DICAS

DE

CARREIRA

PARA

DEVS

ESCRITO POR

ELTON MINETTO

HTTPS://ELTONMINETTO.DEV

Dicas de carreira para devs

eminetto

Esse livro está à venda em http://leanpub.com/dicas-carreira-devs

Essa versão foi publicada em 2024-04-27

Conteúdo

Introdução 1

Como ser um bom profissional de TI? 2

Como melhorar sua carreira? 5

Crie sua marca pessoal 8

Desenvolvedores e portfólios 10

Em que você acredita? 11

Programador Dave Grohl e não Axl Rose 12

Carreira e não emprego 14

Você está se enganando 15

Conheça o seu dia 16

Falta de foco: o mal de uma geração? 17

Por que ir a eventos? 18

Três lições que o AC/DC pode dar para sua carreira 19

Quer melhorar como palestrante? Faça como o Metallica! 21

Como montar um bom currículo 22

Cinco lições que aprendi sobre a carreira de desenvolvedor 24

Minhas dicas de produtividade 26

Windows, Linux ou Mac. O que é melhor para desenvolvedores? 28

A paternidade me tornou um profissional melhor 29

Full Stack vs Full Cycle developer 31

Como evoluir na carreira de dev? **33**

Twitter, uma ferramenta importante para devs **34**

O que é um Great Place to Work para você? **35**

Dicas para desafios técnicos **36**

Liderança técnica: acompanhamento X autonomia **38**

Carreira em Y **39**

Crie um brag document **41**

Como decidir o que estudar? **43**

Documente primeiro **49**

Dicas de livros sobre complexidade **51**

Aqueles que mantém o mundo girando **53**

Programação pessimista **55**

Developer productivity for fun and profit **57**
- Domine suas ferramentas 58
- Documente 58
- Simplifique 60

Responsabilidade e disciplina **61**

Deveríamos parar de nos definir como devs backend ou frontend? **62**

Como organizo minha semana **64**

Por que escrever? **66**

Introdução

Escrever um livro sobre carreira esteve na minha mente por um bom tempo, mas sempre que pensava em colocar a ideia em prática eu sofria com a "síndrome do impostor". Afinal, quem sou eu para dar dicas de carreira? Não criei nada revolucionário, não fiquei rico com algum software que criei, não sou convidado para palestrar mundo a fora, etc, etc.

Mas percebi que eu já estava falando sobre carreira em palestras, conversas com colegas de trabalho e, principalmente, no meu site pessoal. Eu escrevo no https://eltonminetto.dev[1] desde 2003, e neste tempo todo eu acabei tocando no assunto mais de uma vez. Então resolvi colocar as dúvidas de lado, revisar os textos e transformá-los neste e-book que você está lendo.

Os textos são o reflexo de minhas experiências em diferentes momentos na carreira, desde a visão de desenvolvedor, líder técnico e empreendedor. São textos curtos e direto ao assunto e, espero, de fácil assimilação. Mas não são a "verdade absoluta", não são um "guia infalível para ter sucesso na carreira", afinal cada pessoa tem uma série de qualidades e desafios diferentes. São apenas a minha opinião e experiência. Espero que sirvam de inspiração para sua carreira.

Como forma de retribuir toda a ajuda que eu tive na minha carreira, através de amigos, colegas, mentores, posts e palestras, este e-book é 100% gratuito. Desta forma espero atingir o maior número possível de pessoas. Se você gostar do conteúdo do livro e quiser ajudar divulgando nas suas redes sociais, grupos de amigos e colegas eu agradeço imensamente.

Espero que goste do conteúdo deste e-book.

Elton Minetto

Maio de 2019.

[1] https://eltonminetto.dev

Como ser um bom profissional de TI?

Hoje eu estava almoçando e fazendo os cálculos. Comecei a trabalhar na área de TI em 1996, mesmo ano que iniciei na universidade.

Comecei como muitas pessoas, no papel de estagiário. Passei por diversas fases: estagiário de laboratório de informática, estagiário de manutenção de computadores, estagiário de desenvolvimento, desenvolvedor, analista de sistemas, professor, gerente de projetos. Contando agora, são mais de 20 anos trabalhando.

O que eu aprendi nesses anos? É o que eu quero compartilhar nesse capítulo.

Estude e aperfeiçoe-se

Fazer uma universidade ou não é uma discussão enorme. Eu fiz e recomendo. Aprendi diversos assuntos interessantes e importantes, que uso até hoje. Mas independente se você decidir fazer ou não um curso universitário, estudar é obrigatório. Leia livros técnicos, participe de cursos, seminários, etc. Tecnologias surgem todos os dias, se você não prestar atenção é ultrapassado.

Aprenda inglês

Em seu blog, Guilherme Chapiewski fez um excelente post[2] sobre o assunto. Concordo com cada frase. Eu aprendi a ler em inglês desde moleque, comprando quadrinhos importados usados. A vontade de ler era tanta que pegava um dicionário e ficava tentando decifrar o máximo possível. Com o tempo acabei estudando mais e isso abriu algumas oportunidades legais para mim. Como ir para os EUA[3] em 2004, escrever um livro[4], na IBM.

Mantenha o networking

Uma das lições que aprendi é que sozinhos não vamos muito longe nessa área. Eu me considero um bom desenvolvedor, mas não entendo nada de design ou SEO por exemplo. Conhecer pessoas que possam fazer isso, ou te indicar quem possa é muito importante. Manter seu "networking" é essencial. Mas não é bom lembrar das pessoas somente quando precisamos. Conversar sempre com as pessoas, indicar sempre que puder, beber aquela cerveja ou um cafezinho, etc. Tudo isso faz parte do processo.

Vá a conferências e eventos

Ir a conferências é muito legal. Você aprende muito nas palestras, conhece pessoalmente profissionais de grande reconhecimento. Eu vou a todas as conferências que eu posso.

Palestrar em eventos é um bônus. É a máxima "quem é visto, é lembrado". Eu tive a felicidade de palestrar em alguns eventos[5] e a experiência é ótima. Se você conseguir participar da organização de

[2] http://gc.blog.br/2010/10/05/como-anda-o-seu-ingles/
[3] https://eltonminetto.dev/2004/10/05/indo-pro-texas/
[4] http://www.redbooks.ibm.com/abstracts/sg246649.html
[5] https://eltonminetto.dev/talks/

algum evento, faça. É trabalhoso, mas você aprende muito. Eu participei ativamente da organização de alguns eventos e aprendi muito.

Não se apaixone por uma tecnologia

Aquele papo de "PHP é melhor que Java", ou "Linux é melhor do que Windows" é divertido. E só. Não devemos levar isso muito a sério. Claro que todo mundo tem suas preferências em termo de tecnologia. Mas não é bom ficar preso a uma idéia. Particularmente, eu tenho algumas ressalvas quanto a Java. Mas no momento que encontrar um problema que seja melhor solucionado com essa linguagem não vou ter problemas em usá-la (aliás, isso já aconteceu. Em um projeto de TCC de um acadêmico que eu orientei). A idéia é estar focado na solução. Usar a melhor tecnologia para solucionar determinado problema. É o que as empresas precisam e valorizam.

Aprenda outras áreas

Ficar muito "bitolado" em tecnologia nos faz uma pessoa limitada. Isso nunca é bom. Na minha experiência pessoal posso citar um exemplo. Cursei um MBA em Gerenciamento de Projetos. Isso me fez conviver com pessoas de outras áreas (engenheiros, administradores e contadores) e estudar assuntos como administração, contabilidade, finanças. Isso me fez entender outras áreas de conhecimento e me ajudou a ver que o mundo é maior do que a tela de um computador. Lógico que aprendi o suficiente para saber que preciso contratar um contador para me ajudar, mas aprendi coisas importantes.

Tenha um blog/site

Eu mantenho o meu site pessoal[6] desde 2003. Nesse tempo eu recebi comentários de pessoas que foram ajudadas por algum post, algumas que discordaram do que eu escrevi, outras que contribuíram com links melhores, etc. Ele faz parte do meu cartão de visitas. Manter um blog ou site pessoal requer pouco esforço e o retorno é grande. Eu ouço de alguns alunos "mas professor, eu não tenho muito a contribuir". Eu acho isso besteira. Se você passou mais de 30 min estudando algo e postar, isso pode salvar 30 min de outra pessoa. Afinal, quantos minutos você já economizou graças a algum post caridoso? Se não consegue tempo para manter um blog, comente em blogs que você visita. Comentários úteis, lógico. Ninguém gosta de Trolls.

Pratique sempre

Um esportista pratica diariamente seu esporte. Nós podemos fazer o mesmo. Programe e teste sempre que puder. Pequenos trechos de códigos, pequenos exemplo, problemas de lógica. Além de ser divertido mantém sua mente "afiada".

Leia muito

E não só livros técnicos. Leia livros de literatura, revistas. Eu leio muitos TCCs e Monografias. É muito fácil identificar pela escrita uma pessoa que passa o tempo todo lendo textos técnicos e só isso. Quem lê escreve melhor, o texto flui, o vocabulário é mais amplo. Existem muitos autores e livros legais. Tolkien, Asimov, Cornwell, etc. Livros de mistério como os de Arthur Conan Doyle (Sherlock Holmes), Agatha Christie e Edgar Alan Poe (meu favorito) ajudam a manter a lógica enquanto divertem.

[6] https://eltonminetto.dev

Colabore

Sempre que puder, ajude as pessoas. Responda alguma dúvida em alguma lista de discussão, ajude um conhecido ou desconhecido. Isso sempre dá retorno.

Participe de algum GU

Eu participei ativamente do Grupo de Usuários de PHP de meu estado (SC) por muitos anos. Conheci pessoas incríveis, recebi reconhecimento e contatos de trabalho. Tive a oportunidade de indicar pessoas para cursos e oportunidades de trabalho. Participar de GU fortalece a tecnologia que você gosta, além de melhorar o mercado a sua volta. Não tem como ser melhor.

Bom, a essa altura você deve estar se perguntando: Quem é esse cara para dar esses conselhos? Não sou milionário, não inventei nenhuma tecnologia revolucionária nem nada assim. Mas eu me sinto muito mais próximo de realizar essas e outras coisas graças a essas lições que aprendi pela vida.

Como melhorar sua carreira?

Navegando pelo feed do site Dzone[7] encontrei um post interessante chamado "How to advance your career? Read the Passionate Programmer!"[8]. O autor do post baseou-se em um livro chamado Passionate Programer, escrito pelo autor americano Chad Fowler. Achei o conteúdo muito interessante e complementa algumas coisas que eu escrevi no capítulo anterior. Entrei em contato com o autor do texto, que foi muito gentil em permitir que eu traduzisse e adaptasse alguns trechos do post original. Os trechos em itálico são comentários meus.

Vá pelo outro caminho

> Tente competir em nichos de mercado. Se você tentar ir com as massas vai acabar competindo com países com salários mais baixos

Quanto aos nichos eu concordo. Um exemplo já foi desenvolvimento de aplicativos para iOS e Android, hoje em dia é fácil pensar em machine learning. Sempre existe um nicho muito promissor, que pode ser um grande diferencial na sua carreira. Quanto aos países de salário mais baixo, acho que ele se refere a Índia e ao Brasil :)

Conheça o negócio onde você trabalha

> Diferentes negócios operam de diferentes formas. Faz toda a diferença se você conhece os modelos de negócios deles. No fim do dia todos somos recursos em uma empresa (ou para um cliente) que conduzem negócios, e nós fazemos dinheiro!

Joque com os melhores

> Quando eu joguei basquete muito tempo atrás, eu fui colocado em um time que jogava em um nível mais alto. Com certeza foi difícil no início, mas estar em um ambiente mais estressante (partidas importantes) me fez aprender muito mais. O mesmo se aplica em todos os campos: quanto maior as habilidades das pessoas ao seu redor (e as expectativas) melhor você se torna.

Não joguei basquete, mas joguei tênis (só por brincadeira) por um tempo e a afirmação dele é verdade. Você joga melhor quando está jogando com um bom adversário, que sabe mais do que você. E o mesmo vale para programação.

Diversidade

[7] http://www.dzone.com
[8] http://bobbelderbos.com/2011/04/advance-career-read-passionate-programmer/

Aprenda uma nova linguagem de programação todo ano. Por que não? Tente novas coisas, seja mais aberto a novas tecnologias. Quanto maior for seu horizonte melhor você se torna. Não tem certeza de onde Java vai estar em alguns anos? Aprenda Clojure. Ruby ou Python? Faça alguma programação em ambas. Então você irá saber qual delas melhor se aplica em um determinado projeto. Você sabe que tem uma opção porque enriqueceu sua caixa de ferramentas!

Medo, seu pior inimigo

Eu posso só citar esta brilhante frase "Planejamento de carreira guiado pelo medo provavelmente irá levá-lo a uma fazenda de cubículos pelo resto de sua vida e não ao caminho da grandeza. Claro, é seguro, mas não é nada divertido".

Olhe ao seu redor

Você vai tornar-se um especialista somente quando olhar com um escopo mais aberto para a área onde está trabalhando. Programa em PHP? Então tire um tempo para aprender a configurar um servidor Apache com PHP e Mysql. Trabalha um monte com framework? Tente outros. Você pegou a idéia.

Isso é legal. Tentar entender melhor as tecnologias que estão ao seu redor. Estudar um pouco do sistema operacional que usa, da IDE, etc. Isso sempre ajuda

Faça

Não espere por outros para te ensinarem algo, vá e aprenda sozinho!

Encontre um mestre

Encontrar um mestre pode ajudar muito no desafio de aprender nesse ambiente amplo de tecnologias. E lembre-se deste provérbio Zen: "Para seguir o caminho procure o mestre, siga o mestre, caminhe com o mestre, veja através do mestre, torne-se o mestre." *Eu li em algum lugar que uma das melhores formas de aprender Linux era encontrar um "guru local" e tentar aprender o máximo possível com ele. Isso se aplica a linguagens de programação e a quase tudo na vida*

Seja um mentor

Ensinar é uma das melhores formas de aprender. Escrever um blog é muito útil para entender um tópico. Isto o força a dominar alguma coisa e a formular as coisas de uma melhor forma (melhora suas habilidades de escrita). Como diz o livro: "Para descobrir se você realmente sabe algo, tente ensinar isso a outra pessoa".

Perfeito! Isso é algo que eu venho dizendo para meus alunos desde que iniciei como professor. Tente sempre ensinar alguém. Além de tudo é muito gratificante

Pratique, pratique e pratique (disciplina)

> A única forma de dominar alguma coisa é dedicar um monte de prática a isso (tempo). Leia algo, codifique um pouco, falhe, melhore, leia algo, etc. Cuidado com a procrastinação. Geralmente apenas iniciar algo é o suficiente.

Comece pequeno

> Tenha uma realização para reportar todo dia. Mantenha um diário (blog?). Aceite que você não é tão capaz depois de uma semana, apenas aceite que você hoje é melhor do que ontem. Comece lentamente para manter-se motivado.

Aproveite a viagem

> Foco no presente, não no objetivo por si só, aproveite as pequenas vitórias que você poderia perder perseguindo objetivos futuros. Viva no presente. Eu aproveito tanto a programação quanto o resultado final

Não sinta-se confortável

> Quanto mais sucesso você tem, mais fácil de cometer esse erro fatal. Nunca sinta-se muito confortável, especialmente sabendo que o que você sabe hoje pode tornar-se obsoleto amanhã. Nesse setor você nunca pode realmente se dar ao luxo disso.De acordo com o livro, a melhor coisa que pode-se fazer é tornar-se um "generalista". Nunca confie demais em uma tecnologia ou empresa. Algumas habilidades que você domina, mesmo seu trabalho todo, podem tornar-se obsoletas amanhã. Sempre procure formas de melhorar/expandir suas habilidades.

Não concordo 100% com a parte do "generalista", mas isso é assunto para o capítulo Full Stack vs Full Cycle Developer

Venda-se

> Contribua para um projeto existente, escreva um blog, crie e compartilhe códigos, seja útil para uma comunidade existente. Você fará isso com paixão, como um hobby, mas estará indiretamente promovendo seu trabalho/ habilidades/ marca.

Vigie o mercado

> O livro menciona os "geeks alpha", aqueles que estão sempre a frente de novas tecnologias. Eles serão os primeiros a falar sobre coisas que podem tornar-se grandes novidades tecnológicas. Identifique-os e siga seus tweets e posts de blogs.

Considero estas ótimas dicas, que venho seguindo e gostando muito do resultado.

Crie sua marca pessoal

Recentemente encontrei um post[9] que havia lido anos atrás e achei interessante comentá-lo aqui, pois ele continua relevante. O autor fala brevemente da importância de termos uma "*personal brand*" e cita algumas dicas úteis para nós que trabalhamos na área de TI. São elas (em tradução livre e com meus comentários):

Lidere ou crie um grupo de usuários

Posso citar vários amigos da comunidade PHP que trabalharam em grupos de usuários (PHP-SP, PHP-SC, PHP-MS, PHP-Maranhão, etc) e hoje encontram-se em empregos ótimos, e muito respeitados nas suas áreas. É algo que vai ocupar um pouco do seu tempo livre, mas que tem muita recompensa.

Crie ou contribua com um projeto open-soure popular

Também muito importante, pois é uma chance de você mostrar a várias pessoas o seu código, a sua forma de trabalhar. Alguns exemplos: Doctrine[10] e o grande Guilherme Blanco[11], Respect[12] e o seu criador Alexandre Gaigalas[13].

Escreva um blog

Meu blog[14] começou como um site onde eu anotava as coisas que eu ia aprendendo, para poder usar mais tarde caso necessário, e acabou gerando muitos amigos e algumas boas propostas de emprego/projetos. A minha regra é: se eu demorei mais de 30 min para resolver ou aprender algo, no mínimo posso salvar 30 min do tempo de alguém, então isso acaba virando um post.

Publique um livro

Publicar um livro nos dias de hoje é algo um pouco mais fácil do que a alguns anos atrás, graças a popularização dos e-books e seus formatos. Você pode também aproveitar as lojas e serviços como Amazon, iBooks, ou colocar a venda (ou de graça) no seu próprio site, como eu fiz com este livro.

Você também pode optar por um livro impresso e entrar em contato com as editoras, que estão sempre em busca de novos autores e títulos. Eu trabalhei com a Editora Novatec[15] e tive uma boa experiência.

Se você me perguntar se vale a pena eu vou responder que financeiramente não é algo que vá te deixar rico, mas vai te trazer reconhecimento, satisfação e alguns ótimos amigos

[9] http://www.codinghorror.com/blog/2006/04/your-personal-brand.html
[10] http://www.doctrine-project.org
[11] http://twitter.com/guilhermeblanco
[12] https://github.com/Respect
[13] http://twitter.com/alganet
[14] https://eltonminetto.dev
[15] http://www.novatec.com.br/

Palestre em conferências

Existem várias conferências onde você pode mostrar seu conhecimento e aparecer para o mercado. Eu recomendo começar por eventos menores, do seu grupo de usuários, da sua empresa, da sua universidade, para conseguir mais confiança perante ao público e depois partir para as conferências maiores pelo país e até as internacionais (uma ambição que eu ainda tenho). Aqueles velhos medos como "e se as pessoas da platéia souberem mais do que eu?", "e se me perguntarem algo que eu não sei" são infundados, pois todos sabem que ninguém é "dono da verdade". Todas as minhas experiências palestrando foram muito recompensadoras, desde apresentar para 5 pessoas até auditórios cheios de gente.

Como o próprio autor do post comenta, essas são apenas algumas dicas, existem diversas outras formas de se fazer isso, mas já é um bom começo com um bom retorno.

Desenvolvedores e portfólios

Nos últimos anos eu desempenhei o papel de líder técnico, CTO e empreendedor. E uma das atribuições destes cargos é contratar pessoas, especialmente de dois perfis distintos mas que trabalham juntos: desenvolvedores e designers.

Uma diferença interessante que percebi é a forma como os designers apresentam-se para os contratantes. O maior destaque no e-mail de apresentação que enviam não é a sua formação escolar ou as empresas que trabalharam e sim um portfólio onde mostram os trabalhos que realizaram. E muitas vezes mostram também os "projetos pessoais" ou "projetos autorais", que são os trabalhos que eles fazem fora do horário comercial, por puro prazer pela arte ou a profissão.

Os desenvolvedores dão mais destaque ao seu currículo, com os cursos, certificações e empresas que trabalharam. Nesse ponto nós desenvolvedores podemos aprender bastante com os designers. Vamos montar e manter um "portfólio de códigos" e a melhor forma de fazer isso é dar uma atenção maior ao seu perfil no Github[16] ou site pessoal.

