Os caminhos da Engenharia de Software: o que estudar

Como me tornar um(a) engenheiro(a) de software. O que preciso estudar. Tudo isso e mais um pouco nesse artigo sobre os caminhos da engenharia de software.

8/Mai/2018 — 18 minutos de leitura

Editar este artigo

Hoje em dia a maioria dos cargos em Startups e empresas legais é de Software Engineer ou Engenheiro(a) de Software e por isso nos questionamos: Qual o conhecimento necessário para se tornar um(a) engenheiro(a) de software?

Senta que lá vem história...

Como desenvolvedores(as), mais cedo ou mais tarde em nossa carreira, nós estudamos Ciência da Computação e Engenharia de Software, porém quando nós buscamos conhecimento para arrumar o primeiro emprego na área de desenvolvimento procuramos nos buscadores coisas como:

- como me tornar back end
- como me tornar front end
- como me tornar programador(a)
- como me tornar desenvolvedor(a) Android
- como me tornar desenvolvedor(a) iOS

Isso por que buscamos coisas específicas. Carreiras **específicas**. E é extremamente normal, não tem problema algum uma pesquisa dessas. Não estou dizendo que você está errado(a). Se está procurando o primeiro

emprego como desenvolvedor(a), precisa procurar exatamente isso que falei acima.

Em buscas desse tipo normalmente encontramos artigos como:

- Como se tornar um desenvolvedor full-stack no próximo ano
- Como Se Tornar um Programador
- Como me tornar um Desenvolvedor Front End
- Um ano (ou um pouco mais) como desenvolvedor Front End
- Profissão, Desenvolvedor Android
- Quanto custa ser um desenvolvedor para iPhone?
- Os 5 principais requisitos para ser um bom programador
- A carreira de programador: qual curso fazer primeiro?
- Como se tornar um desenvolvedor de jogos?
- Como se tornar um desenvolvedor iOS certificado pela Apple

Essas foram as pesquisas que eu realizei no Google:

- Como me tornar Back End
- Como me tornar front end

- Como me tornar programador
- Como me tornar desenvolvedor android
- Como me tornar desenvolvedor ios

A maior quantidade de resultados veio sobre como se tornar desenvolvedor(a) front-end, mesmo eu estando em aba anônima, mas pra você podem aparecer resultados bem diferentes.

Eu também já escrevi artigos sobre
requisitos de vagas para desenvolvedor(a) front end , sobre
como são entrevistas de emprego para devs JavaScript , sobre como foi meu primeiro ano nessa profissão e vários outros
recursos sobre como se tornar front-end .

Só que nós pecamos em uma coisa quando escrevemos artigos sobre como se tornar desenvolvedor(a) xxx, nos esquecemos de dizer que existe coisa mais importante do que aprender uma linguagem de programação específica, mais importante do que você aprender versionamento de código, do que ter uma conta no GitHub com todos os quadradinhos verdes e por ai vai.

Uma linguagem, tecnologias específicas, Git e GitHub, são coisas que **te empregam**, mas não são as coisas que vão te tornar **um(a) melhor desenvolvedor(a) de software** ou alguém que cria as melhores soluções para os problemas relacionados a software.

Não estou dizendo que essas coisas **não são importantes**. Pelo contrário! Iniciantes em desenvolvimento, precisam aprender essas coisas o mais rápido

Porém elas não são as únicas coisas que você precisa aprender para se tornar um(a) desenvolvedor(a) completo(a). O buraco vai bem mais pra baixo e é aí

que entra o conhecimento em Ciências da Computação e Engenharia de Software.

Eu vivo dizendo que faculdade não importa, mas ela só não importa se você tiver a pre-disposição a estudar algumas matérias de faculdade por conta própria. Se você não estudar os assuntos que eu vou falar nesse artigo, provavelmente uma pessoa formada pode sair na sua frente em uma entrevista de emprego e até no dia-a-dia.

Não quero entrar no assunto faculdade nesse artigo, afinal sou defensor do autodidatismo, porém é uma verdade que a pessoa que fez faculdade foi **forçada a aprender os conceitos e princípios importantes de desenvolvimento de software e computação** que muitas vezes deixamos passar.

