Aplicando técnicas de *Clean Code* em uma fábrica de *software* acadêmica - Qualidade no Desenvolvimento

Matheus Vieira Gonçalves

Centro Universitário de Anápolis – Unievangélica

Caixa Postal 122 e 901 – 75.083-515 – Anápolis – GO – Brazil

matheusvieira1900@hotmail.com

Abstract: The present work aims to describe and explain a primordial concept for the current scenario of software development - the Clean Code, along with the rules and techniques applied by it, seeking in this way to implement them in the factory seeking to improve the already existing development within of this academic environment. To achieve this goal the work will first present the main concepts of the Clean Code together with the problems that many companies face for not following such concepts considered basic. Later on we will explain further the real meaning of the Clean Code and its ideal use.

Keywords: Codification, clean code, quality, organization.

Resumo: O atual trabalho tem por objetivo descrever e explicar um conceito primordial para o cenário atual de desenvolvimento de software - o Clean Code, juntamente com as regras e técnicas aplicadas por ele, buscando dessa forma implementá-las na fábrica buscando melhorar o desenvolvimento já existente dentro desse ambiente acadêmico. Para atingir tal objetivo o trabalho primeiramente apresentará os principais conceitos do Clean Code juntamente com os problemas que muitas empresas enfrentam por não seguir tais conceitos considerados básicos. Posteriormente será explicado mais a fundo o real significado do Clean Code e sua ideal utilização.

Palavras-chave: Codificação, código limpo, qualidade, organização.

1. Introdução

Com as constantes evoluções na área da tecnologia tanto o *hardware* quanto o *software* sofrem diversas mudanças, tornando-se cada vez mais complexos. Dentro do desenvolvimento de *software* em particular as manutenções tornaram-se aspecto de suma importância em seu desenvolvimento, as quais chegam a consumir cerca de 75% de todo seu custo de vida, assim como é descrito pela autora Cristine em seu artigo: "Gerência de configuração de *software*".

A produção de um *software* é realizada pela constante leitura e escrita de código. O autor Robert C. Martin após analisar o histórico de atividades do seu próprio editor enquanto estava programando, afirma que a proporção de tais atividade é em média de 10 para 1, ou seja, grande parte dos esforços realizados na produção se baseiam na leitura e

interpretação de código, por isso, um código bem escrito que facilite essas ações não é apenas desejável, mas necessário no cenário atual.

Após ficar ciente de tal fator, torna-se de grande relevância estudar um conceito que vem tomando maiores proporções na atualidade, o qual é denominado de "*Clean Code*". Tal conceito visa aplicar um conjunto de regras e boas práticas de programação na construção de códigos mais "limpos", os quais visam aumentar gradativamente a qualidade do produto e a produtividade da equipe.

Antes de aprofundarmos diretamente sobre o c*lean code*, vamos falar sobre conceitos básicos que estarão diretamente ligados a esse modelo de programação, visando obter uma melhor compreensão sobre o que está sendo explicado no decorrer do trabalho. Vale ressaltar que o atual trabalho foi baseado na leitura do livro de autor Robert C. Martin: "Código Limpo: Habilidades Práticas do *Agile Software*", e complementado com a leitura de outros artigos da área.

2. Referencial teórico

2.1. O código

Segundo Robert C. Martin, muitos indivíduos acham que a programação (codificação) está se tornando uma área ultrapassada, a qual será substituída por maáquinas capazes de produzir códigos conforme os requisitos passados, entretanto isso não passa de um pensamento equivocado, assim como afirma o autor. Uma explicação para essa afirmação baseia-se no fato de que as máquinas (caso existissem) desenvolveriam o código através de uma explicação clara e concisa do que o *software* deveria ser capaz de fazer, tal explicação nada mais é do que o ato de codificar.