Participe de projetos abertos, faça fork dos projetos que tem interesse, escreva um projeto de exemplo, crie um repositório público no Github, divulgue, convide amigos para escreverem códigos no seu projeto.

Isso é muito útil para sua carreira. Alguns anos atrás me pediram para indicar uma pessoa para uma vaga em uma empresa americana e a minha única resposta foi o perfil do Github do sujeito. 20 minutos depois o recrutador me respondeu agradecendo pela ótima indicação e essa pessoa foi contratada quase que imediatamente.

Por isso desenvolvedores, aprendam com os designers. Vamos melhorar nossos "portfólios de código"!

[16] http://github.com

Em que você acredita?

Ontem zapeando pelo Netflix acabei esbarrando novamente nessa palestra do TED:

http://www.ted.com/talks/lang/pt-br/simon_sinek_how_great_leaders_inspire_action.html[17]

Já havia visto o video algumas vezes, mas desta vez me fez pensar neste texto. Se você ainda não assistiu eu recomendo.

O principal ponto que o Simon Sinek apresenta é que as pessoas e as empresas que lideram a inovação são aquelas que mostram o *POR QUE* elas fazem o que fazem, e não *O QUE* ou *COMO* o fazem. Ele cita como exemplo Martin Luther King e a Apple.

Isso me fez pensar no que eu acredito e como isso se reflete na minha carreira. Eu acredito que a tecnologia é uma arte, que nós realmente podemos fazer do mundo um lugar diferente através dela. Que podemos ser como os Beatles, Rolling Stones e Black Sabbath, que podemos mudar o mundo. Por mais arrogante que isso possa parecer também tem uma porção de verdade.

Ainda sobre o vídeo, o palestrante cita a importância das primeiras pessoas que acreditam no seu sonho, nas que te ajudam a propagá-lo. Ele não poderia estar mais certo, pois nós devemos agradecer aos colegas, familiares, clientes, mentores, todos que ajudam na nossa jornada. São eles que nos fazem acordar todos os dias e enfrentar as dificuldades da carreira.

E você? No que acredita? O que te faz levantar da cama todos os dias?

Pense nisso!

[17] http://www.ted.com/talks/lang/pt-br/simon_sinek_how_great_leaders_inspire_action.html

Programador Dave Grohl e não Axl Rose

Como comentei no capítulo anterior, acredito que programadores podem ser comparados a artistas, mais especificamente a "rockstars". Esse termo "programador rockstar" vem sendo usado por algumas pessoas como pejorativo, como algo a ser evitado.

O CEO da Netguru[18] fez uma ótima apresentação com o título "Why no one is looking for 'rockstar programmers'[19]".

Traduzi/adaptei os trechos mais importantes:

- **Seja um programador sem ego**. Lembre que você não é o seu código. A única forma de melhorar é estar aberto para ao feedback dos outros e dar feedbacks honestos e sem julgamentos.
- **Jogue pelo time**. Construir software é um esporte a ser jogado em grupo. Você não é dono de partes do código e sim responsável por fazer com que todo o pacote funcione como esperado. Não seja o cara que diz "na minha máquina funciona". O software deve funcionar nas máquinas dos usuários.
- **Seja um aprendiz**. A tecnologia muda muito rápido. O que era importante 5 anos atrás não é mais a melhor coisa hoje, e o que é relevante hoje nem existia 5 anos atrás. Você deve aprender novas coisas constantemente (linguagens, bibliotecas, padrões).
- **Seja uma pessoa em formato T**[20]. Se especialize em algum campo e tenha um conhecimento superficial em um conjunto de tecnologias. Isto o faz um recurso muito valioso dentro da empresa.
- **Seja incansavelmente engenhoso**. Torne-se a pessoa que consegue resolver qualquer problema. Você não precisa ser capaz de resolver qualquer problema sozinho, mas você sempre sabe onde ir para encontrar a resposta.

Concordo com todos os pontos que o Wiktor cita na apresentação, só não acho certo dizer que quem não tem essas qualidades é um "rockstar". Pelo menos não no modo como vejo, com os rockstars criando coisas novas e sendo ousados, quebrando o *status quo*.

Eu mudaria os termos para: "**não seja um programador Axl Rose, seja um programador Dave Grohl**".

Ambos são muito talentosos, mas o Axl é famoso[21] por suas excentricidades e manias.

Enquanto isso, o Dave Grohl é conhecido com "o cara mais legal do rock"[22] por sempre estar disposto

[18] https://netguru.co
[19] http://blog.netguru.co/post/56860239654/no-one-is-looking-for-rockstar-programmers
[20] http://en.wikipedia.org/wiki/T-shaped_skills
[21] http://whiplash.net/materias/news_847/138425-gunsnroses.html
[22] http://exame.abril.com.br/estilo-de-vida/noticias/livro-mostra-por-que-dave-grohl-e-o-cara-mais-legal-do-rock

a conversar com fãs e reporteres, por ajudar bandas que estão começando, por saber tocar diversos instrumentos, etc.

Essa é minha sugestão.

Carreira e não emprego

A ideia para este texto surgiu de duas situações. A primeira foi o fato de, por ser mais experiente e participar de diversos eventos, acabo recebendo e-mails de pessoas que estão entrando agora na profissão, com aquelas dúvidas que eu também tive no início.

A segunda motivação foi uma das ótimas conversas geradas em intervalos de palestras em um evento de desenvolvedores que participei (você precisa participar destes eventos!). Estava conversando com outros palestrantes, profissionais experientes e com trajetórias similares a minha (programador que virou gerente de desenvolvimento, de projetos ou de empresas) e constatamos que um problema que ocorre em algumas empresas é o de desenvolvedores que parecem estar mais preocupados apenas com o salário do final do mês, em cumprir apenas a sua tarefa sem dar atenção ao projeto como um todo.

Eu resumiria essas pessoas como aquelas que procuram e se preocupam apenas em manter um emprego e não uma carreira. E são coisas bem diferentes. Ao almejar uma carreira você consegue ter uma visão de longo alcance, consegue perceber mais claramente quais empregos você precisa manter e quais fazem mal para seu crescimento. Algumas vezes você pode se encontrar em um emprego que não paga o suficiente, mas que te coloca em contato com pessoas ou tecnologias que vão ser importantes no seu futuro.

Mas ter em mente uma carreira é só o começo. Ao entrar em um emprego que você identificou ser importante para sua carreira, por um motivo ou outro, você deveria se empenhar ao máximo para fazer o melhor possível para a empresa, colegas e projetos. Envolver-se realmente, tentar entender onde a empresa ou o projeto pretendem ir, visualizar onde seu trabalho influencia nisso, ajudar os colegas a também entenderem e cumprirem suas partes. Este comportamento faz muita diferença na forma como as pessoas vão ver você e isso é ótimo para sua carreira.

Concordo que as vezes determinar para onde quer chegar e que carreira seguir não é fácil. Eu mesmo passei pelo que eu chamo de mini-carreiras: desenvolvedor, professor, gerente de projetos, escritor, palestrante, empresário. Em algumas fui melhor sucedido do que outras, mas tenho orgulho de dizer que me esforcei, e ainda me esforço, ao máximo em cada uma delas. Ainda tenho objetivos a completar em algumas delas (quero algum dia palestrar em um TED, por exemplo) e algumas eu precisei dedicar menos tempo (praticamente parei de ministrar aulas), mas todas me levaram um passo adiante na macro-carreira que escolhi: a de tecnologia (dou aulas de programação, palestro e tenho uma empresa na área).

Então, se eu puder dar mais uma dica para quem está começando, ou já está no mercado, é de escolher uma carreira e manter o foco nela. Ou identificar o quanto antes se você não está satisfeito e mudar de carreira. E sempre se esforçar ao máximo nas suas escolhas.

Você está se enganando

Na minha passagem diária pelo Hacker News encontrei esse post[23] que me deixou pensando em algumas coisas.

Resumindo, o autor comenta que trabalhou para algumas empresas na sua carreira e nelas o que predominava entre os funcionários era a procrastinação, pessoas "fazendo de conta" que trabalhavam, passando o dia navegando no Facebook, Twitter, etc.

Esse foi o primeiro ponto que me fez pensar. Pelos meus cálculos eu já tenho mais de 20 anos de profissão (contando o tempo de estagiário) e deste tempo somente os últimos 9 anos eu trabalho numa empresa onde sou sócio. Durante toda a minha fase de funcionário eu nunca me vi trabalhando apenas para a empresa, mas sim tendo dois "patrões": a minha carreira e a empresa propriamente dita. Quando o caminho que eu queria para a minha carreira não era mais o mesmo que a empresa estava indo eu mudei de empresa. Mas durante o tempo que estive trabalhando lá eu me esforcei ao máximo, porque ao pensar "estou enganando a empresa fazendo de conta que estou trabalhando" na verdade eu estaria me enganando.

O outro ponto do post que me chamou a atenção foi quando ele comentou que "agora ele iria trabalhar para si mesmo" e que "agora eu trabalho nas tarefas que eu quero". Isso está completamente errado! Ao montar uma empresa você vai trabalhar muito mais do que quando era funcionário, vai trabalhar em várias coisas que você não gosta ou não são sua especialidade, apenas porque você PRECISA. Eu amo programar, trabalhar com servidores, redes, arquiteturas, mas ao virar empresário eu precisei fazer coisas como contabilidade, compras, contatos com clientes, contratar pessoas, etc. Atualmente tenho ótimas pessoas ao meu redor que me ajudam em diversas destas tarefas mas é raro o dia que eu consigo sentar e apenas trabalhar no que eu GOSTO e não no que eu PRECISO fazer.

Os recados que tentei passar nesse texto são: se você acha que está enganando seu chefe ao "fazer de conta que trabalha" na verdade você está enganando a si mesmo. E se você está montando uma empresa precisa ser porque você acredita nela, nos seu sócios, produtos e clientes, e não porque acha que vai trabalhar apenas no que gosta, que vai parar de procrastinar quando trabalhar para si mesmo.

P.S.: Não estou dizendo que ter uma empresa é algo ruim, só que não é tão fácil quanto parece ;)

[23] https://medium.com/life-hacks/76632b62eb0b

Conheça o seu dia

Nos últimos anos tenho tentado otimizar meu tempo, fazer mais coisas durante o dia.

Uma das coisas importantes que fiz recentemente foi perceber que meu dia funciona por ciclos. Eu percebi que tenho picos de criatividade e inspiração assim como momentos de cansaço durante o dia. Ao acordar, lá pelas 7 da manhã, perto das 3 da tarde e próximo das 9 da noite são os momentos que meu cérebro está mais criativo, quando minha empolgação está em alta. E perto do meio dia, lá pelas 6 da tarde e depois da meia-noite são os momentos onde eu sinto muito sono, estou muito cansado.

Saber disso me ajuda a planejar melhor o meu dia. Nos momentos de pico de criatividade eu aproveito para realizar tarefas como escrever um post, rascunhar uma ideia para um projeto ou negócio, marcar uma reunião importante de negócios, etc. Nos horários de menor empolgação eu tento fazer tarefas mais simples como responder alguns e-mails, atualizar aplicativos, gerenciar minha lista de tarefas, fazer coisas mais burocráticas de algum projeto, etc. Se eu tentar escrever um post no final do dia ele vai ser horrível e se eu alocar uma tarefa burocrática e repetitiva para as 7 da manhã eu facilmente me irrito com ela, achando que estou desperdiçando meu tempo.

Então fica aqui uma dica. Tente identificar os ciclos do seu humor, do seu ânimo e use esse conhecimento para melhor alocar suas tarefas, sempre que possível. Aposto que você vai conseguir uma produtividade muito maior no seu dia a dia.

Falta de foco: o mal de uma geração?

Uma das minhas atribuições como líder técnico/CTO é encontrar novos talentos. Ao conversar com antigos empregadores buscando referências é muito comum ouvir frases como:

> "Fulano é um bom programador, mas não consegue manter o foco"

> "Ciclano tem grande potencial mas vive perdendo a concentração, fica perdido em redes sociais e a qualidade do trabalho cai bastante"

> "Beltrano é ótimo profissional, mas se passar uma mosca perto ele perde a concentração"

Percebo que esse comportamento é bem mais comum em profissionais mais novos, em início de carreira. Justo no momento onde devemos solidificar nosso nome, quando temos uma grande capacidade de aprender, de crescer. E as pessoas estão desperdiçando essa energia com distrações. Sem falar que podem criar um estigma ruim para seus nomes pois é difícil contratar ou indicar para alguém o "avoado", o "distraído", o "sem-foco".

É óbvio que ninguém consegue sentar na frente do computador e trabalhar horas sem parar, sem um descanso, sem uma pausa para tomar um café, uma água, para ler e-mails e conversar com amigos e parentes nas redes sociais. Mas isso precisa ser uma atividade controlada, é preciso disciplina.

Eu posso dar duas dicas quanto a isso. A primeira é usar alguma técnica para organizar seu tempo, como a Pomodoro[24]. O resultado é muito bom pois além de melhorar a concentração nas tarefas também ajuda a identificar e medir como você está usando seu tempo. E com as paradas programadas pela técnica é possível tomar um café, conversar com alguém, dar uma caminhada pela sala, comer algo.

A segunda dica é: ninguém vai morrer ou deixar de te amar se você não der Like naquele post, se você deixar a mention no Twitter para ser respondida mais tarde, se o e-mail não for respondido imediatamente, se a pessoa que te chamou no chat receber a resposta alguns minutos depois. Eu mantenho uma regra de urgência para as comunicações, do mais urgente para o menos: telefone, WhatsApp/Telegram, chat, e-mail. E respondo nessa ordem. Com isso consegui ter um pouco de paz e diminuir a ansiedade de responder as coisas na hora que a notificação chega.

Fica aqui uma dica, como programador, como gerente de projetos, como gerente de empresa. Não deixe as distrações da vida queimarem sua carreira.

[24] http://pt.wikipedia.org/wiki/T%C3%A9cnica_pomodoro

Por que ir a eventos?

Algumas pessoas me perguntam se vale a pena ir em eventos e neste post quero citar alguns motivos pelos quais acredito que isso é um dos melhores investimentos que você pode fazer para sua carreira.

- **Conhecimento**. Apesar de termos todo conhecimento necessário para resolver qualquer problema ao acesso de uma busca no Google, ir a eventos trás um outro tipo de informação. Assistindo as palestras, conversando com as pessoas nos corredores você consegue adquirir algo que não está tão facilmente disponível na internet: a experiência das pessoas. Ao invés de aprender o que é e como funciona a tecnologia X, você pode ouvir algo similar a "na minha empresa nós usamos X para resolver problema Y, e tivemos essas vantagens, e estes problemas". Isso é muito valioso, por isso eu busco sempre por este tipo de conhecimento nos eventos que participo.
- **É bom para os negócios**. No mercado, assim como na balada, uma coisa que vale bastante é o "ver e ser visto". Participar de eventos, especialmente como palestrante ou organizador, propicia que você e sua empresa sejam vistos pelo mercado. Uma maioria esmagadora dos projetos em que trabalhei nos últimos anos veio de recomendação de pessoas que eu conheci em eventos, ou que assistiram alguma das minhas palestras. É um grande investimento para sua marca pessoal ou mesmo para a sua empresa.
- **Amizades e networking**. Eu comecei a participar de eventos nacionais em meados de 2005 e nestes anos fiz muitas amizades, conheci muitas pessoas incríveis. Consegui conhecer ícones como Eric Raymond, Rasmus Lerdorf, Derick Rethans, Cal Evans, Andi Gutmans, Guilherme Blanco, entre outros. Fiz amizades que já duram mais de uma década e devem continuar por muito tempo, além das parcerias de negócios que surgiram neste processo.
- **É divertido**. Todos esses anos indo a eventos geraram histórias homéricas, mas a grande maioria delas eu não posso citar aqui ;)

Estas são apenas algumas das razões porque eu continuo investindo meu tempo para preparar palestras, organizar eventos e acredito que todo profissional deveria fazer algo similar.

E nos vemos no próximo evento...

Três lições que o AC/DC pode dar para sua carreira

Em uma dupla de episódios do excelente canal Heavy Lero no Youtube[25] fomos apresentados aos primeiros anos da grande banda AC/DC. E em meio a suas desventuras três pontos me chamaram atenção, e que me levaram a escrever este post.

Os irmãos Young começaram a banda na Austrália, mas desde o início eles tinham ambições de serem a maior banda de rock do mundo e não a maior do seu país. Com isso em mente, e após começarem a fazer sucesso em shows locais, eles perceberam que a única forma de conseguir isso era indo para um lugar mais propício: Londres.

Primeira lição:

> Entenda onde você quer chegar na sua carreira. E se o lugar/trabalho/escola/cidade/país onde você está não fornece as chances que você precisa para crescer, mude-se!

Mas chegando em Londres as coisas não foram tão simples quanto eles pensaram. Os primeiros meses foram duros, sem grana e sem conhecer ninguém para pedir ajuda. Mas eles persistiram e um momento em particular foi emblemático. Eles conseguiram marcar um show em uma casa que cabia em torno de 100 pessoas, mas como não eram conhecidos ainda, no dia haviam apenas 10 fãs presentes. Mas a banda tocou com toda a energia que os tornaria famosos, como se a casa estivesse lotada. Reza a lenda que os presentes saíram, ligaram para seus amigos e lotaram a casa antes do fim do show. E com isso a banda começou a criar uma base de fãs e adquirir respeito e reconhecimento.

Segunda lição:

> Trabalhe pelo salário/cargo que quer e não pelo que você tem no momento.

Anos depois, já com fama e sucesso, a banda preparava-se para gravar aquele que seria o seu maior disco até o momento, o Highway to Hell. Contra a vontade dos músicos a gravadora contratou um novo produtor, o experiente Robert John "Mutt" Lange. Uma das importantes contribuições que ele trouxe ao disco foi dar dicas e ensinar técnicas vocais ao veterano vocalista Bon Scott. E estas técnicas vocais ajudaram muito na qualidade do resultado do disco, contribuindo ainda mais para o seu sucesso.

Terceira lição:

[25] https://www.youtube.com/channel/UCA4u8p5rYvuL2-72cAUhXKA

> Não importa o quão experiente, sempre é possível encontrar um mentor, alguém que pode lhe ensinar novas técnicas e ajudar no seu crescimento.

E o fato de encontrar essas lições em um vídeo que não é da minha área (desenvolvimento de software) fortalece a importância de consumirmos conteúdos de outros assuntos e como isso pode nos ajudar a crescer.

Quer melhorar como palestrante? Faça como o Metallica!

Algum tempo atrás um amigo me convidou para assistir ao ensaio de uma banda. Eles estavam ensaiando para um show onde tocariam músicas de uma das minhas bandas favoritas: Rage Against the Machine, então acho que não preciso ser óbvio comentando o quanto foi divertido, certo?

Durante o ensaio não consegui evitar a comparação do processo deles com o que eu faço quando vou preparar uma nova palestra.

Uma banda ensaiando é bem diferente da mesma banda no palco, porque as preocupações são diferentes. No ensaio eles se preocupam com a técnica, com o tempo das músicas, com a ordem que vão tocá-las, tentam encontrar e corrigir erros. No momento do show a situação é bem diferente pois com a interação do público a banda toca com muito mais empolgação e as vezes até muda a ordem das músicas para envolver ainda mais a audiência.

O mesmo acontece quando estamos preparando uma nova palestra. É preciso ensaiar antes de apresentar, entender o tempo de cada slide e exemplo, corrigir as falhas, reservar casos mais avançados para mostrar caso preciso, etc. E durante a apresentação é preciso observar a platéia, entender o nível de conhecimento prévio, tentar captar as suas expectativas, mudar o plano caso necessário. Com a interação com a audiência, suas perguntas e reações, o palestrante consegue uma apresentação melhor.

Neste vídeo é possível ver um pouco desse processo acontecendo com o Metallica

https://www.youtube.com/watch?v=d0ZF2HNRSOk[26]

É interessante ver que, apesar de tocarem juntos a quase 30 anos, eles ainda praticam e ensaiam antes de cada show e escolhem a ordem das músicas pensando na platéia. Também é possível ver a preparação do James antes de entrar no palco, que também é parecido com o processo que ocorre antes de um palestrante iniciar uma apresentação.

Então segue algumas dicas para quem quer palestrar:

- pratique e ensaie
- pense na platéia e prepare-se para ela
- observe as reações da audiência durante a apresentação e use isso a seu favor
- use a interação com as pessoas para fazer uma apresentação melhor

[26] https://www.youtube.com/watch?v=d0ZF2HNRSOk

Como montar um bom currículo

Desde 2010, quando comecei a empreender e liderar equipes, uma das tarefas mais difíceis e importantes que eu tenho realizado é a contratação de pessoas. Neste texto gostaria de compartilhar algumas ideias que podem ajudar na escrita de um currículo para devs e designers.

