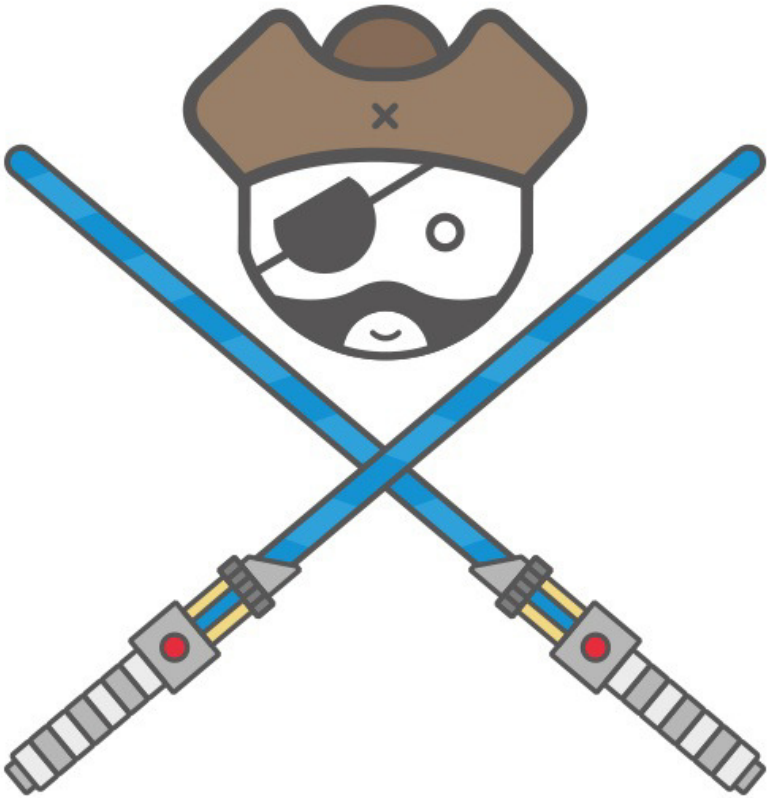
Guia do mestre programador

Pensando como pirata, evoluindo como jedi



Casa do Código

CARLOS BUENO

Edição

Adriano Almeida
Vivian Matsui

Revisão

Bianca Hubert
Vivian Matsui

[2018]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

SOBRE O GRUPO CAELUM

Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código faz parte do Grupo Caelum, um grupo focado na educação e ensino de tecnologia, design e negócios.

Se você gosta de aprender, convidamos você a conhecer a Alura (www.alura.com.br), que é o braço de cursos online do Grupo. Acesse o site deles e veja as centenas de cursos disponíveis para você fazer da sua casa também, no seu computador. Muitos instrutores da Alura são também autores aqui da Casa do Código.

O mesmo vale para os cursos da Caelum (www.caelum.com.br), que é o lado de cursos presenciais, onde você pode aprender junto dos instrutores em tempo real e usando toda a infraestrutura fornecida pela empresa. Veja também as opções disponíveis lá.

ISBN

Impresso e PDF: 978-85-5519-120-6

EPUB: 978-85-5519-121-3

MOBI: 978-85-5519-122-0

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

PREFÁCIO

Figura 1: Prefacio

Quando analisamos nossa vida, a primeira coisa em que pensamos é como vamos ganhar dinheiro para vivermos tranquilamente. Podemos investir em uma poupança – apesar de ser um investimento conservador que não nos trará muita diferença no futuro –, podemos investir em ações na bolsa e ganhar um bom dinheiro ou perdê-lo rapidamente, ou fazer outros investimentos, como compra de casas para venda em longo prazo.

Entretanto, nenhum desses investimentos consegue bater o investimento no conhecimento, pois, como dizem popularmente, uma pessoa pode ganhar dinheiro facilmente por uma questão de sorte, mas também pode ser roubada e nunca mais recuperá-lo. Agora, conhecimento é um dos raros bens humanos que não podem ser roubados.

É como a frase eternizada de Benjamin Franklin: *Investir em conhecimento rende sempre os melhores juros*. Cada vez que adquirimos mais conhecimento, acabamos sendo reconhecidos

pelas pessoas em volta e, com esse reconhecimento, conseguimos colher os frutos rapidamente.

Em minha carreia profissional, de 2009 a 2014, elevei meu salário em 1.000%. Nesse período de 5 anos, será que conseguiria com algum investimento evoluir meus recebíveis mensal de tal forma se não investisse em minha educação? Confesso que não é fácil, mas também se fosse, não colheria a recompensa tão rapidamente.

Então, sabendo disso, aconselho que invista sempre em conhecimento. Alguns cursos e livros são caríssimos e muitas pessoas vão questioná-lo por adquirir algo que não é tangível, e você acabará passando períodos apertados. Porém, é essa capacidade de ficar "apertado" e investir em seu futuro que lhe dará um salto salarial e profissional, enquanto os que dizem isso estarão na mesmice e reclamando.

Para conseguir buscar meu sonho de ser programador, passei por profissões temporárias que davam as condições financeiras para alcançar meus objetivos. Ao contrário de muitos que devem estar lendo este livro, iniciei meu trabalho jovem, ajudando meu pai e lavando copos em sua lanchonete aos 11 anos. Então, comecei a fazer alguns cursos custeados por ele.

Os primeiros foram de MS-DOS, Wordstar e DBase. Logo após, fiz um curso de Clipper e, com o passar do tempo, fui me apaixonando cada vez mais pela área de TI. Comecei a fazer meus primeiros programas, e vi que deveria entrar logo em algo que me direcionasse ao desenvolvimento de software.

Assim, fui estudar em uma Escola Técnica, a qual era

particular. Para isso, precisei, além da ajuda da minha mãe, de um emprego para completar o investimento. Então, comecei a ajudar um senhor a fazer os serviços de manutenção de sua casa: jardinagem, pintura, compras no mercado etc. Nunca me preocupei em fazer essas subtarefas, pois sabia que era temporário, e que logo estaria com um emprego de programador podendo ganhar bons salários.

O caminho foi longo, mas não estava errado. Com o passar do tempo, minhas tarefas laborais eram cada vez mais nobres e meu salário cada vez maior, sendo que, quando olhava para trás, via amigos de famílias mais abastadas no mesmo emprego de anos atrás, com o mesmo salário.

Entretanto, além de conhecimento, a variável mais importante do meu sucesso foi o **foco**, pois em nenhum momento me perdi no vasto oceano de conhecimento da área de TI. Claro que nunca quis ser um especialista, mas também não poderia saber um pouco de cada coisa. Criei um perfil técnico que acreditava ser de sucesso e foquei em me tornar isso, sendo que cada ação que fiz foi puramente planejada para o sucesso futuro.

Além do foco, comecei a entender que o relacionamento era algo importante e que, cada vez que me comunicava melhor, meus resultados também melhoravam. Passei a entender bem Aristóteles quando ele disse: *"O homem é um animal político"*.

Assim, passei a me aprofundar no tema buscando conhecimento em áreas como história, filosofia, ciências políticas, entre outras que estão totalmente fora de Tecnologia. E, com essas skills, consegui entender um pouco mais além do que um profissional comum. Com esse perfil, comecei a perceber o

momento em que deveria apresentar algo a um chefe, quando e como deveria falar com um técnico de um cliente, como tratar meus companheiros de trabalho. Aprendi assim que um programador sozinho pode construir um ótimo sistema, mas um ótimo programador com uma equipe coesa pode revolucionar e obter grandes resultados, desde que saiba se relacionar e potencializar todos ao máximo.

Não foi fácil seguir esse caminho. Aprendi alguns conceitos básicos lendo alguns filósofos, sendo o principal deles: não dar respostas imediatas. Isso foi o que revolucionou tudo para mim. Sempre que alguém me pedia uma solução técnica, ou qualquer outro tipo de questionamento que ocorria em meu trabalho, eu não respondia imediatamente. Mas por que fazia isso?

A resposta é simples. Se eu responder imediatamente, com certeza serei mais levado pela emoção do que pela razão, mesmo sendo uma resposta técnica, pois necessito de tempo para avaliar todo o cenário e ver se existem outras soluções inteligentes que superam, inclusive, a que propus.

Além disso, o padrão escolhido e respondido pode ter um antipadrão, que poderia passar despercebido em uma resposta rápida. Essa decisão de voltar, estudar o caso e apenas depois responder foi o que me fez ser respeitado, pois sabem que raramente proponho algo que dê errado.

Pense bem. Ninguém gosta de uma resposta errada, por mais que seja rápida. Todos à nossa volta praticamente utilizam uma balança que avalia nossos erros e acertos e, com isso, tiram suas impressões sobre nós. Assim, quanto mais erros, pior pode ficar.

Em caso de dúvida, nunca responda. Na certeza, reflita, medite, tente extrair algo além, saia para andar de bicicleta, vá a uma exposição, descanse um pouco a mente, e pense.

Apesar de agir assim, acabava me frustrando um pouco profissionalmente, devido a insucessos em projetos, atitudes corporativistas que vi em empresas por que passei, supergerentes que gerenciam cada detalhe de sua vida, entre outras atitudes que seguem o padrão empresarial tradicional e que frustra a todos.

Não sei se apenas eu sonho por um emprego onde se tenha prazer em trabalhar, e ter prazer não se resume a farra ou comilanças à vontade, mas respeito e crédito no conhecimento que se possui. Na contramão do reconhecimento e respeito, as empresas em sua maioria são lideradas por pessoas com sentimento paternalista, os quais tentam solucionar tudo da sua maneira, colocando todos os outros trabalhadores em um *status quo* inferior, além de impedir que outros deem suas opiniões e ajudem a pensar como solucionar certos problemas.

O pior é que esses chefes – os quais não podemos chamar de líderes – veem empresas como a IDEO, Apple etc., e acham o máximo, mas mal sabem eles que fazem tudo ao contrário, que rasgam e jogam no lixo todo o conhecimento revolucionário em Tecnologia. Então, pensando nisso, eu e mais dois amigos – na realidade, um amigo e uma amiga –, fundamos uma empresa de TI na qual podemos: aplicar todos esses conceitos, tratar a todos que nela trabalham com o mesmo *status quo* e poder, e inovar escutando a todos. Ou seja, uma verdadeira empresa *pirata*, na qual, em tempos de *ditadura empresarial*, lidamos com igualdade, liberdade e experimentação.

Hoje, posso ir trabalhar de bicicleta, levar a equipe para trabalhar em um parque, destinar tempo de estudo (individual, coletivo ou em par), escutar feedback de todos sobre o que é feito, e juntos podemos propor novidades que inovem. O foco não é apenas como empresa na relação com o cliente, mas em sermos mais humanos e termos um espaço saudável em que todos estão preocupados com cada um que ali trabalha.

Por exemplo, fico feliz em saber que um dos membros da equipe pode levar seu filho ao pediatra acompanhado da mãe, pois acredito que o desenvolvimento dessa criança será muito maior. Imagino que, quando ela crescer, o pai poderá assistir suas apresentações e ir a reuniões de pais e mestres, uma vez que poderá fazer *home-office* ou se abster de ir ao trabalho quando julgar que sua vida pessoal está acima naquele dia de trabalho. Claro que não é fácil ter pessoas com ótima autogestão, mas isso é possível de se encontrar no mercado; é só querer.

Finalmente, pensando em todo meu percurso até o presente momento, resolvi escrever esta obra para ajudar a quem se interessar em chegar aqui onde estou, e a conseguir criar círculos mais inclusivos e cooperativos. Espero que gostem do livro e que ele possa guiar seu caminho para ser um *Mestre Programador*.

Por Carlos Bueno

AGRADECIMENTOS

Primeiramente, nossos agradecimentos vão a todas as pessoas que contribuem com o mundo *open source*, seja por um framework, módulos, aplicações, sistemas operacionais, ou qualquer espécie de desenvolvimento que tem como caráter contribuir com uma sociedade mais justa.

Nós pensamos primeiramente na evolução, não queremos que alguém tenha de inventar novamente o que já inventamos e perdemos tempo fazendo. Queremos deixar o que contribuímos, para que seja evoluído e utilizado por todos, sem nenhum tipo de discriminação.

Agradeço a Paulo Jesus, Pedro Araujo, Ronaldo Braghittoni e Luis Buttes pela colaboração que deram em vários capítulos do livro. Paulo Jesus escreveu integralmente o *Capítulo – Política*; Ronaldo Braghittoni contribuiu para a escrita a quatro mãos no *Capítulo – A arte de programar*; Pedro Araujo indicou obras, fez anotações, criticou várias partes do conteúdo e discutiu vários aspectos do livro comigo; e Luis Buttes, um grande artista que criou a capa e toda a identidade visual do livro, com incríveis *comics* que foram inseridas em cada capítulo – com certeza, sem sua ajuda, a obra não teria o mesmo aspecto.

Também agradeço a meu irmão Cleber Bueno, que me indicou a editora Casa do Código e me deu todo apoio. Agradeço a meus pais, Carlos e Célia, meu filho Luiz Carlos Michelin Bueno e minha namorada Karoline Ribeiro, por entenderem e estarem ao meu lado todo o tempo que dediquei escrevendo a obra.

Muito obrigado aos amigos que ajudaram como *beta readers*: Leonardo Dorathoto, Eduardo Teixeira (Santos), Michael Guedes, Dennis Ferrari, Vinicius Baptista (Padawan Pirata), William Mendes. Também, aos companheiros de Portugal: Nuno Almeida, Nuno Duarte, Nuno Franca (Darth Vader *rs*), José Pedro Melo, Tania Sousa, Sandra Correia, Marco Kirino; e aos de Angola: Pedro Quintas e Dilson Manjenje.

SOBRE OS AUTORES

CARLOS BUENO

Diretor de Sistemas na Sennit Tecnologia (http://www.sennit.com.br), trabalha com uma pequena equipe pirata, em que todos trabalham livremente sem serem questionados por um chefe.

É um fã assíduo de *Star Trek*, *Star Wars* e pela trilogia *Fundação*, de Isaac Asimov. Gosta de escutar todos os estilos de Rock, Jazz e música clássica, adora ler comics e mangás semanalmente, jogou muito RPG em sua juventude, mas hoje em dia não participa de nenhum grupo.

Mantém 4 frameworks open source de sua autoria, além de participar no desenvolvimento de outros. Em 2006, criou um *sniffer* para MSN que poderia ser utilizado em um servidor roteador, rodando *unix like*. Este logava todas atividades do MSN na rede por usuário, e essa fonte estava armazenada no codigolivre.org.br, que infelizmente foi descontinuado e teve as fontes perdidas. Era um projeto escrito em C++, que usava a biblioteca *plibcap*, e possuía uma interface web para visualizar os logs de forma amigável, esta escrita em PHP utilizando PostgreSQL.

Além dos frameworks, já escreveu artigos e dicas em alguns sites, como *Viva ao Linux*, *Devmedia*, *Codeproject*, entre outros. Já navegou para mares distantes e trabalhou em equipes de Angola e Portugal, onde sempre pregou a interação da equipe em vez de

processos.

Começou a programar basic aos 10 anos em um MSX. Depois estudou Clipper e dBASE, até ingressar no ensino técnico e ir para Java e, depois, C#. Ingressou na USP, no Instituto de Astronomia e Geofísica, mas o ensino em sala de aula o limitava. Por já estar inserido no mercado de TI, deixou a vida acadêmica e focou em evoluir sua carreira dando mais atenção ao que realmente lhe interessava.

Trabalhos do autor

- *Frameworks OpenSource:*

SennitShareAngularJs (2015) – https://github.com/buenokinder/SennitShareAngularJs

Sennit Sharepoint Automapper (2014) – http://sennitspmapper.codeplex.com/

Sennit Sharepoint Online Helper (2014) – http://ssoh.codeplex.com/

Sennit.Cryptography (2014) – https://www.nuget.org/packages/Sennit.Cryptography/

MSN Sniffer (2006)

- *Artigos:*

Map Your Sharepoint Lists in Strongly Typed Entities (2014) – http://www.codeproject.com/Tips/768600/Map-Your-Sharepoint-Lists-in-Strongly-Typed-Entiti

Encontre a Governança correta e resolva seus problemas futuros – http://www.devmedia.com.br/encontre-a-governanca-correta-e-resolva-seus-problemas-futuros/26568

Instalando um PDC Samba no Debian (2006) – http://www.vivaolinux.com.br/artigo/Instalando-um-PDC-Samba-no-Debian/

PAULO JESUS

Professor de História do Cursinho Comunitário Pré-Universitário Gauss, desde 2004. Além das aulas, também é vereador na cidade de Atibaia, onde nasceu, e agora tenta implantar novas ideias para uma política mais participativa e democrática.

É mestrando em História Econômica na Universidade de São Paulo (USP), e realiza uma pesquisa sobre as políticas econômicas do governo socialista de Salvador Allende, no Chile. Já morou na Austrália, onde trabalhou em diversas atividades e militou em um grupo político de esquerda. Além das diversas atividades políticas, educacionais e acadêmicas, ainda dedica tempo para estudar e debater as ideias marxistas e socialistas para um mundo mais justo e livre da opressão.

RONALDO BRAGHITTONI

Executivo de TI há mais de 15 anos, cofundador e CIO da empresa Sennit, especialista em BI e banco de dados, e evangelista do pensamento ágil e da transformação digital nas empresas. Nerd, fã de *Star Wars* (só que mais fã ainda de *Star Trek*). Claro, tudo regado a muito Iron Maiden.

PEDRO ARAUJO

Pedro Araujo é do interior de São Paulo. Graduou-se em Ciências Sociais pela Universidade Federal de São Paulo, onde participou de grupos de estudo sobre a relação entre Tecnologia e Sociedade, o que possibilitou uma entrada menos dramática no

ambiente totalmente desconhecido do desenvolvimento de software.

Nessa nova rota trabalhando com gerenciamento de projeto, teve contato direto com desenvolvedores front-end e com banco de dados, o que o cativou enormemente. Interessado em música, antropologia, sociologia, desenvolvimento front-end e banco de dados, utiliza grande parte do seu tempo se aprofundando nesse emaranhado de temas. Ou seja, se quiser conversar sobre música, sociologia e tecnologia, é só procurá-lo.

SOBRE O ARTISTA

Luis Buttes

Designer formado pela Universidade Presbiteriana Mackenzie, sempre gostou de desenhar e se apaixonou pelas artes gráficas. Já trabalhou como diagramador, ilustrador e diretor de arte, mas, atualmente, trabalha como *freelancer*. Também escreve sobre a cultura pop para alguns sites. Um nerd padrão dos anos 90: videogames, quadrinhos, livros e rock'n'roll.

Sumário

Versão: 21.9.13

Capítulo 1
INTRODUÇÃO

Para iniciarmos uma discussão sobre arquitetura de software, temos de definir o que é arquitetura. Adoro a definição de Martin Fowler:

"A indústria de software se delicia em pegar palavras e estendê-las em uma miríade de significados sutilmente contraditórios. Uma das maiores sofredoras é "arquitetura". Vejo "arquitetura" como uma daquelas palavras que soam impressionantes, usadas principalmente para indicar que estamos falando algo importante. Contudo, sou pragmático o suficiente para não deixar que meu cinismo atrapalhe o desafio de atrair as pessoas para o meu livro."

Realmente não existe a função de *Arquiteto*. O que existe é uma ideia que serve para agregar valor a uma função. Então, nós, desenvolvedores experientes, devemos ser pragmáticos o suficiente para aproveitarmos e agregarmos valor financeiro e profissional à nossa vida.

Pensando nisso, resolvi escrever este livro que ajuda a esclarecer o caminho para se tornar um verdadeiro e autêntico *Mestre Programador* – ou, como o mercado chama, um *Arquiteto de Software*.

Algumas disciplinas, além da codificação, são extremamente fundamentais para atingir o grau máximo! Algumas muito estranhas, como política, psicologia e outras disciplinas humanas contribuem para esta formação. Mas de que forma isto acontece?

Bom, espero que vocês não desistam da leitura ao saberem disso. Digo isso, pois toda experiência que possuo nessa área demonstrou que a maioria maciça dos desenvolvedores preza por um código mais performático ao mais humano. Mas o que quero dizer com isso?

Existem alguns pensadores de TI, como Uncle Bob, que prega a importância de um código que seja facilmente compreendido por

outros programadores, pois a única certeza que existe em desenvolvimento de software é de que o código será alterado, sendo este o maior motivo por atrasos e custos elevados. Além de Bob, temos Fowler pregando que um bom programador escreve código para outros programadores, e não para máquinas.

Já Jakob Nielsen extrapola as barreiras para o desenvolvimento de software ao alegar que o único meio de comunicação entre o usuário e o sistema é por meio de uma interface. Isso representa que um bom arquiteto deve repensar além da codificação.

Sobre a questão política, dedicarei um tópico específico, pois considero umas das mais importantes disciplinas para um mestre programador *arquiteto*.

Vamos nos apropriar muito do termo *Pirata* e *Marinha Mercante*. Caso não os conheça, recomendo fortemente que veja o filme *Piratas do Vale do Silício* antes de iniciar a leitura.

Resumidamente, no filme, *Marinha Mercante* é como a IBM, em que todos trabalham de terno engomadinho e cantam o "hino" da empresa. Já os *Piratas* são totalmente avessos a esse padrão, e amam arduamente o que fazem e não descansam enquanto não atingem seus resultados.

Essa forma de pensar em toda nossa história foi o propulsor de mudanças, no passado longínquo, em meados de 1.700, na época de ouro dos piratas. Eles não roubavam, apenas não aceitavam o monopólio das rotas oceânicas, e essa maneira de pensar fez o sistema daquela época ir evoluindo ao modelo atual, em que o mar é considerado área internacional.

Outro exemplo pirata mais recente é o Napster, no qual uma

única pessoa com pensamento pirata foi capaz de questionar o modelo da indústria fonográfica existente e criar algo que obrigasse a indústria musical a se reinventar. Antes do Napster, ninguém imaginava a possibilidade de comprar apenas aquela música de que gosta em vez de um álbum completo. Essa forma de agir e pensar é a única capaz de desestruturar e mudar o sistema para algo melhor, por isso entenderemos melhor a forma de pensar de um pirata.

Antes de continuar, gostaria de dizer que este livro não se destina a pessoas que adoram ou são do padrão *Marinha Mercante*, mas sim para quem pensa como um *Pirata*.

Os termos utilizados pela *Marinha Mercante* são, como disse Fowler, mais voltados para impressionar do que definir algo, então, prefiro trabalhar com a ideia de evolução tanto profissional quanto de experiência pessoal. Já que sou um grande fã de *Star Wars*, vejo uma correlação perfeita dessa evolução com a dos Jedis. Então, a partir deste momento, usarei o tema Star Wars como metáfora para a narrativa deste livro.