Outro fator que explica tal afirmação baseia-se na complexidade de se entender e repassar um requisito pedido pelo cliente, já que o mesmo na maioria das vezes não sabe expressar com clareza o que realmente deseja. "Nem mesmo os seres humanos, com toda sua intuição e criatividade, têm sido capazes de criar sistemas bem-sucedidos a partir da carência confusa de seus clientes" (MARTIN, 2008). Dessa forma esse problema só pode ser solucionado aos poucos, à medida que o produto é codificado e mostrado ao cliente.

As máquinas, diferente de nós humanos, necessitam de comandos diretos e claros para serem executados. Para que uma máquina fosse capaz de programar um *software* que atendesse às expectativas do cliente, seria necessária uma grande quantidade de informação concisa e clara, sem brechas para erros ou subjetividade. Tais informações como citadas anteriormente, ainda são uma realidade distante do cenário atual do mercado de *software*.

Portanto é viável afirmar que a programação ainda é e será durante um bom tempo uma área de suma importância na produção de um *software*. Graças a esse fator, torna-se de responsabilidades dos programadores desenvolverem códigos de boa qualidade, visando sempre manter o um bom nível de produtividade independente do estado atual do projeto.

2.2. Código ruim

O código, por mais simples que seja, deve apresentar uma boa qualidade em sua estrutura, já que a mesma pode ser utilizada no decorrer do projeto tomando dessa forma maiores proporções conforme o desenvolver do produto.

Diversos casos foram relatados na internet, alguns inclusive no próprio livro de Robert C. Martin, sobre empresas que desenvolveram um produto de *software,* mas não foram capazes de mantê-lo ativo graças ao desenvolvimento desorganizado de código, o que gerou inúmeros problemas na implementação de novas funcionalidade e manutenção das já existentes.

No geral, os códigos ruins são os responsáveis pela maioria dos problemas que os programadores enfrentam em suas rotinas. Isso gera a seguinte questão: "Então porque eles são produzidos?". Basicamente um código ruim sempre é produzido de forma indireta devido a alguns motivos, sendo eles: falta de experiência, prazo curto que deve ser comprido, funcionalidades complexas de serem implementadas e manutenções em códigos que já estavam ruins.

Pesquisas realizadas dentro da própria Fábrica de Tecnologia Turing comprovam a afirmação anterior, onde os alunos, após adquirirem certa experiência torna-se capazes de identificar códigos mal desenvolvidos e os motivos que levaram ao seu mal desenvolvimento, que no geral estão ligados aos problemas citados anteriormente.

Em sua grande maioria as manutenções em códigos que já estavam ruins são os maiores culpados nesse cenário já que podem ocasionar um efeito "bola de neve". Isso ocorre pois muitas das vezes as pessoas tendem a ter menor preocupação com algo que já está estragado, o que os leva a agravar a situação já existente, tal fator pode ser comprovado através de um experimento realizado nos Estados Unidos assim como foi explicado pelo autor Oliveira em seu artigo "*Clean Code*: Desenvolvimento com qualidade".

O experimento citado anteriormente consistia em deixar um carro com uma janela quebrada em uma esquina. Em alguns dias o caro já possui mais janelas quebradas. Ao longo de algumas semanas o carro já estava todo depenado. (OLIVEIRA, 2014)

Para realizar uma comparação outro carro foi colocado no mesmo local, mas dessa vez não apresentava nenhum defeito. Esse veículo ficou no mesmo local durante algumas semanas (tempo semelhante ao primeiro veículo) e nenhum dano lhe foi causado, diferentemente primeiro carro, assim como afirma Oliveira.

Dessa forma podemos concluir que para a produção de um código deve-se levar em consideração a estrutura do mesmo, visando dessa obter o melhor nível de qualidade possível, possibilitando uma melhora na produtividade, seja na implementação de novas funcionalidade ou na manutenção das já existentes, evitando assim o efeito "bola de neve" que é gerado por códigos ruins no projeto.

