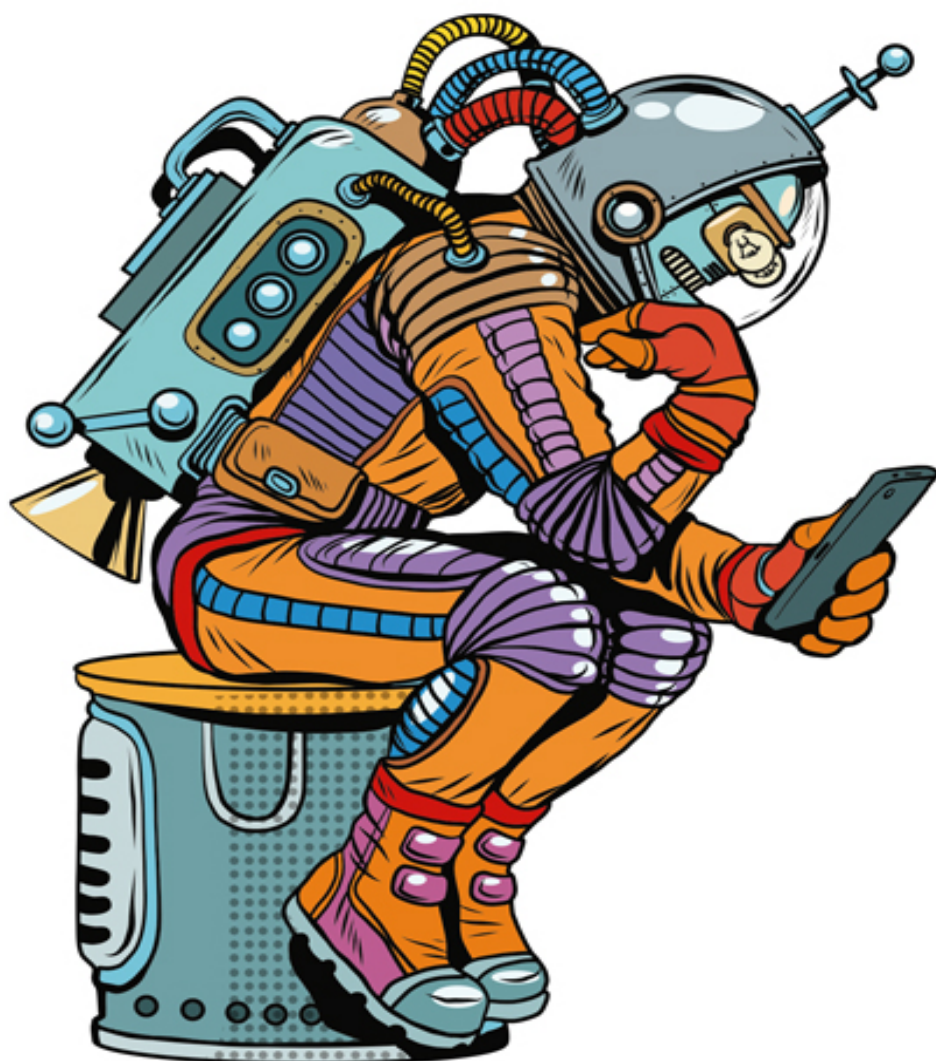


# O universo da programação

Um guia de carreira em desenvolvimento de software



Casa do  
Código

WILLIAM OLIVEIRA

# Sumário

- ISBN
- Dedicatória
- Agradecimentos
- Sobre o autor
- Prefácio
- Introdução
- 1. Uma introdução ao vasto universo da programação
- 2. Por que se envolver com programação
- 3. Pessoas que escrevem programas e lançam foguetes
- 4. Escolhendo um caminho
- 5. Uma galáxia de questões mal resolvidas
- 6. Habilidades de sobrevivência no universo da programação
- 7. No universo afora as pessoas precisam da nossa ajuda
- 8. Conquistando o primeiro emprego com programação
- 9. Aonde ir depois daqui
- 10. Considerações finais

# ISBN

Impresso e PDF: 978-85-94188-90-8

EPUB: 978-85-94188-91-5

MOBI: 978-85-94188-92-2

Caso você deseje submeter alguma errata ou sugestão, acesse  
<http://erratas.casadocodigo.com.br>.

# Dedicatória

Esta obra é dedicada à minha mãe, Almerinda, e à minha companheira, Juliana.

Minha mãe, uma guerreira, solteira criou seus 4 filhos em uma época de extrema dificuldade financeira. Já enfrentava preconceitos por ser moradora de favela, sem estudos e mulher, e só piorou por sua decisão de divorciar, assumir as rédeas de sua vida e não continuar abaixando a cabeça para ninguém de nossa sociedade machista. Ela nunca nos deixou desamparados em nenhum momento. Essa mulher é a minha inspiração de vida.

Juliana me motivou a estudar, a não desistir do meu sonho de trabalhar com programação e me mantém nos trilhos até hoje. Se não fosse por ela eu não teria focado em meus estudos e não trabalharia com o que eu gosto.

# Agradecimentos

Agradeço, principalmente, ao Carlos Zambrana e à Andréa Zambrana que, em uma conversa de almoço, me incentivaram a continuar compartilhando conhecimento sobre carreira e programação em uma das vezes que pensei em parar de escrever na internet.

Ao Vitor Mantovani, por clarear as minhas ideias quando eu planejava qual assunto escrever neste livro, me incentivando a falar sobre carreira.

Ao Paulo Silveira, por todo o apoio que dá às comunidades de tecnologia junto à Caelum, Casa do Código, Alura e iniciativas internas e por também me ajudar a encontrar a ideia central para o meu livro.

Aos meus revisores e revisoras pessoais: Fernanda Bernardo, Luiz Felipe Limeira, Mateus Malaquias e Leandro Bighetti.

Ao Wagner Alcyr, pelo meu primeiro emprego em uma empresa de tecnologia, na WL INFO (agora WL Solutions), ao Humberto Oliveira, pelo meu primeiro emprego como programador, na ResideWeb (agora Grupo Reside) e ao meu primeiro mentor em desenvolvimento de software e finanças, Bruno Salgueiro.

## Sobre o autor

Desenvolvedor de software, começou a carreira em programação como full-stack, atuando com a linguagem PHP e migrando para front-end quando JavaScript ganhou grandes responsabilidades em aplicações de larga escala, dando à pessoa desenvolvedora de software a possibilidade de pensar em interfaces Web como aplicações, não mais como páginas estáticas.

Apaixonado por open source, software livre, conhecimento aberto, comunidades e afins, fundou o FEMUG-ABC, um grupo de meetups de desenvolvedores front-end, participou da fundação do evento ABCDev, um evento de desenvolvimento de software na região do Grande ABC Paulista e é criador da Training Center (<http://trainingcenter.io>), uma comunidade inclusiva focada em ajudar pessoas a entrarem na área de desenvolvimento de software.

Como uma pessoa que veio da favela brasileira, entende bem a dificuldade das pessoas menos privilegiadas, os grupos sub-representados, e busca apoiar a diversidade e inclusão de minorias na área de programação de computadores.

Desde 2014 contribui com a vida das pessoas incentivando a entrarem na área de programação através do blog pessoal [woliveiras.com.br](https://woliveiras.com.br) (<https://woliveiras.com.br>), palestras, workshops e participação em comunidades. Acredita cegamente que somente a educação pode mudar a sociedade em que vivemos, e por isso compartilha pensamentos e conhecimento pela internet afora.

# Prefácio

Quando William me pediu para escrever o prefácio do livro dele eu fiquei extremamente emocionada. Uma leitora viciada como eu sabe que é uma honra enorme escrever um prefácio, ainda mais se o autor é alguém que você admira e considera como um irmão.

Antes de perceber sequer o que estava fazendo eu simplesmente aceitei a encomenda. Fiquei muito feliz. Conteï para o meu marido, ele ficou muito orgulhoso, comentamos durante alguns minutos e só ai percebi o que tinha feito. Obviamente entrei em pânico. No interior da minha cabeça ressonava a seguinte frase: “O que foi que eu fiz? eu não tenho condições de escrever o prefácio de um livro, pessoas que escrevem prefácios são pessoas bem qualificadas, sabem o que estão fazendo, você não!”. Mas eu já tinha me comprometido, então tinha que dar um jeito.

Comecei a ler o livro, esperando um livro com técnicas sobre como programar, como escrever algoritmos, comparações sobre diferentes linguagens e coisas desse tipo, e a verdade é que cada capítulo me deparei com uma Andréa mais surpresa.

Com uma linguagem simples, o autor apresenta a quem está lendo o universo da programação, seus conceitos básicos, sua história e vai nos guiando capítulo a capítulo por uma trilha de conhecimento. Conhecimento este, muito necessário para quem quer vir para a área do desenvolvimento de software.

As pessoas chegam até a programação por diversos motivos e é relativamente simples encontrar conhecimento na internet e em livros especializados, mas este conhecimento geralmente vem em forma muito pontual e específica, ou então em livros enormes e profundos sobre uma linguagem específica, o que para quem está lendo se torna cansativo e desencorajador.

O que William faz com este livro é criar uma trilha de conhecimento cada vez mais encorajadora, que nos incentiva a questionar, que nos ensina a estudar e a procurar mais conhecimento fazendo as perguntas certas.

Inclusive, não para minha surpresa, em alguns momentos, ele nos leva pela mão e nos ajuda a quebrar alguns paradigmas e estereótipos divulgados pela mídia em geral.

Seria uma irresponsabilidade da minha parte, além de uma mentira, dizer que este livro é o único que uma pessoa precisa para iniciar na área, mas com certeza posso afirmar que é um ótimo começo. Especialmente para as pessoas que não decidiram ainda se querem investir o seu tempo aprendendo a programar ou que acham que programação pode não ser para elas por qualquer motivo.

Eu tinha 27 anos em 2007, quando comecei na área. Tive que estudar de forma completamente autodidata, pois não existiam muitos cursos sobre programação e os poucos que existiam estavam muito fora da minha realidade por questões financeiras. Todo o conhecimento a que eu tinha acesso era uma ou outra videoaula e alguns tutoriais. Posso afirmar aqui que eu teria adorado poder colocar minhas mãos em um livro como este, pois teria me ajudado a começar com um conhecimento mais estruturado e tenho certeza de que, hoje, seria uma profissional com uma base muito mais completa.

É por conta disso que agora, após ter concluído a leitura desta obra, escrevo estas linhas encorajando você que está lendo este prefácio a continuar com a leitura deste livro e se dar uma chance de conhecer um pouco mais e a aprender o que é o universo da programação.

Boa leitura!

*Andréa K. F. Zambrana*



# Introdução

Este livro é uma coletânea de informações adquiridas através do tempo, tendo buscado trabalhar com desenvolvimento de software e depois participado de diversas iniciativas focadas em inserir pessoas na área de programação, além das experiências vividas em minha carreira.

Eu conto aqui todas as estratégias que aprendi e utilizei para partir de “uma pessoa com pouco estudo e sem profissão” até me tornar desenvolvedor de software (sem faculdade), uma profissão muito reconhecida no mercado de trabalho atual e futuro.

Aqui encontramos as dicas necessárias para que uma pessoa saia do zero (vinda de outras áreas de atuação, saindo do Ensino Médio ou como estudante de alguma universidade) e vá até o seu primeiro emprego na área, ou mesmo para buscar um crescimento rápido em sua carreira profissional como programador(a).

Eu considero essas dicas importantes para remover algumas barreiras que as pessoas encontram ao pensar em entrar para a área de programação, como os estereótipos, a falta de privilégios sociais, as perguntas sem respostas em fóruns de tecnologia, além da falta de conhecimento sobre planejamento de carreira, que não aprendemos no ensino formal.

## A quem se destina este livro

Este livro é destinado para qualquer pessoa que pensa ou ainda tem dúvidas sobre entrar na área de desenvolvimento de software, assim como para pessoas que estão iniciando na área e se sentem perdidas.

Eu reuni aqui as dicas que dou para as pessoas a que eu presto mentoria, por isso são destinadas a qualquer perfil, além de ser para qualquer área dentro da área de programação, como front-end, back-end, full-stack, mobile, games, engenharia e análise de dados etc. (se você não conhece esses termos, pode se acalmar, eu explico tudo isso).

## **Visão geral**

Partimos de um ponto inicial em que assumimos que a pessoa que está lendo não sabe nem mesmo definir o que é programação, passando pelas diferentes áreas de atuação, dicas para conseguir o primeiro emprego, até o que aprender para se tornar um(a) profissional completo(a).

### **Capítulo 1**

Vamos conhecer o básico sobre o universo da programação, que é: o que é programação, o que faz uma pessoa desenvolvedora de software, como é criado um programa e como este programa chega até nós, usuários finais.

### **Capítulo 2**

Vamos entender que programação não é somente uma carreira, mas temos diversas opções quando aprendemos isso, tanto como trabalhar em uma empresa, até criar o nosso próprio negócio.

### **Capítulo 3**

Vamos desmistificar a área de desenvolvimento de software quebrando alguns estereótipos e entendendo de verdade qual é o perfil de alguém que trabalha profissionalmente com programação, até como é seu dia a dia.

### **Capítulo 4**

Um dos capítulos mais importantes deste livro, aqui conhecemos as diferentes áreas de atuação em desenvolvimento de software. Também aprendemos o que é comum entre todas as áreas, além de receber dicas de como escolher qual será nosso destino com programação.

### **Capítulo 5**

Até aqui a pessoa que vem lendo já criou diversas outras questões em sua mente, além de ter pesquisado bastante (se seguiu minhas dicas) e deve

estar mais perdida do que quando começou a leitura, então voltamos a responder algumas perguntas mal resolvidas que encontramos na internet.

## **Capítulo 6**

Neste capítulo, vamos aprender como estudar tudo o que estamos descobrindo que precisaremos ter em nosso currículo com um bom gerenciamento do tempo e inteligência emocional.

## **Capítulo 7**

Vamos conhecer as comunidades de desenvolvimento de software e aprenderemos a melhor maneira de nos envolvermos com esses grupos para aprendermos mais e mais rápido, além de entender a necessidade de contribuir com esses espaços e também como contribuir.

## **Capítulo 8**

Neste capítulo, pegamos tudo o que sabemos até agora e podemos começar a planejar nossa carreira. Para isso, aprendemos a desmistificar as vagas de emprego, a identificar boas oportunidades de trabalho, como conseguir experiência de trabalho sem nunca ter trabalhado formalmente em uma empresa, até mesmo conhecer os salários das áreas de programação para não cairmos em armadilhas.

## **Capítulo 9**

No penúltimo capítulo do livro, conhecemos o que vamos precisar aprender para nos tornarmos melhores profissionais e ir ainda mais longe em nossa carreira.

## **Capítulo 10**

O último capítulo é destinado a minhas dicas pessoais para quem vai encarar a viagem pelo universo da programação.

## CAPÍTULO 1

# Uma introdução ao vasto universo da programação

Desenvolvimento de software não é mais somente uma área de trabalho, mas um verdadeiro **universo**. São tantas opções de áreas de atuação, tantas tecnologias distintas e plataformas para se trabalhar, que é muito fácil de se perder em meio a uma chuva de informação. E o mais legal é que é um universo em constante expansão, algo que está sempre mudando e de maneira tão rápida que parece que não conseguiremos nunca entender tudo o que ouvimos falar sobre esta área. Isso garante que há sempre algo novo para aprendermos.

Este capítulo é uma introdução à área de desenvolvimento de software no geral. Podemos aqui nem mesmo saber nada sobre programação de computadores, pois vamos aprender neste livro o que é cada coisa. Vamos descobrir o que realmente é um software, como eles são desenvolvidos e como são distribuídos para nossos computadores e dispositivos portáteis.

Nos próximos capítulos também nos aprofundaremos nos assuntos relacionados ao futuro da humanidade graças aos avanços na área de tecnologia. Vamos desmistificar a profissão programador(a), conhecer os diferentes caminhos para poder escolher nossa área de atuação como programador(a) e vamos adquirir habilidades e conhecimento para irmos mais longe em nossa carreira.

## 1.1 Tudo começou antes do computador

Parte do nosso treinamento introdutório para conseguirmos viajar por este universo consiste em conhecermos o mínimo da história desta área de conhecimento, que começa muito antes do primeiro computador. Isso mesmo: **programação existe antes do computador**.

Partindo do pressuposto de que programar é escrever um software, e um software são instruções que são lidas pela máquina, precisamos, então, de uma linguagem que esta máquina entenda (vamos ver mais sobre isso mais à frente no livro). Este “idioma” são as tais linguagens de programação. E essas apareceram entre 1842 e 1843, conforme podemos ver na “História das linguagens de programação” na Wikipédia, que diz:

*"As linguagens de programação são anteriores ao advento do primeiro computador moderno. De início, as linguagens eram, apenas, códigos. Durante um período de nove meses entre 1842-1843, Ada Lovelace traduziu as memórias do matemático italiano Luigi Menabrea sobre a mais nova máquina proposta por Charles Babbage, a sua máquina analítica. Com o artigo, ela anexou uma série de anotações que especificavam em completo detalhe um método para calcular números de Bernoulli com a máquina, reconhecido por alguns historiadores como o primeiro programa de computador do mundo."*

E o primeiro computador, chamado Z1, veio somente em 1936, construído por Konrad Zuse, um engenheiro alemão que montou o computador através de relês que executavam os cálculos e dados lidos a partir de fitas perfuradas.

Então, no começo, as linguagens de programação eram usadas para programar essas máquinas de perfurar cartões e com isso conseguiam resolver cálculos complexos em menor tempo do que o cérebro humano, o que seria uma potente arma de guerra mais para a frente na história da humanidade.

Desde aquela época, a área de programação de computadores cresceu tão rápido e surgiram tantas linguagens de programação que é extremamente comum se perder entre elas, entre os ambientes de execução e profissões disponíveis, fora as profissões que estão nascendo e que ainda estão para nascer graças a essa evolução toda.

## 1.2 O que é programação

Nós já ouvimos algum desses termos alguma vez na televisão ou rádio, lido em artigos na internet ou em livros de diferentes temas:

- programação
- programadores(as)
- hackers
- software
- sistema
- aplicativo
- aplicação

A tecnologia vem ganhando muito destaque e a mídia vem deixando claro que programação é uma área que terá muita demanda no futuro - na realidade já temos muita demanda hoje mesmo, em 2018.

Conforme podemos confirmar a seguir no gráfico do Google Trends, site onde podemos ver o volume de buscas por uma palavra, a pesquisa pelo termo "programação" cresceu desde 2004 e ganha atenção das pessoas quando acontece algo na mídia relacionado ao assunto como o lançamento do filme “A rede social”, em 2010, onde o gráfico estava subindo, começa a cair e, logo depois do lançamento do filme em dezembro de 2010, volta a subir.

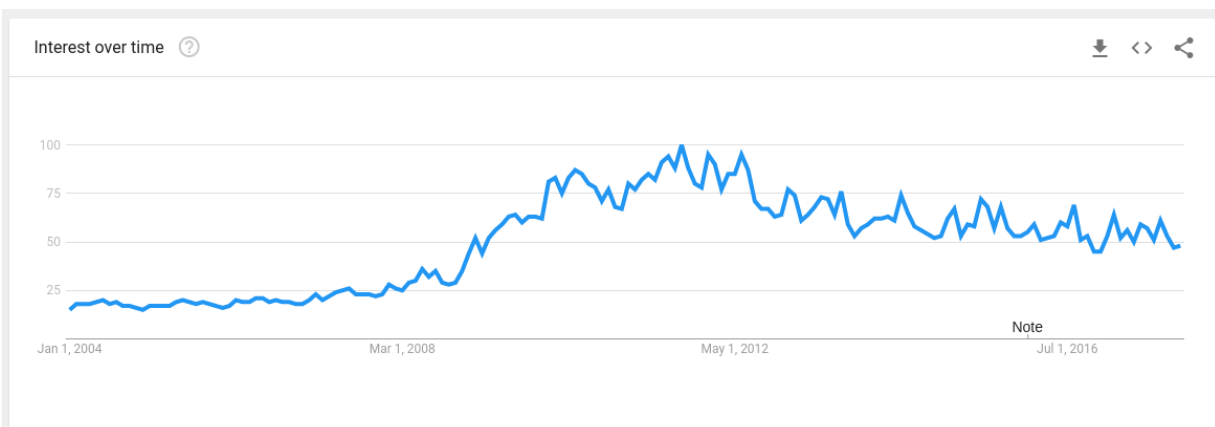


Figura 1.1: Google Trends para a pesquisa por programação

Claro que relacionar o volume de buscas na internet por um termo específico somente ao lançamento de um filme é muita especulação da minha parte. Mas conheço algumas pessoas pensaram em criar uma rede social depois de assistirem a este filme e para isso tiveram que pesquisar sobre o assunto, aumentando esse volume.

O mesmo acontece se acompanhamos o interesse das pessoas por “programação de computadores”.

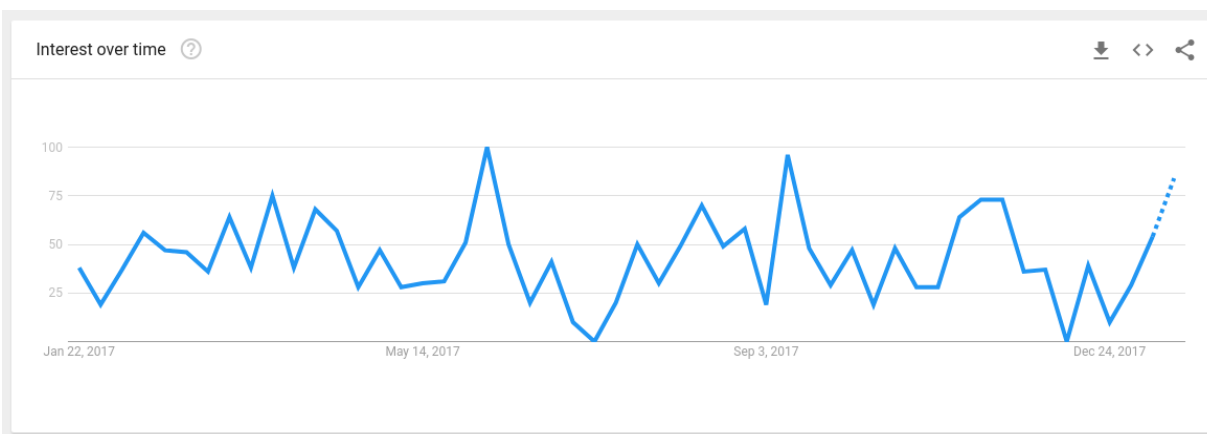


Figura 1.2: Google Trends para a pesquisa por programação de computadores

E quando pesquisamos pelo interesse das pessoas em “criação de aplicativos”:

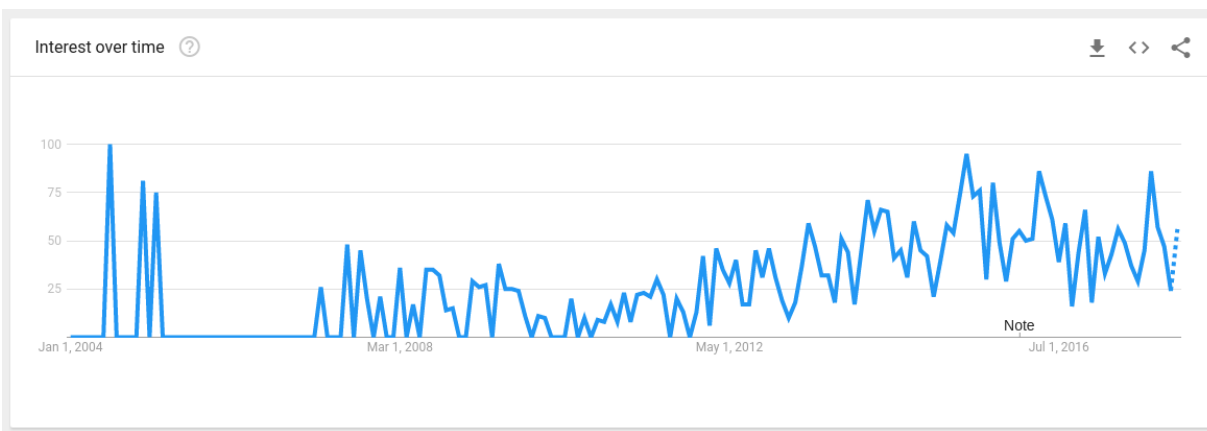


Figura 1.3: Google Trends para a pesquisa por criação de apps

Porém, algo pode ter passado despercebido pela mídia, que é: **o que é, de fato, programação?**

Sem os programas que o controlam, o computador não passa de uma máquina inanimada. Imaginemos nosso computador como um foguete. Programar é o ato de passar coordenadas para o foguete de como ele deverá se comportar para que você vá a milhares de lugares. De maneira bem simplista, claro, isto é programação: escrever instruções para computadores executarem tarefas.

Agora, imagine que ir de um planeta para outro no espaço seja um problema que uma pessoa precisa resolver. O fato de escrever as instruções para o foguete seria o mesmo que resolver um problema, certo?

Programar é isso: resolver problemas por meio da escrita de instruções para um computador executar.

Essas instruções podem ser escritas em uma linguagem de programação, que será compilada em linguagem de máquina e será, enfim, executada pelo nosso computador, smartphone, placa eletrônica etc.

Não se preocupe com os termos novos que vão aparecendo ao decorrer da leitura deste livro, entenderemos tudo isso conforme formos explorando mais a fundo este vasto universo.

### **1.3 O que é um programa de computador**

Sabendo que programar é escrever instruções e que estas instruções podem ser escritas em linguagens de programação, então o que realmente é um programa?

Ao nosso redor temos programas funcionando o tempo todo e às vezes nem percebemos. Um programa de computador nada mais é que a junção das instruções que nós escrevemos (a programação propriamente dita) em uma



linguagem de programação que pode ser compilada (ou não) e executada em computadores, smartphones, placas eletrônicas etc. Ao código escrito por quem desenvolve o software damos o nome de código-fonte.

Compilação, que citei anteriormente, é o processo de pegar um código-fonte escrito em uma linguagem de programação que, normalmente, se aproxima bastante da linguagem humana (se isso, faça aquilo; enquanto isso, faça aquele outro) e transformar em linguagem de máquina que o computador vai entender e fazer o que mandamos (01010011 01101111 01100110 01110100 01110111 01100001 01110010 01100101).

Isso significa que: algumas linguagens podem ser escritas e executadas da maneira como estão e outras precisam passar por um processo que as transforma em outra coisa antes de rodar no computador. Este processo de transformação é feito por um programa chamado compilador e o processo de leitura do código é feito pelo interpretador da linguagem da programação.

Se a linguagem é interpretada, ou seja, executada como é escrita, então o interpretador precisará ser instalado onde o programa funcionará, seja em um sistema operacional, em um smartphone, uma geladeira etc.

Se for uma linguagem compilada, quando nós escrevemos o programa em uma linguagem, esse programa passa por um outro software (o compilador) que o transforma em código de máquina e então podemos executar no computador. O compilador só precisa estar instalado onde for acontecer o processo de compilação; normalmente em um servidor, que é responsável por isso, e somente o código gerado é enviado para o ambiente de execução ou plataforma (nosso computador ou um servidor, por exemplo).

Você já deve ter rodado um programa .exe no Windows. Este foi um software que foi compilado para a plataforma Windows.

Para clarear a visão, vamos dar uma olhada neste exemplo de código-fonte escrito em uma **linguagem de programação de alto nível**, o JavaScript:

```
alert('Hello, world!');
```

Para executar o código, bastaria utilizar nosso navegador, pois ele possui um interpretador de JavaScript embutido.

Para tal, bastaria abrirmos as ferramentas de desenvolvedor do navegador, que, na maioria dos casos, são acessadas clicando em menu , mais ferramentas e em seguida em ferramentas de desenvolvedor .

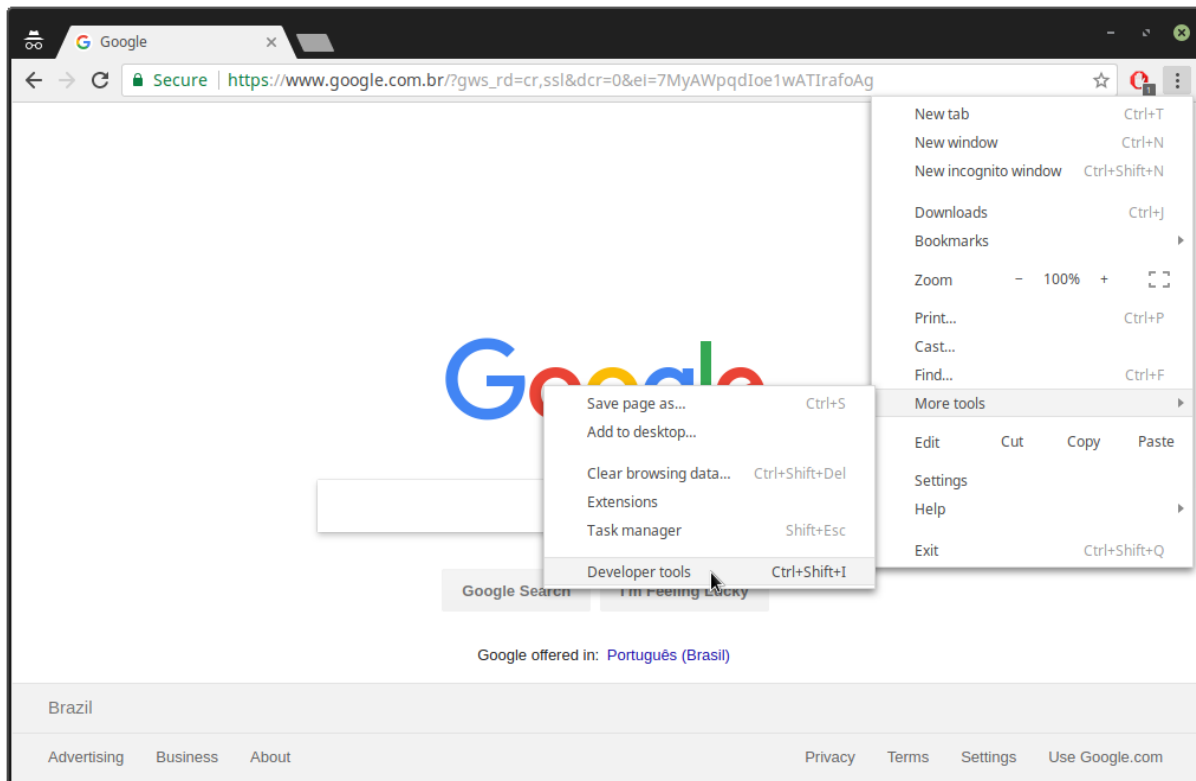


Figura 1.4: Acessando as ferramentas de desenvolvedor

E então colar o código na aba console da caixa de ferramentas que apareceu e o navegador iria exibir uma mensagem de “Hello, world”.

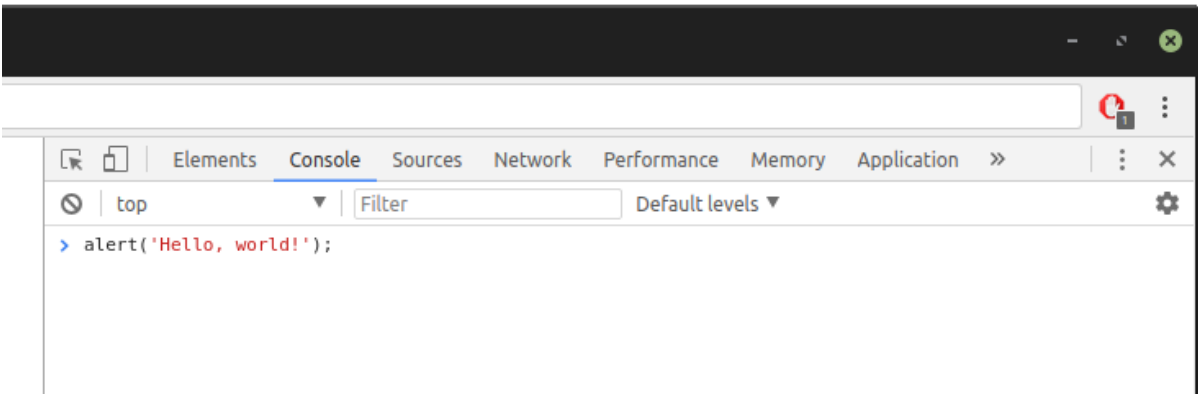


Figura 1.5: Enviando nosso hello world

O nosso ambiente de execução (o navegador) vai se encarregar de transformar as instruções em código de máquina e executar em tempo real.

O comando da linguagem JavaScript `alert` fará um pop-up pular na tela com a mensagem que colocamos entre aspas. O legal é percebermos que linguagens de alto nível são escritas quase de modo natural para nós. Se traduzirmos “alert” do inglês teremos a palavra “alerta” e é justamente o que acontece no navegador, ele envia um alerta para nosso usuário (o pop-up).

Outro exemplo de linguagem de programação de alto nível é a linguagem Lua, que é muito utilizada por jogadores do game de MMORPG (jogos RPG de múltiplos jogadores online) Tibia para configurar programas que automatizam tarefas chatas e roubar no game, os bots. Sua sintaxe (disposição das palavras na frase e das frases em um discurso) é muito inteligível mesmo sem termos muito conhecimento em programação.

```
if itemcount("mana potion") < 20 then
  gotolabel("refiller")
end
```

Se traduzirmos do inglês, teremos algo como:

Se a contagem de itens de “mana potion” for menor do que 20, então vá para o identificador “refiller”

Sendo que `itemcount`, `"mana potion"` e `"refiller"` são palavras inventadas pelas pessoas que programaram este bot, não específicas da linguagem, e executam algo específico que o programador ou programadora especificou. Mas perceba que as instruções não passam de conexões lógicas entre as palavras `"se"` ( `if` ), sinal matemático de menor ( `<` ), `"então"` ( `then` ), `"vá para o identificador"` ( `goto label` ), `"fim"` ( `end` ). É compreensível para um ser humano com conhecimento de inglês,

Agora um exemplo em uma linguagem que se aproxima da linguagem de máquina, uma **linguagem de baixo nível**, o Assembly:

```
section      .text
global      _start

_start:

    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80

    mov     eax, 1
    int     0x80

section      .data

msg         db  'Hello, world!', 0xa
len         equ $ - msg
```

Exemplo extraído de *Introduction to UNIX assembly programming* (<http://bit.ly/asm-hello-world>).

Ambos os exemplos de JavaScript e Assembly são um programa de computador que exibe uma mensagem na tela, a mensagem `"Hello, world!"`, mas percebemos que a diferença de escrita/leitura entre linguagens de alto e baixo nível são bem nítidas. Ainda mais quando lemos o trecho do programa escrito em Lua.

Até aqui você já aprendeu que um programa nada mais é do que instruções escritas em uma linguagem de programação em um arquivo (código-fonte) e que existem linguagens que “se aproximam do metal”, que são as de baixo nível, e linguagens que se aproximam de nossa maneira de nos comunicarmos, as de alto nível.

Neste momento você deve estar pensando “então como ou onde eu escrevo um código-fonte?”, “preciso de um supercomputador para programar?”, “o que mais preciso conhecer antes de escrever código pela primeira vez?”.

O objetivo deste capítulo até aqui é este mesmo: gerar questionamentos. Vamos partir para o próximo nível e entender como é desenvolvido um programa de computador.

## **1.4 Como é desenvolvido um programa de computador**

Já sabemos o que é programação, o que é um programa e o que são linguagens de programação, mas, então, como escrevemos um software?

Existem diversas maneiras de se escrever um programa de computador e isso vai variar conforme a linguagem de programação e ambiente de execução deste programa, mas tudo começa a partir de uma ideia ou problema que precisamos resolver e uma pessoa com habilidades em programação com um computador em mãos.

Se a pessoa estiver escrevendo um código-fonte para rodar no navegador, como JavaScript, ela pode usar qualquer editor de texto comum e depois é só colocar para rodar em uma página web, pois o navegador fará o resto do trabalho de compilar e interpretar a linguagem.

E se a pessoa estiver desenvolvendo para smartphones com o sistema operacional Android e a linguagem Java ou Kotlin?

Aí é um pouco diferente. Normalmente, a pessoa vai usar uma IDE, ferramentas de escrita de código muito poderosas e um pouco mais complicadas do que simples editores de textos para escrever o código e

compilar. Isso para tornar o trabalho mais produtivo, pois, como veremos adiante, IDEs possuem muitos recursos para facilitar a vida de quem está desenvolvendo as utilizando.

A escolha entre editor de texto e IDE é bem pessoal, mas em alguns casos (para algumas linguagens) é quase obrigatório o uso de uma IDE para que você tenha produtividade.

## **Editores de texto**

Editores de texto são programas simples, como o bloco de notas do Windows (sim, dá pra programar usando o bloco de notas) ou o Gedit, ferramenta equivalente ao bloco de notas para Linux. Mas, para facilitar a nossa vida *codificando* de maneira profissional, optamos por editores de texto com poderes extras, como destacar a linguagem de programação deixando o texto colorido para facilitar a leitura do código, opção de autocompletar o que estamos escrevendo e mais algumas coisas legais que aumentam a nossa produtividade.

Um exemplo de editor de textos desse tipo é o Atom, um editor gratuito e de código aberto com diversos complementos que podem deixá-lo ainda mais potente (e pesado na memória do seu computador, caso sua máquina não seja muito potente).

