a pandemia do COVID-19 ao responder a essa pergunta. Senti pena de todas as pessoas que perderam o emprego, mas pelo lado positivo, estou muito agradecido por podermos trabalhar remotamente em tecnologia. Sair da maioria das mídias sociais (hoje em dia sou um usuário leve do Twitter) também me ajudou a evitar essa ansiedade contínua que às vezes me fazia perder o foco das coisas que realmente importam. Então, em resumo, acho que diria: permaneça positivo(a) e evite as mídias sociais”

Netto Farah (Segment):

> "Tento otimizar blocos de 2-3 horas de trabalho intenso e concentrado, que geralmente valem um dia inteiro de procrastinação. Eu guardo o telefone, silencio todas as notificações, coloco os fones de ouvido e tento fazer o máximo de código/experimento/escrita/revisão possível. Em seguida, tento ficar mais relaxado e me entrego a algumas coisas de que gosto:
>
> - Aproveitando o meu tempo para cozinhar e desfrutar do meu almoço
> - Ir à academia no meio do dia
> - Sair para correr"

Loiane Groner (Citibank):

"Todas essas coisas são fatores externos que você não pode controlar. Às vezes, pode até sugar toda a sua energia e não fará nada bem a você. Nosso tempo e energia são limitados, por isso sempre tento me concentrar nas coisas que posso controlar. É claro, é mais fácil falar do que colocar em prática, mas o melhor que podemos fazer nesse caso é educar as pessoas mais próximas de nós.

Nesses dias que me preocupo com fatores externos e não estou me sentindo bem, tento fazer algo que me traga alegria, como dar um passeio em um parque, entrar em contato com a natureza, assistir um programa de TV leve, passar tempo com os meus familiares, para que possa clarear a cabeça. Quando me sinto melhor, me concentro novamente no que tenho que fazer para alcançar meus objetivos."

Hábito 14: Pare de esperar

“Sofremos mais frequentemente na imaginação do que na realidade.” — Seneca

Uma característica, presente em absolutamente todos os seres humanos, é o fato de nunca estarmos satisfeitos com a vida. Pode reparar, dos mais pobres aos mais ricos, sempre existe algo faltando para conquistar ou algo para reclamar.

“Meu trabalho é péssimo, preciso trocar de emprego…”

“Este país é horrível, preciso me mudar o mais rápido possível…”

“Eu tenho uma ideia incrível, preciso colocar em prática…”

E não há nada de errado nisso. Desejar um futuro melhor para si mesmo é um princípio básico do ser humano. Essa é a razão pela qual você ainda está lendo este livro. O problema é o que sucede essas frases.

“…mas tenho medo de mudar.”

“…mas eu não sei outro idioma.”

“…mas eu não tenho tempo.”

O fato é que a maioria das pessoas tem a capacidade de reconhecer o que as incomoda, mas apenas algumas têm coragem e determinação para enfrentar esses desafios.

A única coisa que impede você de conseguir algo é você mesmo.

Quer mudar de emprego? Então, quais são as principais empresas que você gostaria de trabalhar? Quantas vezes você entrevistou em cada uma delas? Que tipo de habilidades você precisa aprender para conseguir esse emprego? Quais livros você está lendo para preencher essa lacuna?

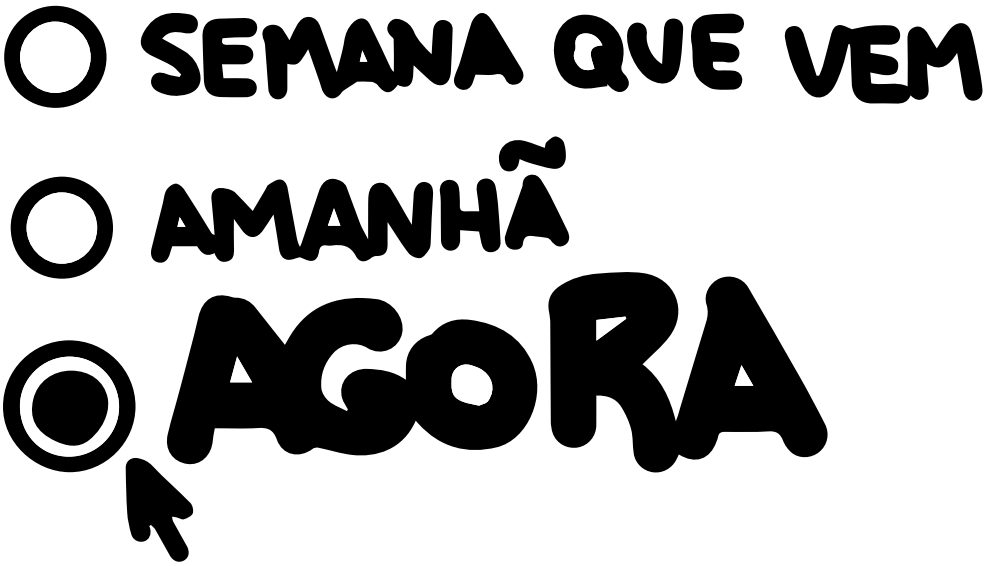
Você quer morar no exterior? Então quantos dias da semana você está investindo para aprender o idioma desse país? Quanto você economiza por mês para pagar pelos custos de mudança? Que tipo de visto você precisa para morar nesse país? Quais documentos você precisa?

