# 65 dicas para ser um programador melhor

## Algumas dicas que podem ajudar muito a sua maneira de programar e pensar.

## Introdução

Fala pessoal, nos últimos dias eu não tenho passado bem de saúde e com isso não tenho ficado muito no computador, para não me entediar tanto, resolvi ler alguns livros e um deles é um livro incrível, que resolvi pegar as dicas dele e passar para vocês.

Esse livro eu li pela primeira vez quando estava trabalhando na HUGE, eles compraram o livro e eu peguei emprestado para ler. Eu achei tão bom que resolvi comprar um para mim =)

E qual é esse livro afinal hein? É o [The Pragmatic Programmer](https://www.amazon.com.br/Pragmatic-Programmer-Journeyman-Master/dp/020161622X), que se você ainda não conhecia, dá uma olhada com bastante carinho, acho que todo desenvolvedor deveria ler.

Junto com esse post, eu vou ouvindo o novo album da banda [Royal Blood](https://open.spotify.com/album/3Rz6kF8eGqrDOEteo5YsBj) chamado de [How Did We Get So Dark?](https://open.spotify.com/album/3Rz6kF8eGqrDOEteo5YsBj), são só 2 caras com uma bateria e um baixo que tocam um rock muito bom, outra coisa que vale a pena ser ouvida.

Bom, vamos lá, eu basicamente vou passar aqui as dicas adaptadas do livro para vocês. Leiam e releiam essas dicas com carinho, se puderem, imprimam em algum lugar para sempre ler ou salvem nos favoritos. Compartilhem com TODOS que puderem, pois um mundo onde os programadores seguem essas dicas, é um mundo melhor.