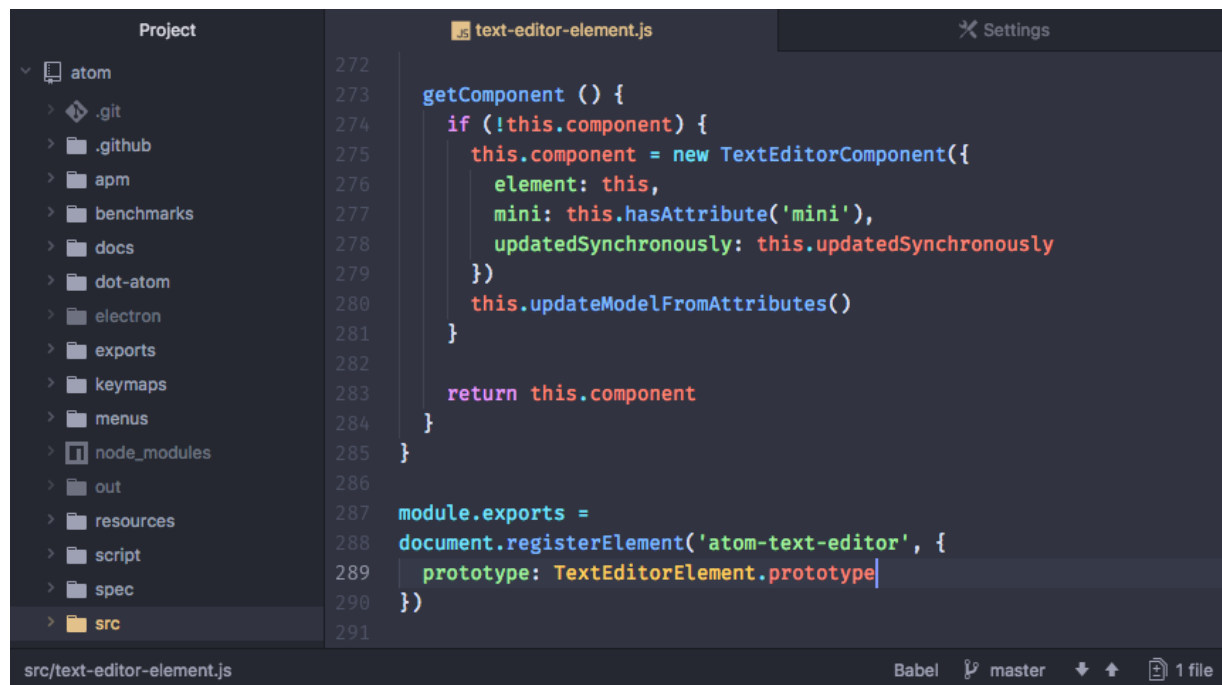


Figura 1.6: Imagem do Atom Editor (extraída do site oficial)

## IDEs

Já as IDEs são ambientes de desenvolvimento completos. Nessas ferramentas temos, além de todos os benefícios dos editores de texto profissionais, opção de conectar em bancos de dados direto de sua interface, geração de modelos gráficos que mostram como todo o nosso código está organizado, emuladores do ambiente de execução que mostram dentro da IDE como o programa vai se comportar, análise do código em tempo de desenvolvimento, reescrita de código de vários arquivos com só um comando e mais utilitários ainda.

Um grande exemplo de IDE é o Android Studio, que é utilizado no desenvolvimento de aplicativos para Android.

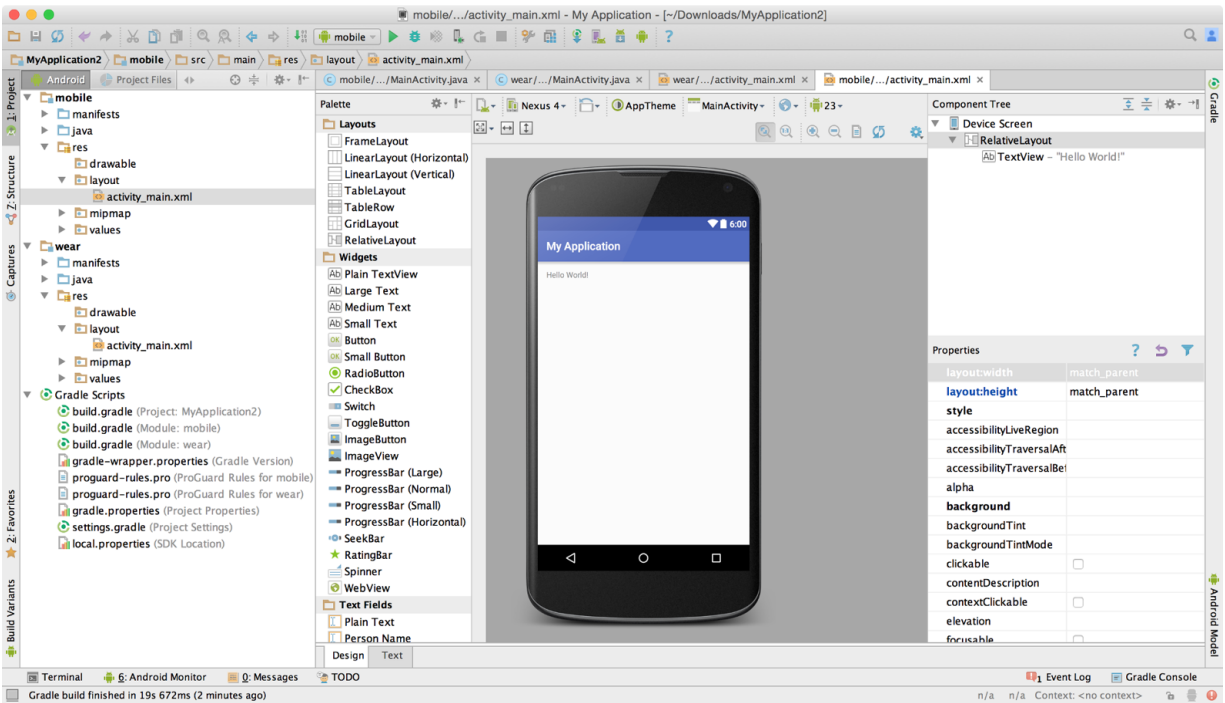


Figura 1.7: Imagem do Android Studio (extraída do site oficial)

Observando as imagens do Atom e do Android Studio, percebemos que o segundo parece mais complicado por ter muito mais botões na tela. Porém, isso é uma “meia verdade”. O que as IDEs têm de complicadas elas têm de recursos para tornar o desenvolvimento mais ágil .

Muitas pessoas entram na briga entre **IDE vs. editores de texto** justamente porque hoje os editores também possuem quase tudo do que precisamos para performar bem, mas isso é uma escolha muito pessoal. Devemos testar cada opção e ver com qual delas nos saímos melhores de acordo com o nosso contexto. A não ser que a linguagem de programação com que formos trabalhar necessite de IDEs por causa da complexidade de escrita de código e sejamos obrigados(as) a utilizá-la sem a opção de testar os editores.

Então, para escrever um programa, tudo o que precisamos é de uma pessoa com conhecimento em programação, uma ideia ou problema, um computador e um editor de textos ou IDE. A pessoa vai escrever o código-fonte e distribuir este código de alguma maneira.



De acordo com que nos aprofundamos neste universo vamos percebendo que programação de computadores não é algo tão complicado como é pintado por aí.

## **1.5 Como um programa é distribuído**

Como já vimos, existem diversos ambientes/plataformas de execução de programas. Podemos rodar um programa em um sistema operacional no nosso computador pessoal, em um servidor na internet, em um supercomputador, em placas eletrônicas, em televisores, em smartphones e em locais que nem sequer poderíamos imaginar que roda um software.

Cada plataforma tem suas peculiaridades e a forma como um programa é enviado para cada uma difere um pouco ou muito de acordo com elas.

Vamos explorar agora como um programa é empacotado e enviado para vários lugares. Como um programa é distribuído para nós, usuários, instalarmos ou usarmos.

Visando não fazer um compilado muito grande e acabarmos viajando por muito conteúdo, vamos separar por partes: primeiro veremos sistemas Web e websites, em seguida sobre desktop e mobile (apps Android e iOS), que são as áreas mais palpáveis para nós, pessoas comuns.

Porém, antes disso, vamos entender qual a diferença de sistema, software ou aplicativo, pois não são a mesma coisa.

### **Sistema, software ou aplicativo**

Nosso editor de textos favorito é nada mais, nada menos, que um software. Ele é chamado de software porque sua função é limitada à edição de texto. Ele faz somente uma coisa.

Não é o que o software faz que define que ele é um software, mas o fato de possuir somente uma função. Um software pode fazer algo bem simples ou algo muito complexo. Se é um programa de gerenciamento de estoque de

uma loja e gerencia entradas e saídas pode ser chamado de software e se for um programa que roda na máquina de uma sala de cirurgia para analisar em qual vértebra fazer uma incisão, também é um software.

Um sistema, por sua vez, é um conjunto de softwares. O nosso Android/iOS ou nosso Windows/Linux/Mac são sistemas. Sistemas operacionais que controlam todo o computador. No exemplo de um sistema operacional, o mesmo controla milhares de outros softwares, pois ele gerencia o som, o display, o teclado, as entradas e saídas, o que é exibido no display, a navegação na internet, a conexão com outros sistemas etc.

Imagine um sistema completo que gerencia toda uma empresa, um ERP (*Enterprise Resource Planning*, sistemas completos de gestão empresarial). Ele possui controle de estoque, controle financeiro, gerenciamento de pessoas, controle de caixa etc. Cada parte deste sistema é um software, um módulo do sistema, e que poderia até existir sozinho, porém faz parte de um todo e esse todo é o sistema.

O autor André Luis também explica a sutil diferença entre software e sistema da seguinte maneira no portal Profissionais TI:

"Escrevemos linhas de código na nossa ferramenta de desenvolvimento e compilamos os arquivos para gerar um executável, certo? Esse executável (também conhecido como artefato) é o que chamamos de software. Em outras palavras, é o programa que será instalado no computador do usuário e disponibilizado para uso. Mas não é só isso! O termo “software” ainda engloba os arquivos que serão distribuídos com o executável, como bibliotecas, banco de dados, demais arquivos de configuração e, claro, a documentação do programa. Um sistema, por sua vez, é um conjunto de softwares que se interagem para atingir um objetivo em comum. Portanto, quando mencionamos “sistema”, estamos nos referindo a uma solução abrangente que envolve várias partes interligadas, oferecendo um composto de funcionalidades para atender as necessidades do usuário." (<http://bit.ly/software-e-sistema>)

Executável, que é falado no texto do André, significa que o arquivo foi compilado e pode rodar em nosso sistema operacional. Ou seja, um executável é um programa pronto para funcionar.

Já o termo **aplicativo** ganhou fama como sendo algo específico para celulares, porém não é bem assim. Aplicativo ou aplicação é um software com uma função. Ou seja, um aplicativo é um software. Se falarmos aplicativo de celular, então estamos nos referindo a um programa com especialidade única que roda em um sistema operacional de um celular.

Não é errado falar aplicativo de celular ou aplicação web, o mercado adotou esses termos e todo mundo entende quando os utilizamos.

Basicamente os termos vão sempre se afunilando até chegar a sua raiz que é software e sistema. Independente da forma como as pessoas chamam o programa ou sistema que estão desenvolvendo, você agora sabe que tudo, no fim das contas, ou é um software ou é um sistema.

## **Sistemas Web e websites**

A internet é uma plataforma imensa. A maioria das coisas, hoje em dia, se conecta na internet e com isso executam as mais diversas tarefas possíveis. Podemos coletar dados de uso do automóvel para pensar em como melhorar a engenharia por trás do veículo ou estradas ou até executar bloqueio de aparelhos celulares em caso de furto. Mas a maior aplicação do uso da internet é para sistemas Web e websites, vamos entender como eles funcionam.

Programas escritos para rodar na internet podem ser enviados para o usuário final por meio do navegador ou para servidores na internet. O que é executado no navegador é chamado hoje pelo mercado de aplicação Web front-end, e o que é executado no servidor é chamado de aplicação Web back-end.

Navegador é o software que utilizamos para navegar na internet, como o Google Chrome ou Mozilla Firefox.

Já o servidor é um computador conectado na internet com tarefas específicas, como servir páginas web, enviar dados para o usuário, para outro servidor ou executar uma tarefa sob comando do usuário.

Esse modelo de funcionamento com um servidor e uma aplicação que consome os recursos deste servidor é conhecido como arquitetura cliente-servidor e serve para distribuir tarefas entre computadores. O cliente, que neste contexto é nossa aplicação front-end, executa algumas tarefas no computador do usuário e o servidor é responsável por executar outras tarefas.

O modelo cliente-servidor para aplicações Web não é específico da internet, a nossa rede mundial de computadores, mas é um modelo de conexão e comunicação entre computadores muito antigo. Os computadores já se comunicavam no modelo cliente-servidor em redes locais (dentro da nossa casa, na empresa etc.) para dividir recursos como impressoras e fax.

A dinâmica cliente-servidor, no contexto de aplicações web, funciona da seguinte maneira: o usuário acessa um endereço de um site, como o site da Casa do Código:

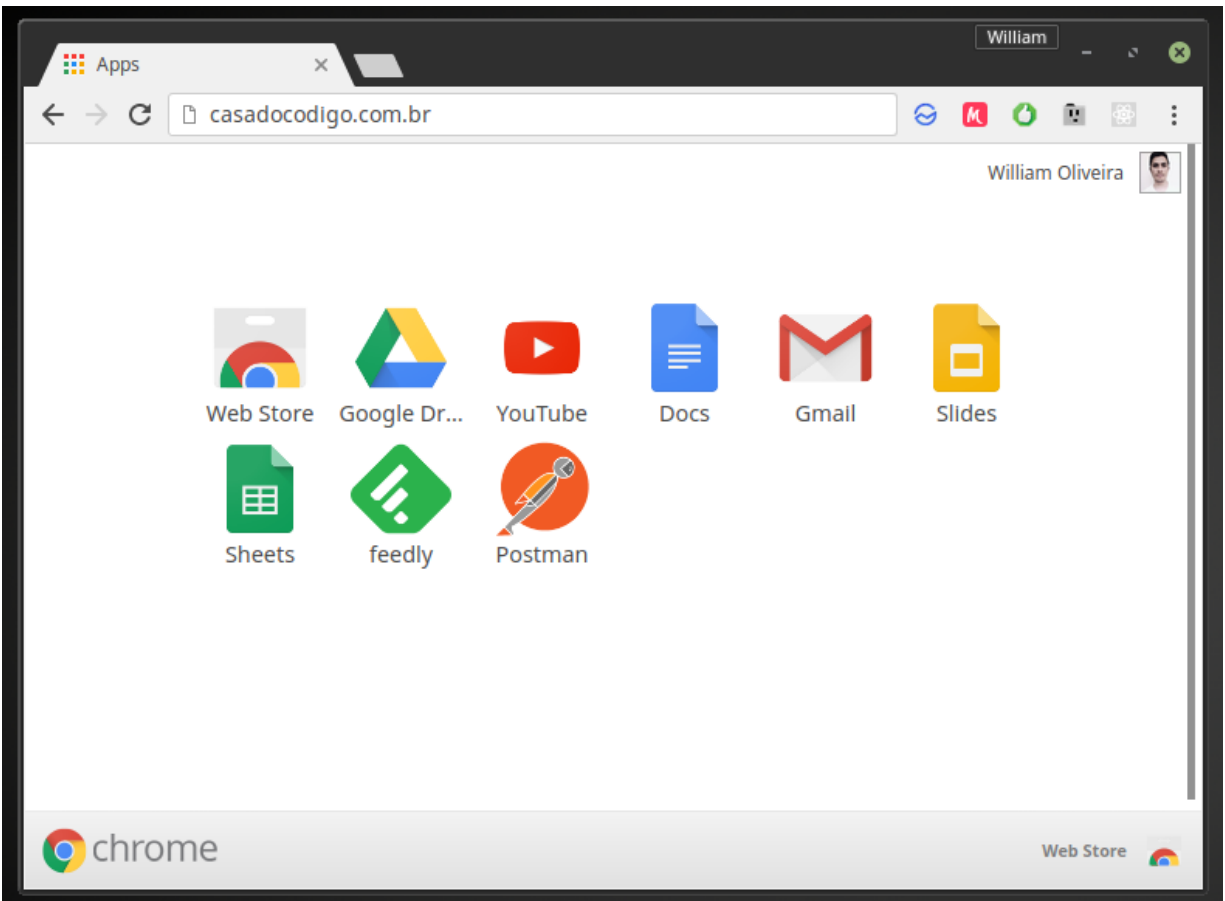
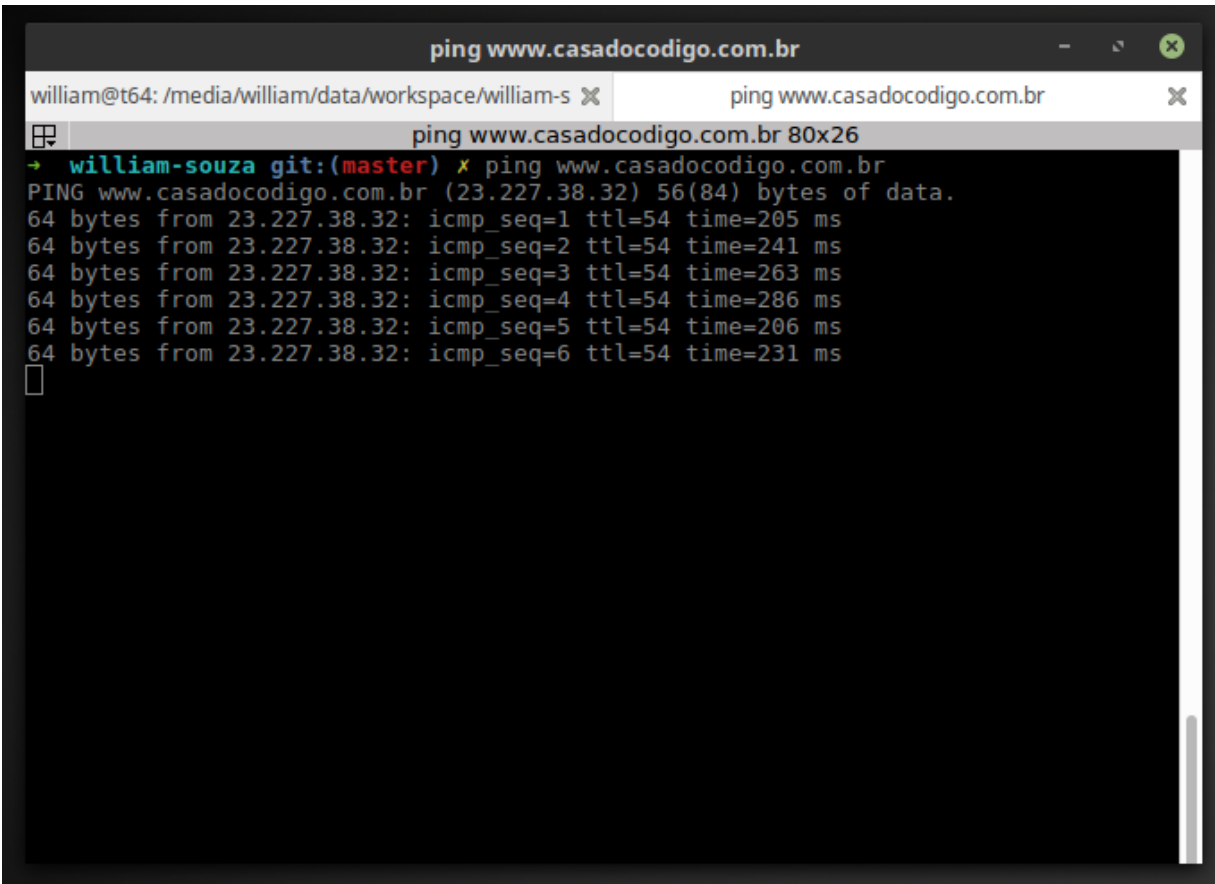


Figura 1.8: Digitando o endereço casadocodigo.com.br no Google Chrome

Assim que o usuário pressiona a tecla enter , o navegador executa diversas tarefas e então abre comunicação com o servidor da Casa do Código, que nesse dia estava no endereço de rede 23.227.38.32, como podemos ver na imagem:



```
ping www.casadocodigo.com.br
william@t64: /media/william/data/workspace/william-s  ping www.casadocodigo.com.br
ping www.casadocodigo.com.br 80x26
→ william-souza git:(master) x ping www.casadocodigo.com.br
PING www.casadocodigo.com.br (23.227.38.32) 56(84) bytes of data.
64 bytes from 23.227.38.32: icmp_seq=1 ttl=54 time=205 ms
64 bytes from 23.227.38.32: icmp_seq=2 ttl=54 time=241 ms
64 bytes from 23.227.38.32: icmp_seq=3 ttl=54 time=263 ms
64 bytes from 23.227.38.32: icmp_seq=4 ttl=54 time=286 ms
64 bytes from 23.227.38.32: icmp_seq=5 ttl=54 time=206 ms
64 bytes from 23.227.38.32: icmp_seq=6 ttl=54 time=231 ms
□
```

Figura 1.9: Endereço IP da casadocodigo.com.br

Não se preocupe se você não sabe o que é um endereço de rede ou como a internet funciona, mais à frente veremos sobre isso.

Assim que o servidor recebe a mensagem que enviamos, ele a processa e envia a resposta, que pode ser uma página web. Podemos ver a resposta do servidor na aba network das ferramentas de desenvolvedor do navegador (vimos como acessar as ferramentas de desenvolvedor na seção **O que é um programa de computador**). Neste exemplo vemos os arquivos que o servidor está nos enviando:

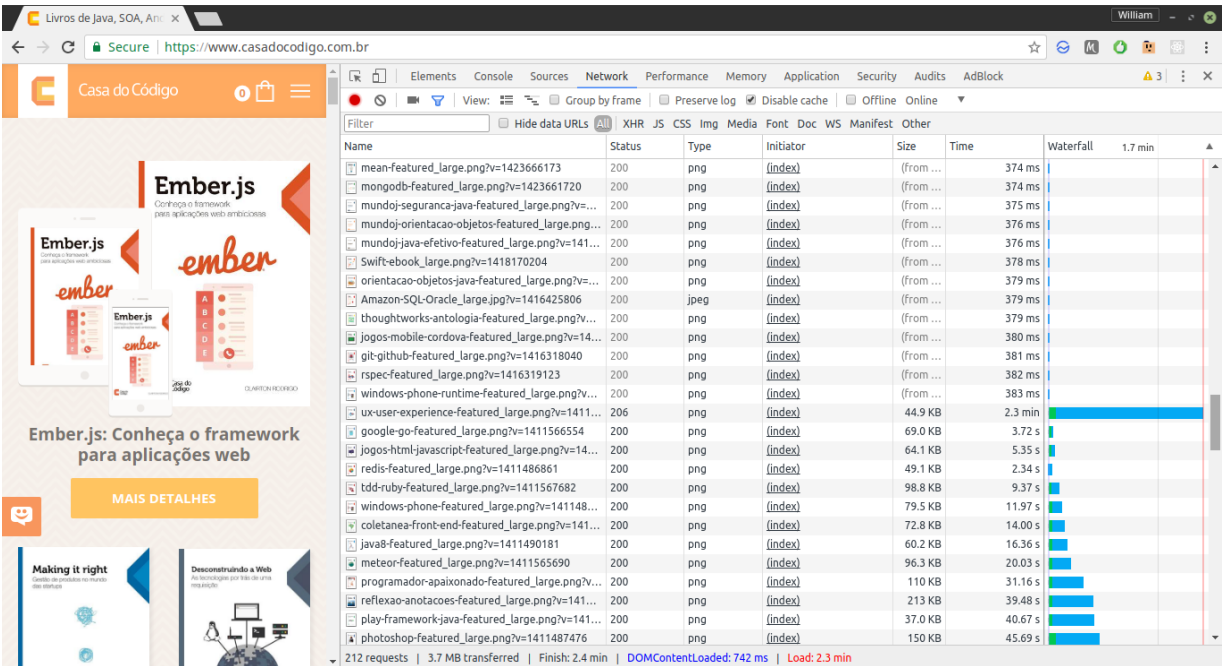


Figura 1.10: Baixando recursos em rede

Conseguimos identificar, por exemplo, as imagens que o servidor nos envia:

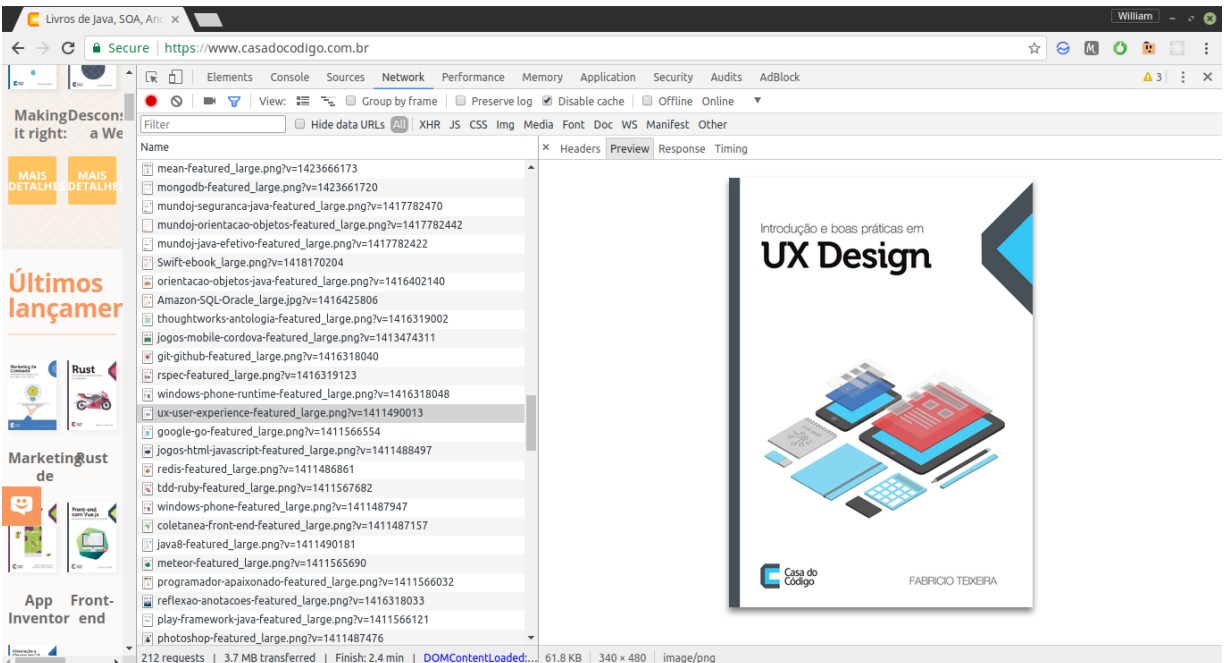


Figura 1.11: Exemplo de imagem baixada pelo navegador

Websites são nada mais que uma ou várias páginas em HTML, uma linguagem de marcação de texto, que é enviada para o nosso computador e interpretada pelo navegador. E através dessa página mais um monte de coisa é baixado, como os estilos visuais da página, que são os arquivos CSS, imagens, vídeos, músicas, arquivos JavaScript e outros arquivos que podem ser importantes para o funcionamento do site.

Sistemas Web diferem de websites em algumas coisas, pois sistemas Web possuem uma complexidade maior. Temos diversas interações com a pessoa que está acessando o sistema, coletamos dados pela aplicação front-end e enviamos para o servidor, o servidor rodará o programa responsável por processar esses dados (o back-end) e enviar o resultado do processamento de volta para o cliente e essa interação continua a modo de executar algo para o usuário, como enviar mensagens na web, controlar suas finanças, ver fotos de seus contatos, compartilhar informações etc.



Já websites, normalmente, não possuem tanta interação com o usuário. Existem ali para exibir conteúdo e somente isso. Alguns websites até possuem alta complexidade de código, porém continuam sendo websites devido à sua natureza única: exibir informações.

Dizer que um sistema Web é mais complexo do que um website pode ser errado, pois pode existir um site com tanta interatividade, animações, ações para proporcionar uma experiência legal para o usuário que no fim das contas é mais complexo de manter do que um sistema web. Porém, aqui estamos focando na natureza do produto digital e não na complexidade de seu código-fonte.

Lembremos da diferença entre sistemas e softwares. O website é um software e um sistema Web é, por baixo dos panos, um monte de softwares trabalhando em conjunto para nos servir algo no navegador.

Grandes e-commerces, por exemplo, possuem muitas integrações entre diversos sistemas para que a nossa compra seja executada. São integrados com sistemas de pagamentos de outros bancos, com softwares de cálculo de frete, softwares de geração de boleto e vários outros.

Então a distribuição de sistemas Web e websites pode acontecer de duas maneiras: se estivermos pensando no que é enviado para o usuário (a aplicação front-end), esse código é hospedado em um servidor que o envia para o usuário quando ele entra no site; se é o código da aplicação que executa direto no servidor (a aplicação back-end), alguém será responsável por hospedar o código no servidor e configurar a aplicação dentro do sistema operacional. Hoje em dia isso tudo é muito bem automatizado para distribuirmos os softwares rapidamente.

Muita coisa acontece por debaixo dos panos quando utilizamos a internet. Se você quiser saber mais sobre como a internet funciona, aprender sobre redes, saber como um navegador e um servidor funcionam, existe um outro livro fantástico que explica tudo por baixo dos panos que é o livro *Desconstruindo a Web: As tecnologias por trás de uma requisição*, do Willian Molinari (<http://bit.ly/livro-desconstruindo-a-web>).

## Softwares desktop

Desktop, quando falamos “softwares desktop”, nada mais é que o nosso computador. Softwares que serão executados pelo nosso sistema operacional, seja Windows, Linux, MacOS etc.

Diferente de aplicações web, estes softwares precisam que seus códigos sejam enviados, instalados e executados diretamente em nossas máquinas e podem até usar dados compartilhados por um servidor Web ou não.

Estes códigos são armazenados em alguma pasta do sistema operacional e cada sistema operacional pode tratar isso de uma maneira distinta. No Windows teríamos os programas instalados na pasta *C:\Arquivos de Programas\WindowsApps* e em sistemas baseados em UNIX, como Linux e MacOS, temos os programas espalhados de acordo com sua especialização.

Vamos analisar um exemplo: o nosso editor de textos Atom.

As pessoas escrevem seu código-fonte e em seguida esse código é compilado e transformado em um executável. Este executável é enviado para o nosso sistema operacional e nosso sistema pode, então, instalar e rodar o editor de acordo com suas especificações.

Se acessarmos o site do Atom (<http://bit.ly/atom-editor>), veremos que ele é um software de código aberto. Isso significa que o código-fonte do software está armazenado em algum lugar público onde podemos pegar esses códigos, analisar e até mesmo editar.

Se acessarmos o local onde está armazenado o código-fonte do Atom veremos uma grande quantidade de arquivos. Este monte de arquivos é compilado em um arquivo único executável pelo sistema operacional e, para que as pessoas baixem este executável, será necessário um website com um link para o código compilado.

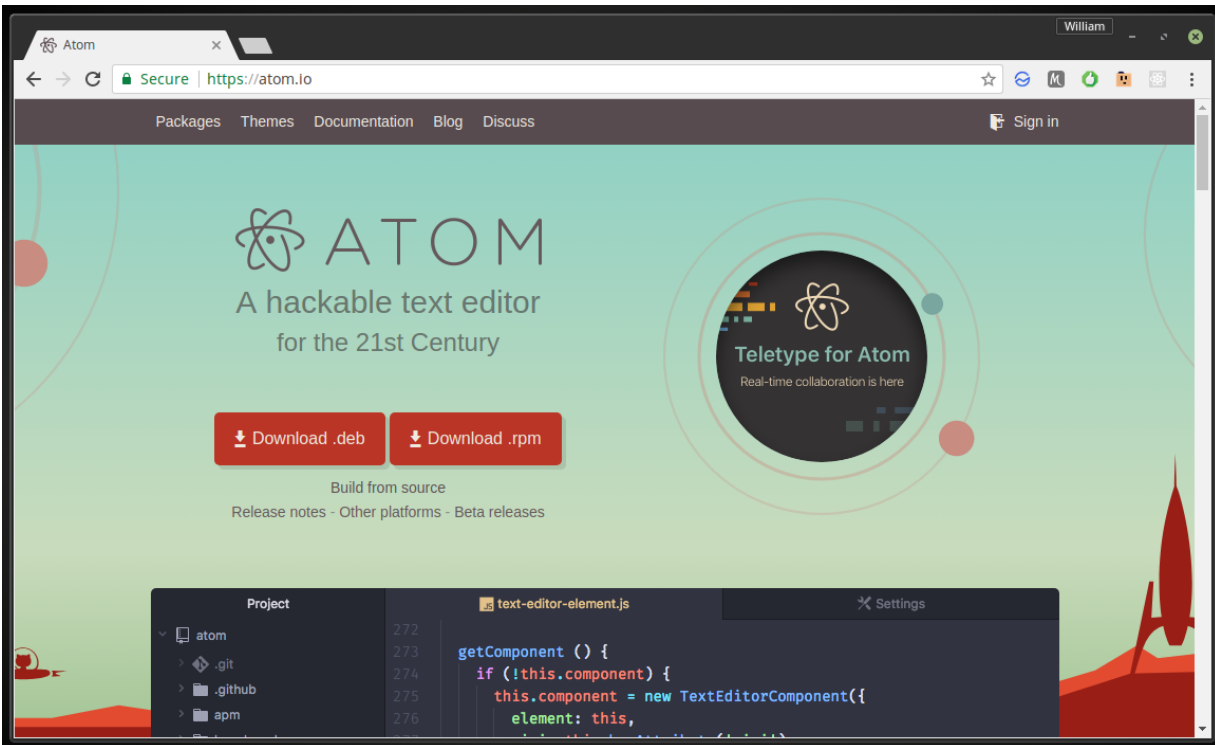


Figura 1.13: Página de download do Atom Editor

Note que para cada sistema operacional é necessária uma versão de instalador:

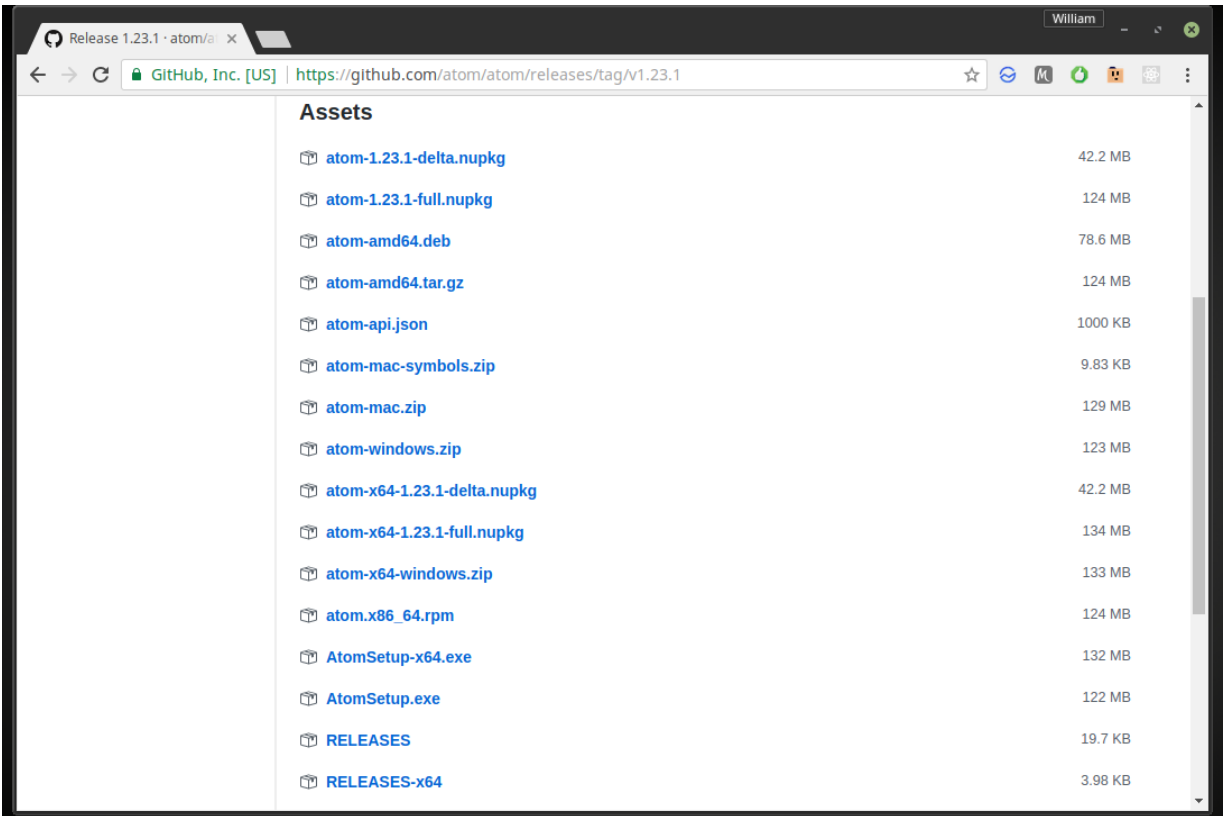


Figura 1.14: Diferentes distribuições do Atom Editor

Isso porque o código executável difere de um sistema operacional para outro de acordo com suas peculiaridades, como já foi citado.

No caso do Atom, a linguagem de programação utilizada é o JavaScript, mas poderíamos utilizar linguagens como Java, C#, Python, Rust, Golang e diversas outras para criar softwares desktop.

Então a maneira de distribuir programas desktop é com um website. Porém, também conseguimos compartilhar programas por meio de pen drives, diretórios de rede ou alguma outra maneira de copiarmos o código executável de um computador para outro.

## Mobile Apps

Aplicativos de celular são muito parecidos com softwares de desktop, com a diferença de que grande parte deles precisa ser instalada diretamente pela

loja de aplicativos do sistema operacional, seja Android ou iOS, por exemplo.

Em distribuições Linux mais atuais também possuímos centrais de aplicativos bem parecidas com as de aplicativos mobile. Isso facilita muito a vida de usuários mais leigos em tecnologia. Recentemente a Microsoft também começou a seguir esta linha para seus sistemas operacionais mais modernos (a partir do Windows 8).

Também podemos compartilhar arquivos executáveis de um celular para o outro, como acontece com os programas de computador.

Vamos analisar um fantástico jogo de celular chamado Dungeon Crawl Stone Soup, pois, além de ser um game de código aberto, ele funciona e está disponível na maioria das plataformas mobile.



Figura 1.15: Jogo DCSS

Este jogo é escrito na linguagem de programação C++, como podemos ver no seu repositório no GitHub. Mas a grande maioria de aplicativos de celular é escrita em Java e Kotlin, para Android, ou Objective-C e Swift, para iOS. Normalmente se utiliza a linguagem C ou C++ quando a

aplicação demanda muita performance e recursos mais complexos do sistema operacional, principalmente em games.