Você quer investir em uma ideia? Então quantas horas você está se dedicando a este projeto? Você pode acordar 1 hora antes? Você pode dormir 30 minutos depois? E o tempo de almoço, você pode diminuir?

Eu entendo, todos nós temos responsabilidades, todos nós temos contas para pagar e todos nós temos pessoas que dependem de nós. Nada na vida é fácil e você sabe muito bem disso.

Mas se por um segundo você parar de ser romântico(a) e começar a ser prático(a), notará que por trás de cada desculpa existe uma alternativa. Por trás de cada objetivo, há uma série de tarefas que podem ser feitas hoje.

Você acabou de ler este livro. Você pode fechar ele e continuar com sua vida, ou pode tentar adotar esses hábitos a partir de hoje. Então, o que você vai fazer? O que você está esperando?



Parte sete: O fim

Agradecimentos

“E agora que o fim está próximo. E então eu encaro a cortina final. Meu amigo, eu vou dizer isso claramente. Vou expor o meu caso, pois dele eu tenho certeza. Eu vivi uma vida cheia. Eu viajei por toda e qualquer estrada. E mais, muito mais que isso. Eu fiz do meu jeito.” — Frank Sinatra

Eu gostaria de começar agradecendo aos meus avós. Vocês já não estão mais aqui entre nós, mas as suas atitudes me inspiram até hoje.

Papa, você criou quatro meninas sozinho, você dedicou sua vida aos seus alunos, você fez de tudo para melhorar a sua comunidade. Mais do que isso, você sempre foi gentil com todo mundo. Você sempre tratou as pessoas da mesma forma, não importava o quão ricos ou o quão pobres fossem, você sempre deixava uma conversa com um sorriso no rosto dos outros. *Vó Penha*, você viveu de forma leve e alegre. Você recebeu várias pessoas na sua casa e, independente de quem entrasse por aquela porta, você sempre fazia uma festa enorme. Seu jeito, sua espontaneidade e sua gargalhada estão na minha memória até hoje. *Vô Carlos*, você sempre foi diferente de todo mundo. A sua ambição era algo fora do normal, sua energia era inesgotável, sua vontade de aprender era incansável, você enxergava coisas que ninguém compreendia naquela época. Eu queria muito que vocês estivessem aqui para ler isso. Obrigado por servirem de modelo para mim.

Se não fosse pela dedicação dos meus pais eu definitivamente não estaria aqui.

Mãe, você abdicou dos seus sonhos pra que eu pudesse viver os meus. Você fez o possível e o impossível para nos dar a melhor educação possível. Você é o motivo de eu chorar com qualquer filme bobo. O seu amor não tem limites e eu tenho muito orgulho de ser seu filho. *Pai*, você me mostrou o significado de trabalhar duro. Tem pessoas que ensinam com livros, você me ensinou pelo exemplo. Não tinha problema que você não resolvesse, não importava o dia ou horário. A única coisa que você se importava na vida era

a sua família. Espero um dia ser um pai tão bom quanto você foi. *Briza*, minha irmã, não tem outro jeito de te definir - você é a minha metade. Você teve a paciência de sentar e me ensinar tudo o que aprendia. No quebra-cabeça da minha vida, você é uma peça fundamental. Você me ajudou, não só com o design desse livro, mas em cada fase da vida. Você está sempre presente mesmo que separados geograficamente. Um dia nossos pais não vão mais estar aqui, mas pode ter certeza que eu sempre vou estar aqui para você.

Nos últimos anos uma pessoa entrou na minha vida e virou o mundo de cabeça para baixo. Essa pessoa hoje é minha esposa.