Nesse artigo eu vou deixar uma lista de assuntos, matérias, tópicos importantes para um(a) Engenheiro(a) de Software e links para estudo.

Eu fiz essa lista a algum tempo e busco estudar o que está nela para me tornar um melhor desenvolvedor. Se eu deixei faltar alguma coisa, por favor contribua com esse artigo comentando logo abaixo, você vai ajudar outros leitores e o autor que te escreve agora. :grin:

deixam passar

A ideia para esse artigo veio de uma conversa no Slack do Training Center , uma comunidade para iniciantes em programação, sobre o que aprender

quando parece que já dominamos o que usamos no trabalho como linguagens de programação, bancos de dados relacionais e não relacionais, http, frameworks, metodologias de trabalho, etc.

Então veio a tona o que sempre deixamos passar: **os conhecimentos em Ciência da Computação e Engenharia de Software**. Estamos tão imersos em qual linguagem escolher e qual framework utilizar para resolver um problema que esquecemos que nosso trabalho vai muito além disso.

Nós estamos aqui para resolver problemas através de softwares, mas será que estamos escrevendo software da melhor maneira possível?

Uma rápida pesquisa no Google sobre engenharia de software mostra que estamos deixando passar algumas coisas importantes nos nossos artigos mais específicos, como você deve descobrir olhando neste artigo da Wikipedia: Software Engineering . Da uma olhada no link e pare para pensar honestamente se você já domina tudo o que está escrito nele.

Um bom profissional em desenvolvimento de software deve dominar os seguintes tópicos:

- Arquitetura de Computadores
- Sistemas Operacionais

- Linguagens de Programação
- Paradigmas de Programação
- Estruturas de Dados
- Matemática Computacional
- Padrões de Projetos
- Testes de Software
- Qualidade de Software
- Manutenção de Software
- Gerenciamento de Software
- Processos de desenvolvimento de software

E muito mais! — Olha que eu nem citei **soft-skills**.

Vamos analisar cada tópico.

Arquitetura de Computadores

Esse assunto é muito importante para que você entenda o coração da tecnologia que você usa diariamente: o computador.

Estudando isso você passa a entender como funciona a memória do computador, o processador, barramentos, clock, cache, threads, um pouco (ou muito, dependendo de que fonte de estudos você utilizar) sobre paralelismo e concorrência, portas lógicas, dentre outros tópicos importantes para compreender como o computador funciona e até a história por trás das

mudanças que tivemos nos processadores e memórias nos últimos tempos.

É nessa matéria que ouvimos falar sobre a Lei de Moore, por exemplo:

> *Até meados de 1965 não havia nenhuma previsão real sobre o futuro do hardware, quando Gordon E. Moore fez sua profecia, na qual o número de transistores dos chips teria um aumento de 100%, pelo mesmo custo, a cada período de 18 meses. Essa profecia tornou-se realidade e acabou ganhando o nome de Lei de Moore. Wikipedia*

Alguns recursos para estudarmos Arquitetura de Computadores:

- Computer Architecture , por Universidade de Princeton no Coursera
- Os vídeos da matéria de Organização de Computadores da UNIVESP .

Ouça meu podcast

Livros:

- Organização Estruturada de Computadores, por Andrew S. Tanenbaum
- Arquitetura e Organização de Computadores, por William Stallings

Sistemas Operacionais

Além de entender a parte de Hardware de como funciona o computador, um(a) engenheiro(a) deve entender os sistemas operacionais que os gerenciam.

Esse assunto é muito abordado em qualquer faculdade e normalmente abrange tópicos como arquitetura de sistemas operacionais, também aborda concorrência e paralelismo na camada do sistema, gerencia de memória e de processador, escalonamento de processos, memória virtual, classificação (tipos) dos sistemas operacionais e muito mais.

Você já parou pra pensar como são organizados os processos de cada software aberto em um sistema operacional, como o S.O. gerencia a fila de tarefas a serem executadas ou as janelas abertas na tela?