A ideia é simples: os Jedis são os verdadeiros programadores, os que farão a diferença! Eles iniciam sua carreira como *Youngling* ou *Padawan*, e depois evoluem para *Cavaleiros Jedis* e, por fim, *Mestres Jedis*.

Nem todos os desenvolvedores entram nessa classificação, em outras palavras, nunca serão Jedis. Para tornar-se um, deve-se amar o que faz, e percorrer um árduo e longo caminho. É como Martin Fowler já disse sobre Design Patterns:

"Quando você os apresenta a um grupo de novos

desenvolvedores, todos irão reclamar. Depois de um tempo, alguns irão amar e dizer que a utilização os mudou para melhor! Outros irão gostar e usar. Mas existe um grupo que irá odiar!" .

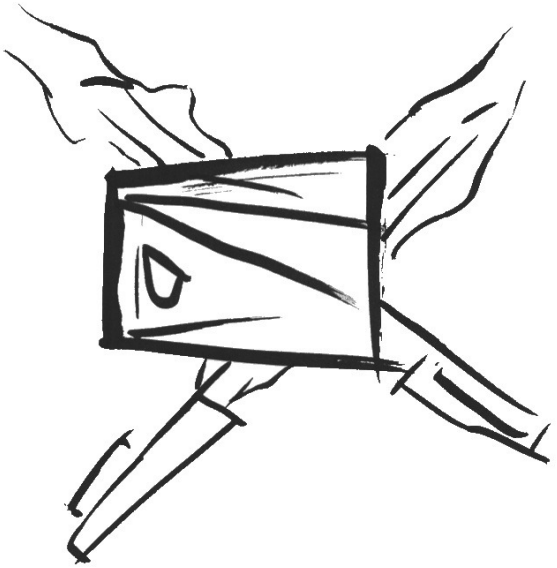
Esse último grupo são os que podem ser excluídos da categoria Jedi, pois não possuem em seu DNA as características analíticas e objetivas necessárias.

Outro ponto que entenderemos no livro é que o desenvolvimento de software está mais próximo da ciência humana do que da exata. O processo de análise e desenvolvimento (construção) de software é como pintar quadros: se solicitarmos para dois artistas distintos pintarem o quadro de uma pessoa, em ambos os resultados saberemos que os artistas retrataram a pessoa, mas não serão idênticos! Cada quadro terá as técnicas e características do pintor.

Tal fato é semelhante e acontece no desenvolvimento de software, pois se pedirmos para dois desenvolvedores distintos fazerem o mesmo software, o resultado final será distinto, mas conseguiremos identificar que ambos os softwares possuem a mesma finalidade. Entretanto, eles vão possuir técnicas e características dos seus desenvolvedores.

Segundo Philippe Kruchten: *"Programar é divertido, mas desenvolver um software com qualidade é difícil"*.

Em outras palavras, programar é um estado da arte! No capítulo *A arte de desenvolver*, aprofundaremos sobre o assunto.



Por Carlos Bueno

Capítulo 2

PIRATAS

"Melhor ser pirata do que marinheiro..." – Steve Jobs

2.1 POR QUE SER UM PIRATA?

Trataremos de forma alegórica o conceito de Piratas. Não é o

mesmo conceito utilizado em "Piratas de Computador", tão pouco para crime de pirataria! Como foi dito, é um termo alegórico. Temos de entender que a correlação é feita na essência do pensamento e transportada para o cotidiano de um desenvolvedor, para que se faça entender o espírito de como ir além... Inovar!

Lembre-se de que, assim como um pirata, estamos em uma jornada, sozinhos em um mar com toda marinha mercante – as grandes corporações –, que ditam as tendências e as regras.

Para conseguirmos sozinhos vencer a marinha e sermos vitoriosos, temos de agir como Piratas! Vamos entender melhor a ideia mais adiante. No momento, precisamos apenas saber que, agindo como Marinheiros, apenas seremos domados e estaremos mais distantes para (se é possível) alcançar nossos sonhos.

2.2 COMO AGEM OS PIRATAS?

"Bons artistas copiam, grandes artistas roubam." – Picasso

Picasso não quis dizer em roubar no sentido estrito da palavra, mas sim na inspiração do inovador; vestir o novo e revolucionário e fazer de tudo isso como fosse seu. Se observarmos grandes nomes de sucesso, eles fizeram o mesmo! Na área da tecnologia, temos o revolucionário Steve Jobs, que entendia mais desse conceito do que ninguém, e conseguiu criar um império que se baseou em tornar seu o que era revolucionário e rejeitado por outros.

A concepção principal do termo Pirata está no conceito de que eles navegavam na rota dos navios mercantes e os saqueavam. Assim, a ideia de saquear que temos de ter em mente é a mesma da de Picasso e Jobs! Então, observar a Marinha mercante e utilizar

suas tecnologias revolucionárias como se fossem suas são atos piratas que darão vantagem sobre os marinheiros! Designaremos o termo *Marinheiro* para o programador com o perfil da *Marinha Mercante*.

Para avançarmos mais, é importante conhecermos mais sobre os Piratas! Acredito que exista alguma energia no universo que sempre me fez imaginar que o motivo que levaria um pirata a ter um tapa-olho era unicamente pela falta de visão, ou do olho, por algum confronto ao longo de sua vida pirata.

Acredito que vocês ficarão pasmos, assim como eu fiquei, ao saber que estávamos errados. Na realidade, os piratas usavam tal acessório devido à constante condição de mudança do ambiente interno para externo do navio. Mas espera aí, o que isso tem a ver? Qual o motivo?

O motivo é que nossos olhos se adaptam rapidamente quando há transição de um ambiente escuro para um ambiente claro. Mas o processo oposto pode levar mais de 20 minutos, tendo em vista que nossos olhos precisam regenerar os fotopigmentos.

Devido à constante alteração de ambiente (dentro do navio, que era quase uma escuridão completa, e fora, que era sob a luz do dia), o tapa-olho era utilizado para manter um dos olhos adaptado à escuridão. Assim, quando alternava de ambiente, ele trocava o tapa-olho de lado e poderia enxergar facilmente.

Um Mestre Programador, assim como o Pirata, deve estar acostumado a dois ambientes: o *greenfield* e o *brownfield*; ou seja, não deve estar sempre trabalhando com equipes que criam códigos perfeitos, com fácil manutenção e elegância. Deve também estar

acostumado a trabalhar com códigos ou equipes que estejam no pântano! Assim, estará habituado a lidar com momentos de dificuldade ou quando cair em equipes sem qualidade. Acredite, isto acontece!

Um Pirata deve ser democrático e agir cordialmente com todos os outros programadores, e compartilhar seu conhecimento, ensinando todos os seus semelhantes, independentemente de sua classe social, nacionalidade, opção sexual e qualquer outro cunho.

Ainda nesse sentido, deve-se sempre que possível avaliar seus códigos, transformá-los em frameworks e distribuí-los livremente para todos.

"No seu centro, havia uma república pirata, uma zona de liberdade no meio de uma era autoritária." – Collin Woodward

A república dos Piratas – Collin Woodward (2007)

Os verdadeiros em sua época de ouro agiam muitas vezes em conjunto, deixando de lado questões como nacionalidade. A tripulação tinha direito de voto podendo igualmente eleger um capitão, e também o depondo – se necessário –, além de tomarem decisões importantes em conselho aberto. Isso era totalmente o oposto ao encontrado na Marinha Mercante naquela época. Em uma época em que os marinheiros não recebiam qualquer tipo de proteção, os piratas possuíam, por exemplo, benefício de incapacidade.

2.3 CRIANDO UM ESPAÇO PIRATA

O conceito "Pirata" não é apenas empregado para programadores, mas para empresas que têm o mesmo tipo de pensamento. Geralmente, são empresas criadas por Piratas que querem criar espaços democráticos, nos quais querem lutar em igualdade com grandes empresas da Marinha Mercante.

Não pense que uma empresa é pirata apenas por dar diversão e comida à vontade para seus colaboradores. Esse tipo de empresa adota o modelo político do pão e circo, assim todos trabalham felizes, produzem, mas no final permanecem na Marinha Mercante.

Algo extremamente interessante sobre os Piratas, em sua Era de ouro, era o fato de atacarem navios negreiros e transformarem os escravos em membros de sua tripulação, com igualdade de direito. Inclusive, muitos chegaram até a serem capitães de naus. Assim como os verdadeiros Piratas, os programadores não enxergam diferenças no conhecimento, e lutarão sempre para que os mais oprimidos tenham condições de lutar igualmente com os mais abastados.

O ponto fundamental que diferencia um Pirata de um programador tradicional é a questão democrática. O Pirata ouve seus companheiros e os mantêm no mesmo *status quo* em qualquer debate. Além disso, incentiva-os a participarem em todas as decisões.

Uma boa equipe pirata possui seu capitão, que, assim como nas leis piratas, deve ser eleito por todos e pode ser destituído a qualquer momento ao haver uma nova eleição. O capitão não está

acima de qualquer outro membro da tripulação, apenas está temporariamente em posição de decisão para que as ações do grupo sejam coordenadas e eficazes.

Então, imagine nossa equipe sem líderes fixos, que, na realidade, seriam chefes. Como seria um cenário assim? Sabemos que muitos projetos acabam tendo insucesso devido a vaidades pessoais para simplesmente massagear o ego. Então, manter líderes fixos e que não têm aprovação dos membros eleva exponencialmente a possibilidade de insucesso.

Além disso, estamos suprimindo possíveis tripulantes com potencial, limitando-os a meros operadores de tecnologia, simples executores. Isso, além do insucesso, leva os profissionais a se sentirem reduzidos e incapazes, e a procurarem novos horizontes por se sentirem ineficazes.

Sabemos também que a maioria das reclamações sobre emprego na área de TI está relacionada à insatisfação sobre as realizações do profissional dentro do ambiente de trabalho, gerada pela falta de espaço para expor suas ideias, para aprender novas tecnologias e para aplicar seus conhecimentos no dia a dia. Isso se trata de uma variável de fácil controle, que não depende de algo escasso como, por exemplo, recurso financeiro, mas sim de decisão política. Mesmo assim, a maioria dos empresários ou profissionais que estão na posição de "Capitães" acaba por não perceber a verdadeira natureza dessa insatisfação, perdendo seus recursos – ou patrimônio - humanos, acarretando em insucessos seguidos em seus projetos, o que leva também ao declínio financeiro da companhia.

Sabendo que, agindo como Mercante, terei insucesso,

como poderia ter uma estrutura Pirata para conseguir lutar contra o resto do mundo?

Primeiro, esqueça os modelos existentes. Desaprenda-os, pois muitos vão nos limitar. Após isso, lute para que consiga ser ouvido e para ouvir também. Neste momento, podem ocorrer alguns problemas, como: sou apenas um programador da equipe e ninguém me dá o espaço devido na estrutura mercante. Nesse caso, a demissão é a única opção restante. Tenha certeza de que encontrará uma empresa na atualidade que lhe dê a possibilidade de ouvir e ser ouvido.

Pode ser que você seja um líder, mas que seus superiores o impeçam de qualquer tipo de transformação. Este também é caso de procurar um outro emprego. Você pode achar irônico o que estou dizendo de forma simplista, mas ser Pirata é uma opção, então, quem optou por ela quer ter um caminho de sucesso, e ter sucesso começa com ter possibilidades. E, nos casos citados, estaríamos, como dizem, "dando murros em ponta de facas".

O próximo passo ao entrar em uma empresa que lhe proporcione um espaço mais democrático é tentar romper as barreiras hierárquicas existentes. Não digo que não deva existir um diretor etc. Digo que, na equipe, as posições não devem existir e, se existirem, será apenas por questão salarial. O interessante é mostrar democraticamente que todos trabalhando em conjunto, com o poder de opinar e dizer qualquer coisa abertamente, será benéfico aos projetos e a empresa. Sabemos que, quando não há espaço para falar abertamente sem represálias, iniciam-se as intrigas, e não há algo mais prejudicial do que uma tripulação envolvida em intrigas.

Após romper as hierarquias, inicia-se um processo de humanização da equipe. Para tanto, todos devem ter espaço para opinar sem ser oprimido, tendo como regra fundamental: *nunca criticar uma ideia*, pois uma ideia, por mais absurda que seja, deve ser ouvida e respeitada, e depois votada.

Elege-se sempre a melhor ideia. Caso haja críticas a uma, o único resultado será desestimular as pessoas a darem as suas, pois elas serão refutadas. A ideia simplesmente do voto não ridiculariza ou diminui a pessoa, e ainda possibilita que qualquer um tenha sua ideia aceita.

No momento em que as ideias começam a ser colocadas livremente, é a hora de a equipe escolher democraticamente um capitão. Este guiará todos em suas execuções. Entretanto, tenha em mente que um capitão não pensa em tudo; quem pensa é a tripulação, ele apenas coordena a execução após tudo ser definido e, a partir desse momento, todos os Piratas o seguirão até o fim.

A equipe em conjunto com o capitão iniciará o processo de estímulo do conhecimento. Caso você seja um Pirata solitário que não possui uma equipe, esse fato não o impede da busca pelo conhecimento, mesmo que a melhor forma de adquiri-lo seja sempre em conjunto. Isso porque um poderá ver a fragilidade do companheiro, e ajudá-lo a enxergar e a melhorar os seus pontos fracos. Além de ter uma aderência maior da informação aprendida, também pode disciplinar os que, por algum motivo, sozinhos não conseguem buscar conhecimento. Isso prova que a colaboração é uma marca fundamental de um programador Pirata.

O último passo que trataremos é sobre como devemos enfrentar o trabalho. Para um Pirata:

"Aquele que é mestre na arte de viver faz pouca distinção entre seu trabalho e seu tempo livre, entre sua mente e seu corpo, entre a sua educação e sua recreação, entre o seu amor e sua religião. Distingue uma coisa da outra com dificuldade. Almeja, simplesmente, a excelência em qualquer coisa que faça, deixando aos demais a tarefa de decidir se está trabalhando ou se divertindo. Ela acredita que está sempre fazendo as duas coisas ao mesmo tempo." – Domenico de Masi (*O Ócio Criativo*, 2000)

Masi diz que, além da arte de viver, é preciso amar nossa profissão. Partindo dessa premissa, um Pirata deve quebrar esse paradigma do trabalho ser apenas uma tarefa para sobrevivência. Um bom Pirata não diferencia nada, ou seja, a qualquer momento pode estar fazendo exercícios físicos, estudando e até mesmo trabalhando, não fazendo distinção do senso comum para evitar a limitação.

Enquanto pessoas comuns perdem seu tempo no transporte individual, passando horas no trânsito, nós utilizamos o transporte público para aproveitar o tempo para adquirir conhecimento. Sendo mais ousados e sábios (quando possível), usamos uma bicicleta para aproveitar o tempo de locomoção e para fazer exercícios. Isso é sobre não diferenciar, não se limitar. Essa mensagem deve estar na mente de todos que querem seguir o caminho para se tornar um mestre, não apenas em desenvolvimento, mas também na vida.

Existem estudos dizendo que os *insights* ocorrem geralmente em mentes à deriva:

"Embora a divagação da mente possa prejudicar nosso foco imediato em alguma tarefa específica, ela funciona a serviço de

resolver problemas importantes para as nossas vidas." – Daniel Goleman (*Inteligência emocional*, 1995)

Como dissemos, por exemplo, uma atividade como andar de bicicleta, além de substituir uma academia, também mantém a mente à deriva. Assim, há a possibilidade real para resolução inovadora de questões não resolvidas, pois uma mente a vagar permite que nossa essência flua. Enquanto nossas mentes divagam, nos tornamos melhores em qualquer coisa que dependa de um lampejo de insight, de jogos de palavras criativos a invenções e ideias originais.

Essas atividades, segundo Goleman, são de extrema importância para o nosso cérebro, pois ele necessita de descanso e dormir. Pasmem: não é a atividade que realmente o descansa, o que acontece é que, ao dormir, ele continua trabalhando sem parar e, para conseguirmos desligá-lo das atividades nas quais estamos ligados, devemos fazer algo prazeroso, como: pintar, andar de bicicleta, passear em um parque, ir a uma peça de teatro, entre outras atividades, que só o próprio sujeito pode identificar.

2.4 PIRATAS INOVAM

"A conspiração ganhou força quando os 'desenvolvedores cowboys' da contracultura, combinando suas percepções e atitudes libertárias, de beats, hippies, usuários de alucinógenos, roqueiros, hackers, cyberpunks e visionários eletrônicos partiram para o Vale do Silício e realçaram o maravilhoso roubo de mentes, desenvolvendo o grande equalizador: o computador pessoal." – Dr. Timothy Leary (*Chaos and cyber culture*, 1994)

Até agora falamos sobre ser Pirata, mas qual diferença entre ser um Pirata ou estar na Marinha Mercante?

A diferença é inovar. A evolução do sistema capitalista depende de atitudes piratas, pois a tendência analisada até o presente momento é a criação de monopólios que dificultam a evolução da sociedade. Sem (ou com pouca) concorrência, grandes empresas podem esperar muito tempo para inovar ou tão pouco pensar em modelos melhores. Isso porque estas ações geraram custos que não são aceitáveis devido à inexistência de concorrência.

É exatamente nesse momento que entra a "atitude" pirata. Segundo ela, em vez de se vender às grandes corporações, tenta-se inovar para desestabilizar o sistema e gerar novas empresas que, com a inovação, conseguirão derrubar o monopólio. Na realidade, o que um Pirata quer não é o fim do capitalismo, mas apenas a sua evolução.

Historicamente, assistimos a tais mudanças sem percebê-las nitidamente. Os piratas em sua Era de ouro, de 1.700 a 1.720, desestabilizaram o monopólio das rotas marítimas, resultando em muitas mudanças na relação entre o homem e o mar. Já nos anos

80, eles quebraram o monopólio da tecnologia criando computadores para todos e, na década de 90, criando o Napster, que mudou a indústria fonográfica. Também nos anos 90, iniciou-se a criação em massa de softwares livre, o que encadeou a possibilidade de termos os smartphones com muitos aplicativos gratuitos. Nada disso foi pensado pela Marinha Mercante, foi a atitude pirata que guiou todos esses grandes acontecimentos.

Além desses exemplos, o livro *O Programador Apaixonado*, de Chad Fowler, traz um trecho fantástico sobre ser visionário e o retorno de sua escolha:

"Quinze anos atrás, uma escolha de baixo risco seria aprender a programar em COBOL. Claro, também havia tantos programadores COBOL com quem concorrer, que o salário médio nessa época já não era fenomenal. Você podia facilmente encontrar trabalho, mas não seria algo tão lucrativo. Baixo risco. Baixa recompensa.

Por outro lado, se àquela época você tivesse decidido investigar a nova linguagem de programação da Sun Microsystems, o Java, durante um tempo talvez fosse difícil encontrar um emprego em algum lugar que estivesse usando essa linguagem. Quem iria saber se realmente faria algo em Java?

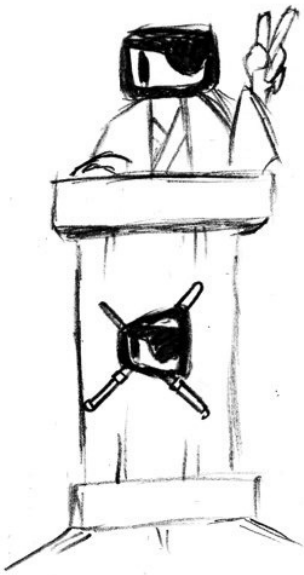
Mas se você estivesse de olho em como estava a indústria naquela época, da forma como a Sun estava, você poderia ter visto algo especial no Java. Poderia ter sentido que aquilo se tornaria grande. Investir naquela tecnologia logo no início faria de você um líder em uma grande e influente tecnologia.

É claro, nesse caso você teria se dado bem. E se tivesse jogado suas cartas corretamente, seu investimento em Java teria sido muito lucrativo. Alto risco. Alta recompensa."

Por Carlos Bueno

Capítulo 3

POLÍTICA



3.1 PRODUTO DO MEIO

Para um aprendiz se tornar um Mestre Programador, ele precisa compreender uma coisa, uma das coisas mais importantes nessa jornada: o ser humano vive em sociedade. E essa sociedade

não é digital, artificial e composta por perfis criados pela tecnologia. A nossa sociedade é sim muito humana. Sendo humana, ela é composta de relações entre pessoas de carne e osso, que têm interesses distintos, habilidades distintas, defeitos distintos e, acima de tudo, possuem diferentes níveis de empatia com as pessoas ao seu redor.

A capacidade humana de se relacionar, resolver ou criar problemas, produzir novas coisas e de se organizar em sociedade (seja um país, uma cidade ou uma empresa) nada mais é do que fazer **política**. Isso mesmo, todas as nossas ações são ações políticas. E aí está a chave para o sucesso de qualquer Mestre Programador, como de qualquer ser humano.

Durante toda a nossa vida, fomos educados pelos nossos pais, amigos, vizinhos, professores, parentes etc. a entender que política é um negócio sujo, feito por pessoas desonestas. Você já deve ter dito ou ouvido diversas vezes as seguintes frases: "Eu não me envolvo com política", ou "Eu nunca precisei de política para nada", ou ainda "Eu não sou político".

Essas frases são reflexos de uma visão limitada de política, construída na ideia de que ela seria apenas a ação de governar uma cidade ou país, de fazer as suas leis, e de ganhar dinheiro e poder a partir disso.

Um famoso mestre grego, Aristóteles, trabalhou muito bem a ideia de que o homem é um "animal político". Essa ideia reside no fato de que todos nós, seres humanos, não vivemos sozinhos e precisamos dos outros para satisfazer nossas necessidades, sejam elas materiais (alimentação, vestuário, abrigo) ou emocionais (relacionamentos, diversão). Essa relação entre as pessoas cria uma

sociedade que, quanto maior e mais desenvolvida, se torna mais complexa e dependente de uma organização maior e uma participação mais efetiva de seus membros. Já que não conseguimos viver sozinhos, e nossas relações são cada vez mais dinâmicas, interdependentes e complexas, Aristóteles deduziu que o homem é naturalmente político.

Sendo assim, o aprendiz que quer chegar a Mestre Programador precisa entender que ele faz parte de uma sociedade, que ele não é imune ou independente ao que acontece à sua volta. Quando um Mestre Programador elabora um programa, ele tem em mente que está fazendo aquilo para alguém. Ele sabe que suas habilidades não servem apenas para ganhar dinheiro ou fazer a sua empresa crescer, mas servem para tornar a sociedade melhor, mais desenvolvida e justa.