É válido ressaltar também que prazos curtos ou implementações complexas não devem ser motivos para se aceitar a produção de um código mal estruturado. Manter o código com o melhor nível de qualidade possível vai consequentemente auxiliar a cumprir com o prazo e a desenvolver uma implementação mais complexa, além de prover diversos

benefícios ao decorrer do projeto, como facilidade na manutenção ou nova implementação e até mesmo servirá como forma de consulta para novatos no projeto ou para futuras implementações. (MARTIN, 2008)

2.3. Problemas Envolvendo Código mal desenvolvido

Qualquer código quando mal produzido pode afetar drasticamente o projeto, seja atrasando a produtividade de quem o produziu ou de outras pessoas que ficaram responsáveis por dar manutenção ou implementar uma nova funcionalidade através dele.

Uma parte do livro de Robert retrata isso claramente através de um exemplo bem comum em empresas que estão iniciando no mercado. No exemplo dado temos uma equipe que acaba de iniciar um projeto, a taxa de produtividade está alta, mas a maioria dos programadores ainda não possui alto nível de maturidade na produção de um bom código.

Com o decorrer do tempo o projeto começa a ficar mais complexo, novas funcionalidades são implementadas a todo momento e o código já não apresenta uma boa legibilidade. Quanto mais o tempo passa mais a taxa de produtividade decai prejudicando prazos e aumentando gradativamente os custos.

A gerência do projeto tenta solucionar o problema alocando mais pessoas para a equipe, entretanto os novos integrantes não possuem um nível de conhecimento alto sobre o produto que está em produção, impossibilitando assim a tomada de decisões sobre o que pode ser ruim ou benéfico para o projeto, além de não possuírem nenhuma familiaridade com um código que está um completo “caos”.

A junção de todos esses problemas leva a apenas uma possível solução: começar o projeto do zero com uma nova estruturação. Mesmo que a gerência não goste da ideia graças aos custos, acaba aceitando por ser a única solução. Para aplicá-la a equipe se divide em duas, os mais experientes começam a nova estruturação e os novatos cuidam da manutenção do código.

Um dos problemas desta solução é que os novatos provavelmente não serão capazes de realizar a devida manutenção ao *software* já produzido e muito menos conseguirão implementar novas funcionalidades ou modificações. Outro problema é que nada garante que o novo *software* que está sendo produzido com a nova estruturação não tenha o mesmo destino de seu antecessor, o que poderia levar a empresa a cancelar o projeto ou nos piores casos a falência.

Portanto através desse exemplo é possível notar a importância da produção de um código de qualidade desde o início do projeto, visando prevenir problemas futuros que podem ocorrer com o aumento da complexidade do produto. Entretanto, o que que realmente seria um código de boa qualidade? Tal pergunta será respondida a partir dos próximos tópicos onde começaremos a abordar sobre o *Clean Code.*

2.4. Definição de *Clean Code* (Código limpo)

Antes de entrarmos mais a fundo em como desenvolver um código limpo ou *clean code*, precisamos primeiro saber o que ele realmente significa. Como código limpo pode ter

diversas definições que variam de programador para programador, o livro de Martin busca trazer as definições utilizadas pelos programadores mais conhecidos e com maior experiência na área, sendo eles: Bjarne Stroustrup, criado do C++; Grady Booch, autor do livro Object Oriented Analysis and Design with applications; Michael Feathers, autor de Working Effectively with Legacy Code, entre outros. A seguir serão listadas as definições de cada programador citado acima.

Primeiramente temos Bjarne Stroustrup, um cientista da computação dinamarquês e professor catedrático da Universidade do Texas A&M, define um código limpo como um código:

> [...] elegante e eficiente. A lógica deve ser feita para dificultar o encobrimento de bugs, as dependências mínimas para facilitar a manutenção, o tratamento de erro completo de acordo com uma estratégia clara e o desempenho próximo do mais eficiente de modo a não incitar as pessoas a tornarem o código confuso com otimizações sorrateiras. O código limpo faz apenas uma coisa.