Conversando com outros líderes de equipe e empreendedores, é possível ver que a maioria dos processos de contratação de devs/designers passa por um ciclo similar a este:

> leitura de currículos -> entrevistas -> teste técnico -> entrevistas com a equipe -> contratação

O processo é um "funil", ou seja, o currículo é a primeira etapa e é geralmente onde encontro o maior número de problemas. Currículos com pouca informação, bagunçados e que acabam mais atrapalhando do que ajudando.

Quando recebo um currículo para analisar, as informações que eu procuro geralmente são: quem é a pessoa, o que ela fez, onde ela trabalhou e o que ela conhece, então vou separar o texto nestes quatro tópicos. Não sou especialista em recursos humanos então esta é apenas a minha opinião e experiência depois de vários anos lendo currículos.

Quem é você?

As informações mais importantes são Nome, idade, cidade, estado e formas de contato como e-mail, site pessoal. Algumas informações não são importantes como rua (não vou mandar uma carta para você), Facebook, Twitter, telefone (eu dificilmente ligo para os meus pais, imagina para uma pessoa que acabei de conhecer). A ideia aqui é facilitar a vida de quem vai ler seu currículo para entrar em contato e separar por cidade por exemplo.

O que você fez?

Esse é um ponto importante e fico surpreso como pouca gente usa isso. É aqui a sua chance de mostrar o que é capaz de fazer, o que já realizou. Se você é um desenvolvedor tente lembrar os projetos que teve orgulho de trabalhar, que desempenhou um papel importante, quais projetos open source contribuiu, etc. Se é palestrante inclua links para slides, ou posts, ou livros. Designers podem citar links para seus portfolios em sites pessoais ou especializados como Behance, Dribbble, etc.

Onde você trabalhou?

Uma lista dos locais onde você trabalhou, com uma pequena frase resumindo quais papeis desempenhou lá. Se você teve experiências em áreas diferentes da relacionada a vaga eu não acho muito importante. Já recebi currículos para vaga de desenvolvedor citando experiências que não agregam muito à area, como mecânico.

O que você conhece?

Aqui pode-se citar os cursos de graduação, pós-graduação e quaisquer outros cursos e certificações que sejam relevantes. Curso de datilografia e onde você fez o ensino médio dificilmente contam muitos pontos ;)

E para finalizar, uma dica importante: o formato do arquivo. Por favor, não mande em nada diferente de PDF. É a melhor forma porque todos os sistemas operacionais tem leitores de PDF, você garante que nenhuma fonte legal que usou vai quebrar quando alguém abrir o seu documento do Word no Pages ou OpenOffice. E se quiser facilitar ainda mais a vida das pessoas (e ganhar ainda mais pontos) nomeie o arquivo com seu nome. Algo como CurriculoEltonMinetto.pdf. Sem acentos ou espaços, porque você quer mostrar o quanto entende de tecnologia, certo?

Como eu comentei lá no começo, essas dicas são baseadas apenas na minha experiência e em nenhuma ciência ou norma. Então outra dica útil é tentar conversar com quem já trabalha na empresa para quem você está enviando o currículo, e perguntar qual é o formato mais importante para eles. Nem sempre esta informação vai ser fácil de encontrar, mas com certeza vai ajudar muito a destacar seu currículo dentre os outros candidatos.

Cinco lições que aprendi sobre a carreira de desenvolvedor

Eu realmente acredito que *"o desenvolvimento de software é uma arte que deve ser feita com paixão e dedicação"*. Por isso quero aqui usar algumas analogias com minhas formas favoritas de arte, a música e o cinema, para citar algumas lições que aprendi nos meus mais de 20 anos de profissão.

Apesar de não estarem na minha lista de bandas favoritas é inegável a importância e o impacto que os Beatles tiveram na música e na cultura pop. Mas mesmo os autores de grandes clássicos as vezes criam algo como Ob-La-Di, Ob-La-Da[27], que foi eleita uma das piores letras já escritas[28]. Por isso **não se preocupe se nem todos os seus códigos são dignos de orgulho, principalmente no começo da carreira**. O importante é continuar praticando até compor seu Helter Skelter[29]

Todo mundo conhece o escritor/diretor M. Night Shyamalan por seus suspenses como Sexto Sentido, Sinais ou Corpo Fechado. Mas antes destes filmes ele também escreveu o roteiro de O Pequeno Stuart Little. A lição aqui é que **as vezes você vai precisar trabalhar em algum projeto/código que não te deixe com tanto orgulho, mas que pode abrir um caminho interessante para sua carreira**.

Sou grande fã do Quentin Tarantino e consumidor de todas as suas obras. Ele começou a despontar com o excelente filme Cães de Aluguel, que ele financiou com a venda do roteiro do filme Assassinos por Natureza. Em entrevistas posteriores ele comentou que o diretor Oliver Stone alterou completamente o roteiro que ele fez, e que ele sentia muito por isso. Algo similar já aconteceu comigo várias vezes, quando ao terminar um projeto entreguei o código e o vi sendo transformado em algo que eu não concordava. Mas, assim como o caso do Tarantino, isso está fora do nosso controle e **temos que nos desapegar do código depois de entregue**.

George Clooney fez o pior Batman da história do cinema. Ele mesmo pediu desculpas por isso em uma entrevista[30] onde comentou que achava que o filme seria ótimo para sua carreira, mas foi um desastre. Algumas vezes nosso código não é tão elegante, ou tão eficaz quando poderia, ou possui bugs. **Um desenvolvedor não é julgado apenas por um ou outro deslize, mas pela capacidade de aprender com seus erros, melhorar e continuar crescendo na sua carreira**. Da mesma forma que o George Clooney conseguiu superar o Batman e continua fazendo sucesso.

Para a última lição eu poderia citar várias obras que foram criticadas duramente, mas que seus autores continuaram seu caminho. Isso é algo que vai acontecer constantemente na carreira de um desenvolvedor. **Seu código vai ser criticado, seu Pull Request não vai ser aceito, sua sugestão de arquitetura não vai ser implementada, etc. Questione, entenda os motivos, aprenda e não**

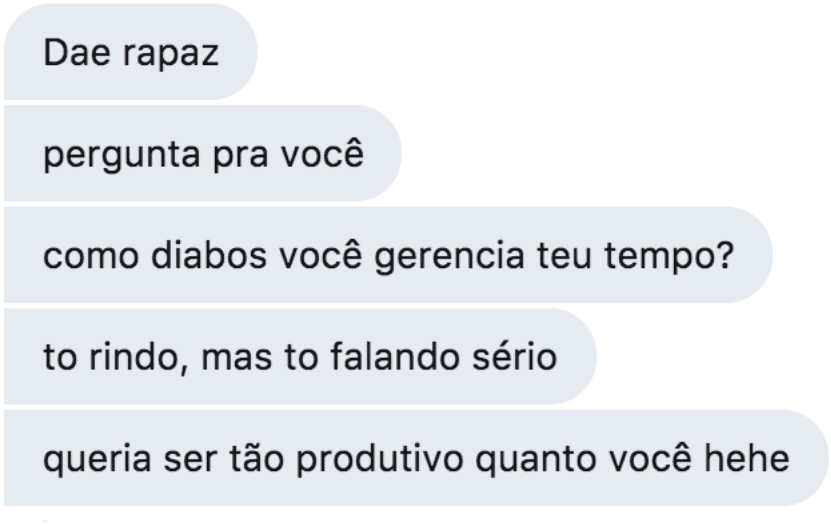
[27] https://www.letras.mus.br/the-beatles/168/
[28] http://news.bbc.co.uk/2/hi/entertainment/3998301.stm
[29] https://www.letras.mus.br/the-beatles/304/
[30] https://www.youtube.com/watch?v=AXzcSCf3kwg

guarde mágoa. É seu código que está sendo criticado e não você. Um desenvolvedor é muito mais do que um código e sim uma carreira toda de entregas e soluções.

E você? Que lições tem para compartilhar?

Minhas dicas de produtividade

Algum tempo atrás um amigo me mandou essa DM no Twitter:



Sinceramente, não acho que eu faça algo de especial, mas achei interessante escrever este texto como um incentivo para que outras pessoas compartilhem suas dicas de produtividade.

Eu posso resumir o que eu faço em três tópicos: listas, anotações e rotina. O que isso significa?

Listas

Com o passar dos anos aprendi que meu cérebro não é nada bom em controlar diversas coisas ao mesmo tempo. Se eu não me controlar sou facilmente distraído e perco o foco do que preciso fazer. Para evitar isso eu anoto tudo o que precisa ser feito em listas de tarefas, o mais rápido possível.

Existem diversas técnicas como a Matriz de Eisenhower[31], Bullet Journal[32] e até usar a linha de comando[33] para gerenciar tarefas tarefas. O importante é selecionar o que faz mais sentido para você e tornar seu uso um hábito.

Anotações

Como comentei, minha memória não é muito confiável, então eu anoto tudo. O que foi decidido em alguma reunião, comandos do Linux/Mac, snippets de código, consultas complexas de bancos de dados, rascunhos de posts e palestras. Também testei várias ferramentas e para selecionar uma eu elenquei as funcionalidades mais importantes (estar sempre acessível, suporte a markdown, consulta rápida, organização por projetos ou tags) e optei pelo Bear[34]. É preciso pagar uma assinatura para

[31] https://eltonminetto.net/2014/08/05/gerenciando-tarefas-com-o-trello-e-a-matriz-de-eisenhower/
[32] https://eltonminetto.net/post/2017-03-03-gerenciando-tarefas-bujo/
[33] https://eltonminetto.net/post/2018-04-12-gerenciando-tarefas-linha-comando/
[34] http://www.bear-writer.com/

ter acesso aos recursos completos, mas vale os U$ 14.99 anuais. Mas existem diversas soluções open source ou gratuitas para todas as plataformas, que podem se tornar uma extensão da sua memória.

Rotina

Ultimamente algumas pessoas tem falado bastante sobre o 5am club[35] e as vantagens em acordar as 5 horas da manhã todos os dias. O que eu acredito tem semelhança com este argumento, mas não necessariamente acordar as 5am vai automaticamente tornar qualquer pessoa o próximo Steve Jobs. O importante é conciliar dois fatores: o conhecimento de como seu corpo funciona e a sua realidade.

Com a experiência eu observei que eu sou mais criativo nas primeiras horas do dia, antes das 9:00. E meu ânimo tem uma queda drástica nas primeiras horas da tarde. Com isso em mente eu tento organizar o meu dia para favorecer esse ciclo. Eu acordo as 7 e uso as duas primeiras horas do dia para escrever posts e palestras, anotar ideias para algum projeto ou produto que esteja trabalhando. E, se possível, marco as reuniões para antes do meio dia ou ao final da tarde, depois que meu ânimo voltou ao seu normal. Tento reservar as primeiras horas da tarde para tarefas mais repetitivas como responder e-mails, revisar algum texto ou pull request, etc.

Claro que ninguém tem controle total sobre o seu tempo, por isso eu preciso ter consciência que vou precisar ajustar as agendas com colegas, clientes, amigos, família (sou eu quem leva e busca a minha filha na escola por exemplo). Mas o fato de conhecer meu ritmo me ajuda a não ficar bloqueado em alguma tarefa criativa, por exemplo, as 13:30, porque sei que nada de muito inventivo vai surgir neste horário.

Como eu disse no começo, nada de especial :) Mas espero que este texto incentive outras pessoas a compartilharem suas dicas, pois é um assunto interessante e útil.

[35] https://www.google.com.br/search?q=5am+club&oq=5am+club&aqs=chrome..69i57j0l5.3033j0j7&sourceid=chrome&ie=UTF-8

Windows, Linux ou Mac. O que é melhor para desenvolvedores?

A menos que você desenvolva aplicativos exclusivamente para alguma plataforma já deve ter ouvido esta discussão entre as comunidades que participa.

Mas afinal, faz diferença o sistema operacional, ou mesmo a IDE que o desenvolvedor usa? Na minha opinião não faz a menor diferença! O que torna uma pessoa melhor profissional do que outra não é o fato de estar usando Ubuntu, MacOS, Windows, Visual Studio Code ou vi. O que faz toda a diferença é a atitude que esta pessoa tem em relação a suas escolhas.

Em um texto que escrevi anos atrás eu falei sobre as leis vikings[36]. E uma delas cabe bem neste contexto:

> Mantenha suas armas em boas condições

Se você optou por usar Windows, ou qualquer outra ferramenta, esforce-se para deixar sua "arma" pronta para ser usada da melhor forma. Aprenda o máximo que puder sobre como ela funciona, quais os pontos fortes, fracos, como adaptá-la para seu uso diário.

Quando desenvolvemos a plataforma de desafios da Codenation[37] uma das decisões que precisávamos tomar era se forneceríamos ou não uma IDE web para que os desenvolvedores pudessem escrever seus códigos. Optamos pelo caminho mais "difícil", pelo menos para nós, que foi o de convencer os desenvolvedores a investirem um tempo aprendendo como criarem o seu ambiente local, configurarem e aprimorarem suas ferramentas. Isso nos trouxe mais trabalho de suporte, amenizado pela própria comunidade se ajudando no forum[38], mas os resultados tem aparecido, com desenvolvedores criando e configurando suas máquinas para resolver os desafios.

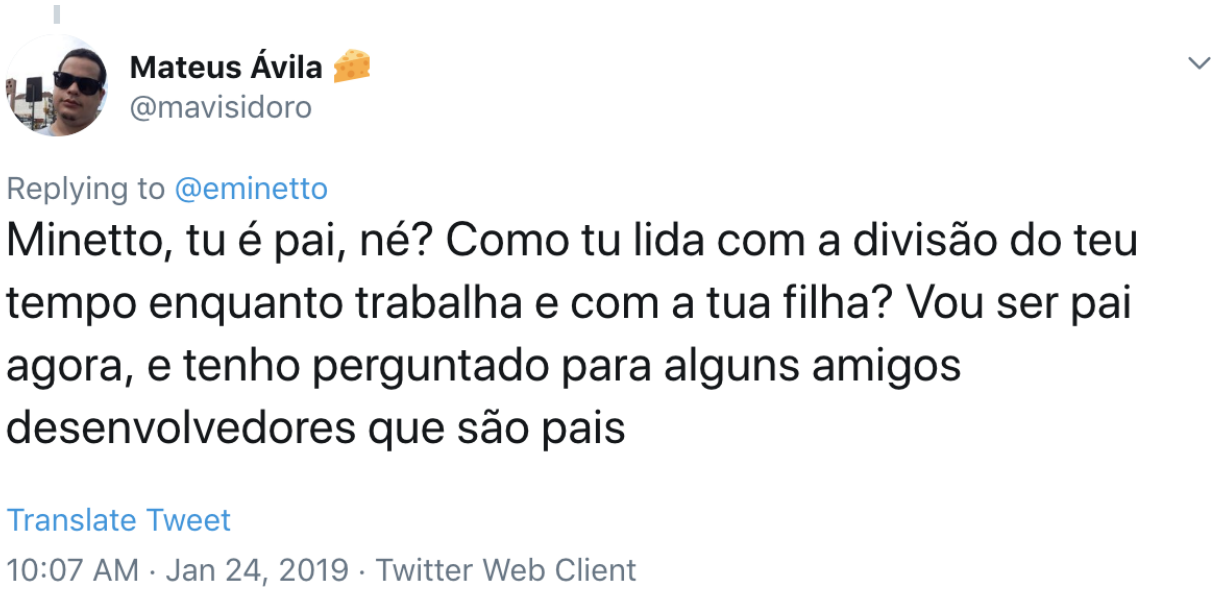
Quando eu comecei a minha carreira de desenvolvedor eu criava aplicativos para a plataforma Windows. Quando comecei a desenvolver para a web aproveitei a oportunidade para aprender Linux e, depois de alguns anos, MacOS. Em todas as fases eu me esforcei para aprender o máximo possível do ambiente e customizá-lo de forma que eu fosse o mais produtivo possível.

Então, se alguém falar que você é um desenvolvedor inferior (ou superior) porque usa [insira sua ferramenta favorita aqui], não se importe com isso. Apenas continue se esforçando para tornar-se mestre nesta ferramenta e continue crescendo na sua carreira ;)

[36] https://eltonminetto.net/2012/06/21/as-leis-vikings/
[37] https://www.codenation.dev
[38] https://forum.codenation.com.br

A paternidade me tornou um profissional melhor

Resolvi escrever este post depois de receber uma pergunta interessante no Twitter:



Durante toda minha vida adulta a minha carreira sempre foi o ponto focal. Nas decisões que tomei no decorrer dos anos o que era melhor para minha carreira sempre teve um peso muito grande.

Quando descobri que eu iria ser pai, passadas a euforia e alegrias iniciais, uma das minhas preocupações foi como a paternidade iria afetar minha profissão.

Agora, passados mais de quatro anos desde que a linda Alice nasceu, eu posso refletir com calma e afirmar: a paternidade me tornou um profissional melhor.

Eu sempre fui dedicado em todos desafios profissionais pelos quais eu passei, mas agora eu sinto que tenho um foco ainda mais preciso em quase tudo que eu faço. Antes eu trabalhava mais horas por dia mas hoje eu percebo que consigo ser muito mais eficiente, pois as oito horas que passo em frente ao computador são completamente focadas no que estou fazendo. As distrações como redes sociais, papos aleatórios com os colegas, idas a cafeteria mais próxima para pegar um café, tudo isso ficou relegado ao passado.

Um dos meus mentores pessoais, além de dar ótimos conselhos profissionais, certa vez me falou algo que me marcou: em se tratando de filhos a qualidade do tempo que você passa com eles é mais importante do que a quantidade. Quando eu estou com a Alice eu tento ao máximo desligar de tudo

o mais, telefone, TV, outros pensamentos. Passar o tempo com ela além de fazer bem para a educação da Alice também serve como uma ótima forma de diminuir o stress do trabalho.

Ter uma rotina é algo que eu sempre defendi como importante para a vida e com uma criança pequena isso se torna ainda mais relevante. A Alice tem hora para ir para a escola, para acordar, comer, dormir. E com isso bem definido fica fácil também organizar os meus períodos de trabalho, estudo, recreação, descanso, etc. Isto também ajuda a organizar os horários com os compromissos de trabalho. A equipe da Coderockr, onde trabalhei durante os primeiros três anos da Alice, e agora a da Codenation sempre foi compreensiva ao entender que eu tenho compromissos com a família em determinados horários. E com isso conseguimos marcar reuniões, eventos e viagens com antecedência e encontrando uma forma que seja legal para todos.

É claro que cada pessoa tem realidades diferentes. Eu jamais conseguiria ter esse equilíbrio entre a vida pessoal e profissional sem o apoio incondicional da minha esposa. Imagino a complexidade que é fazer isso sem uma "rede de apoio", um(a) parceiro(a) ao lado e tenho muita admiração pelas pessoas que fazem isso sozinhas pois são heróis.

Full Stack vs Full Cycle developer

Full Stack developer

Nos últimos anos o termo *full stack developer* ganhou destaque na descrição de vagas de emprego, especialmente em startups. Segundo este post[39], um *full stack developer* é (tradução minha):

> [...] um engenheiro que pode dar conta de todo o trabalho, desde bancos de dados, servidores e a parte cliente da aplicação. Dependendo do projeto, o que os clientes necessitam pode ser uma aplicação mobile, web ou desktop.

E o post segue citando tudo que seria necessário aprender: uma série de linguagens de programação, bancos de dados, backend, frontend, cache, design, usabilidade, etc, etc.

Se isso é complicado para uma pessoa que é desenvolvedora há vários anos, imagine o desespero que pode causar em alguém que está iniciando na carreira. Recentemente fui convidado a participar de um papo sobre desenvolvimento web no podcast/canal do Youtube do Robson Cristian, o Simplificando[40]. Um dos tópicos que comentamos foi exatamente isso: o que uma pessoa precisa aprender para iniciar ou evoluir na carreira web. No episódio eu dei minha opinião rápida sobre o assunto, que estou aprofundando neste post. O que eu acredito é que no início da nossa carreira é natural sermos *full stack* porque precisamos ganhar mais experiência antes de escolhermos um caminho.

Antes de continuar minha opinião, vamos ver o que é *full cycle developer*

Full Cycle Developer

O termo ganhou destaque em um post do blog de tecnologia do Netflix[41], em meados de 2018.