O Mestre Programador sabe que ele é um político e que as suas atitudes podem mudar a vidas das pessoas para melhor ou pior. Tudo depende das decisões que ele tomar e da forma como ele encarar a sua vida dentro da sociedade.

Quando ampliamos o nosso conceito de política e passamos a entender que também somos políticos, independente se temos ou não um cargo dentro de um governo, uma nova fase inicia-se na vida do aprendiz. É a partir daí que entendemos que, enquanto algumas pessoas entendem que são políticas e utilizam as suas capacidades para prejudicar outras pessoas e tornar o mundo um lugar pior – seja desviando dinheiro de uma obra pública, enganando um cliente na compra de um produto, recebendo um troco a mais e não devolvendo –, o aprendiz que pretende ascender ao grau de mestre usa suas capacidades para produzir uma

sociedade melhor.

3.2 ANIMAL POLÍTICO

A partir dessa primeira constatação, de que o homem é um animal político e de que política não é aquele mundinho das pessoas que disputam eleições e ocupam cargos públicos, precisamos avançar nos mistérios que tornam o aprendiz programador em mestre.

O nosso segundo ponto pode ser tirado dos ensinamentos de outro grande mestre, chamado Rousseau. Em sua riquíssima e vastíssima obra, encontramos a seguinte frase: *"O homem nasce bom e a sociedade o corrompe"*.

Para desvendarmos o nosso segundo ensinamento, proponho que mudemos a palavra "corrompe" por "transforma". Desta forma, nós nascemos e vivemos em sociedade, pois somos animais políticos, mas essa sociedade pode nos transformar. Mas transformar em quê?

Pode ser para muito melhor ou muito pior, mas isso dependerá de cada um. O que importa para o nosso aprendizado é entendermos que, como políticos, somos parte da sociedade, e essa sociedade pode transformar as pessoas! Sendo assim, chegamos à conclusão de que nós podemos transformar as pessoas por meio de nossas ações e, transformando as pessoas, podemos transformar toda a sociedade!

De que forma um Mestre Programador transforma o mundo? A resposta é simples: optando por colocar em prática atitudes piratas. Entretanto, não estamos falando da prática de pirataria,

mas sim do conceito apresentado no livro: liberdade e inovação.

Você pode estar se perguntando: como assim? Imagine-se como um mestre programador: você sabe que é um político, pois vive em sociedade e depende de milhares ou até milhões de pessoas para poder ter suas necessidades realizadas (pense por quantas pessoas o arroz de seu alimento teve de passar para chegar até você, ou mesmo o mouse que você usa). Você também sabe que as suas atitudes podem transformar as pessoas, e são essas pessoas que formam a sociedade.

Desta forma, um Mestre Programador sabe que pode melhorar ou piorar a sociedade. Nesse cenário, ele escolhe melhorá-la e acaba colocando práticas piratas em execução. Um exemplo de atitude pirata seria o mestre criar um código e disponibilizá-lo, para que outras pessoas possam utilizá-lo livremente. Uma atitude simples como essa permitirá que outro programador que não tenha as mesmas habilidades que você consiga criar uma coisa nova a partir da sua inovação. Pensando nos milhares de interações entre os seres humanos, quem sabe um código que você crie hoje e disponibilize para todos não retorne como um benefício para a sociedade por meio de uma grande invenção na área da medicina daqui há alguns anos, depois de ter passado por diversos outros programadores e pensadores?

A chave para um aprendiz programador se tornar um mestre está nessas duas habilidades: saber que ele é um político, e saber que ele pode transformar a sociedade. Muitas dúvidas podem estar surgindo agora na sua cabeça, pois o que a maioria das pessoas diz é que, para ser um Mestre Programador, você precisa mesmo é ser um gênio na programação. Nunca ninguém disse nada de ser

político para ser um grande programador. Mas é aí que está o ponto principal.

3.3 LIDERANÇA E ESTRATÉGIA

Para você entender melhor, vamos dar alguns exemplos. Imagine dois funcionários trabalhando em uma empresa. O primeiro é um gênio. O cara é muito fera e programa "melhor" que o segundo funcionário, mas é fechado em seu mundo. Ele não entende ou não quer entender que vive em uma sociedade e que, por isso, ele pode influenciar as outras pessoas. Já o segundo funcionário não é tão bom como programador, mas tem uma vantagem sobre o outro: ele é um mestre programador! Ele sabe que é político e, por isso, sabe que vive em sociedade e pode transformá-la.

É por isso que, mesmo com os problemas de comunicação que ele possa ter, ele se esforça para entender aqueles que estão à sua volta, relaciona-se bem com as pessoas e é muito agradável. Em um determinado dia, surge uma vaga de diretor nessa empresa. Qual dos dois será promovido?

Será o funcionário gênio que não se relaciona com ninguém e só cumpre as ordens, ou o Mestre Programador que não é tão bom, mas conhece todos da empresa (do presidente até o funcionário da limpeza), comunica-se com todos e influencia a vida das pessoas em seu ambiente de trabalho? Certamente, este último será escolhido e por um motivo muito simples: o Mestre Programador sabe que existe uma sociedade e que ele vai transformá-la. Já o programador que é um gênio, mas não tem esses conhecimentos, continuará sendo um gênio e cumprindo as

ordens do Mestre Programador. **As empresas querem líderes para liderar, e bons cumpridores de ordens para cumprir ordens.**

Podemos dar outro exemplo para você poder entender melhor. Imagine um programador trabalhando em uma empresa. Em um determinado dia, ele desenvolveu um código importante e quer apresentar a novidade para o chefe. Infelizmente, no dia em que ele tinha conseguido marcar uma reunião, acontece alguma coisa ruim com o chefe (o filho dele está doente no hospital, ou algum parente morreu, ou mesmo ele foi assaltado).

Um Mestre Programador sabe que não é o momento ideal para apresentar a novidade, pois isso requer tempo para ser compreendido. Ele consegue ter empatia com as pessoas que estão à sua volta e, assim, consegue tirar melhor proveito das situações e das pessoas.

Mas não se engane! O Mestre Programador não é nenhum oportunista ou uma pessoa que quer levar vantagem sobre as pessoas. Na verdade, ele é uma pessoa que tenta tirar o melhor das pessoas e das oportunidades.

Entender que o chefe está passando por um momento difícil e, por isso, ter o *timing* correto para recuar com uma ideia que será boa para ele, para o chefe e para a empresa, não é a mesma coisa que ser oportunista (no sentido mal da palavra), mas sim atencioso com o próximo e esperto para não queimar uma boa ideia. Esta que, devido às circunstâncias, iria para uma gaveta qualquer na mesa do chefe.

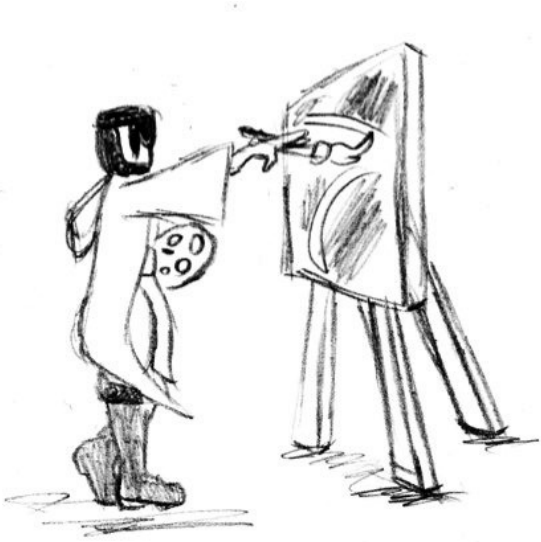
Quando um Mestre Programador é atencioso com seus colegas de empresa e não faz distinção entre o chefe e a pessoa que serve o

cafezinho, isso só mostra o desenvolvimento de sua liderança dentro da equipe em que trabalha.

Por Paulo Jesus

Capítulo 4

A ARTE DE PROGRAMAR



"Não há ignorância, há conhecimento." – Mestre Yoda

4.1 ARTESÕES, E NÃO OPERÁRIOS

É comum ouvir a comparação entre a construção de um software com a construção de uma casa. Tão comum quanto é

ouvir a comparação de uma equipe de desenvolvimento de sistemas com uma fábrica.

Nada poderia estar mais errado do que essas comparações.

Primeiro, desconstruindo a comparação entre o software e a casa, vamos entender que a construção de uma casa, por mais complexa que possa ser, tem um resultado esperado, conhecido. Ou seja, posso utilizar as plantas e equações civis para construir duas casas e, se bem executadas, as duas serão **exatamente** iguais.

Desenvolver um sistema está muito longe disso! Quando iniciamos o processo de criação do software, o resultado ainda é desconhecido. Faz-se a concepção em um processo de exploração e validações de hipóteses entre o desenvolvedor e o demandante, os dois lados corroborando para a criação de algo novo que vai atender às necessidades de negócio sob os preceitos técnicos. E o interessante é que o resultado só é conhecido quando concluído!

Como prova derradeira desse raciocínio, se uma especificação for encaminhada a dois times distintos de desenvolvedores, por mais que ela seja detalhada, os resultados serão **completamente** diferentes, mesmo que ambos atendam aos requisitos com 100% de aderência!

Com relação à comparação de um time de desenvolvimento com uma fábrica, a aberração é ainda maior. E o estrago também. Quando se entende que construir um software é o mesmo do que construir um carro em uma linha de montagem, bem ao estilo Ford, todo o processo se desenha de forma errada.

Vamos comentar um pouco mais sobre esse problema à frente (quando tratarmos sobre a gestão dos artistas), mas o

entendimento é que a eficiência de um modelo desses na construção de um software seria o mesmo do que pintar a Monalisa, colocando-se um quadro em branco no começo de uma esteira rolante cercada de pintores, onde cada artista daria uma pincelada enquanto o quadro caminhasse para o fim da esteira. Você realmente acha que sairia a Monalisa?

Desenvolver um sistema é um processo de exploração, não de extração! O resultado é desconhecido e a qualidade será intimamente relacionada a quem estiver desenvolvendo, muito mais do que a quem especificou. Sim, fazer bons softwares depende intrinsecamente de se ter bons desenvolvedores, ou seja, bons artistas!

Uncle Bob diz em seu livro chamado *Código limpo*:

"As más notícias são que escrever um código limpo é como pintar um quadro. A maioria de nós sabe quando a figura foi bem ou mal pintada. Mas ser capaz de distinguir uma boa arte de uma ruim não significa que você sabe pintar. Assim como saber distinguir um código limpo de um ruim não quer dizer que saibamos escrever um código limpo."

4.2 MAS POR QUE HUMANAS E NÃO EXATAS?

Se você não ficou bravo comigo até aqui, talvez agora fique. Desculpe mas se você é um desenvolvedor, você não trabalha com 100% com Exatas, você trabalha muito mais com **Humanas**.

Para iniciarmos sobre o assunto, devemos entender que, em desenvolvimento de software, há uma parte matemática. Mas, ao

dizer isso, não estamos negando a ideia de software ser arte? Definitivamente não. Assim como em um quadro ou uma escultura, o bom artista leva em conta muitos conceitos matemáticos como geometria, trigonometria, entre outros. Na realidade, o artista que possui tais conhecimentos de Exatas acaba produzindo obras fantásticas. Então, assim como um escultor, um programador deve entender de Exatas, ao mesmo tempo em que deve saber que um sistema não se limita a isso.

Como temos dito, o processo de construir um sistema é algo exploratório. É desvendar dos nossos demandantes aquilo de que eles **precisam** muito mais do que aquilo que eles *querem** (sim, nem sempre são as mesmas coisas). Envolvem-se habilidades políticas e psicológicas, treina-se a empatia, e cria-se um vínculo!

Nesse sentido, devemos perceber que as habilidades técnicas de um desenvolvedor o permite construir códigos performáticos. Entretanto, performance não está ligada às necessidades do mundo real.

Para criarmos algo realmente útil, devemos ter percepção das necessidades do usuário, e ser capazes de dar soluções criativas. Essas características não podem ser reproduzidas por fórmulas ou metodologias. Os resultados são distintos e característicos de cada artista, o que demonstra que realmente a criação de sistemas é uma profissão puramente artística.

Tanto Fowler quanto Bob trabalham fortemente com o conceito que escrever comentários é péssimo, e que um bom código é autoexplicativo. Mas, apesar de darem algumas dicas de como conseguir fazer um código sem comentários, ambos não conseguem criar um mecanismo que, ao ser aplicado, remova o

comentário dos códigos. Ainda nesse sentido, vemos muitos debates em grupos de discussão sobre o assunto, o que demonstra que não são todos os programadores que conseguem captar a essência passada por eles, por se tratar de algo não exato, mas sim puramente humano.

Quando lemos livros como o *Refatoração* de Martin Fowler, ou o *Código limpo* do Uncle Bob, vemos que a maior parte do tempo eles trabalham com conceitos filosóficos. Uncle Bob menciona o "cheiro do software", que é a sensação que se tem ao se ver algo não concretamente ruim, mas que faz os sentidos de um artista desenvolvedor ficarem em alerta a algum problema existente.

Mas e quanto ao conhecimento técnico? Não precisamos ser dominadores das tecnologias para sermos artistas programadores? Isso não é assunto de Exatas?

Claro, com certeza precisamos dominar a tecnologia. Mas, hoje em dia, a linguagem de programação está muito mais próxima de uma linguagem humana (como português ou inglês) do que a linguagem de máquina necessária para se criar um *driver*.

Da mesma forma que Carlos Drummond de Andrade precisou dominar profundamente o português para criar obras antológicas, como *A Rosa do Povo* ou *Amar se Aprende Amando*, um programador que deseja fazer códigos antológicos precisa, minimamente, ser um grande **mestre** nas tecnologias que pretende implementar.

E não para por aí: ao pintar um quadro, ou ao escrever uma poesia, conhecimentos de trigonometria, lógica etc. são sempre

necessários. Da mesma forma, o bom sistema vai depender de conhecimentos externos à programação em si, como economia, marketing, produção, habilidades sociais e outras (que trataremos nos capítulos sobre o caminho Jedi).

Quanto mais conhecimento o programador tiver no negócio em que seu software for implementado, mais fácil será o processo exploratório e melhor será a empatia dele com o demandante.

Por isso mesmo, as faculdades de tecnologia não são um bom ponto de partida. Nelas, você aprende **tecnologia**. E uma tecnologia defasada.

Mas a tecnologia, como um bom Pirata, você aprende por conta própria. Você estuda em livros, lê blogs, assiste a tutoriais, pergunta a seus amigos. Não na faculdade.

Use a faculdade para aprender sobre marketing, administração etc. Afinal, você acha que seus softwares vão atender a quem? À TI? Ledo engano.

Use a faculdade para lhe dar aquilo que o aprendizado de tecnologias não lhe dá, que são os conhecimentos de negócio.

Nosso caminho não é fácil. E quem pretende receber tudo pronto, ouvindo a um professor, talvez consiga até ser um bom programador, mas nunca será um **Mestre**. A maestria depende, dentre tantas outras coisas que falamos e falaremos, de um "espírito livre", da quebra do senso comum, da criação colaborativa.

Conforme descrito no Manifesto do Artesanato de Software:

"Como aspirantes a verdadeiros Artesãos de Software, estamos ampliando as fronteiras da qualidade no desenvolvimento profissional de programas de computadores através da prática e ensinando outros a aprender nossa arte. Graças a esse trabalho, valorizamos:

Não apenas software em funcionamento, ***mas software de excelente qualidade****.*

Não apenas responder a mudanças, ***mas agregar valor de forma constante e crescente****.*

Não apenas indivíduos e suas interações, ***mas uma comunidade de profissionais****.*

Não apenas a colaboração do cliente, ***mas parcerias produtivas****.*

Sendo assim descobrimos que, para atingir os objetivos à esquerda, os que estão à direita são indispensáveis."

4.3 GESTÃO DE ARTISTAS

Para entender um pouco mais sobre o conceito de que programar é uma arte, e não uma ciência exata e previsível, vamos desbravar as diferenças entre as abordagens de gestão.

Figura 4.2: Quadrantes de processos de gestão

Na figura, temos os 4 quadrantes que exemplificam os ambientes em que cada processo de gestão se aplica.

Temos que o processo *Simple* pode ser gerido com a aplicação das melhores práticas. Um exemplo seria um balcão de atendimento.

No *Complicated*, temos processos que são complicados, mas possuem resultados conhecidos. Para a gestão desses ambientes, usamos metodologias fabris, como o PMI, Prince2 e outras que partem de premissas conhecidas. O exemplo clássico desse ambiente é a construção civil.

No ambiente *Complex*, temos nós, os desenvolvedores de softwares. Os processos são de exploração e descobrimento, onde o método de gestão deve ser "inventado" a cada empreitada. Para esses casos, temos as metodologias ágeis e pragmáticas que pregam

o teste ("probe") do processo, a avaliação do resultado e a ação baseada em resultados.

Em *Chaotic*, tem-se a gestão de processos imprevisíveis. No caso desses ambientes, deve-se agir e pronto! Se a ação não deu o resultado esperado, aja de outra forma e continue o ciclo. Uma crise climática seria um exemplo.

O ponto interessante é a mancha negra ao centro. A chamada *Disorder*, que é quando a metodologia de um ambiente é usada em outro! Na maioria das vezes, temos empresas usando PMI e Prince2 para o processo de criação de softwares e de soluções ainda não conhecidas. O resultado é... confusão! E isso tem um motivo clássico.

Quando se estabelece um processo de gestão baseado em boas práticas (PMI, por exemplo), tem-se a definição de um cronograma a partir de premissas. Dado que o resultado é conhecido, sabe-se das premissas envolvidas, então se podem estabelecer prazos, metas, responsabilidades etc.

Mas o problema da criação de um software é que as **premissas não são conhecidas**. Logo, todas as previsões e cronogramas são fadados ao fracasso. É o mesmo que dar a Michelangelo a tarefa de pintar a Capela Cistina sob um cronograma no Project! Imagine o WBS:

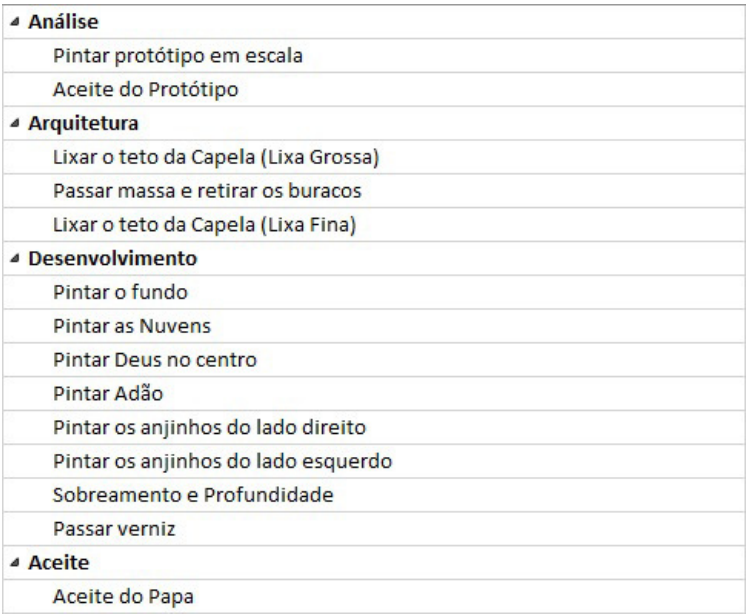


Figura 4.3: Cronograma

Não faz o menor sentido, certo? No aceite do "protótipo", ele já levaria a vida toda.

O processo foi formado por diversas pinturas de diversos artistas ao longo da história, tendo como a "obra prima" a pintura central de Michelangelo, que foi feita diretamente no teto da Capela, tendo sido vedada até o término (tanto que o Papa mandou Daniele da Volterra cobrir as "partes íntimas" do pessoal ao redor de Deus e Adão, porque não aceitou que Michelangelo tivesse deixado tudo à mostra).

Então, gerir um time de Mestres Programadores é o mesmo que gerir artistas, porque é isso que nós somos!

Temos prazos, sim. Temos orçamentos, sim. Mas temos de ter

espaço, margens e o entendimento de que o conceber um sistema é um processo artístico e humano, mais do que um processo fabril automático.

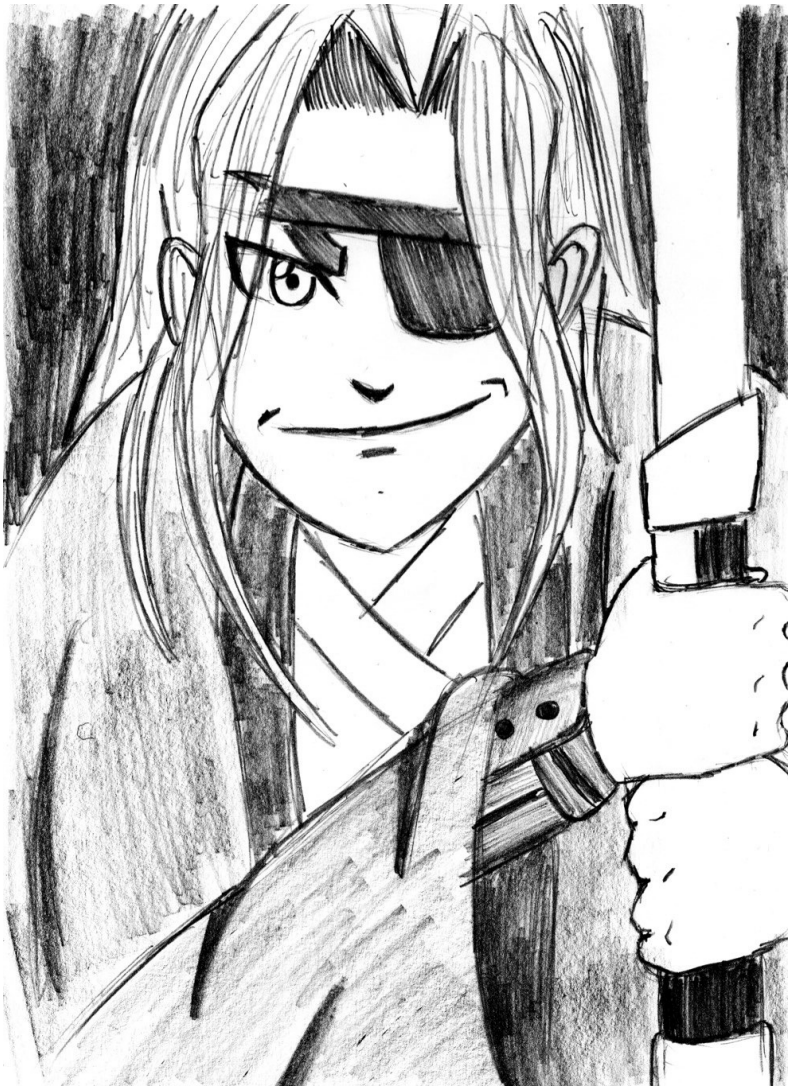
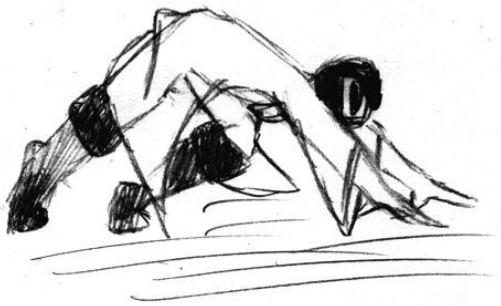


Figura 4.4: Pirata Jedi

Por Ronaldo Braghittoni

Capítulo 5

INICIADO



"Faça ou não faça. A tentativa não existe." – Mestre Yoda

Como havíamos visto inicialmente, vamos trabalhar com a ideia de evolução por estágios, graus, ou como desejar chamar. Usaremos as nomenclaturas dos estágios por que um Jedi passa até atingir o grau Mestre.