Onde podemos aprender mais sobre isso:

- Sistemas Operacionais Modernos, por Andrew S. Tanenbaum

Linguagens de Programação

Isso mesmo “linguagens”. Como desenvolvedores nós escolhemos uma linguagem de programação principal e focamos nela. Porém uma linguagem pode não possuir todos os benefícios para resolver um problema x em seu sistema e é nessa hora que o conhecimento em outras linguagens se faz importante.

Isso não significa que você precisa dominar 100% de todas as linguagens de programação existentes, mas seria muito útil que você aprende-se uma outra linguagem com outros paradigmas e formas diferentes de resolver os problemas que você enfrenta todos os dias com a linguagem que escolheu trabalhar.

Estudar outra linguagem de programação pode significar uma nova maneira de pensar sobre os problemas que você já resolveu ou mesmo que ainda vai resolver.

Mas aprender uma nova linguagem nem é o ponto principal desse tópico, no assunto linguagens de programação você deve aprender **como as linguagens funcionam**, não somente a declarar variáveis, funções e métodos e utilizá-las. Levando para o meu contexto de trabalho, um(a) bom(boa) desenvolvedor(a)

JavaScript deveria entender como funciona o "JIT", por exemplo, para saber como seu código roda dentro do navegador ou pelo menos saber que isso existe.

Nesse tópico também entram alguns conceitos de paradigmas de programação, classificação de linguagens de programação, tipos, abstração de dados e muito mais.

Para saber mais sobre linguagens de programação:

- Programming Languages, Wikipedia

Livros:

- Conceitos de Linguagens de Programação, por Robert W. Sebesta

Paradigmas de Programação

Aposto que você tem ouvido muito, ultimamente, sobre Programação Funcional. Quando estamos aprendendo programação também ouvimos que precisamos saber Orientação a Objetos.

Se você está lendo esse artigo eu acredito que você até saiba o que são paradigmas de programação, mas você já parou para estudar outros paradigmas além do que você trabalha diariamente?

Nessa matéria aprendemos as diferentes maneiras de estruturarmos nossos

orientação a objetos e pronto. Acabou. Mas não é bem assim.

Raramente vemos faculdades ensinando o paradigma funcional, por exemplo, mas se você depender somente de um curso para aprender algo e não buscar conhecimento por conta própria estará em risco de permanecer em uma zona de conforto e possivelmente a um futuro sofrimento por conta de mudanças no mercado de trabalho.

Assim como aprender outras linguagens de programação te ajuda a ter um canivete suíço para resolver os problemas das mais variadas maneiras, conhecer outros paradigmas também vai te ajudar a pensar diferente e também a estar pronto para uma mudança de mercado ou de ferramentaria no desenvolvimento de software.

Alguns paradigmas importantes a conhecermos, mas não estrito somente a eles: imperativo, orientação a objetos, orientação a eventos e funcional.

Para não ficar somente em minhas palavras, da uma olhada no que o Eduardo Cesar Borsato comentou em seu artigo “Paradigmas de programação são importantes?”

> *Programação é resolução de problemas. Procure conhecer mais sobre paradigmas de programação; se trabalha com uma linguagem que suporta apenas um paradigma, estude-o a fundo e aplique esse conhecimento em sua linguagem. Mas procure conhecer outros, afinal, não é*

> *bom se fechar apenas em uma linguagem ou paradigma. [...] iMasters*

Estruturas de Dados

A matéria que costuma tirar as pessoas dos cursos em faculdades de Sistemas da Informação, Ciência da Computação, Analise e Desenvolvimento de Sistemas, Engenharia da Computação e outras relacionadas a programação.

Essa matéria se torna uma arma de eliminação de alunos nas mãos do professor certo.