## Dicas

1. Se preocupe com o que constrói - Por que se gastar sua vida desenvolvendo software se você não se preocupa com ele ser bom?
2. Pense! Sobre o seu trabalho - Desligue o piloto automático e tome o controle. Constantemente critique e avalie o seu trabalho.
3. Dê opções, não crie desculpas - Ao invés de dar desculpas, dê opções. Não diga que não pode ser feito; explique como pode ser feito.
4. Não conviva com janelas quebradas - Corrija designs ruins, decisões erradas e código ruim quando vê-los.
5. Seja um catalisador de mudanças - Você não pode forçar as pessoas a mudarem. Mas você pode mostrá-las como o futuro pode ser e ajudá-las a participar na criação disso.
6. Veja o quadro geral - Não fique tão focado só nos detalhes que esqueça de verificar o que está acontecendo ao se redor.
7. Faça da Qualidade um requisito - Envolva os usuários para determinar a real qualidade que o projeto precisa ter e cumpra.
8. Invista regularmente no seu Portfolio de Conhecimento - Faça do aprender um hábito.
9. Analise criticamente o que você lê ou escuta - Não seja influenciado pelo hype das coisas. Analise a informação em termos de você e seu projeto.
10. É o que você diz e a maneira que você diz - Não faz sentido você ter grandes ideias se você não consegue comunicá-las efetivamente.
11. DRY - Don't Repeat Yourself (Não se repita) - Cada pedaço de código deve ser único e o mais genérico possível.
12. Faça ser fácil de reutilizar - Se é fácil de reutilizar, pessoas vão reutilizar. Crie um ambiente que suporte o reuso.
13. Elimine os efeitos colaterais entre coisas não relacionadas - Crie componentes que sejam auto-contidos, independentes e tenham um único e bem definido propósito.
14. Não existem decisões finais - Nenhuma decisão é escrita em pedra. Ao invés disso, pense que as decisões são escritas em areia de praia, e se planeje para ir sempre mudando e adaptando.
15. Use "tracer bullets" para achar o alvo - Faça as coisas e as deixe serem vistas, assim você pode ver o quanto elas chegam do que seu cliente realmente deseja.
16. Faça protótipos para aprender - Prototipar é uma experiência de aprendizado. O seu valor não está no código em que você produz, mas nas lições que aprendeu para montar.
17. Programe perto do problema - Crie e codifique na linguagem do seu usuário, como se você fosse ele.
18. Estime para evitar surpresas - Estime antes de começar. Você irá possivelmente ver problemas a frente e conseguirá se organizar melhor.
19. Itere o tempo com o código - Use a experiência que você ganha construindo a aplicação para refinar a escala de tempo do projeto e ter melhores estimativas.
20. Escreva e Rabisque - Textos e rabiscos nunca se tornam obsoletos. Eles ajudam a melhorar o seu trabalho e simplificar depurações e testes.
21. Use o poder da linha de comando - Use linha de comando quando interfaces não são necessárias.
22. Use um bom editor - O editor precisa ser uma extensão da sua mão, tenha certeza que seu editor é configurável, extensível e vocês trabalham bem juntos.
23. Sempre use versionamento de código - Versionamento de código são como máquinas do tempo para o seu trabalho, aqui é possível voltar atrás!
24. Corrija o problema. Não fique culpando - Não importa se o bug é sua culpa ou de outro, é um problema e precisa ser corrigido.
25. Não entre em pânico quando estiver debugando - Respire bem fundo e pense! sobre o que pode estar causando o bug.
26. O "select" não está quebrado - É muito raro achar um problema que seja de fato do sistema operacional ou do compilador. O bug na maioria das vezes está na sua aplicação.
27. Não presuma, prove! - Prove suas premissas num ambiente com dado real e sob as mesmas condições.
28. Aprenda manipulação de texto na sua linguagem - você passa boa parte do tempo trabalhando com textos. Por que não deixar o computador fazer um pouco desse trabalho para você?
29. Escreva código que faça código - Geradores de código aumentam sua produtividade e ajudam a evitar duplicações.
30. Você não pode escrever um software perfeito - Softwares não podem ser perfeitos. Proteja seu código e os usuários dos erros inevitáveis.
31. Faça o Design com contratos - Use contratos para documentar e verificar se o código faz o que se é pedido.
32. Quebre cedo - Um programa morto normalmente dá menos problemas que um programa deficiente.
33. Use assertions para previnir o impossível - use assertions para validar suas premisas. E use para proteger seu código de incertezas.
34. Use exceptions para problemas excepcionais - Exceptions podem prejudicar a leitura e manutenibilidade do projeto, criando os famosos código spaguetti. Reserve exeptions para coisas de fato excepcionais.
35. Termine o que você começou - Quando possível, a rotina ou objeto responsável por alocar aquele recurso, precisa ser responsável por desalocar também.
36. Minimize o acoplamento entre os módulos - Evite o acoplamento aplicando a [Lei de Demeter](https://en.wikipedia.org/wiki/Law_of_Demeter).
37. Configure, não integre - implemente escolhas de tecnologia para uma aplicação como se opções de configuração, não somente como processo de integração ou código agregado.
38. Coloque abstrações no código, detalhes no metadata - Codifique para o caso geral e coloque os casos específicos de forma separada do código base.
39. Analise o Workflow para melhorar a concorrência - busque formas de realizar código em concorrência no workflow de seu usuário.
40. Desenvolva usando serviços - Codifique em termos de serviços, independentes, bem definidos e com interfaces consistentes.
41. Separe as Views das Models - Para ganhar flexibilidade a baixo custo, desenvolva sua aplicação em termos de models e views.
42. Não programe por coincidência - Confie em coisas confiáveis. Tome cuidado com a complexidade acidental e não confunda uma feliz coincidência com o plano final.
43. Estime a ordem de seus algoritmos - Tenha uma ideia de o quão demoradas as coisas podem acontecer antes de escrever seu código.
44. Teste suas estimativas - Análises matemáticas de algoritmos não dizem tudo. Teste seu código sempre no ambiente mais próximo do real.
45. Refatore cedo, refatore frequentemente - Assim como você pode arrumar e reorganizar um jardim, reescreva, retrabalhe e rearquitete seu código sempre que necessário. Corrija a raiz dos problemas.
46. Codifique para testar - Comece pensando sobre como testar antes de escrever uma linha de código.
47. Teste seu software, ou seus usuários irão testar por você - Teste implacavelmente. Não deixe seus usuários acharem bugs para você.
48. Não use código mágico que você não entende - Mágicas podem gerar códigos complicados. Tenha certeza que você entende tudo que o código faz, antes de implementá-lo em seu projeto.
49. Não pegue requisitos - cave até eles - Os requisitos raramente estão só na superfície. Eles estão enterrados em camadas de premissas, equívocos e política. Veja além do que foi pedido.
50. Trabalhe com o usuário para pensar como o usuário - Teste com o usuário, é a única maneira de saber como ele de fato será usado. Pegue todos os insights possíveis assim.
51. Abstrações vivem mais que detalhes - Invista em abstrações, não na implementação. Abstrações podem sobreviver as mudanças de diferentes implementações e tecnologias.
52. Cria um Glossário do projeto - Crie e mantenha um lugar com todos os termos e vocabulários do projeto, pessoas novas podem não saber certas definições do projeto.
53. Não pense fora da caixa, ache a caixa! - Ao enfrentar com problemas impossíveis. Pergunte a si mesmo: "Isso precisa ser feito dessa maneira? Isso realmente precisa existir?".
54. Comece quando você estiver pronto - Você tem adquirido experiência com o tempo, não ignore as dúvidas que tiver.
55. Algumas coisas são melhor quando feitas do que descritas - Não caia numa espiral de criar especificações e pensamentos, uma hora você precisa começar a codificar.
56. Não seja um escravo de métodos formais - Não adote cegamente uma técnica sem colocar no contexto do seu desenvolvimento e suas capacidades.
57. Ferramentas caras não produzem designs melhores - Tome cuidado com o hype das ferramentas, dogmas da indústria e custos. Julgue suas ferramentas pelos seus méritos.
58. Organize equipes por funcionalidades - Não separe designers de desenvolvedores, monte equipes da mesma forma que você cria o código.
59. Teste cedo, teste frequentemente, teste automaticamente - Testes que rodam a cada build são muito mais efetivos que testes esporádicos.
60. O código não está pronto até que os testes rodem - só isso.
61. Use sabotadores para testar seu teste - Introduza bugs de propósito em partes separadas do código somente para verificar se os seus testes vão pegar.
62. Teste o estado de cobertura, não a cobertura do teste - Identifique e teste os diferentes estados do código. Testar só linhas de código não é o suficiente.
63. Ache bugs uma única vez - Uma vez que um humano encontrou um bug, essa deverá ser a última vez que ele irá achá-lo. Crie testes para sempre verificar o bug.
64. Inglês é uma outra linguagem de programação - Escreva documentos como você escreve código, honre o princípio do DRY, use metadata e geração automática.
65. Assine o seu trabalho - os grandes construtores ficam orgulhosos de assinar suas obras, você também deveria se orgulhar.

## Conclusão

Espero que depois dessa leitura, você tenha identificado várias das suas falhas e também vários dos seus acertos e se organize para seguir sempre assim. E lembre-se o aprendizado é algo contínuo e importante, então sempre dê uma lida nessas dicas e pensa se está utilizando tudo que está ali ou se está se sabotando em algum momento.

É isso aí galera, deixem aí nos comentários o que acharam dessas dicas, sejam críticos também! Se tem algo que não concordam ou que querem falar um pouquinho mais, a área de comentários é o lugar para isso. E compartilhem no trabalho de vocês!

https://willianjusten.com.br/65-dicas-para-ser-um-programador-melhor/