Quando olhamos o código-fonte do jogo, reparamos que é um amontoado de arquivos que precisa ser compilado para só depois ser enviado para uma loja de aplicativos para que o usuário possa baixar.

Não é tão simples subir um aplicativo para uma loja ou central de programas. As centrais de programas possuem suas regulamentações que a pessoa que desenvolveu o software precisa seguir. Isso tanto para que o usuário de uma plataforma (Android, iOS) possa ter uma boa experiência de uso quanto para analisar a segurança do app.

Quando lançado o game Pokémon Go, em 2016, diversas pessoas não conseguiram segurar a ansiedade (como eu) de esperar o lançamento em todos os países e baixaram arquivos **.apk**, que é o executável para sistemas Android, diretamente de websites não confiáveis. Isso expõe o usuário a um risco de segurança muito grande, pois não houve verificação de permissões do app, análise profunda do mesmo ou coisas do tipo pela central de apps.

Contudo, algumas empresas utilizam da estratégia de enviar diretamente os **.apks** para seus usuários a modo de poder fazer atualizações mais rápidas, sem passar pelo período de espera de uma central de apps. O que não é muito legal, mas acontece.

## **1.6 Conclusão**

Entendemos, até aqui, que existem diversas maneiras de se enviar softwares para o nosso usuário final. Alguns softwares precisam ser compilados e enviados direto para o computador do usuário e outros rodam em servidores na internet trocando mensagens através de protocolos de rede com nossos navegadores.

Sistemas Web e websites utilizam o modelo cliente-servidor, no qual temos trocas de dados entre ambos e temos softwares rodando no navegador do usuário (cliente) e no servidor.

Softwares Desktop precisam ser compilados e seus executáveis são baixados por meio de um website diretamente para o sistema operacional do usuário, porém podemos enviar os executáveis de um computador para o outro via pen drives ou pastas de rede, por exemplo.

Aplicativos de celular têm a dinâmica parecida com softwares desktop com a diferença de que precisam ser enviados para uma loja de aplicativos com certa regulamentação e também conseguimos compartilhar entre celulares ou alguma outra maneira de enviar seus executáveis para os dispositivos.

Com todo esse conhecimento, já demos nosso primeiro mergulho no fantástico universo da programação, pois entendemos o que é programação, o que é um programa, o que são linguagens de programação e como são escritos os programas e até mesmo como enviar os programas para as máquinas de nossos usuários.

Vamos continuar nossa viagem neste universo analisando no próximo capítulo alguns motivos pelos quais é interessante se envolver nesta área de atuação.

## CAPÍTULO 2

# Por que se envolver com programação

Com o passar dos tempos, a humanidade vivenciou algumas revoluções industriais e tecnológicas que transformaram a vida e a sociedade com a automação, inovação e com maquinários poderosos que substituem o ser humano em diversas posições de trabalho. Hoje vivemos mais uma revolução... Agora, a **revolução digital**.

Todos os dias uma empresa ou pessoa lança alguma ideia fantástica de automação para agilizar processos, diminuir mão de obra ou somente para mostrar que dá para fazer algo com inteligência artificial e algumas placas eletrônicas.

E a área de inteligência artificial nem é algo novo. Ela existe desde 1956, logo depois da Segunda Guerra Mundial. Essa área só ganhou mais visibilidade agora porque a evolução do hardware possibilitou o uso disso em vários locais.

Temos empresas automatizando suas colheitas para economizar água, saber a hora exata de colher os produtos, controlar seus tratores de arado. Outras cujo estoque é todo automatizado e seu setor de logística é incrivelmente funcional e eficaz. Existem as que já estão investindo em nanorrobôs para matar células de câncer e também não podemos deixar de pensar em automação residencial e cidades inteligentes, onde podemos utilizar AI (*artificial intelligence*) para os mais variados fins.

Graças a toda essa revolução algumas profissões vão desaparecer, outras vão surgir e acredito fortemente que **programação** será para todas as pessoas o que hoje é o inglês ou uma faculdade: um requisito básico no currículo.

Toda essa mudança é muito comum e previsível em uma sociedade que busca a evolução. A tecnologia faz parte da mudança e auxilia a inovação, e programação faz parte de tecnologia.



Se o nosso emprego for muito repetitivo e não demandar certo trabalho cognitivo ou criativo, que só um ser humano conseguiria inventar, então corremos o risco de sermos substituídos por máquinas em um futuro não muito distante.

Acredito que utilizaremos em breve softwares que recebem linhas de comandos ou onde inserimos comandos lógicos para aumentar sua capacidade de fazer o que queremos. Já temos há muito tempo essa possibilidade com macros do Excel, por exemplo, que é um código gerado através da gravação de ações do usuário para automação de processos.

A grande questão é: onde puder ter um computador, poderá ter uma pessoa para programar seus softwares. Digo "pode", porque, quem sabe, no futuro os próprios softwares não vão se multiplicar onde houver necessidade em partes triviais dos sistemas por meio de inteligência artificial?

Essa mudança global, por si só, já seria uma boa motivação para que comecemos a observar a área de software com mais atenção, mas vamos nos aprofundar no assunto e conhecer outros bons argumentos para mergulharmos de vez neste universo fantástico.

## **2.1 Programação como treinamento para resolver problemas rapidamente**

Quando estamos aprendendo programação, precisamos aprender a abstrair ações, problemas, fluxos, assim como acontece enquanto estudamos ou trabalhamos com matemática.

Isso porque precisamos entender um problema, traduzir para uma linguagem de programação e então criar uma solução computacional para isso. Na matemática transformamos os problemas em fórmulas para conseguir o mesmo.

Enquanto estudamos programação, somos ensinados a pensar de maneira lógica. Pegar um problema, quebrar em pequenas partes, tratar cada parte

dessas a fim de conseguir avanço na solução e então juntamos tudo e obtemos uma resolução.

O raciocínio lógico pode servir em diversas partes da nossa vida, tanto para trabalhar com computadores quanto para analisarmos melhor os pontos em uma discussão, pois isso não é algo novo. Aristóteles estabelece a lógica como disciplina em sua obra *Organon* e depois disso o trabalho do filósofo foi usado para criar diversos tipos de lógica no Ocidente e na Europa.

Ao programar, adquirimos o pensamento sistêmico, que é uma forma de analisarmos e descrevermos como solucionar problemas. É a capacidade de analisar diversos cenários, eventos e “prever” algo que possa acontecer e se adiantar para enfrentar isso.

Imagina que, com o passar do tempo, isso se torna tão natural para nossa mente que agimos de modo sistêmico usando lógica para resolver a maioria dos problemas cotidianos!

## **2.2 Programação como negócio**

Não é de hoje que empresas ganham dinheiro com código. Existem empresas que desenvolvem sistemas para outras empresas há mais de 100 anos, como é o caso da IBM, que existe desde 1888, tem quase 500 mil empregados e vale mais de 202,5 bilhões de dólares.

Existem diversas maneiras de se explorar o mercado de software, seja criando uma consultoria que desenvolve sistemas para outras, criando um software e dando manutenção a ele como produto principal ou mesmo fazendo os dois, como no caso da Amazon, que possui seu e-commerce de varejo e também sua área de Cloud Computing e prestação de serviços, a AWS (Amazon Web Services).

De longe este deveria ser o motivo menos importante para nos envolvermos com códigos, pois pode ser uma mudança grande em nossas vidas. Seria melhor entrar na área por paixão do que somente pela parte financeira, mas

vamos analisar um pouco as opções quando pensamos em *ganhar dinheiro com código*.

## 2.3 Freelance

A possibilidade de nunca ter um patrão ou patroa, poder fazer nosso próprio horário, trabalhar de casa ou de um café e outros benefícios desse tipo, atrai muita gente para empregos freelance, uma modalidade bem legal e importante de trabalho na qual nos colocamos disponíveis para pessoas e empresas que buscam alguém para fazer um projeto esporádico ou até para um longo contrato.

O mais legal é poder passar por diversos projetos de empresas grandes, médias, pequenas, para clientes comuns também (pessoa física) e conhecer diversos modelos de negócio, projetos fantásticos e muitas vezes ajudarmos alguém a sair do zero ao produto de suas vidas sem precisarmos estar presos a alguém depois que o projeto acaba.

Hoje existem diversos sites que indexam perfis de pessoas que buscam freelancers, seja para produzir um site, aplicativo, um sistema completo ou para arrumar algum bug, fazer algum ajuste em algo já existente. Veremos um pouco mais sobre trabalhar como freelancer nos próximos capítulos.

## 2.4 Criação de um produto digital

Além das várias possibilidades de criarmos uma empresa especialista na criação de softwares para as empresas ou trabalhar como freelancer, também podemos criar um software ou sistema de sucesso baseado em um problema que nós queremos resolver.

Alguns produtos digitais famosos exploram as mais diversas necessidades da sociedade e resolvem com maestria a maioria dos problemas, como o aplicativo 99 (<https://99app.com>) para a mobilidade pública, dr.consulta

(<https://drconsulta.com>) para a área da saúde, NuBank (<https://nubank.com.br>) para serviços financeiros.

Existem sites onde encontramos o melhor imóvel de uma região baseado em nossas preferências, pesquisas, como o VivaReal (<https://vivareal.com.br>), aplicativos de gestão financeira que nos ajudam a controlar nossos gastos e fazer com que o salário dure até o final do mês, como o GuiaBolso (<https://guiabolso.com.br>).

E a forma de monetização de um produto digital pode variar muito e isso deve ser aprendido de acordo com o uso do produto. Existem diversos cursos e livros a respeito da criação de um produto digital, dentre os quais recomendo fortemente a leitura sobre startups, como o *A Startup Enxuta*, de Eric Ries e outros livros sobre empreendedorismo no geral.

Se você tiver o sonho de resolver algum problema social, criar um produto digital pode ser a sua grande chance.

## **2.5 Entender como as coisas funcionam por baixo do motor**

Difícilmente levamos nosso carro em um serviço de mecânica por falta de água ou óleo no motor. O mesmo acontece quando precisamos trocar o pneu de uma bicicleta ou fazer a manutenção da nossa geladeira.

É comum nós mesmos resolvermos um problema desses porque parece algo mais palpável para a maioria das realidades, mas será que não poderíamos resolver algum problema de um sistema que utilizamos no navegador?

Mesmo que não tenhamos acesso ao código-fonte de um sistema, saber por que algo acontece pode nos poupar muito tempo, como saber que um site não está carregando porque tem um script pesado na página, mas que poderíamos desativá-lo e finalmente acessar o conteúdo.

Até mesmo um aplicativo de celular pode ser analisado utilizando as ferramentas de desenvolvedor que podemos ativar nas configurações do sistema operacional mobile.

Saber programar ou saber como funcionam as coisas por debaixo dos panos nos dá o poder de resolver algo e seguirmos nossa vida, assim como podemos parar num posto de gasolina e colocar água no reservatório do carro e seguir nossa viagem.

## **2.6 Para nos mantermos em segurança**

Nossos dados/nossa informação são muito importantes e precisamos buscar nossa privacidade e segurança na internet.

Como sabemos se um sistema está coletando dados invasivos à nossa privacidade se não conhecemos como as coisas funcionam por debaixo do capô, como citado anteriormente?

Quando entramos em um site e ele é lento, clicamos em algo e demora vários segundos para que algo aconteça ou comportamento similar, muitas vezes é por conta da coleta de dados.

Isso não é ruim, nem ilegal (ainda), as empresas coletam nossos dados para melhorarem seus serviços e nos entregarem uma experiência melhor em seus produtos. Porém, algum sistema pode, sim, estar coletando algo que não queremos que seja coletado, como nossa localização, gravando o que falamos próximo aos nossos smartphones etc.

Saber desenvolvimento de software pode nos garantir conhecimento suficiente para nos protegemos na internet ou ao menos nos mantermos mais atualizados(as) sobre o mundo dos aplicativos e sistemas que usamos, o que traz uma segurança básica.

## **2.7 Programação como exercício para o cérebro**

O cérebro do ser humano, por mais que seja uma máquina quase perfeita, pode começar a apresentar problemas com o passar do tempo. E não queremos que o nosso controlador central comece a falhar, certo?

O exercício para fortalecer nossa massa encefálica deve ser diário, assim como exercitar nosso físico. Caso não nos atentemos ao cuidado com nosso cérebro, com o tempo começamos a esquecer coisas, a ficar menos ágeis para processar informações, tudo pode começar a ficar mais complicado de se pensar.

Aprender diferentes linguagens de programação tem o mesmo efeito de fortalecimento do cérebro quanto aprender outros idiomas e isso não é só teoria, temos pesquisadores(as) atrás da resposta em seus estudos. Como a professora Janet Siegmund, da Universidade de Passau, que realizou exames de ressonância magnética em 17 voluntários enquanto liam fragmentos de código para um estudo em 2014.

"Descobrimos a primeira evidência empírica de que tanto a linguagem natural quanto a de programação exigem as mesmas áreas do cérebro. Com base nisso, podemos inferir que a compreensão das linguagens de programação e os idiomas naturais parecer similares.", afirma a professora.

Entendemos, então, que a programação pode ser excelente para “fortalecer nossa cabeça”, evitando os problemas que a falta de exercícios cerebrais nos trariam.

## 2.8 Conclusão

Percebemos, até aqui, que as possibilidades são imensas e vão desde um emprego formal até a criação de um produto que pode revolucionar e mudar a história de uma sociedade inteira. Realmente o poder está nas mãos de quem sabe programar.

Mas, mesmo com tantas possibilidades, podemos ainda possuir dúvidas se a área de desenvolvimento é algo em que nos daríamos bem. Podemos sentir receio pela nossa idade, pela falta de acesso à informação até aqui e falta de uma formação de qualidade (falo de Ensino Médio ruim mesmo), medo de

largar uma carreira em que já estamos estáveis... Isso tudo é extremamente normal de se passar por nossa mente quando imaginamos algo novo para nossa vida.

O mais importante é entendermos que existem milhares de possibilidades e nós podemos, sim, encontrar nosso lugar neste vasto universo. No próximo capítulo vamos conhecer um pouco sobre o perfil de pessoas que trabalham com programação e também quebrar alguns estereótipos que temos sobre programadores(as), como o de pessoas desenvolvedoras de software serem mais inteligentes que todo mundo ou serem pessoas que nunca dormem para virar a noite atrás do computador codificando.

## **CAPÍTULO 3**

# **Pessoas que escrevem programas e lançam foguetes**

Muita gente que sente vontade de entrar na área de programação fica com medo do desafio porque algumas pessoas que falam dessa área pela internet ou mesmo em jornais na TV (que trabalham com programação ou não) exageram um pouco sobre a dificuldade de se aprender a programar.

O ego é algo que existe em todas as áreas de atuação e a romantização do que fazemos é clara na maioria das palestras de eventos de desenvolvimento. Todo mundo que sente orgulho do que faz, mais cedo ou mais tarde, vai aumentar um pouco sobre a complexidade da sua profissão em uma roda de conversa para que todo mundo venha a admirar sua função.

Fora essa romantização da área, da parte dos profissionais de mercado para se enaltecer, ainda temos estereótipos alimentados em filmes, séries, preconceito social com pessoas que gostam de estudar (conhecidas como nerds) etc. Sempre que aparece um(a) programador(a) em uma cena de filme, é alguém extremamente inteligente, com QI muito acima da média, normalmente uma pessoa branca, com espinhas, óculos fundo de garrafa, e a maioria das vezes, antissocial.

Isso, além de preconceito com quem atua na área, é muito diferente da realidade e muita gente reconhece isso depois que começa a trabalhar em desenvolvimento ou mesmo quando nos conhece melhor.

Não precisa ser um Sheldon ou a Amy, personagens do The Big Bang Theory, com seus QIs super altos para aprender programação de computadores.

Vamos explorar um pouco mais esses estereótipos e entender o que, de fato, é necessário para que aprendamos a programar.

Já adianto: aprender a programar não é difícil. Fazer um aplicativo ou um sistema não é difícil. O difícil é fazer algo duradouro e escalável, algo que



vai aguentar uma grande quantidade de usuários e vai ficar em funcionamento por muitos anos. Mas quem está aprendendo, quem acabou de começar a desenvolver ou quem só tem curiosidade com programação não precisa conseguir fazer isso desde o primeiro software escrito.

Todo mundo passa pela fase inicial. Experiência é algo que se adquire com o tempo e prática, e não devemos nos desanimar achando que precisamos começar já sabendo de tudo.

### **3.1 Quem são as pessoas que escrevem programas, onde vivem, de que se alimentam**

Descobriremos as respostas para essas perguntas neste capítulo!

Energético, café, pizza, alimentos calóricos dentro do quarto, com a porta trancada para ninguém incomodar ou no escritório com o fone de ouvido para ninguém cutucar. Será que é essa a pessoa que tem o perfil para se tornar desenvolvedora de software?

Como eu comentei anteriormente, as pessoas que trabalham com programação são pessoas normais. Não possuem superpoderes, não se alimentam com uma dieta que lhes proporciona mais neurônios ou algo do tipo e muito menos vivem somente de alimentos não saudáveis. Porém, existem algumas características pessoais que podem influenciar nossa vida rumo ao aprendizado e nós vamos fazer uma análise superficial sobre elas agora. Vale lembrar que toda característica citada aqui pode ser adquirida ou contornada com outra habilidade e/ou disposição pessoal.

Outro adendo importante: os pontos que vamos observar agora não são obrigatórios ou resultado de alguma pesquisa comportamental revolucionária, mas é algo que acabamos observando em pessoas que trabalham com desenvolvimento ao nosso lado no dia a dia ou que encontramos em eventos da área. Portanto, é uma visão muito pessoal que estou transmitindo agora.

## Curiosidade

A curiosidade é uma característica bem comum em pessoas da nossa área. A maioria das pessoas se envolvem com programação pela vontade de entender como o computador e os softwares instalados nos sistemas operacionais funcionam.

Isso pode ser útil para continuarmos sempre com vontade de entender algo novo, o que é outra característica importante de boas pessoas nessa área e que vou aprofundar mais à frente.

No meu caso, eu desmontava meus brinquedos, estragava eletrônicos e, posteriormente, quis entender como o computador e a internet funcionavam. Eu achava fantástico como diversos componentes eletrônicos eram conectados, se comunicavam e faziam a ponte entre eu e meus familiares de outra cidade através das redes sociais e comunicadores online.

Caso você seja uma pessoa curiosa, que desde a infância desmontava coisas para entender como elas funcionavam, programação **pode ser** uma boa área para você. Será um caminho “mais fácil”. Mas, se você não sente tanta curiosidade assim, não tem problema. Com o tempo aprendemos a nos manter em constante atualização profissional mesmo sem ter tanto interesse em algo.

A curiosidade acaba sendo contornada pelo hábito de estudo constante. Quando lançam uma tecnologia nova, nós sabemos que teremos que estudá-la e por isso liberamos espaço na agenda, fazemos cursos, vemos palestras e coisas do tipo.

Esse aprendizado também pode vir por meio do convívio com pessoas da área. Portanto, se não temos essa característica, é legal começarmos a fazermos amizade com as pessoas que consideramos boas e ou simplesmente as seguirmos nas redes sociais.

## Gostar de aprender

A nossa área é muito ampla e está sempre mudando, por isso é extremamente importante gostar de aprender. Todos os dias podem ser

lançadas uma ou mais ferramentas novas que vão facilitar nosso trabalho, uma nova boa prática de programação ou uma outra convenção que vai facilitar a manutenção dos nossos softwares. O tempo todo precisamos nos atualizar sobre o nosso universo.

E quando digo gostar de aprender, não significa que devemos gostar de estar em uma sala de aula, ver videoaulas ou mesmo ficar lendo horas de um livro técnico até decorar. Aprender pode ser mais orgânico na base da tentativa e erro. Isso depende do nosso perfil e de como aprendemos melhor.

Se somos do tipo que gosta de cursos, escolas e relacionados, teremos espaço para aprender lá. Se não somos desse tipo, sem problema, a internet é um universo aberto e acessível cheio de conteúdo para aprendermos algo novo.

Por isso, pessoas curiosas já têm uma habilidade auxiliar nessa área de atuação, pois elas estão sempre querendo aprender.

### **Gostar de resolver problemas**

Programação é resolver problemas utilizando o poder dos computadores, então nada mais justo que pessoas que gostem de programar também sejam pessoas que gostam de resolver problemas. Sejam problemas pessoais, problemas de economia, problemas de comunicação. Somos rodeados de problemas esperando para serem resolvidos por um software.

O que percebo é que a maioria das pessoas programadoras sentem uma vontade imensa de enfrentar um problema para encontrar uma boa solução para ele. Essa é uma característica importante no nosso meio, pois todos os dias temos problemas diferentes para resolver.

Precisamos gostar de passar algumas horas imaginando como facilitar a vida das pessoas, como deixar um processo mais otimizado ou barato em uma indústria ou mesmo como tirar os dados de uma planilha e transformá-los em uma informação útil.

Dessa característica não temos como fugir. O nosso dia a dia é cercado de bugs e novas funcionalidades solicitadas pela área de negócios da empresa ou por nossos clientes.

## **Gostar de tecnologia**

Seria difícil para alguém que não gosta de tecnologia (computadores, smartphones e afins) gostar de programação, pois esse é nosso mundo. É o que nos move e é para o que trabalhamos. Se existe algo comum em pessoas desenvolvedoras deve ser o fato de elas quererem utilizar tecnologia para resolver tudo.

Gostar de tecnologia não é o mesmo que ter um parente próximo que sabe tudo sobre redes sociais ou estar antenado com o que está acontecendo nos blogs por aí. Isso é gostar de redes sociais ou de informação e é um risco pensar que alguém que domina o uso de algum programa, site ou aplicativo, possa vir a gostar de programar só por isso. Nós não vemos pessoas se tornando chef de cozinha só porque adoram comer spaghetti.

Quem realmente gosta de tecnologia mais do que somente do que ela nos proporciona é aquela pessoa que quer entender **como** aquilo pode existir, como algo acontece por baixo dos panos, e, normalmente, vai ativar sua curiosidade e ir atrás de entender aquilo.

Alguém que domina o uso de um sistema operacional e quer sempre ir mais a fundo em como isso funciona pode tanto acabar na área de programação quanto vir a trabalhar com algo relacionado a sistemas operacionais, como eu comento no próximo capítulo, em que falo sobre as diferentes profissões da área de tecnologia.

O mesmo se aplica para quem adora as redes sociais e outros sistemas que usamos no nosso dia a dia. A pessoa pode vir a trabalhar com redes sociais, mas não diretamente relacionado com programação.

Entretanto, citar estas características para nossa análise (curiosidade, vontade de aprender, gostar de resolver problemas e gostar de tecnologia) pode também ser uma forma de estereotipar ou de padronizar, mas não de limitar. Se você não tiver todas as características citadas aqui, mas tiver

força de vontade, você vai aprender a programar, assim como qualquer outra pessoa.

Essas características podem facilitar a nossa vida, mas nada pode nos impedir de aprender algo que realmente desejamos.

### **3.2 Com quantos anos devemos começar a estudar programação**

Uma questão comum e que sempre é levantada em fóruns de tecnologia é se existe idade para começar a programar. Quando é muito cedo e quando é muito tarde para aprender a escrever códigos de computador?

Encontramos várias notícias de pessoas que começaram a programar enquanto crianças e hoje possuem suas empresas milionárias no ramo de tecnologia ou desenvolveram softwares e revolucionaram tudo. É o caso de Bill Gates, que começou a programar aos 13 anos e aos 17 estava lançando a Microsoft junto a outros jovens programadores, ou de Linus Torvalds, criador do Linux, que começou a se interessar por programação aos 11 anos. São exemplos clássicos quando entramos no debate de idade para começar a programar.

Com casos de sucesso como estes, de pessoas que começaram tão cedo e mudaram a história da humanidade, quem se interessa por programação depois de certa idade fica com muito receio de se envolver na área.

É claro que, quanto mais cedo começarmos, mais fácil será de aprender, pois é natural que tenhamos mais tempo livre, mais energia; mas a idade não nos limita quando se trata de computação.

Temos, hoje em dia, pessoas que estão aprendendo a programar depois dos 60, 70 e 80 anos. Como é o caso de Shirley M. McKerrow, uma empresária aposentada e política que começou a programar depois dos 80 para criar aplicativos e sites.

Com um exemplo como o de Shirley podemos afirmar que **não existe idade máxima para começar**. Mas será que existe idade mínima?

Hoje temos livros como “HTML para bebês”, “JavaScript para bebês” etc. nas prateleiras das bibliotecas e lojas. Isso é bem engraçado, mas ainda não temos certeza se é efetivo, pois pode ser divertido para a criança e depois de alguns anos ela não se interessar pelo assunto, assim como nem toda criança que desenha na infância se torna artista ou nem toda criança que dança na escola se torna dançarina.

Na minha infância eu desenhava muito bem, depois de um tempo eu dançava hip-hop muito bem, durante algum tempo trabalhei como grafiteiro e a minha vida parecia se direcionar para as artes, mas quando conheci tecnologia da informação minha vida inteira mudou e hoje eu só faço alguns rabiscos e mando alguns passos bem toscos de break.

O Code Club, uma iniciativa internacional e voluntária para ensinar programação para crianças, recomenda a idade de 9 a 13 anos para que tenham um melhor aproveitamento do conteúdo que será ministrado em sala de aula e para que as aulas sejam algo realmente efetivo para escolhas futuras das crianças que aprenderam a programar.

Então, eu, pessoalmente, recomendo que se apresente o universo dos códigos para crianças a partir de 9 anos e que não nos limitemos a nossa idade para começarmos a aprender.

### **3.3 Existe idade para procurar emprego na área de programação**

Além da dúvida de quando devemos começar a aprender, ao observarmos o mercado de trabalho vemos muitos jovens nas empresas de desenvolvimento de software e isso pode assustar um pouco.

Podemos encontrar pessoas em nível sênior ou especialista, os níveis “mais altos”, com 50 anos de idade, mas também existem pessoas de 30 na mesma

posição. O mesmo quando procuramos profissionais em nível pleno, o “nível intermediário”, com 30 e outros com 20 na mesma posição. Temos a sequência: a pessoa, normalmente, começa com um estágio, se torna júnior, pleno e depois sênior, daí para a frente pode se tornar especialista ou partir para um cargo de liderança ou gestão. Isso se viermos direto da faculdade para o primeiro emprego. Podemos, claro, cair direto como juniores e ir crescendo também.

Percebemos então como é uma área dinâmica e não existe um tempo predeterminado para que alguém evolua de posição em uma empresa. Encontrarmos pessoas de todas as idades em posições diferentes não deveria ser um limitador para nossa força de vontade, mas um motivador. É sinal de que o mercado tem espaço para todas as pessoas. Só vai depender da nossa capacidade técnica e força de vontade em nos especializarmos cada vez mais.

Não podemos deixar que o medo de arriscar se torne um fator limitante em nossa vida. A área de programação é muito promissora e com um bom planejamento podemos até mesmo mudar de uma área que já atuamos há algum tempo para o desenvolvimento de software sem muito sofrimento.

Tudo, no final, vai depender do nosso planejamento, seja quando jovens ou quando mais experientes.

### **3.4 Sem inglês dá para aprender programação**

Sempre que encontramos guias para programadores(as) iniciantes na internet, o inglês é citado como obrigatório para aprender programação. Eu discordo disso.

Antigamente todo o material didático da área de computação era em inglês. Isso porque o mercado fora do nosso país era muito maior, mas isso mudou de uns 10 anos para cá.

Hoje em dia temos ótimos livros em português ensinando os mais variados conceitos, desde carreira em programação (como este livro), até os

paradigmas mais complicados ou padrões de projetos que antes só encontrávamos na biblioteca das faculdades e, normalmente, em inglês.

Não somente os livros, temos artigos, podcasts, ferramentas e linguagens que podem auxiliar aquela pessoa que possui mais dificuldade com o “idioma universal”.

Para aprender, não precisamos do inglês. Precisaremos do idioma quando vamos trabalhar profissionalmente com programação, pois os códigos devem ser escritos em inglês, documentações deveriam ser escritas em inglês e assim por diante. Mas, percebemos que isso são coisas com que só precisamos nos preocupar quando realmente vamos fazer algo grande ou às quais outras pessoas vão dar manutenção.

Nossos primeiros programas, normalmente, não são nem exibidos para nossos amigos e amigas por vergonha ou algum medo que sentimos.

Existe uma pseudolinguagem de programação chamada Portugol, também conhecida como Português estruturado, e essa linguagem é extremamente utilizada para aprendermos lógica de programação e algoritmos, o que precisamos para fazer programas.

Vamos analisar um exemplo de programa escrito em Portugol para confirmar se conseguimos entender o que está acontecendo:

```
algoritmo "calculadora"
```

```
var
```

```
x, y, saida: real
```

```
operacao: caractere
```

```
inicio
```

```
escreval("Insira um número para x")
```

```
leia(x)
```

```
escreva("Qual a operação?")
```

```
leia(operacao)
```

```
escreval("Insira um número para y")
```

```
leia(y)
```



```

escolha (operacao)
  caso "+"
    saida <- x + y

  caso "-"
    saida <- x - y

  caso "*"
    saida <- x * y

  caso "/"
    se y=0 entao
      escreva("Opa! Precisamos de um número maior que zero em y
para acontecer a divisão de x")
    senao
      saida <- x / y
    fimse
fimescolha

escreval("Resultado", saida)

finalgoritmo

```

Algumas coisas neste exemplo podem confundir, como a palavra-chave `var`, os tipos de dados `real` e `caractere`, porque temos que “ler” algo e porque `x` e `y` são passados para esse tal de `leia()`, mas este é só um exemplo para entendermos que dá sim para aprender programação sem saber nada de inglês.

Depois que aprendermos lógica de programação, então podemos partir para uma linguagem de programação, como JavaScript, Java, Python etc. para enfim criarmos nossos programas mais aplicáveis e será nessa hora que o inglês pode fazer alguma falta. Porém, conseguimos sobreviver isso com alguns conceitos básicos do idioma.

O inglês só fará muita falta na hora em que desejarmos subir mais ainda o nível do nosso conhecimento, pois daí precisaremos acompanhar tanto as

pessoas da nossa nacionalidade quanto as pessoas de fora do país para sabermos o que está acontecendo pelo mundo afora.

Mas cuidado para não cair na zona de conforto e se prender à facilidade de entender Portugal. Aprender nunca é demais. Pode ser legal aproveitarmos que estamos aprendendo programação, e que as linguagens de programação são em inglês, e começarmos a forçar um pouco o idioma.

Tente escrever seus softwares, desde o período de aprendizagem, em inglês. Pode ser uma boa ajuda para dar um *start* no novo idioma. Caso não saiba dizer algo, use o Google Translate ou outra ferramenta de idiomas.

### **3.5 Tem muita fórmula matemática na área de programação**

Antigamente programação era algo muito complexo. As pessoas precisavam fazer muitos cálculos e saber traduzir diversas fórmulas matemáticas para algoritmos computacionais, pois não existia muita coisa pronta.

Quando digo “coisa pronta”, falo sobre termos à nossa disposição funcionalidades das linguagens de programação que nos ajudem a fazer contas ou mesmo que nos ajudem a escrever menos código. Hoje em dia possuímos diversas funções das próprias linguagens que facilitam muito nossa vida.

Aquelas fórmulas que vimos no Ensino Médio e que afugenta grande parte das pessoas não serão utilizadas pela maioria dos programadores e programadoras.

Eu nunca vi alguém da nossa área dizendo que utilizou a fórmula de Bhaskara para subir um e-commerce para produção. Mas, lembre-se: não é porque eu nunca vi, que isso não existe.

Existem pessoas da área que trabalham com sistemas que precisam de muito conhecimento em matemática, mas isso vai depender de onde queremos nos envolver.

Para criar modelos de inteligência artificial será necessário conhecimento em estatística, para comandar um avião a partir de um software será necessário conhecimento específico de aeronáutica e assim vai. Tudo vai depender de onde queremos nos enfiar nesse aglomerado de possibilidades que a programação nos traz.

O conhecimento em matemática pode nos ajudar a sermos profissionais melhores, sim, mas torno a dizer: para **aprender a programar** você não vai precisar de muita matemática.

Eu sempre indico para as pessoas: “Quando tiver um tempo, estude matemática, sim!”. Isso faz diferença na nossa vida, pois matemática é a base de tudo dentro de um computador. Não vai custar muito tempo estudarmos um pouco de matemática computacional ou matemática discreta e valerá muito a pena.

### **3.6 O dia a dia de quem escreve programas: rotina de trabalho**

Outro estereótipo extremamente disseminado é a figura da pessoa programadora do Google. No filme *Os estagiários*, vemos um pouco da realidade da empresa que é sonho de muitas pessoas. Mas, será que toda empresa de tecnologia é igual ao Google com tudo bem colorido, melhores computadores para trabalhar e metodologias de RH superinovadoras?

Quando as pessoas veem algo desse tipo começam a mistificar a área. Acham que uma pessoa desenvolvedora não tem horário a cumprir, não tem carteira assinada firmando compromisso com a empresa, mas é muito pelo contrário.

Temos, sim, carga horária de trabalho, porém as empresas já perceberam que o horário flexível é mais produtivo, então a maioria (não todas) permite que os funcionários entrem e saiam em horários alternativos.

Também temos carteira assinada, como qualquer profissão, a não ser que seja uma pessoa que trabalha no modelo PJ (pessoa jurídica), no qual

teremos um prestador ou prestadora de serviço e um contrato diferente com cada empresa.

E como será que chegam as demandas de serviço para o pessoal de desenvolvimento?

Cada empresa ou até mesmo cada equipe dentro de uma mesma empresa pode ter um modelo de trabalho diferente. Algumas possuem um sistema de chamados, em que alguém cadastra o que precisa ser feito ou corrigido e as pessoas são marcadas para resolver. Outras possuem modelos ágeis de trabalho e utilizam algum software onde listam tudo o que precisa ser feito e a pessoa vai lá procurar uma tarefa.

Uma tarefa pode ser algo como “criar a tela de cadastro de usuários do sistema x”, “adicionar os dados de estorno do cartão de crédito no retorno do sistema”, “criar um relatório xyz” ou “corrigir erro 0x000yz que aparece quando o usuário executa tal função”, “corrigir tela de login que está desalinhada”. Mas tudo isso pode variar muito de empresa para empresa.

Em todo caso, somos pessoas normais, que têm trabalhos comuns e responsabilidades que todo mundo tem. Precisamos cumprir carga horária, levamos atestado médico (quando a empresa solicita), precisamos avisar se estamos doentes, saímos de férias, temos chefes (às vezes legais, às vezes não) etc. assim como qualquer outro trabalho formal.

### 3.7 Conclusão

Vimos até aqui que, para **aprender**, não são necessárias muitas habilidades, nem nada de tão especial como é pintado nos estereótipos, o principal será a nossa força de vontade, um tempo disponível para estudo e, acima de tudo, não desistir de continuar aprendendo.

Não há idade certa, ou requisito obrigatório e não existe habilidade ou elemento especial que somente pessoas programadoras possuem.

É claro, com o tempo precisaremos nos aprofundar em assuntos que não cobrimos no começo dos estudos, como o inglês ou matemática, mas isso não nos impede de começar a aprender.

Também percebemos que trabalhar com tecnologia não significa que seja diferente de tudo e que teremos nossas responsabilidades com as empresas, e as empresas, conosco.

Espero que com essas respostas você esteja se sentindo mais confortável para se envolver com o universo da programação, pois os próximos capítulos servirão para deixá-lo com mais vontade ainda.

## CAPÍTULO 4

# Escolhendo um caminho

A área de programação é recheada de oportunidades. Quando conhecemos as profissões em que podemos nos encaixar, é uma verdadeira enxurrada de caminhos diferentes. São tantas opções que é bem comum nos perdermos entre elas.

Alguns caminhos são totalmente diferentes uns dos outros, pois existem áreas de atuação que são bem peculiares, outros são muito parecidos. Cada direção que podemos tomar em nossa carreira possui uma gama de conhecimentos e habilidades específicas para conseguirmos performar em nossas funções como pessoas desenvolvedoras de software.

Como desenvolvedores(as) poderemos trabalhar com diversas plataformas, aquelas que conhecemos no primeiro capítulo, linguagens de programação e nichos diferentes de sistemas. Sabemos, portanto, que existem vários “tipos” de programadores(as), pois para cada plataforma ou para cada conhecimento específico existe um(a) profissional dedicado(a) à tal. Então, o que cada programador(a) faz?

Por exemplo: quem é o(a) profissional que trabalha com *client-side*? O que essa pessoa faz no dia a dia? Quem é realmente a pessoa que faz aplicativos de celular e o que essa pessoa precisa conhecer? Quem é a pessoa que cria programas para a internet ou aquele jogo maneiro de Nintendo Switch?

Neste capítulo vamos conhecer as profissões mais famosas da área de programação, as que têm mais vagas disponíveis no momento, mas nem de longe são as únicas, que são: front-end, back-end, infraestrutura de plataformas, full-stack, mobile, game developer, cientista de dados e engenharia de dados.

Com o crescimento dos e-commerces, a criação de plataformas sociais, evolução dos navegadores e da própria internet, o mercado para Web cresceu muito e, quanto mais madura fica a plataforma Web, mais oportunidades aparecem. Por isso, as áreas que mais ganharam mercado

foram as voltadas para o ambiente Web e dados (análise de dados e afins). Até mesmo os jogos estão conectados online e precisam de pessoas desenvolvedoras Web. Atualmente, temos jogos conectados entre PC, celular (Android e iOS) e consoles (Xbox, PlayStation, Nintendo Switch), como o game Fortnite.

No dia em que escrevo este capítulo, fiz uma breve pesquisa de vagas relacionadas com Web no Brasil no **indeed.com**, um site de vagas empregos, e nessa busca foram localizados **997** resultados para pessoas da área de front-end.



The image shows a search interface on Indeed. On the left, under the heading "o quê" (what), there is a search box containing the text "front-end". Below this box is the placeholder text "cargo, palavra-chave ou nome da empresa". On the right, under the heading "onde" (where), there is a search box containing the text "Brasil". Below this box is the placeholder text "cidade, estado ou região". At the bottom right of the search area, there is a green button with the text "Vagas 1 a 10 de 997".