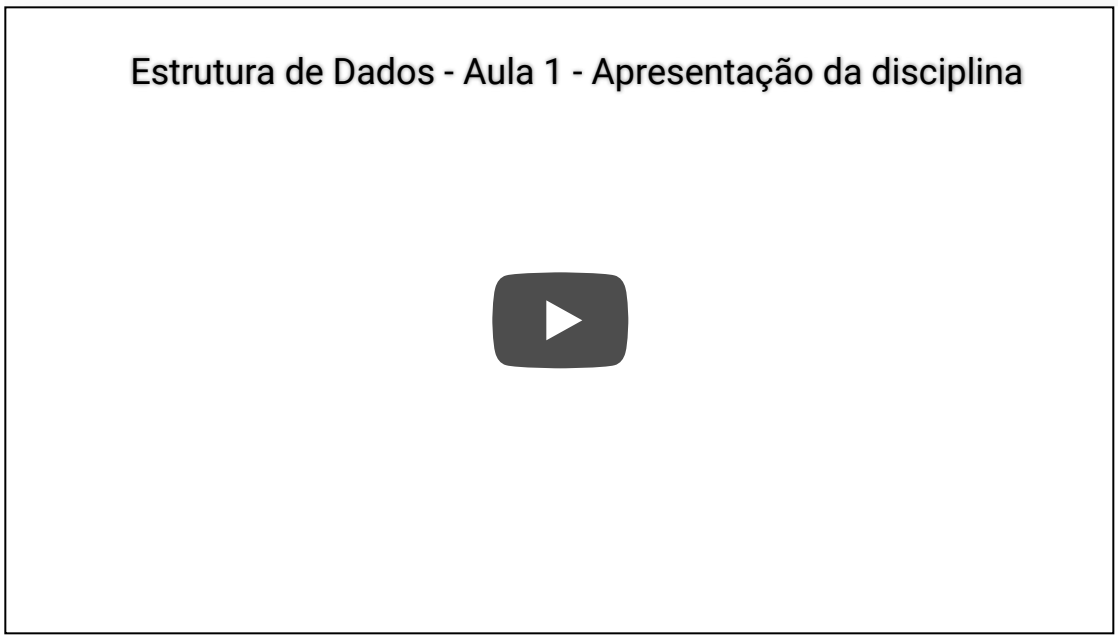
Normalmente ensinada usando C ou Java, é um assunto muito importante para que você entenda como trabalhar com dados em seu sistema. Depois que eu estudei sobre isso minha mente abriu muito sobre a utilização de arrays e objetos associativos.

Depois de estudar sobre estruturas de dados você nunca mais vai ver o .shift, .unshift, .push, .pop e .concat da mesma maneira em JavaScript e outras linguagens que implementam esses métodos para manipulação de arrays.

Eu não concordo com o uso da linguagem C para ensinar estrutura de dados, afinal a pessoa fica tão perdida no ambiente de desenvolvimento e na linguagem de programação que acaba só pensando em desistir do curso ou deixa de aprender pra ficar corrigindo erro de compilação por causa de ponteiros. — Mas isso também fica para outro artigo reclamando de faculdades.

Ouça meu podcast

Um excelente material para aprender estruturas de dados é a playlist também da UNIVESP:



O professor faz ótimas comparações entre C e Java enquanto ensina sobre o assunto.

Livros:

- Estrutura de dados, Algoritmos, Análise da complexidade e implementações em Java e C/C++, por Ana Fernanda Gomes Ascencio e Graziela Santos Araújo

Aqui entramos em um assunto que muitos podem reclamar. Algumas pessoas, assim como eu já fiz, podem alegar que aprendemos matemática e não utilizamos no dia a dia escrevendo software.—sóquenão

O conhecimento em Matemática Computacional nos ajuda a entender alguns erros de conversão numérica nas linguagens de programação (o conhecimento em Arquitetura de Computadores também ajuda a entender isso quando estudamos sobre como o computador faz cálculos), qual o limite de cálculo de um computador.

A maioria dos cursos que ensinam essa matemática nem é nada tão avançado. Se você aprende bem sobre a teoria de conjuntos, por exemplo, fica mais fácil entender os comandos SQL.

É nessa matéria que aprendemos os conceitos mais legais de árvores binárias, grafos e máquinas de estado. Talvez você se assuste, como eu me assustei, ao ouvir falar desses conceitos, mas não se preocupe por que existem diversos recursos bons sobre isso na internet.