Lembrem-se de que não temos relação com a obra Jedi e, em alguns momentos, criaremos ou alteraremos alguns conceitos para adequar à realidade do livro. Porém, jamais fugiremos da ideia central, trabalhando com analogias de forma alegórica.

5.1 O INICIADO

"Assim como um exército e governos, a Ordem Jedi segue uma hierarquia para ajudar o fluxo de comando. Ainda que somos todos iguais na Força, os membros mais antigos possuem uma experiência que merece respeito por aqueles que ainda não atingiram tal nível" – Daniel Wallace (O Caminho Jedi, 2013)

Um iniciado desperta orgulho e esperança nos desenvolvedores Jedi, os quais vão observá-lo. Em pouco tempo, ele estará integrado com algum Jedi, formando uma "família". Geralmente, os escolhidos acabam integrando algum clã, como Javeiros, .Net, Ruby Rails (Emo), entre outros, que serão acolhidos por seus membros mais experientes, e guiados em sua jornada como aprendizes, até serem escolhidos na sua jornada como Aprendiz Padawan.

Geralmente, um iniciado desperta muita atenção nas pessoas em volta. Muitas vão procurá-lo para buscar sabedoria, e isso é um ponto que define bem que o desenvolvedor é um iniciado e que seguirá (ou não) a trilha rumo a se tornar um Jedi.

Todos iniciados devem saber que os três pilares Jedis são:

- Força
- Conhecimento
- Autodisciplina

Para se tonar um Mestre Desenvolvedor – ou seja, um Arquiteto –, temos de entender esses pilares em nossa realidade.

5.2 PRIMEIRO PILAR: CONHECIMENTO

Um profissional que quer trilhar o caminho deve sempre buscar conhecimento. Para um Jedi, *"não há ignorância, há conhecimento"*.

Assim sendo, nossa busca deve ser eterna, pois algo que não sabemos hoje é exclusivamente por não a conhecermos. Se avançarmos estudando, atingiremos o conhecimento que precisamos sempre.

Um iniciado deve estar sempre buscando fontes de pesquisa, integrando-se a grupos de discussões, buscando opiniões de profissionais mais experientes, e participando de eventos para poder observar e adquirir mais conhecimento. Essa prática deve se tornar um hábito, assim, refinaremos cada vez mais nossa capacidade de adquirir sabedoria.

Geralmente, um iniciado é um jovem entre 11 e 16 anos, e é nesse período que ele se destaca em TI, não utilizando a tecnologia, mas começando a compreender como pode mudá-la, adapta-la ou já criando pequenos códigos, ou quem sabe até sistemas revolucionários. Entretanto, a motivação que o leva a guiar esse caminho são suas escolhas políticas e sua base familiar, sendo que esta última está sendo devastada nos últimos anos.

Então, quem quiser incentivar um jovem a se tornar um iniciado deve guiá-lo na busca do conhecimento, a ser uma pessoa questionadora, e nunca calá-lo quando ele indagar sobre algo. É preciso conduzi-lo ao caminho da resolução do problema, para então elogiá-lo. Alguns estudiosos afirmam que 90% de nosso cérebro é formado até os 6 anos, e essa fase é praticamente o nosso nascer como “animal político”.

Quando o iniciado começa a se descobrir, ele inicia a busca pelo conhecimento de forma natural. Ao conhecermos pessoas que estão a trilhar esse caminho, é interessante apoiá-los, sendo a melhor forma apresentando boas fontes de conhecimento onde podem beber do néctar dos deuses e maravilhar-se com a possibilidade de criação que a área lhe proporciona.

Um fato importante é que vivemos na era da informação, mas isso não quer dizer conhecimento, pois hoje há tanta informação que, ao iniciarmos uma pesquisa, se não tivermos disciplina, terminaremos lendo algo extremamente diferente do que foi inicialmente objetivado. A era da informação pode se tornar inimiga da qualidade da informação e construção do conhecimento.

Para o iniciado que deseja alcançar o posto de Mestre, deve ter em sua cabeceira o livro *Código Limpo*, de Robert C. Martin. Esse livro mostra os princípios básicos que um Jedi deve possuir. Nele, você vai compreender que código possui cheiro; conseguirá, através da força, identificar os cheiros ruins (*code smells*); e saberá tomar as decisões corretas para corrigi-los.

Não se esqueça de tê-lo como livro de cabeceira, pois, de tempos em tempos, você deverá visitá-lo para aprimorar princípios que anteriormente eram obscuros e outros que a cada momento darão uma visão diferente.

5.3 SEGUNDO PILAR: AUTODISCIPLINA

A autodisciplina é a mais importante dos três pilares, pois é com ela que você vai criar hábitos de estudo para estar em uma

evolução constante. Temos de criar formas de conseguir fazer o que propomos. Se queremos aprender uma linguagem nova, por exemplo, temos de nos disciplinar a ponto de criar um hábito.

Para atingir este objetivo, é preciso três componentes: o primeiro é um gatilho para poder tornar o hábito automático; o segundo é a rotina; e o terceiro é a recompensa. A combinação deles é responsável por fazer seu cérebro dirigir, sem praticamente se esforçar muito. Fazendo uma analogia prática: ligamos o carro (gatilho), dirigimos (automático) e chegamos, por exemplo, cedo ao trabalho (recompensa).

5.4 TERCEIRO PILAR: FORÇA

A Força é o pilar mais conhecido por todos os fãs de *Star Wars*, e também de suma importância para um futuro Mestre Programador. O conceito de força é resumidamente o poder que nossa mente exerce sobre tudo. Em outras palavras:

"Tudo que a mente humana pode conhecer, ela pode conquistar." – Napoleon Hill (*Mente magnética*, 2014)

Os grandes pensadores conheciam muito bem o poder da Força. Ela foi usada por pessoas como Albert Einstein, ao afirmar que *"a imaginação é tudo. Ela é uma prévia das próximas atrações. A imaginação envolve o mundo.". Podemos observar isso também em Johann Goethe:* "quando uma criatura humana desperta para um grande sonho e sobre ele lança toda a força de sua alma, todo o universo conspira a seu favor" *(_Mente magnética*, 2014).

Existem muitos outros pensadores que falam sobre tal tema e que descrevem que suas descobertas nem sempre vieram

totalmente de métodos. Para sua descoberta, simplesmente em algum momento tudo brotava em sua mente como um *insight*.

Existem muitos livros que tratam do assunto e de algumas ordens herméticas. Não estamos falando de algo novo e surpreendente, mas algo que poucos utilizam; a Força nada mais é que uma lei natural.

Assim como a lei da gravidade, a Força está ai, não é nada sobrenatural. Assim sendo, ela não é algo simplesmente bom, podendo ser usada para o bem e para o mal. Por exemplo: a lei da gravidade estará agindo se pular em uma cama elástica, mas também funcionará em uma queda de avião; já a força, por ser uma lei assim como a gravidade, ela não consegue discernir algo bom de algo ruim. Ambas funcionarão sempre, independentemente dos seus resultados.

Sabendo isso, temos de aprender a utilizar a força para atingirmos nosso resultado esperado. Entretanto, devemos ter cuidado, pois se mal empregada, podemos atrair o oposto e podemos ser levados para o lado negro da força.

"Se você pensar que pode ou que não pode, de qualquer forma, você estará certo." – Henry Ford

5.5 INGRESSANDO EM UM CLÃ

A escolha de um clã é extremamente importante, pois, nesse momento, você estará escolhendo o que o guiará praticamente pelo resto de sua vida como desenvolvedor. Aconselho que a escolha não seja apenas subjetiva, algo por um simples gosto. Avalie o mercado e escolha a que lhe dará maior retorno.

Chad Fowler, em *O Programador Apaixonado*, aborda que, ao escolhermos, devemos ter em vista a relação Risco/Retorno, ou seja, Baixo Risco/Baixo Retorno, ou também Alto Risco/Alto Retorno.

Javeiros (java)

- **Pontos fortes:**

O clã Javeiros é uma comunidade composta de membros poderosos, que possuem muito conhecimento. Praticamente todas as boas literaturas de Orientação a Objetos são escritas e direcionadas para eles. Eles são percursores da boa prática, da utilização dos padrões de projetos, além de, no geral, levarem o desenvolvimento de forma bem artística.

Outro ponto que não pode ser deixado de lado é a grande comunidade livre. Existem praticamente frameworks para o que se imaginar.

- **Pontos fracos:**

Por ser umas das linguagens mais utilizadas do mundo (no momento em que escrevia, era a segunda), e uma das mais usadas pelo mercado *offshore*, isso acaba tornando o valor pago ao profissional mais baixo do que deveria ser, devido à alta quantidade de profissionais. Isso forma, como Marx chama, "o exército industrial de reserva", a força de trabalho excedente às necessidades da produção, garantindo ao capitalista que os salários sejam rebaixados.

dotNot (.net)

- **Pontos fortes:**

O clã dotNot não possui grandes membros. No geral, os profissionais são medianos, o que é um ponto forte, pois, se você for um iniciado, conseguirá se sobressair facilmente e engrenar em um mercado crescente. Nesse momento, o C# é a quinta linguagem mais utilizada, mas em crescente.

- **Pontos fracos:**

Os percursores da linguagem vêm da velha guarda do Visual Basic 5 e 6, que não era orientada a objetos. Logo, encontramos facilmente muitos artigos escritos por essas pessoas, e estes, apensar de serem C# – que é orientado a objetos –, não utilizam nada de orientação, além de estarem cheios de más praticas de uso.

Outro ponto fraco é a comunidade de frameworks livre que, apesar de estar em crescimento é muito pequena e com sistemas frágeis ainda.

Emos (Ruby)

- **Pontos fortes:**

O clã dos Emos, programadores Ruby, tem esse nome por programarem Ruby por moda e também a sintaxe de Rails ser semelhante ao modo de falar dos Emos. Mas, brincadeiras a parte, o ponto forte da linguagem é a alta produtividade. Além disso, muitos dos seus membros são de uma nova leva, o que provavelmente dará uma vida longa à linguagem.

Outro ponto forte é a linguagem ser *open source*, o que facilita sua adoção pelos desenvolvedores e empresas.

- **Pontos fracos:**

A linguagem gera códigos mais lentos, cerca de 20 vezes mais lentos quando comparados ao Java. Além disso, há uma certa demora com as atualizações e correções, sendo uma das mais lentas de todas.

5.6 CARACTERÍSTICA QUE IDENTIFICAM UM JOVEM JEDI

Quando se trata de identificar um Jedi emergente, facilmente encontramos algumas características presentes neles desde a sua tenra idade. Por exemplo, em jovens dedicados em trabalhos por algo maior que suas próprias preocupações pessoais, como ajudar pessoas necessitadas, o meio ambiente, entre outros problemas existentes na sociedade.

Esses jovens possuem grande controle emocional. Existem muitos estudos que constatam que possuir muita inteligência não o leva ao sucesso, tampouco nascer em família rica.

Além desse ponto, um futuro Jedi possui uma grande capacidade de se concentrar, e sabemos que atualmente essa capacidade é raridade. Sabemos que os jovens da geração do novo milênio são fortemente ligados à tecnologia, o que prejudica fortemente seu foco. Logo, sua capacidade de compreensão e imersão está totalmente prejudicada.

> *"...mensagens de texto (substituíram as ligações telefônicas como modo preferido de se manter em contato). Na nova regra social, ignorar as pessoas em torno, fixando-se num dispositivo eletrônico, tornou-se a norma, e não um sinal de indelicadeza."* – Daniel Goleman (*Liderança*, 2014)

Sabendo disso, facilmente conseguimos identificar um futuro Jedi, pois este não se adapta ao grupo "normal" de jovens nos dias atuais, uma vez que quer explorar o mundo de outra forma. Ele sente-se atraído pelas relações mais pessoais, e se distancia cada vez mais do impessoal.

5.7 QUANDO UM INICIADO ESTÁ APTO A SE TORNAR UM PADAWAN?

Um iniciado que vai seguir a área tecnológica como um Jedi deve compreender bem os três pilares que foram demonstrados anteriormente. Para entendê-los, nem todo iniciado percorrerá o mesmo caminho, pois não existe um caminho único. Pode ser que alguns não cheguem a se tornar Padawans, mas aprenderão que a Força deseja outro caminho para atingir suas expectativas e vontades.

Geralmente, um iniciado sente sua passagem quando começa a cooperar, desenvolvendo seus primeiros códigos em conjunto e conseguindo por em prática seu "kata", que vem aprendendo de forma exemplar. Já alguns compreenderão que ser Jedi não foi designado para ele.

Assim que superar esse período, certamente você vai conhecer algum mestre Jedi. Este pode ser algum amigo que programe ou um professor, e será seu tutor. Normalmente, é a pessoa que vai lhe empolgar.

Esse mestre não necessariamente é um programador que está fisicamente próximo, pode ser um escritor de tecnologia do qual gosta e cujos livros e artigos você segue. Ele lhe preparará para sua maior busca, que é se tornar um Mestre Desenvolvedor.

5.8 ROTINA DE TREINO

- Leia um 1 livro a cada 3 meses e, caso seja possível, 1 livro técnico a cada 6 meses.
- Comece a interagir com as comunidades de desenvolvedores, fazendo perguntas e tentando responder as feitas por outros usuários.
- É extremamente importante que um iniciado passe a interagir com o ecossistema cujo quer fazer parte. A interação ao participar é a melhor forma para isso, pois assim passa a conhecer o que existe de melhor no meio, os jargões utilizados, além, é claro, das nerdices, como: filmes, séries, mangás, comics etc.

Eventos e comunidades

Existem várias comunidades e eventos. Citaremos alguns, mas sugiro uma busca para conhecer outros:

Comunidades:

- *Viva ao Linux*, uma comunidade onde você poderá encontrar dicas para Linux, scripts, entre outros – http://www.vivaolinux.com.br/.

- *CodePlex*, um repositório de softwares *open source* voltados para plataformas Microsoft, sendo que os códigos variam entre C#, Sharepoint, VB.NET, C++ e outros – http://www.codeplex.com.

- *Software Livre Brasil*, uma comunidade de software livre do Brasil – http://softwarelivre.org.

Eventos:

- *Fórum Internacional do Software Livre* (FISL), um dos maiores fóruns de software livre do mundo, que acontece em Porto Alegre.

- *Ruby Conference* – http://www.rubyconf.com.br/.

Nerdices

"Uma coisa fundamental na sua vida é lembrar de uma coisa: você não deve crescer nunca. Quando você cresce, você perde a criatividade." – Chuck E. Cheese (fundador da Atari)

Leia mais sobre em http://www.techtudo.com.br/noticias/noticia/2013/01/nolan-bushnell-fala-sobre-steve-jobs-importancia-da-inovacao-e-o-futuro-dos-games.html.

Jogue sempre. Mesmo conforme cresça, não deixe de jogar games.

Assista a séries boas, como:

- The Big Bang Theory;
- Fringe;
- Arquivo-X;
- Millenium;
- Silicon Valley;
- Star Trek;
- Doctor Who;
- The It Crowd;
- Chuck;
- Numb3rs.

Jogue RPG de papel

O RPG de papel é importante, sendo seus benefícios variados, como por exemplo:

- Hábito da leitura;
- Interpretação;
- Capacidade cognitiva;
- Estratégia;
- Criatividade;

- Socialização;
- Matemática.

Se não joga ainda, procure um grupo de jogadores. Caso já jogue, procure sempre estar em um grupo ativo.

Por Carlos Bueno

Capítulo 6

PADAWAN

"Em um lugar escuro nos encontramos, e um pouco mais de conhecimento ilumina nosso caminho." – Mestre Yoda

6.1 O QUE É UM PADAWAN?

Em nosso livro, Padawan é um termo emprestado do filme *Star*

Wars, e será utilizado apenas de forma alegórica, não tendo fidelidade com o original do filme; Usamos essa analogia tendo em vista que a grande maioria dos profissionais que querem ser arquiteto conhece esse filme.

Padawans são aprendizes Jedi, sendo treinados por um Cavaleiro Jedi ou Mestre Jedi até sua graduação a Cavaleiro. Um fato interessante sobre um Padawan é que, no seu rito de passagem de Aprendiz a Cavaleiro, ele deve construir seu próprio sabre de luz.

Um bom Padawan deve possuir seu "Guru", Mestre Jedi, o qual vai seguir e guiar sua jornada para cavaleiro, sendo sua referência e ajudando-o a aprimorar seu conhecimento. Alguns Padawans mais avançados buscam mestres renomados, e os seguem por meio de livros, artigos, entre outros meios existentes.

Um aspirante a Cavaleiro deve entender que, se não tivermos foco, não chegaremos a lugar algum. Esse é um ponto focal para um Jedi e será trabalhado evolutivamente no decorrer do livro.

Daniel Cates, quando muito jovem, descobriu sua habilidade com games, e logo começou a jogar *Command & Conquer*. Sua dedicação era tão grande que decidiu muitas vezes não brincar com outras crianças para passar horas trancado em seu quarto jogando.

Passado algum tempo, quando estava na escola, passou a faltar em algumas aulas para ficar horas e horas jogando Campo Minado, sendo que de início era um péssimo jogador. Mas, depois, descobria facilmente todas as minas em pouco mais de 1 minuto, uma conquista que não parecia possível quando começou a jogar.

Um pouco mais velho, em torno de seus 17 anos, começou a jogar Poker online, e não demorou muito para faturar muito dinheiro. Com seus 20 anos, já havia recebido mais de 5 milhões de dólares em prêmios. Essa conquista foi marcada não por sorte, mas por ter suado muito e focado seus esforços em jogar várias partidas simultâneas.

Segundo Daniel Goleman: *"Nossa capacidade de atenção determina o nível de competência com que realizamos determinada tarefa"*.

Além da atenção, existem outro pontos que determinam nossa capacidade de atingir algo que mentalizamos. A fraqueza pode destruir uma vida ou uma carreira, enquanto a força eleva o sucesso.

Se observarmos as grandes empresas, como o Google, elas incentivam seus funcionários a parar por alguns minutos durante o dia e prestar atenção na sua própria respiração. Essa rotina foi criada na empresa para que o circuito cerebral responsável pela concentração seja ativado.

Seguindo esse caminho, um Padawan deve tirar em torno de 10 minutos do seu dia para não fazer nada, trancar-se em algum lugar e ficar solitário, sem contato com e-mail, sem receber ligações, reuniões ou qualquer outro tipo de distração. Além disso, deve dormir bem; nível de escuridão e som ajudam a melhorar o sono.

Segundo Daniel Goleman, o foco é dividido em três tipos: interno, externo e no outro. O foco interno é a capacidade de se concentrar, independentemente do que aconteça à sua volta. O externo é a capacidade de conseguir captar tudo à sua volta, prestar

atenção em tudo o que está ocorrendo, e analisar o ambiente. Já o foco no outro é conseguir prestar atenção em alguém, assim conseguindo entender as pessoas ao seu redor, e perceber quando é preciso motivá-las.

Um grande especialista em foco externo, Steve Jobs, uma vez disse: *"As pessoas não sabem o que querem, até mostrarmos a elas"*. Isso demonstra que, além de prestar atenção nas pessoas, ele era exímio em propor soluções que iam de encontro com suas necessidades.

Além de trilhar o caminho da Força por meio de foco, existe uma outra capacidade que devemos trabalhar para nos elevarmos a ponto de sermos um bom Cavaleiro: a habilidade de perceber os sentimentos de outra pessoa e agir de maneira a enfatizá-los mais ainda. Isso é empatia, e ela é o poder de exercer o controle sobre as emoções do outro, sendo a essência da arte de relacionar-se.

6.2 O PODER DO DOMÍNIO

"Se você quer permanecer relevante, vai ter que ir fundo no domínio do negócio dentro do qual você está." – Chad Fowler (*O Programador Apaixonado*, 2014)

Poucos desenvolvedores dão valor ao **domínio**, mas mal sabem que, quando se tornarem grandes desenvolvedores, saberão a diferença que faz compreender bem um domínio. Isto é, se um dia se tornarem, pois não é certeza que todos os profissionais em um momento de sua carreira terão a compreensão. Existem alguns que estão há muitos anos na área e já ouviram falar muitas vezes sobre o assunto, mas não sabem verdadeiramente o que é.

Domínio é um conjunto de particularidades que determinam um grupo de problemas ao qual devem-se dar soluções. Para se entender bem, Eric Evans prega que a equipe técnica e os usuários devem falar uma linguagem ubíqua, ou seja, todos devem entender o real significado de cada termo utilizado.

Foque no domínio

Entender o negócio do cliente e saber modelar o **domínio** são artes que fazem toda a diferença em um projeto, valendo-se de foco. Nesse entendimento, propague isso para toda a equipe, e garanta que todos os seus companheiros saibam o mesmo sobre o domínio que você, pois todos têm de entender perfeitamente do que se trata o problema do cliente e como resolvê-lo.

Deixar o entendimento do domínio de lado certamente levará o projeto ao fracasso.

Exemplo Real

Trabalhei em uma consultoria por duas semanas antes de partir para um trabalho em Angola. Nesse período, fui designado a ajudar uma equipe a finalizar um projeto que estava atrasado há muito tempo.