Figura 4.1: Indeed para a busca de front-end

**581** resultados para a área de back-end.



The image shows a search interface on Indeed. On the left, under the heading "o quê" (what), there is a search box containing the text "back-end". Below this box is the placeholder text "cargo, palavra-chave ou nome da empresa". On the right, under the heading "onde" (where), there is a search box containing the text "Brasil". Below this box is the placeholder text "cidade, estado ou região". At the bottom right of the search area, there is a green button with the text "Vagas 1 a 10 de 581".

Figura 4.2: Indeed para a busca de back-end

E **896** oportunidades para trabalhar com Android, representando a área de desenvolvimento mobile, que poderia ser Android ou iOS atualmente.

o quê

android

cargo, palavra-chave ou nome da empresa

onde

Brasil

cidade, estado ou região

Vagas 1 a 10 de 896

Figura 4.3: Indeed para a busca de android

Claro que isso pode variar de acordo com o que o mercado de trabalho esteja necessitando e, neste momento, as empresas estão precisando muito dos cargos que vamos conhecer agora!

Como nos últimos anos o mercado para front-end está em constante ascensão, é comum que encontremos mais vagas para essa área. Porém, a área de Web, em geral, tem muita oportunidade já que todo mundo precisa de presença online hoje em dia. Até mesmo Seu João e a Dona Maria da padaria precisam ter seu site para divulgar seu comércio de pães. A área de mobile também tem muita oportunidade, afinal até mesmo a pizzaria do bairro deseja ter um aplicativo para centralizar seus pedidos e fidelizar seus clientes.

Neste capítulo vamos entender mais sobre as profissões e receber uma verdadeira chuva de termos novos da área de desenvolvimento de software, mas não se preocupe, mais à frente no livro eu ensino como pesquisar mais profundamente sobre cada tema citado aqui.

## 4.1 Conhecimento comum entre todas as áreas de desenvolvimento de software

Sempre que pensamos em pessoas que trabalham com código, mesmo com as divisões por especialidades (front, back, full-stack, mobile, jogos, dados etc.), temos alguns conhecimentos técnicos que são “comuns”, são



obrigatórios para todas as pessoas, e extremamente necessários para trabalharmos nessa área.

Esses conhecimentos são importantes para todas as pessoas porque, apesar das peculiaridades das especializações, no final das contas todos fazemos o mesmo: desenvolvemos software. E desenvolvimento de software sempre segue algumas regras e padrões preestabelecidos.

Por agora vamos conhecer os tópicos que são comuns a todas as pessoas que trabalham com programação e um breve resumo do que significam, pois separei um capítulo com dicas e recursos para nos aprofundarmos mais sobre cada assunto mais à frente no livro.

## **Lógica e alguma linguagem de programação**

Lógica de programação é o tópico mais importante de tudo o que precisamos aprender. Lógica é uma maneira de pensar e por isso precisa ser muito praticada, pois é quase uma mudança em como nós fazemos algo, como tomamos decisões. A lógica de programação é a técnica de escrever os programas para que o computador tenha uma lista de tarefas para executar e realizar uma ação. Uma linguagem de programação pode ou não entrar nesse ponto de aprendizagem, pois podemos aprender raciocínio lógico sem depender de uma linguagem, mas aconselho a já estudar lógica com a linguagem que vamos trabalhar. Também é importante conhecer outras linguagens de programação depois de já possuímos alguma experiência para aprendermos também outros **paradigmas de programação** (as maneiras como nós estruturamos o nosso programa para resolver os problemas).

## **Estruturas de dados e algoritmos**

Algumas áreas de desenvolvimento de software cobram mais este conhecimento do que outras, mas estruturas de dados e algoritmos são armas poderosas para alcançarmos o melhor desempenho do nosso programa.

Algoritmos nada mais são do que os passos que criamos para a execução do programa. Estudar como algumas pessoas resolveram os mesmos problemas pelos quais passamos ou como alguns algoritmos famosos (que encontramos em livros acadêmicos) podem nos preparar para enfrentar alguns problemas que vão aparecer no nosso dia a dia.

Estruturas de dados são, muitas vezes, intercaladas com o estudo de algoritmos, pois é o estudo das melhores maneiras de planejar como os dados vão fluir em nosso programa e como são estruturados (armazenados e organizados).

## **Padrões de projeto**

Alguns problemas que enfrentaremos no desenvolvimento de software são recorrentes e existem pessoas que já passaram por eles e catalogaram **boas formas** de resolver estes problemas anos atrás. Esses modelos catalogados foram chamados de padrões de projeto.

## **Versionamento de código**

Quando trabalhamos com escrita de um texto, por exemplo, normalmente, passamos por algumas versões dele. Um exemplo é a escrita de um livro: nós escrevemos um capítulo, então temos uma versão do livro com um capítulo, escrevemos outro capítulo, então temos outra versão do livro com dois capítulos e assim por diante.

Também é comum versionar aqueles trabalhos de escola/universidade:

- trabalho-de-matematica-1.doc, trabalho-de-matematica-2.docx;
- trabalho-de-matematica-agora-vai.doc;
- trabalho-de-matematica-agora-vai-2.docx;
- trabalho-de-matematica-nao-aguento-mais.docx;

Isso é uma garantia de que não percamos a última versão do nosso trabalho e caso tenhamos qualquer problema conseguimos resgatar a versão anterior. Essa ideia serve também para o versionamento de código-fonte. Criamos nosso código, temos a primeira versão, alteramos algo, então temos outra versão e assim vai. E para cuidar disso utilizamos alguns sistemas de

versionamento de código, que são softwares capazes de facilitar a nossa vida na criação de versões do nosso programa (melhor do que ficar criando cópias e colocando nomes com final 1, 2, 3).

## **Qualidade de software**

Nós seguimos padrões e convenções para que nosso código seja legível e de fácil manutenção por outras pessoas que vão trabalhar em nossos programas e para garantir a segurança do código, garantir que um módulo criado agora não atrapalhe outro já existente. Isso é pensar em qualidade. Podemos adicionar testes de software dentro do espectro da qualidade de software, porém não são somente os testes que garantem isso, é o conjunto dos testes com boas práticas e convenções que precisamos conhecer.

## **Protocolo HTTP**

Quando estamos em rede (internet ou rede local), nossos dispositivos (computadores, impressoras, celulares) se comunicam. Para que eles consigam se entender existe um protocolo a seguir para trafegar dados na rede, esse é o protocolo HTTP. Existem outros protocolos de rede, mas o mínimo que todo mundo deve conhecer é o HTTP.

## **Gerenciamento de dependências**

Enquanto trabalhamos com programação, na maioria das vezes, não precisamos escrever todo o nosso programa, algumas partes comuns já existem, como a comunicação com servidores, um modelo para trabalhar com bancos de dados ou coisas do tipo, nós podemos importar esses programas preexistentes para dentro dos nossos sistemas. Isso é chamado de dependência. Esses pacotes de programas externos a nossa equipe recebem o nome de dependência porque, uma vez que nós os adicionamos aos nossos códigos, eles se tornam parte do resultado final e se os removermos, então o programa pode parar de funcionar.

## **Testes de software e testes automatizados**

Enquanto desenvolvemos nosso código precisamos testá-los. Será que isso está funcionando conforme foi planejado? Se eu passar um valor diferente para o meu programa (como colocar um nome no lugar de uma data) ele vai dar erro? Será que o programa aguenta uma quantidade elevada de usuários simultâneos?

Basicamente escrevemos o código e colocamos para rodar em nosso ambiente local e vemos que tudo está OK. Porém, só isso não é garantia para as questões que citei, é necessário conhecer algumas práticas que vão garantir a segurança do nosso código-fonte. São os testes de software, técnicas que utilizamos para garantir a segurança do código, e os testes de software automatizados, que são os testes executados automaticamente por softwares especialistas em testar nosso código-fonte e todo o funcionamento do nosso programa, assegurando algumas partes que nós, como seres humanos, podemos acabar esquecendo de testar.

## **Manutenção de software**

Nós sempre precisamos modificar e continuar cuidando de um software existente. Para que façamos tudo correto, precisamos estudar sobre manutenção de software, que é o conhecimento de como monitorar o programa, como testar o código-fonte e como trocar partes do programa de maneira que ele não pare de funcionar, pois, na grande maioria das vezes, o programa está rodando e rendendo lucro ou auxiliando milhares de pessoas e não podemos deixá-lo sem funcionar para mudar algo e colocá-lo no ar novamente depois.

Existem também alguns conceitos mais opcionais, mas que podem dar um *plus* em nosso conhecimento e produtividade, que são:

## **O próprio sistema operacional**

Conhecer muito bem o nosso sistema operacional vai nos trazer certo desempenho no trabalho. Se nós conhecermos os atalhos para executar uma funcionalidade ou os comandos para executarmos no terminal que automatizam o que fazemos no sistema operacional (como configurar

coisas, rodar programas etc.) conseguiremos mais produtividade no dia a dia.

## **Terminal**

Sabe aquela tela preta que aparece nos filmes de *hackeragem* (filmes de hackers)? Então, aquilo é o terminal. O terminal é o programa que utilizamos para rodar alguns comandos em nosso sistema operacional. Utilizamos isso porque, muitas vezes, é mais rápido rodar diversos comandos de uma só vez do que abrir uma janela e clicar em diversos botões para executar a mesma tarefa. Precisamos conhecer bem essa ferramenta porque existem sistemas operacionais que não tem a interface gráfica (aquela parte visual que enxergamos no PC), principalmente servidores em que temos somente a tela preta do terminal e os comandos a serem executados.

## **Diferentes ambientes de execução: local, produção e testes**

Podemos resumir ambientes como: o local onde nosso programa está sendo executado, que pode ser em nossa máquina, no servidor ou na máquina/celular/etc. do nosso usuário final. Enquanto criamos o software em nossa máquina, chamamos de ambiente de desenvolvimento local. O ambiente onde o usuário final utiliza é o ambiente de produção. E temos um ambiente que deve ser o máximo possível igual ao de produção, para nós, programadores(as), testarmos, chamado ambiente de testes. Ao menos devemos saber que existem o ambiente local e existe o ambiente de produção.

Estes são os conhecimentos comuns entre todas as áreas de desenvolvimento de software. Vamos agora conhecer as áreas de atuação e suas necessidades específicas.

## **4.2 Front-end**

Quando dividimos os sistemas Web, no primeiro capítulo, conhecemos o modelo de funcionamento de sistemas, em que temos o front-end ou client-side, que é o que vemos no navegador, e o back-end ou server-side, que é o programa rodando no servidor, mas não foi informado, até agora, o que uma pessoa que trabalha com front-end faz no seu dia a dia, certo?

Uma pessoa desenvolvedora front-end é responsável por programar e fazer manutenção de componentes visuais da aplicação Web, assim como encaixá-los na tela e fazer com que esses componentes funcionem com integração com o servidor. Tudo o que vemos no navegador deve ter sido escrito por uma pessoa de front-end. Entendemos como componentes os itens que aparecem na interface como os botões, as caixas de texto, caixas de diálogo, alertas. Enfim, tudo o que está na tela.

No seu dia a dia, a pessoa desenvolvedora front-end recebe interfaces/páginas ou componentes que alguém responsável pela área de design do produto ou da empresa criou e reproduz isso em código-fonte que vai rodar no navegador.

Fora transformar a arte desenhada em código para o navegador apresentar na tela, também faz a integração com o back-end, animações, interoperabilidade entre navegadores (pois entre Internet Explorer, Google Chrome e Firefox pode acontecer de um mesmo código funcionar de maneira diferente, ainda mais quando os usuários utilizam versões antigas dos navegadores), cuida da acessibilidade de um site, que é deixar a interface usável por pessoas com deficiência (visual, motora, cognitiva etc.), fora o trabalho de estruturar o código de um modo que os *search engines* (motores de busca, como o Google, Bing ou DuckDuckGo) possam encontrar nosso site.

Essa pessoa também será responsável por desenvolver uma interface que funcione em um navegador de celular, tablet ou desktop - chamamos isso de responsividade -, e programar testes para que tenhamos garantia do correto funcionamento da interface.

Alguns conhecimentos técnicos que um(a) desenvolvedor(a) front-end precisa possuir:

**HTML:** é a linguagem de marcação, a linguagem que estrutura o conteúdo da nossa página Web de um modo que o navegador e outros leitores de código entendam o que é cada frase ou item na tela (se é um título, um parágrafo, uma seção da página, uma imagem etc.).

**CSS:** é a linguagem que dita o estilo visual da nossa página (as cores, os formatos, o layout, as animações). Utilizando CSS nós transformamos o HTML que o navegador está mostrando na tela de uma coisa sem vida para o desenho que a pessoa de design criou.

**JavaScript:** é uma linguagem de programação que traz interatividade para o client-side. Faz a comunicação com servidores e executa ações que o usuário ativa na página.

**Node.js:** além da linguagem JavaScript, hoje em dia pessoas desenvolvedoras front-end precisam conhecer Node.js, uma plataforma de desenvolvimento de softwares que executa JavaScript, pois utilizamos essa tecnologia para muitas coisas no nosso dia a dia, como automatizar tarefas repetitivas de front-end ou mesmo servir nossa aplicação client-side para o usuário final.

**Performance Web:** a performance (ou desempenho) de uma aplicação Web é a velocidade com que uma página carrega, a quantidade de dados que essa página baixa para a máquina do usuário, dentre outras preocupações. Performance é uma responsabilidade imensa de uma pessoa de front-end, pois o client-side é responsável por 80 a 90% da velocidade de carregamento de um site e ninguém quer permanecer em um site que fica travando ou demora muito para carregar, muito menos um site que consome toda a banda que contratamos com nossa operadora de celular.

**Semântica:** a linguagem HTML possui elementos específicos para cada item que está aparecendo na tela. Se é um botão, existe o elemento certo para isso, se é um parágrafo, também. Semântica, em front-end, é o uso correto da linguagem HTML para marcação do conteúdo. Isso é muito importante, pois, junto com técnicas de SEO, melhora o conteúdo para os buscadores localizarem nosso site e também ajuda na acessibilidade, pois os leitores de tela, softwares que leem a tela para usuários com deficiência

visual, encontram e entendem mais facilmente o conteúdo e componentes na aplicação.

**SEO:** o marketing sempre foi algo importante para qualquer empresa. SEO, hoje, faz parte do marketing da maioria dos produtos digitais, pois o nosso site precisa aparecer nas primeiras páginas do buscador se não a empresa não tem vendas, o site não tem acessos, o conteúdo não é encontrado. SEO (*Search Engine Optimization*) é otimizar o nosso site para os search engines os encontrarem mais facilmente baseado em otimizações que esses mesmos sites nos orientam a fazer em nossas páginas.

**Acessibilidade:** é tornar a interface usável por pessoas com deficiência visual, motora, cognitiva ou outro tipo. A internet é algo que deve ser de livre acesso a todas as pessoas, inclusive pessoas com alguma deficiência. É nossa responsabilidade criar interfaces e sites que essas pessoas consigam utilizar.

**Testes de interface:** trata-se de técnicas para confirmar se nada parou de funcionar depois que mexemos em alguma parte da tela/página/interface ou em algum código front-end, assim como os testes de software e testes automatizados, que comentei anteriormente.

**User experience (UX):** como diz a tradução, é a experiência do usuário, é cuidar para que a interface tenha uma experiência agradável de navegação e utilização, junto com as pessoas do design do produto/site. Uma pessoa de front-end não precisa dominar UX design, mas ter um conhecimento básico vai ajudar a garantir a qualidade da tela e a auxiliar a equipe de design na criação de um produto de muita qualidade para o usuário final.

**Integração com back-end:** guardamos informações importantes no back-end da nossa aplicação para manter a segurança dos dados e também para não encher a máquina do nosso usuário de dados indesejados por ele, por isso é importante que a pessoa de front-end domine as técnicas, protocolos e modos de guardar dados no back-end.

## 4.3 Back-end



O conhecimento de uma pessoa que trabalha com desenvolvimento back-end é voltado para aquilo que nós “não enxergamos” nos sistemas Web (ou aplicações com integração com a internet) mas que estão ali!

Essa é a pessoa responsável por criar a parte do sistema que armazena os dados no banco de dados, que interage com outros sistemas e processa dados para transformar em informação e devolver resultados otimizados ou organizados para o client-side. É no back-end que fica toda a regra de negócio de um produto/site/aplicativo. É onde fica o modo como um produto funciona.

Enquanto no front-end temos as regras de visualização (o que deve aparecer na tela em determinada interação do usuário), no back-end temos os modelos de negócio de uma empresa, como taxas, valores, recomendações para o usuário etc. São coisas do núcleo dos sistemas e que devem ficar bem seguras guardadas em nossos servidores.

Regras de negócio no client-side poderiam ser facilmente encontradas, pois bastaria pegar o código-fonte, que já está carregado no nosso navegador, e utilizar alguma técnica de análise de códigos para exibir o seu conteúdo e entendermos como tudo funciona no software. Isso só não acontece com certa frequência porque as regras mais importantes estão protegidas no back-end de nossas aplicações e a dificuldade de invadir um servidor é muito maior.

Como, normalmente, são tratadas grandes quantidades de dados, é importante que uma pessoa de back-end possua um bom conhecimento em estruturas de dados, algoritmos de busca e ordenação, que são os algoritmos utilizados para organizar e encontrar dados nas estruturas de dados, para criar rotinas performáticas, códigos que respondem rápido às ações de nossos usuários dos softwares, e devolver uma resposta otimizada para quando o front-end solicitar informações.

Quando digo devolver a resposta otimizada é porque, durante a comunicação entre front-end e back-end, serão trocadas mensagens (com o uso do protocolo HTTP, por exemplo). O sistema buscará dados no banco e

devolverá esses dados para o front-end exibir na tela. Esse trâmite de pegar os dados do banco é um dos mais custosos para uma aplicação.

Alguns conhecimentos técnicos que uma pessoa desenvolvedora back-end precisa possuir são:

**Uma linguagem de programação para back-end:** pode ser uma entre as várias que são utilizadas para criar sistemas back-end, como Java, PHP, JavaScript, Golang, Ruby, C#. As linguagens possuem suas diferenças e a escolha pode influenciar muito em nossos sistemas. Abordarei melhor a escolha da linguagem de programação no próximo capítulo.

**Sistemas operacionais:** a pessoa de back-end precisa entender o que acontece “por baixo dos panos”, conhecer como os sistemas operacionais funcionam a fundo, não somente utilizá-los. Será necessário profundo estudo sobre processos, memória, armazenamento de dados etc.

**Bancos de dados:** bancos de dados são coleções de dados organizados de maneira organizada para que possamos consultá-los, alterá-los ou deletá-los com certa segurança. Será necessário conhecer sobre isso, pois é onde serão salvos os dados tratados pela aplicação. Também é importante aprender a linguagem de consulta SQL, assim como o modelo não relacional de armazenamento de dados, um modo diferente de guardar dados que ganhou força graças a sua facilidade de desenvolvimento, performance e escalabilidade, o NoSQL.

**Arquitetura de comunicação Web:** precisaremos conhecer modelos de comunicação entre aplicações (front-end com back-end ou mesmo back-end com back-end), como o REST ou SOAP que são arquiteturas de comunicação Web, modos de se trafegar os dados entre aplicações, pois nós trocamos dados entre sistemas o tempo todo em nossos sistemas mais complexos.

**Servidores Web:** são os programas que fazem a ponte de comunicação entre nosso back-end e o front-end da aplicação, eles são instalados no sistema operacional e fazem o papel de enviar o que foi pedido pelo client-side para a aplicação server-side por meio de protocolos de rede.

**Protocolos de comunicação em rede:** além do HTTP, existem diversos outros protocolos de comunicação em rede e o back-end de nossa aplicação tem muita responsabilidade nisso, pois pode precisar enviar arquivos, e-mail ou outras mensagens de outras maneiras por meio de protocolos específicos, como o SMTP ou FTP, que são os protocolos de comunicação via e-mail e transferência de arquivos respectivamente.

## 4.4 Infraestrutura (DevOps)

Temos, em Tecnologia da Informação, alguns tipos de infraestrutura, como infraestrutura de TI, que engloba a rede física de uma empresa (os computadores, os sistemas e softwares instalados nessas máquinas, os equipamentos e cabos de rede), e a infraestrutura de aplicações, que evoluiu muito nos últimos tempos com o avanço da computação em nuvem e provê o sistema operacional, hardware e softwares para rodarmos nossa aplicação em instâncias provisionadas por alguma empresa externa, como a Amazon AWS ou Microsoft com o serviço Azure.

Quando digo provisionamento, estou falando que uma empresa nos entrega alguns serviços prontos. Antigamente teríamos que ter uma máquina ou várias máquinas, o servidor, fisicamente em nossa empresa. Teríamos que instalar o sistema operacional, configurar os serviços, programas e instalar os servidores Web/bancos de dados/nossa aplicação, configurar a rede para comunicação externa a empresa. Hoje em dia não precisamos mais disso, basta contratarmos esse serviço com alguma empresa especializada e pronto! Já temos tudo pronto, com toda a segurança já planejada e a infraestrutura de TI em responsabilidade da provedora. Nós só precisaremos adaptar nossos sistemas para que trabalhem com esse modelo.

Durante a evolução do desenvolvimento e da tecnologia, apareceu uma nova palavra para nosso dicionário, a palavra **DevOps**. DevOps não é uma profissão, mas é bem comum encontrarmos vagas com esse nome na descrição, até por isso eu coloquei esse nome no subtítulo desta seção. DevOps é, na realidade, um movimento/uma cultura que nasceu para deixar o desenvolvimento de software mais ágil juntando desenvolvimento

(*development*) com operações (*operations*, que é a parte de cuidar da infraestrutura das aplicações), pois antigamente eram coisas separadas: uma pessoa desenvolvia, enviava um pacote para a pessoa de infra e essa seria responsável por configurar o servidor, instalar a versão do sistema etc., todo aquele passo que comentei anteriormente.

Com a filosofia DevOps o time deve ser responsável pela infraestrutura da aplicação, mas em times onde temos alguém responsável pela infra, essa pessoa deverá saber muito sobre:

**Automação de processos:** automação faz parte da agilidade que a cultura DevOps provê, pois devemos automatizar a entrega do software (enviar o nosso código para produção, deixar o sistema disponível para o usuário final), os processos que são executados antes de criar o pacote que será enviado para produção e todo o resto que for possível automatizar.

**Cloud services:** serviços de Cloud (computação em nuvem, que é a utilização de recursos como memória, armazenamento, processamento externos, computadores conectados a internet que nos fornecem sua capacidade para utilizarmos em nossas aplicações), são os serviços disponibilizados pelas empresas que atuam provisionando a infraestrutura para nossas aplicações, como a Amazon AWS, Microsoft Azure, Google Cloud, Heroku, IBM Cloud, etc. A pessoa deve conhecer muito bem esses provedores e os seus serviços para buscar o produto correto para cada necessidade de nossa aplicação, provendo economia e segurança ao nosso sistema.

**Sistemas operacionais:** essa pessoa será responsável por configurar todo o sistema para funcionar na nuvem com seus pacotes planejados para não deixar a aplicação lenta, segurança e tudo para que ele não pare de funcionar. Principalmente conhecimento em Linux, que é o sistema operacional mais utilizado em servidores.

**Redes (configuração, funcionamento etc.):** redes de computadores é um conjunto de máquinas, computadores ou não, conectados através de meios de comunicação, como cabos, wireless (Wi-Fi) ou similares, essa pessoa

possui essa atribuição, pois configurará serviços que se comunicam em rede entre os sistemas e aplicações conectadas na nuvem.

**Segurança da informação:** segurança é parte do processo de configuração de ambientes, redes, bancos de dados etc. e todo um conjunto de outras funcionalidades dos nossos sistemas e aplicações, porém a pessoa de infraestrutura consegue aumentar a segurança configurando muito bem um sistema operacional e monitorando nossas aplicações contra ataques ou tentativas de invasão.

**Programação:** também é necessário saber programar, afinal essa pessoa desenvolve e cuida da infraestrutura e a automação de processos é feita por meio de programação.

**Integração contínua:** ferramentas de integração contínua nos auxiliam a integrar (juntar) o nosso código à base central de códigos para garantir que o código adicionado não vá “quebrar” a aplicação depois de unido com o código que já está em produção.

**Deploy contínuo:** ferramentas de deploy contínuo são utilizadas para automatizar o deploy (disponibilizar nosso programa para nossos usuários, seja um site ou aplicativo de celular), depois de certas regras (que nós configuramos) forem satisfatórias, então a ferramenta trata de colocar o código em produção para a equipe.

**Configuração de ambientes:** o ambiente, como já comentado, é onde nosso programa está rodando. Configuração de ambientes (local, de produção ou de testes) tem diferenças consideráveis de configuração e uso, pois o ambiente de produção não pode apresentar problemas, se não nosso usuário ficará frustrado. O ambiente de testes deve ser o mais similar possível ao de produção, porém com algumas peculiaridades, como um hardware com menos recursos (menos memória, um processador mais fraco), pois tem menos acesso que o de produção. O ambiente local é nosso ambiente pessoal (nosso computador) e a pessoa responsável pela infraestrutura deve garantir que tudo esteja funcionando corretamente e da melhor maneira possível.

**Monitoramento de aplicações:** é uma parte importante do trabalho de infraestrutura, afinal essa pessoa quem cuida dos ambientes, ela consegue aumentar a quantidade de memória ou processador, se for preciso, e existem diversas ferramentas e modos de se monitorar uma aplicação em produção ou testes, a pessoa de infra deve dominar elas.

## 4.5 Full-stack

Alguém desempenhando o papel de full-stack consegue executar as funções de várias outras áreas, como front-end e back-end e infra de uma só vez. Ou seja, a pessoa full-stack é um(a) profissional verdadeiramente multidisciplinar.

Enquanto front-end e back-end buscam se especializar em somente uma função ou área de trabalho, apesar de ter conhecimento bem superficial das demais, a pessoa desenvolvedora full-stack busca saber sobre tudo o que envolve desenvolvimento de software, apesar de continuar sendo especialista em alguma área.

Essa especialização em alguma área acontece naturalmente, pois são muitos tópicos para dominar e seria quase humanamente impossível alcançar excelência em todas as áreas de uma só vez. Como as pessoas têm afinidade ou gostam mais de uma função, elas acabam se especializando nessa função, porém, como full-stack, conhece o necessário para conseguir atuar em todas as outras.

Em todo caso, full-stack é alguém que sabe criar um sistema inteiro, do zero e até sozinho. Não que as outras pessoas também não saibam, mas quem se responsabiliza em ser full-stack em uma empresa decidiu que em seu dia a dia não teria uma especialização, mas resolveria qualquer problema que aparece. Ela vai tratar de modelar o banco de dados, configurar e provisionar o servidor, criar a aplicação server e client-side, enfim, tudo o que precisamos em sistemas Web ou mesmo em outras áreas, como mobile.

Como é de imaginar, uma pessoa desenvolvedora full-stack precisa ter o conhecimento técnico de infraestrutura, front-end, back-end e o que mais ela sentir necessário aprender. Se ela quiser ser full-stack em mobile, por exemplo, precisará saber tudo sobre infraestrutura, back-end e mobile. Se quiser atuar em games, infraestrutura, back-end e games... E assim por diante.

Muita gente acha que full-stack é uma evolução natural de outras áreas. Acreditam que começamos como front-end, back-end ou outra especialização e vamos nos tornando mais generalistas com o tempo, porém isso não é uma verdade. As pessoas podem trabalhar a vida inteira como front-end, back-end, mobile etc. e nunca querer se envolver com outras partes do sistema (apesar de saber como o sistema funciona).

Nós também não começamos em uma área e vamos evoluindo de nível profissional, como front-end júnior, front-end pleno, front-end sênior e finalmente nos tornamos full-stack. Isso não é uma realidade. A pessoa pode começar, desde seu primeiro dia de trabalho, já atuando como full-stack, pois isso não é um nível de conhecimento, como iniciante, intermediário, avançado, mas uma área de atuação.

Outro erro é achar que uma pessoa full-stack é a mais sabichona de todas as áreas. Como eu disse anteriormente, a pessoa desenvolvedora full-stack pode possuir conhecimento generalista e dominar um assunto, mas uma especialista pode saber mais que ela de outra área que ela não domina tanto. Um exemplo: alguém que domina muito back-end, como full-stack, pode saber menos de mobile do que alguém com especialização em mobile.

## **4.6 Mobile Android e iOS**

Mobile, como já aprendemos anteriormente, engloba os celulares, smartphones e tablets. A pessoa que atua nessa área será responsável por desenvolver e manter aplicativos para dispositivos mobile.

Essa pessoa precisará conhecer o sistema operacional com o qual vai trabalhar no dispositivo, seja Android ou iOS (iPhone). Vale lembrar que os sistemas operacionais de celulares, assim como de computadores, controlam todo o nosso aparelho. São responsáveis por desde o funcionamento de um botão até os aplicativos que estão instalados.

Quando compramos um smartphone, hoje em dia, temos a escolha entre Android ou iOS, Android para a maioria dos aparelhos, como Galaxy, ZenFone, Moto E/G/X/Z, Xperia e iOS para os aparelhos da Apple, iPhones e iPads.

Não é recomendável instalar um sistema Android em um aparelho da Apple, por exemplo, apesar de isso ser possível, pelas integrações do sistema operacional com o hardware. Os aparelhos com Android possuem hardware compatível com Android e os Apple com o iOS.

Existem diferenças significativas de versão entre sistemas antigos (versão 1, 2, 3 etc.) e versões mais atuais dos sistemas e isso aumenta a importância de conhecer muito bem os sistemas operacionais e acompanhar sua evolução. Além disso, as empresas podem alterar o Android que vai em seus dispositivos para que possua características específicas para os seus aparelhos. Também enfrentamos problemas de hardware do aparelho, como espaço de armazenamento interno, memória, cartão de memória ou até mesmo rede quando trabalhamos com mobile.

Hoje em dia os aplicativos fazem tanta coisa e são tão complexos que podemos ter muito mais que alguns poucos profissionais cuidando de um só aplicativo. Isso tanto em uma empresa de tecnologia quanto em uma empresa de outro setor que possua o aplicativo como fonte de renda. As possibilidades são imensas.

Mas, aqui, temos uma divisão entre Android e iOS, isso acontece porque cada sistema possui todo um ecossistema diferente um do outro para a pessoa aprender, como Java e Kotlin, linguagens de programação para Android, e Objective-C e Swift, as linguagens de programação para iOS. Seria muita coisa para uma pessoa sozinha dominar, por isso é um mercado dividido entre essas duas frentes.



Somente para conhecimento, existem outros sistemas operacionais de celular, porém os mais "importantes" para quem quer entrar na área de mobile são Android e iOS atualmente.

Uma pessoa que trabalha com mobile precisará conhecer:

**Uma linguagem de programação:** a escolha da linguagem de programação vai variar de acordo com o ambiente (sistema operacional e aparelhos) escolhido para desenvolver. Se quisermos programar para Android vamos precisar aprender Java e Kotlin, se for para iOS, então é Objective-C e Swift. Antigamente se utilizava uma linguagem para o desenvolvimento, porém foram criadas duas linguagens novas e com mais poderes para facilitar o desenvolvimento mobile, Kotlin e Swift, e alguns módulos ainda precisam ser mantidos com as linguagens "antigas" por motivos de versão ou dificuldade de manutenção/reescrita.

**Sistemas operacionais mobile:** é necessário dominar os sistemas operacionais para contornar ou corrigir os problemas de versão (uma versão antiga de Android/iOS pode não aceitar alguma coisa que utilizamos e então temos que buscar alternativas para isso), conhecer os recursos que o sistema nos disponibiliza, entender como o usuário poderá usá-lo. Assim como cada sistema operacional tem suas peculiaridades, também existe a maneira certa para se trabalhar com eles, de criar um item na tela, de fazer animações, de tratar o ciclo de vida do aplicativo (desde quando abrimos um aplicativo, fazemos algo nele, colocamos em suspensão ao trocar de aplicativo ou fechamos algum App) e outras minúcias de cada sistema.

**Tamanhos de telas:** os tamanhos de tela também variam muito e é por isso que é importante conhecê-los e saber adaptar nosso aplicativo. Existem técnicas de desenvolvimento para que os aplicativos se ajustem para as telas e será necessário conhecer isso.

**Processos de deploy (publicação):** o processo de deploy de aplicativos também é diferente dos demais, pois eles são enviados para as lojas de aplicativos, por isso é necessário que a pessoa desenvolvedora mobile conheça suas especificações e limitações.

**Integração e sincronização com back-end:** algumas informações importantes estão na internet, como dados dos usuários, preços de produtos etc., estão no back-end da nossa aplicação, então é necessário que a pessoa consiga solicitar esses dados utilizando troca de mensagens nos protocolos de comunicação da Web.

**Recursos dos dispositivos mobile (hardware):** itens como GPS, Bluetooth, Tela, são os recursos do aparelho que a pessoa deve dominar. Um aplicativo pode precisar da localização de um usuário para executar determinada ação, ou pode precisar piscar a tela, utilizar a câmera ou algum outro recurso de hardware, por exemplo.

## 4.7 Desenvolvimento de jogos

Uma boa quantidade de gente conhece a área de programação graças aos games. As pessoas querem criar um jogo ou mesmo utilizam programas que "facilitam" os games por meio de scripts (programas interpretados pelos "facilitadores"), os famosos *bots* ou *hacks*, que são meios para roubar nos jogos.

Isso é uma das partes legais de se trabalhar com programação: podemos trabalhar com algo que gostamos muito ou que utilizamos o tempo todo no dia a dia. As pessoas que se divertem bastante com jogos com certeza vão se divertir criando jogos.

Alguém responsável pela programação de um game vai criar a física do jogo, o motor de renderização (o código que faz as coisas aparecerem na tela), implementar a mecânica dos jogos, movimentação de personagens, colisão de objetos e mais algumas funções.

É um mercado não muito atrativo no Brasil, porém sempre é possível conseguir uma oportunidade para um emprego remoto, para mudar de país ou mesmo conseguir trabalhar como freelancer. Isso acontece porque as maiores empresas de desenvolvimento de jogos não estão no Brasil (não possuem setor de desenvolvimento por aqui), infelizmente.

Mas isso não significa que a profissão está escassa. O mercado de games como um todo e desenvolvimento de jogos no Brasil é um dos que mais crescem no mundo! As pessoas estão dispostas a comprar os jogos e compram bastante. Nesse cenário de muita venda de games, é de se imaginar que o mercado de criação de jogos também está crescendo no país e logo teremos mais vagas na área.

Não temos muita coisa diferente de outras áreas de desenvolvimento, além de conhecimentos específicas de games como é o caso de motores de renderização, física, roteiro, colisão de objeto etc. e detalhes de implementação como som e design.

Alguns dos conhecimentos necessários para trabalhar com games são:

**Linguagens de programação:** C, C++, Java e linguagens Web, como JavaScript, são algumas das linguagens que a pessoa desenvolvedora de games deverá dominar - não todas! Afinal nós não somos dicionários ambulantes. Pois a maioria dos jogos exige muitos recursos das plataformas onde serão executados, seja no celular, no computador, em um videogame e, hoje, temos também jogos sendo escritos com JavaScript rodando no navegador. Algumas pessoas migram de área durante sua carreira, então podem ir de uma linguagem como Python para Java ou C/C++ para desenvolver games.

**Performance de software:** o motivo de pessoas desenvolvedoras de games precisarem aprender linguagens como C e C++ é por sua performance extremamente poderosa. São linguagens que geram um software muito performático, mas somente as utilizar não garantirá um bom desempenho do game. Alguém de games precisará conhecer muito sobre performance de software para garantir que os jogos não travem, apresentem lentidão ou algo do tipo.

**Plataformas de jogos:** é necessário conhecer muito bem sobre a plataforma a qual irá rodar o game, pois muda bastante programar para computador e para consoles, como PlayStation, Xbox etc.

**Mecânica dos jogos:** é um dos assuntos mais importantes quando se fala de desenvolvimento de games. A mecânica dos jogos é o tratamento do

funcionamento do game. Movimento de personagens/itens/objetos, o que acontece no jogo quando fazemos algo, a dinâmica, isso é a tal mecânica quando dizemos mecânica de jogos.

**Engines de games:** as engines, motores de games, são ferramentas que facilitam o desenvolvimento dos jogos. A parte da física do jogo pode ser complicada e repetitiva de se fazer quando vamos criar um jogo do zero e essas ferramentas já trazem recursos prontos para utilizarmos em nossos jogos. Algumas engines conhecidas são: Unreal Engine, Unity e CryEngine.

Mas aí nos questionamos: sempre que ouvimos falar de games aparecem os termos jogabilidade, gamificação, personagens, roteiro etc. nós não mexemos com isso?

Esses tópicos são parte da experiência do usuário, mas são criados com base em pesquisa e criação de uma pessoa de design, não da pessoa desenvolvedora de games. Quem programa se encarrega de aplicar o que lhe foi passado pelo setor de design para dentro do jogo, igual acontece com front-end e mobile devs.

## 4.8 Área de Big Data

No mundo moderno possuímos um novo termo na nossa galáxia de letrinhas, o Big Data. Isso é: uma grande quantidade de dados gerados todos os dias por dispositivos de todos os tipos como mobile, carros, geladeiras, televisores, computadores. Tudo o que está conectado na internet pode estar gerando dados e alimentando uma base de dados que será utilizada depois para gerar informação útil.

Para que isso seja possível, é necessário ter uma plataforma e aplicações para coleta de dados muito bem planejadas e essa massa de dados precisa ser tratada e disponibilizada de modo que possa ser utilizada como informação pelas pessoas da área de negócios das empresas ou mesmo para nós, usuários finais. Os responsáveis por essa área são os profissionais de dados, como cientistas e engenheiros(as) de dados.

Entre essas duas profissões temos alguns pontos em comum, assim como os pontos que qualquer pessoa que trabalha com programação possui, que são:

**Linguagens de programação:** como toda profissão envolvida com programação, será necessário dominar uma ou mais linguagens de programação, sendo a mais utilizada atualmente a linguagem Python, mas também pode ser necessário conhecer Java, Julia e existe uma linguagem especial para tratamento de dados e estatística, que é a linguagem R.

**Matemática e estatística:** a matemática fará parte do dia a dia e por isso é necessário profundo conhecimento na arte dos números, principalmente em estatística.