Carol, você faz com que cada um dos meus dias seja uma aventura. Você pode não admitir, mas você é um exemplo pra mim. Você é um exemplo de sempre buscar ser uma pessoa melhor, de sempre encarar seus próprios monstros ao invés de varrer eles pra debaixo do tapete. Você é uma pessoa incrível e eu agradeço muito você ter escolhido a mim para dividir sua vida. Obrigado por me aturar e por aguentar as minhas ideias malucas.

Por fim, gostaria de agradecer a todos os meus colegas que um dia me deram a oportunidade de trabalhar com eles. Esse livro é um compilado de aprendizados que obtive com vocês.

Muito obrigado por tudo.

Sobre os entrevistados

Este livro é uma coleção de aprendizados valiosos não apenas meus, mas de programadores experientes de todo o mundo. Eu gostaria de dedicar esse espaço para apresentar cada um deles.

Addy Osmani *(Sunnyvale, Estados Unidos)* - Addy é Engineering Manager do Google Chrome. Ele lidera uma equipe de ferramentas de desenvolvimento focada em medir as experiências do usuário para manter a web rápida. Alguns dos projetos de sua equipe incluem Lighthouse, PageSpeed Insights e o Chrome User Experience Report.

Berg Brandt *(Santa Mônica, Estados Unidos)* - Berg é Head of Studios Applications na Amazon. Ele é um líder de tecnologia e produto com mais de 20 anos de experiência construindo produtos de tecnologia de alto engajamento e experiência de usuário. Antes da Amazon, Berg trabalhou no Yahoo por mais de 8 anos.

Blake Williams *(Boston, Estados Unidos)* - Blake é Software Engineer no GitHub, ele é apaixonado por solucionar problemas, pragmatismo e processo.

Caio Gondim *(Nova Iorque, Estados Unidos)* - Caio é Senior Software Engineer no The New York Times. Ele trabalhou anteriormente na Booking.com em Amsterdã, Globo.com no Brasil e CHRI na Índia. Hoje ele trabalha na equipe de plataformas web no The New York Times. Ele publicou vários pacotes no npm que foram baixados mais de 32 milhões de vezes.

Daniel Buchner *(Redmond, Estados Unidos)* - Daniel lidera o desenvolvimento de padrões e código aberto na área de Identidade Descentralizada na Microsoft. Ele é apaixonado por desenvolver aplicativos e serviços que impactam positivamente a vida das pessoas em escala global. Antes da Microsoft, ele dirigia o grupo de produtos em Developer Ecosystem na Mozilla.

Fabio Costa *(São Francisco, Estados Unidos)* - Fabio é Front-end Engineer apaixonado pela criação de UIs com performance. Atualmente, ele é o Líder Técnico do Website Builder da GoDaddy, que permite que os usuários tenham uma forte presença online, simplificando a criação de um site atraente, sem nenhum conhecimento de codificação ou design. Ele trabalhou anteriormente na equipe WebSpeed no Facebook, tornando mais rápidos vários aspectos do facebook.com.

Fernando Tadashi *(São Paulo, Brasil)* - Fernando é Technical Leader na Adobe. Ele é um engenheiro de software multidisciplinar que cria soluções em diferentes áreas, como arquitetura e desenvolvimento de sistemas corporativos com escalabilidade em mente. Trabalha como consultor desde 2012, o que o ajudou a ser mais resiliente em situações de crise com os clientes. Suas realizações mais significativas foram o desenvolvimento de sistemas de alto desempenho para empresas governamentais e privadas.

Lais Andrade *(Londres, Reino Unido)* - Lais é Software Engineer na Google. Depois de se formar na Universidade Federal de Pernambuco, ela trabalhou part-time por alguns anos em uma startup, enquanto também desenvolvia seu mestrado em lógica e ciência da computação teórica. Depois de terminar, voltou ao setor e começou a trabalhar na Liferay, onde teve sua primeira chance de colaborar com pessoas de todo o mundo. Desde então, ela se mudou para Londres para se juntar ao Google, onde trabalha nos últimos anos.

Loiane Groner *(Tampa, Estados Unidos)* - Loiane é Business Analysis Senior Manager no Citibank. Ela trabalha com desenvolvimento de software há mais de 14 anos. Loiane é autora de livros da Packt Publishing, além de Google Developer Expert, Microsoft MVP, Sencha MVP, Oracle Groundbreaker Ambassador e Java Champion. Nas horas vagas, publica artigos em seu blog e vídeo tutoriais online.