O projeto era um sistema de boletagem de operações financeiras, e estava sendo desenvolvido para um banco brasileiro. Então, comecei a observar que os desenvolvedores não compreendiam os termos utilizados pelo cliente, tampouco entendiam o negócio. A minha surpresa foi maior ainda quando um dia cheguei e um dos desenvolvedores me pediu uma ajuda. Nisso, ele me mostrou uma tela e falou: *"Aqui tem ativos, não sei o que devo ativar e inativar quando clicar aqui."*

Ao ouvir isso, quase cai da cadeira. Na hora me veio Eric Evans na cabeça; o mestre sempre esteve certo. Acontece que, no caso em questão, a palavra "Ativos" não tem o significado de inativo e ativo, mas sim de Ativos Financeiros, papéis da bolsa, como Petrobras, Vale etc.

"Você pode ser "apenas um programador", mas ser capaz de falar com seus clientes do negócio na língua de seu domínio de negócio é uma habilidade única." – Chad Fowler (*O Programador Apaixonado*, 2014)

Quando um Padawan começa a compreender e querer se

aprofundar nos domínios de negócio dos projetos em que está trabalhando, ele começa a ser recompensado, pois, além de ser perceptível seu papel focal no projeto, ele também começa a crescer como profissional. Além de perder certos medos, como o de ser demitido, pois sabe que, com o poder que possui em mãos, rapidamente encontraria outro emprego.

Entendendo melhor o domínio

A criação de um domínio é puramente artística, pois temos de conseguir captar e transformar os requisitos do cliente em um modelo. Por exemplo, uma entidade Cliente pode fazer parte de mais de um domínio. Como assim?

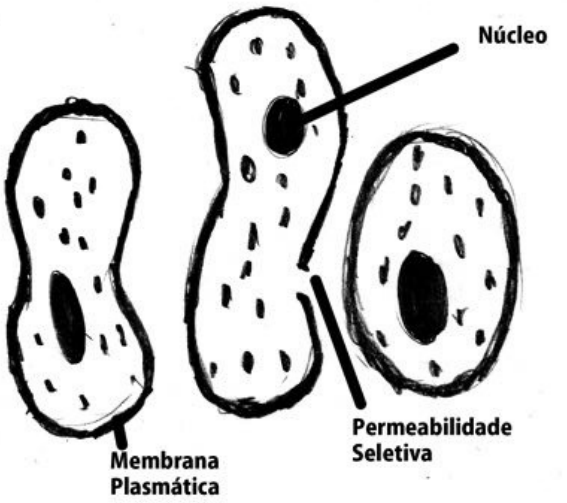
Bom, o domínio possui entidades que podem pertencer a um domínio, mas ser usado em outros. Por exemplo, pedidos podem fazer parte de um domínio chamado *Cliente* em uma determinada aplicação, mas pode ser necessário em vários outros domínios, como Fornecedores, Faturamento etc.

Entretanto, devemos nos atentar ao fato de que os domínios possuem núcleos, sendo que estes não podem ser utilizados por outros domínios. Os núcleos são as partes mais importantes de um domínio, o coração.

> Um bom Padawan se aprofundará no assunto seguindo o Mestre **Eric Evans**, em seu livro *Domain-Driven Design*.

Gosto da relação entre domínio e célula. Observe a figura a

seguir:



Uma célula possui em seu plasma materiais que são trocados entre células através da membrana plasmática, por meio da permeabilidade seletiva. Ela possui uma membrana que define seus limites e um núcleo que não pode ser compartilhado, e possui todo o material genético.

Assim como a célula, um domínio possui seus limites, mas pode permear seletivamente alguns elementos entre domínios. Ou seja, imagine uma entidade que pode ser usada por mais de um domínio. Mas também, como a célula, um domínio possui seu núcleo que contém seu material "genético" e não pode ser compartilhado.

6.3 AS VIRTUDES DE UM BOM

PROGRAMADOR

"A maioria de vocês está familiarizada com as virtudes de um programador. Existem três, é claro: preguiça, impaciência e soberba." — Larry Wall

O criador da linguagem Perl foi feliz ao proferir essa frase. Acontece que um Padawan não é um programador que segue esse estereotipo, ele deve ajudar a mudar ou evitar profissionais que se portem assim. Eles se sobressaem apenas por possuírem um desvio padrão de QI, mas não é suficiente para ser um profissional de sucesso.

Ter sucesso é a contramão dessas virtudes. Um Padawan nunca poderá, por exemplo, se gabar com um código por mais bem feito que esteja.

> Mas Luke sabia que não era um Jedi... ainda não. E o rigoroso programa de treinamento determinado por Yoda levara-o quase ao fim de sua resistência.
>
> — Pensei que estivesse em boa forma — balbuciou ele.
>
> — Mas por quais padrões, eu lhe pergunto? — disse o instrutor, zombeteiramente. — Os seus padrões antigos deve esquecer. Desaprenda, desaprenda!.
>
> **Luke Skywalker e Mestre Yoda (*O Império contra ataca*)**

Um programador Padawan que quer se tornar um Jedi deve

esquecer os seus padrões anteriores, como Yoda disse: Desaprenda, desaprenda! Os padrões normais de ensino e uso devem ser rompidos, o Padawan deve sempre procurar conhecimento vindo de grandes mestres e colocar em prática tudo de forma exemplar e para isso necessita de dedicação, muito suor, bem como Thomas Edison disse "1% inspiração 99% transpiração".

— Oh, não! — gemeu Luke. — Agora não vamos mais conseguir sair daqui!

Yoda seguira-os e bateu com o pé, furioso, ao ouvir o comentário de Luke.

— Tanta certeza assim você tem? — indagou o Mestre Jedi, em tom de censura. — Tentou já? Sempre com você não pode ser feito. Do que eu falei nada ouviu?

O pequeno rosto encarquilhado estava ainda mais contraído, numa expressão furiosa. Luke olhou para seu mestre, depois tornou a olhar, em dúvida, para a espaçonave afundada. E disse, ainda cético:

— Mestre, levantar pedras é uma coisa. Mas isso é um pouco diferente. — Yoda ficou agora realmente zangado.

— Não! Diferente não! Na sua mente estão as diferenças! Delas trate de se livrar! Para você não são mais de qualquer utilidade!.

Em qualquer novo desafio, um Padawan sempre tem de ter em

mente que não pode se limitar, algumas diferenças estão apenas em suas cabeças. Assim sendo, um bom aprendiz deve sempre aprofundar-se no negócio em que está trabalhando e tentar o máximo de abstração. Ele deve sempre saber que código é fácil de se conseguir, uma simples pesquisa no Google pode retornar o que ele deseja implementar. Já soluções e entender o negócio em que está trabalhando, isso nenhum Google pode fazer por nós. É aí o lugar onde o Padawan deve utilizar toda a sua energia e o poder de abstração; é a chave de tudo, então, nunca se limite. Saiba que sempre que encontramos diferenças ao tentar entender algo em sistemas, essa diferença está em sua mente.

Pensando assim, devemos entender que conseguimos conquistar tudo o que nossa mente puder imaginar. Para isso, devemos, além da imaginação, possuir **foco** e muita força de vontade, pois muitos desenvolvedores geralmente culpam terceiros por suas falhas ou por falta de conhecimento em alguma tecnologia. Não foram poucas as vezes em que ouvi vários colegas dizerem que estavam defasados, pois a empresa não inovava tecnologicamente, algo extremamente absurdo de se ouvir.

Nesse sentido, é interessante a observação de Chad Fowler:

"Fico muito irritado quando pergunto às pessoas se elas já viram ou usaram algumas tecnologias não tradicionais e ouço como resposta: 'não me deram a oportunidade de trabalhar com isso'. Não lhe deram oportunidade? Pra mim também não! Eu fiz a oportunidade para aprender." – Chad Fowler (*O Programador Apaixonado*, 2014)

6.4 TRABALHANDO COM SEUS MESTRES

Como dissemos anteriormente, um Padawan deve buscar um Mestre para ser sua referência e, a partir disso, começar a trabalhar com ele. Bom, agora surgiu uma dúvida: mas se já foi explicado que um Mestre pode ser alguém renomado ou (quem sabe) de outro país, então como farei para trabalhar com ele?

Primeiramente, devemos entender que a essência da ideia não é ter um Mestre nos moldes de *Star Wars* ou como em ordens herméticas, mas sim utilizar seus livros, artigos e entrevistas como base para evoluir seu conhecimento.

O interessante é que, quanto mais avança em tentar entender o trabalho de grandes mestres, mais sua visão é ampliada e passa a perceber tudo ao seu redor de forma distinta.

Por exemplo, se seguirmos Martin Fowler, que é um grande Mestre em Orientação a Objetos, a cada novo artigo ou livro que lermos sobre ele, além de evoluirmos em OO, passamos a ter uma nova leitura sobre o que já lemos dele. Se voltarmos a ler um livro dele mais uma, duas ou três vezes, perceberemos que tudo estará mais claro, sendo que muito conteúdo que não fazia qualquer sentido antes passa a fazer total diferença. Por isso é interessante não apenas ler artigos soltos, mas ter um Mestre como referência.

Agora, é claro que você pode ter um Mestre com que tenha contato, pois existem muitos bons programadores dispostos a passar seu conhecimento. Podemos perceber isso com a existência de grandes comunidades, onde muitos passam o dia todo ajudando outros programadores. Então, caso exista algum em seu trabalho, faculdade ou círculo de amizades, você pode tê-lo como Mestre.

Nesse caso, pode até aproveitar mais. Por exemplo, você pode

discutir temas com seu Mestre, pedir ajuda e também exercer algo muito importante: **ouvir**. Isso porque, com certeza, alguém com muita experiência tem muito a dizer, com grande carga de importância, sendo que esse tipo de conteúdo de borda não se consegue facilmente pela internet ou livros.

Um passo importante é, além de aprender com seu Mestre, trabalhar com ele. Para isso, perceba se ele não escreve artigos, *HowTos* etc. e se oferece ajuda, pois você poderá contribuir em algo, nem que seja inicialmente revisando. Outro ponto é perceber que o seu trabalho, às vezes, está criando um framework na empresa onde trabalha ou open source. Então, é o momento de ingressar junto a ele, pois a grande diferença entre um Padawan e um programador júnior é que o júnior vai usar eternamente frameworks criados por terceiros, já um Padawan construirá seus frameworks, assim como o Anakin Skywalker, em *A Ameaça Fantasma*, construiu seu Pod de corrida.

Os programadores Padawans devem construir seus frameworks ou quem sabe middlewares, sistemas operacionais, drivers para hardwares, novos protocolos... (bem, vou parar por aqui pois a lista é praticamente infinita do que pode ser construído), algumas que praticamente não percebemos, mas é de extrema importância para a vida pós-industrial. Porém, saiba que tudo que pode ser mentalizado, pode ser construído. Acredite!

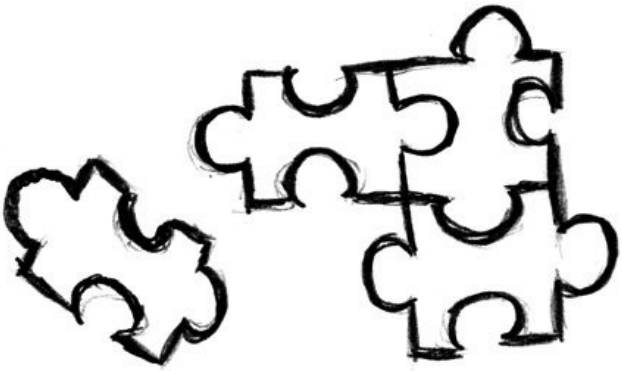
A Força está com você!

6.5 ORIENTAÇÃO A OBJETOS E GOVERNANÇA

Esta seção foi originalmente postado pelo autor, na DevMedia, em http://www.devmedia.com.br/encontre-a-governanca-correta-e-resolva-seus-problemas-futuros/26568.

No dia a dia de um arquiteto, é comum perceber a imensa redundância de código, pois grande parte dos desenvolvedores possui uma enorme dificuldade para identificar quem é responsável pelo que em seus códigos. Eles, em geral, estão preocupados em colocar seus códigos para funcionar custe o que custar, e tal irresponsabilidade "natural" custa caro.

Com base nisso, demonstraremos a partir de um cenário como identificar a governança correta. O exemplo é meramente ilustrativo, e foi utilizada uma funcionalidade simples que geralmente é colocada fora do seu devido lugar.



Cenário:

Em nosso sistema, possuímos Mensagens (`Alert` , `Dialogs`) disparadas utilizando o mesmo padrão, mudando apenas a mensagem e, possivelmente, um redirect (ou não).

Solução senso comum:

O desenvolvedor chama uma rotina que dispara a Mensagem na tela e a coloca dentro de cada formulário web, assim tendo de replicar o mesmo código a cada novo formulário.

```
ScriptManager.RegisterStartupScript(page, page.GetType(), identif
icador, "alert('Alteração realizada com sucesso');", true);
```

Entendendo um pouco de Governança e Orientação a Objetos

Quando implementamos um código, temos de nos ater ao local em que o depositamos, pois atribuímos ao seu repositório a responsabilidade por ele. Sabendo disso, pense um pouco: será que é responsabilidade de um formulário web tratar o alerta de mensagem? Não seria correto pensar que ele é apenas responsável pela chamada de alerta?

Então é correto definir como governança de alertas a criação de uma classe chamada `alerta` , e nela encapsular tudo que se trata

de alertas, tratando suas particularidades e frameworks usados como jQuery UI, EXJs etc.

Sempre reflita se o que você está fazendo é responsável por aquilo que faz. Sabemos que é comum construirmos classes com métodos alienígenas a ela e, pior, métodos que fazem muito mais do que foi definido. Por exemplo, um método chamado `MisturarCafeComLeite` pode estar colocando apenas café e misturando com nada – ou seja, fazendo menos do que foi definido –, ou às vezes, além de misturar café com leite, coloca também açúcar, o que seria um erro crasso, pois estaria indo além de sua responsabilidade.

Além disso, podemos verificar que o princípio SOLID foi desrespeitado. SOLID é a abreviação dos cinco primeiros princípios básicos da programação orientada a objetos, que foi introduzida por Michael Feathers, na primeira década de 2000.

Alguns escritores como Kent adoram atribuir cheiro ao software, e esse cheiro basicamente é medido por meio do respeito a esse princípio, cujo significado de cada uma das letras é:

- ***S**ingle Responsibility:* um objeto deve possuir apenas uma responsabilidade.
- ***O**pen-Close:* entidades devem ser abertas para extensão, mas fechadas para alteração.
- ***L**iskov substitution:* objetos de cliente muito específicos devem ser substituíveis por instâncias de seus subtipos.
- ***I**nterface segregation:* pequenas e específicas interfaces são melhores que uma interface genérica.
- ***D**ependency inversion:* não dependa de implementações concretas, mas sim de abstrações.

Problema da implementação

A solução apresentada pelo programador comum, que chamaremos de John Doe, como vimos não respeita o princípio SOLID. Uma manutenção ou alteração no sistema deixaria John e sua equipe em apuros. Pior ainda, John Doe, por algum motivo, desliga-se da empresa. Em sua solução, ele atribuiu a responsabilidade sobre o alerta a cada formulário, e não a um repositório de alertas, o que deixou o alerta fora de governança gerando várias dificuldades técnicas e duplicação desnecessária de código.

Imagine se tivéssemos de ir formulário a formulário, corrigindo um erro na mensagem de alerta ou, então, alterá-la? Seria um trabalho árduo, além de ser um multiplicador de erro, pois a possibilidade de errar é multiplicada pela quantidade de vezes que o código se repete.

Solução

A solução para o problema é a criação de uma classe a que será delegada a responsabilidade de disponibilizar as funcionalidades, que podem ser usadas de forma genérica pelos formulários ou, caso precise, podemos criar uma classe específica para os `Alert` s. Entretanto, isso depende do caso.

No nosso caso, por não termos muitas funcionalidades de alerta, e sendo nossa intenção apenas desacoplar do formulário funcionalidades que podem ser usadas de forma genérica, então podemos criar uma classe de funcionalidades do formulário, e nela colocar o nosso código.

```
public static class FormularioFuncionalidades
{
    public static void Alert(Page page, string mensagem, string i
dentificador)
    {
        string strScript = "alert('" + mensagem + "')";

        ScriptManager.RegisterStartupScript(page, page.GetType(),
 identificador, strScript, true);
    }
}
```

Em nossos formulários web, chamaríamos o `alert` dessa maneira:

```
FormularioFuncionalidades.Alert(page, "Atualização efetuada com s
ucesso.", href, "OK2");
```

Vantagens

Respeitamos o princípio SOLID, e nossa classe e método possuem apenas uma responsabilidade, prontos para extensão. Pensando ainda na solução, imagine que agora queremos implementar um redirect e que a mensagem seja disparada utilizando jQuery UI. Nesse caso, seria fácil fazê-la com a governança correta. Vejam que, ao fazermos as alterações sugeridas, todos os formulários passaram a usufruir das novas funcionalidades.

```
public static void Alert(Page page, string mensagem, string href,
 string identificador)
{
    page.Controls.Add(PreparaMensagem(mensagem));
    ScriptManager.RegisterStartupScript(page, page.GetType(), ide
ntificador, PreparaScript(href), true);
}

private static string PreparaScript(string href)
{
```

```
string strScript = string.Format("$(document).ready(function () {
    + "$('#dialog').dialog({"
    + "               modal:true,"
    + "                 width: 400,"
    + "                show: 'drop',"
    + "            hide: 'drop',"
    + "             resizable: false,"
    + "                    buttons: {"
    + "            Ok: function () {"
    + "    $(this).dialog('close');"
    + "                    }"
    + "         }"
    + "}).bind('dialogclose', "
    + "      function(event, ui) "
    + "               { {0}"
    + "               }"
    + ");;"
    + "});", (string.IsNullOrEmpty("")) ? "" : "window.location.h
ref = '" + href + "'; ");
    return strScript;
}

private static Literal PreparaMensagem(string mensagem)
{
    return new Literal() { Text = String.Format("<div id="dialog"
 title="Titulo"><p>{0}</p></div>", mensagem) };
}
```

Conclusão

Respeitar os princípios SOLID nos leva a construir códigos mais elegantes e de fácil manutenção. Podemos ver que, ao encapsular a chamada de mensagem, além de podermos facilmente corrigir um erro que esteja nesse código, isso também nos facilita sua extensão e modificação, caso precisemos.

"Qualquer tolo consegue escrever código que um computador entenda. Bons programadores escrevem código que humanos possam entender." – Martin Fowler (*Refatoração*, 2008)

6.6 ROTINA DE TREINO

Procure a cada 3 meses conhecer um framework novo e aprofundar-se nele. Analise antes se existem outros que fazem a mesma função e, caso exista, antes de iniciar os estudos, veja os seus diferenciais e procure nas comunidades os comparativos de usuários mais experientes. Após feito isso, aprofunde-se nele.

No período de 12 meses, aprofunde-se em uma nova linguagem. Nesse tempo, tente aprender o máximo dela, e crie projetos pessoais utilizando apenas ela.

Plataformas

Existem algumas plataformas de ensino pagas e gratuitas. É importante buscar em seu clã as melhores.

- Plural Sight – http://www.pluralsight.com
- Code School – http://www.codeschool.com
- Alura – http://www.alura.com.br
- Codecamedy – https://www.codecademy.com/pt

Exercite a criação de domínios

- Acompanhe site do Martin Folwer, em http://www.martinfowler.com. Lá existem muitas informações sobre domínio.
- Procure artigos e livros de Eric Evans. Acompanhe o site em http://dddcommunity.org/tag/eric-evans/.
- Existe um ótimo desenvolvedor brasileiro que escreve muitas coisas sobre DDD, Daniel Cuckier. Em seu site, há uma ampla literatura e vídeos sobre o tema, em http://www.agileandart.com.

Participe de eventos

Participe, principalmente, dos que se referem ao seu Clã.

Citarei alguns grandes eventos, mas pesquise e fale com a comunidade para indicar outros tão bons quanto.

- TDC (The Developers Conference) – http://www.thedevelopersconference.com.br/
- Summit Office 365 – http://summit.office.com/events
- Visual Studio Summit –

http://www.visualstudiosummit.com.br

- Java One – https://www.oracle.com/javaone/br/register/index.html
- QCon – http://qconsp.com/

Por Carlos Bueno

Capítulo 7

CAVALEIRO JEDI

"Qualquer um pode zangar-se - isso é fácil. Mas zangar-se com a pessoa certa, na medida certa, na hora certa, pelo motivo certo e da maneira certa - não é fácil." – Aristóteles

7.1 VALORES DE UM CAVALEIRO

Um desenvolvedor que entra na posição de Jedi deve entender o quão importante é saber se posicionar sobre tudo à sua volta. Nesse momento, deve treinar mais e mais sua capacidade de empatia, como também deve evoluir na capacidade de compreensão de negócio, e tomar muito cuidado com os clientes com que aceitou trabalhar. Isso é o que direcionará seu futuro como desenvolvedor, pois, sem perceber, cada vez mais estará dentro de sistemas de mesmo domínio, logo, escolha trabalhar em domínios que sejam promissores.

Exemplo real

Um certo dia, eu estava trabalhando em uma equipe na empresa Adiantados. Esta era formada pelos profissionais rejeitados pela equipe de experts da empresa. Lembrem-se de que experts nem sempre são Jedis, o conceito é distinto: um expert possui limitações e, às vezes, trilha por caminhos que o levam para o lado negro da Força.

Eles não eram rejeitados por falta de capacidade técnica, mas devido à existência de um forte corporativismo, que nasceu com o crescimento da empresa. Inicialmente, a empresa possuía alguns técnicos que, por grande desempenho, fizeram-na crescer muito, gerando novas contratações. Esse grupo inicial se fechou, e trabalhavam entre eles. O que ocorria era que a empresa perdia muito em negócio, ou seja, tudo o que eles conheciam sobre o dia a dia dos clientes não era disseminado por toda a empresa, acarretando na criação de novos sistemas sem qualidade, por falta de conhecimento de negócio dos novos funcionários.

Cada dia mais esse tipo de atitude minava a empresa. Então, comecei a agir tentando integrar componentes dos grupos distintos, o que, inicialmente, trazia muita disputa. Porém, um Desenvolvedor Jedi deve, nesse momento, entender a situação e fazer a coisa certa. Deve trabalhar fortemente sua inteligência emocional e ter um grande autocontrole, pois o sucesso não está exclusivamente no seu conhecimento técnico, mas na capacidade de saber lidar com a situação.

7.2 MANTENHA SEU SABRE DE LUZ LIMPO

Um Cavaleiro sabe que seu código deve estar sempre limpo, então, ele precisa estar atento e seguir uma regra de escoteiro: *deixe a área do acampamento mais limpa do que como você a encontrou*. Pois, caso mantenha o contrário, a tendência do código é a degeneração, assim como uma área transformada em um aterro sanitário.