No post o autor conta os problemas que eles estavam enfrentando e os experimentos que realizaram até chegar a este modelo (tradução minha):

> [...] nós chegamos a um modelo onde um time, equipado com ferramentas de produtividade, é responsável por todo o ciclo de desenvolvimento do software: análise/arquitetura, desenvolvimento, teste, deploy, operação e suporte.

Ok, ok, nem todo mundo tem os problemas de performance, escalabilidade ou complexidade que o Netflix tem, mas não é este o ponto principal. Substitua "time" por "desenvolvedor(a)" e chegamos ao ponto que me fez escrever este post.

[39] https://hackernoon.com/6-essential-tips-on-how-to-become-a-full-stack-developer-1d10965aaead
[40] https://robsoncristian.com/simplificando-episodio-13-programacao-web-com-elton-minetto/
[41] https://medium.com/netflix-techblog/full-cycle-developers-at-netflix-a08c31f83249

Ao invés de aprender superficialmente todas as tecnologias envolvidas no desenvolvimento de um software (backend, frontend, banco de dados, usabilidade, etc) me parece muito mais eficaz uma pessoa que é capaz de iniciar uma tarefa e conseguir ser responsável por ela até o final do seu ciclo. É o que venho experimentando nos últimos oito ou nove anos, enquanto estava a frente da tecnologia da Coderockr[42] e agora na Codenation[43]. Neste tempo todo sempre trabalhei com especialistas em suas áreas, seja mobile, frontend ou backend, mas que eram capazes, com conhecimento e ferramentas, de serem responsáveis por todo o ciclo de desenvolvimento: definir arquitetura/análise de requisitos, desenvolver o software, testar, fazer o deploy e resolver problemas em produção.

Eu acredito que esta é uma forma mais saudável de evoluir em uma carreira em desenvolvimento de software, pois gera profissionais responsáveis, especialistas em suas áreas, capazes de trabalhar em equipe (junto com outros especialistas) e mais completos.

[42] http://coderockr.com
[43] https://www.codenation.com.br

Como evoluir na carreira de dev?

Algum tempo atrás meu sócio da Codenation[44] me fez uma pergunta interessante:

> "Se dinheiro não fosse uma limitação, o que você faria para evoluir na carreira de dev?"

Para responder essa pergunta/provocação eu precisei parar um pouco e pensar o que eu acredito ser necessário para evoluir na carreira de desenvolvedor(a) de software.

Na minha opinião, são dois fatores que fazem esta evolução: educação e experiência.

Quando eu falo educação não estou me referindo apenas a uma graduação, pós-graduação ou mestrado. Mas sim qualquer investimento que a pessoa possa fazer para aumentar a sua educação na área. Pode ser um livro, um curso, horas investidas na leitura de posts, documentação de linguagens/frameworks, a ida a um evento ou meetup.

Quanto a experiência, ela serve para complementar o conhecimento adquirido. Eu tive o privilégio de, no começo da minha carreira, poder passar dois anos com um salário de estagiário. Eu tinha 17 anos, morava com meus pais e eles pagavam meus estudos na universidade. Aproveitei este tempo para fazer estágios como atendente de laboratório de computação, manutenção de computadores, suporte de redes e, finalmente, como desenvolvedor. Isso fortaleceu muito o que eu estava estudando. Depois disso eu sempre procurei por trabalhos e projetos que me dariam experiência nas áreas que eu tinha interesse: gerenciamento de equipes, arquitetura de software, novas tecnologias, empreendedorismo, etc. Claro que nem todo mundo pode ter essa sorte que eu tive, mas é importante sempre tentar adquirir o máximo de experiência que tiver ao nosso alcance. Assumir novos projetos na empresa que trabalhamos, ou novos empregos, ou participar de projetos open source, etc.

Então, respondendo a pergunta do meu sócio:

> se o dinheiro não fosse um limite eu investiria em educação e experiência, provavelmente em cursos e projetos fora da minha área de conforto, talvez em outro país

E você? Qual seria sua resposta para a mesma pergunta? E o que você considera os fatores para crescimento na carreira?

[44] https://codenation.dev

Twitter, uma ferramenta importante para devs

Algum tempo atrás tive a oportunidade de participar de um evento internacional, o WebSummit, em Lisboa.

Assisti palestras de diversos assuntos, desde cloud, data science, robótica, empreendedorismo e liderança. Mas uma coisa a grande maioria das apresentações, especialmente as mais técnicas, tinha em comum: o palestrante estampava sua conta do Twitter nos slides.

Eu fico surpreso como devs brasileiros usam pouco esta relevante rede social. Alguns motivos pelos quais eu acredito que devs deveriam usar mais a rede do passarinho azul:

- facilidade de encontrar e conversar com as pessoas mais relevantes da nossa área. Quer saber o que o criador do Laravel[45] ou do Symfony[46] estão preparando para estes frameworks? Saber das novidades da próxima versão da linguagem Go[47]? Basta seguir as pessoas que fazem parte destes e de outros projetos.
- Objetividade. Por ser uma rede baseada em pequenas mensagens, atualmente 280 caracteres, os usuários tendem a serem mais objetivos na comunicação. Ou seja: sem textão! E é fácil dar uma passada rápida no seu feed enquanto o build ou o pipeline de deploy estão executando ;)

Se consegui convencer você a investir seu tempo no Twitter dou aqui duas dicas:

- siga apenas as pessoas que são relevantes para as tecnologias que você usa. Eu tenho uma regra pessoal que é seguir no máximo 200 contas[48]. Mais do que isso eu acabo me perdendo nos assuntos. Mas isso é uma regra que funciona pra mim, crie a sua.
- Seja objetivo mas sem ser mal educado. São pessoas do outro lado da conta, mesmo que for de um projeto open source ou empresa.

Ahh, e aproveita e me segue também[49] ;)

[45] https://twitter.com/taylorotwell
[46] https://twitter.com/fabpot
[47] https://twitter.com/golang
[48] https://twitter.com/eminetto/following
[49] https://twitter.com/eminetto

O que é um Great Place to Work para você?

Recentemente entrei no site de uma empresa onde prestei consultoria para parte da equipe de desenvolvimento de software, e fiquei bem feliz em ver que eles ganharam novamente o selo "Great Place to Work"[50].

Isso me fez pensar nos pontos que tornam uma empresa um bom lugar para trabalhar, e cheguei a uma lista de quatro itens que importam para mim:

- **desafios técnicos**: preciso estar em um lugar que me desafie com novos problemas para resolver, novas formas de melhorar como profissional. Quando falo "desafios técnicos" não me refiro apenas a linguagens de programação e algoritmos, mas tudo o que está ao redor disso, desde arquiteturas, códigos, processos e pessoas.
- **propósito**: parece um clichê, mas trabalhar em algo em que eu acredito me motiva muito.
- **pessoas**: gosto muito da frase *"se você é a pessoa mais inteligente da sala você está na sala errada"*. Estar rodeado de pessoas com quem eu consigo aprender coisas novas, tanto técnicas quanto outras áreas, é algo que me fascina. E poder ter um bom relacionamento com estas pessoas é fundamental, lógico.
- **compensação financeira**: porque no fim do dia é o que paga nossas contas, certo? Receber um salário/benefícios justos pelo trabalho realizado é algo que todo profissional procura e merece.

No decorrer dos anos sempre procurei trabalhar em empresas que me proporcionassem isso. E quando tive a oportunidade de ser sócio da empresa sempre tentei ao máximo criar ambientes assim, com diferentes níveis de sucesso.

E você? Quais são os pontos que fazem você dar seu selo pessoal de "Great Place to Work"?

[50] https://gptw.com.br

Dicas para desafios técnicos

Na última década ou mais, uma das tarefas mais importantes que eu tenho desempenhado é a contratação de pessoas para trabalharem como dev nos times que eu gerencio. Parte do processo de contratação geralmente é a análise de código de um desafio técnico e neste capítulo vou citar algumas dicas que podem ser úteis para quem está avaliando ou sendo avaliado.

As expectativas em relação ao código mudam bastante de acordo com o nível de senioridade da vaga. Como na nossa área não existe um consenso sobre o que é júnior, pleno e sênior, eu vou usar a definição a seguir, que encontrei neste post[51] e que eu traduzi livremente.

> Quando você é júnior, a "coachabilidade" **(OBS: não encontrei uma tradução melhor, mas seria a capacidade de ser treinado)** é o que o faz ser contratado. Com que rapidez podemos treiná-lo para ser eficaz?
>
> Quando você é pleno, habilidades técnicas fazem com que você seja contratado. Você pode fazer um trabalho útil agora e preencher rapidamente as lacunas de conhecimento?
>
> Quando você é sênior, sua experiência e opiniões fazem com que você seja contratado. A empresa não sabe como resolver seus problemas. Eles nem conhecem os problemas. Eles apenas sabem que dói. Por que não estamos entregando? Por que nossa arquitetura está quebrada? Por que não podemos escalar?

Geralmente vagas procurando por pessoas sênior não envolvem desafios técnicos sendo que a entrevista é a parte mais importante. Por isso vou focar as dicas nos dois primeiros perfis.

Quando eu estou avaliando uma pessoa com pouca experiência um ponto que eu acho importante é tentar entender o raciocínio que ela usou para escrever o código. Uma forma de demonstrar isso no desafio técnico é organizando o histórico de commits do código. Antes de começar a programar, tente quebrar o problema em partes menores e fazer commits incrementais demonstrando como você se organizou para resolver o problema. Fez primeiro todas as tarefas do backend? Implementou toda a lógica e depois fez a parte visual? etc.

Isso mostra muito sobre sua forma de pensar e se organizar, o que é bem importante para quem está iniciando na área. Uma dica legal é dar atenção às mensagens de commit e para isso eu sugiro o uso de um padrão como o Conventional Commits[52].

Outra dica é usar bem os comentários nos códigos para demonstrar seu raciocínio e decisões. Mas ao invés de escrever no comentário **o que** o código, faz prefira falar sobre **o porque**. Por que você quebrou o código em funções? Qual o motivo de ter escolhido esta abordagem e não outra? etc.

[51] https://swizec.com/blog/youre-not-asking-for-a-job-youre-selling-a-service
[52] https://www.conventionalcommits.org/pt-br/v1.0.0/

A próxima dica é sobre o uso ou não de frameworks, especialmente se você for pleno. Se a vaga pede especificamente experiência em determinado framework é esperado que você o use e será cobrado por isso. Mas se a vaga tiver uma descrição mais aberta, não fazendo menção a nenhum framework específico, a escolha da ferramenta vai contar bastante na sua avaliação.

Neste caso eu sugiro que você evite usar frameworks "full stack" como o Laravel, Rails, Django, Buffalo, etc. Eles são ótimos frameworks, mas várias das decisões importantes já foram tomadas pelos seus criadores, como arquitetura, componentes, padrões de código, estrutura de diretórios, etc. E isso vai tirar uma boa oportunidade de demonstrar seus conhecimentos. Prefira escolher micro-frameworks como o Lumen, ou bibliotecas como o Gin, para citar apenas dois exemplos, em PHP e Go. Escolhendo estes micro-frameworks/bibliotecas permite que você demonstre as decisões importantes como estrutura de diretórios do projeto, arquitetura escolhida, formato dos testes, etc. E isso conta muito para alguém nos níveis pleno e sênior.

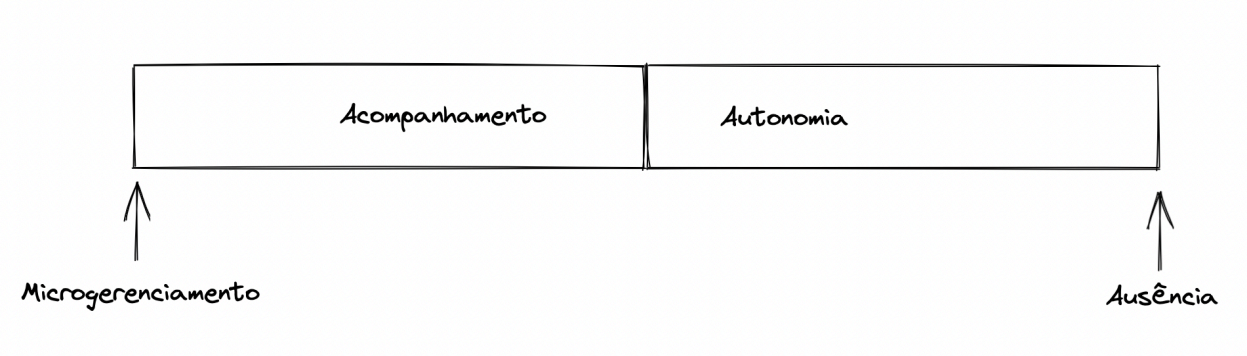
E quanto a testes? Para vagas júnior eu considero a existência de testes no código um bônus muito valioso e que conta muitos pontos. Mas para uma vaga pleno a ausência de testes pode ser considerada uma falha bem grave! Não é necessário cobertura de testes de todo o código, mas procure escrever códigos para as partes mais importantes. A decisão do que é mais importante na sua concepção mostra bastante sobre sua maturidade, então analise bem o que e como testar para encantar a pessoa que está fazendo a avaliação.

Para finalizar, mas não menos importante, capriche no README.md (ou outro nome que quiser dar para a documentação do projeto). Descreva os passos que a pessoa precisa fazer para executar o seu código, fale sobre as decisões e escolhas que fez, acrescente um *To do List* com as coisas extras que gostaria de implementar ou que você tentou fazer e não teve tempo, etc. Isso vale para desafios de qualquer nível e mostra bastante sobre sua organização, sua preocupação em facilitar a vida dos colegas, etc.

Liderança técnica: acompanhamento X autonomia

Ao olhar para minha carreira eu posso dizer que tenho mais experiência e bagagem teórica em assuntos relacionados a tecnologia, em especial na área de desenvolvimento de software. Isso devido a ter feito uma graduação e pós-graduação em Ciência da Computação, bem como ter lido uma boa quantidade de livros e muitos posts, além de ter colocado um grande número de produtos em produção. Digo isso pois, apesar de vir liderando times técnicos a pouco mais de uma década e lido bons livros e posts, eu me sinto mais seguro em escrever sobre tecnologia do que sobre liderança. Então, leve isso em consideração ao ler este capítulo :)

O que eu gostaria de compartilhar neste texto é um pequeno "framework" que venho usando regularmente nas minhas reuniões de 1-1 com as pessoas dos meus times. Eu tento descrever para eles dois aspectos da liderança, que demonstrei nesta imagem:



De um lado temos o acompanhamento que uma liderança deve fazer em relação ao trabalho e carreira das pessoas. E do outro temos a autonomia que as pessoas precisam para realizar seu trabalho. Nos dois extremos do espectro estão duas características que eu acho muito nocivas em uma liderança: o microgerenciamento e a ausência. O que eu pergunto regularmente para as pessoas que eu lidero é se eu não estou me aproximando demais de algum dos extremos. Meu objetivo é sempre ter um equilíbrio entre acompanhamento e autonomia, mas esse pêndulo pode pender um pouco para um dos lados, dependendo de coisas como a maturidade do time, do negócio, etc. Mas em nenhum cenário eu vejo motivo para uma liderança chegar a um dos extremos desenhados acima.

Mas porque esse texto em um livro sobre dicas de carreira para devs? Pode ser que em algum momento você se encontre na encruzilhada entre a carreira de dev ou de gestão (assunto do próximo capítulo) e isso pode ser útil para você. Caso não queira se tornar gestor de pessoas, compartilhe esse texto com sua liderança, e isso pode ser útil para ambos ;)

Carreira em Y

Provavelmente você conhece a fábula da pessoa que era desenvolvedora sênior e foi "promovida" a gerente e não se saiu tão bem no novo cargo. Talvez você já tenha trabalhado com alguém assim, ou isso já tenha acontecido com você. Até alguns anos atrás o curso natural da carreira de desenvolvimento de software era depois de sênior você se tornar gerente, líder técnico, ou outro título similar.

Felizmente isso tem mudado, graças a famosa carreira em Y.

Um pouco de história

Para ilustrar vou contar a minha história...

Eu comecei a trabalhar como desenvolvedor de software por volta de 1998, enquanto fazia a graduação em Ciência da Computação. Em 2008 eu já era considerado desenvolvedor sênior, com experiência em uma série de projetos diferentes e neste momento resolvi mudar de empresa (sim, trabalhei todos esses anos no mesmo lugar, algo impensável para os dias de hoje mas que foi incrível para mim) e tentar o caminho de liderança.

Depois de mais de 10 anos atuando como líder técnico, empreendedor e CTO posso dizer, sem falsa modéstia, que me tornei um bom líder. Isso graças a dedicação, muitos erros e aprendizados, mentorias e a oportunidade de ter trabalhado com pessoas incríveis na (falecida) Drimio, Coderockr[53] e Codenation[54].

No começo de 2021, já na Trybe[55], eu tive a oportunidade de repensar minha carreira, graças a nossa trajetória de carreira em Y. Para ter uma ideia de como funciona uma trajetória de carreira em Y, eu recomendo a leitura deste exemplo[56], que usamos como uma das inspirações na Trybe.

Nesse tempo todo em que eu passei liderando equipes, eu sempre atuei como líder de times relativamente pequenos, de no máximo 10 pessoas. Isso me permitiu praticar tanto a parte de soft skills (gestão de pessoas e conflitos, gestão de riscos e stakeholders, planejamento de carreira das pessoas, etc) quanto os técnicos (decisões de arquitetura, padrões de código, provas de conceito, code reviews, etc). Mas para dar o próximo passo na carreira de gestão eu precisaria fazer algo novo para mim: ser gestor de líderes técnicos, me dedicando cada vez mais a área de soft skills/gestão e me afastado da parte técnica.

E eu tomei a decisão de não seguir o caminho da gestão e sim voltar para o caminho de "*contribuinte individual*". Eu cheguei a essa decisão usando de algumas "ferramentas":

[53] http://coderockr.com
[54] https://codenation.dev
[55] https://betrybe.com
[56] https://docs.google.com/spreadsheets/d/1k4sO6pyCl_YYnf0PAXSBcX776rNcTjSOqDxZ5SDty-4/edit#gid=0

- **Mentorias**. Procurei pessoas que seguiram os dois caminhos, entrei em contato e pedi se elas poderiam dedicar alguns minutos do seu tempo para me contarem como tomaram a decisão, como foi o processo, o que gostam e não gostam da área, etc. Graças a anos de dedicação em networking, consegui conversar com pessoas incríveis, tanto de empresas pequenas quanto gigantes como Netflix, Google e Digital Ocean.
- **Leituras**. Li muitos posts e livros sobre o assunto. Dois que eu posso recomendar são: Business model you: o modelo de negócios pessoal[57] e Staff Engineer: Leadership beyond the management track[58]. O primeiro me ajudou a analisar minha carreira e entender o que eu mais gosto de fazer e onde eu posso gerar mais impacto. O segundo é sobre pessoas que seguiram o caminho de contribuidores individuais e continuam atuando como líderes e mentores técnicos.
- **Conversas com minha liderança**. Uma das coisas legais em trabalhar na Trybe é a preocupação que as lideranças tem com a carreira das pessoas. Conversei bastante com o João Daniel, CTO da Trybe, sobre os meus planos e como a empresa poderia me ajudar a atingi-los, bem como os impactos que eu poderia gerar em cada passo do caminho.

E em Maio de 2022 eu fiz a transição para este "novo" caminho de carreira ao me tornar Principal Software Engineer no PicPay[59] onde eu continuo liderando pessoas desenvolvedoras, resolvendo problemas técnicos cada vez mais complexos, mentorando e ajudando outras pessoas a evoluirem em suas carreiras.

Além desta transição eu comecei um novo projeto pessoal. Trata-se da newsletter Mais que Senior[60] onde entrevisto pessoas que seguiram a carreira técnica ao invés da gestão. É inspirador conhecer suas trajetórias e aprendizados. O convido a ler as entrevistas e assinar a newsletter para receber as próximas edições.

A "moral da história" que podemos tirar desse meu caso é que não existe um caminho só a ser seguido e sempre é possível fazer transições entre as vertentes. Sempre observe como você se sente, qual é o cenário ao seu redor e como você pode se adaptar para potencializar sua carreira.

[57] https://www.amazon.com.br/dp/8576087790/ref=cm_sw_r_tw_dp_2FQXCQ3GTE043V012M10
[58] https://www.amazon.com.br/dp/B08RMSHYGG/ref=cm_sw_r_tw_dp_WKX9P3JNTEV9503Z17HG
[59] https://picpay.com
[60] https://maisquesenior.dev

Crie um brag document

Se você está no mercado de trabalho provavelmente já passou por um cenário parecido com os seguintes:

> A empresa onde você trabalha possui um processo formal de avaliação para promoções, o tal do "Performance Review".

ou

> Você vai direto conversar com sua liderança em busca de uma promoção ou aumento de salário.