Outro grande nome na área - Grady Booch, um informático estadunidense, criador dos livros "Software Engineering with Ada" e "Object-Oriented Analysis and Design with Applications", ganhador dos prêmios IBM Fellow (2003) e Turing Lecture (2007), define um código limpo como: “[...] simples e direto. Ele é tão bem legível quanto uma prosa bem escrita. Ele jamais torna confuso o objetivo do desenvolvedor, em vez disso, ele está repleto de abstrações claras e linhas de controle objetivas.”

Michael Feathers, autor do livro “Working Effectively with Legacy Code”, através de uma afirmação mostra a importância do código limpo e alguns de seus benefícios:

> Eu poderia listar todas as qualidades que vejo em um código limpo, mas há uma predominante que leva a todas as outras. Um código limpo sempre parece que foi criado por alguém que se importava. Não há nada de óbvio no que se pode fazer para torná-lo melhor. Tudo foi pensado pelo autor do código, e se tentar pensar em algumas melhoras, você voltará ao início, ou seja, apreciando o código deixado para você por alguém que sem importa bastante com essa tarefa.

2.5. Práticas para desenvolver um bom *Clean Code*

Para o desenvolvimento de um código limpo, não basta apenas saber o que ele é ou sua importância, é necessário se adequar a um conjunto de regras e práticas que auxiliam na sua criação, além de colocá-los em prática continuamente. Não basta saber todos os conceitos por trás de um Clean Code, é necessário praticar esses conceitos regularmente a fim de torná-los comum em sua vida profissional.

A seguir serão listados alguns dos recursos mais utilizados durante uma codificação e também a forma correta de utilizá-los conforme os conceitos e práticas do *clean code.*

2.5.1. Nomes significativos

Durante a construção de um *software* um dos fatores mais presente e utilizados são os nomes. Os nomes são atribuídos às funções, procedimentos, variáveis, parâmetros, classes, diretórios, entre outros (MARTIN, 2008). Dessa forma, os nomes possuem papel de destaque dentro do Clean Code. Um nome bem-criado atua diretamente no nível de produtividade do programador e da equipe em si, facilitando buscas e leituras no código e até mesmo podendo suprir a necessidade de comentários.

Para a construção de bons nomes algumas técnicas podem ser utilizadas, sendo elas: utilizar nomes que revelem seu propósito; utilizar verbos para funções, métodos e procedimentos e utilizar substantivos para diretórios, arquivos, classes, objetos e variáveis.

A escolha de bons nomes pode levar tempo e até mesmo ser cansativa, por isso é bem comum ver variáveis ou métodos que apresentam nomes abreviados que não são capazes de expressar sua real função dentro do código. Em alguns casos variáveis e métodos podem apresentar nomes totalmente diferentes de seu real propósito, prejudicando dessa forma a leitura e interpretação do código. Dessa forma chegamos a um problema muito comum: Nomes confusos ou desatualizado.

Através de uma pesquisa de campo realizada dentro da Fábrica de Tecnologias Turing, foi possível provar o problema citado anteriormente, onde grande parte dos códigos analisados apresentavam nomes que no geral eram confusos ou desatualizados.

É normal que uma parte do código (seja ela uma classe, função ou variável) que foi criada no presente seja alterado no futuro, algumas podem ser modificadas apenas a fim de melhorias podendo ter seu objetivo totalmente distorcido do original, dessa forma torna-se necessário ficar atento as modificações que um código sofre ao longo do tempo, buscando sempre que necessário atualizar os nomes criados a fim de não ocasionar problemas futuros ligados a nomes desatualizados.