Luciano Sousa *(Montreal, Canadá)* - Luciano é Software Developer no Shopify. Programador desde 2010, fugiu do Java em 2011 por encontrar a simplicidade em Ruby. Ele é usuário Vim desde 2007 e preza pela frugalidade na escolha de suas ferramentas de trabalho. Apaixonado por viagem, mora em Montréal, Canadá desde 2018, onde continua tentando dominar o Francês Quebécois.

Manuel de la Peña *(Toledo, Espanha)* - Manuel é Senior Software Engineer na Elastic. Ele trabalha na equipe de Observability da Elastic, mais especificamente na equipe de Produtividade de Engenharia, onde aprimora constantemente a qualidade dos processos e produtos do lado da automação e testes. Antes de ingressar na Elastic, trabalhou na Liferay, melhorando os processos de desenvolvimento, adotando a integração contínua e a entrega contínua, o que permitia aos times mover código dos laptops para produção o mais rápido possível. Manuel é formado em Ciência da Computação e mestre em Pesquisa em Engenharia de Software e Sistemas de Computação pela UNED da Espanha.

Michael Lancaster *(Irvine, Estados Unidos)* - Michael é Senior Software Architect na BlackBerry. Ele tem 10 anos de experiência na área de engenharia de software, trabalhando em empresas conhecidas desde PR/marketing, startups unicórnio, a corporações bem estabelecidas. Nas horas vagas, Michael gosta de montar projetos paralelos e praticar Jiu-Jitsu.

Netto Farah *(São Francisco, Estados Unidos)* - Netto é Principal Software Engineer na Segment focado na criação de ferramentas que capacita os engenheiros de produto a desenvolver de maneira mais rápida e confiável. Netto acredita na solução de problemas organizacionais complexos com design em vez de processos. Seu lema atual de engenharia é: “Não multe os ciclistas por andar de bicicleta no meio dos carros. Pinte as ciclovias”.

Silvio Gustavo *(Estocolmo, Suécia)* - Silvio é Senior Software Engineer no Spotify. Ele começou sua carreira um pouco depois que a Apple e o Google lançaram seus primeiros smartphones. Logo ele percebeu que queria se tornar um desenvolvedor de dispositivos móveis. Ele tem uma profunda experiência em Android, mas também trabalhou em projetos usando iOS e outras tecnologias cross-platform. É apaixonado por código bem feito, arquitetura de software e qualidade. Silvio é formado em Engenharia da Computação e mestre em Visão Computacional, ambos pela Universidade Federal de Pernambuco.

Observação: as opiniões expressas neste livro são próprias e não refletem as opiniões de seus empregadores.

Sobre o autor

Zeno Rocha é um criador e programador brasileiro. Atualmente, vive em Los Angeles, Califórnia, onde é Chief Product Officer na Liferay Cloud. Sua vontade de criar software e compartilhar conhecimento o levou a falar em mais de 110 conferências em todo o mundo. Sua paixão por código aberto o colocou entre os top 20 usuários mais ativos do GitHub aos 22 anos. Antes de se mudar para os EUA, Zeno desenvolveu várias aplicações, orientou startups e trabalhou em grandes empresas da América Latina, como Globo e Petrobras.

Você pode encontrar ele em zenorocha.com.

Bônus

Hábito 2: Foco Nos Fundamentos

A comunidade de código aberto criou um roteiro de fundamentos para desenvolvedores DevOps, Front-end e Back-end.

Acesse 14habits.com/br/bonus/2 para ver.

Hábito 5: Faça Para O Seu Futuro Eu

Criei uma lista dos meus 5 livros favoritos sobre arquitetura de software e boas práticas.

Acesse 14habits.com/br/bonus/5 para receber.

Hábito 8: Projetos Paralelos

Eu listei 7 ideias para começar um projeto paralelo, incluindo exemplos da vida real de meus próprios projetos pessoais.

Acesse 14habits.com/br/bonus/8 para pegar.

E agora?

Se você aprendeu alguma coisa com este livro, se este livro te inspirou de alguma forma, compartilhe com seus amigos.

Fique à vontade para me enviar perguntas, comentários ou sugestões por e-mail: **zeno@14habits.com**.

Muito obrigado por usar o seu tempo para ler esse livro!

P.S.: eu ganharia meu dia se você deixasse uma avaliação na Amazon (https://amazon.com.br/dp/B08BF7PZZX), isso ajuda outros leitores a encontrar este livro. Você sabe como esses algoritmos funcionam :)

</ livro >