Alguns programadores vão confrontar muitas vezes um Jedi. Esses são os programadores que trilharam o lado negro da força, e que querem a qualquer custo manter péssimos códigos. Eles encontrarão várias desculpas para justificar suas atitudes, mas tenha em mente que a única constante em um ciclo de vida de um software é:

ELE VAI MUDAR !

Sabendo disso, não podemos cair em tentação e trilhar o lado negro, devemos ter foco e desembainhar os sabres de luz a qualquer ameaça hostil que tente nos guiar ao contrário, pois o final disso é a entropia.

O bom Cavaleiro manterá seu código limpo toda vez que sentir um mau cheiro.

MAU CHEIRO?

Isso mesmo. Mau cheiro. Assim como Uncle Bob definiu em *Código limpo*, o código possui cheiro, e isso é um dos motivos pelos quais o desenvolvimento está mais para a arte do que para as ciências Exatas, uma vez que não é qualquer desenvolvedor que pode sentir o mau cheiro.

Isso depende de algumas técnicas que não podem ser facilmente ensinadas, como Uncle Bob diz: *"Livros sobre arte não prometem lhe tornar artistas. Tudo o que ele pode é lhe oferecer algumas das ferramentas, técnicas e linhas de pensamento que outros artistas usaram"*.

Vamos trabalhar alguns conceitos para manter seu código mais limpo. Trabalharemos com a linguagem C# do clã dotNot, e apresentaremos apenas os conceitos que julgamos de extrema importância para o domínio de um Cavaleiro. Entretanto, seria interessante nos aprofundarmos em livros, como: *Código limpo*, de Robert Martin, e *Refatoração*, de Martin Fowler.

Limpando seu código

Geralmente, ouvimos algumas discussões entre desenvolvedores sobre comentários, e alguns brigando devido ao fato de o código não possuir comentários. Acontece que existem dois problemas nesse caso. O primeiro é que realmente o código foi mal escrito, pois, se há necessidade de comentários, é porque o desenvolvedor não conseguiu deixar clara sua intenção ao escrevê-lo, algo que é muito comum. Para mudar isso, além de treinamento, é necessária muita capacidade artística.

Outro problema é em relação ao desenvolvedor que está solicitando uma inserção de comentários no código. Esse também não consegue perceber que o código não está escrito a ponto de ser claro e mostrar sua intenção. A partir do exemplo a seguir, vamos mostrar alguns problemas e soluções para entendermos melhor a necessidade de um código limpo, e tambem por que não é necessário comentar um código:

```
//Busca todos Status de Cliente
public string Status(string ID_CLI) {
    CLI entidade = _repositorio.BuscarCli(ID_CLI_SIST);
    if(entidade != null)
    {
        string status = string.Empty;
        for (int i = 0; i < entidade.st_cli.lenght; i++)
        {
            status += entidade.st_cli[i];
        }
        return entidade.;
    }

    return string.Empty;
}
```

Ao ler esse código, ficaremos um tanto quanto perdidos. Por exemplo, o que é `ID_CLI` ? Esse tipo de escrita é conhecida como

Notação Húngara, e foi criada por Charles Simonui. Ela leva esse nome pois os primeiros a terem contato com ela falaram que era tão estranha que se assemelhava ao húngaro.

Ela é muito usada por profissionais de banco de dados, então geralmente teremos contato com algo do tipo quando estivermos trabalhando com bancos de sistemas legado, ou empresas que não atualizaram seus profissionais e procedimentos. O grande problema desse tipo de escrita é que não nos deixa claro o verdadeiro nome, podendo nos levar a grandes equívocos. Para evitar isso e tornar seu código claro, devemos evitar tais nomes, utilizando em seu lugar nomes significativos.

No nosso exemplo, `ID_CLI` poderia ser substituído por `idCliente` e `st_cli` por status, assim saberíamos do que se trata.

```
//Busca todos Status de Cliente
public string Status(string idCliente) {
    CLI entidade = _repositorio.BuscarCli(idCliente);
    if(entidade != null)
    {
        string status = string.Empty;
        for (int i = 0; i < entidade.status.lenght; i++)
        {
            status += entidade.status[i];
        }
        return entidade.;
    }

    return string.Empty;
}
```

Outro ponto a que devemos nos ater é com o mapeamento mental. Geralmente, usamos variáveis de uma só letra, como `i`, `j` etc. Esses tipos de variáveis são um grande perigo, pois, em um codigo grande, podemos utilizá-las repetidas vezes sem perceber,

gerando grandes erros. Logo, poderíamos alterar em nosso código a variável `i` por `contadorStatus`, por exemplo. Ficaria assim:

```
//Busca todos Status de Cliente
public string Status(string idCliente) {
    CLI entidade = _repositorio.BuscarCli(idCliente);
    if(entidade != null)
    {
        string status = string.Empty;
        for (int contadorStatus = 0; contadorStatus < entidade.st
atus.lenght; contadorStatus++)
        {
            status += entidade.status[contadorStatus];
        }
        return entidade.;
    }

    return string.Empty;
}
```

Além disso, nomes devem ser significativos, e devem também estar inseridos no domínio da aplicação. Se o Jedi conseguir trabalhar a linguagem ubíqua – ou seja, uma linguagem comum entre usuário e desenvolvedores, que expresse a ação da aplicação –, quando qualquer desenvolvedor estiver em contato com um usuário e conversar sobre o sistema, eles estarão falando no mesmo nível de entendimento. Assim, erros comuns de entendimento, como já citados no livro, não vão ocorrer.

Em nosso caso, temos a palavra `_CLI_` sem muito sentido, fazendo mais sentido ser chamada de `Cliente`. Outra palavra que deve ser substituída, seguindo o mesmo pensamento, é `BuscarCli` para `BuscarClientePorId`.

Além disso, temos `buscarStatus` e, acima do método, um comentário já que o nome não é significativo o suficiente. Poderíamos substituir por `buscarStatusClienteConcatenado`,

assim, qualquer pessoa que o lesse, mesmo não sendo um programador, seria capaz de entender, e poderíamos remover o comentário.

```
public string buscarStatusClienteConcatenado(string idCliente) {
    Cliente entidade = _repositorio.BuscarClientePorId(idCliente)
;
    if(entidade != null)
    {
        string status = string.Empty;
        for (int contadorStatus = 0; contadorStatus < entidade.st
atus.lenght; contadorStatus++)
        {
            status += entidade.status[contadorStatus];
        }
        return entidade.;
    }

    return string.Empty;
}
```

Refatorando seu código

A intenção do livro não é ser um guia de refatoração e nem de boas práticas de Orientação a Objetos. O que queremos demonstrar com os exemplos é a diferença entre códigos quando se aplicam os conceitos de limpeza de código e refatoração. Então, caso queira se aprofundar, sugiro que leia os grandes Mestres Jedis que foram citados no início da sessão *Mantenha seu sabre de luz limpo*.

Continuando com o mesmo código, ainda existem pontos que não estão claros. Por exemplo, em um momento, ele faz algumas verificações e retorna os status, logo, poderíamos facilmente encapsular esse algoritmo, dando um nome significativo, e colocá-lo em outro método. Essa técnica de refatoração chama-se **extrair método**

A seguir, vamos executá-la, batizando o algoritmo com o nome `retornarStatusConcatenadoCliente`.

```
public string buscarStatusClienteConcatenado(string idCliente) {
    Cliente entidade = _repositorio.BuscarClientePorId(idCliente)
;
    if(entidade != null)
        concatenarStatusCliente(entidade);

    return string.Empty;
}

public string concatenarStatusCliente(Cliente entidade){
    string status = string.Empty;
    for (int contadorStatus = 0; contadorStatus < entidade.status
.lenght; contadorStatus++)
    {
        status += entidade.status[contadorStatus];
    }
    return status;
}
```

Agora, nosso código está praticamente legível, sem nenhuma necessidade de comentários. Vamos fazer outra refatoração para não ser necessário instanciarmos uma propriedade para `Cliente` e remover uma linha de código. Removeremos a linha onde é criada e passaremos o retorno do método `BuscarCliente diretor` no método `retornarStatusConcatenadoCliente`.

```
public string buscarStatusClienteConcatenado(string idCliente) {
    if(entidade != null)
        concatenarStatusCliente(_repositorio.BuscarClientePorId(i
dCliente));

    return string.Empty;
}

public string concatenarStatusCliente(Cliente entidade){
    string status = string.Empty;
    for (int contadorStatus = 0; contadorStatus < entidade.status
.lenght; contadorStatus++)
```

```
    {
        status += entidade.status[contadorStatus];
    }
    return status;
}
```

Após limpar nosso método e refatorá-lo, é hora de comparar e ver a diferença entre os dois. Vamos apenas avaliar o método original, e removeremos na comparação o método `concatenarStatusCliente` , pois se trata de outro método. Com isso, vamos perceber a facilidade de entender o método sem a necessidade de comentários. Isso acontece uma vez que o código é autoexplicativo, e o anterior não.

Por incrível que pareça, o anterior é o que é normalmente feito por um programador comum, mas não por um Jedi. Lembre de que ser Jedi não é apenas ter as características intelectuais, mas também as emocionais, como vontade. Ser um Jedi requer ser extremamente disciplinado.

Em nosso código anterior, temos:

```
//Busca todos Status de Cliente
public string Status(string ID_CLI) {
    CLI entidade = _repositorio.BuscarCli(ID_CLI);
    if(entidade != null)
    {
        string status = string.Empty;
        for (int i = 0; i < entidade.st_cli.lenght; i++)
        {
            status += entidade.st_cli[i];
        }
        return entidade.;
    }

    return string.Empty;
}
```

Vamos comparar com o código final:

```
public string buscarStatusClienteConcatenado(string idCliente) {
    if(entidade != null)
        concatenarStatusCliente(_repositorio.BuscarClientePorId(i
dCliente));

    return string.Empty;
}
```

No primeiro, se removermos o comentário, praticamente não entederemos nada se não ficarmos avaliando o código. Essa avaliação passa por um processo que atrapalha a capacidade de desenvolver, pois nossos olhos olham para cima e para baixo, indo e voltando, para avaliar o que foi feito. Tal ato pode nos levar a erros de entendimento, consequentemente, acarretando erros ao codificar alterações.

O correto é termos um código cuja leitura seja fluida. Para isso, necessitamos que métodos e propriedades possuam nomes pronunciáveis e significativos. Não usaremos Notação Hungára, pois assim conseguiremos um código de fácil leitura e entendimento, o que facilitará muito a entrada de novos programadores em um projeto em andamento.

7.3 DESVENDANDO PADRÕES

Em meados da década de 80, alguns estudantes de ciência da computação criaram os primeiros **padrões de projetos** para área. Mas, o que é um **padrão de projeto**?

Padrão de projeto é uma solução para um problema recorrente que pode ser reutilizada. Entretanto, não é um código propriamente dito, mas sim um modelo a ser usado quando nos deparamos com o problema, sempre o adaptando ao cenário em que nos encontramos.

Em 1995, os criadores dos padrões de projetos, conhecidos como *Gang of Four*, lançaram o livro *Design Pattens: elements of reusable object-oriented software*, contendo 23 padrões que eram divididos em 3 famílias: padrões de criação, estruturais e comportamentais.

Os padrões por eles apresentados são:

- **Criação:**
    - Abstract Factory
    - Builder
    - Factory Method
    - Prototype
    - Singleton
- **Estruturais:**
    - Adapter
    - Bridge
    - Composite
    - Decorator
    - Facade
    - Flyweight
    - Proxy
- **Comportamentais:**
    - Chain of Responsibility
    - Command
    - Iterpreter
    - Mediator
    - Memento

- Observer
- State
- Strategy
- Template Method
- Visitor

Esses padrões devem ser aprendidos por todos os Cavaleiros e, além de aprendidos, devem ser também praticados e difundidos em sua equipe. Um Jedi tem a responsabilidade em disseminar o conhecimento necessário para que se atinja excelência em códigos, pois ele é o maior defensor de quem realmente será prejudicado ao utilizar sistemas mal feitos.

Assim, tenha em mente que existe o programador que trabalha por dinheiro e os Jedis que trabalham por amor. Porém, ambos devem, ao mínimo, respeito a quem paga, pois em ambos os casos – dinheiro ou amor –, o comum é a fonte pagadora que, independente de sua escolha em ser Jedi ou não, deve respeito a quem utilizará o sistema.

Implementando o padrão Facade

Assim como na sessão *Mantenha seu sabre de luz limpo*, aqui vamos mostrar a aplicação de um padrão em um código. Em nosso caso, será uma aplicação de caixa eletrônico com a função de saque.

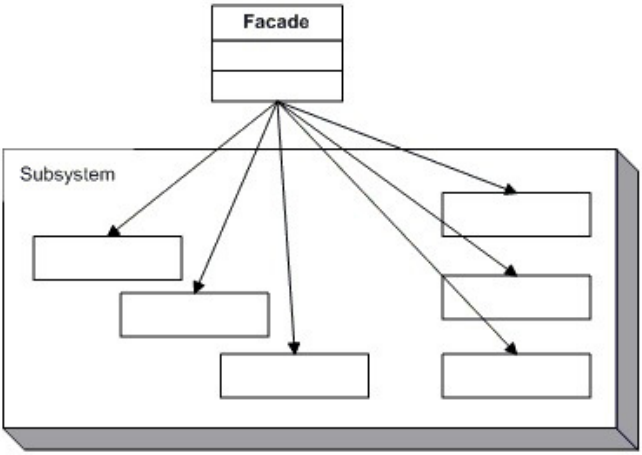


Figura 7.2: Facade. Fonte: DoFactory.com)

```
public class CaixaEletronico
    public string Saque(string contaCorrrente,string senha, doubl
 valorSaque) {
        RepositorioCliente _repositorioUsuario = new RepositorioC
liente();

        Usuario usuario = _repositorioUsuario.buscaClientePorSenh
a(contaCorrente,senha);
        if(usuario != null)
        {
            RepositorioContaCorrente contaCorrente = new Reposito
rioContaCorrente();
            if(contacorrente.buscarSaldo(contaCorrente) => valorS
aque)
            {
                return "saque autorizado";
            }
            else
            {
                return "saldo insuficiente";
            }
        }else{
            return "Senha invalida";
        }
```

```
    }

    public double ConsultarSaldo(){
        RepositorioCliente _repositorioUsuario = new RepositorioC
liente();
        _repositorioUsuario.buscaClientePorSenha(senha);
         if(usuario != null)
        {
            RepositorioContaCorrente contaCorrente = new Reposito
rioContaCorrente();
            return contacorrente.buscarSaldo(usuario) ;
        }else{
            return "Senha invalida";
        }

    }
}
```

Percebemos facilmente a repetição de alguns códigos, como a busca de `Saldo` , além de o algoritmo estar fora de governança.

Então, vamos aplicar o padrão estrutural *Facade*, no qual criaremos uma interface de nível mais elevado, encapsulando as funcionalidades dentro da classe que realmente tem responsabilidade sobre o algoritmo. Em nosso caso, a classe será correntista, pois quem deve saber se a senha é valida, o saldo ou efetuar o saque é a classe `Correntista` , e não a `CaixaEletronico` . Como foi criada inicialmente, ela desrespeitava o conceito SOLID, pois `CaixaEletronico` assumia mais de uma responsabilidade.

A classe `Correntista` será nossa classe que encapsulará os métodos referentes aos correntistas, até porque nem sempre o caixa eletrônico vai solicitar saldos, saques etc. Por exemplo, podemos ter um caixa fazendo essas funcões, um aplicativo de celular ou um website.

No modelo original, teríamos de repetir os mesmos códigos em cada uma das implementações, então, imagine quando uma alteração fosse necessária? Teríamos de sair procurando todos os lugares que executam funções refentes ao `Correntista` para alterar, sendo que faria mais sentido ir apenas em `Correntista` e efetuar as alterações.

Então, vamos às mudanças, criando primeiramente a classe `Correntista`:

```
public class Correntista

    //Obs: Trabalhar com Injeção de dependencia
    private RepositorioCliente _repositorioUsuario;
    RepositorioContaCorrente _contaCorrente;

    private string _contaCorrente = string.Empty;
    private string _senha = string.Empty;

    public Correntista(string contaCorrente, string senha){
        _contaCorrente = contaCorrente;
        _senha = senha;
    }

    private bool validarSenha(){
      if  (_repositorioUsuario.buscaClientePorSenha(contaCorrente
,senha) != null)
        return true;
      else
        trow new exception("Senha Inválida");

      return false;
    }

    public double ConsultarSaldo(){
        if(validarSenha())
            return _contaCorrente.buscarSaldo(contaCorrente);
    }

    public string EfetuarSaque(double valorSaque) {
        if(VerificaSaldo() >= valorSaque)
```

```
            return "Saque Autorizado!"
        else
            return "Saldo Insuficiente!"

    }
}
```

Agora que criamos a classe utilizando o padrão Facade, e encapsulamos toda a lógica da classe final, vamos implementar na classe `CaixaEletronico` e ver como ficou:

```
public class CaixaEletronico
    private Correntista _correntista;

    public string Saque(string contaCorrrente,string senha, doubl
 valorSaque) {
        _correntista = new Correntista(contaCorrente,senha);
        try{
           return _correntista.saque(valorSaque);
        }catch(ex exception){
            return ex.message;
        }

    }

    public double ConsultarSaldo(string contaCorrrente,string sen
ha){
       _correntista = new Correntista(contaCorrente,senha);
        try{
            return _correntista.ConsultarSaldo();
        }catch(ex exception){
            return ex.message;
        }

    }
}
```

Quando utilizar?

- Quando precisarmos encapsular regras de negócios que usam outras classes.

É um padrão extremamente usado, pois, na maior parte do tempo, construímos métodos que interagem com outras classes.

Por que utilizar?

- Para não expor métodos ou classes que não devam ser expostos.
- Para não duplicar código.

Implementando o padrão Strategy

É um padrão comportamental que permite encapsular cada algoritmo em classes, assim, podemos aplicar em cada classe suas estratégias, ou seja, o algoritmo que necessita da forma que precisar.

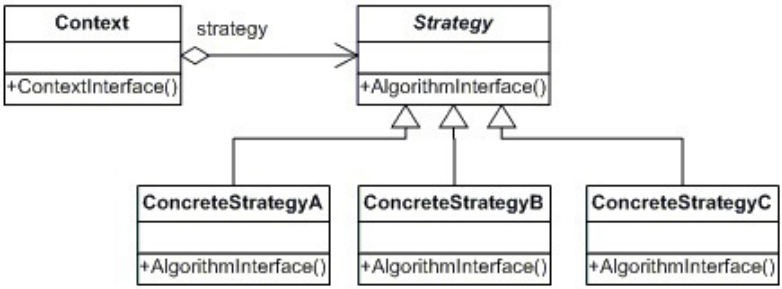


Figura 7.3: Strategy. Fonte: DoFactory.com

Temos vários cenários para a sua utilização. Imagine uma aplicação de solicitações que, por exemplo, tenha solicitações de reserva de salas, de serviços de manutenção etc. Em cada um dos casos, eles não deixam de ser solicitações, pois ambos possuem as mesmas ações; entretanto, as ações e as estratégias se portam de forma distinta. Logo, nesse caso, esse padrão se enquadra perfeitamente.

Hands On!

Precisamos criar uma classe que calcule 2 tipos distintos de impostos e pensar na possibilidade de, no futuro, outros impostos surgirem. Com isso, devemos criar de forma que respeite o conceito Open/Close de OO.

Para isso, usaremos o padrão Strategy, criando classes especialistas para calcular cada imposto, chamadas `CalculaImpostoA` e `CalculaImpostoB` . Além delas, também uma classe `Imposto` , que retornará o cálculo do imposto desejado.

Primeiramente, devemos criar um contrato (interface), para ser

assinado entre as classes que calcularão o imposto.

A seguir, veja a interface:

```
public interface ICalculaImposto
{
    string CalcularImposto();
}
```

Agora, com o contrato, devemos criar as classes para os cálculos dos impostos, implementar a interface `ICalculaImposto` e, em cada uma, implementar o algoritmo para cálculo específico do imposto.

```
public class CalcularImpostoA : ICalculaImposto
{
    public string CalcularImposto(){
        return "Imposto A Calculado";
    }
}
```

A classe `CalculaImpostoB`:

```
public class CalcularImpostoB : ICalculaImposto
{
    public string CalcularImposto(){
        return "Imposto B Calculado";
    }
}
```

Agora, vamos criar a classe `Imposto`, que encapsulará os algoritmos das classes e calculará o imposto conforme o seu tipo que for passado em seu construtor ao instanciar.

```
public class Imposto
{
    private ICalculaImposto _calculaImposto;

    public Imposto( ICalcularImposto calculaImposto){
        _calculaImposto = calculaImposto;
    }
```

```
    public string CalcularImposto()
    {
        return _caculaImposto.CalcularImposto();
    }
}
```

Agora que tudo está pronto, podemos ver como funciona nossa `Strategy`. Usaremos um *Console Application* no Visual Studio para criar uma simples aplicação que vai calcular o imposto desejado e exibir o resultado na tela.

No exemplo a seguir, a estratégia utilizada foi o cálculo do Imposto A. Poderíamos, logo a seguir, pedir para calcular o Imposto B ou outros impostos que houvessem sido criados.

```
class Program
{
    static void Main(string[] args)
    {
        Imposto imposto = new Imposto(new CalcularImpostoA());
        Console.WriteLine imposto.CalcularImposto();
        Console.ReadKeys();
    }
}
```

A seguir, veja um exemplo utilizando o cálculo do Imposto B:

```
class Program
{
    static void Main(string[] args)
    {
        Imposto imposto = new Imposto(new CalcularImpostoB());
        Console.WriteLine imposto.CalcularImposto();
        Console.ReadKeys();
    }
}
```

QUANDO UTILIZAR?

- Quando precisamos encapsular regras de negócios que usam outras classes.

É um padrão extremamente usado, pois na maior parte do tempo construímos métodos que interagem com outras classes.

Por que utilizar?

- Para não expor métodos ou classes que não devam ser expostos.
- Para não duplicar código.

7.4 CAMADA DE APRESENTAÇÃO E REGRAS DE NEGÓCIO

Até aqui, aprendemos a importância de:

- Orientação a Objetos;
- Domínio de uma aplicação;
- Manter um código limpo;
- Refatorar código.