Em ambos os casos é um momento onde você precisa exercitar sua capacidade de negociação, precisa mostrar para a empresa o quanto você tem evoluido e gerado de valor.

Nestes momentos o melhor argumento é você poder mostrar uma lista de contribuições que você fez, afinal sua liderança pode não lembrar de todos os detalhes pois você não é a única pessoa liderada por ela. É aqui que brilha o conceito do "brag document", que em tradução livre seria o seu "documento para contar vantagem" :)

Essa ideia ficou famosa a partir de um post da Julia Evans[61] e foi citado em outros lugares, como no livro Staff Engineer: Leadership beyond the management track[62].

Apesar do termo parecer estranho não é nada muito complexo, apenas um documento que você atualiza constantemente com todas as coisas que você fez e que tem orgulho, ou que geraram valor para sua equipe/empresa. E quando você for passar pelo processo de pedir uma promoção você pode levar esta lista para ajudar a sua liderança a concordar com você.

Eu comecei a fazer isso recentemente e estou gostando do resultado. O que eu fiz foi criar um documento no Notion e um evento recorrente no meu Google Calendar para que toda sexta-feira eu revise tudo o que fiz na semana e atualize o documento, caso tenha feito algo relevante.

[61] https://jvns.ca/blog/brag-documents/
[62] https://www.amazon.com.br/dp/B08RMSHYGG/ref=cm_sw_r_tw_dp_WKX9P3JNTEV9503Z17HG

Brag Document

Table + Add view Filter Sort New

Eventos

Name	Data	Descrição	Link	Tags
Escrevi um post sobre brag documents	April 14, 2022	Post sobre brag documents	https://eltonminetto.dev/post/2022-04-14-brag-document/	site carreira
Dei um curso básico de Go para uma disciplina de pós-graduação	April 2, 2022	Disciplina de 4 horas sobre introdução a Go		aulas pós go
Criei um workflow para gestão de incidentes na Trybe	March 31, 2022	Workflow criado para gerenciar os incidentes da empresa via Slack		trybe
Entreguei a decisão de documentarmos os bancos de dados no formato DBML	March 25, 2022	Entreguei a decisão da empresa documentar os bancos de dados no formato DBML		trybe
Palestra Arquitetura de Software e a Clean Architecture na empresa Unico	March 22, 2022	Palestra para a equipe interna da empresa Unico		eventos
Terminei o curso de Helm da Cloud Academy	March 16, 2022	Curso sobre o Helm	https://cloudacademy.com/course/introduction-to-helm-1034/course-introduction/?context_resource=lp&context_id=2065	treinamento
Publiquei o post sobre Generics em Go em inglês	March 15, 2022	Tradução do post de generics para ingles	https://eltonminetto.dev/en/post/2022-03-11-fun-with-generics/	site go
Masterclass de Arquitetura de Software e Clean Architecture	March 15, 2022	Palestra sobre arquitetura que apresentei para os estudantes da turma 13 da Tryb		arquitetura eventos trybe
Escrevi post sobre Generics em Go	March 11, 2022	Post sobre generics para meu site	https://eltonminetto.dev/post/2022-03-11-fun-with-generics/	site go
Terminei o curso de Kubernetes da Cloud Academy	March 11, 2022	Curso introdutório a Kubernetes	https://cloudacademy.com/course/introduction-to-kubernetes/results/	treinamento kubernetes
Participei de live sobre Arquitetura de Software	March 10, 2022	Participei de uma live sobre arquitetura de software	https://www.youtube.com/watch?v=n0OC8GGm3n8	arquitetura eventos
Apresentei o status dos projetos do meu squad na Trybe	March 10, 2022	O time recebeu elogios pelas entregas e pelo andamento dos projetos		liderança trybe
Organizei Meetup de Go da Comunidade de SC	March 9, 2022	Organizei e apresentei o evento	https://www.youtube.com/watch?v=XL9WLeKWuZ0	go eventos
Palestra Arquitetura de Software e a Clean Architecture na Guilda de Backend	March 3, 2022	Palestra para a equipe de desenvolvimento		eventos trybe

Outra vantagem é que o fato de revisar minhas entregas e anotar as mais relevantes também tem me ajudado a comemorar as vitórias do dia a dia. Isso ajuda muito a não cair na armadilha da síndrome do impostor.

Claro que não é toda a semana que tenho algo relevante para anotar, mas o processo de estar constantemente pensando na minha carreira tem me ajudado bastante. Espero que esse texto também te ajude a acompanhar suas conquistas.

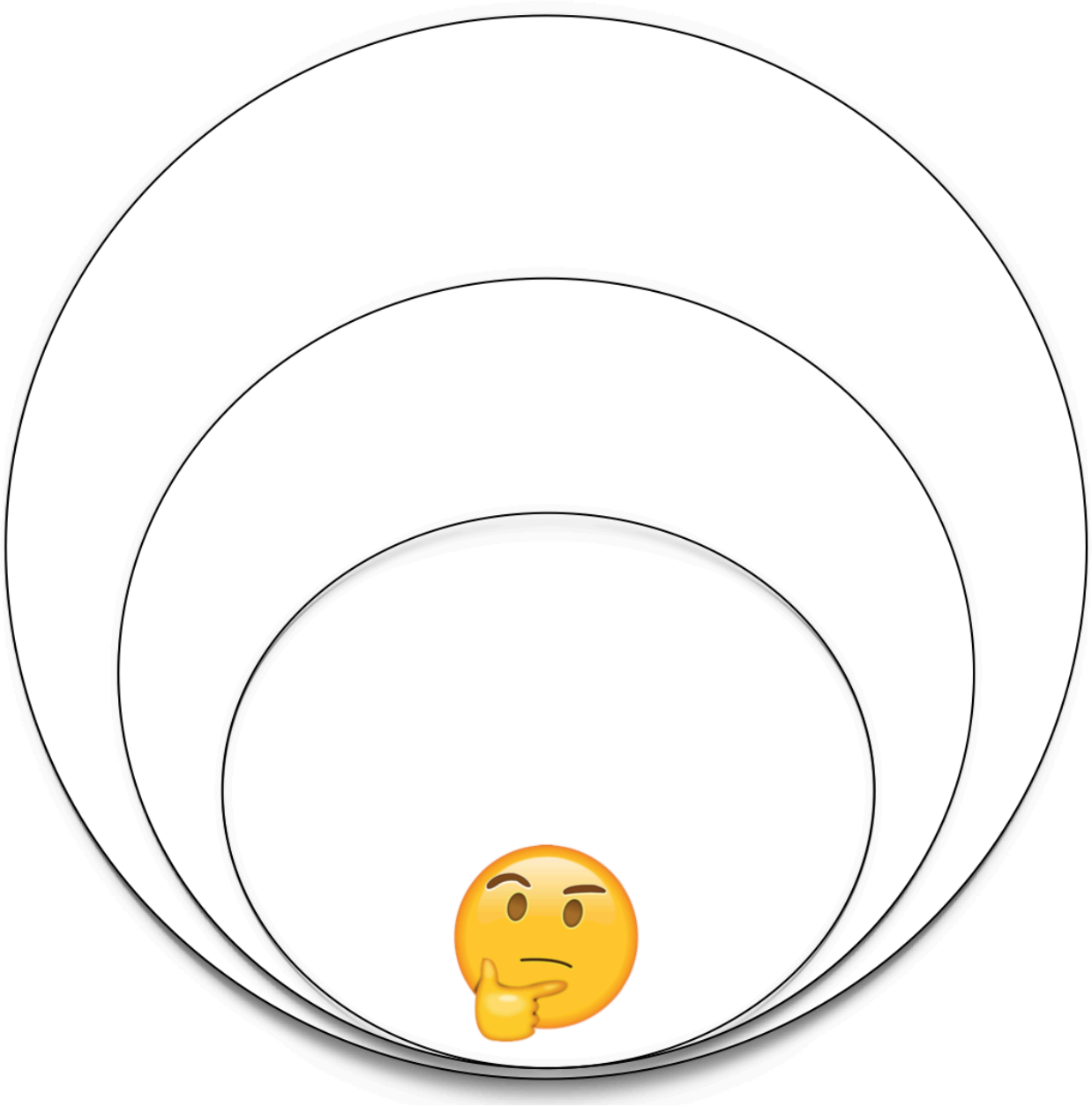
Como decidir o que estudar?

Frequentemente converso sobre carreira com colegas de profissão, tanto pessoas iniciantes quanto mais experientes e uma pergunta que aparece bastante é:

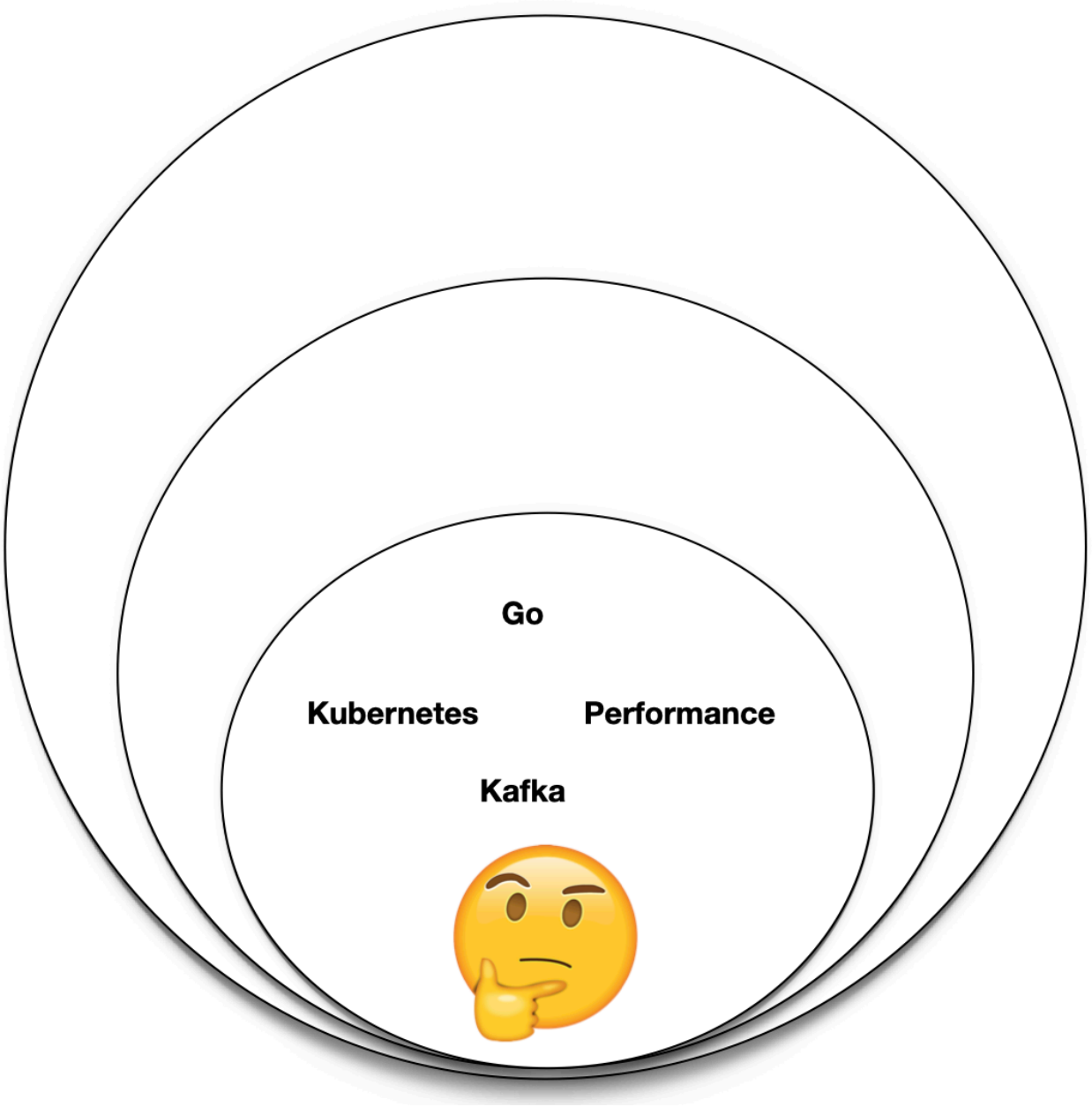
> Dentre tantas opções de assuntos e tecnologias existentes, como eu escolho o que estudar primeiro?

Para tentar responder essa dúvida eu imaginei um processo, um “framework” talvez?

O primeiro passo é exercitar seus dons artísticos (coisa que eu obviamente não tenho) e criar uma série de círculos, com você no centro:

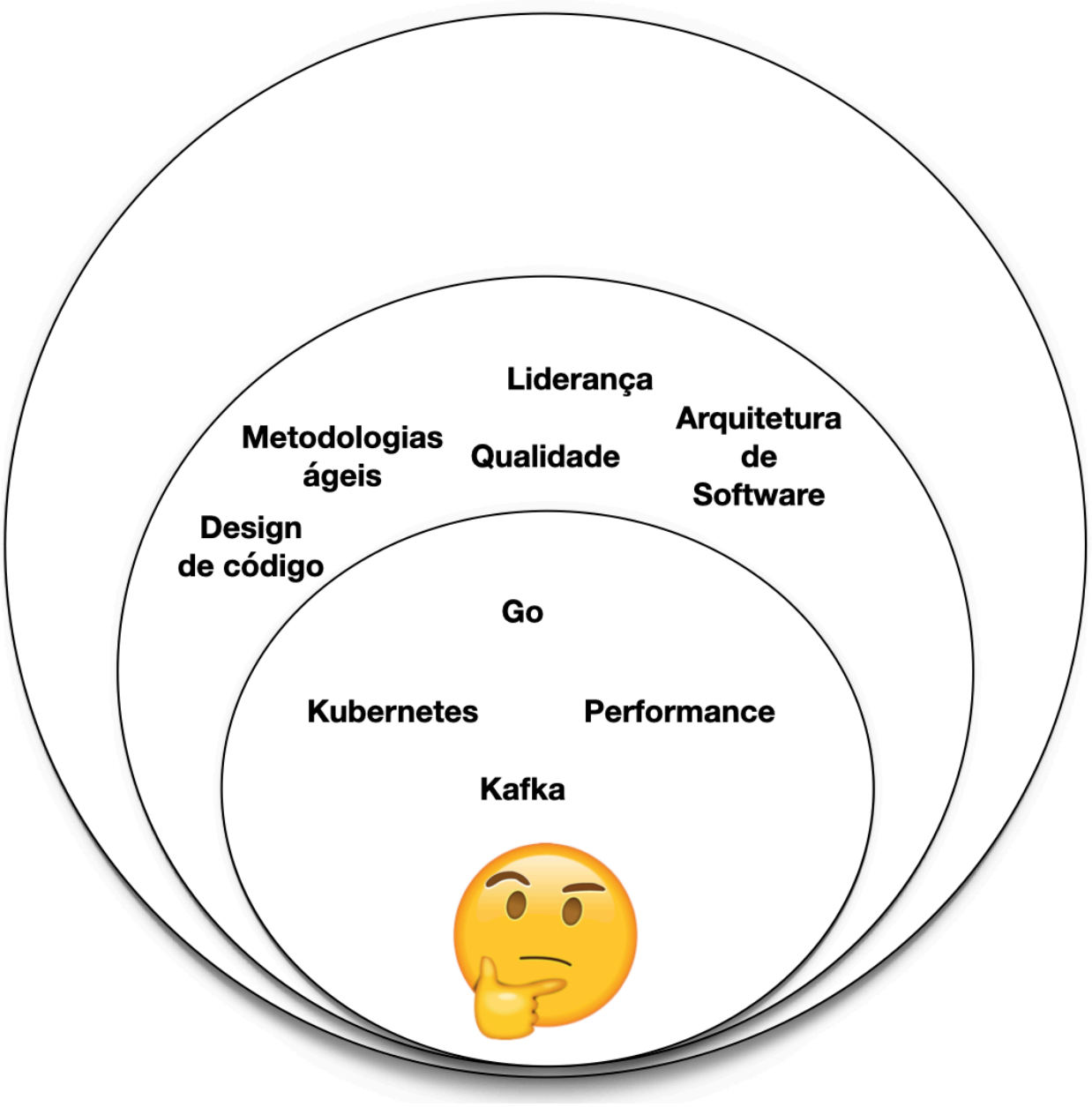


No primeiro círculo ao seu redor, anote os tópicos/tecnologias que, se você se especializar, vão causar o maior impacto possível na sua carreira neste momento. Para exemplificar, vou fazer o meu desenho:



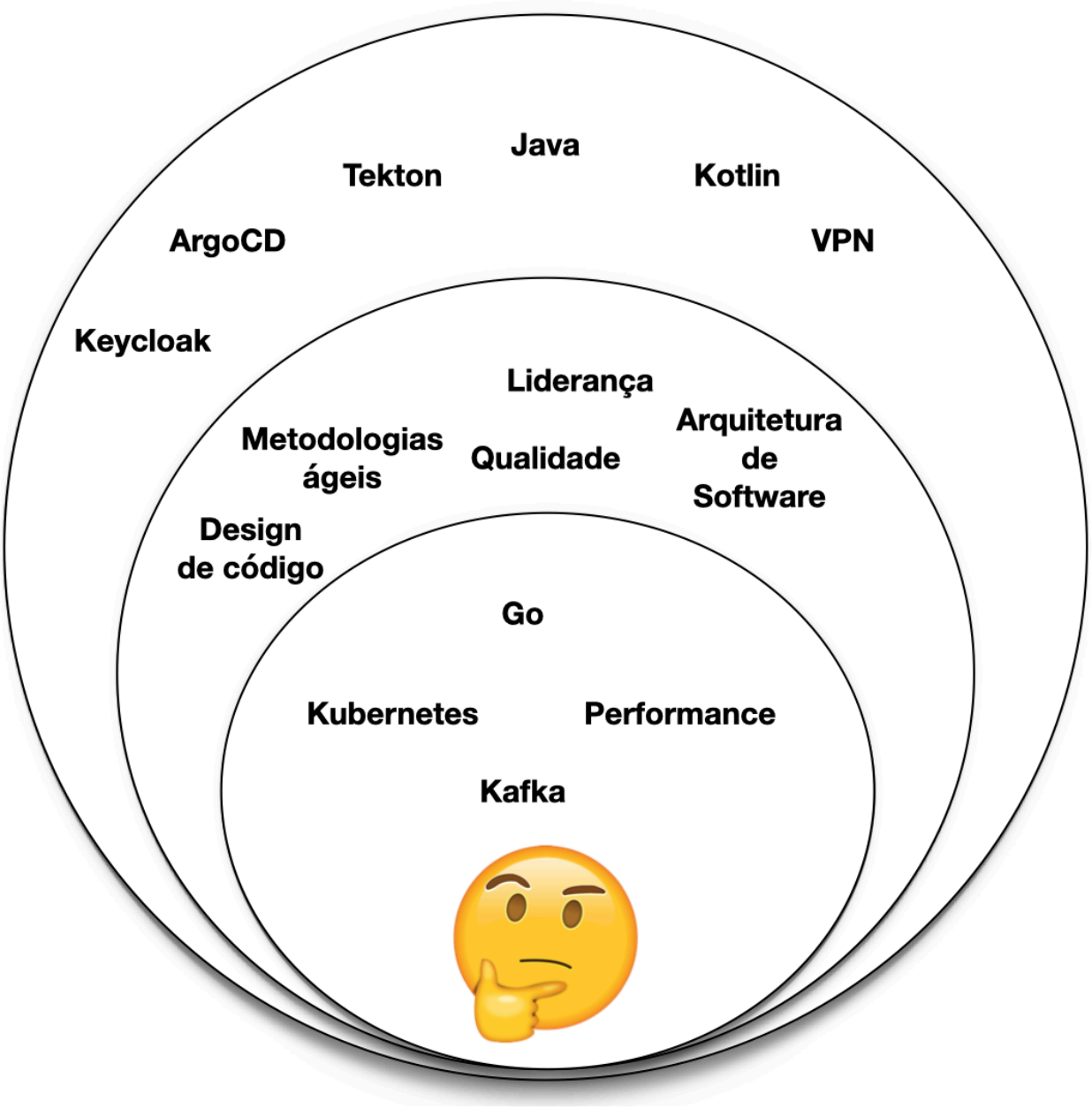
Neste momento, em Junho de 2022, os tópicos que causam mais impacto no meu desempenho são Go, Kubernetes, Performance e Kafka. São os assuntos que eu vou me aprofundar muito nas próximas semanas e meses, pois são os que vão ajudar nos meus resultados mais imediatos.