**Bancos de dados:** bancos e linguagem de consultas SQL e NoSQL também serão extremamente necessários, afinal a pessoa vai trabalhar com dados o tempo todo, precisa conhecer a melhor ferramenta e melhor modo de modelar os dados para serem armazenados.

**Ferramentas específicas para Big Data:** existem ferramentas específicas para trabalhar com Big Data, como Hadoop e Spark e será necessário dominá-las.

Vamos conhecer melhor cada uma das duas profissões, além dos tópicos citados.

## **Ciência de dados**

Cientista de dados é alguém responsável por tratar os dados para transformar em informação utilizável pela organização ou pelo usuário final. Essa pessoa vai criar algoritmos para extrair informação da massa de dados.

Os dados extraídos não são suficientes para todo mundo utilizar. Tudo pode ser um dado. Um número solto na mesa, como o número 42, é um dado, mas quando relacionamos o número com uma pessoa e dizemos que isso é uma idade, então o dado passa a ter contexto/sentido, por isso se torna informação.

É por isso que cientistas de dados são muito importantes no cenário atual. A capacidade analítica e conhecimento em estatística de um(a) cientista de dados consegue transformar todo o rumo de um produto ou empresa.

## **Engenharia de dados**

A pessoa que trabalha na área de engenharia de dados é responsável por provisionar a infraestrutura necessária para o trabalho do(a) cientista de dados. É quem planeja como os dados são armazenados, escala os sistemas (faz o sistema suportar maiores quantidades de dados/tráfego/acesso).

Como a pessoa de engenharia de dados é mais focada em sistemas, a grande diferença em conhecimento é a habilidade de provisionar aplicações, conhecer mais sobre infraestrutura e programação de aplicações no geral. Seu maior foco é em Big Data, *machine learning*, bancos de dados e programação, não tanto em matemática e estatística (parte analítica).

Os conhecimentos de uma pessoa engenheira de dados é parecido com a de ciência de dados, com a diferença de que uma trabalha mais com sistemas, o(a) engenheiro(a), e a outra trabalha mais com estatística e visualização dos dados, o(a) cientista.

## **4.9 O que levar em consideração na hora de escolher a profissão**

Muitos detalhes precisam ser levados em consideração na hora de escolher a profissão na qual vamos trabalhar. Conhecemos aqui as características de cada profissão justamente para que seja analisado se existe compatibilidade e interesse de nossa parte no que é feito por cada profissional com as coisas que gostamos. Esse é um dos pontos: afinidade com a profissão.

Precisamos gostar do que é feito para poder escolher a área. Não vai ser muito legal se formos alguém que ama muito de back-end, mas que trabalha no dia a dia com front-end e vice-versa. São muitas peculiaridades das áreas

que podem nos frustrar profissionalmente se não estivermos trabalhando com o que realmente gostamos.

Fora isso, pode acontecer de escolhermos trabalhar com algo que não tem vagas em nossa região e nessa hora precisamos ter planejamento ou conformidade, infelizmente. Porém, se planejarmos bem, podemos entrar em uma área de desenvolvimento somente para ganhar experiência, para ter convívio com código, ver se é realmente o que gostamos, e posteriormente partir para outra região onde tem oportunidades com o que realmente gostamos de trabalhar. Então esse é outro ponto: mercado de trabalho, vagas.

Mas trocar de cidade/estado não é privilégio para muitas pessoas, por isso existe outra possibilidade: emprego remoto. Caso não tenhamos oportunidade em nossa região, podemos, ainda, tentar o emprego remoto como alternativa para fazer o que gostamos. Só não podemos desistir de trabalhar com algo que realmente nos empolga!

Outro ponto muito importante em nossa escolha: as comunidades de tecnologia.

Comunidades são grupos que se reúnem de alguma maneira, seja pessoalmente ou em fóruns, para discutir tecnologia e para compartilhar conhecimento. É a comunidade que gera conteúdo para pessoas iniciantes (ou já experientes), é o grupo ou pessoa que abraça a quem está começando e diz: “Venha por aqui... Nós o ajudaremos”. Analisar se a comunidade em torno do que desejamos fazer é acolhedora pode ser bom antes de tomarmos uma decisão de carreira, pois o apoio de outras pessoas faz muita diferença em nossa vida profissional. O crescimento é muito mais rápido quando temos ajuda.

Então temos os seguintes pontos a considerar na hora de escolher a profissão:

- compatibilidade/afinidade com a profissão
- região (cidade/estado) onde moramos/quantidade de vagas
- possibilidade para mudar de cidade caso necessário
- possibilidade de emprego remoto caso necessário

- comunidade e quantidade de conteúdo disponível para estudo

## 4.10 Como testar cada área antes de escolher uma profissão

Apesar de agora sabermos o que cada profissional faz e quais são os conhecimentos necessários para cada pessoa, ainda ficam dúvidas sobre o que escolher, pois não temos como realmente saber se gostamos de algo sem provar. Certo?

A única maneira de testar se vamos gostar de uma profissão é realmente nos envolvendo com ela. Devemos procurar pessoas que trabalham na área e lhes questionar sobre seu dia a dia, devemos começar a estudar o assunto, podemos começar a participar de projetos que precisam de gente para ajudar em algo que desejamos testar (o software open source pode ser um excelente local para testar se iremos gostar de uma profissão), precisamos participar de eventos da área. Ir a eventos e participar de comunidades já vão nos mostrar muito sobre a profissão que estamos pensando em nos envolver, então é importante tentar participar.

Existem também competições de programação específicas por especialidade. Existem os *hackathons*, que são campeonatos de um dia e muitas vezes de até mais dias. Podemos cair de cabeça nesses eventos e sentir na pele como é criar um projeto do zero.

Fora os hackathons presenciais, temos os recursos online, como:

- Para **front-end** podemos olhar os projetos no Codepen (<https://codepen.io/>), um site de exposição e prototipação de interfaces onde podemos ver ótimos exemplos de código front-end.
- Para todas as profissões podemos olhar os projetos no GitHub (<https://github.com/>), um site de hospedagem de projetos open source e também uma “rede social” para pessoas desenvolvedoras de software.
- Para pessoas da área de **dados**, temos também o kaggle (<https://www.kaggle.com/>), um site com diversos desafios relacionados a dados.



Além desses, temos sites como HackerRank (<https://www.hackerrank.com/>), Code Wars (<https://www.codewars.com/>) e Exercism (<http://exercism.io/>), que possuem desafios de programação para aprendermos mais sobre algoritmos e analisar se também gostamos de resolver problemas lógicos.

## 4.11 A decisão de hoje pode não ser a última

A decisão de escolher uma profissão não é a nossa única decisão na vida profissional. Muito pode acontecer, até mesmo chegar uma hora em que desejaremos trocar de área. Mudar de área, seja de uma área técnica para outra área técnica ou não, é sempre muito difícil. Não sabemos quando devemos mudar, não conseguimos imaginar se devemos mudar. Mas isso pode acontecer.

Podemos começar nossa vida escolhendo a área de front-end, back-end ou outras relacionadas a Web e de repente nos encontrarmos em ciência de dados porque, de repente, descobrimos que gostamos disso.

O que precisamos ter preparação para fazer, a qualquer momento, é: planejar **como** migrar de área.

Quando somos mais jovens é uma decisão “mais fácil”. Podemos largar nosso salário, emprego atual, às vezes até nossa cidade, e nos aventurarmos tudo de novo em outra área, mas quando temos muitas responsabilidades é um pouco mais complicado.

Por isso é necessário pensar bem antes. Só não podemos nos deixar sentir acorrentados(as) a algo que não nos estiver trazendo felicidade.

Algo que sempre vai acontecer em nossa vida profissional: começamos com uma linguagem de programação e depois de um tempo precisamos mudar por algum motivo. Seja por causa do produto em que trabalhamos, de uma mudança de empresa ou porque a nova linguagem é melhor para resolver um problema específico. As mudanças fazem parte da vida e precisamos estar preparados(as) para elas.

Devemos sempre buscar ter resiliência, a habilidade de nos adaptarmos às mudanças. Essa será uma habilidade essencial para nossa carreira e até no contexto pessoal.

## **4.12 Conclusão**

Não existe uma profissão melhor que a outra, pois o que é bom para uma pessoa pode não ser legal para as demais. O que devemos levar em consideração na hora de escolher uma profissão é o quanto nos empolgamos ao pensar em trabalhar com aquilo. É a afinidade e compatibilidade com a profissão.

Será que sentiríamos orgulho de dizer, em um bate papo qualquer, que fazemos o que fazemos?

Devemos analisar bem o que cada profissão faz. Podemos perguntar em fóruns, devemos perguntar para conhecidos(as). Temos que buscar o máximo de dados antes de cada decisão a se tomar quando se trata da nossa carreira.

E mesmo depois de tanta pesquisa, podemos passar alguns anos em uma profissão e decidir que ela não faz mais sentido para nossa vida. Nesse caso, devemos voltar às pesquisas e procurar algo que nos anime novamente.

O mais importante é: faça algo que você gosta de fazer.

## CAPÍTULO 5

# Uma galáxia de questões mal resolvidas

Quanto mais nos envolvemos com o mercado de trabalho em programação de computadores, mais dúvidas vão surgindo. Já possuímos as respostas para algumas questões, como: o que é um programa de computador, como é criado um programa, diferença entre software, sistemas e aplicações e algumas profissões em desenvolvimento de software. Entretanto surgiram outras, como: qual linguagem de programação aprender primeiro, qual faculdade/curso fazer (se é que temos que fazer isso), dentre outras.

Essas questões fazem parte da carreira de qualquer pessoa iniciante e as encontramos repetidas vezes em fóruns de desenvolvimento de software, eventos e comunidades de programação. Não devemos nos sentir mal por termos tal indecisão, pois é extremamente comum.

As respostas exatas para essas questões dependem do contexto de cada pessoa. Afinal, se você quer aprender desenvolvimento de games, sabemos que terá que aprender uma gama de especialidades, se quer desenvolver para celulares, são outros pontos. Então, a resposta para quase todas as perguntas de fóruns de tecnologia é: depende do seu contexto. Porém existe uma maneira de saber qual a resposta certa para o nosso contexto:

**aprendendo a pesquisar.** Vamos fazer isso agora.

Neste capítulo, vamos aprender a pesquisar e descobrir qual linguagem de programação aprender, qual o melhor sistema operacional para trabalhar com programação, qual a melhor IDE, qual faculdade fazer (se fizermos uma) e todo o resto que é realmente importante entender nesse primeiro momento de nossa carreira como pessoas desenvolvedoras de software.

## 5.1 Muitas linguagens de programação, qual aprender?

PHP, Python, Java, C#, Ruby, Elixir, Clojure, JavaScript... Essas são algumas das linguagens que podemos usar para programar para Web. Java, Kotlin, C e C++ servem para Mobile. Python, R, F# para trabalhar com dados. Java, C, C++ e JavaScript para Games... São muitas opções!

Quando fazemos uma pesquisa no Google procurando por “qual linguagem de programação aprender?”, não chegamos a nenhuma conclusão significativa que realmente norteie nossas decisões. Isso acontece pelo que motivei anteriormente, o contexto. Cada pessoa vai responder de acordo com sua experiência de vida, pelo que passou, pela região onde mora e outros fatores que a levaram a tomar aquela decisão.

As linguagens são somente **ferramentas** para criarmos softwares para as plataformas onde elas podem ser executadas.

Porque, então, existem tantas linguagens de programação? Não poderia existir somente uma linguagem para trabalhar em todas as áreas de desenvolvimento de software?

De acordo com a evolução dos computadores nós ganhamos mais recursos, como mais memória, maior capacidade de processamento, mais armazenamento interno, mais conectividade e as pessoas desenvolvedoras buscam alternativas para melhor aproveitá-los. As linguagens de programação podem ser uma ferramenta importantíssima para explorar mais ainda cada recurso, assim como os paradigmas de programação, que comentei no capítulo anterior. Isso porque algumas linguagens funcionam bem para trabalhar com menor quantidade de memória - pois o seu código é compilado e muito otimizado para alcançar um baixo consumo de recursos -, enquanto outras são linguagens com outro foco, que consomem mais recursos, mas dão mais velocidade de escrita do código com sua sintaxe mais enxuta, oferecendo mais produtividade no dia a dia e reduzindo o trabalho para quem escreve código.

Como exemplo, temos o caso da linguagem Ruby que, segundo seu criador, foi criada para ser divertida e fluida de se trabalhar. Python também é exemplo de linguagem interessante que possui uma filosofia de que o código deve ser fácil de se ler para todas as pessoas.

Se pegarmos o contexto histórico, veremos que as pessoas desenvolviam páginas Web utilizando a linguagem C ou Perl, mas depois a linguagem PHP facilitou o processo, então os desenvolvedores migraram para ela. Posteriormente vieram outras, ou até ao mesmo tempo em que PHP ganhava fama outras linguagens começavam a aparecer no mercado, com a mesma proposta: **melhorar a maneira como nós escrevemos nossos softwares ou melhorar como nosso software funciona**. É por isso que existem várias linguagens e várias abordagens diferentes para fazer a mesma coisa, que é desenvolver software.

A escolha da linguagem de programação deve ser pautada em:

- nosso objetivo pessoal;
- nosso objetivo profissional.

Precisamos aprender a linguagem para começar a trabalhar ou queremos uma linguagem somente para aprender programação por hobby ou curiosidade?

Se queremos ou temos tempo disponível para aprender programação com uma linguagem mais fácil e depois partir para a que realmente vamos trabalhar ou então só desejamos aprender a programar por curiosidade, eu sou um grande incentivador da linguagem **JavaScript**.

Devido à sua facilidade de aprendizado, material disponível na internet, comunidade extremamente ativa, milhares de empresas investindo na tecnologia e similaridade com a maioria das linguagens de programação disponíveis no mercado (conhecendo JavaScript, é mais fácil aprender outras linguagens depois). E ainda podemos aprendê-la com um navegador e um editor de textos qualquer em nossas mãos; ferramentas simples que já possuímos por padrão em nossos computadores.

Para esse caso, eu só analisei alguns pontos focados em aprendizagem de programação, e não o mercado de trabalho como um todo. Se a decisão for pautada em **a linguagem com a qual vamos começar a trabalhar**, então não há dúvidas de que precisamos começar com a tecnologia que funciona na plataforma que escolhermos, seja web, mobile, dados, games etc., para diminuirmos o caminho da aprendizagem até o primeiro emprego.

Mesmo quando já temos a plataforma definida, encontramos muitas opções para um mesmo objetivo, então devemos utilizar da **inteligência de mercado** para analisar qual a linguagem que vai nos empregar neste primeiro momento. Se pensarmos no ambiente Web teremos muitas vagas com as linguagens Java, PHP, C#, JavaScript, Python e Ruby no cenário atual e não são todas essas tecnologias que empregam onde moramos. Devemos fazer algumas buscas para saber qual linguagem possui mais vagas abertas e colocar isso como um ponto positivo para ela.

A inteligência de mercado seria utilizar a nossa capacidade de pesquisa, agrupamento e padronização de dados para identificar possíveis oportunidades, riscos e necessidades. Ou seja: pesquisar ao máximo o mercado de trabalho (local se for para trabalharmos na mesma cidade/estado ou até mesmo de outros estados), ter muitos dados em mãos e agrupar tudo isso para que tenhamos uma amostragem que nos ajude a tomada de decisão. Quanto mais dados tivermos em mãos mais conseguimos entender o mercado. Conseguimos saber o que as empresas estão realmente buscando, os riscos de investirmos nosso tempo em uma determinada tecnologia, assim como identificar possíveis oportunidades de investimento em algo novo.

Por exemplo: se a linguagem mais utilizada na minha cidade é Java, faz sentido eu estudar Rust? Depende. Depende se eu quero trabalhar na minha cidade, se eu quero sair do estado, se quero um emprego remoto e se a tecnologia demonstra algum potencial de crescimento futuro.

Outro fator importante, pensando em mercado de trabalho, é: **onde queremos trabalhar**. A empresa em que almejamos trabalhar pode não usar Ruby, por exemplo, podem usar Java, como uma grande quantidade de empresas em nosso país. Nesse caso não seria muito vantajoso, a curto prazo, estudar Ruby, seria melhor estudar logo Java e ir atrás do sonhado emprego.

Então, dependendo da área na qual desejamos trabalhar devemos pesquisar quais as linguagens utilizadas e os prós e contras de cada uma para nos auxiliar na escolha. Nós vamos aprender a fazer essa pesquisa agora.

## Facilitando nossa decisão

Para facilitar nossa decisão, podemos criar uma tabela para nos ajudar a escolher a nossa primeira linguagem onde vamos listar, inicialmente: dificuldade de aprendizagem, quantidade de material disponível na internet (gratuito ou pago, dependendo do nosso momento financeiro), comunidade ativa e vagas disponíveis. E, para encontrar esses dados, devemos pesquisar na internet sobre cada tópico.

Vamos utilizar o exemplo do desenvolvimento Web para ilustrar como seria o preenchimento desta tabela, pois é onde mais temos opções de linguagens atualmente. Isso vai ajudar a montar uma tabela com mais insumos a fim de um melhor entendimento do método de pesquisa.

O estado inicial da tabela seria algo como:

### Tabela limpa

Linguagem	Vagas	Dificuldade	Material disponível
-	-	-	-

Podemos começar nossas pesquisas para preencher a planilha respondendo a questão "quais as principais linguagens para trabalhar com Web?". Pegamos então as linguagens e colocamos na lista.

### Principais linguagens Web

Linguagem	Vagas	Dificuldade	Material disponível
PHP	-	-	-
Java	-	-	-
Ruby	-	-	-
Python	-	-	-
C#	-	-	-

JavaScript <b>Linguagem</b>	<b>Vagas</b>	<b>Dificuldade</b>	<b>Material disponível</b>
--------------------------------	--------------	--------------------	----------------------------

E continuamos respondendo a todas as questões, até chegar a um resultado parecido com esse aqui:

<b>Linguagem</b>	<b>Vagas</b>	<b>Dificuldade</b>	<b>Material disponível</b>
PHP	400	fácil	muito
Java	600	difícil	muito
Ruby	200	fácil	médio
Python	200	fácil	médio
C#	500	médio	muito
JavaScript	600	fácil	muito

Mas o exemplo da tabela é algo fictício, pois eu não preenchi com dados reais para não enviesar alguém que somente lê o livro e já quer tomar a decisão por ansiedade. Devemos pesquisar e preencher essa tabela com os resultados que achamos relevantes. Podemos também adicionar outros campos na tabela, como “comunidade receptiva”, “alguém próximo pode me ensinar” etc. Devemos adicionar os prós e contras bem específicos para o nosso contexto social (como custo de aprendizagem) e pessoal, além de nossos próprios gostos.

## 5.2 Nós devemos utilizar Linux para alcançar a maestria

Existe uma premissa em algumas comunidades de programação, que diz: Windows não presta para pessoas programadoras. E, dependendo da tecnologia que escolhermos para trabalhar, vamos receber essa resposta em algum momento de nossa vida profissional. Será mesmo que o sistema operacional de computadores pessoais mais utilizado no mundo não serve para trabalhar com programação?



Já vou logo adiantando que eu vou defender o Windows aqui, mas não vou defender 100%. Se eu falar para alguém "Pode continuar sempre utilizando Windows", estaria sendo hipócrita, uma vez que larguei o sistema operacional muito tempo atrás, antes mesmo de aprender a programar e nunca mais quis voltar.

Mas eu não deixei o Windows porque ele travava ou porque ele tinha um monte de vírus: isso nunca foi um problema para mim uma vez que eu estudei como melhor utilizá-lo, fazia configurações de desempenho e possuía políticas de segurança bem definidas. Eu deixei o S.O. (sistema operacional) porque optei por utilizar outro de código-fonte aberto e que segue a filosofia do software livre, que é algo que me inspira desde quando conheci computação e vou comentar mais à frente no livro, o Kurumin Linux.

O Kurumin era uma distribuição de Linux com o objetivo de ser fácil de se usar, pois naquela época as distribuições Linux não eram tão fáceis quanto são hoje. Uma pena é que ele foi descontinuado pouco tempo depois de eu o conhecer, em 2009.

O verdadeiro problema do Windows, para mim, é que dependendo da tecnologia que vamos trabalhar, realmente ele não é **a melhor ferramenta**, pois as configurações da linguagem podem ser complicadas no S.O. e isso pode virar uma pedra em nosso sapato. Repare que eu comento que ele pode não ser a melhor ferramenta, não que ele **não deve** ser usado ou que **não presta** para trabalhar.

A linguagem Ruby, o Node.js e a linguagem Golang, por exemplo não possuem o melhor ecossistema para se trabalhar no Windows devido às peculiaridades do sistema.

Isso acontece porque o Windows tem peculiaridades como o trabalho mais complicado de configurar variáveis de ambiente (configurações que fazemos no S.O. para utilizar em nossas linguagens) ou o padrão de acesso a pastas (em sistemas Unix, como o Linux ou OS X o padrão de acesso a diretórios é com a barra / e em Windows com a barra invertida \ ),

comandos diferentes na linha de comando (como o comando de listagem de pastas que é `ls` em todos os sistemas e `dir` no Windows).

Toda essa diferença do Windows para Linux ou OS X existe porque, como qualquer outro S.O., o sistema foi pensado para uma finalidade e a finalidade dele é funcionar bem para o **usuário final** e para **pessoas que usam tecnologias Microsoft**, como o C# e SQL Server.

Em sistemas Linux nós não teremos algumas dores de cabeça que o sistema da Microsoft nos traz de vez em quando, pois com alguns comandos já temos tudo instalado, configurado e pronto para trabalharmos. Já no Windows são vários cliques e reiniciar diversas vezes até que tudo esteja OK. Mas isso não é um problema para todo mundo, somente para quem quer algo mais produtivo.

A própria comunidade ao redor das tecnologias produz alternativas e faz com que as coisas que usamos em Linux passem a funcionar no Windows com menos dores de cabeça. Temos artigos na internet que podem nos ajudar a configurar as ferramentas e não passarmos por nenhuma dor de cabeça com o uso desse S.O.

Hoje em dia a história tem mudado, uma vez que o Windows 10 possui até mesmo um modo de utilização de subsistemas Linux dentro dele, um modo de rodar o Linux ao mesmo tempo que usamos o Windows. Podemos utilizar o melhor da compatibilidade com hardware do Windows, que não é a melhor em sistemas Linux, com o melhor do mundo de desenvolvimento dos sistemas Unix.

Temos também outro ponto: para uma pessoa iniciante em programação, **tanto faz** o sistema operacional utilizado. Sua maior dor de cabeça será entender lógica de programação e aprender a programar, não o quão rápido o sistema operacional inicializa.

Devemos usar o que nos deixa confortáveis e que não nos atrapalhe em nada. Se for Windows, que continuemos com Windows até o dia em que o sistema começar a nos atrapalhar, se isso acontecer. Se for Linux, devemos seguir a mesma filosofia e continuar utilizando até quando não fizer mais

sentido. O importante mesmo será aproveitarmos do sistema operacional que mais nos traz produtividade no dia a dia.

### 5.3 Faculdade na área de TI, fazer ou não fazer?

O processo padrão de algumas profissões é fazermos o ensino fundamental, ensino médio, talvez um curso técnico, então uma faculdade e podemos, finalmente, conseguir um emprego na área. Em desenvolvimento de software podemos esquecer os padrões. Porém, mais uma vez, nosso contexto importa muito.

Uma grande parte das pessoas desenvolvedoras de software que eu conheço não possuem formação padrão (uma graduação) e são adeptas de dizer para todo mundo que não devemos fazer faculdade para trabalhar com programação.

Eu concordo. **Não precisamos obrigatoriamente de faculdade para trabalhar com programação**, mas, dependendo de onde moramos, as vagas de emprego podem nos cobrar essa formação padrão.

Mais uma vez, assim como acontece com as linguagens de programação, devemos utilizar a inteligência de mercado e pesquisas para entender se devemos ou não fazer uma graduação. Em cidades como São Paulo e Rio de Janeiro, em contato com amigos da área, percebi que não é cobrado graduação na maioria das empresas, mas não acontece o mesmo no nordeste do país, por exemplo.

A grande maioria das empresas não vai nos cobrar a formação, porém vão nos pedir um portfólio, uma participação ativa em comunidades de desenvolvimento de software ou algum outro diferencial de mercado.

A faculdade pode ser o meio mais fácil de entrar no mercado de trabalho, afinal conseguir um estágio é mais fácil do que conseguir o primeiro emprego como júnior em uma empresa devido à procura que elas têm por esses profissionais.

Existe outro ponto importante quando falamos de faculdade, que é a possibilidade financeira de se fazer um curso superior. Quem não possui recursos financeiros para fazer um curso não deveria ficar de fora da nossa área e, graças à internet, essas pessoas não vão ficar sem oportunidade, assim como eu não fiquei.

Todo conhecimento que precisamos para trabalhar com programação está na internet, está em livros como este, em artigos, em vídeos no YouTube. Não precisamos gastar muito dinheiro para começar e, se necessário, não precisamos gastar nada além do nosso tempo.

Claro que a ausência da faculdade poderá significar que nós, como profissionais, precisaremos nos provar mais em entrevistas de emprego, como possuir um portfólio melhor/maior ou termos estudado conteúdos específicos, por não termos sido obrigados(as) a estudar o conteúdo da faculdade, mas isso não nos impede de sermos bons profissionais.

Quando eu comecei na área de desenvolvimento de software cheguei a pensar em iniciar em uma faculdade, porém a dificuldade financeira limitou a minha escolha, assim como a minha disposição para estudar para um vestibular em uma universidade gratuita (eu amo estudar programação, mas não gosto tanto de estudar outros assuntos). Para conseguir o primeiro emprego eu tive que utilizar outras estratégias, como estudar e praticar muito e construir um portfólio para mostrar para as empresas que eu sabia o que eu dizia saber em meu curriculum.

Concluindo, para saber se realmente precisaremos de uma faculdade para iniciar na área, será necessário analisar o mercado local. Se o mercado não exige faculdade, então não é necessário. Se o mercado exige, então é importante, sim, infelizmente.

Eu recomendo a faculdade para pessoas que desejam aprender mais teoria e não conseguem estudar isso por conta ou para pessoas que desejam seguir carreira acadêmica. Em todo caso, a faculdade é só mais uma das fontes de conhecimento que podemos buscar, não a mais importante na minha opinião e experiência profissional baseada em meu contexto.

Eu sempre recomendo para as pessoas a busca de conteúdo na internet, como leitura de artigos, participação em comunidades de programação, livros, como os da Casa do Código, cursos abertos no YouTube ou os da Caelum e Alura, e não estou dizendo isso porque este livro foi produzido pela editora, mas porque realmente indico o conteúdo dessas fontes.

## **5.4 Diferenciando as faculdades de programação**

Se analisarmos o ponto anterior e for decidido que devemos fazer uma faculdade, em qual curso deveríamos adentrar? Pois existe Ciência da Computação, Engenharia da Computação, Sistemas de Informação, Análise e Desenvolvimento de Sistemas e Sistemas para Internet, só focados em nossa área. Isso além de outros cursos específicos que vêm nascendo dia após dia para suprir as necessidades do mercado.

Cada curso desses possui uma razão para existir e conhecimentos importantes para determinadas áreas de atuação.

Vale a pena levar em consideração que os cursos citados a seguir existem na rede pública de ensino nas universidades federais e FATECs. Então, se acontecer o interesse por algum deles, recomendo procurar na universidade mais próxima se ministram algum deles.

### **Ciência da Computação**

No curso de Ciência da Computação a pessoa receberá uma base sólida de programação, matemática e assuntos mais profundos relacionados a algoritmos e também ganhará forte conhecimento relacionado à pesquisa.

### **Engenharia da Computação**

Neste curso, a pessoa recebe mais foco em hardware, além de tudo o que é relacionado desenvolvimento de sistemas. É a pessoa que recebe habilidades para criar componentes físicos de computadores. Seus estudos envolvem muita física e linguagens de baixo nível.

## **Sistemas da Informação**

Em Sistemas de Informação, além de desenvolvimento de sistemas, a pessoa aprende muito sobre áreas de negócios, como administração de empresas e gestão. Estará altamente habilitada a planejar e criar softwares para automação de empresas no geral.

## **Análise e Desenvolvimento de Sistemas**

É um curso mais generalista e que não se aprofunda tanto em hardware ou algoritmos, como Engenharia e Ciência da Computação ou em negócios como Sistemas da Informação, porém recebemos o conteúdo ideal para saber planejar e criar sistemas e nos tornamos muito maleáveis para futuras necessidades de aprendizagem.

## **Sistemas para Internet**

Bem parecido com Análise e Desenvolvimento de Sistemas, com a diferença de que, nesse curso, seu foco é para sistemas que rodam na plataforma Web, sejam ERPs, sites, portais etc., mas não recebe tanto foco em outras áreas como desktop.

Desses cursos, temos Ciência da Computação, Engenharia da Computação e Sistemas da Informação como cursos no modelo bacharelado, com duração de 4 anos, e Análise e Desenvolvimento de Sistemas e Sistemas para Internet como cursos tecnólogo, com duração de 2 anos e meio.

A escolha do curso vai variar conforme nossa necessidade e também com a disponibilidade de tempo. Minha única recomendação na escolha do curso é que procuremos uma boa instituição de ensino, se economicamente possível, pois isso faz muita diferença em nosso aproveitamento do período da graduação.

Para encontrar algum dos cursos em uma universidade gratuita, podemos fazer uma pesquisa no Google como “Ciência da Computação universidade pública Rio de Janeiro”, onde mudamos Rio de Janeiro para nossa localização. Serão apresentadas diversas pesquisas, principalmente

pesquisas de X melhores universidades da localidade (10 melhores faculdades do Rio de Janeiro, por exemplo). Essas pesquisas levam em consideração a avaliação do público e também a nota do MEC, o Ministério da Educação.

## **5.5 Para dominar programação preciso trabalhar somente com coisas difíceis**

Um comportamento muito replicado por pessoas iniciantes é achar que quem programa em uma linguagem de baixo nível, uma linguagem “mais difícil”, quem usa uma distribuição de Linux mais complicada de configurar, um editor de textos que precisa de um curso para aprender usar ou coisa do tipo é melhor programador(a) do que outras pessoas.

Existe também aquela velha imagem da pessoa que não dorme, pessoa que vive à custa da má alimentação e ingestão de café ou energéticos e outras características que nós conheceremos de acordo com que nos envolvendo mais com comunidades.

Isso tudo não passam de estereótipos replicados pela mídia ou por pessoas que acham que esse comportamento é legal, mas a realidade é que nem toda pessoa desenvolvedora de software precisa sofrer com esses comportamentos, muito pelo contrário. Uma pessoa que não dorme, se alimenta mal ou vive a custas de cafeína não será a melhor pessoa desenvolvedora de softwares, somente uma pessoa com problemas de saúde que trabalha com programação.

O mesmo podemos afirmar sobre linguagens de alto ou baixo nível, uso do sistema operacional mais complicado de usar. Isso não nos define como bons profissionais ou como pessoas mais inteligentes do que a maioria das pessoas. É só uma decisão pessoal, de mercado ou necessidade especial dependendo da plataforma na qual trabalhamos. Então, não. Ninguém que deseja aprender a programar ou trabalhar com desenvolvimento de software precisa sofrer todo dia para ser o(a) melhor profissional que existe.

## 5.6 Conclusão

Ainda surgirão outras questões durante nossa carreira, mas temos que analisar tudo com muita criticidade, levar em conta o nosso contexto pessoal e até história de vida para tomar as decisões.

Sempre que encontrarmos alguém dizendo que precisamos fazer exatamente uma coisa, devemos tomar cuidado. Não existem verdades absolutas, mas pontos de vista diferentes relacionados com cada pessoa. Devemos utilizar do nosso senso analítico e sempre pesquisar de várias fontes para conseguir bons insumos para nossas tomadas de decisão.

Neste capítulo focamos em tirar do nosso caminho as dúvidas técnicas. Para o próximo capítulo vamos aprender mais e tirar dúvidas não técnicas, como: como saber se uma informação é boa, como controlar nossa ansiedade para manter o foco, como se proteger de verdades absolutas e até como fazer boas perguntas na internet para que nossa inserção no universo da programação seja ainda mais fácil.



## **CAPÍTULO 6**

# **Habilidades de sobrevivência no universo da programação**

Em nossa vida, precisamos adquirir certas habilidades especiais para resistir a alguns acontecimentos, como buscarmos a resiliência para aqueles momentos de adversidade, ou habilidades de comunicação para conseguirmos uma boa convivência em sociedade.

Na área de tecnologia existem algumas habilidades especiais que acredito que precisamos adquirir para que algumas coisas não nos atrapalhem, como a ansiedade, o medo de arriscar algo novo ou simplesmente para que aprendamos mais rápido e consigamos acompanhar o mercado. Também acho importante que busquemos ter mais criticidade com o conteúdo que consumimos e existem algumas técnicas para conquistarmos essa produtividade da qual tanto corremos atrás.

Essas técnicas e conselhos podem nos ajudar para o resto da vida, tanto no aspecto profissional quanto em outras áreas da nossa vida e eu as aprendi durante minha busca pelo equilíbrio profissional e pessoal por meio de palestras que assisti, livros e cursos. Acho muito importante compartilhar essa informação para que todos tenhamos a mesma oportunidade de crescer na área de desenvolvimento de maneira menos “atribulada”.

## **6.1 Sobrevivendo a uma chuva de informações**

A carreira de uma pessoa desenvolvedora de software é cercada por pesquisa e estudo constante. Durante nossas pesquisas vamos encontrar muito conteúdo repetido, muito texto pouco explicativo e muita informação espalhada em diversas fontes. Podem ser livros, artigos, vídeos, e-books, posts em redes sociais etc. e podem ou não ser boas fontes de conteúdo.

Uma pessoa pode escrever um artigo dizendo que a tecnologia X é a melhor tecnologia do universo, enquanto outra pessoa pode dizer que isso não presta. O mesmo acontece com os livros, um mesmo assunto pode ser ensinado de diversas maneiras e podemos nos confundir. Qual pessoa está falando o modo correto de se trabalhar, de programar, de pensar na nossa carreira?

Talvez a resposta seja: todas elas. Todas as pessoas estão falando o modo certo **para elas**. Elas estão falando da perspectiva delas, com a história que elas carregam, seu contexto social, suas práticas familiares, o que aprenderam na escola, em curso, em faculdade. Estão falando com base no seu próprio contexto e muitas vezes não levam em consideração o contexto de outras pessoas e é por isso que pode acontecer de mais nos atrapalharmos do que crescermos seguindo certos conselhos que recebemos nos artigos ou via redes sociais de alguns profissionais influenciadores do mercado de programação.

Cada pessoa aprende de um jeito e, na grande maioria das vezes, desse mesmo jeito ela transmite o conhecimento para o próximo. Pode não ser o melhor modo de aprender ou de ensinar, mas para ela serviu e **ela acredita que esse é o correto**.

Para que esse monte de conteúdo repetido e espalhado não nos atrapalhe, devemos procurar o conteúdo mais relevante. A pessoa que compartilha a informação demonstra domínio daquilo que ela está ensinando? O conteúdo está bem-estruturado de maneira que uma pessoa no nosso nível de conhecimento atual consiga entender? Essas são perguntas essenciais para que possamos confiar no que estamos consumindo.

### **Como saber se um conteúdo está bem-estruturado**

Quando estamos começando a estudar algo é comum que não entendamos todos os termos técnicos, que não compreendamos todos os exemplos. Porém existem maneiras de descomplicar um conteúdo para o público alvo.

Se o público alvo é mais iniciante a pessoa vai utilizar linguagem que alguém em nível básico entenda. Se não, se o público alvo é formado por

profissionais experientes, então o escritor ou escritora não vai se importar tanto com linguagem mais técnica e exemplos quase nada possíveis de se entender para um ser humano comum.

Este livro, por exemplo, explica os conceitos que acreditamos serem desconhecidos para a maioria das pessoas, pois o seu conteúdo deve ser universal. Ou seja, deve servir de orientador para uma pessoa iniciante ou uma pessoa com algum tempo de carreira.

Alguns conteúdos na internet seguem essa mesma linha de pensamento e nos orientam muito bem quando iniciantes. Só precisamos procurar uma boa fonte de conhecimento e absorver o máximo dela.

Muitas vezes também vamos consumir um conteúdo que é destinado para iniciantes, mas mesmo assim nos perdemos no meio, não entendemos tudo o que está sendo ensinado ali. Isso é extremamente normal.

Recorrentemente precisamos ler, reler, testar, praticar, para depois entender um assunto, principalmente quando é algo extremamente novo e não tem muito conteúdo diversificado na internet para encontrarmos outras fontes. Não devemos nos achar menos capazes por ter que reler um texto ou por ter que praticar mais um assunto para o entender. Todo mundo passa por isso.

## **6.2 Onde encontrar tudo o que preciso aprender**

Aprendemos desde muito cedo que o conhecimento é adquirido indo a uma instituição de ensino, uma escola, ouvindo uma pessoa que se coloca no papel de instrutora, indo sempre naquele horário predefinido, fazendo avaliações e “passando de ano”.

Esse modelo de ensino é mais antigo do que a tecnologia a qual temos acesso hoje em dia e nem de longe é o melhor. A troca de conhecimento e o estímulo do protagonismo no aprendizado é o que devemos buscar.

Quando trocamos experiências com outras pessoas aprendemos muito mais do que somente ouvindo alguém falar e irmos embora, pois recebemos uma responsabilidade maior com o que vamos falar/transmitir e por isso

precisamos pesquisar mais, aprender mais, não somente repetir o que nos é falado, sem tomar nossas conclusões sobre algo para sabermos o que dizer.

É extremamente comum encontrarmos pessoas legais, que conseguem compartilhar informação de um modo que nós conseguimos entender e que nos ajudam quando precisamos. Essas pessoas podem se tornar nossas mentoras e nos guiar para não precisarmos passar por certos problemas. Esse é o papel de uma pessoa que se coloca como professor(a) em uma instituição de ensino, o problema é que o papel de mentor(a) foi substituído pelo de ditador do conhecimento nas escolas e instituições de ensino atuais, onde temos uma pessoa que acha deter todo o conhecimento do mundo e muitas vezes não sabe transmitir para frente, somente jorra conteúdo de qualquer modo e quem não entender é porque não estudou o suficiente, não levando em consideração as dificuldades e inteligências individuais.