Algumas playlists que podem nos ajudar a aprender isso:

Ouça meu podcast

Matemática - Aula 1 - Lógica e argumentação na linguage...

Tem uns vídeos repetidos no meio, mas em geral é um conteúdo muito legal.

Livros:

Ouça meu podcast

Padrões de Projetos

É claro que, como bons desenvolvedores que somos, aprendemos Padrões de Projetos para aplicar ao dia a dia, né?

Então... Tem sempre alguém que diz que padrões de projetos não são importantes, que preferem inventar suas soluções do zero (ou como conhecido “na unha”).

Aprender padrões de verdade é muito importante, pois, com certeza, nós sempre temos problemas repetidos em desenvolvimento de software que já foram solucionados pelos padrões a anos atrás e são usados até hoje.

Só que ai vem alguém super dotado(a) de um conhecimento descomunal e diz que não se interessa em aprender padrões de projetos por que suas soluções são melhores do que as inventadas e estudadas a anos. OK.

Nesse link da Wikipedia temos um belo resumo sobre o que abrange os conceitos de padrões de projetos no desenvolvimento de software.

Recursos para aprender sobre Padrões de Projetos:

- Padrões de Projeto
- O que são Padrões de Projeto GoF (Gang Of Four)?

Livros:

- Refatorando com padrões de projeto, por Marcos Brizeno
- Padrões de Projetos, por Erich Gamma

Testes de Software

Além do Paradigma Funcional, outra coisa que está sendo muito cobrada hoje em dia (acho que sempre foi) é o tema Testes de Software.

Testes de software não são coisa nova, isso existe e é usado a muito tempo e na faculdade as pessoas aprendem coisas como TDD e BDD, mas tem muita coisa nesse tópico que pode e deve ser estudado.

Testes de software faz parte da matéria de Processos de desenvolvimento de software e envolve algo crítico em sistemas que é a **integridade do sistema**. Os testes automatizados num processo de integração continua , por exemplo, garantem uma segurança (bugs e erros) no código de uma aplicação e facilitam o deploy.

Os testes são uma fase muito importante para manter a Qualidade de Software e é uma matéria também abordada em Processos de desenvolvimento de software, mas que tem seu ramo separado cheio de coisas importantes a se estudar.

- Entendendo Testes de Software
- Teste de software: Introdução, conceitos básicos e tipos de testes
- JS com TDD na Prática
- Software Testing
- Software Testing, Udacity

Qualidade de Software

Qualidade de software não é só falar que escrevemos código bonitinho, indentado, com bons nomes de variáveis ou métodos.

Quando eu comecei a ler o livro Clean Code tomei um verdadeiro *tapa na cara*, pois achava que o que eu fazia era Qualidade de Software.

Da uma olhada na descrição de Qualidade de Software pela Wikipedia:

> *[...] as características desejadas de produtos de software, a extensão em que um produto de software em particular possui essas características e aos processos, ferramentas e técnicas que são usadas para garantir essas características*

Nessa matéria também entra o custo do software, métricas de Gerenciamento de Software e os Testes de Software. Qualidade é algo muito abrangente e um

assunto muito extenso e muitas vezes acreditamos saber sobre o assunto, mas estamos falhando nisso.

Recursos para estudar esse assunto:

- Qualidade de software? O que é e porque você precisa ter!
- Qualidade, Qualidade de Software e Garantia da Qualidade de Software são as mesmas coisas?

Livros:

- Garantia de Qualidade de Software, por Alexandre Bartié

Alguns outros livros de Engenharia de Software também tocam nesse assunto e vou deixar a lista mais abaixo.

Manutenção de Software

O que mais fazemos em um software em produção são correções, adição de novas features e evolução da plataforma como um todo como renovação das tecnologias utilizadas, por exemplo.

São poucos momentos em nossa carreira no mercado de trabalho em que realmente criamos algo do zero. Normalmente somos contratados para dar manutenção em sistemas já existentes, que estão rendendo dinheiro e poucas vezes somos alocados ou contratados para criar algo desde o princípio.

Mas existem estratégias para que as mudanças não afetem o ciclo de vida de um software ou não afetem um negócio (empresa) e é importante

conhecermos essas estratégias.