A seguir podemos preencher o próximo círculo:



São assuntos igualmente importantes e interessantes, mas eu vou dar menos foco neles a curto prazo. Sempre que eu encontrar algum tópico relacionado a eles eu vou salvá-lo em um bookmark, documento do Notion, To Do list, etc, para me aprofundar em outro momento.

E o último círculo eu preencho com assuntos que não vou me aprofundar muito neste momento, mas quero manter no meu "radar", tendo um conhecimento superficial, mas que sei que serão importantes em breve.



Assuntos que estão fora deste radar eu vou apenas observar de longe, no máximo saber para o que servem, quais suas vantagens, etc. Por exemplo, todo o universo "crypto" ou mesmo o desenvolvimento frontend, apesar de interessantes e importantes, não vou observar agora, pois eles não causam tanto impacto no meu desempenho a curto e médio prazo. Com isso eu ganho foco e diminuo a síndrome de FOMO[63].

Outro ponto importante é revisar este gráfico conforme você evolui na sua carreira, ou quando muda de time ou empresa. Ou mesmo ter mais de um diagrama, talvez um para o trabalho e outro para

[63] https://pt.wikipedia.org/wiki/S%C3%ADndrome_de_FOMO

seus hobbies? A velocidade com que você cria novos diagramas vai depender de vários fatores, como novos projetos, sua velocidade de aprendizado, etc. Uma sugestão extra é você guardar os diagramas conforme vai criando novos, assim você pode ter um histórico do que focou em cada momento da sua carreira.

Espero que esta técnica seja útil para você, assim como tem sido importante para mim.

P.S.: aceito sugestões de nomes para este diagrama :)

Documente primeiro

É um consenso na comunidade de desenvolvimento de software que documentação é algo muito importante. Mas ao mesmo tempo não é uma das tarefas mais glamurosas, especialmente se comparado com a escrita de códigos. Então é natural que uma ideia nos venha à mente:

> e se eu gerar a documentação a partir do código fonte??

Eu já usei essa abordagem em alguns projetos, inclusive escrevi um post[64] sobre isso alguns anos atrás.

Mas nos últimos anos eu adotei uma abordagem um pouco diferente, que eu chamo de "documentation first". Vou ilustrar com um exemplo.

Imagine uma equipe formada por uma pessoa desenvolvedora backend e outra frontend. Para esta equipe é dada a tarefa:

> Desenvolver o formulário de cadastro de produtos

No momento do refinamento eu sempre sugiro que a equipe quebre essa demanda em sub-tarefas, como:

- Definir o contrato da API
- Desenvolver a API
- Desenvolver o formulário
- etc, etc

Para o contexto deste texto, apenas a primeira tarefa é importante. Nela a equipe trabalha em par com o objetivo de definir o contrato da API e ambos discutem formato de dados, se a API vai ser Rest ou RPC, autenticação, compressão de dados, e outros assuntos importantes. A entrega desta tarefa é a documentação, preferencialmente em um padrão como OpenAPI[65] ou API Blueprint[66] (meu formato preferido).

Com esta tarefa finalizada o trabalho pode ser novamente paralelizado. O desenvolvimento do backend vai focar em entregar a API que foi definida, enquanto que o frontend vai implementar o formulário sabendo o que vai enviar e receber do backend. Se a documentação for feita em um dos padrões de mercado citados acima é possível também gerar mocks e stubs, facilitando o desenvolvimento e os testes.

[64] https://eltonminetto.dev/2016/06/01/gerando-documentacao-de-apis/
[65] https://spec.openapis.org/oas/latest.html
[66] https://eltonminetto.dev/post/2017-06-29-definindo-apis-com-api-blueprint/

Claro que durante o desenvolvimento podem ser encontradas algumas arestas que não foram pensadas durante a primeira tarefa. Quando isso acontece basta um novo trabalho rápido em par para ajustar a documentação e o trabalho segue como o esperado.

Com esta abordagem encontrei algumas vantagens como:

- a equipe dedica mais tempo para pensar nos cenários em que a API vai ser aplicada, gerando maior entendimento antes de qualquer linha de código ser escrita;
- acontece um maior entrosamento entre os membros da equipe;
- a documentação mantém-se atualizada e viva.

Eu citei aqui um exemplo do desenvolvimento de uma API, mas a mesma abordagem pode ser aplicada em outros cenários, especialmente onde temos integração entre partes do sistema, como em um ambiente de microsserviços.

Como comentei anteriormente, venho usando essa abordagem nos últimos anos e o resultado tem sido bem proveitoso.

Dicas de livros sobre complexidade

Eu sempre tive a impressão de que um dos maiores males do desenvolvimento de software moderno é a complexidade. Não me refiro a complexidade dos problemas que são resolvidos atualmente, pois esses são realmente maiores do que décadas atrás. Machine learning, carros autônomos, microsserviços, IA, etc, esses cenários possuem uma complexidade inerente e pouco podemos fazer para mitigar isso. Eu me refiro a complexidade que incutimos aos nossos códigos. Já vi aplicações que eram pouco mais do que CRUDs com várias camadas e frameworks que só tornavam o desenvolvimento e manutenção tarefas hercúleas.

Recentemente encontrei dois autores que corroboram com essa minha impressão. O primeiro deles é o A Philosophy of Software Design[67], do professor John K. Ousterhout. Eu gostei tanto do livro que gerei alguns conteúdos inspirados na obra:

- Post Sobre design de software e complexidade[68] para o blog do PicPay
- Palestra[69] que apresentei em diversos eventos
- Videos no YouTube[70]
- Live[71] falando sobre o assunto no canal do Mauricio Linhares

O segundo livro é o *Code Simplicity: The Fundamentals of Software* que agora pode ser baixado de maneira gratuita no site do autor[72].

Como já falei bastante sobre o primeiro livro, vou aproveitar esse texto para comentar um pouco sobre o *"Code Simplicity"*. É um livro bem curto, 80 páginas, e bem direto. Ele dividiu o conteúdo em *Fatos*, *Regras* e *Leis* do design de software e usou as páginas do livro para detalhar exemplos e como esses conceitos se aplicam a praticamente todas as aplicações. Dentre algumas "pérolas" do livro posso destacar algumas (tradução minha):

- Fato: A diferença entre um mau programador e um bom programador é a compreensão. Ou seja, programadores ruins não entendem o que estão fazendo, e bons programadores entendem.
- Fato: Todo mundo que escreve software é um designer.
- Regra: O nível de qualidade do seu projeto deve ser proporcional ao tempo futuro em que seu sistema continuará a ajudar as pessoas.
- Entre outras.

[67] https://www.amazon.com.br/gp/product/B09B8LFKQL/
[68] https://medium.com/inside-picpay/sobre-design-de-software-e-complexidade-2df2a13f01c2
[69] https://speakerdeck.com/eminetto/reflexoes-sobre-design-de-software
[70] https://www.youtube.com/watch?v=w3kJe53pEjA&list=PL0qudqr7_CuQ5lI5rFvLD8Bh_rMPlbuUt
[71] https://www.youtube.com/watch?v=o9jjtpYF3ww
[72] https://www.codesimplicity.com/post/code-simplicity-the-fundamentals-of-software-is-now-free/

E as suas 6 leis do design de software (novamente, tradução minha) com alguns comentários

- **O propósito do software é ajudar as pessoas**.

Achei essa frase muito interessante e concordo 100% com ela. Ajuda muito a tomar decisões importantes, como priorizações e mesmo se uma feature deve ou não ser desenvolvida

- **A equação do design de software. O que demonstra que é mais importante reduzir o esforço de manutenção do que reduzir o esforço de implementação**.

No livro o autor faz uma equação para demonstrar isso, que eu deixei de fora deste capítulo de propósito, para incentivar a leitura ;)

- **A Lei da Probabilidade do Defeito: A chance de introduzir um defeito em seu programa é proporcional ao tamanho das mudanças que você faz nele**.

Ou seja: pequenas e contínuas mudanças, pull requests pequenos, agilidade

- **A Lei da Simplicidade: A facilidade de manutenção de qualquer porção de software é proporcional à simplicidade de suas porções individuais**.

Códigos bem estruturados, pequenas funções ou classes, a visão de que um software é formado por vários pequenos componentes bem construídos e testados.

- **A Lei do Teste: O grau em que você sabe como seu software se comporta é o grau em que você o testou com precisão**.

Ou seja: testes, testes, testes :)

Apenas citar as leis não tem tanto impacto quanto a leitura do livro, onde é possível ver exemplos e cenários, mas eu achei válido incluí-las neste texto para instigá-lo a ler o livro todo.

Sei que o fato de ainda não ter tradução para o português pode ser uma barreira para muitas pessoas, mas mesmo assim vejo esses livros como ótimas recomendações para pessoas desenvolvedoras de todos os níveis. Acredito que o Google Translator ajude a diminuir essa barreira e recomendo a tentativa pois o conteúdo é bem importante.

Aqueles que mantém o mundo girando

Em 2012 Scott Hanselman[73] escreveu um post que ganhou certa popularidade: Dark Matter Developers: The Unseen 99%[74]. Ele começa o seu texto definindo o que seria **Matéria escura**:

> Na astronomia e na cosmologia, a matéria escura é um tipo atualmente indeterminado de matéria que responde por uma grande parte da massa do universo, mas que não emite nem espalha luz ou outra radiação eletromagnética e, portanto, não pode ser vista diretamente com telescópios.

E continua com um trecho que resume bem o que ele quer dizer:

> Você não pode ver a matéria escura, mas temos certeza de que ela está lá. Não só está lá, mas é a maior parte do que está lá. Nós sabemos disso e não podemos ver. Nunca aparece.

Ele usou esse conceito para cunhar o termo *Dark Matter Developers* (tradução e adaptação minha):

> Eles não leem muitos blogs, nunca escrevem blogs, não vão a grupos de usuários, não estão no Twitter ou Facebook e você não os vê com frequência em grandes conferências. Onde estão estes desenvolvedores? Provavelmente fazendo seu trabalho. Talvez usando ASP.NET 1.1 em um pequeno escritório. Talvez trabalhando em uma fábrica de engarrafamento no México em VB6. Talvez eles estejam escrevendo aplicativos de calendário PHP em um grande fabricante de chips.

Essas ideias continuam ressoando em 2024, como podemos ver em outro post que teve boa repercussão, da excelente Jean Yang: Building for the 99% Developers[75].

Lembrei destes textos ao coletar material e estudar sobre um assunto que tem me interessado bastante: *"developer productivity"* (não sei se já temos um termo em português adotado pelo mercado) e é importante entender as diferenças entre as empresas, times e pessoas. Nem toda empresa é uma MANGA (sigla formada pelas iniciais das grande empresas de tecnologia Meta, Amazon, Netflix, Google e Apple), nem todo mundo precisa e deve usar as ferramentas da moda, cada time tem seus fluxos de trabalho e especificidades.

[73] https://www.hanselman.com/about
[74] https://www.hanselman.com/blog/dark-matter-developers-the-unseen-99
[75] https://future.com/software-development-building-for-99-developers/

Veja bem, não estou dizendo que você deveria parar de escrever posts, ir a conferências, gerar conteúdo em vídeo, palestrar, etc. Eu incentivo muito quem tiver interesse em fazer isso, pois sou a prova de que tem um impacto enorme na carreira. O ponto é que para cada um de nós que está no palco palestrando, para cada post escrito, cada video de dancinha no TikTok explicando programação (nem sei se existe isso e fiquei com medo de procurar) existem milhares de devs que estão fazendo suas 8 horas de trabalho diário (muitas vezes mais que isso), desligando o seu computador e vivendo outras experiências.

Por isso, levantem suas xícaras de café e saúdam todos os tipos de devs, principalmente quem está lá escrevendo o código que mantém esse mundão tecnológico girando.

P.S. eu adoro o trecho final do post do Scott Hanselman e acho que seria uma ótima forma de terminar este também:

> O Dark Matter Developer nunca lerá esta postagem no blog porque eles estão trabalhando, usando tecnologia de dez anos atrás e tudo bem. Eu sei que eles estão lá e continuarei a apoiá-los em seu trabalho.

Programação pessimista

Alguns anos atrás Sam Newman publicou o livro Building Microservices[76] que se tornou uma grande referência quando falamos em microsserviços.

Em um dos capítulos, entitulado *Microservices at scale*, e que está público neste link[77], ele faz algumas afirmações interessantes (tradução minha):

> As falhas estão em toda parte
>
> Partir da suposição de que tudo pode e irá falhar leva você a pensar de forma diferente sobre como resolver problemas.

Isso me fez refletir sobre como somos otimistas quando escrevemos nossos códigos. É bem comum escrevermos o código e os testes pensando no "caminho feliz".

Mas como o Sam fala, se partirmos da noção de que as coisas muito provavelmente vão falhar faz com que façamos perguntas importantes como:

- o que vai acontecer quando o banco de dados da aplicação parar de responder? Eu posso usar um *pool de conexões*? Ou um *cache*?
- e se a máquina (ou *pod*, *VM*, ou *container*, etc) onde o software estiver rodando ficar com falta de memória, cpu ou disco? O que vai acontecer se o sistema operacional resolver matar a minha aplicação? Eu estou preparado para fazer um *graceful shutdown*?
- e se a fila para onde eu estou enviando as mensagens ficar sobrecarregada? Posso ter um algoritmo para fazer uma nova tentativa de processamento? E estou tratando a *idempotência*?
- e se o microsserviço que eu estou usando como dependência parar de responder? Estou preparado para tratar isso via algum tipo de *circuit break* e/ou um *load balancer*?
- e se o cliente que está consumindo minha API fizer uma inundação de acessos em pouco tempo? Estou preparado para tratar isso com algum mecanismo de *rate limit*?
- etc, etc, etc

Outro ponto importante que consta neste capítulo é a noção de que nem todo serviço é igual. Alguns são mais críticos do que os outros e nestes casos precisamos ser muito mais pessimistas/cuidadosos. Em contrapartida, alguns serviços são mais simples, tem menor impacto em caso de falha. Ter essa noção é importante pois nos ajuda a determinar quanto vamos aprofundar nas salva-guardas que comentei acima.

[76] https://www.oreilly.com/library/view/building-microservices/9781491950340/?utm_source=oreilly&utm_medium=newsite&utm_campaign=microservices-at-scale-top

[77] https://www.oreilly.com/radar/microservices-at-scale/

Recomendo muito a leitura deste capítulo, mesmo se você não está usando a arquitetura de microsserviços. Mas resumindo, devemos ser mais pessimistas (ou realistas?) em relação aos cenários que estamos desenvolvendo, usando a análise de criticidade para avaliar o nível de cuidado que cada serviço precisa.

Developer productivity for fun and profit

Seja em cenários de crescimento acelerado ou mesmo no infeliz momento de *layoffs* que algumas empresas passaram, horas de desenvolvimento são um dos recursos mais caros e valiosos para as empresas. Desta forma, a produtividade e eficiência tornam-se diferenciais importantes para profissionais e times.

Mas qual é a diferença entre produtividade e eficiência? Gostei bastante da definição que encontrei neste post[78]:

> Enquanto a produtividade visa mais resultados com o mesmo esforço, a eficiência visa menos esforço, mantendo o mesmo resultado.

Nesta texto vou apresentar formas como Devs podem melhorar a sua produtividade e eficiência em tarefas e projetos, garantindo seus empregos, lucros e satisfação no trabalho.

Antes de tudo, esse conteúdo é fruto de minhas experiências como desenvolvedor, líder técnico e tech manager, assim como resultado de leituras no decorrer dos anos.

> E você não vai falar nada sobre como medir a produtividade???

[78] https://medium.com/wise-engineering/platform-engineering-kpis-6a3215f0ee14

Realmente, um dos assuntos que sempre nos vem na mente quando falamos sobre isso é "e como vamos medir se estamos melhorando?". Esse é um assunto complexo e vou deixar de fora deste capítulo, mas recomendo muito a leitura de dois materiais importante sobre isso:

- What are DORA Metrics and Why Do They Matter?[79]
- The SPACE of Developer Productivity[80]

Dito isso, vamos aos tópicos.

Domine suas ferramentas

Sou muito fã da cultura e mitologia nórdicas, e alguns anos atrás encontrei um texto que gosto muito de citar de tempos em tempos. Trata-se das leis Vikings[81] e uma delas se encaixa perfeitamente neste contexto. É a "Mantenha suas armas em boas condições", e ela se refere ao fato de que um guerreiro viking poderia entrar em combate a qualquer momento, então ter suas armas sempre em boas condições poderia ser uma diferença de vida ou morte.

Menos dramático no nosso dia a dia, as ferramentas (armas) que usamos podem ser cruciais para aumentar nossa produtividade. Dedique tempo para estudar a linguagem que você usa, a IDE, o seu sistema operacional[82]. Crie e faça uso de atalhos, crie *snippets de código*, faça scripts para tarefas repetitivas como build da aplicação[83]. Aprenda a usar o Terminal do sistema operacional, bem como criar scripts em *shell* ou usando ferramentas como o *make*.

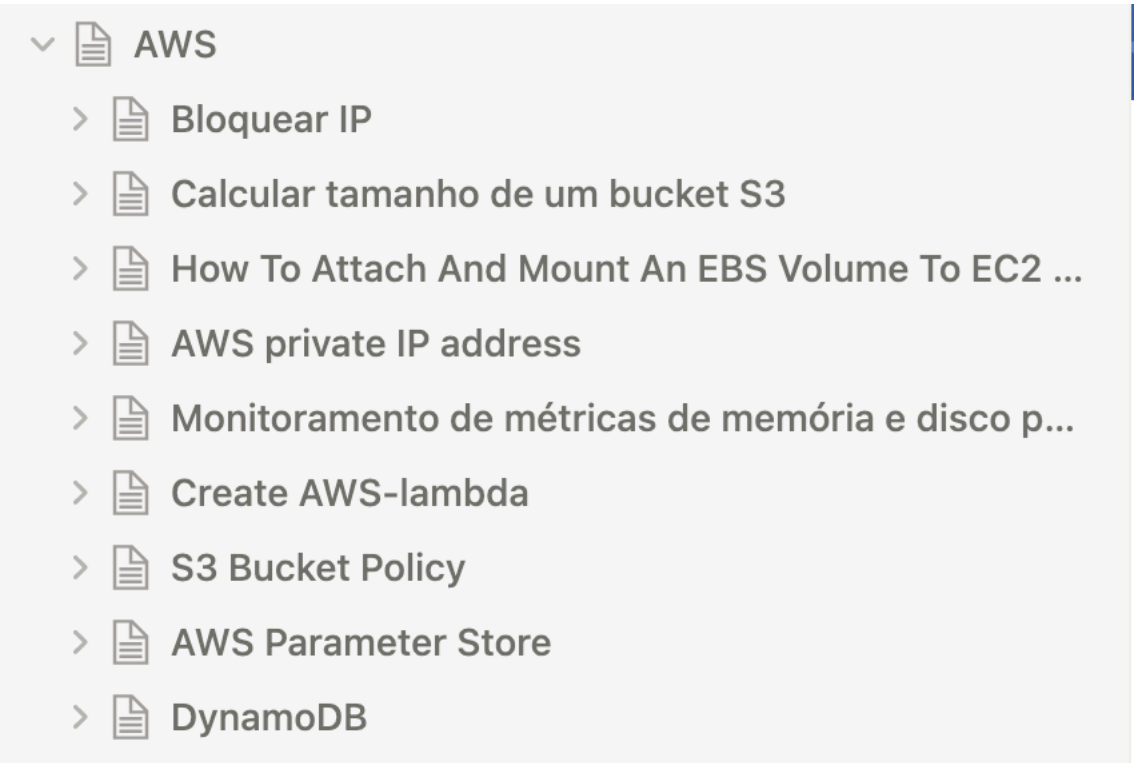
Quanto a automação de tarefas, gosto de usar como referência este post[84], de um amigo meu. Quando eu preciso executar uma tarefa a primeira vez, eu faço ela o mais rápido possível, geralmente de forma manual. Se a mesma tarefa aparecer uma segunda vez eu ainda executo manualmente, mas começo a dedicar mais atenção a ela, pois deixou de ser uma exceção e passa a se tornar uma coincidência. Se ela aparecer novamente ela se torna uma tendência, e neste momento eu crio um script para não precisar realizar o processo manualmente daí em diante. Com isso evito de automatizar coisas desnecessariamente.