A informação que precisamos absorver está toda espalhada na internet. Temos o Google como grande centralizador da informação nos dias atuais, temos blogs fantásticos, publicações no Medium, vídeos gratuitos no YouTube e muito conteúdo relevante para nosso aprendizado e crescimento profissional.

O conhecimento está espalhado por aí, devemos aprender a ir atrás dele.

## **6.3 Aprendendo tudo na velocidade da luz**

Ao iniciar nossos estudos algo vai nos afetar bastante: a ansiedade. Nós queremos aprender tudo rapidamente e, pior ainda, tudo ao mesmo tempo. O maior desejo é aprender tudo num dia só e já sair criando o próximo programa revolucionário que vai mudar a história da humanidade. Mas calma. Não é assim que funciona.

Cada pessoa possui um ritmo de aprendizado, um tempo específico disponível para os estudos, além de diversas responsabilidades pessoais de nossa rotina diária. O conteúdo que precisamos aprender para nos

tornarmos bons/boas programadores(as) também não é pouco, não dá para absorvermos tudo de uma vez.

Antes de começar a acelerar as coisas, é necessário preparar o terreno para os estudos. Se desejamos aprender algo rapidamente, precisamos ter em mente que não vamos aprender tudo de uma só vez, mas que podemos aproveitar muito melhor o nosso tempo e aprender de maneira produtiva. É muito melhor estudar um pouco todo dia do que pegar um final de semana inteiro para estudar algo e na segunda-feira ter esquecido tudo.

Existem algumas técnicas que nos ajudam a aprender mais rápido. Eu estudei diversos modelos, sigo algumas dicas e acabei aprendendo e adaptando algumas coisas enquanto ajudava pessoas a aprender programação.

A maneira como faço é: montar um ambiente de estudos, utilizar uma técnica para manter o foco, buscar sempre estudar de maneira ativa, seguir algumas técnicas de estudos e sempre tirar um tempo para descompressão/relaxamento.

Vamos entender cada ponto.

## **1. Preparar um ambiente de estudos**

Nós não somos como os computadores que possuem um processador com vários núcleos e pode separar uma tarefa em cada núcleo para processar tudo paralelamente, por isso é importante que no nosso ambiente de estudos tenha o mínimo de distrações possível.

De preferência um lugar que esteja organizado, como uma mesa de estudos. Não precisa ser uma mesa especial, inicialmente. Eu uso a mesa da cozinha enquanto escrevo este livro, pois é o lugar onde eu não escuto nenhum barulho além da música no meu fone de ouvido e de quebra tenho a geladeira por perto para fazer um lanchinho nas pausas. Aliás, colocar uma música legal também pode ser um bom estímulo para espantar a preguiça.

Um ambiente organizado e sem distrações vai facilitar para que não nos percamos em meio a outras tarefas, como ouvir o que a pessoa que

apresenta o telejornal está falando, olhar uma imagem bonita que apareceu na tela do celular etc.

Falando em celular, afaste-o ou desabilite as notificações. Os celulares megaconectados de hoje em dia são uma armadilha para nossa produtividade. Deixe na mesa somente o que é importante para a tarefa que será executada.

Enquanto eu estudo, normalmente, deixo somente o notebook, se estiver lendo um livro, deixo o livro, um copo d'água e um caderno para anotar para depois as ideias mirabolantes que aparecem enquanto estamos estudando e que tiram totalmente nosso foco.

## **2. Mantendo o foco**

Com o ambiente organizado conseguimos manter o foco nos estudos, porém nem sempre o que acontece externamente é o único problema. Existem também aqueles momentos em que começamos a estudar algo superimportante e, de repente, nos vemos fazendo pesquisas inúteis na internet. Eu sempre caía na armadilha de entrar nas redes sociais para dar uma olhadinha e quando via já haviam passado 30/40 minutos, esgotava minha energia e eu não queria mais continuar estudando.

Precisamos manter o foco e para isso podemos estipular um horário em que não vamos fazer nada além de focar 100% nos estudos e horários para uma pequena pausa. Se aparecer aquela vontade de pesquisar algo que não vai agregar em nossos estudos, podemos anotar isso para pesquisar/fazer depois e usamos somente o tempo da pausa para redes sociais ou outras distrações.

Existe uma técnica muito boa que pode nos ajudar a manter o foco, a Técnica de Pomodoro, uma técnica inventada por Francesco Cirillo, por volta de 1980. Essa técnica maximiza a eficiência reservando 25 minutos de foco ininterrupto, tirando pausas para um descanso de 5 e 10/15/30 minutos e voltando para os 25 minutos de foco intenso.

O passo a passo para utilizarmos a Técnica de Pomodoro é:

1. Escolher e listar as tarefas a serem executadas;

2. Ajustar um cronômetro para o tempo de um pomodoro (25 minutos);
3. Escolher a tarefa inicial;
4. Trabalhar na tarefa escolhida até que o alarme toque. Se alguma distração importante surgir, anotá-la e voltar o foco imediatamente à tarefa em execução;
5. Quando o alarme tocar, marcar um "x" na lista de tarefas;
6. A cada pomodoro tirar uma pausa de 5 minutos;
7. Quando finalizarmos quatro pomodoros, fazer uma pausa mais longa (15-30 minutos), zerando a contagem de marcações e retornando ao passo 1 ou continuar na tarefa.

Se somarmos os 4 pomodoros (25 min) + 3 pausas de 5 minutos e uma de 15 minutos, nós teremos 2 horas e um pouco mais de trabalho intenso! Aplicando isso aos estudos temos a chave para ir mais longe, que é o foco. Se for possível estudar essas 2 horas todos os dias durante a semana (sem contar finais de semana), teremos 10 horas de estudos muito bem utilizadas.

Existem excelentes aplicativos que podem nos ajudar a fazermos nossos pomodoros. Eu costumo utilizar o Brain Focus (<https://brainfocus.io>).

Muita gente não consegue utilizar a Técnica de Pomodoro porque acredita que precisam começar desde o primeiro dia sendo 100% produtivas, mas isso não é uma realidade. Para cultivarmos o hábito de utilizar o pomodoro, podemos sim começar de modo mais desleixado, fazendo pomodoros não tão focados ou então pausas mais longas.

### **3. Aprendizado ativo**

Existem duas maneiras de aprender: a maneira passiva e a maneira ativa.

O aprendizado passivo é aquele em que nós somente deixamos o conteúdo passar pela nossa mente e ir embora, como a leitura de um livro, assistir uma aula ou um curso de maneira corrida, sem parar nenhuma hora para pensar sobre aquilo que vimos. Somente consumir um conteúdo, sem colocar nosso cérebro para trabalhar faz com que, em poucos dias, esqueçamos tudo o que foi lido.

Quando lemos uma fábula é comum que esqueçamos alguns detalhes, mas aqueles momentos que fizeram a nossa mente viajar ficam marcados em nossas mentes e são eles que retornam quando lembramos da leitura ou quando vamos indicar o livro para alguém.

Já o aprendizado ativo é aquele onde anotamos, fazemos resumos, grifamos pedaços dos livros, fazemos exercícios práticos e também compartilhamos o que aprendemos. É um modo de pensar mais sobre um tema do que somente ver que existe e deixar passar a fim de ver/receber mais informações em menos tempo.

Uma maneira de estudar ativamente é formular questões enquanto estudamos. Consumir um conteúdo e formular perguntas sobre isso fará com que nosso cérebro trabalhe em dobro em cima do que estamos aprendendo.

Colocar em prática o que aprendemos também é extremamente importante, principalmente quando falamos de programação. Claro que não dá para colocar tudo em prática... Não é todo mundo que pode colocar um foguete em órbita ou possui peças para criar um robô. Mas quando estudamos programação podemos sempre acrescentar a prática em nossa rotina de estudos.

Que tal criar um sistema, um aplicativo ou qualquer coisa com a linguagem de programação que acabou de aprender?

Colocar em prática nos ajuda a ir além do material básico de estudos, aquele que uma pessoa planejou e criou um caminho perfeito a ser seguido, que eu chamo de o caminho do arco-íris, que nos leva diretamente ao pote de ouro, mas não nos apresenta desafio algum ou que os desafios são previsíveis. Quando praticamos aparecem problemas que nunca imaginávamos encontrar e o esforço para solucionar estes problemas aumenta muito nosso conhecimento.

#### **4. Técnicas de estudos**

Cada pessoa possui um modo de produtividade diferente. Algumas são produtivas de manhã, outras à tarde, outras à noite. Algumas pessoas



conseguem estudar no ônibus em movimento, outras não. Outras conseguem aprender tudo rápido até com vídeos em velocidade 5x, outras precisam ver e rever várias vezes.

Existem técnicas de estudos que podem nos auxiliar a aprender, mas também é importante identificar qual é o horário em que cada um de nós aproveitamos melhor os estudos. Precisamos pesquisar e **testar** o máximo de técnicas e horários de estudos possíveis para conseguirmos algo que seja eficaz em nosso contexto.

Algumas técnicas conhecidas e que podemos pesquisar e testar são:

1. Mapa mental;
2. Fichamento ou resumo;
3. Gravação de áudios;
4. Resolver o máximo de exercícios possíveis;
5. Estudo em grupos;
6. Autoexplicação;
7. Estudo intercalado;
8. Repetição espaçada;
9. Prática distribuída.

Vale lembrar que essas técnicas são modelos de **aprendizado ativo**. O que é excelente.

No meu caso, testei diversas técnicas e o que mais funcionou pra mim foi: fazer resumos, estudo em grupo, estudo intercalado, repetição espaçada e prática distribuída. Eu faço o seguinte:

- **Resumos:** escrevo artigos sobre o que eu estudo, que podem ser publicados em meu blog (<https://woliveiras.com.br>) ou não.
- **Grupos de estudos:** sempre que possível eu participo de grupos de estudos para trocar experiência com as pessoas e aprender ainda mais.
- **Estudo intercalado e prática distribuída:** normalmente estudo mais de uma coisa ao mesmo tempo (não na mesma hora ou dia), sendo que seleciono dois dias da semana para um assunto (2 horas por dia seguindo os pomodoros) e dois dias de outro assunto. Um dia da semana eu deixo para descansar, sem contar finais de semana.

- **Repetição espaçada:** de tempos em tempos eu tento revisar algo que aprendi. Eu não costumo escolher um tempo específico para voltar a ver algo, sempre que tenho um tempo livre eu costumo revisar as coisas que escrevi ou pesquisar de novo sobre algo.

Isso é o que funcionou para mim, mas o ideal é que cada pessoa exercite e aprenda técnicas de estudos para que consiga o melhor para o seu ritmo e estilo de vida.

## 5. Descansar é importante

Sobrecarregar nunca é bom. Um carro sobrecarregado não sai do lugar, um cérebro sobrecarregado não vai guardar as informações por muito tempo, assim como começará a evitar os períodos de estudos, deixando-nos desconfortáveis com isso e, muito possivelmente, pode começar a apresentar problemas graves.

É importante que tenhamos um hobby, pratiquemos esportes e selecionemos um tempo para simplesmente **não fazer nada**. De preferência que procuremos a prática de esportes para fortalecer nosso corpo e mente.

O descanso e o movimento do corpo vão nos garantir a saúde e disposição de que precisamos para continuar aprendendo mais. Somente estudar ou fazer qualquer coisa sem um limite para tirarmos um tempo para nós mesmos não é nada saudável e logo teremos a fadiga de estar sempre fazendo o mesmo todos os dias e o desânimo dessa rotina.

Se for possível, devemos procurar a prática de esportes todos os dias da semana, pelo menos 30 minutos, segundo o professor doutor de Educação Física João Carlos Bouzas Marins, da Universidade Federal de Viçosa (UFV) (Veja mais em: <http://bit.ly/faca-exercicios>).

Somando um ambiente organizado, foco, técnicas e aprendizado ativo com o descanso e a prática de esportes, conseguiremos uma absorção muito maior do que estivermos estudando. É assim que aprendemos mais em menos tempo: otimizando nosso tempo e cuidando da nossa saúde.

## 6.4 Gerenciando melhor o tempo de estudo e trabalho

Algo que nos aflige diariamente é a vontade/necessidade de estudar algo e não ter tempo graças à rotina de trabalho, que ocupa mais da metade do nosso tempo de vida. Isso atinge a grande maioria das pessoas, afinal precisamos trabalhar para nos manter. Felizmente nós podemos encontrar um equilíbrio entre trabalho e estudos se tivermos um bom planejamento da nossa rotina.

Rotina, aqui, é a palavra-chave. Muita gente quando vê a palavra *rotina* associa a algo chato, ruim, monótono, porém não é isso. A rotina nos ajuda a conquistar a produtividade. Se tivermos tudo bem planejado e seguirmos esse nosso plano, conseguiremos equilibrar nossa vida a fim de conseguirmos executar mais tarefas do que viver sem planejamento. No final, vai parecer até que temos mais tempo do que as outras pessoas, mas na realidade só estamos usando melhor o nosso tempo.






É extremamente importante que conheçamos o que nos atrapalha quando pensamos em ter mais tempo para estudar. Será que não temos tempo útil porque o trabalho está ocupando todo o nosso dia ou será que passamos mais tempo nas redes sociais do que fazendo o que realmente importa para nossa vida?

Uma maneira de acompanhar isso é fazendo listas de tarefas e, sempre que uma tarefa não for cumprida na hora certa, anotar o porquê da falha no planejamento. Uma reflexão no final da noite vai nos ajudar. Lembrar o que fizemos durante o dia e o que tirou a nossa produtividade nos ajudará a encontrar o que realmente suga nosso tempo e até a nos conhecermos melhor.

Existem aplicativos que podem nos ajudar a acompanhar nosso dia a dia e descobrir exatamente onde gastamos nosso tempo. Eu recomendo o QualityTime e o SmarterTime, porém podemos pesquisar na loja de aplicativos o que tem mais informações úteis para o nosso caso pessoal, basta procurar por time tracker.

Utilizando o aplicativo QualityTime (<http://qualitytimeapp.com>) eu acompanho minha utilização do celular e o resultado é bem assustador quando observamos onde foi gasto meu tempo durante uma certa semana.

## Weekly Usage 29h 0m

- 1  Twitter 6h 59m
- 2  Telegram X 6h 51m
- 3  Instagram 4h 31m
- 4  Chrome 1h 46m
- 5  feedly 1h 32m

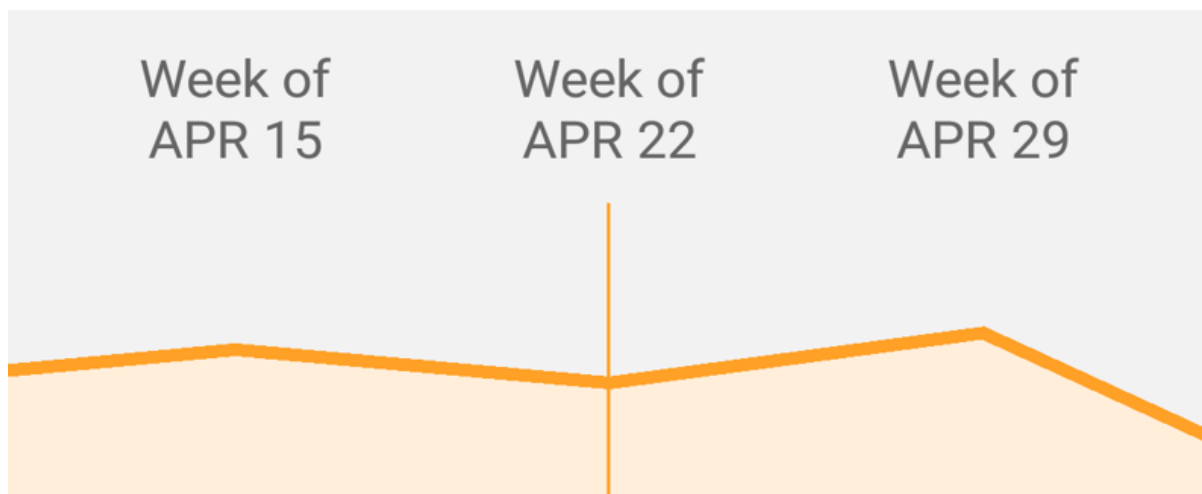







Figura 6.1: Relatório semanal do QualityTime

No relatório vemos que gastei 29 horas da minha semana na utilização do celular, sendo o Twitter o aplicativo mais utilizado, seguido do Telegram. Ambos eu utilizo para me comunicar com as pessoas, mas raramente para aprender algo novo.

Daily Frequency **271 times**

- 1  Telegram X 110 times
- 2  Twitter 34 times
- 3  Chrome 30 times
- 4  WhatsApp 24 times
- 5  Gmail 11 times

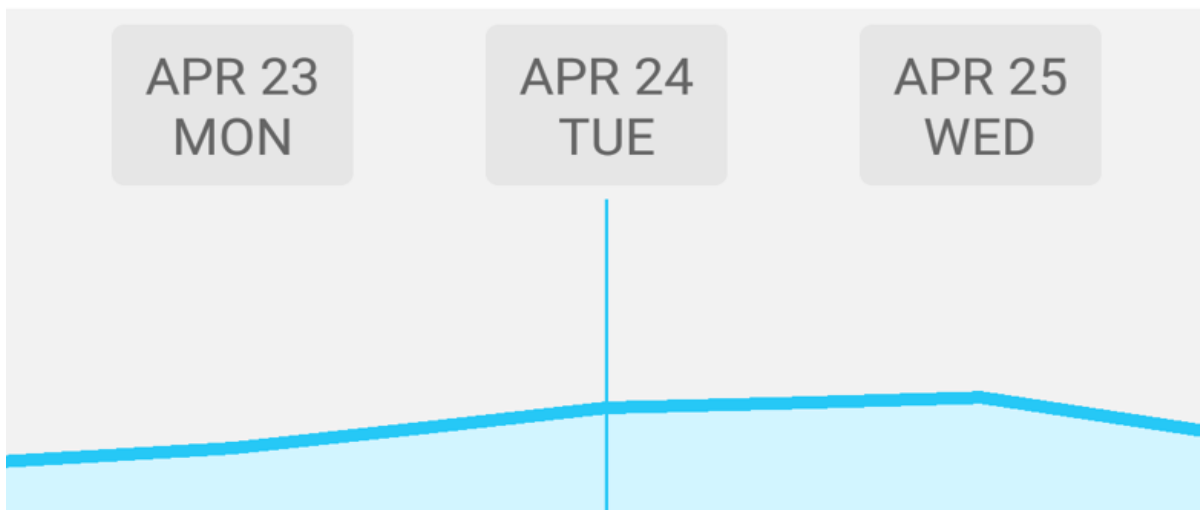


Figura 6.2: Frequência diária de utilização do celular

Percebemos na imagem que o que eu mais faço é conversar com as pessoas, afinal abri o Telegram 110 vezes e o WhatsApp 24 em um único dia.

Quanto tempo será que foi perdido de maneira não produtiva nesses casos?

Com esses dados eu não estou querendo dizer que temos que parar de conversar com as pessoas... Longe disso! Somos animais sociais e precisamos nos comunicar. Porém, podemos escolher melhor **quanto tempo gastamos conversando sobre assuntos não produtivos**.

A maioria das pessoas se perde no uso de redes sociais. Se repararmos no ônibus as pessoas estão de olhar fixo no celular rolando a página vendo a próxima foto no Instagram, outra publicação no Facebook ou falando no WhatsApp nos grupos de “zoeira”. Esse é o comportamento padrão da nossa sociedade, afinal não temos o costume de refletir muito sobre onde estamos gastando nosso tempo.

O SmarterTime (<http://smartertime.com>), além de acompanhar a utilização no celular, também nos ajuda a descobrir onde estamos gastando o tempo no computador. Eu, pessoalmente, não o utilizo diariamente, mas o utilizei durante um período para analisar onde eu perdia mais tempo no PC e descobri que o que mais me atrapalhava era o celular mesmo, por isso fiquei somente com o QualityTime.

Podemos procurar o aplicativo que mais se adapta a nossa necessidade. Como eu descobri que o celular é o grande ladrão do meu tempo, passei a utilizar o QualityTime também para limitar a minha utilização do aparelho.

Quando eu parei de utilizar o celular no percurso até o meu trabalho eu li, em um mês, três livros, somente no transporte coletivo, e em outro mês li quatro livros. O detalhe é que eu não li esses livros de modo corrido, aquele aprendizado passivo, li de maneira ativa. Inclusive alguns desses livros foram consumidos para melhorar a minha didática aqui nesta obra, outro foi para melhorar minha apresentação em público em minhas palestras. Ou seja, eu estava anotando, grifando e fazendo resumos dos capítulos que eu lia.



É incrível perceber que somente abandonando o uso descontrolado do aparelho celular foi “liberado” essa quantidade de tempo produtivo.

Livro	Mês
Planejando livros de sucesso	January
Escreva seu livro agora	January
Técnicas de aprender: conteúdos e habilidades	January
Comunicação Não-Violenta	February
Morando sozinha	February
TED Talks	March
Produtividade para quem quer tempo	March
Um apelo a consciência: os melhores discursos de Martin Luther King	March
Por que fazemos o que fazemos	March

Figura 6.3: Livros lidos depois de abandonar o celular no transporte coletivo

Essa é uma maneira de conseguirmos mais tempo para estudar: **descobrir onde nosso tempo está sendo queimado durante o dia.**

A leitura no ônibus/metrô é outro modo de conseguirmos fazer mais, com “menos tempo”. Se colocarmos na ponta do lápis o tempo que gastamos em transporte coletivo perceberemos que estamos perdendo muito tempo útil da nossa vida. Se gastamos 1 hora e meia para ir para o trabalho e mais 1 hora e meia para voltar para casa, estamos perdendo 3 horas do nosso dia e quando colocamos isso na semana (5 dias úteis) estamos perdendo 15 horas da nossa vida semanalmente.

Para pessoas que não se sentem bem ao ler com o veículo em movimento, existem livros em áudio, vídeos de palestras e até cursos que podemos baixar e ir ouvindo no percurso. Podemos utilizar tudo isso como ferramentas para não perder nosso tempo na locomoção, mas o utilizar de maneira produtiva.

Se unirmos isso com uma hora de estudos em casa todos os dias conseguimos 20 horas de estudo por semana. Sinal de que o tempo está ali, nós só não estamos priorizando as coisas certas.

Nem todo mundo consegue esse tempo disponível, isso é fato, a rotina diária é muito diferente de uma pessoa para outra, por isso é importante **acompanhar onde estamos queimando o nosso tempo**, com aplicativos ou não, para identificarmos o que está sugando nossa vida e só depois de saber exatamente o que não é útil é que planejamos os estudos no tempo que vai “sobrar” sem o desperdício.

Pessoalmente recomendo acompanharmos nossa rotina natural durante 1 mês anotando tudo o que fazemos e só depois criarmos uma rotina produtiva e começar a segui-la.

## **6.5 Como se ajudar a receber ajuda (como fazer boas perguntas)**

Durante nossa carreira sempre vamos precisar de ajuda, seja buscando apoio de alguém ou procurando conteúdo e não importa o nosso nível profissional. O difícil é saber exatamente o que pesquisar quando estamos no começo da carreira. Quando estamos aprendendo aparecem erros de programação que raramente vamos saber resolver de primeira e é aí que começa o desafio de saber fazer a pergunta certa na internet ou para a pessoa ao lado.

Saber formular uma boa pergunta pode nos ajudar a receber ajuda mais rápido, afinal a outra pessoa vai saber exatamente do que precisamos e até as pesquisas na internet serão facilitadas com a questão correta em mãos. Então, como fazer boas perguntas sobre programação?

Algo que temos que ter em mente é: a outra pessoa não sabe o que estamos passando só porque ela é mais experiente ou porque está se dispondo a ajudar, nós precisamos contextualizar a pessoa e dizer exatamente o que estamos enfrentando.

A contextualização virá a partir do momento em que conseguirmos responder essas perguntas:

- O que eu estava fazendo? (ou o que eu já tentei fazer para resolver)
- O que eu esperava que acontecesse?
- O que aconteceu?

Vale lembrar que, se for uma pesquisa na internet, basta procurarmos por “o que aconteceu” e receberemos um resultado melhor sobre o nosso problema, porém, nos fóruns ou para nossos companheiros e companheiras de trabalho, devemos utilizar os recursos disponíveis, como mostrar uma imagem do erro que apareceu em nossa tela e compartilhar a parte de código que não está funcionando como desejamos.

Alguns recursos úteis para compartilhamento nos fóruns é o uso do imgur (<https://imgur.com>), para compartilhamento de imagens, e do Gist (<https://gist.github.com>) para compartilhar trechos do nosso código-fonte.

Essa dica é muito boa para quando estivermos recebendo um erro, mas e quando desejamos criar algo e não sabemos nem como começar?

Basta pensarmos nas seguintes questões e para formular a melhor pergunta:

- O que eu quero fazer?
- O que eu sei fazer?
- Eu já fiz isso alguma vez?
- Existe alguma referência na qual estou me baseando?

Essas questões nos ajudam a contextualizar a outra pessoa sobre qual o nosso nível de conhecimento e facilita até mesmo a colocação das palavras por parte dela para que tenhamos melhor entendimento do que ela vai nos dizer.

Sempre que formos fazer uma pergunta na internet devemos levar em consideração essas questões e utilizá-las para facilitar a nossa comunicação. O uso das imagens e compartilhamento do código é extremamente útil, uma vez que a pessoa pode analisar exatamente o que aconteceu e o que estávamos fazendo.

## 6.6 Conclusão

Não existe uma fórmula mágica da produtividade ou da aprendizagem ágil, como dizem por aí, pois tudo depende do nosso contexto e nem tudo se aplica para todas as pessoas. Por isso é extremamente importante que busquemos nos conhecer melhor e entender as nossas dificuldades, adaptando toda a nossa vida a essas variáveis.

Se soubermos onde estamos gastando nosso tempo e melhorar a forma como priorizamos as tarefas do dia conseguimos um nível de produtividade maior. Além disso, testar técnicas de estudos pode nos ajudar a encontrar o melhor meio de estudar para a nossa realidade.

Foram apresentadas algumas técnicas de estudos e a Técnica de Pomodoro, e seria muito bom testarmos essas dicas e analisar o que se encaixa em nossa vida, assim como buscarmos fazer as perguntas certas na internet ou para amigos(as) e colegas de trabalho.

Algo que sempre temos que ter em mente é que não existe uma bala de prata e que nem tudo o que falam por aí é correto. Não devemos acreditar em verdades absolutas. Devemos sempre pesquisar muito mais sobre um assunto do que acreditar na primeira fonte que encontramos. Ainda mais agora que vamos conhecer as comunidades de desenvolvimento de software.

## **CAPÍTULO 7**

# **No universo afora as pessoas precisam da nossa ajuda**

Algo extremamente importante, e que mais me anima no universo da programação, é o modo como as pessoas se importam em compartilhar conhecimento, seja algo de sua autoria ou até mesmo divulgando o conteúdo de outras pessoas. Foi graças a esse tipo de movimento que fui incluído no desenvolvimento de software e eu não poderia deixar de comentar sobre isso.

Existem muitos eventos, comunidades, blogs, canais e iniciativas para que a informação chegue ao maior número de pessoas no mundo inteiro.

Existem bons motivos para contribuírmos com essas iniciativas. A democratização do acesso à informação por si só já é relevante suficiente para que olhemos com bons olhos para o movimento das comunidades de programação.

Muita gente não possui recursos financeiros para adquirir um livro, os melhores cursos ou para estar em algum lugar em uma data específica de um evento. Uma convenção que aconteça em São Paulo, por exemplo, pode não ser acessível por pessoas de outro estado que não conseguem viajar e custear a hospedagem e alimentação no período do evento.

Podemos pensar: “Mas um e-book da Casa do Código custa 30 reais, atualmente!”, mas esse pode ser o valor que uma pessoa utiliza para custear os recursos básicos e não dispensáveis de sua vida, como uma conta de energia, a despesa de casa, conta de água etc.

No mundo afora, as pessoas mais carentes não têm a escolha de gastar dinheiro em um curso, pois seu objetivo de vida é um só: sobreviver. Realmente não tem como priorizar um curso quando se passa fome, mesmo que a longo prazo seja muito melhor.

Claro que existem exceções e exemplos de pessoas que passam pelas mais diversas situações para investir em sua carreira, às vezes deixando de comer para tal, porém não podemos tratar todo mundo como sendo iguais. Cada pessoa possui uma história e necessidades diferentes. O grande ponto é: as pessoas precisam de ajuda e **nós podemos ajudar**.

Às vezes, não nos achamos capazes de ajudar o mundo, mas a internet nos empoderou e hoje podemos mudar a vida de milhares de pessoas através de uma postagem nas redes sociais. Temos comunidades e pessoas utilizando esse recurso para chegar ao maior número de gente possível.

Mas o que é uma comunidade, afinal de contas? Vamos entender melhor o que são e quais são as comunidades de tecnologia que podemos começar a nos envolvermos desde o início da carreira e aprender maneiras de contribuir com elas e ajudar a mudar a vida das pessoas por aí.

## 7.1 Comunidades, eventos, fóruns e hackathons

Uma das maneiras mais legais de se aprender programação é em grupo. É quando nos juntamos para fazer um grupo de estudos sobre uma linguagem de programação, sobre uma tecnologia nova, compartilhar desafios e soluções ou ler um livro em conjunto.

A troca de experiência proporcionada pelos grupos de estudos é muito boa para nossa evolução profissional e também para que aprendamos mais em menos tempo. Isso pode acontecer de diversos modos, dentre os quais está a nossa organização em comunidades e fóruns.

As comunidades são grupos de pessoas que se unem, seja online ou presencial, por um interesse em comum, pela cultura do grupo, às vezes por histórias parecidas ou somente por objetivos iguais. Algumas comunidades nascem em volta de tecnologias, como linguagens de programação, ferramentas de desenvolvimento, e outras vezes por um objetivo foco, como compartilhar conhecimento.

Temos alguns exemplos de comunidades brasileiras que nasceram por um interesse em comum, como Python Brasil, PHP Brasil, JavaScript Brasil, Ruby Brasil, Java Brasil. Essas comunidades existem porque as pessoas se interessam por essas linguagens e querem trocar experiências, links relevantes, mostrar o que fazem e se ajudar a aprender mais sobre elas.

Outras comunidades não nasceram em volta de uma tecnologia, mas em volta de um objetivo, como compartilhar conhecimento independente da área ou tecnologia, como o Training Center, o Garoa Hacker Clube, Things Hacker Team ou o Code Club que são grupos que estimulam a troca de experiências e até o ensino de programação para crianças.

Existem grupos com forte apelo social, que levam conhecimento para grupos sub-representados, como pessoas carentes, mulheres, pessoas negras, pessoas com deficiência, indígenas, imigrantes etc., como o Codamos, Django Girls, Rails Girls, Afro Python ou WoMakersCode.

Se procurarmos na internet encontramos comunidades de todos os tipos, com as mais variadas filosofias e objetivos. Basta achar algo com que nos identifiquemos e nos unirmos a elas. Entrar em comunidades logo no início da nossa carreira pode ser de grande ajuda, pois a experiência dos outros pode facilitar demais a nossa vida e contribuir para nossa ascensão profissional.

Os fóruns de discussão são ferramentas utilizadas por essas comunidades para troca de conhecimento onde alguém posta uma pergunta e outra pessoa tenta ajudar a resolver seu problema. Esses fóruns podem existir dos mais diversos modos, como grupos em redes sociais ou plataformas próprias.

Já os hackathons são grandes competições de programação que acontecem, normalmente, com um objetivo legal, como resolver um problema social, ajudar uma comunidade ou tecnologia a crescer. Além disso, eles possuem premiação. Em hackathons, as pessoas formam times multidisciplinares que competem para trazer a melhor solução, que será avaliada por uma bancada alinhada com os objetivos da competição e com os critérios definidos.

Em todo caso, quanto mais participamos de grupos mais aprendemos. Mas é importante, além de recebermos esse conhecimento, que compartilhemos e

tentemos ajudar as pessoas que estão ali. Só assim as comunidades podem continuar existindo.

Por mais que, no começo, nos sentimos incapazes de ajudar, sempre conseguimos contribuir com as comunidades de alguma forma. Em questões de fóruns, por exemplo, podemos dizer se não entendemos uma pergunta, para que ela seja mais bem formulada, ou podemos tentar responder até onde sabemos. Não é preciso ir além do nosso conhecimento atual, mas já podemos ajudar com o que temos e, de acordo com que vamos participando mais vamos evoluindo profissionalmente e conseguindo contribuir de outras formas.

## **7.2 Gerando conteúdo para a internet**

O melhor modo de atingirmos a maior quantidade de pessoas, hoje em dia, é utilizando a internet. Sempre que precisamos aprender algo novo, nem que seja algo como “como fazer arroz”, corremos para a internet. Em desenvolvimento de software é o mesmo processo, principalmente para pessoas mais jovens, pois já crescemos no ambiente de pesquisa na internet, então já estamos mais acostumados com isso.

Um exemplo da força da internet é que mais de 90% das pessoas pesquisam na internet antes de comprar um produto, segundo um levantamento do TNS Research International, uma consultoria de pesquisas.

Por isso, um excelente modo de contribuir com as comunidades é gerando conteúdo, seja em texto, como blogs, ou vídeos, como os canais em plataformas como o YouTube, assim como em áudio, em podcasts.

Podemos contribuir com uma iniciativa existente, como o blog de uma comunidade, um portal, ou criar nossa própria. Isso depende mais do nosso objetivo pessoal (divulgar nosso nome ou contribuir com um grupo) ou se nos encaixamos em alguma filosofia dos grupos disponíveis.

A criação de conteúdo é algo bem pessoal, pois podemos falar sobre qualquer assunto que achamos importante ou que seja relevante para



mercado, como a tecnologia mais atual ou sobre nossas linguagens de programação.

Em 2018, temos muita gente falando sobre JavaScript, pois é uma linguagem em ascensão, assim como foi feito com Java, PHP e logo teremos outra linguagem na crista da onda, mas também temos muita gente que fala sobre algo que o mercado também precisa e que pode até estar sendo esquecido, como *soft skills*, que são as habilidades diferentes das habilidades técnicas, como comunicação, empatia etc.

Mas como uma pessoa iniciante pode contribuir com conteúdo?

Como iniciantes ou não, compartilhar sobre o que estamos aprendendo pode ser uma excelente maneira de reforçarmos o nosso conhecimento, pois vai demandar mais pesquisa e mais esforço para criarmos bons exemplos sobre algo que vamos falar. Além disso, podemos receber feedbacks sobre o que estamos transmitindo para melhorar ainda mais nosso entendimento sobre qualquer assunto.

A grande parte do mercado é formada de pessoas em início de carreira, conforme podemos confirmar na pesquisa do Stack Overflow, um site de perguntas e respostas sobre desenvolvimento de software, de 2018, em que constatamos que 30.1% das pessoas trabalham com programação entre 0 e 2 anos e 27.4% entre 3 e 5 anos (<http://bit.ly/stackoverflow-survey-2018>).

## Years Coding Professionally

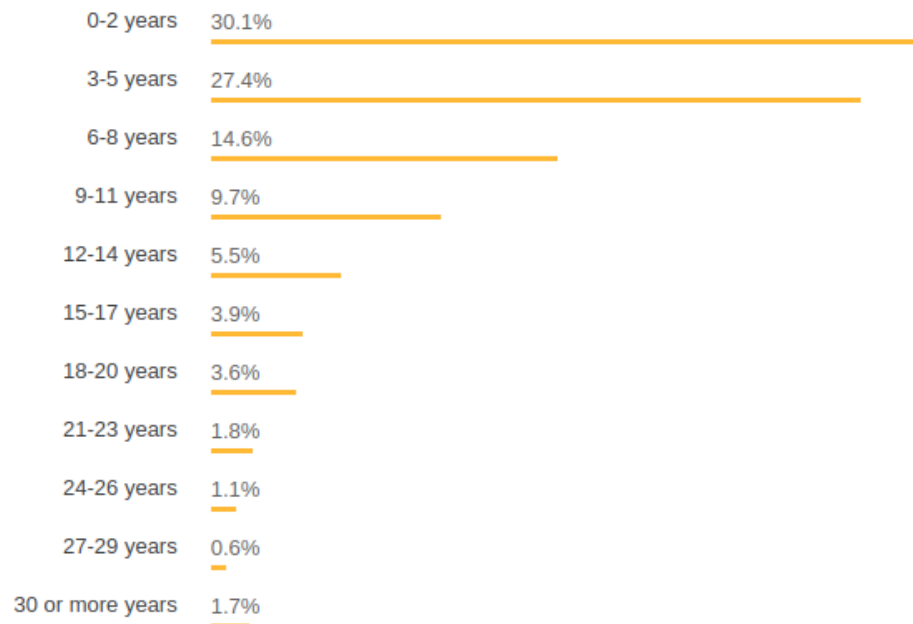


Figura 7.1: Stack Overflow Survey 2018

Então, se compartilhamos o que estamos aprendendo no começo da carreira podemos ajudar muita gente.

## 7.3 Compartilhando conhecimento através de palestras

Quando começamos a comparecer em eventos, logo nos impressionamos com as pessoas que palestram. Os assuntos falados, o modo como compartilham o que aprenderam, dentre outros pontos. O que quase nunca imaginamos é que nós também podemos fazer isso!

Palestrar pode ser um desafio imenso para algumas pessoas, pois apresentação em público não é algo que todo mundo gosta de fazer, porém pode ser extremamente recompensador pessoalmente, pois é onde temos o retorno mais rápido do que fazemos.

Enquanto estamos palestrando, vemos no olhar das pessoas o quanto estamos mudando suas vidas com o conteúdo que apresentamos. É muito mais rápido do que esperar um comentário em nossos artigos ou vídeos.

A melhor maneira de aprender a palestrar é seguindo um livro muito bom e conhecendo os vídeos do TED. O TED é uma conferência que nasceu para compartilhar ideias e ganhou o mundo com suas palestras incríveis. No livro *TED Talks - O guia oficial do TED para falar em público* aprendemos as melhores técnicas para palestrar. No seu canal do YouTube podemos conferir as melhores palestras do TED.