Agora trataremos um pouco sobre a camada de apresentação. Vamos trabalhar com um ponto crucial da camada que é a validação, muito confundido com regras de negócio. Entenderemos a diferença entre os dois e os impactos causados em

um eventual erro.

> Originalmente publicado pelo autor em http://goo.gl/hHaZcF.

Por que erramos?

Aplicações são construídas de forma estanque, ou seja, cada um constrói um pedaço sem conhecer o todo. Esse tipo de desenvolvimento de aplicações parte de conceitos muito antigos, mas que são utilizados até hoje pela grande massa. Esse tipo de pensamento é o principal responsável por aplicações com problemas em seu coração – quando falamos em coração, nos referimos ao **modelo de domínio**.

> **Dica**
>
> Nenhum modelo pode ser construído ou implementado com um microconhecimento do sistema, pois afeta diretamente o *core* da aplicação.

Imaginemos que um desenvolvedor recebe a tarefa de implementar uma parte do sistema que gerencia fornecedores. Se ele não souber de outras regras do sistema que envolva fornecedores, como vendas, compras, logística, entre outras, o sistema pode estar comprometido.

Isso ocorre porque ele não tem conhecimento do domínio que envolve fornecedores, mas sim apenas a funcionalidade pela qual ele está responsável no momento. Para se resolver isso, devemos aplicar metodologias de desenvolvimento ágil. Por exemplo, DDD prega a participação de toda equipe a cada levantamento, e que o modelo seja construído por todos em conjunto.

Assim ninguém constrói algo sem conhecer o **todo**, o que diminui bastante um futuro erro na regra por falta de conhecimento do projeto.

Vamos continuar errando?

O assunto abordado até o momento não possui nada de revolucionário, mas a pergunta que devemos responder é: *Sabendo disso, por que continuamos fazendo errado?*

O maior problema é que dificilmente alguém gosta de sair da posição de conforto para aplicar algo que demanda um esforço extra e um retorno desconhecido. Seguindo esse pensamento, chegamos à conclusão de que a zona de conforto é um *brownfield*.

Figura 7.4: Brownfield

Viver em uma zona de conforto dessas não é nenhum pouco interessante, então devemos mudar para conseguirmos conquistar algo melhor. Podemos pensar nisso como a Teoria do Caronista. Resumidamente, temos duas opções: continuar no Brownfield e viver sempre com todos os problemas conhecidos no desenvolvimento; ou buscar mudanças e ir para o *GreenField*.

Nesse nosso caso, mudar é uma aposta que, na pior das hipóteses, nos manterá inertes. Porém, se conseguirmos, teremos bons resultados. Não seria, então, a hora de arregaçar as mangas e mudar?

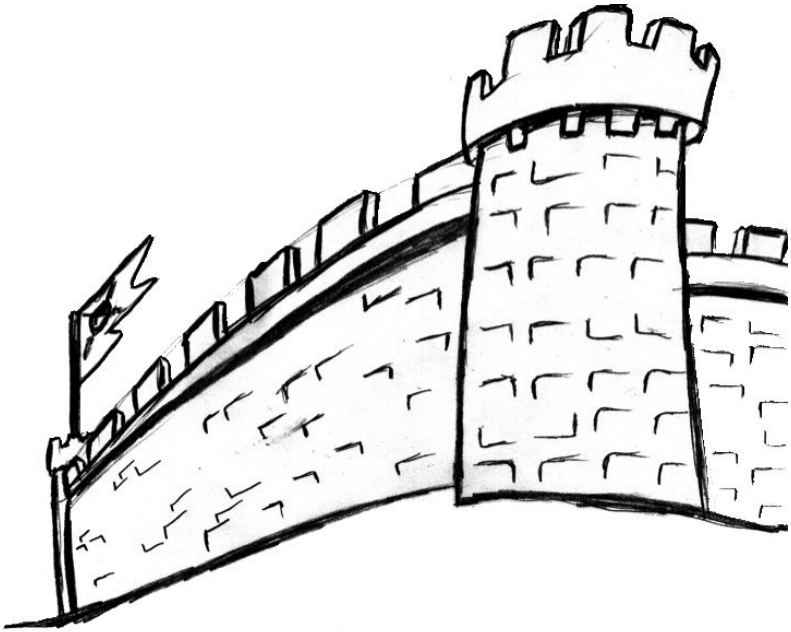


Figura 7.5: Greenfield, uma paisagem dessas é um prazer olhar

Indo ao ponto: como identificar um vazamento?

Vira e mexe nos deparamos com desenvolvedores discutindo padrões a serem adotados na validação da regra de negócio, sendo que o grande X da questão é: *o que deve ser validado na regra de negócio, e o que é feito na camada de apresentação?*

Existem algumas linhas de raciocínio, por isso, não tome esta que seguiremos como verdade absoluta. Essa questão tem um certo ar subjetivo, então cada um vai defender a forma a que mais se adapta.

Assim, seguiremos pela corrente dos precavidos, ou seja, toda regra sem exceção deve estar presente na camada de regra, por

mais simples que seja a validação. Mas também devemos fazer algumas das validações no lado do cliente. Seria um *double check*, a checagem no cliente o ajuda a garantir uma boa usabilidade do sistema, e a checagem no servidor seria a "Final Frontier" com o serviço que estamos disponibilizando ao usuário, seja ele uma simples *entity service* (repositório) ou não.

Martin Fowler, em *Padrões de arquitetura de aplicações corporativas*, prega que: *"Normalmente, as pessoas escreviam essa lógica no cliente, mas isso era desajeitado e, normalmente, feito embutindo-se a lógica diretamente nas telas da interface com o usuário. À medida que a lógica se tornava mais complexa, ficava muito difícil trabalhar com este código. Além disso, embutir lógica nas telas facilitava a duplicação de código, o que significava que alterações simples resultavam em buscas de código semelhante em muitas telas"*.

Fica nítido, seguindo o raciocínio de Fowler, que colocar regra na UI (*User Interface*) causa duplicação de código, gerando um grande problema nas futuras alterações, pois teremos dois ou mais lugares para alterar a mesma coisa.

Veja um exemplo de validação que vazou:

```
try
{
    Pessoa pessoa = new Pessoa() { Nome = "Carlos Bueno", Status
= 1 };
    if (pessoa.Nome.Length > 30) // ---> essa validacao nao dever
ia estar na Apresentacao e sim na Regra de negocio
                throw new Exception("Usuario nao pode possuir mais
 de 30 carateres");

    DominioPessoa.SalvarPessoa(Pessoa);
}
catch (Exception ex)
```

```
{
    Response.Write(ex.Message)
}
`
```

O código dessa listagem é um exemplo de vazamento, pois a validação está sendo feita dentro da apresentação. Imagine se tivermos de fazer outro sistema que grave os dados de pessoas, não teríamos de verificar o tamanho do nome novamente? Essa validação deveria estar dentro da classe responsável por salvar pessoas.

Agora, veja a seguir um exemplo em que não houve vazamento:

```
try
{
    Pessoa pessoa = new Pessoa() { Nome = "Carlos Bueno", Status
= 1 };
    DominioPessoa.SalvarPessoa(Pessoa);
}
catch (Exception ex)
{
    Response.Write(ex.Message)
}
`
```

Nessa listagem, podemos ver nitidamente que não houve vazamento da regra, pois nenhuma validação foi necessária na camada de apresentação, não gerando repetição de código.

Figura 7.6: Todo vazamento traz grandes impactos. Cuide sempre para que ele não ocorra.

Um sistema com vazamento de regra pode possuir inúmeras falhas de segurança, além de, em algum momento, poder gerar erros com danos incalculáveis.

Seguindo o pensamento de Martin, podemos visualizar que, ao criamos uma UI que use a camada de regra, quando viermos a utilizar outra interface de acesso à regra (*timer job*, serviço, uma outra UI), teremos de repetir algum código. Pode ter certeza de que seu código da regra de negócio vazou para a interface de usuário, pois essa duplicação cheira bem mal.

Ainda no livro de Martin Fowler, ele diz para imaginarmos além da UI que criamos, uma outra, por exemplo, uma aplicação prompt que acesse nossas regras. Esta, caso tenha *"que duplicar*

alguma funcionalidade, é um sinal de que a lógica de domínio vazou para apresentação".

Validações Cliente em aplicações web

Falando de aplicações web, a checagem de clientes seria uma validação que não envolve acesso ao servidor, como verificar se o usuário está digitando um número inteiro, fazer as devidas formatações das entradas de dados, coisas desse tipo. Um acesso ao banco com cálculos ou outra regras seriam exclusivos da camada de domínio da aplicação.

7.5 AMANDO SUA PROFISSÃO

Já falamos no *Capítulo Piratas* que um desenvolvedor que queira evoluir deve amar sua profissão. Já agora, como um Cavaleiro Jedi, ele deve ir além; nesse momento, sua profissão deve ser algo prazeroso.

"A maioria dos bons programadores programa não porque espera receber pagamento ou aplausos do público, mas sim porque é divertido programar." – Linus Torvalds

É fácil identificar se programar é algo prazeroso para você, se ele o for. Você troca algumas vezes diversões com os amigos por tardes desenvolvendo aquele código em que você vem trabalhando a tempos. Além de que trabalhar assiduamente em códigos fará você querer distribuí-los cada vez mais livremente, pois você terá orgulho de ver suas criações serem utilizadas pelo mundo. Você também saberá que, ao distribuir gratuitamente, seu conhecimento cooperou para que os outros não precisassem perder tempo, pois

aquilo que você criou os ajudaria, possibilitando-os usar seu tempo agora criando algo mais avançado, graças aos seus esforços.

Quando já somos Cavaleiros, além de nos sentirmos felizes desenvolvendo, vamos começar a refletir sobre o que podemos melhorar em nossa volta. Aquela vontade de fazer algo diferente além do código começará a despertar. Nesse momento, os Jedis começam a questionar o modelo social em que se enquadram, e ficarão preocupados com o excesso de lixo, desperdício de comida e água, questionando o modelo de mobilidade urbana. Isso pois já estará ganhando dinheiro suficiente para possuir um bom carro, mas, mesmo assim, escolherá o transporte público ou a bicicleta devido à sua eficiência para o que lhe é proposto. Não faz sentido arrastar 1 tonelada de ferro para passear por 10 ou 14 quilômetros levando apenas em media 80 kg. Ele pensará o quão ineficiente é esse transporte, além de ser o principal responsável pelo lançamento de poluentes na atmosfera terrestre.

Esse questionamento também se estende à indústria à qual faz parte, até que descobre que "o software privado é dependência e isso leva à colonização eletrônica. As empresas do software privado querem colonizar todos os países: eles tomaram os Estado Unidos, Europa e outros lugares do mundo." (Richard Stallman, fundador do movimento do software livre).

Assim, ele começará a fazer cada vez mais parte do círculo Jedi, começando a perceber que, além de ajudar na evolução de softwares, pode passar seus conhecimentos e *modus vivendi* a outros jovens, e colocá-los no caminho da Força.

Nesse momento, além de estar preparado para colocar alguém no caminho da Força, ele deverá se aprofundar em controlar suas

emoções e cada vez mais ampliar suas relações políticas pela empresa em que estiver trabalhando e outras com que se relaciona. O ponto chave a se domar aqui é o ego, pois grandes programadores tendem a se achar demais, o que os faz ir de encontro ao lado negro da Força.

Esse tipo de atitude gera atrito no ambiente de trabalho, o que não é nada saudável para a empresa, pois, em momentos de disputa de egos, deixa-se de fazer muita coisa justamente para sabotar um programador ou uma equipe, sendo que, no final, todos perderão com essas atitudes. Quando confrontados, os Jedis preferem o silêncio, relaxar, meditar sobre o fato e dar uma resposta fundamentada na razão, pois sabe que qualquer resposta momentânea estará cheia de emoções.

Sobre o controle das emoções, vivenciei uma situação fora do ambiente profissional, mas que deixa claro o poder de quem sabe fazer a coisa certa. Foi em um cruzamento de veículos, em São Paulo, que era cortado por uma ciclovia. Estava atravessando a rua pela faixa de pedestres e, ao parar no canteiro central da avenida, presenciei um motorista parando para um ciclista atravessar. Esse motorista estava irritado e reclamando por ter parado, para que o ciclista atravessasse; este, ao cruzar em frente ao carro, gentilmente acenou e suavemente pronunciou um bom dia.

Nesse momento, o que me impressionou foi a mudança no semblante do motorista, que retribuiu o bom dia e, ao mesmo tempo, abriu um belo sorriso. O ciclista utilizou-se da forma de inteligência mais preciosa para o crescimento de qualquer ser: a inteligência emocional.

Outra capacidade que o Jedi deve começar a evoluir

gritantemente é com relação a pensar antes de responder, ou seja, autoconsciência, sempre avaliar e ter plena certeza antes de qualquer resposta. Quando houver uma equipe envolvida, nunca dê uma solução técnica sem consultar todos os envolvidos, pois uma solução técnica dada no calor da emoção é como cavar sua própria cova.

Geralmente, qualquer solução técnica deve ser avaliada cuidadosamente e dada em outro momento, nunca logo após os requisitos serem passados pelo cliente, pois sabemos que muita coisa deve ser levada em consideração para que haja sucesso, e qualquer erro pode acarretar em um insucesso. Assim sendo, não custa nada perder um pouco mais de tempo em um momento crucial como esse, no qual, às vezes, uma decisão errada pode levar um projeto todo ao abismo.

Para conseguirmos crescer cada vez mais, devemos seguir dois conselhos: um que é apontado por mim, e outro que escutei do reitor quando ingressei em uma universidade pública renomada. O conselho dele foi: "Aqui não é como outras universidades. Para passar aqui, você precisa fazer apenas 3 coisas: **estudar, estudar e estudar**". Já o meu conselho para conseguir ser um Cavaleiro Jedi, além do proferido pelo reitor, é: **"PRATICAR, PRATICAR E PRATICAR"**.

Esses dois conselhos juntos assemelham-se a uma fantástica frase de um grande pedagogo brasileiro, Paulo Freire, que diz: "É fundamental diminuir a distância entre o que se diz e o que se faz, de tal forma que, num dado momento, a tua fala seja a tua prática." Isso quer dizer que devemos estudar e colocar tudo o que estudamos em prática.

Vejo muitas vezes pessoas dizendo coisas do tipo:

- O correto é fazer TDD!
- Se tivéssemos utilizado DDD, não teríamos problemas.

O que ocorre é que alguns técnicos leem, não praticam, e acabam conhecendo muito teoricamente, mas sem a prática. Logo, nunca vão conseguir fazer algo que faça alguma diferença, e ficarão com toda essa demagogia.

Sabendo disso, um Cavaleiro deve saber que não deve apenas utilizar o trabalho como sala de aula para evoluir como desenvolvedor, mas deve usar algum tempo fora do trabalho para estudar e praticar. Por exemplo, você pode criar pequenos sistemas de que precisa em seu dia a dia, utilizando as tecnologias que quer aprender. Nesse caso, além de estudar e praticar, você ainda constrói sistemas para as suas necessidades.

7.6 INDO ALÉM

Um Cavaleiro não se limita à utilização de tecnologia de uma companhia específica. Assim, sempre está buscando novas tecnologias open source, mesmo que não as use em seu trabalho. Sua busca não é em passar parâmetros para frameworks ou configurar servidores, mas sim abrir as fontes do Apache ou Mono e entender como foram criados.

Depois que tomar essas atitudes, um Jedi perceberá que vai começar a enxergar a construção de software de outras formas, e não ficará preso a tecnologias. Vai ver também que, para construir algo, não importa se está utilizando .Net, Ruby ou Java, mas sim os modelos utilizados para construir o que deseja.

7.7 ROTINA DE TREINO

Mantenha o código limpo

- Sempre que achar uma duplicação de código, refatore na hora. Tente manter seus códigos sempre limpos.
- Pegue códigos de outros programadores e tente mantê-los limpos.

Escreva códigos em padrões

- Tente, antes de sentar e programar, visualizar seus códigos dentro de padrões de projetos. Exercite isso ao máximo.
- Comece a olhar seus códigos já escritos e refatore-os para transformá-los em padrões.

Frameworks

- Começe a explorar os códigos de grandes frameworks, e tente entendê-los;
- Aos poucos, procure contribuir;
- Comece a automatizar seus códigos e transformar as automatizações em frameworks.

Sugestões de leitura

- *Código limpo: habilidades práticas do Agile software* – Robert C. Martin (2008);
- *Domain Driven Design: atacando as complexidades no*

coração do software – Eric Evans (2012);

- *Refatorando para padrões* – Joshua Kerievsky (2008);
- *Refatoração: aperfeiçoando o projeto de código existente* – Martin Fowler (2008);
- *Padrões de arquitetura de aplicações corporativas* – Martin Fowler (2007);

Por Carlos Bueno

Capítulo 8

MESTRE JEDI

"É preciso ter força para manter a emoção sob o controle da razão." – Thomas Lickona

O Jedi que se torna Mestre é facilmente identificado por possuir a capacidade de controlar suas emoções. Ele compreende que não ter controle o levará para o lado negro da Força.

Podemos notar que pessoas em TI que ocupam importantes

posições e têm altíssimo conhecimento e foram para o outro lado da Força são pessoas que impõem suas decisões e, geralmente, não controlam suas emoções. Isso as torna pessoas totalmente odiadas por toda a equipe. Isso não quer dizer que um Jedi não tomará decisões más ou que chateiem os outros membros da equipe, mas ele terá muito controle e saberá identificar os momentos em que trilha o lado negro.

A capacidade de controlar suas emoções e saber usá-las na hora certa é o que diferencia um Cavaleiro de um Mestre. O Mestre passa a maior parte do seu tempo meditando e procurando seus pontos fracos para melhorar sempre. Vamos detalhar esses pontos baseando-nos em estudos existentes, para entender melhor como a mente e a sociedade funcionam. A partir disso, ele terá um entendimento objetivo para sabiamente se tornar um grande Mestre.

8.1 AUTOCONSCIÊNCIA

"Não é a conversa das pessoas ao nosso redor que tem mais poder de nos distrair, mas a conversa da nossa própria mente." – Daniel Goleman (*Inteligência Emocional*, 1995)

Algumas ordens, como Rosa Cruz, pregam a máxima: *"Homem, conhece a ti mesmo"*, conselho do Oráculo de Delfos. Conhecer a si mesmo é não ser crítico demais ou um confiante excessivamente.

Sobre autoconsciência, Daniel Goleman define como *"uma compreensão profunda das próprias emoções, forças, fraquezas, necessidades e impulsos"* (*Liderança*, 1995).

Alguém autoconfiante é o tipo de pessoa honesta com todos que fazem parte da sua vida. Com essa honestidade, ele sabe analisar um projeto que está com o prazo apertado, e ser realista dizendo que tal prazo é impossível de se concretizar. Além disso, em casos como esse, é preciso manter o foco e a dedicação, pois não reconhecer um erro pode levar a atrasos. Se você for esse tipo de profissional, sempre será escolhido por clientes mais exigentes.

Um Mestre Jedi sabe sempre seu caminho e possui seus valores, sendo que rejeitará, por exemplo, uma proposta melhor de emprego, por mais tentadora que seja, se tal oferta não se enquadrar aos seus princípios e objetivos futuros.

Para atingir seus objetivos, o Mestre deve tomar decisões conscientes. Para isso, deve tomar muito cuidado com respostas imediatas, além de saber a hora de dizer "não". Mas, como assim dizer não?

Vamos nos aprofundar um pouco mais para entender esse *não*. Para isso, se observarmos, perceberemos que um profissional muito experiente, que é mestre em sua área, é extremamente requisitado. Isso é um imenso problema, pois, caso esteja com muitos assuntos a serem tratados, ele pode perder o foco e, sem isso, sabemos que poderá executar suas tarefas com equívocos, baixa qualidade etc.

Então, um mestre em sua profissão deve saber o quanto é capaz de produzir, e não ir além dos seus limites. Na realidade, nem deve chegar ao seu limite, deve sempre manter uma margem de risco, para que, em casos extremos, consiga entregar o que lhe foi proposto no prazo e com qualidade.

8.2 AUTOGESTÃO

"Impulsos biológicos dirigem nossas emoções. Não podemos eliminá-los, mas podemos fazer muita coisa para administrá-los." – Daniel Goleman (*Liderança*, 1995)

Autogestão é basicamente a força, que é aquela energia interior que conversa contigo a todo instante. Um Mestre deve possuir controle sobre essa energia, para que não fique preso a seus sentimentos.

Imagine-se no seguinte cenário: você acaba de assistir a uma apresentação de sua equipe sobre o projeto e percebe que eles não estão no caminho certo, errando em princípios básicos. A atitude comum de uma pessoa nessa posição seria, no mínimo, olhar a todos com aquele olhar fulminante, ou até dar um tapa na mesa, coisas do gênero. Mas um Mestre teria autocontrole, pensaria bem as palavras que usaria para suas respostas, analisaria sua participação no fracasso, pensaria em possíveis fatores mitigantes e, então, após refletir, reuniria todos e falaria sobre os impactos do fato na empresa, falaria sobre seus sentimentos e apresentaria uma solução fundamentada.

Assim como Goleman diz:

"A autogestão revela-se no autocontrole emocional (como permanecer calmo e lúcido sob estresse elevado ou recuperar-se dele rapidamente), na adaptabilidade e em se manter concentrado ao buscar metas." (*Liderança*, 1995)

O autocontrole é fundamental para um Mestre Jedi. Sem ela, o ambiente de trabalho facilmente será tomado por politicagens,

rivalidades, entre outros fatores que a levariam a um baixo desempenho.

Além do baixo desempenho, o mercado atual está impregnado com o modelo **empresarial tradicional**, em que todos são meros executores de tarefas passadas por um chefe. Porém, nós, Jedis, não trabalhamos assim, não necessitamos de pessoas dizendo o que devemos fazer, estudar etc.

Entendemos que a autogestão é saber a hora certa de fazer cada tarefa, sem a necessidade de alguém solicitando ou verificando o que foi feito. Estamos sempre além do esperado, pois não executamos apenas, mas procuramos entender a importância de cada uma, o momento em que deve ser entregue, a qualidade necessária e a expectativa. Ao fazermos isso, posicionamo-nos infinitamente à frente dos desenvolvedores tradicionais, que, infelizmente, possuem a mentalidade de funcionários do modelo empresarial tradicional.

8.3 EMPATIA

A empatia que um arquiteto deve possuir não é a de sentir as emoções de alguém, ou seja, ser sentimentalista. O que precisa é compreender e levar em conta os sentimentos dos outros profissionais nos processos decisórios.