Documente

Se tem uma coisa que eu aprendi com o passar dos anos é que o cérebro humano é feito para criar coisas e não guardá-las para sempre. Pelo menos o meu cérebro é assim :) Parei de confiar na minha memória e passei a anotar tudo que eu aprendo e faço. Com isso minha cabeça fica mais livre para criar coisas novas.

[79] https://codeclimate.com/blog/dora-metrics/
[80] https://queue.acm.org/detail.cfm?id=3454124
[81] https://eltonminetto.dev/2012/06/21/as-leis-vikings/
[82] https://eltonminetto.dev/post/2018-09-06-windows-linux-mac/
[83] https://eltonminetto.dev/post/2022-08-31-improve-local-development-tilt/
[84] https://medium.com/@miere/regra-da-exce%C3%A7%C3%A3o-coincid%C3%AAncia-ou-tend%C3%AAncia-b0e8be7c0a01

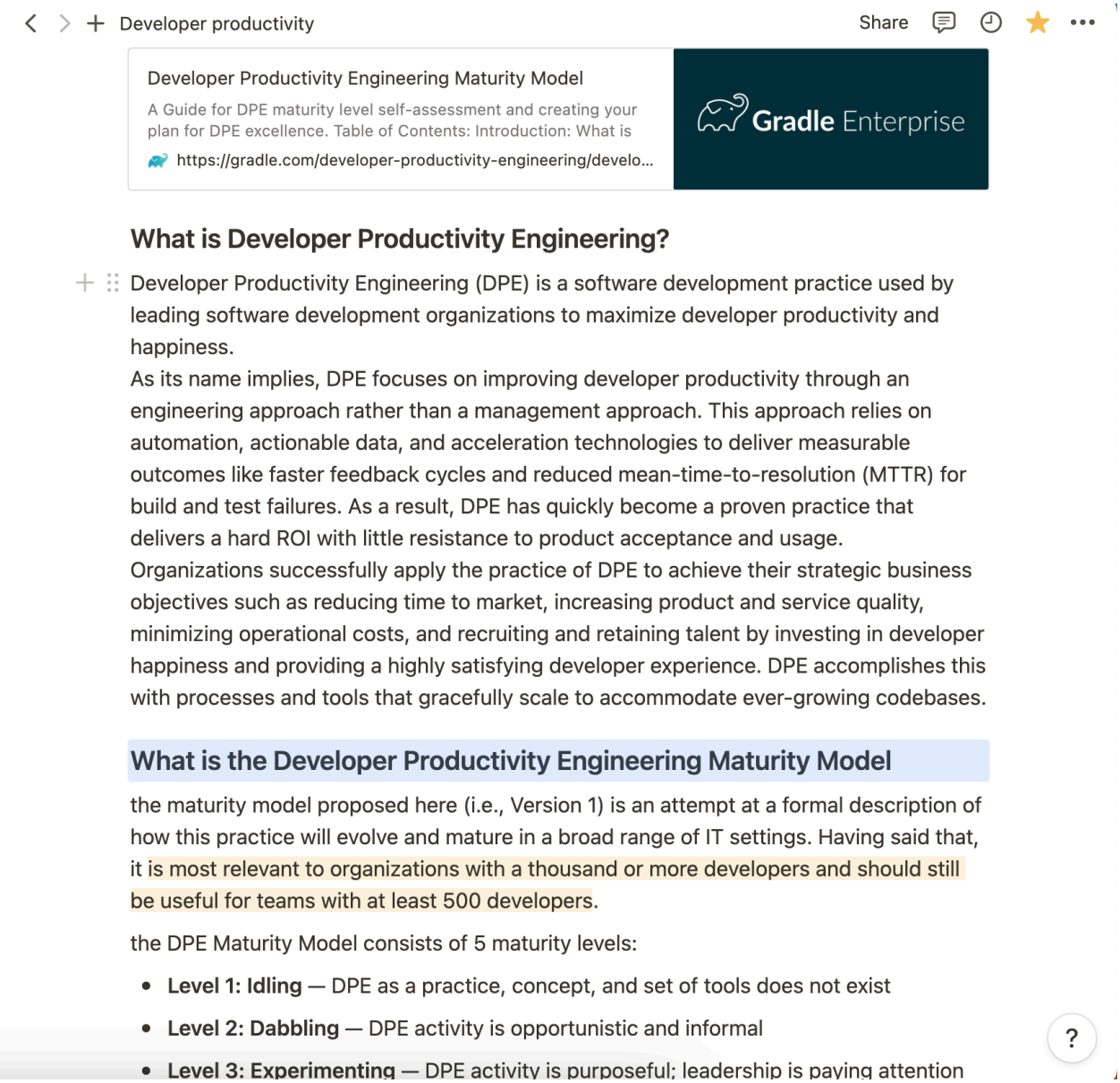
Eu sugiro que você crie um registro dos seus aprendizados. Meu site pessoal[85] surgiu para esse fim, e venho fazendo isso nos últimos 20 anos. Mas você não precisa fazer isso em público, pode anotar em um documento de texto, em um Google Docs ou Notion. O importante é que seja algo fácil de você encontrar quando precisar. Eu tenho várias anotações armazenadas, de coisas que uso no dia a dia:



Aliás, para adicionar estas imagens no texto eu procurei na minha anotação como fazer, pois não lembrava qual era o diretório correto para salvá-las ;)

Outra coisa que eu tenho feito e tem me ajudado bastante é, ao ler um post complexo ou livro, fazer anotações em um documento. Pontos importantes do texto, anotações, etc. Isso tem me ajudado a absorver melhor o conhecimento e ajuda a encontrar a informação quando eu preciso:

[85] https://eltonminetto.dev



E um último item que eu posso incluir nessa categoria é o Brag Document, que foi comentado em um capítulo anterior.

Simplifique

A complexidade é um dos maiores males da tecnologia e é algo que geralmente está sob nosso controle, pelo menos parcialmente. Falei bastante sobre isso em outro capítulo.

Com certeza eu não consegui esgotar esse assunto, e nem acredito ser possível fazer isso neste texto, mas espero que estes tópicos façam sentido para você como tem feito para a minha experiência.

Responsabilidade e disciplina

Recentemente eu li dois livros que falam sobre duas das característica que eu considero cruciais para a carreira de qualquer profissional: responsabilidade e disciplina.

Ambos os livros tem em comum um dos seus autores, o Jocko Willink[86]:

> John Gretton "Jocko" Willink é um autor americano, podcaster e oficial aposentado da Marinha dos Estados Unidos que serviu nos Navy SEALs e é ex-membro do SEAL Team 3.

Ele é co-autor do Responsabilidade extrema: Como os Navy Seals lideram e vencem[87] e autor do Disciplina É Liberdade: Manual do Campo[88].

O primeiro livro é voltado para pessoas que atuam como lideranças, mas que pode ser aproveitado por outras posições de carreira. É um livro de leitura rápida, onde cada capítulo começa com uma história real que os autores vivenciaram no exército. No restante do capítulo eles adaptam a experiência para o mundo corporativo e profissional. Isso torna a leitura bem cativante e traz uma visão importante sobre como um lider pode fazer a diferença ao assumir a responsabilidade por seus atos e os do seu time. Força o leitor a perceber que muitas vezes perdemos grandes oportunidades ao não assumir a responsabilidade e encarar os problemas/desafios de frente. Gostei muito deste livro e venho adotando alguns dos comportamentos que foram apresentados no meu dia a dia.

O segundo livro tem uma pegada um pouco mais "auto-ajuda" mas traz algumas reflexões bem interessantes. Ele é dividido em duas grande partes: Reflexões, que é minha parte favorita, onde o autor traz capítulos curtos e com algumas sugestões bem contundentes. A segunda parte, Ações, traz sugestões de alimentação e exercícios físicos para quem quer se tornar mais saudável. Eu não gostei muito desta segunda parte pois prefiro seguir as instruções da minha nutricionista e os exercícios da academa que frequento, mas mesmo assim tem algumas dicas interessantes para quem estiver interessado em entender um pouco da mente de um ex-militar. E talvez aplicar essas dicas na sua rotina de alimentação e treinos.

Apesar de trazer reflexões bem importantes, como todo livro, estes devem ser lidos com ressalvas. São escritos por uma pessoa que está em um contexto bem específico (o ex-militar norte-americano) e precisamos adequar/adaptar para nossa realidade. Nem todo mundo quer ser uma máquina de combate, mas todo mundo pode ter grandes resultados ao aplicar mais responsabilidade e disciplina no seu dia a dia. Por isso eu recomendo a leitura dos livros e a adaptação para a sua realidade, pois acredito que possa trazer grandes benefícios para sua carreira e vida pessoal.

[86] https://en.wikipedia.org/wiki/Jocko_Willink

[87] https://www.amazon.com.br/Responsabilidade-extrema-Seals-lideram-vencem/dp/8550815551/ref=sr_1_1?qid=1675865940&refinements=p_27%3AJocko+Willink&s=books&sr=1-1

[88] https://www.amazon.com.br/Disciplina-%C3%89-Liberdade-Manual-Campo/dp/6555204869/ref=sr_1_2?qid=1675865940&refinements=p_27%3AJocko+Willink&s=books&sr=1-2

Deveríamos parar de nos definir como devs backend ou frontend?

Este post é inspirado no texto[89] escrito pela Michelle Lim[90] em 2020. Vou traduzir alguns pontos que achei relevante e fazer meus comentários sobre.

> O eixo "Frontend/Backend" não mapeia bem as motivações dos engenheiros de software. Se você usar apenas esse eixo, pode acabar em projetos de que não gosta ou, pior ainda, desistir prematuramente da engenharia. Em vez disso, tente usar o eixo "Produto/Infraestrutura" como o primeiro eixo para entender sua preferência de carreira.

A minha primeira contribuição é sugerir uma mudança de termo. Desde 2023 a tendência é usarmos o termo "**Plataforma**" ao invés de "**Infraestrutura**", pois reflete melhor o caminho que o mercado e os times estão seguindo. O pessoal da LINUXtips tem um video[91] sobre isso que eu recomendo.

Dando continuidade ao post da Michelle, a seguir ela detalha um pouco mais a ideia:

> "Produto/Plataforma" mapeia perfeitamente a psicologia de como os engenheiros escolhem projetos e suas motivações para aprender a codificar. De um modo geral, existem 2 tipos de engenheiros:
>
> 1. Engenheiros "Product-first" são obcecados em usar o código para resolver um problema do usuário e veem o código apenas como um meio para um fim.
>
> 1. Engenheiros "Code-first" são obcecados com as abstrações, arquitetura, ferramentas e bibliotecas no código. Código elegante é o fim.

Não tenho certeza se concordo com o trecho *"Código elegante é o fim"*, mas o restante do texto faz bastante sentido para mim.

Devs *"product-first"* estão *[...] construindo, lançando e mantendo recursos que resolvem os problemas do usuário. Eles geralmente adoram estar na mesma sala que designers e gerentes de produto para aprender sobre os usuários e adoram encontrar oportunidades técnicas que possam melhorar o produto.*

[89] https://medium.com/@michlimlim/stop-just-using-frontend-or-backend-to-describe-the-engineering-you-like-e8c392956ada
[90] https://medium.com/@michlimlim
[91] https://www.youtube.com/watch?v=6x3UkAMjRiw

Por outro lado, devs *"code-first"*, ou *"de plataforma"* são aqueles que vão *construir plataformas de infraestrutura que suportam aplicativos, seja por meio da construção de pipelines de CI/CD, implementação de logs ou suporte a alto tráfego, arquitetura de código, etc.*

Mas e quanto a dicotomia *"backend"* x *"frontend"*? Onde se encaixa nesse contexto? O argumento é que dentro de cada eixo *"produto"* x *"plataforma"* você vai precisar entender tanto de backend quanto de frontend. Não digo que o ideal é sermos full-stack (eu não acredito em full-stack, como comentei em outro capítulo), mas sim que **devemos nos especializar no pensamento voltado para produto ou plataforma**. Eu me vejo muito mais no lado de plataforma, onde me especializei em backend. Mas eu participo de discussões que envolvem decisões de frontend, sempre pensando no tipo de cliente que plataforma atende (outros devs).

Eu li o post da Michelle poucos meses depois da sua publicação, em 2020. Desde então eu liderei e participei de times tanto de produto quanto de plataforma e tenho validado este conceito no meu dia a dia. É muito visível a insatisfação de uma pessoa quando ela está na área onde ela não se sente desafiada. Pessoas de produto podem se frustrar muito em um time de plataforma, e vice-versa.

Por isso, se você é liderança de algum time, recomendo fazer essa análise com seus liderados para entender se eles estão trabalhando nas tarefas que fazem mais sentido para eles. E se você for dev, se pergunte que tipo de projeto traria maior satisfação para o seu dia a dia. E lembre-se que muitas vezes satisfação aumenta muito a produtividade, enquanto que o inverso também é uma verdade.

Como organizo minha semana

Em Maio/2024 completei meu segundo ano trabalhando como Principal Software Engineer. Neste post[92] eu comentei sobre o que eu faço, inclusive como eu organizo minha agenda mas gostaria de detalhar um pouco mais este assunto.

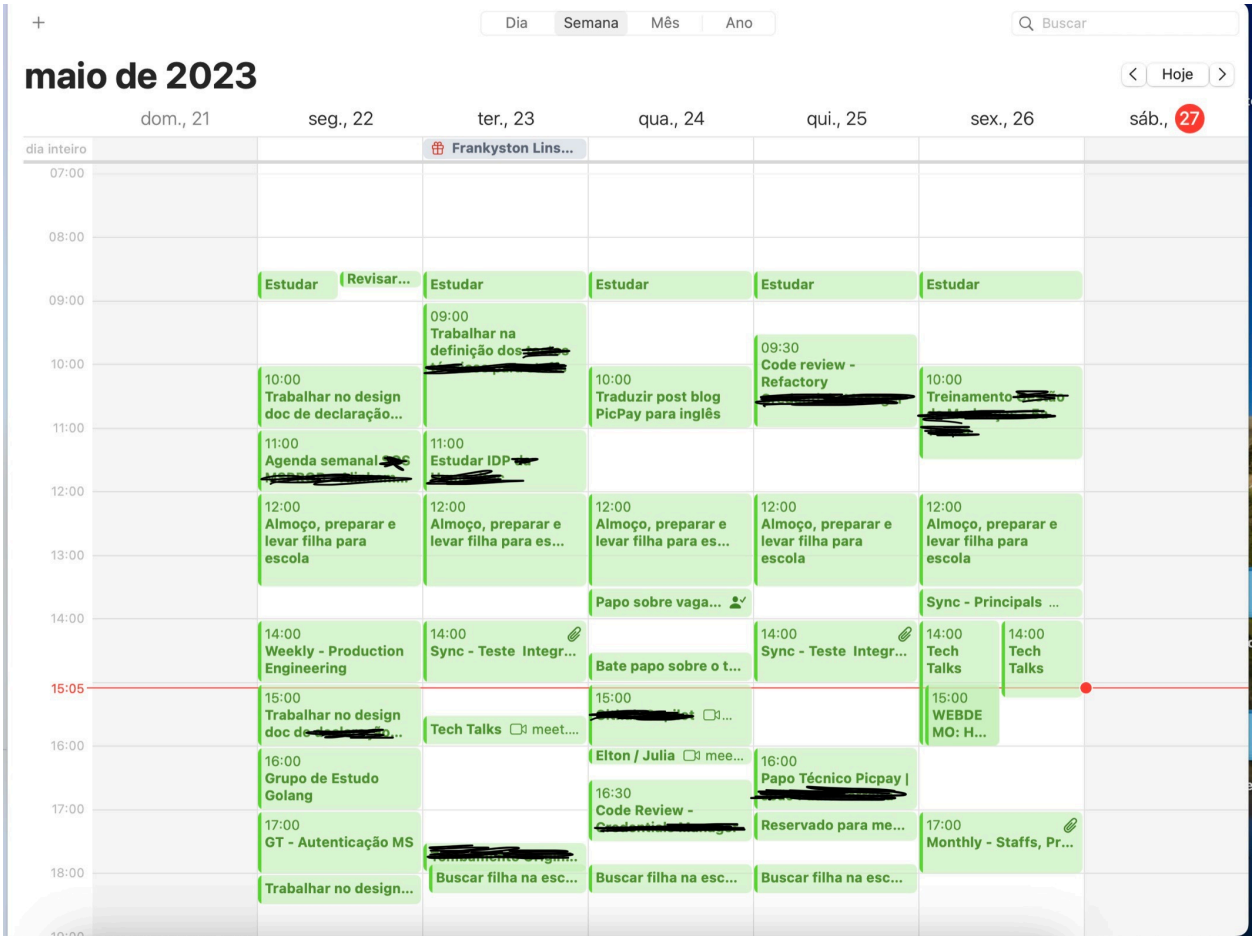
Posições "staff+", como Principal, tem algumas características interessantes:

- eu estou alocado em uma área, que é formada por alguns times. Desta forma, eu "navego" entre os contextos dos times e vejo como eles se integram nos objetivos da área e da empresa;
- tenho bastante autonomia para organizar as minhas prioridades e tarefas. Mas como dizia o sábio Tio Ben: "com grandes poderes vem grandes responsabilidades";
- sou envolvido em várias discussões técnicas durante a semana, bem como dou apoio em alguns incidentes que podem acontecer;
- tenho um papel importante de mentoria dentro dos times.

Diferentes papéis e contextos, somado com autonomia pode ser a receita para algumas coisas não muito saudáveis como sobrecarga, burnout e perda de foco no que é importante. Para evitar isso eu tenho feito algumas coisas:

- toda segunda-feira eu faço uma lista das prioridades da semana. Aponto quais são as tarefas, áreas e times que eu pretendo dar foco nos próximos dias. Compartilho essa lista com meus pares, outros Principal da área, e minha liderança. Eles dão feedbacks e apontam prioridades que eu possa ter deixado de fora, complementam com sugestões e ideias e se oferecem para ajudar em algo. Alguns colegas também compartilham a sua lista e eu faço a mesma revisão;
- ainda na segunda eu tenho uma reunião com as lideranças da área para alinharmos as prioridades de todos. Posso fazer algum ajuste na minha lista de prioridades caso necessário;
- a última tarefa da segunda é criar bloqueios na minha agenda da semana para dar foco em alguns itens da minha lista de prioridades. Assim eu consigo dar atenção ao que é importante e também comunico para meus colegas dos times os momentos que eu não posso participar de alguma reunião ou mentoria. Claro que urgências aparecem e preciso reorganizar a agenda algumas vezes, mas no geral funciona muito bem. Eu tomo cuidado de não bloquear minha agenda toda, nem mesmo bloquear um dia inteiro para uma tarefa. Prefiro dividir algo grande em blocos de uma ou duas horas, deixando espaços na agenda para que eu esteja disponível para os times me envolverem em discussões técnicas, revisão de código, pair programming e mentorias;

[92] https://maisquesenior.substack.com/p/elton-minetto

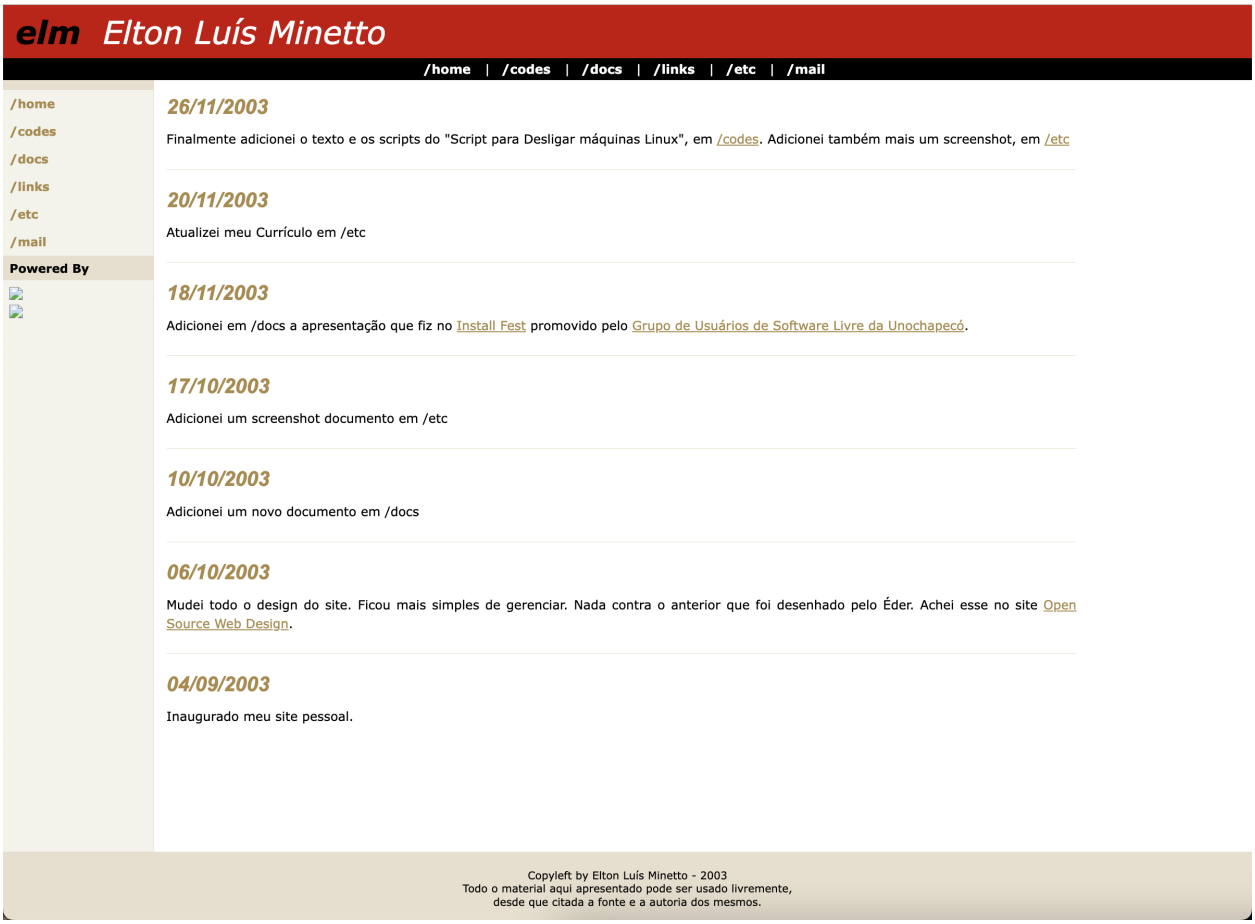


- na sexta-feira, a última tarefa da semana é atualizar o meu brag document. Com isso eu consigo anotar as minhas entregas e revisar o que eu tinha planejado x o que eu realizei. Isso tem me ajudado muito, conforme descrevi em um capítulo deste livro.