Com isso e nossas ideias, teremos tudo o que é necessário para começarmos a palestrar. Em seguida, basta procurarmos eventos e comunidades que gostaríamos de ajudar.

## 7.4 Eventos, comunidades e fóruns para nos envolvermos

Como já dito, existem diversas comunidades praticamente para todas as tecnologias ou áreas de trabalho. Para encontrar a comunidade ideal para nos ajudar em nossas necessidades podemos pesquisar visando as habilidades técnicas que queremos adquirir.

Inicialmente, é legal nos envolvermos mais com comunidades não focadas em uma tecnologia específica, pois elas são mais introdutórias ao mercado de trabalho do que as comunidades específicas. Isso porque no começo nós podemos nem saber qual tecnologia devemos aprender.

Caso já saibamos com o que desejamos trabalhar, como trabalhar com back-end utilizando a linguagem Python, então podemos procurar, **também**, uma comunidade de Python especificamente.

Então, no começo, é legal procurarmos comunidades como front-end Brasil, back-end Brasil, Android Brasil etc. Vamos conhecer algumas delas.

Fiz um levantamento das comunidades que eu conheço e de que participo, além de perguntar para algumas pessoas quais elas indicam para iniciantes.

## Comunidades generalistas

Comunidades focadas em ajudar pessoas de qualquer área de atuação ou tecnologia:

- **Training Center:** focada em ajudar pessoas a entrarem na área de desenvolvimento. (<https://trainingcenter.io/>)
- **WoMakersCode:** focada em ajudar mulheres a entrarem na área de desenvolvimento. (<http://womakerscode.org/>)
- **devbeers:** comunidade descontraída de desenvolvimento de software. (<https://devbeers.io/>)
- **freeCodeCamp:** comunidade internacional focada em inserir pessoas na área de desenvolvimento. (<https://freecodecamp.org/>)

## Comunidades por áreas de atuação

Comunidades focadas em discussão sobre alguma área de atuação específica:

- **Front-end Brasil:** comunidade brasileira da área de front-end. (<https://frontendbr.com.br/>)
- **Back-end Brasil:** comunidade brasileira da área de back-end. (<http://backendbrasil.com.br/>)
- **Android Dev BR:** comunidade brasileira da área de desenvolvimento Android. (<https://github.com/androiddevbr/>)
- **CocoaHeads:** comunidade internacional de desenvolvimento iOS. (<http://cocoaheads.com.br/>)
- **Data Hackers Brasil:** comunidade brasileira da área de big data, engenharia de dados e ciência de dados. (<https://datahackers.com.br/>)
- **GameDev Brasil:** comunidade brasileira de desenvolvimento de jogos. (<http://gamedev.com.br/>)

Essas são as comunidades que eu indico para qualquer pessoa iniciante. A partir daí, é ideal seguir para comunidades específicas para conversar com mais gente com interesse em comum sobre linguagens de programação ou sobre alguma tecnologia que desejamos focar em aprender, como procurar

por PHP Brasil, Python Brasil, Node Brasil, BrazilJS etc. Uma grande parte dessas comunidades estão nas redes sociais ou no GitHub.

No site do Codamos (<https://codamos.club/>) encontramos mais comunidades, eventos e, o principal, lugares seguros para pessoas de todos os níveis de conhecimento e sem preconceito com idade, classe social, mulheres, pessoas LGBTQI+, pessoas negras, pessoas com deficiência. Também temos o portal iMasters que agrega em sua agenda os eventos mais legais de desenvolvimento de software no geral (<https://imasters.com.br/>).

## **7.5 Conclusão**

Compartilhar conhecimento é extremamente importante, quando pensamos no contexto social, e pode ainda nos render uma grande evolução profissional se participarmos ativamente das comunidades, eventos e fóruns.

Podemos criar um blog para compartilhar conhecimento, assim como canais de vídeos, podcasts ou qualquer iniciativa que chegue a um grande número de pessoas que precisam de informação.

Devemos buscar um lugar onde nos sentimos bem em compartilhar e ajudar pessoas e então podemos nos unir para fortalecer o movimento além de aprendermos muito mais rápido quando em grupo.

## CAPÍTULO 8

# Conquistando o primeiro emprego com programação

A grande dificuldade de se conseguir o nosso primeiro emprego em programação é a mesma que em todas as áreas: darmos o passo inicial em direção às vagas de trabalho. É muito difícil acreditar que estamos preparados(as) para enviar um currículo ou clicar naquele botão de se candidatar que encontramos nos sites de empregos quando vemos os requisitos das vagas cheios de palavras que não conhecemos.

Mas, se não tentarmos, nunca vamos saber se estamos prontos(as) para o mercado e também não saberemos se aquela oportunidade se encaixava em nosso nível de conhecimento atual.

Enviar o currículo ou se candidatar a uma vaga para nossa área de atuação e não passar não é “queimar” uma oportunidade. Se não estivermos aptos(as) a entrar no emprego agora, a pessoa do recrutamento simplesmente não nos chamará para a oportunidade **agora**. O que não podemos fazer, de modo algum, é deixar de tentar.

Se não entrarmos dessa vez, então devemos continuar nos especializando e tentar novamente depois que adquirirmos mais conhecimento ou a experiência exigida pela empresa. — Isso se o nosso grande desejo for realmente trabalhar naquela empresa.

Ficamos com medo porque não sabemos onde procurar emprego, o que as empresas realmente querem de um(a) candidato(a) ou como conseguir a tal experiência que as empresas pedem para um **primeiro emprego**, às vezes para estagiários(as). Neste capítulo, vamos entender melhor sobre esses tópicos e um pouco mais.

Vamos entender como conquistar o emprego e não somente o primeiro emprego na área, mas como nos mantermos com boa empregabilidade, isto é, com mais chances de negociação futura em nossa carreira. Afinal,

trabalhar, para uma empresa ou não, é negociar o valor do nosso tempo pela nossa exclusividade com alguém.

## 8.1 Onde procurar emprego

Existem várias formas de se conseguir um emprego e, em desenvolvimento, uma das mais efetivas é ter uma presença online ou pessoalmente nas comunidades de programação. Isso porque o elemento mais poderoso que pode nos ajudar a fechar uma contratação rapidamente é uma boa **indicação**.

Se tivermos uma presença online, ou seja, participarmos de fóruns online, gerarmos conteúdo para a internet e participarmos das comunidades (eventos, conferências e outros modos que vimos no capítulo anterior), quando surgir uma oportunidade nas empresas as pessoas vão se lembrar de nós e podem nos indicar pelo conhecimento que demonstramos ter ou mesmo pelas características pessoais que elas enxergam em nosso comportamento por aí.

Mas, mesmo quem não deseja participar tanto das comunidades de programação consegue ter uma boa presença online. Hoje em dia o maior espaço de contratação, utilizado pelas empresas, de tecnologia ou não, é o LinkedIn (<https://http://linkedin.com/>).

O LinkedIn é uma rede social profissional onde colocamos nosso currículo à mostra, e outras pessoas, recrutadoras ou nossas conexões pessoais (alguém que nos conhece de algum lugar ou trabalhou conosco) podem nos adicionar. É como acontece no Facebook, porém com o foco em gerar uma rede de contatos profissionais para facilitar as contratações.

Um bom perfil no LinkedIn, um perfil com todas as informações que o site nos solicita e planejado para os requisitos das vagas, pode nos gerar muita visibilidade, como podemos ver no meu caso, considerado um perfil campeão somente por estar bem preenchido:



Figura 8.1: Meu painel do LinkedIn

No meu painel do LinkedIn vemos que tenho mais de 1000 visualizações no perfil em um mês e o perfil apareceu mais de 200 vezes nos resultados de pesquisa feitos no site para a minha profissão.

E, assim como o LinkedIn, existem outros sites de emprego legais para adicionarmos nosso perfil. No início da carreira, em quanto mais sites desse tipo nos inscrevermos, melhor! Não devemos nos limitar a ter um currículo em um local. Devemos mandar para o máximo de empresas, avisar o máximo de pessoas e preencher muitos sites de vagas para que o mercado saiba que estamos procurando emprego.

Não tem como o mercado e as pessoas empregadoras saberem que estamos procurando emprego se não fizermos barulho. Na próxima seção, vamos entender como conseguir um perfil campeão no LinkedIn e em outros sites de emprego para fazer esse tal “barulho” na internet.

As empresas também possuem páginas onde listam suas vagas que ficam, normalmente, em sua área institucional. Podemos entrar no site do Google, por exemplo, clicar em “sobre” e depois procurar a área “carreiras”. Na página de carreiras conseguimos encontrar as posições abertas ou um local para cadastrarmos nosso perfil e aguardar quando aparecer uma vaga. Então bastaria preenchermos e aguardar.

São esses os modos efetivos de se procurar um emprego na área de programação: fazendo uma rede de contatos com nossa presença online ou nas comunidades, tendo um LinkedIn devidamente preenchido, manter um cadastro nos sites de empregos e pesquisar oportunidades abertas nas páginas institucionais das empresas (e se candidatar, é claro).



Sabendo disso, precisamos agora aprender a criar um bom perfil no LinkedIn e outros sites, assim como aprender como entender os requisitos das oportunidades de emprego e mais alguns macetes para não cairmos em furadas em nossa primeira experiência na área de programação.

## **8.2 Como aparecer nas buscas de emprego**

Mesmo como iniciantes na área devemos deixar nosso currículo e perfis em sites de emprego como o de alguém que busca uma oportunidade na área de desenvolvimento de software, afinal essa é a área em que queremos trabalhar. Ainda que a nossa experiência de trabalho seja com algo não relacionado com programação, é importante deixar claro que temos muito interesse nisso.

Para alguém que já trabalha na área, é um pouco mais fácil deixar o perfil de acordo com o que recrutadores precisam, afinal já possui as habilidades que o pessoal de recrutamento mais procura. Porém, mesmo assim, é importante colocar informações com estratégia, não somente colocar tudo o que sabemos. Deixar um currículo com 5 páginas e com informações pouco relevantes ao recrutamento não será nada efetivo.

Quando alguém analisa o nosso perfil, está procurando por palavras-chaves e termos relacionados a uma posição específica na empresa ou em um cliente e, ao se deparar com um currículo imenso com informações vagas, sem valor para a área, dificilmente vai passá-lo para a próxima fase.

Se estamos procurando por pessoas para uma vaga de front-end, vamos procurar por: HTML, CSS, JavaScript, React, Angular, Vue.js, tecnologias que estamos utilizando em nosso software. Se é para vagas de back-end, DevOps, engenheiros(as) ou analistas de dados, Mobile ou outras oportunidades, funcionaria do mesmo jeito: pesquisamos no currículo ou perfil pelas tecnologias e habilidades que achamos necessário para aquela oportunidade em questão.

Ao montar nosso currículo devemos pensar nele como uma vitrine que servirá para vender nosso produto, que é o nosso trabalho, então devemos colocar exatamente o que queremos que a pessoa saiba de nós e o que se encaixa para aquela empresa. Por exemplo: se é uma empresa de publicidade online, será muito importante ressaltarmos se temos conhecimento em SEO (otimização para motores de busca), pois eles precisam aparecer no Google e outros buscadores, se é uma empresa de robótica, então eles precisam saber se temos o conhecimento em inteligência artificial, eletrônica, e por aí vai.

Uma boa estratégia é montar vários currículos de acordo com as empresas em que deseja entrar. Não compensa enviarmos o mesmo currículo ressaltando nosso conhecimento em redes e administração de servidores, ou desenho artístico e publicitário, para uma empresa de varejo, pois isso não será muito útil para a empresa.

Já quando falamos do perfil em sites de emprego, normalmente, só conseguimos criar um perfil. Nesse caso é extremamente importante que deixemos claro qual o nosso objetivo e quais são as nossas habilidades.

### **O que colocar e o que não colocar em um currículo para programador(a)**

Estamos montando nossa vitrine então é normal querermos mostrar **tudo** o que sabemos fazer, mas já sabemos que isso não será relevante para o recrutamento. Devemos enxugar o currículo e colocar somente o que é necessário, mas como vamos adivinhar esse “o que é necessário”?

Para saber o que colocar no perfil é importante analisar cada situação e também os nossos objetivos profissionais. E a maneira mais prática de se fazer isso é olhando várias vagas de emprego antes de preencher o nosso currículo.

Com essa pesquisa pelas vagas, vamos identificar as habilidades que as empresas estão procurando e, se realmente as possuímos, então vamos colocando-as em nosso currículo.

Pessoalmente já descobri que algo que eu não achava tão importante destacar no currículo era muito procurado pelas empresas nesse tipo de análise de perfil que era a questão de participações em comunidades de programação.

Em um currículo é importante colocar os nossos projetos para que ganhem destaque pela nossa experiência. Como projetos open source, algo que fizemos para amigos(as) ou iniciativas de que participamos. Isso ajuda a pessoa de recrutamento e seleção a identificar a nossa experiência e também nossos interesses individuais.

Colocar as comunidades de que participamos de maneira ativa e em quais projetos contribuímos é muito bom, visto que isso vai demonstrar mais da nossa experiência com trabalho em equipe ou mesmo habilidades técnicas dependendo de como contribuímos com as comunidades.

O que não devemos colocar, a não ser que seja solicitado, são informações ambíguas como porcentagem de algo que dominamos, foto de perfil ou redes sociais que não demonstram nada do nosso perfil profissional (como um Snapchat, que é somente para compartilhar momentos legais de nossa vida com nossos amigos e amigas).

Vamos pensar nesse exemplo de porcentagem: uma pessoa procurando emprego como back-end coloca que domina **75% de JavaScript**.

O que seria 75% de JavaScript ou qualquer outra linguagem de programação? Seria ter lido 75% da documentação, conhecer 75% das bibliotecas que existem, dominar 75% de lógica de programação com essa linguagem?

Percebemos que algo desse tipo, ao invés de gerar confiança em nosso perfil, só gera dúvidas.

Colocar que conhecemos uma linguagem de programação X ou uma tecnologia já é suficiente para despertar interesse no recrutamento para um contato por telefone ou por e-mail para que seja questionado o que realmente dominamos sobre aquilo.

Também não é interessante encher o nosso currículo de cursos que não têm relação nenhuma com a vaga que estamos concorrendo. Eu já fiz curso de desenho artístico e publicitário e também já dei aulas de patinação e isso não é nada importante para alguém que quer saber se eu conheço programação.

Algo que pode ser interessante adicionarmos em nosso currículo, e que o próprio LinkedIn recomenda, é um resumo de nosso conhecimento logo no início. Esse resumo pode poupar tempo do recrutador ou recrutadora e ainda despertar interesse para que essa pessoa continue a leitura do nosso currículo.

No meu perfil no LinkedIn e no meu currículo em PDF, eu deixo assim, atualmente:

Experiência em desenvolvimento de SPAs (Single Page Applications) com Angular e React, module bundlers (webpack), gerenciadores de pacote (NPM, PIP), transpilers (Babel), pré-processadores CSS (Sass e Stylus), task managers (Grunt, Gulp, NPM Scripts, Shell Script), arquiteturas CSS (BEM, SMACSS), Git, Web Performance, SEO, Acessibilidade e dados estruturados (Schema.org) Sou apaixonado por open source, Linux, comunidades de tecnologia, JavaScript, mobile, arquitetura de software e cultura ágil. Habilidades pessoais: treinamento de equipes e mentoria de desenvolvedores iniciantes, gerenciamento de comunidades e eventos, facilidade de comunicação em público e escrita de artigos técnicos.

Neste começo eu já destaco o que eu sei fazer, o que eu gosto de fazer e as habilidades não técnicas que acredito serem importantes para as oportunidades de emprego, já com foco na minha área de atuação, que é front-end.

## **Perfil campeão no LinkedIn**

O LinkedIn, por si, nos ajuda a criar esse perfil campeão. Mas acabamos ou não valorizando as informações que a rede nos pede ou mesmo não

acreditamos que devamos colocar algo lá.

Por exemplo: o que colocar nas competências?

Entra sempre o foco da nossa carreira. Se estamos procurando emprego na área de desenvolvimento, devemos fazer aquela pesquisa e colocar o que sabemos e que as empresas pedem.

Quando estamos estudando algo ou temos muito interesse naquilo, também é legal colocar ali, pois poderemos dizer na nossa introdução ou quando alguém entrar em contato conosco qual é o nosso nível de conhecimento em cada competência.

O que não devemos fazer, no LinkedIn, é colocar em campos específicos o que a rede não pede. Por exemplo, no **título profissional** o LinkedIn nos solicita uma informação “qual o seu título profissional”. Não devemos colocar ali algo como “buscando recolocação no mercado”, pois nosso perfil não será encontrado nas buscas quando as pessoas de recrutamento pesquisarem pelo título “desenvolvedor(a) xyz”. Também não devemos colocar nossas certificações no título, algo como “CEH, Security, LPIC-2, ITIL, COBIT”, para isso o LinkedIn tem uma seção específica que é a seção conquistas. Já vi esse tipo de informação no nome das pessoas e isso não é nada efetivo, visto que na hora que alguém faz uma busca, ela filtra pelas conquistas e não pelo nome do usuário na rede.

### **Como citar experiências anteriores no currículo**

Recrutadores costumam dizer que devemos colocar as últimas 3 experiências profissionais no currículo/LinkedIn, pois isso é suficiente para saber nosso nível profissional e em quais tipos de projetos já trabalhamos.

Algo legal de se fazer ao descrever um emprego anterior é colocar uma breve descrição do que é a empresa/produto em que trabalhamos e quais as frentes em que atuamos, ou seja, quais os produtos em que realmente fizemos algo.

Uma empresa pode ter milhares de produtos/software e não atuamos em todos eles e por isso é bom colocarmos exatamente onde trabalhamos para

que não seja criada uma expectativa em cima do nosso perfil e quando recrutadores entrarem em contato essa expectativa ser quebrada, parecendo que mentimos no nosso perfil.

Imagina que trabalhamos em um e-commerce, podendo ser no aplicativo, no back-end, no front-end, com big data ou algo do tipo. Nós podemos atuar em várias frentes dentro desse e-commerce, como na área de logística, na área de recomendação de produtos, em perfis de usuários, em segurança da informação etc.

Exemplo de descrição com explicação:

Imaginemos que coloquei no meu perfil (como profissional front-end) que trabalho/trabalhei em um e-commerce, a empresa x. Na descrição dessa citação eu teria colocado algo como:

A empresa x é líder em vendas no Brasil e possui mais de 1 milhão de usuários acessando o site 24 horas por dia, tem em sua filosofia a cultura ágil e foco em inovação utilizando as tecnologias mais atuais para trazer o melhor produto ao usuário final. Eu trabalho/trabalhei no time de recomendações, onde eu desenvolvo as interfaces de representação de dados e também os scripts para inserir os produtos recomendados na home page e detalhes de produtos do site. Meu principal desafio neste produto é criar scripts performáticos, dada a imensa quantidade de dados que preciso exibir na tela e o risco de travar o navegador com esse processamento. O meu time recebe as demandas vindas do time de negócios, e trabalho em conjunto com a pessoa de design da minha equipe para entregarmos valor.

Esse tipo de descrição vai nos ajudar em: a pessoa de recrutamento vai conhecer qual o tipo de produto em que trabalhamos, nesse caso um e-commerce com grande quantidade de acessos e líder de mercado, passando a sensação de que sabemos o peso de uma mudança brusca no sistema ou uma queda da plataforma; assim como saber como era o trabalho em equipe nessa empresa, visto que já comentamos como chegam as demandas e como é a interação em time. Também vai conhecer um pouco mais do que nós

dominamos tecnicamente, visto que já colocamos qual a dificuldade técnica que temos no dia a dia.

Claro que nesse exemplo eu coloquei muito texto, mas é para demonstrar que temos como explicar melhor o que fazemos em uma determinada posição de trabalho. Devemos tentar ser o mais direto possível.

Exemplo da mesma descrição, porém com menor quantidade de texto:

A empresa x é um e-commerce líder de mercado com cultura ágil e foco em produtos inovadores. Eu trabalho/trabalhei no time de recomendações onde desenvolvo interfaces de representação de dados e scripts de inserção de recomendações na home e detalhes de produtos com demandas dos times de design e negócios.

Uma descrição dessas, além de facilitar o trabalho de reconhecimento das nossas habilidades, ainda ajuda que as pessoas entendam melhor o que realmente fazemos em desenvolvimento de software.

### 8.3 Desmistificando vagas de emprego e escapando de ciladas

As vagas de emprego da área de desenvolvimento podem nos assustar, **e muito**, no início de carreira, pois, além de a maioria dizer que procura pessoas com experiência, ainda temos os requisitos técnicos em forma de listas imensas.

Já adianto que não devemos nos assustar com isso. Nós **não precisamos possuir todos os requisitos** de uma descrição para nos candidatarmos à vaga. É a crença de que é necessário 100% dos requisitos preenchidos que nos limita e faz com que não enviemos o currículo para uma vaga com que tanto sonhamos.

Quando a pessoa de recrutamento entrar em contato conosco, ela vai confirmar o que sabemos e o que não sabemos. Se não cumprimos todos os

requisitos, o importante neste contato é demonstrar que temos interesse em aprender, o que não dominamos daquela lista de requisitos e a empresa, se tiver boa cultura e estiver em um bom momento de contratação (quando não estão desesperados por profissionais em nível sênior para apagar incêndios nos softwares), por acabar nos contratando e ajudando a desenvolver o que faltar.

O grande risco nas vagas de programação é que podemos identificar direto nas próprias descrições são as vagas para determinada função e que pedem habilidades de outra nada parecida com o que fazemos.

Exemplo: existe uma vaga para o perfil front-end, mas quando analisamos os requisitos vemos que, na realidade, o que a empresa procura é alguém full-stack. Sabemos, visto os capítulos anteriores, que front-end tem uma atribuição e full-stack tem outra.

**Desenvolvedor Front End em Santana de Parnaíba - SP**

**Sobre a vaga**

**Salário**

- A combinar

**Descrição**

- Área e especialização profissional: Informática, TI, Telecomunicações - Programador / Desenvolvedor
- Nível hierárquico: Especialista
- Local de trabalho: Santana de Parnaíba, SP
- Regime de contratação de tipo Efetivo - CLT
- **Desenvolvedor Full Stack web Análise de sistemas** elaboração de solução técnica modelagem de dados e desenvolvimento de GUI teste e documentação de funcionalidades Trabalhar com times multidisciplinares envolvendo diversas áreas da empresa local de trabalho: Alphaville Contratação CLT Preferência experiência em projeto Utilities interessados enviar cv com pretensão salarial

**Exigências**

- **Programação: ASP.Net, C#, Dot Net, Java, JavaScript, jQuery, VB.Net**

**Benefícios adicionais**

- Assistência médica, Assistência odontológica, Auxílio creche, Refeição no local, Seguro de Vida, Vale-alimentação, Vale-refeição, Vale-transporte



Figura 8.2: Vaga cilada

Na imagem vemos que o título é front-end, mas na descrição está falando especificamente full-stack e tem como requisitos bancos de dados e tecnologias de back-end. Habilidades que não deveriam ser cobradas de uma pessoa que iria trabalhar somente com front-end.



Esse título não condizente com a descrição da vaga pode tanto ser um erro da área de recursos humanos da empresa quanto uma vaga cilada, aquele emprego onde recebemos 1x por termos registro em carteira como uma profissão, mas deveríamos receber 3x, pois desempenhamos outra função.

Não tem como adivinhar qual é a real intenção de quem escreveu a vaga e por isso partimos do pressuposto que alguém errou no cadastro, mas é muito importante questionar qual a real atribuição de quem vai desempenhar essa função dentro da empresa. Quando a pessoa de recrutamento entrar em contato conosco, devemos questionar bastante para descobrir isso e, se for uma vaga cilada, dizemos que não estamos interessados(as) no momento.

## **8.4 Como saber se é uma boa empresa para se trabalhar**

Principalmente em início de carreira não conhecemos as empresas legais e as empresas não muito legais de se trabalhar, mas existem ferramentas que nos ajudam a identificar bons locais para onde enviar o nosso currículo.

Os sites de oportunidades de emprego, normalmente, possuem filtros baseados em recomendações ou avaliações de usuários que já trabalharam nas empresas, mas no Brasil o site que mais consultamos para saber como anda uma empresa é o Love Mondays (<https://lovemondays.com.br/>). Também existe a pesquisa anual do Great Place to Work (<https://gptw.com.br/>). Eu, pessoalmente, prefiro o Love Mondays pois podemos ler um comentário anônimo de alguém que trabalhou lá e vai dizer como exatamente qual foi sua experiência a partir do seu ponto de vista.

O Love Mondays também lista as melhores empresas para se trabalhar baseado nas avaliações que recebe durante o ano. Para exemplificar, no ano de 2018 temos a empresa ThoughtWorks como melhor empresa para se trabalhar segundo as avaliações dos funcionários.

Mas, claro, não devemos levar somente esses dados em consideração, pois a empresa que entrou em contato conosco pode não estar listada no Love

Mondays porque nunca recebeu nenhuma avaliação, mas ser uma boa empresa.

O melhor modo de saber se a empresa é realmente legal de se trabalhar é entrando em contato com alguém que conhecemos e que trabalha lá e questionando o que essa pessoa acha do local. Caso não tenhamos ninguém conhecido na empresa pode não ser legal entrar em contato com alguém direto pelo LinkedIn, pois pode ser bem invasivo. — Eu sei que você já tinha pensado em sair chamando todo mundo no LinkedIn

Se não tivermos informação em nenhum local da internet sobre a empresa que nos contatou, realmente, teremos que arriscar. Podemos olhar o site da empresa, o que eles mostram sobre sua cultura, sobre sua filosofia, missão, valores e torcer para que o que eles dizem no site seja a realidade da empresa. Quem sabe não tenhamos uma surpresa com essa empresa e não possamos ser a primeira pessoa a fazer uma bela avaliação dela no Love Mondays, assim como fizeram um dia sobre a ThoughtWorks, não é?

## **8.5 Usando empregos ruins como propulsores**

O primeiro emprego ou os nossos empregos atuais não serão os últimos de nossa carreira, provavelmente. Empregos ruins farão parte da nossa vida, afinal nem sempre vamos acertar ou nem sempre teremos as habilidades necessárias para o melhor emprego em um momento de necessidade.

Conseguir um emprego legal pode variar dependendo da nossa capacitação assim como do nosso momento de vida. Às vezes passamos por momentos de vida em que precisamos recorrer a uma oportunidade não muito boa para nos mantermos, afinal todos pagamos boletos, mas como extrair coisas boas de empregos ruins?

Em toda oportunidade podemos aprender algo novo. Em toda empresa existem pessoas que podem nos ensinar alguma coisa, nem que seja sem querer. Às vezes aprendemos por osmose, só por estar perto de alguém que trabalha bem. Até por isso existe uma premissa que é constantemente

disseminada nas palestras sobre carreira: “não seja a melhor pessoa de uma equipe”. Quando somos iniciantes ou não somos os melhores em algo podemos aprender muito com as pessoas de qualquer empresa!

Precisamos exercitar a nossa resiliência e buscar tirar algo bom de qualquer oportunidade que tivermos. Isso vai nos garantir grandes qualificações para nosso futuro profissional.

Na área de programação, muitas vezes, achamos um emprego ruim porque não temos oportunidade de usar a última tecnologia de mercado ou porque não podemos “matar” um software antigo que é ruim de dar manutenção. Por que não encarar isso como desafios para melhorarmos profissionalmente?

Já deixemos claro: nem sempre vamos trabalhar com a melhor tecnologia de mercado ou a mais atual que todo mundo diz ser a melhor e nem com softwares muito bem escritos, onde temos muito prazer de alterar algo nele. A maioria das oportunidades de emprego são para trabalharmos com um software já existente e darmos manutenção nele e esse pode ter sido muito bem escrito ou não.

O que podemos fazer é: propor soluções e trazer dados do real valor de se utilizar as ferramentas novas, aprender a traçar planos de reescrita de softwares legados (os softwares antigos que continuam essenciais para um negócio) ou mesmo aprender a criar novos softwares que vão suprimindo as necessidades com o tempo até que não precisemos mais do legado. Ou seja, quase sempre conseguimos extrair boa experiência de empregos “ruins”.

Claro, nem toda empresa vai nos ouvir ou nos dar poder suficiente de reescrever um software ou que mudemos as ferramentas internas, frameworks ou libs que utilizamos para trabalhar ou algo do tipo, mas sempre vale a pena *tentar* fazer isso antes de sair de um emprego.

## **8.6 Como conseguir experiência sem nunca ter trabalhado em uma empresa**

Como requisito das vagas de emprego sabemos que sempre vem a palavra **experiência** em tecnologia xyz, mas como conseguir experiência para o primeiro emprego se não conseguimos entrar em neste primeiro emprego sem experiência? Nós acabamos de cair no loop de inconsistência do time de recursos humanos.

A resposta está em: projetos pessoais, projetos open source ou projetos gratuitos.

Muita gente sente uma dor no coração quando falamos em trabalhar de graça, mas, sim, em determinadas circunstâncias podemos precisar trabalhar de graça para conseguir a tal experiência, ainda mais dependendo do produto a ser desenvolvido.

A experiência de criar um software de gestão para uma microempresa local (do nosso bairro) é muito boa para o nosso currículo, mas muitas vezes essas empresas (normalmente mercadinhos, vendas, cabeleireiros) não possuem a renda para pagar por um software ou mesmo não veem valor em instalar um software visto que trabalhar com agendas e papel tem funcionado para eles durante anos.

Então o modo de convencer esses(as) microempresários(as) a aceitarem nosso trabalho (primeiro trabalho, ainda por cima) é se propor a fazer algo gratuito. Se tivermos a sorte de o local ter um bom faturamento até podemos cobrar um valor abaixo do mercado, afinal é nosso primeiro projeto.

Neste caso, vale lembrar que nosso objetivo com esse trabalho não é ganhar dinheiro, mas conseguir a tal experiência que o mercado tanto pede para conquistarmos nosso primeiro emprego, onde, sim, vamos ganhar dinheiro e depois vamos crescer e ganhar mais ainda. Pensemos nisso como um investimento de longo prazo, se formos pensar em “ganhar dinheiro”, e de curto prazo se o lucro por esse trabalho gratuito for conseguirmos o primeiro emprego.

Não somente fazer um projeto, para uma microempresa, mas podemos ir de comércio a comércio, bazar a bazar etc., depois de fazer nosso primeiro projeto, para tentar fazer mais softwares ainda e conseguir mais

experiência. Com isso, estaremos montando nosso portfólio sem nunca ter trabalhado formalmente em uma empresa.

Pensando em cada contexto:

- Para a área de web, podemos criar sistemas web, sites institucionais.
- Para a área de mobile, podemos criar o aplicativo para a empresa.
- Para a área de dados, podemos criar projetos que melhorem o faturamento da empresa.
- Para a área de desktop, podemos criar os softwares PDV (aqueles de caixa de mercado).

Para construção desses projetos podemos utilizar o que já sabemos ou mesmo arriscar algo novo para aprender. Se queremos entrar em área x, então pegamos um projeto em que vamos precisar dessas habilidades para desenvolver o produto.

As comunidades de programação também podem nos ajudar nessa primeira experiência e ainda nos ajudar a formar a nossa rede de contatos, afinal trabalhamos em time e conhecemos mais gente ainda através do open source. Podemos pegar projetos na internet, no GitHub, GitLab ou BitBucket e começar a ajudar eles, resolvendo os problemas que estiverem relatados contribuindo com um projeto já existente ou criando algo do zero para a comunidade.

Nos projetos abertos (open source) entramos no mesmo contexto: procurar onde ajudar e encaixar a nossa capacidade ou então arriscar algo novo. Existem muitas comunidades com projetos legais que poderiam se tornar aplicativos ou mesmo que não têm nenhum dado para se direcionar e seria um bom caso para análise de dados, assim como a necessidade de apoio em seus sites.

Assim como o open source, temos diversas ONGs precisando de ajuda na internet. Quando entramos nos sites de algumas ONGs sentimos até uma tristeza de quem depende daquele site para alguma coisa, pois encontramos sites com uma experiência de navegação ou visual muito ruins, sites pesados, aplicativos lentos ou mesmo entidades sem nada disso e que se beneficiariam bastante se ajudarmos. Nesse caso, além de ganhar

experiência e ter um grande projeto para mostrar em uma entrevista/em nosso currículo, ainda ajudamos uma instituição que dá um bom retorno para a sociedade.

Alguns sites onde podemos encontrar ONGs para entrarmos em contato:

- ongsbrasil (<http://ongsbrasil.com.br/>)
- voluntariado (<http://voluntariado.org.br/LP/>)
- e-solidario (<https://e-solidario.com.br/>)

E a última, mas não menos importante, maneira de conseguir a experiência sem ter um emprego formal é criar projetos pessoais. Nós temos muitas ideias de criação de softwares quando começamos a entender como eles funcionam e como são criados. Podemos usar essas ideias para praticar desenvolvimento.

Para quem faz controle financeiro, recorrentemente passamos tempos e mais tempos tentando adaptar uma planilha à nossa necessidade de preenchimento e isso pode acontecer porque a planilha já não é mais o que precisamos, queremos algo mais **personalizado** para o nosso estilo de uso. Esse seria um bom caso para aplicarmos nosso conhecimento: quando algum software já não nos atende mais.

Quando os softwares não nos atendem é um excelente caso para praticar, pois podemos iniciar o nosso desenvolvimento partindo do ponto já existente, ou seja, das telas, do fluxo lógico, de toda a parte que funciona bem para o nosso caso no software.

Desses modos de se praticar, o que eu mais gostei no início da minha carreira e também indicaria para qualquer pessoa é criar algo para um cliente. Esse cliente pode vir daquelas microempresas locais ou mesmo um parente ou conhecido. No meu caso eu pegava freelances para fazer em sites de freela, que vou comentar mais para a frente no livro.

Eu gosto disso porque temos a pressão do prazo, temos que analisar o caso real de um cliente, entrar na cabeça da pessoa e aprender a resolver problemas. Ajudar ONGs também entra nesse mesmo patamar, afinal teremos as pessoas responsáveis pela ONG como um cliente que nos

fornecerá prazos e também a questão de análise dos requisitos do projeto que vamos desenvolver.

Estes são os cenários mais parecidos com o ambiente de trabalho, com a diferença de não ganhar nada ou ganhar pouco dinheiro. Se conseguirmos um projeto em que possamos envolver uma equipe, melhor ainda, pois se aproxima mais de um ambiente de trabalho real.

## **8.7 Qual o salário de programadores(as)**

Antes de aceitar um emprego, devemos analisar os prós e contras da empresa e da vaga. O salário está entre os prós, é claro. O trabalho assalariado é uma troca: nós produzimos e a empresa nos paga pela produção. É uma troca simples e deve ser justa.

O salário de desenvolvedores(as) de software não é o mais alto que existe, porém está longe de ser o mais baixo. É um bom salário e é bem mais alto do que a média salarial da população brasileira. Pelo menos em grandes centros e boas empresas conseguimos salários inimagináveis por algumas pessoas — por mim mesmo quando comecei na área.

Isso pode variar muito de região para região e muitas vezes até de cidade para cidade em um mesmo estado e por isso é bem importante uma pesquisa profunda antes de aceitar uma proposta salarial ou mesmo de dizer qual a nossa proposta.

Para saber qual o salário que as empresas estão ofertando devemos fazer uma grande pesquisa e obter o máximo de valores para entender realmente o cenário em que estamos atualmente.

Podemos abrir uma discussão em um fórum, perguntar para amigos(as) que trabalham na área ou pesquisar em sites de vagas, que são especializados no assunto e sempre têm uma base de dados legal para comparação baseado na oferta das empresas ou na declaração dos próprios funcionários.

Eu montei uma tabela com a média salarial baseada nos resultados da Love Mondays, em março de 2018. Para cada profissão que comentei, eu fiz uma pesquisa pelos salários e agrupei os resultados. Basicamente entrei em [lovemondays.com.br/salarios/cargo](http://lovemondays.com.br/salarios/cargo) e fui coletando os dados para alimentar nossa tabela. Para cada área eu peguei o valor do salário em início de carreira, como júnior, já com boa experiência, como pleno e por fim o cargo “mais alto”, o cargo de sênior. Vamos conferir os resultados.

Cargo	Salário
front-end júnior	R\$ 2.743
front-end pleno	R\$ 4.569
front-end sênior	R\$ 6.239
back-end júnior	R\$ 3.029
back-end pleno	R\$ 4.942
back-end sênior	R\$ 7.641
android dev júnior	R\$ 3.307
android dev pleno	R\$ 5.626
android dev sênior	R\$ 9.035
ios dev júnior	R\$ 2.834
ios dev pleno	R\$ 5.586
ios dev sênior	R\$ 8.978

Para as demais profissões (engenharia e ciência de dados, jogos, infra) não achei uma quantidade de dados significativa, porém encontrei para o cargo “Software Developer”, que pode ser considerada nossa profissão full-stack:

Cargo	Salário
software dev júnior	R\$ 4.424



Cargo	Salário
software dev pleno	R\$ 6.476
software dev sênior	R\$ 9.571

Vale lembrar que essa tabela é só um exemplo para comparação básica e não levou em consideração a região onde moramos, tecnologias com as quais trabalhamos, anos de experiência ou outros detalhes que podem aumentar ou diminuir nosso valor no mercado de trabalho.

Se acrescentarmos tecnologias específicas, como, por exemplo, procurar por “desenvolvedor(a) java” no Love Mondays a tabela ficaria assim:

Cargo	Salário
java dev júnior	R\$ 3.020
java dev pleno	R\$ 5.198
java dev sênior	R\$ 8.614

Percebemos que muda um pouco. Porém é Java para o quê? Android? Back-end? Ainda fica vago e por isso sempre devemos fazer uma pesquisa mais ampla, com mais dados para uma boa comparação.

Em todo caso, o salário é negociável e devemos discutir a oferta com a pessoa que nos entrevista. O ponto principal é: pesquise muito em sites especializados, com conhecidos(as) e nas comunidades de programação para conhecer bem o mercado.

## 8.8 Como é um processo seletivo para programadores(as)

O processo seletivo para pessoas desenvolvedoras de software é o mesmo de outras profissões: alguém vai pesquisar nosso perfil, vai entrar em contato conosco para perguntar se queremos participar da seleção e então inicia-se o processo.