Para manter uma estratégia de manutenção de software vai ser bom conhecer bem sobre Gerenciamento de Projetos.

Recursos para entender sobre Manutenção de Software:

- Software maintenance
- Manutenção De Software

Gerenciamento de Software

Não é a mesma coisa que Gerencia de Projetos, mas a Gerencia de Projetos também é importante para a Gerencia de Software.

Algumas faculdades chamam de Gerência de Configuração de Software.

A melhor explicação sobre o assunto que eu encontrei foi esse artigo: O que é Gerência de Configuração de Software?, no site da Pronus.

> *Gerência de Configuração de Software (GCS) é um conjunto de atividades de apoio que permite a absorção ordenada das mudanças inerentes ao desenvolvimento de software, mantendo a integridade e a estabilidade durante a evolução do projeto. Pronus*

Pela Wikipédia encontramos a seguinte definição:

> *Gerência de configuração de software, gerência de configuração ou ainda gestão de configuração de software é uma área da engenharia de software responsável por fornecer o apoio para o desenvolvimento de software. Suas principais atribuições são o controle de versão, o controle de mudança e a auditoria das configurações. Wikipedia*

Muitas melhorias no software são encontradas através dos conhecimentos que adquirimos estudando essa matéria.

Processos de desenvolvimento de software

Finalmente acabou a lista!

Essa é a matéria que aborda quase tudo o que falei até agora e um pouco mais.

Em Processos de desenvolvimento de Software aprendemos Analise de Custo, Analise de Requisitos, Arquitetura de Software, Testes, Qualidade, Manutenção e assim vai.

software na matéria de Analise de Sistemas.

Recursos para aprender sobre o assunto:

- Software Development Process, no Udacity

Conclusão

Me desculpe se esse texto ficou muito longo, mas são assuntos extremamente importantes e que precisamos estudar em nossa carreira como desenvolvedores(as) tenhamos o título de Engenheiros(as) ou não.

Os livros e materiais indicados nesse artigo vieram de dicas que coletei com alguns amigos muito antes de escrever esse texto e eu ainda não consumi todo o conteúdo. Se você souber de mais algum material legal, pode nos indicar (pra mim e para outros leitores) nos comentários.

Alguns outros recursos para estudar Engenharia de Software:

- Teach Yourself Computer Science
- Jornada à teoria da informação
- Algoritmos
- Introdução à Ciência da Computação
- Intro to Theoretical Computer Science
- CS 50
- CC 50

O Pedro Fernandes deixou uma dica muito boa pra gente aqui:

- Coding Interview University

O Danilo Silva também

- OSSU Computer Science

Dica do Clayton Santos da Silva :

- Playlist sobre sistemas distribuídos

Mais livros:

- Introdução à Arquitetura de Design de Software
- Entrega Contínua. Como Entregar Software de Forma Rápida e Confiável
- Lançamento de Software
- Padrões de Implementação
- Implementando o Desenvolvimento Lean de Software. Do Conceito ao Dinheiro
- Custo de Software

E alguns artigos sobre como se tornar um(a) engenheiro(a) de software:

- How to Become a Software Engineer

- Software Engineer

Não me recordo as pessoas que me indicaram esses conteúdos para estudar, pois vem de diversas perguntas que eu já fiz nas comunidades no começo da carreira, porém deixo meu agradecimento ao grande William Oliveira (????) e a todos que indicam esse tipo de conteúdo pelas comunidades afora.

Este conteúdo te ajudou?
Se eu consegui te ajudar, considere contribuir com o meu trabalho através dos links abaixo.
Qualquer valor é muito bem vindo e os apoios começam a partir de 1 real. Apoiar via Apoia.se Apoiar via PicPay

Apoiar via PayPal

Espalhe a palavra!

Compartilhe este artigo nas redes sociais clicando nos ícones.

Tags: carreira programação computação

Ouça meu podcast

William Oliveira

Inclusão social através do ensino de programação e front-end

Desenvolvido com Eleventy e Hylia Eleventy Starter Kit v0.7.0.