Imaginemos um cenário no qual uma empresa perde algumas contas. Então, o profissional líder reúne todos e diz: *"vamos nos preparar, pois acredito que a empresa poderá demitir devido à perda de clientes"*. Já no mesmo cenário, outro líder reúne toda a equipe e diz estar preocupado com o ocorrido e que manteria todos sempre

bem informados sobre qualquer acontecimento. O que ocorreu entre os dois foi que o primeiro levou seu medo para toda equipe. Já o segundo foi honesto com todos, apresentou suas preocupações e os manteve calmos, enquanto o primeiro com certeza trouxe uma desestabilidade à empresa em um momento crucial.

8.4 HABILIDADE SOCIAL

A última das quatro habilidades que um Mestre deve possuir é a habilidade social. Não se trata de ser alguém simplesmente *cool*, aquele tipo que todos amam; na verdade, essa habilidade está além de ser apenas cordial.

Segundo Daniel Goleman, ela se resume em: *"[...] cordialidade com um propósito: conduzir as pessoas na direção que você deseja, seja a concordância com uma estratégia de marketing nova ou o entusiasmo com um novo produto."* (*Liderança*, 1995)

Um Mestre sabe que essa habilidade é muito importante, pois compreende que um desenvolvedor sozinho, por mais habilidoso que seja, não consegue atingir imensos resultados. Como gosto de dizer: um ótimo programador cria ótimos sistemas,; um ótimo programador, trabalhando harmoniosamente em equipe, inova.

8.5 PENSANDO COMO MESTRE

Mestres Jedis pensam além das suas simples tarefas, eles estão além de sua inteligência. Suas estratégias não se pautam apenas no bem de sua organização, mas em algo maior, pois sabem que são líderes que podem mudar muito mais que códigos, ou se beneficiar com seu conhecimento apenas para fazer fama ou fortuna. Assim,

sua inteligência e capacidade de lidar com as pessoas os colocam em posição de direcionar pessoas que os seguem, para algo maior.

A exemplo do que foi explanado, temos o Prêmio Nobel de economia, Muhammad Yunus, fundador do Grammen, que possibilitou muitas pessoas a saírem da pobreza extrema na Índia, ou Bill Gates que, além de estar emprenhado com a filantropia, convenceu vários bilionários a doarem metade da sua fortuna.

Devemos refletir com a frase de Steve Jobs: *"Bom, você sabe, o objetivo não é ser o homem mais rico no cemitério. Não é o meu objetivo pelo menos"*.

No mesmo sentido, um Mestre Programador deve possuir a habilidade de conectar pessoas a pessoas, ouvir atentamente e influenciar para melhor.

Um Mestre não desvia seu foco facilmente. Ele sabe que trabalhar em muitas coisas ao mesmo tempo o distancia da perfeição. A prática da meditação passa a ser cada vez mais constante; ele deve conseguir facilmente desligar sua mente de tudo, conseguir se focar em suas tarefas e fazê-las, uma a uma, com perfeição.

Devemos sempre lembrar do que disse Mestre Yoda: *"Teu foco é tua realidade"*. Qualquer líder que concentrar suas energias em uma tarefa específica levantará voo, já aqueles que não o fizerem, ficarão atolados em um pântano.

O ato de conduzir bem a atenção é uma habilidade que os Mestres devem ter bem desenvolvida: saber quando, por que e em que direção precisa guinar sua consciência.

Quando um Mestre atinge seu ápice, ele consegue entrar em estado de fluxo. Nesse momento, transpõe qualquer barreira limitante, e suas entregas são sempre em dia e com altíssima qualidade. Logo, um mestre, assim que inicia seu trabalho, elimina qualquer tipo de distração para poder entrar nesse estado. Geralmente, o maior inimigo é o Smartphone, então, todo mestre treina sua habilidade de fluxo mantendo ele fora de seu alcance.

8.6 ARTE DE NEGOCIAR

"O mérito supremo consiste em quebrar a resistência do outro sem lutar." – Lupércio Hilsdorf (*Saber negociar*, 2014)

Negociar é uma perícia que deve ser trabalhada por qualquer bom profissional. Seu *mindset* deve estar sempre focado no "correto" para poder atingir suas metas, assim será respeitado por todos e conseguirá baixar as guardas de quem quer obter algo.

A verdade sempre

Imagine o seguinte cenário: um bug retorna para você, e o *GoLive* da aplicação está próxima. Ao avaliar o sistema, você descobre que o desenvolvedor responsável não aplicou nenhuma boa prática e o código, além de errado, está com alta complexidade devido a usos indevidos de `if`s, `switch`s e `for` sem necessidade. O que você faz? Nesse caso, o caminho correto é refatorar ou reescrever do zero, porque a tentativa de correção vai apenas levar cada vez mais o código em questão para o *brownfield*.

Mas o mais importante é a honestidade com o cliente da aplicação. Deve-se dizer a verdade.

Exemplo real

Passei por muitas situações semelhantes, mas uma foi a mais emblemática. A implantação acabou sendo *shiftada* em 3 dias e, ao ir ao cliente para acompanhar a aplicação dos pacotes em produção, o gerente da área me questionou sobre a demora. Naquele momento, o gerente da minha empresa me chamou de canto e disse para contar algumas coisas que eram lindas, mas não eram verdadeiras.

Então, fui sincero, disse que o código com bug não havia sido bem escrito por nós, assim sendo, não estava a altura deles, e que, por isso, tive de refazer tudo utilizando as boas práticas e, o mais importante, entendendo o domínio onde se encaixava.

O cliente ficou extremamente satisfeito em assumirmos nossos erros e principalmente por avaliarmos que o código bem escrito seria melhor para eles.

Sobre obter algo, sabemos que é a necessidade constante de qualquer pessoa. Para atingir suas metas, você depende de uma estratégia e de que pessoas comprem suas ideias. Nesse sentido, o bom Mestre sempre procura identificar quem decide, quem influencia e quem coloca obstáculos na decisão, para que possa atingir os resultados esperados.

Além disso, mantive meu foco em entregas com qualidade e dentro dos prazos, sendo que todos os projetos em que trabalhei foram entregues assim. Um deles, em especial, reverteu a opinião

do cliente sobre nós, pois um dos diretores do cliente, que queria nossa cabeça, se encantou com a entrega feita, e esse sistema foi feito apenas por mim. Então, não tive dúvidas, demonstrei isso aos superiores, ganhei um bom aumento e, além disso, passei a ser reconhecido e respeitado por todos na empresa.

Esses resultados são atingidos facilmente se entendermos que, quando começamos a interagir com alguém para obter algo, não importa o quão bom seja, mas sim como o outro o percebe. Para isso, você deve ter uma empatia bem aguçada para perceber se está ou não indo bem e, caso perceba o contrário, você deve mudar sua estratégia com quem está negociando.

Comunique-se bem

Um Mestre deve entender que a comunicação é algo importante e que vai criar a sua marca. Então, precisa entender bem o momento certo em dizer algo e fazer a apresentação do que foi feito.

Se prestarmos atenção, vemos facilmente muitos profissionais dizendo que estão insatisfeitos por não serem reconhecidos pelos seus trabalhos, mas, ao avaliarmos, percebemos que apenas ele sabe isso, ninguém mais sabe. Ou quando outros sabem não são as pessoas que deveriam saber. Nesse caso, que engloba a maioria maciça dos profissionais, falta, além da comunicação, entender quais pessoas devem saber e como, pois uma abordagem mal feita pode ter efeito contrário – aí entra a empatia.

Exemplo real

Entrei em uma empresa, em meados de 2011, como especialista, para ajudar nas entregas que geralmente atrasavam e estavam com baixa qualidade.

Ao entrar, descobri os pontos que estavam gerando os problemas e, cada vez que executava uma tarefa que mudava esses pontos para melhor, informava as pessoas que eram responsáveis tecnicamente pelo time. Quando algo era grandioso, encaminhava também para alguns superiores.

A comunicação é um ponto importante para o crescimento profissional. Porém, além dele, precisamos também possuir estratégias de como se comunicar, pois diretamente não será seu conhecimento e experiência que fará as pessoas o reconhecerem. O ponto focal é a percepção que os outros têm de você. O que queremos dizer é que o seu crescimento está relacionado a uma batalha a ser travada na cabeça das pessoas.

Sabendo disso, um Mestre deve, por meio das capacidades emocionais (que já tratamos neste capítulo), entender seu ambiente social e trabalhar de forma que todos admirem. Resumidamente: trabalhar com a criação da sua marca pessoal.

Conto Árabe sobre os sonhos

Uma conhecida anedota árabe conta que um sultão sonhou

que havia perdido todos os dentes. Logo que despertou, mandou chamar um adivinho para que interpretasse seu sonho.

– Que desgraça, senhor! – exclamou o adivinho. – Cada dente caído representa a perda de um parente de Vossa Majestade.

– Mas que insolente – gritou o sultão, enfurecido. – Como te atreves a dizer-me semelhante coisa? Fora daqui!

Chamou os guardas e ordenou que lhe dessem cem açoites. Mandou que trouxessem outro adivinho e lhe contou sobre o sonho. Este, após ouvir o sultão com atenção, disse-lhe:

– Excelso senhor! Grande felicidade vos está reservada. O sonho significa que haveis de sobreviver a todos os vossos parentes.

A fisionomia do sultão iluminou-se num sorriso, e ele mandou dar cem moedas de ouro ao segundo adivinho. E quando este saía do palácio, um dos cortesãos lhe disse admirado:

– Não é possível! A interpretação que você fez foi a mesma que o seu colega havia feito. Não entendo por que ao primeiro ele pagou com cem açoites e a você com cem moedas de ouro.

– Lembra-te meu amigo – respondeu o adivinho – que tudo depende da maneira de dizer... Um dos grandes desafios da humanidade é aprender a arte de comunicar-se. Da comunicação depende, muitas vezes, a felicidade ou a desgraça, a paz ou a guerra. Que a verdade deve ser dita em qualquer situação, não resta dúvida. Mas a forma com que ela

é comunicada é que tem provocado, em alguns casos, grandes problemas.

A verdade pode ser comparada a uma pedra preciosa. Se a lançarmos no rosto de alguém pode ferir, provocando dor e revolta. Mas se a envolvemos em delicada embalagem e a oferecemos com ternura, certamente será aceite com facilidade.

Para conseguir atingir resultados ao negociar, como foi dito, devemos ser percebidos de forma boa. Para isso, é necessário se ater à comunicação, sendo importante ser claro e somente fazer afirmações quando tiver certeza.

8.7 CONHECENDO DE TUDO UM POUCO

Um mestre sabe que deve saber de tudo um pouco, e que é claro que deve aprofundar em alguns pontos, mas compreende que não pode ser especialista. Ele sabe que um especialista pode ser rápido no que faz, mas, quando necessita de interação com outras áreas (como infraestrutura ou negócio), tudo paralisa-se, e leva-se muito mais tempo do que o esperado.

Além de que, quando uma empresa que possui todas as funções extremamente separadas (como gerente de projetos, programador, infraestrutura etc.) passa por uma crise e precisa demitir, com certeza quem entender um pouco de outras áreas será mais útil e ficará no quadro da empresa. Logo, ser generalista é uma opção, não apenas de evolução, como profissional e também de sobrevivência.

Já me deparei com muitos cenários em que ser generalista foi um grande diferencial, por exemplo, em um grande projeto do qual participei, em que era um dos desenvolvedores C#. Nele, era utilizado a plataforma WEB com MVC. Acontece que, quando a aplicação foi para o ambiente de homologação do cliente – gerido por uma empresa especializada –, acabaram ocorrendo bugs que impediram que o sistema operasse.

Todos ficaram desesperados procurando erros em códigos e versões de bibliotecas, enquanto o erro era em configurações no servidor IIS, que consegui detectar e pude auxiliar os analistas de infraestrutura a colocarem o sistema em homologação.

Nesse caso, acabei garantindo meu emprego quando a empresa precisou se desfazer de algumas pessoas depois de perder alguns contratos. Além de que passei a participar de todas as equipes de desenvolvimento, pois todos queriam que eu participasse para garantir que o projeto sempre chegaria ao sucesso.

Assim, é interessante que você se mantenha como Mestre, uma visão estratégica de seus conhecimentos. Caso programe, por exemplo, em C#, você deve conhecer sobre toda a infraestrutura por trás, ou seja, deve conhecer de IIS, AD, Sharepoint, Exchange, SQL Server. Nesse caso, não apenas alguns comandos básicos, mas entender como configurar , operar e programar em SQL Server, pois sempre será útil manter-se em evidência no time ou na empresa.

8.8 ROTINA DE TREINO

Sugestões de leitura

- *O programador apaixonado: construindo uma carreira notável em desenvolvimento de software* – Chad Fowler (2014);
- *Liderança: a inteligência emocional na formação do líder de sucesso* – Daniel Goleman (1995);
- *Saber negociar: competência essencial* - Lupércio Hilsdorf (2014);
- *Foco: a atenção e seu papel fundamental para o sucesso* – Daniel Goleman (2013);
- *O ócio criativo* – Domenico De Masi (2000);
- *Mente magnética: a ciência para atrair riqueza, prosperidade e tudo mais que você desejar* – Rogério Job (2014);
- *Inteligência emocional: a teoria revolucionária que redefine o que é ser inteligente* – Daniel Goleman (1995);
- *As ferramentas intelectuais dos gênios* – I.C Robledo (2012);
- *O poder do hábito: por que fazermos o que fazemos na vida e nos negócios* – Charles Duhigg (1995);
- *O líder Alfa: desenvolva o instinto da liderança e forme equipes de alta performance* – Renato Grinberg (2014).

Agindo com a cabeça

- Não dê respostas técnicas imediatamente. Volte para sua empresa ou casa, e pense. No dia seguinte, após analisar

todas as possíveis respostas, dê a melhor.

- Seja sempre cordial, mas verdadeiro em suas respostas.
- Tente meditar ao menos 3 vezes por semana. Se possível, no meio do expediente do trabalho, vá a algum lugar reservado e fique 10 minutos sem celular, sem nada, apenas prestando atenção em sua respiração.
- Sugerimos se aprofundar no assunto no site do Daniel Cuckier http://www.agileandart.com/category/meditation/.

Treinando sua mente

Existem alguns games que trabalham a plasticidade cerebral, aperfeiçoando algumas aptidões. Procure criar o hábito de jogá-los.

Lumosity é um site que possui games criados por neurocientistas, que trabalham a plasticidade (http://www.lumosity.com).

Por Carlos Bueno

Capítulo 9

CONCLUSÃO

Para evoluir como desenvolvedor e nos tornarmos um Mestre, devemos trilhar um caminho árduo. Vimos que, inicialmente, devemos pensar diferente e romper muitos paradigmas, como foi visto no Capítulo *Pirata*.

Enquanto escrevia o livro, ouvi algumas críticas em relação a esse capítulo, então, foi o que me fez pensar mais e mais em mantê-lo. Se olharem, por exemplo, a série Silicon Valley, verão que a equipe de desenvolvimento tem uma bandeira pirata, a qual aparece em vários momentos.

Apesar de a série ser uma comédia, ela tem uma boa ambientação e sabe usar perfeitamente a ideia de piratas. É uma ótima série para poder compreender melhor como verdadeiros programadores pensam, sentem e agem, além de mostrar que apenas programar não é tudo, temos de conhecer desde *business plan* até gestão, e estar sempre com a mente sã.

Outro material interessante que demonstra o pensamento pirata é o filme, já citado, *Piratas do Vale do Silício*, que conta a trajetória dos Stev(s) (Wozniak e Jobs) e Gates, mostrando como os *beatniks*, *hackers* e outros da contracultura tornaram-se as maiores empresas do mundo.

Além de pensarmos diferentes, aprendemos que existe um caminho evolutivo para chegarmos ao nível máximo. Mas a evolução não está relacionada à idade, pois podem existir programadores de 40 anos que nem Padawan são, e podem existir jovens de 18 que possuem todas as características de um Mestre.

O interessante é conseguir enxergar em que nível você se enquadra para poder progredir. Alguns perceberão que, intuitivamente, já estavam no caminho certo, então, podem analisar alguns pontos que não conheciam e se aprofundar para melhorarem mais ainda suas capacidades.

Ainda na obra, foi apresentado que devemos nos ater, na maioria do tempo, a pontos fora da área técnica, pois, para evoluir tecnicamente, temos de evoluir interiormente. E essa evolução, como foi visto, só acontece quando a pessoa consegue ser autocrítico e perceber que sua postura pode sempre ser melhorada. Essa é a regra do espelho: muitas pessoas têm medo de se olharem no espelho e verem que são, na verdade, tudo o que não gostariam de ser.

Assim, podemos trabalhar e ir vagarosamente esculpindo nosso ser, pois nosso interior reflete exteriormente. E o processo é duro, não conseguimos mudar hábitos da noite para o dia. Alguns, quando estão em transformação, parecem ser feitos mecanicamente, ser falsos, forçados, mas, com o tempo, ele se torna um verdadeiro hábito e passa a fazer parte do seu ser interno e externo.

Entretanto, lembre-se de duas coisas. Primeiro é que transformar maus hábitos em bons leva tempo e vontade. Muitas vezes lutaremos contra nossa própria consciência, mas temos de

persistir e, principalmente, não cair quando ouvirmos alguém dizer algo que estimule o pensamento da nossa cabeça. Outro ponto é que os maus hábitos não podem ser apagados, mas podem ser substituídos por melhores. Porém, cuidado, uma hora eles podem aparecer, pois, como disse, não podemos apagá-lo. Quem quiser se aprofundar sobre o tema, recomendo ler o livro *O poder do hábito: por que fazermos o que fazemos na vida e nos negócios*, de Charles Duhigg.

Relacionado a esse assunto, abordamos o que faz uma pessoa ser um líder de sucesso. Vimos no decorrer dos capítulos que, baseado em estudos, o principal para o sucesso de um profissional em qualquer área do conhecimento é sua capacidade de interagir e pensar de forma consciente. Ou seja, aquele *nerd* inteligente pode não ser uma pessoa de sucesso, pois, na realidade, um grande líder necessita ter um pequeno desvio-padrão de QI para poder lidar com as complexidades do seu trabalho, entretanto, o principal é a **inteligência emocional**.

Alguns estudiosos acompanharam muitas crianças por um período de 30 anos e descobriram que os com maior capacidade de se relacionar foram as pessoas com maior sucesso em todos os quesitos: familiar, profissional, educacional etc.

Mas devemos entender que a parte técnica possui sua importância também. As habilidades para contornar a constante alteração são as mais importantes, e qualquer tipo de perícia que envolva a arte de construir bons códigos utilizando padrões é bem-vinda. Foi mostrada a diferença entre os códigos que usavam certas técnicas e outros que não utilizaram, e ficou nítida a necessidade de um bom Mestre evoluir nessa área.

A obra é apenas um guia de referência para conseguir evoluir, pois não seria possível tratar detalhadamente cada um dos pontos em apenas um livro. O que realmente queremos passar é o espírito e as características necessárias para se alcançar o sucesso. Agora, dependerá de você seguir o caminho e buscar cada vez mais conhecimento.

Acompanhem mais em nossos endereços:

- Fanpage no Facebook – https://www.facebook.com/guiadomestreprogramador
- Códigos-fonte do livro – https://github.com/Sennit/Mestre-Programador
- Blog – http://guiadomestreprogramador.com/
- Lista de discussão do livro – http://forum.casadocodigo.com.br

Que a Força esteja com você!

Por Carlos Bueno

Capítulo 10

BIBLIOGRAFIA

DE MASI, Domenico. *O ócio criativo.* Rio de Janeiro: GMT Editores, 2000.

DORMEHL, Luke. *A revolução Apple: Steve Jobs, a contracultura e como os loucos dominaram o mundo*. Rio de

Janeiro: Alta Books Editora, 2014.

DUHIGG, Charles. *O poder do hábito: por que fazermos o que fazemos na vida e nos negócios*. Rio de Janeiro: Editora Objetiva, 1995.

EVANS, Eric. *Domain Driven Design: atacando as complexidades no coração do software*. São Paulo: Alta Books, 2012.

FOWLER, Martin. *Padrões de arquitetura de aplicações corporativas.* Rio de Janeiro: Editora Bookman, 2007.

FOWLER, Martin. *Refatoração: aperfeiçoando o projeto de código existente*. Rio de Janeiro: Editora Bookman, 2008.

FOWLER, Chad. *O Programador Apaixonado: construindo uma carreira notável em desenvolvimento de software*. São Paulo: Casa do Código, 2014.

FREIRE, Paulo. *Pedagogia da autonomia: saberes necessários à prática educativa*. São Paulo: Editora Paz e Terra, 1997.

GOLEMAN, Daniel. *Inteligência emocional: a teoria revolucionária que redefine o que é ser inteligente*. Rio de Janeiro: Editora Objetiva, 1995.

GOLEMAN, Daniel. *Foco: a atenção e seu papel fundamental para o sucesso*. Rio de Janeiro: Editora Objetiva, 2013.

GOLEMAN, Daniel. *Liderança: a inteligência emocional na formação do líder de sucesso*. Rio de Janeiro: Editora Objetiva, 2014.

GRINBERG, Renato. *O líder Alfa: desenvolva o instinto da*

liderança e forme equipes de alta performance. São Paulo: Editora Gente, 2014.

HILSDORF, Lupércio. *Saber negociar: competência essencial*. São Paulo: DVS Editora, 2014.

ISAACSON, Walter. *Steve Jobs: a biografia*. São Paulo: Companhia das Letras, 2011.

JOB, Rogerio. *Mente magnética: a ciência para atrair riqueza, prosperidade e tudo mais que você desejar*. Mente livre: 2014.

KERIEVSKY, Joshua. *Refatoração para padrões*. Porto Alegre: Bookman, 2008.

LEARY, Timonthy. *Chaos and cyber culture*. Berkeley: Ronin Publishing, 1994.

MARTIN, Robert C. *Código limpo: habilidades práticas do Agile software*. São Paulo: Alta Books, 2008.

MARX, Karl. *O Capital. O processo de Produção do Capital*. Editora Civilização Brasileira, 2014.

ROBLEDO, I.C. *As ferramentas intelectuais dos gênios*. Amazon: 2012.

SCHEIN, Edgar H. *Cultura organizacional e liderança*. São Paulo: Editora Atlas, 2009.

STOËTARD, Michel. *Jean-Jacques Rousseau*. Recife: Editora Massangana, 2010.

WALLACE, Daniel. *O caminho Jedi*. São Paulo: Editora Bertrand Brasil, 2013.

WOODWARD, Collin. *A República dos Piratas: a verdadeira história dos Piratas do Caribe e do homem que os derrotou*. Barueri: Novo Século, 2007.