A grande diferença está no teste prático, que pode vir antes ou depois da primeira parte ou mesmo ser dividido em vários testes práticos, ser somente uma prova na empresa ou em algum site de desafios de programação.

As entrevistas pessoais, assim como em outras profissões, servem para analisar nosso perfil e o fit cultural com a empresa, ou seja, se nos adaptamos à filosofia, aos valores e à missão da empresa.

Já o teste prático serve para analisar nosso nível de conhecimento técnico e como tomamos decisões de engenharia de software, arquitetura e boas práticas de programação. Tudo isso nós aprendemos com o tempo, então, para vagas de iniciantes, a cobrança maior é em saber se conseguimos entregar a solução para o problema proposto e se temos disposição para aprender.

Como dito anteriormente, não precisamos cumprir todos os requisitos técnicos para nos candidatar em uma oportunidade de emprego justamente porque agora é que ficará visível o que sabemos ou não.

O mais importante no teste prático é não querer fazer algo muito a mais do que já sabemos, visto que a maioria dos testes são feitos em casa e com um prazo de no mínimo 4 dias para entregar. Não é nada legal pesquisar tudo de última hora, fazer aquele teste lindo, sendo que realmente não sabemos/não dominamos aquilo que adicionamos no teste, pois depois teremos que explicar porque tomamos cada decisão e ficará claro que não sabemos realmente o que estávamos fazendo.

Melhor um teste bem-feito com o que realmente conhecemos do que um teste mal feito com algo que adicionamos somente para parecer que sabemos.

Claro que temos pessoas que aprendem tudo de última hora para fazer o teste prático de uma seleção e acaba se dando bem, mas fica a nosso cargo se queremos arriscar ou não.

## **8.9 Como se preparar para testes práticos de programação**

O exercício para conseguirmos fazer testes de programação, sem muito sofrimento além da ansiedade pela oportunidade de emprego, é criar projetos. É realmente programarmos. Quanto mais linhas de códigos escrevermos em nossa vida para projetos diferentes, mais estaremos preparados(as) para os testes práticos das empresas.

Não adianta muito decorar diversas páginas de um livro técnico, ler milhares de artigos todos os meses e não praticar nada daquilo, pois esqueceremos tudo na hora do nervosismo do teste. A prática é a única aliada na hora da preparação para as entrevistas técnicas.

Existem sites especializados em praticar programação, como:

- hackerrank (<https://hackerrank.com/>)
- codewars (<https://codewars.com/>)
- exercism (<http://exercism.io/>)
- urionlinejudge (<https://urionlinejudge.com.br/>)

Vale ressaltar que esses sites, além de servir para pessoas que já sabem programação e só procuram melhorar suas habilidades e/ou se preparar para testes de empresas, também possuem guias para aprendermos programação do zero através da prática dos exercícios.

Caso o inglês seja uma barreira, o urionlinejudge.com.br possui versão em português.

Estes exercícios por si só nos preparam somente para testes de lógica e muitas empresas fazem testes práticos em formas de projetos inteiros, como “criar uma interface de loja virtual”, “criar uma aplicação back-end que se comunique com parceiros”, “criar um aplicativo para listar os últimos jogos do seu time”, “criar um modelo para analisar os dados da base xyz”, “criar um jogo da força”. Então é ideal que realmente pratiquemos muito criando coisas novas, que coloquemos nossas ideias em prática e desenvolvemos softwares.

Caso formos bem nos testes de perfil e teste prático, normalmente existem entrevistas com pessoas dos times e até mesmo com a diretoria da empresa

para que confirmar o alinhamento com as expectativas da empresa como um todo.

## **8.10 Toda entrevista é uma oportunidade de aprender mais**

Por mais que nos esforcarmos bastante, colocarmos nossa energia em praticar, gastar muito tempo estudando e fazendo diversos desafios práticos para nos prepararmos para as entrevistas de emprego, não será sempre que passaremos nos testes de emprego.

Seja pelo teste prático, por não se encaixar na cultura da empresa ou mesmo pelo nosso nervosismo na hora de falarmos sobre nós, algo pode não ser interessante para os recrutadores ou para a equipe que nos entrevistou.

A entrevista serve não somente para avaliar nosso conhecimento técnico, capacidade de resolver problemas e extrair informações que não foram passadas pelo nosso currículo, mas também para que a empresa conheça o nosso momento de carreira, pretensão salarial e outras competências que cada empresa pode precisar particularmente.

São milhares de variáveis possíveis e pontos que nós não conseguimos prever, por serem peculiares a cada empresa, e por isso é bem normal não passarmos em várias entrevistas no começo da carreira. O importante é extrair alguma experiência de cada uma.

Pode ser que façamos 100 entrevistas até conseguir nosso primeiro emprego na área. Também pode ser que façamos 5 e consigamos rapidamente. Não existe uma fórmula mágica para passar em entrevistas, pois é uma troca de interesses que deve ser muito bem-feita.

Não passar em uma entrevista não é algo ruim. Em cada processo seletivo temos a oportunidade de conhecer pessoas novas, empresas diferentes, culturas de times e tipos de testes diferentes. Se não der certo em uma oportunidade, estaremos mais preparados(as) para a próxima e assim por diante.

Sempre que possível é ideal que busquemos por feedback a respeito do motivo por que não passamos em um processo, assim conseguimos identificar onde podemos melhorar. Só devemos deixar claro que estamos procurando saber por que não passamos, por desejarmos nos aperfeiçoar, do contrário fica parecendo que não concordamos com a decisão da empresa/do time e isso não está dentro da nossa alçada. Quem decide se nos contrata ou não é a empresa de acordo com a sua necessidade e da negociação que fizeram conosco no processo, achar que devemos passar baseado em nosso próprio ego é no mínimo arrogante e não devemos alimentar esse tipo de achismo em nossas vidas.

Podemos retornar um e-mail para a pessoa de recrutamento ou do time que nos avaliou solicitando um feedback e pontos a melhorar. Nem todas as empresas respondem isso, na verdade raramente elas respondem. O padrão é a pessoa do recrutamento sumir sem nos dar notícias, mas tudo bem, alguma empresa vai nos responder e vai contribuir muito para o nosso crescimento profissional.

Uma estratégia quando não recebemos feedback sobre uma oportunidade em que não passamos é contar para a nossa rede de contatos sobre o processo seletivo, sobre como foi o teste e todos os processos para que as pessoas nos orientem onde podemos melhorar.

Talvez, contando para alguém o que fizemos e mostrando nosso teste prático para alguém mais experiente tenhamos o feedback que deveríamos receber da empresa.

Tendo esses feedbacks em mãos podemos ir muito mais longe em nossa carreira, afinal devemos buscar a melhoria contínua. Sabendo quais são os pontos a melhorar teremos mais foco.

## **8.11 Conseguindo nossos primeiros freelas**

O freelance é uma ótima oportunidade de trabalho. Muita gente não vê valor nessa modalidade porque acredita que trabalho mesmo é só aquele

que fazemos com carteira assinada e com patrão/patroa, mas a realidade não é essa. O trabalho como freela é muito valioso e existem pessoas que levam a vida somente nessa modalidade.

Trabalhar como freelancer não é bagunçado. Nós temos que nos responsabilizar pelos nossos horários, pelos nossos impostos, por geração de nota fiscal, por captação e retenção de clientes, além dos cuidados legais (ter um contrato) para a nossa empresa. Afinal, freelancer é um(a) profissional que possui uma empresa aberta, porém trabalha a sós.

Existem pegadinhas na profissão freelancer e dicas que podem nos ajudar a ir mais longe como profissionais individuais sobre como conseguir um cliente ou como tomar cuidado para não tomar um golpe. Vamos aprender mais sobre isso.

## **Como procurar freelas**

Quando decidimos nos tornar freelancers profissionais precisamos aprender diversas minúcias de administração de empresas com que não nos importariamos tanto trabalhando para uma empresa. E conseguir clientes é uma delas. Ao pensarmos em uma pessoa começando agora na área de programação parece mais complicado ainda, porém não é tão difícil assim.

Como já citei anteriormente, o trabalho para conhecidos ou para microempreendedores locais pode ser a nossa entrada na área de programação. Esses também podem ser nossos primeiros freelas, além de possíveis clientes de longa data, caso façamos um bom trabalho e conquistemos essas pessoas. Empresários, de qualquer tamanho, sempre precisam de um(a) programador(a).

Para conquistar nossos clientes, devemos mostrar para eles o quanto nosso trabalho pode melhorar suas vidas. Mostrar o quanto um site, um sistema, um aplicativo etc. pode contribuir com o crescimento do seu negócio.

Se temos uma ideia de aplicação que poderia melhorar como a empresa funciona, então estamos imaginando uma solução para um problema que o(a) dono(a) da empresa pode nunca ter imaginado que poderia ser resolvido. Devemos levantar o máximo de informação sobre esse problema,

o quanto a empresa perde por causa dele e então mostrar como a nossa solução transformaria o negócio de nosso possível cliente.

Um exemplo bem simples pode ser a redução de custo em um minimercado devido à quantidade de produtos que vencem no estoque por causa da reposição em excesso, feito “no olho”, aquela reposição em que alguém observa o estoque e chuta uma quantidade para comprar sem nenhuma base estatística. Um sistema com controle de reposição do estoque somente quando realmente precisa, baseado em épocas de vendas, já ajudaria na redução de custo e ainda por cima auxilia ao não desperdício, criando uma empresa sustentável.

É isso que deveria ser falado para nosso possível cliente. Porém, com números, claro.

Em um cenário fictício o dono do mercado sofre perda de 15 mil por mês somente com produtos vencidos no estoque. Um sistema que ajuda a trazer o lote com data de vencimento mais próxima para as prateleiras ajudaria a reduzir 50% desse desperdício, sendo assim o lucro do estabelecimento subiria R\$ 7.500,00, por exemplo.

E para todos os casos devemos fazer o mesmo: analisar o cenário, descobrir suas falhas, criar soluções, provar nossas soluções matematicamente (com estatísticas) e então levar isso para o possível cliente.

Claro que isso nem sempre vai funcionar de primeira, pois existem pessoas que simplesmente não desejam mudar a forma como seu negócio funciona; e tudo bem. Podemos criar essa solução, vender para outro cliente e a vida segue.

## **Como calcular o valor de um freela**

Conseguindo o primeiro cliente vem uma das partes mais difíceis de se trabalhar por conta que é precificar nosso trabalho.

Existem riscos ao calcular nosso valor como:

- perder o cliente por conta de o preço estar alto;

- perder dinheiro por cobrar muito barato por um trabalho muito alto;
- perder dinheiro e tempo com um cliente que desiste do produto enquanto o estamos desenvolvendo.

E também existem diversas maneiras de dar preço a um software, site etc. Antes de passar um preço é ideal que façamos uma pesquisa de mercado para saber o quanto outras pessoas estão cobrando, fazer uma bela pesquisa de mercado.

Essa pesquisa pode ser embasada pelo preço que encontramos em sites de empresas, outros softwares que encontramos com proposta parecida com a nossa ou mesmo entrando em contato com as empresas de desenvolvimento ou agências e fazendo um orçamento.

Com esse preço em mãos podemos praticar um valor parecido, jogar um valor mais alto ou mais baixo, dependendo da nossa capacidade, da complexidade do produto que vamos criar ou mesmo da nossa necessidade de carreira.

Uma das maneiras de se precificar o desenvolvimento é pelo tempo que vamos gastar para criar o produto e o valor da nossa hora de trabalho. O valor da hora varia muito pela nossa experiência na área, quanto mais experiência maior o valor hora.

Existe um site que pode nos ajudar a precificar a hora, que é o “minha hora, entre outros” com sua ferramenta *\*quanto custa minha hora* (<http://minhahora.entreoutros.com/>). No site nós colocamos quanto desejamos ganhar por mês, em quais dias da semana vamos trabalhar, área de atuação, ambiente de trabalho, dentre outras informações e ele traz um preço equivalente que podemos praticar ou não, dependendo da nossa confiança em utilizar esse valor.

Mesmo que não utilizemos o valor informado pelo site, ele é uma excelente ferramenta para entendermos o que devemos analisar ao precificar o nosso trabalho.

Só que mesmo sabendo agora como cobrar, tendo um possível cliente quase fechando conosco, como escapar de vender nosso trabalho e não receber



por isso?

## **Como escapar de freelas cilada**

Em tudo o que formos fazer é extremamente importante que façamos um contrato especificando valores, tempo de desenvolvimento e também o que será feito e que foi previamente acordado com nosso cliente (ou futuro cliente). O contrato é a nossa única garantia legal de que ambas as partes vão agir de boa fé e se não agirem teremos resguarda da lei para conseguirmos receber nossos honorários pelo trabalho prestado.

Mesmo com um contrato em mãos, as pessoas que fazem muito freelance costumam praticar um esquema de pagamento onde recebem pelo menos uma parte do valor em caso de calote, que é dividir o pagamento em duas ou mais partes. Uma parte do valor nós recebemos no começo do desenvolvimento e a outra parte quando entregamos o produto ou quebrado em várias partes e recebemos por entrega.

Eu indico dividir em mais partes e trabalhar com pequenas entregas mensuráveis. Ou seja, entregar o produto em partes que nosso cliente já pode sair usando e para cada parte dessas ele paga uma parcela do produto final. Isso é bom para ambas as partes, pois o usuário já começa a usufruir dos recursos que inventamos para lhe ajudar e nós já recebemos algum valor pelo trabalho que estamos desempenhando.

Outro problema recorrente no mundo do freelancer é o caso do cliente que participa de uma reunião conosco, fazemos todo o levantamento de requisitos, planejamos todo o projeto e até dizemos como vamos fazer e esse cliente, depois dessa reunião, nunca mais aparece e, quando menos esperamos, o projeto que nós planejamos está no ar.

Isso acontece porque, a pessoa pega tudo o que planejamos, leva para um amigo ou amiga (ou para o famoso sobrinho) que está aprendendo a programar e passa o nosso trabalho para essa pessoa fazer de graça.

Uma cilada dessas só é evitada cobrando um valor pelo início do trabalho. Ou seja: fechando contrato desde o começo. Claro que, nem todo cliente quer pagar antes de ter algo concreto e por isso temos que usar nossos

argumentos para que a pessoa confie em nosso trabalho, assim como entenda o nosso lado. Devemos demonstrar que somos profissionais e que desde a hora em que o contrato está assinado estamos nos comprometendo com o prazo e que esse contrato, assim como a entrada, nada mais é do que uma segurança para ambas as partes.

Em todo caso, não podemos deixar de fazer um contrato, de preferência com reconhecimento de firma no cartório e com tudo muito bem definido antes de começar a desenvolver.

Algo extremamente comum é utilizarmos sites de freelancers para conseguir clientes. Eu, pessoalmente, detesto esses serviços, pois são um verdadeiro leilão da nossa força de trabalho. Nós damos o valor X, que é o valor da nossa hora de trabalho, outra pessoa joga o valor X - 1, outra lança X - 2 e assim por diante, até que o cliente fecha com quem cobra o menor valor, não se importando com a qualidade do nosso trabalho.

Se o cliente quer pagar 2, 3 vezes menos do que nós cobramos, é um bom sinal de que esse cliente pode nos dar uma bela dor de cabeça no futuro. Minhas experiências com clientes assim foram de pessoas que pagam muito baixo e cobram por um serviço 5 estrelas, com a entrega o mais rápido possível, não importando que outro cliente pagou mais alto pela exclusividade e entrega mais rápida.

Existem serviços que vem com uma proposta diferente desses sites de leilão freelancer, em que a plataforma indica para o cliente o melhor profissional baseado em suas necessidades, como a Crowd (<https://crowd.br.com/>) e a Toptal (<https://toptal.com/>).

## **Cuidados fiscais**

Como prestadores de serviços, quando freelancer ou quando trabalhando diretamente para uma empresa como cliente fixo (conhecido como pessoa jurídica, o PJ), temos que emitir notas fiscais sobre o que fazemos e o que recebemos. Para emitir as notas precisamos ter empresa aberta e conhecer as regras para emissão da nossa localidade.

Abrir uma empresa não será difícil, o trabalho maior é procurar orientação na prefeitura local para saber sobre os encargos da nota que vamos gerar. Podemos conseguir as orientações corretas para a nossa cidade procurando pela Secretaria de Finanças na prefeitura ou no site da prefeitura local.

Como empreendedores individuais, podemos abrir o famoso MEI (Microempreendedor Individual), que, em nosso país, podemos fazer direto pelo site do governo, o Portal do Empreendedor (<http://www.portaldoempreendedor.gov.br/>).

Eu também utilizei os serviços de um contador quando prestei serviços, para evitar qualquer problema fiscal em que eu poderia cair por falta de conhecimento, por isso recorri aos serviços da Contabilizei (<https://contabilizei.com.br/>). Mas existem mais contabilidades online ou físicas que podem facilitar a nossa vida. É um investimento mínimo para não correr risco de errar e acabar tendo que pagar alguma multa por isso.

## **8.12 Conclusão**

Devemos criar uma boa imagem online, pensar bem no nosso currículo, que é a nossa vitrine de marketing pessoal, criar currículos com foco no que as pessoas de recrutamento realmente querem saber de nós e sempre acreditar que nós podemos e vamos conseguir nosso emprego na área que gostamos.

O processo até conseguir o primeiro emprego na área pode não ser nada fácil ou animador, pois podemos/vamos receber vários não's, mas isso é extremamente normal, acontece com todo mundo e em cada processo podemos aprender mais e nos prepararmos para o próximo.

De qualquer modo podemos contar com o apoio das comunidades e dos amigos e amigas que trabalham na área, por isso devemos buscar ajuda sempre que sentirmos medo ou tivermos dúvidas sobre um processo seletivo ou sobre o que fizemos “de errado” para não passarmos na entrevista.

## CAPÍTULO 9

# Aonde ir depois daqui

Este livro é basicamente sobre **carreira em programação**. Nós aprendemos desde o que é um software, como os programas são criados, como são desenvolvidos os sites e aplicativos de celular, quais as áreas em que podemos trabalhar até chegar em como conseguir nosso primeiro emprego ou trabalhar como freelancer. Mas não aprendemos, de fato, a programar, somente como praticar programação e como criar um portfólio.

Agora vem a parte mais legal de tudo isso que é realmente buscar conhecimento e aprender a programar para usar todas as dicas que recebemos neste livro.

*Reforço que isso tudo não é obrigatório.* Podemos seguir nosso próprio caminho e buscar conhecimento em programação, praticar e aprender sem necessariamente seguir este fluxo ou aproveitando todos os tópicos. Tem alguns dos tópicos listados aqui em que eu só fui focar depois de algum tempo trabalhando na área, como Design Patterns ou Testes de Software.

Durante nossos estudos temos alguns assuntos que são muito importantes, mas não são muito divertidos. Precisamos conhecer sua importância desses tópicos desde já para não acabar deixando-os de lado pela sua "complexidade".

Por isso vamos conhecer agora quais são esses tópicos e por que eles são tão importantes no universo da programação. Desde sua teoria até onde serão importantes.

## 9.1 Algoritmos e estruturas de dados

Nós aprendemos o que é um algoritmo. Sabemos que é aquele fluxo de instruções (regras, receita) que criamos para resolver um problema, mas

realmente é tão importante aprender a criar algoritmos primeiro e não partir **direto** para uma linguagem de programação?

Sim, eu considero muito importante. Os algoritmos são a base de tudo ao nosso redor e, se não entendemos, estudamos ou praticamos isso, criar programas será muito mais difícil. Nós acabaremos decorando instruções das linguagens e como criar programas que vimos na internet, porém não saberemos criar um programa do zero.

Existem comandos nas linguagens que fazem algumas rotinas que teríamos que criar algoritmos para tal. Como o comando `reduce`, de JavaScript, que funciona da seguinte maneira:

```
const nossoArray = [1, 2, 3, 4, 5]

const valorTotal = nossoArray.reduce((acumulador, itemAtual) => {
  return acumulador + itemAtual
})

alert(valorTotal)
```

Nós temos uma estrutura de dados chamada `array`, que nada mais é do que uma coleção de valores. Se precisamos somar todos os valores desse `array`, podemos usar o `reduce`, que executará uma ação para cada item do `array`. Ou seja, se temos um `array` (`nossoArray`) com 5 itens, então ele vai realizar uma ação para cada um dos 5 itens e retornar algo. Esse retorno pode ser exibido na tela como no exemplo acima quando usamos o comando `alert(valorTotal)`.

Mas, nem tudo existe pronto e algumas vezes precisamos implementar/criar algo do zero para atender a nossa necessidade. Se não existisse o `reduce`, teríamos que saber criar um algoritmo para fazer o que ele faz, que seria algo como:

```
const nossoArray = [1, 2, 3, 4, 5]
const tamanhoDoArray = nossoArray.length
let valorTotal = 0

for (let i = 0; i < tamanhoDoArray; i++) {
```

```
    valorTotal = valorTotal + nossoArray[i]
}

alert(valorTotal)
```

Aqui temos a nossa coleção de valores e, em vez de usar `reduce`, usamos o comando `for`, de JavaScript e outras linguagens de programação, para chegar ao mesmo resultado.

Um comando não existir na nossa linguagem de programação e ser necessário implementar essa rotina do zero pode ser considerado o **último** motivo pelo qual devemos aprender algoritmos. É só um exemplo para entendermos que precisamos aprender a pensar em algoritmos, pensar em lógica de programação.

As estruturas de dados fazem parte deste mesmo patamar de importância, pois sabendo isso conseguiremos um melhor desempenho (velocidade) em nossos programas assim como facilitar a nossa vida quando trabalhamos com dados.

Estruturas de dados são o modo como organizamos e armazenamos os dados em nossos programas. Quando temos tudo bem organizado e disposto em nossos algoritmos conseguimos acessar esses dados de uma maneira mais fácil, assim como pesquisar por um registro importante em uma grande quantidade de dados.

O nosso array, em JavaScript, é um exemplo clássico de como podemos usar estruturas de dados para organizar melhor nosso código. Se queremos representar diversos valores em um programa precisamos criar um identificador para cada, por exemplo: imaginemos que vamos ter, em nosso software, 4 notas de alunos da escola, podemos fazer:

```
const nota1 = 10
const nota2 = 7
const nota3 = 8
const nota4 = 9
```

Mas e se precisarmos armazenar mais notas ainda, digamos 11 x 4 notas (4 notas por mês durante o ano vigente de aulas)?

Será necessário continuar criando mais notas ainda em nosso programa manualmente. `nota5` , `nota6` , `nota7` , `nota8` etc.

E depois, para acessar esses valores para fazer uma soma?

Seria necessário criar um código somando vários identificadores.

```
nota1 + nota2 + nota3 + nota4 + nota5 + nota6 ...
```

Se usarmos um array , podemos ter um identificador `notas` e nesse local colocar todas elas.

```
const notas = [10, 7, 8, 9, 5, 6, 10, 8, 7, 6, 9]
```

Se quisermos a primeira nota do ano deste aluno bastaria fazer: `notas[0]` , sendo que `0` equivale, em JavaScript, ao primeiro item de uma cadeia de valores, o array. Se precisarmos somar todos os valores, podemos utilizar nosso querido `reduce` e assim por diante, facilitando a nossa organização de código, assim como manipulação dos dados.

Podemos aprender mais sobre algoritmos e lógica de programação através de bons livros ou cursos:

- Curso em vídeo: Algoritmos e lógica de programação  
(<https://www.cursoemvideo.com/course/curso-de-algoritmos/>)
- Curso online de estruturas de dados da Alura  
(<https://www.alura.com.br/curso-online-estrutura-de-dados/>)
- Apostila da Caelum de estruturas de dados  
(<https://www.caelum.com.br/apostila-java-estrutura-dados/>)
- *Estruturas de dados com Jogos*, por Roberto Ferrari  
(<https://www.amazon.com.br/Estruturas-Dados-Jogos-Roberto-Ferrari/dp/8535278044/>)
- *Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++*, por Ana Fernanda Gomes Ascencio e Graziela Santos Araújo  
(<https://www.amazon.com.br/Estruturas-Dados-algoritmos-complexidade-implementações-ebook/dp/B0167CVDQ4/>)

- *Entendendo Algoritmos*, por Aditya Y. Bhargava  
(<https://www.amazon.com.br/Entendendo-Algoritmos-Aditya-Y-Bhargava/dp/8575225634/>)

## 9.2 Paradigmas de programação

Ao trabalharmos com programação, temos diferentes maneiras de estruturar o nosso algoritmo, de escrever a nossa lógica. Não é somente sair colocando as regrinhas uma embaixo da outra. Existem diferentes modos de implementar essas regras. Essas diferentes maneiras de se estruturar nosso programa são chamadas de paradigmas de programação.

Conhecer muito bem os paradigmas vai nos ajudar a entender a melhor maneira de se resolver cada problema. Quando um paradigma não nos atender, podemos utilizar outro (se a nossa linguagem possibilitar o uso de múltiplos paradigmas).

Os paradigmas mais conhecidos, hoje em dia, são: Orientação a Objetos, Programação Funcional e Programação Imperativa. Mas existem mais paradigmas em Ciência da Computação e podemos aprender mais sobre eles por meio destes recursos:

- *Análise e Design Orientados a Objetos*, por Hélio Engholm  
(<https://www.amazon.com.br/Análise-Design-Orientados-Objetos-Engholm-ebook/dp/B07199C5LH/>)
- *Orientação a Objetos: Aprenda seus conceitos e suas aplicabilidades de forma efetiva*, por Thiago Leite e Carvalho  
(<https://www.casadocodigo.com.br/products/livro-oo-conceitos/>)
- *Haskell: Uma introdução à programação funcional*, por Alexandre Garcia de Oliveira (<https://www.casadocodigo.com.br/products/livro-haskell/>)

E Programação Imperativa é o modelo clássico que aprendemos quando estudamos programação em qualquer linguagem, por isso basta estudar



programação e algoritmos, com os recursos citados anteriormente, que conhecemos o Paradigma Imperativo de programação.

### 9.3 Design Patterns

Assim como os paradigmas e estruturas de dados, existem algumas “regras” que podemos seguir para facilitar e melhorar a nossa vida como pessoas desenvolvedoras de software. Essas regras são os padrões de projeto.

Para programar só é necessário entender lógica de programação e aprender uma linguagem, mas para criar um bom software, um sistema que seja fácil de manter, atualizar e escalar (crescer), é muito importante que usemos os *design patterns* (padrões de projeto).

Os padrões de projeto nada mais são que práticas de desenvolvimento de software que algumas pessoas criaram há um tempo atrás para que a criação de grandes aplicações fosse menos dolorida, pois antigamente as pessoas desenvolvedoras passavam por certos apertos na hora de escrever os programas e dar manutenção neles.

Esses padrões foram extensivamente adotados e são estimulados até hoje. O melhor local para aprender sobre esse tema é o livro *Padrões de Projetos*, de Erich Gamma (<https://www.amazon.com.br/Padrões-Projetos-Erich-Gamma/dp/8573076100/>).

Claro, não adianta muito aprendermos padrões de projetos se nem entendemos sobre construção de softwares, por isso este tópico vem depois de algoritmos, estruturas de dados e paradigmas de programação.

### 9.4 Testes de software

Como dito anteriormente, garantir a qualidade de nosso software é responsabilidade nossa. E uma das maneiras de se garantir certa qualidade é fazendo testes automatizados.

Existem diversos tipos de testes, para cada tipo de software e para cada parte do nosso programa e precisamos utilizar esses testes para conseguirmos assegurar da segurança da aplicação.

É garantir que nada vai parar do nada por conta da alteração de uma linha de código ou porque alguém mudou a resposta de uma aplicação back=end.

Uma maneira de aprender sobre esse tema é com o livro do Mauricio Aniche, *Testes automatizados de software* (<https://www.casadocodigo.com.br/products/livro-testes-de-software/>).

## 9.5 Arquitetura de computadores, redes e sistemas operacionais

Para aprender arquitetura de computadores, redes e sistemas operacionais, os melhores livros são um pouco densos (“chatos”), porém muito importantes!

Temos:

- *Arquitetura e Organização de Computadores*, por William Stallings (<https://www.amazon.com.br/Arquitetura-Organização-Computadores-William-Stallings/dp/8576055643/>)
- *Organização Estruturada de Computadores*, por Andrew S. Tanenbaum (<https://www.amazon.com/Organiza%C3%A7%C3%A3o-Estruturada-Computadores-Portuguese-Tanenbaum-ebook/dp/B018KHG1D4/>)
- *Redes de computadores*, por Tanenbaum (<https://www.amazon.com.br/Redes-Computadores-Andrew-S-Tanenbaum/dp/857605924X/>)
- *Sistemas Operacionais Modernos*, também por Tanenbaum (<https://www.amazon.com.br/Sistemas-Operacionais-Modernos-Andrew-Tanenbaum/dp/8576052377/>)

## 9.6 Versionamento de código

Hoje em dia o versionador de código mais utilizado, sem sombra de dúvidas, é o Git.

Criado por Linus Torvalds, o também criador do Linux Kernel, foi rapidamente adotado pelo mercado por possuir muitos pontos positivos se comparado a outros versionadores, como o fato de funcionar de modo distribuído e offline.

Existem diversas plataformas de hospedagem de código-fonte baseadas em Git e a mais famosa delas é o GitHub. O GitHub possui um recurso bem legal para aprendermos Git, que é o try.github.io (<https://try.github.io/>).

## 9.7 Inglês

Alguns dos conteúdos mais atualizados, mais completos e bibliografias antigas (ainda muito boas e recomendadas) estão em inglês. É primordial que tiremos um tempo para aprendermos este idioma, pois vamos utilizá-lo para tudo em nossa carreira.

Desde estudos até uma reunião com um time estrangeiro ou uma entrevista de emprego, o inglês abre muitas portas em nossa vida profissional.

Existe um projeto muito bonito que ajuda pessoas que não têm condições de estudar o idioma imediatamente a se manterem atualizadas e continuarem estudando programação mais e mais, que é o devtranslate (<https://medium.com/devtranslate/>). Mas, mesmo assim, é ideal priorizarmos o inglês em algum momento de nossas vidas.

Alguns aplicativos e plataformas podem nos ajudar a aprender o idioma, seja no caminho para o trabalho (no transporte público ou não), assim como para praticar mais, para aprendermos mais rápido:

- Duolingo (<https://duolingo.com/>)
- italki (<https://italki.com/>)
- Tandem (<https://tandem.net/>)

- OpenTalk (<https://getopentalk.com/>)

## 9.8 Livros cuja leitura deveria ser lei

Fora os livros técnicos e conteúdo extremamente direcionado ao código nós somos pessoas e, como pessoas, temos nossos problemas que não são resolvidos com mais leitura sobre programação.

Sempre que alguém me diz que estuda programação há dois anos e não se sente confortável para aplicar em uma vaga ou que se sente em posição desigual a outras pessoas que estudam ao mesmo tempo eu logo questiono sobre a prática.

- Você tem praticado bastante?

Quando a resposta é sim, logo percebemos que o problema da pessoa não está em programação, mas em sua própria autoestima ou insegurança. Para isso não existem cursos técnicos ou livros sobre códigos que vão nos ajudar, precisamos aprender a nos entender e nos avaliar.

Existem obras perfeitas para isso e que servem para todo tipo de problema que vamos passar em nossas carreiras. Vamos conhecer alguns.

1. *O programador apaixonado*, por Chad Fowler  
(<https://www.casadocodigo.com.br/products/livro-programador-apaixonado/>)
2. *O guia do mestre programador*, por Carlos Bueno  
(<https://www.casadocodigo.com.br/products/livro-guia-mestre-programador/>)
3. *Foco*, por Daniel Goleman  
(<https://www.amazon.com.br/dp/8539005352/>)
4. *Inteligência emocional*, por Daniel Goleman  
(<https://www.amazon.com.br/Inteligência-Emocional-Daniel-Goleman/dp/8573020806/>)
5. *O poder do hábito*, por Charles Duhigg  
(<https://www.amazon.com.br/Poder-do-Hábito-Charles->)

- [Duhigg/dp/8539004119/](https://www.amazon.com.br/Duhigg/dp/8539004119/))
6. *O poder da ação*, por Paulo Vieira  
(<https://www.amazon.com.br/Poder-Ação-Ph-D-Paulo-Vieira/dp/854520034X/>)
  7. *Rápido e devagar, duas formas de pensar*, por Daniel Kahneman  
(<https://www.amazon.com.br/Rápido-Devagar-Daniel-Kahneman/dp/853900383X/>)
  8. *Atenção plena: Mindfulness*, por Mark Williams  
(<https://www.amazon.com.br/Atenção-Plena-Mindfulness-Mark-Williams/dp/8543101875/>)
  9. *Por que fazemos o que fazemos?*, por Mario Sergio Cortella  
(<https://www.amazon.com.br/Por-que-Fazemos/dp/8542207416/>)
  10. *O cérebro de alta performance*, por Fernando Garcia  
(<https://www.amazon.com.br/cérebro-alta-performance-Fernando-Garcia/dp/8573129085/>)
  11. *Produtividade para quem quer tempo*, por Geronimo Theml  
(<https://www.amazon.com.br/Produtividade-para-quem-quer-tempo/dp/8545200978/>)
  12. *Ansiedade*, por Augusto Cury  
(<https://www.amazon.com.br/Ansiedade-Augusto-Cury/dp/8502218484/>)
  13. *O Princípio 80/20. Os segredos para conseguir mais com menos*, por Richard Koch  
(<https://www.amazon.com.br/Princípio-Segredos-Para-Conseguir-Menos/dp/858235259X/>)
  14. *A Mente Organizada. Como pensar com clareza na Era da sobrecarga de informação*, por Daniel J. Levitin  
(<https://www.amazon.com.br/Organizada-Pensar-Clareza-Sobrecarga-Informação/dp/8539006995/>)
  15. *A arte da meditação*, por Daniel Goleman  
(<https://www.amazon.com.br/Arte-Meditação-Autoestima-CD/dp/8575421867/>)

Parecem muitos títulos para lermos de uma só vez, porém são importantes para aprendermos a equilibrar as nossas vidas, nos entendermos cada vez mais e conquistarmos uma qualidade de vida maior. Podemos aproveitar as

dicas de produtividade que apresentei até aqui para consumirmos o conteúdo desses livros.

Deles, os mais importantes para a nossa área, são: *O programador apaixonado* e *O guia do mestre programador*. Então, se você não quiser ler os demais, leia pelo menos esses dois em complemento ao que aprendemos até aqui.

Tendo nos preparado melhor psicologicamente, podemos agora buscar entender um pouco mais de desenvolvimento com livros técnicos avançados:

1. *Refactoring: Improving the Design of Existing Code*, por Martin Fowler, Kent Beck e John Brant  
(<https://www.amazon.com.br/Refactoring-Improving-Design-Existing-Code/dp/0201485672/>)
2. *Domain Driven Design*, por Eric Evans  
(<https://www.amazon.com.br/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215/>)
3. *Padrões de Arquitetura de Aplicações Corporativas*, por Martin Fowler (<https://www.amazon.com.br/Padr%C3%B5es-Arquitetura-Aplica%C3%A7%C3%B5es-Corporativas-Martin/dp/8536306386/>)
4. *Refatoração para Padrões*, por Joshua Kerievsky  
(<https://www.amazon.com/Refatoração-Padrões-Portuguese-Joshua-Kerievsky-ebook/dp/B019HIO22C/>)
5. *Clean Code: A Handbook of Agile Software Craftsmanship*, por Robert C. Martin, Michael C. Feathers e Timothy R. Ottinger  
(<https://www.amazon.com.br/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882/>)

## 9.9 Conclusão

Aprendemos muito durante a leitura deste livro e agora recebemos materiais e dicas para aprender mais ainda e irmos mais longe em nossa jornada pelo universo da programação.

Devemos evitar negligenciar os temas que vimos que são importantes, por mais que, as vezes, estudar estes temas seja meio maçante. Todo esse trabalho vai nos auxiliar a nos tornarmos profissionais melhores e para que tenhamos habilidades para ir mais longe em nossa carreira.

## CAPÍTULO 10

# Considerações finais

Nossa jornada nesta área fantástica, que é o universo da programação, somente começou com os conhecimentos que adquirimos até aqui.

Nós temos em mãos, agora, o conhecimento básico e a inspiração para entrarmos na área de desenvolvimento de software de uma vez por todas, aprender o máximo e conseguir nosso emprego (ou mesmo trabalhar por conta própria).

Nenhum grande sonho é alcançado facilmente, mas não será tão difícil alcançar a maestria em programação. A grande dificuldade aqui está em removermos os limites que nós mesmos(as) colocamos em nossa vida, que é a falta de disciplina, falta de confiança e até mesmo aquela preguiça de estudar.

Outra grande barreira é a social. Nós sabemos que nem todo mundo possui recursos financeiros para comprar um bom curso de programação, como os da Caelum ou Alura, ou os livros da nossa área que eu indiquei no capítulo anterior e que, muitas vezes, passam da casa dos 80 reais.

Mas não desista e nem desanime por isso! Busque conteúdo gratuito na internet, procure se apoiar nas comunidades de programação, me procure. Em todo caso, podemos contar com as comunidades de desenvolvimento de software para nos ajudarem quando estivermos em apuros ou com medo de continuar investindo nosso tempo em programação.

Caso você não entenda algo que foi ensinado no livro ou discorde do que eu penso/falei, fique à vontade para entrar em contato comigo. Ficarei muito feliz de ouvir seu feedback e melhorar esta obra.

Quero finalizar este livro com um desafio: se você conseguir seguir as dicas que foram dadas e conquistar seu primeiro emprego em menos de um ano, me envie um e-mail com o assunto **eu conquistei o universo da programação** contando como foi seu processo.



Adoro saber quando as pessoas conquistam seus objetivos/sonhos e vou ficar muito feliz de celebrar junto contigo esta vitória. Meus contatos estão logo adiante.

- Twitter: w\_oliveiras ([https://twitter.com/w\\_oliveiras/](https://twitter.com/w_oliveiras/)) (onde sou mais ativo)
- LinkedIn: william-oliveira (<https://linkedin.com/in/william-oliveira/>)
- Website: woliveiras (<https://woliveiras.com.br/>)

Caso algum contato mude, estará em meu website.

Tenha uma excelente experiência de carreira e divirta-se conhecendo mais e mais do fantástico universo da programação.