Compartilhei estas dicas na expectativa de serem úteis para outros profissionais, mesmo os que não atuam como Staff ou Principal.

Por que escrever?

Em Setembro de 2023 meu site pessoal[93] completou 20 anos! É realmente algo que eu não esperava quando escrevi meu primeiro "hello world"[94], em uma página HTML no diretório do meu usuário no servidor Linux da universidade:



Eu gostaria de falar sobre os motivos que me fazem continuar a escrever todos os meses no site (e em outros lugares como este livro).

É uma ótima forma de aprendizado

Dizem que uma forma de se aprender algo é ensinando o assunto para outra pessoa. Richard Feynman, pioneiro da eletrodinâmica quântica e ganhador do Nobel de 1965, criou uma técnica[95] que permitia que ele aprendesse virtualmente qualquer coisa. E um dos passos mais importantes é

[93] https://eltonminetto.dev
[94] https://eltonminetto.dev/2003/09/04/19/
[95] https://ead.pucpr.br/blog/tecnica-feynman

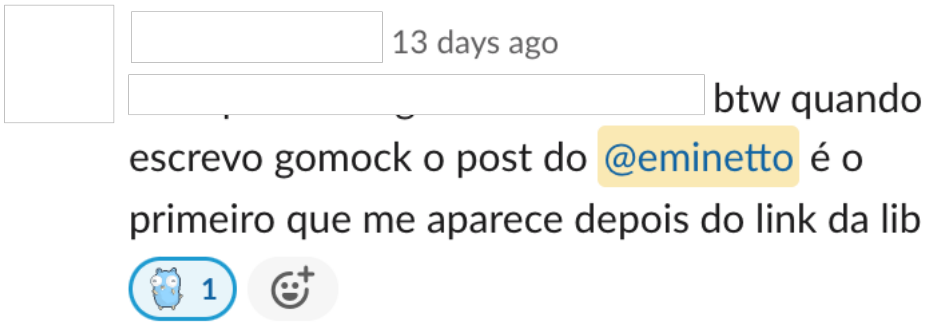
"Ensine esse conceito para um leigo". Ensinar na forma de um texto é algo ainda mais desafiador, pois você precisa pensar em uma série de fatores importantes como tom de voz, público alvo, profundidade, tamanho do texto, etc. Eu tenho usado os posts do meu site como forma de aprendizado e fortalecimento de conceitos durante toda minha carreira (eu comecei a trabalhar com programação em 1997 e a escrever o site em 2003) e posso testemunhar a favor desta ótima ferramenta.

Você pratica uma habilidade muito importante

Talvez uma das *skills* mais importantes para uma pessoa desenvolvedora é escrever bem. Isso sempre está entre as *soft skills* mais valorizadas nas empresas. Você vai precisar escrever descrições de *commits* e *pull requests*, documentação de projetos, relatórios para lideranças, sua própria análise de performance (algo bem comum em empresas de médio a grande porte e em startups). Sem falar no fato de que o ambiente remoto tornou-se quase que padrão nos últimos anos e passamos mais tempo no Slack/Teams do que nas nossas IDEs/terminais. Escrever bem mensagens de texto e documentos é crucial para um bom relacionamento com colegas e lideranças.

Você cria autoridade ao redor do seu nome

Conforme você aumenta a quantia de conteúdo gerado, bons efeitos começam a acontecer, como:



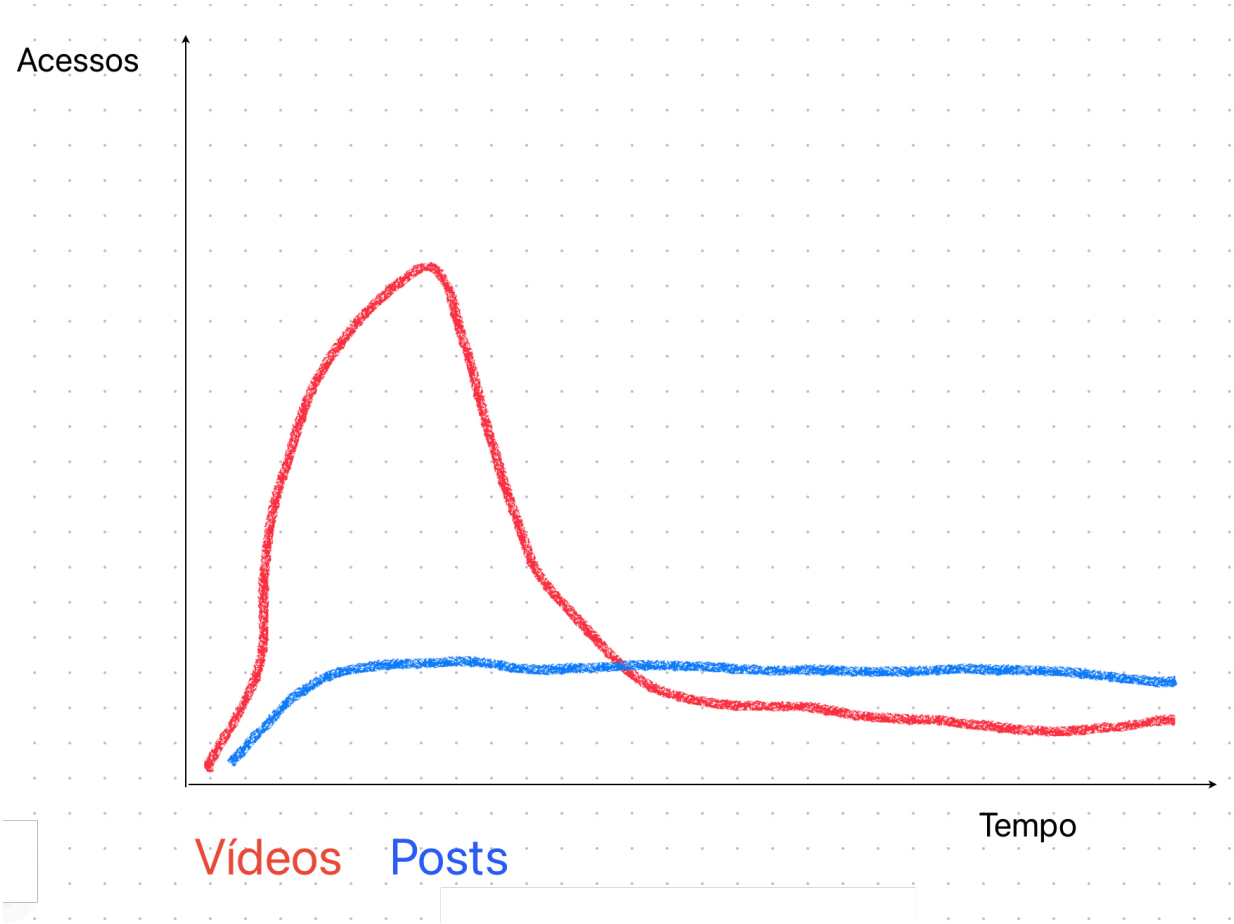
E:

Isso pode parecer um pouco de "ego trip", mas é importante para nossa carreira sermos conhecidos como referências em alguns assuntos. Isso nos ajuda a ter melhores ofertas de trabalho, ser chamado para eventos e palestras, tornam nosso currículo mais atrativo. E abrem oportunidades para conhecermos pessoas, contribuirmos com projetos e comunidades. É um retorno que vale muito o investimento.

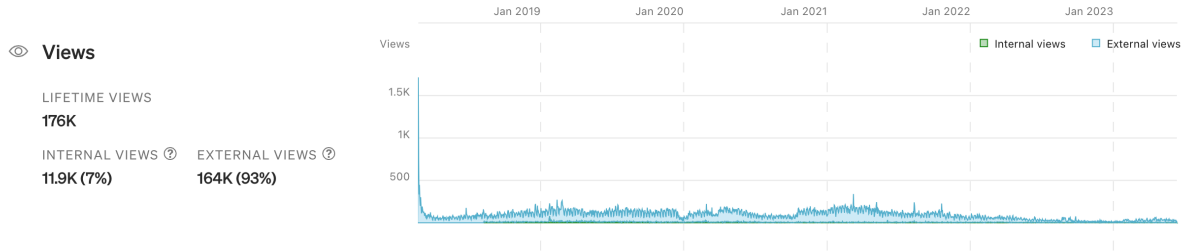
Claro que isso vale para qualquer tipo de conteúdo, não é exclusivo para textos. O que nos leva ao próximo ponto.

Seu conteúdo tem um alcance mais longo

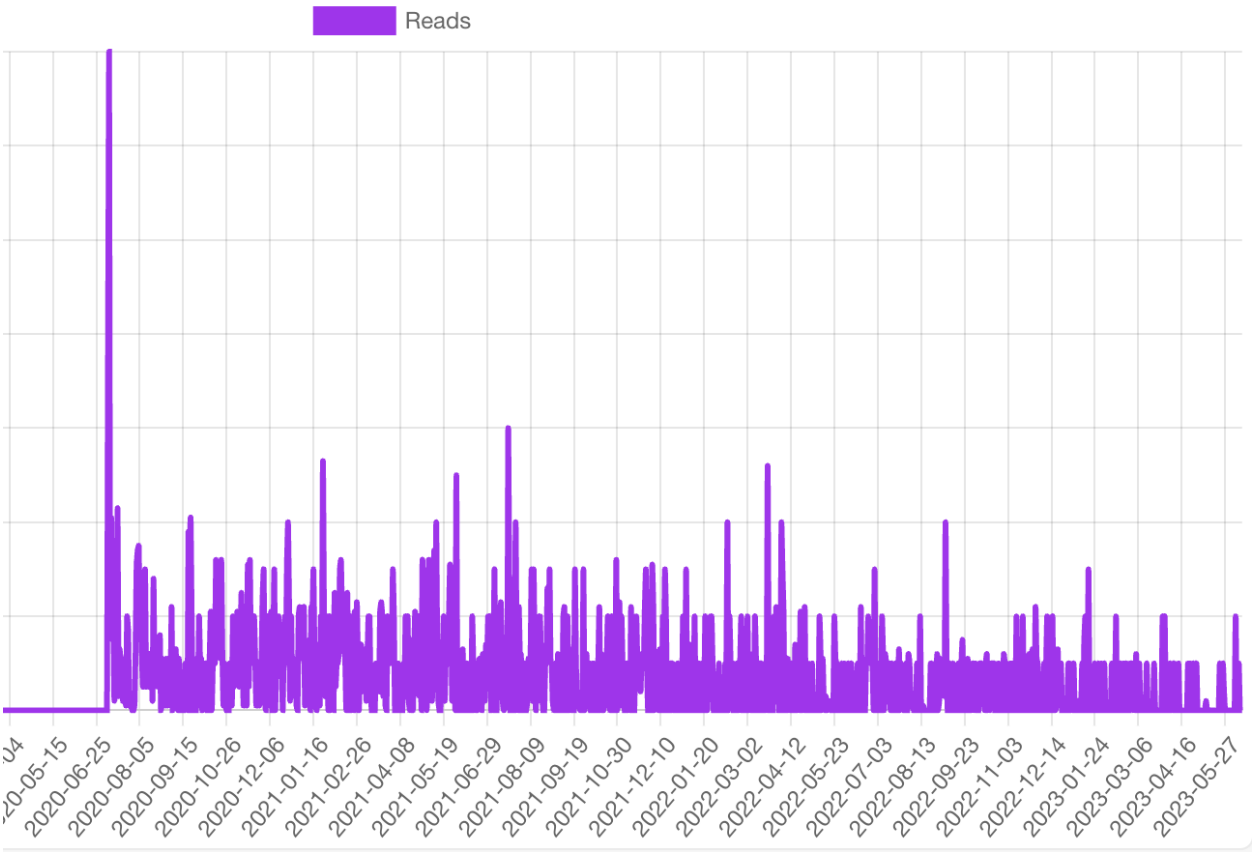
Eu sempre tive uma teoria, de que o conteúdo escrito tem uma "vida" mais longa do que videos. Algo como:



Dei uma olhada nos acessos dos meus posts para ver se isso se mostra perto do real:



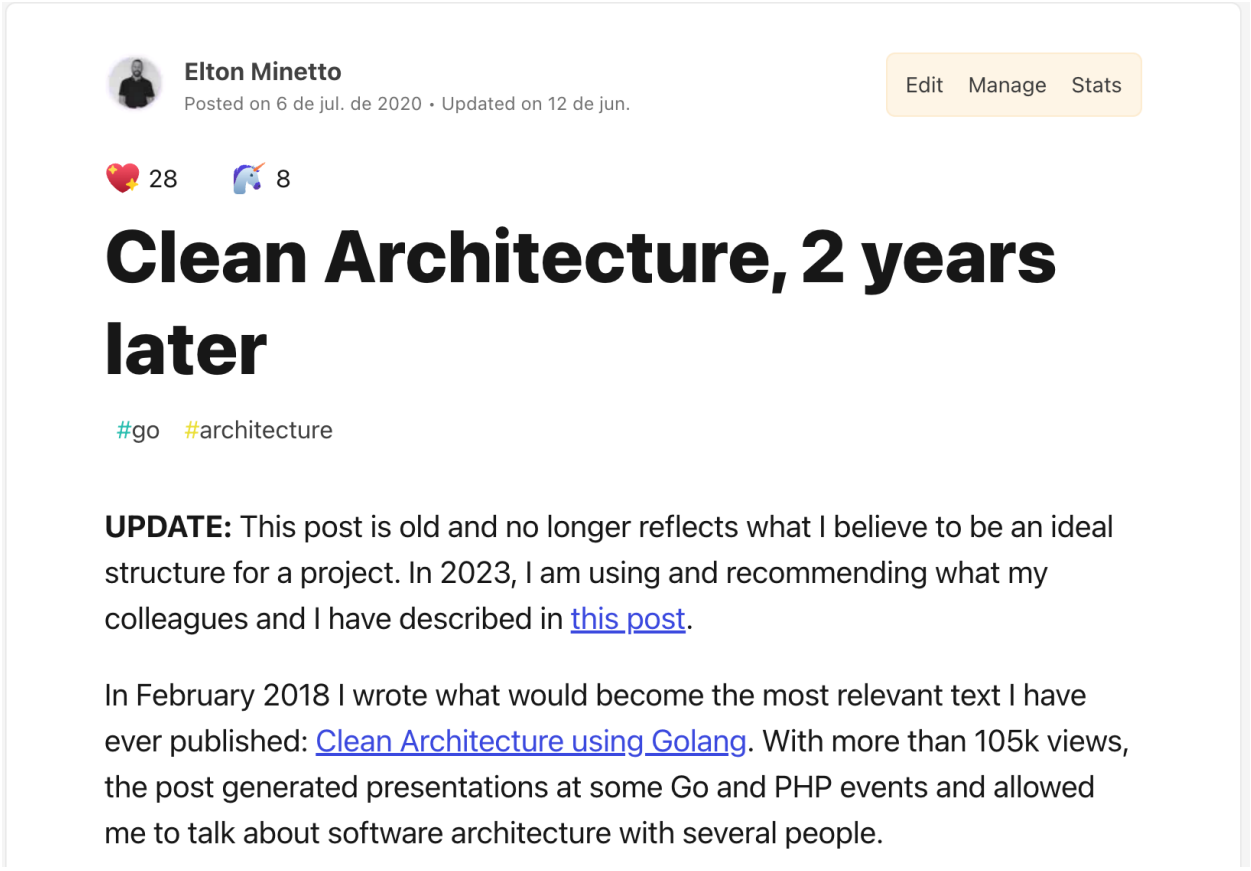
Outro post:



Como eu não tenho dados para comprovar se o comportamento dos acessos a conteúdos em vídeo bate com minha teoria eu usei a sabedoria da Internet. Perguntei em um tweet[96] e as respostas são bem interessantes, recomendo a leitura, inclusive para ver opiniões contrárias a minha.

Outra vantagem do texto versus o video é a facilidade para correções e atualizações de conteúdo. Por exemplo, em 2018 eu escrevi um post sobre Clean Architecture em Go[97]. Em 2020 eu fiz um novo post[98] revisando o que eu tinha aprendido em dois anos usando a arquitetura. E em 2023 eu escrevi um terceiro[99], descrevendo o que eu havia aprendido nos últimos três anos e como eu estou usando a tecnologia desde então. Em cada novo post foi muito fácil eu atualizar o anterior, guiando o leitor para a versão mais atual, sem deixar um assunto defasado que poderia causar problemas para quem está aprendendo:

[96] https://twitter.com/eminetto/status/1668244591987019776?s=61&t=sPotmpGH6d6KNNLETtizgA
[97] https://eminetto.medium.com/clean-architecture-using-golang-b63587aa5e3f
[98] https://eltonminetto.dev/en/post/2020-07-06-clean-architecture-2years-later/
[99] https://medium.com/inside-picpay/organizing-projects-and-defining-names-in-go-7f0eab45375d



Elton Minetto
Posted on 6 de jul. de 2020 • Updated on 12 de jun.

Edit Manage Stats

❤️ 28 🦄 8

Clean Architecture, 2 years later

#go #architecture

UPDATE: This post is old and no longer reflects what I believe to be an ideal structure for a project. In 2023, I am using and recommending what my colleagues and I have described in this post.

In February 2018 I wrote what would become the most relevant text I have ever published: Clean Architecture using Golang. With more than 105k views, the post generated presentations at some Go and PHP events and allowed me to talk about software architecture with several people.

Fazer isso em nos vídeos daria muito mais trabalho (pelo menos para mim, que não sou muito bom em ferramentas de edição).

Neste capítulo eu foquei nas vantagens para quem está produzindo o texto, mas também vejo algumas vantagens quando estou no papel de consumidor, como:

- posso ler no meu tempo, salvar para ler mais tarde, começar a ler no computador e terminar no celular;
- posso copiar e colar código para uma IDE ou terminal;
- posso guardar trechos para citação futura;
- posso pesquisar o conteúdo por palavras;
- etc.

Antes que me categorizem como velho retrógrado (pelo menos não por esse motivo) eu quero deixar claro que gosto de conteúdos em vídeo e até tenho alguns no meu YouTube[100]. Gosto principalmente para aprender alguns conceitos mais complexos ou mesmo para visualizar a demonstração de algum produto ou ferramenta. Produzir conteúdo, seja qual formato, é uma ótima forma de aprendizado, compartilhamento de conhecimento, etc. Se você prefere produzir e consumir videos, ótimo, isso vai

[100] https://www.youtube.com/eltonminetto

ser bom pra você. Meu objetivo com este texto era tentar incentivar a prática da escrita e mostrar suas vantagens.