Para criar um nome de uma classe ou função é necessário que o mesmo seja capaz de expressar seu real significado, ou seja, ele deveria dizer porque existe, o que faz e como deve ser utilizado. Dessa forma um programador ao se depara com essa classe ou função não precisa de muito tempo para entender seu propósito. Se um nome necessita de um comentário para ser entendido, o mesmo deve ser repensado e refeito, pois não é capaz de revelar seu objetivo. (MARTIN, 2008)

Nomes adequados criados através de boas práticas de padronização podem auxiliar na leitura de um código implicando dessa forma em um aumento de produtividade. (NEITZKE, 2011)

2.5.2. Cuidados ao criar funções

O primeiro passo para a criação de uma função de qualidade é a preocupação com o seu tamanho. Para criar uma função de qualidade corretamente, ela deve apresentar o menor tamanho possível (MARTIN, 2008), isso pois, o seu tamanho impacto diretamente em sua utilização/manutenção.

Para a criação de funções, também deve ser levado em conta a endentação das linhas e a utilização adequada dos blocos de instruções, tais como *if, else, while*, buscando

manter o código o mais organizado possível. Dessa forma, a leitura e interpretação de uma função pode ser melhorada de forma exponencial a sua organização.

É de grande importância que uma função só seja responsável por realizar apenas uma tarefa (MARTIN, 2008), ou seja, uma função de soma por exemplo só deve ser responsável por somar e não por subtrair ou multiplicar. Assim como dito no livro, "As funções devem fazer uma coisa. Devem fazê-la bem. Devem fazer apenas ela".

Implementar funções que realizam apenas uma tarefa proporciona uma maior flexibilidade de código, além de reduzir consideravelmente a complexidade do mesmo. Dessa forma aplicar tal prática pode trazer grande benéficos conforme o andamento do projeto.

2.5.3. Utilizar comentários corretamente

No geral, assim como estudado nos dois tópicos anteriores, um código sempre deve ser autoexplicativo, ou seja, ele deve ser capaz de realizar uma determinada ação e explicar como ela está sendo realizada. Entretanto, existem momentos na programação em que um código não será capaz de transmitir todas as informações necessárias. Para solucionar tal problema, é adotado a utilização de comentários.

Autores como Brian W. Kernighan afirmam que não se deve inserir comentários em um código ruim, apenas reescreve-lo, entretanto nem sempre apenas códigos ruins vão necessitar de comentários, já que isso depende de diversos fatores, tais como: complexidade do algoritmo e do projeto, implementações inovadoras e ainda desconhecidas pelos demais membros, locais críticos no código, entre outros.

Quando um comentário é necessário, o mesmo deve ser utilizado da melhor maneira possível, dessa forma um comentario deve seguir algumas regras de implementação.

Uma das regras está ligada ao seu tamanho. Um comentário deve ser o mais enxuto possível, visando explicar de forma clara e direta determinado motivo que levou a sua utilização.

Um comentário é utilizado como uma ferramenta de comunicação entre o programador e o código, dessa forma o mesmo deve ser sempre atualizado conforme o possível, visando não transmitir mensagens erradas aos desenvolvedores ocasionadas por mudanças no projeto.

Geralmente os comentários são utilizados para: transmitir uma informação em específico, explicar uma intenção, gerar um esclarecimento do que está sendo trabalhado e alerta sobre as consequências que um código pode gerar,

Um bom comentário pode possuir grande importância dentro de um código, dessa forma sua utilização deve ser feita de forma cuidadosa visando não causar problemas ao invés de solucionar os já existentes. Uma forma de verificar se um comentário é bom é analisá-lo e tentar descobrir uma forma melhor de reescrevê-lo, caso não seja possível esse é um bom comentário. (MARTIN, 2008)

3. Método de Pesquisa

O método de pesquisa utilizado na coleta de dados para a criação do atual trabalho se baseou na junção de três principais formas de pesquisa, sendo elas a pesquisa quantitativa, a pesquisa qualitativa e a revisão da literatura.

Tanto a qualitativa quanto a quantitativa foram utilizados na pesquisa de campo, realizada dentro da Fábrica de Tecnologias Turing (FTT), onde as qualitativas buscaram investigar fatores que levavam a produção de código "ruim", e as quantitativas foram utilizadas para avaliar a quantidade de aparições que determinados erros ocorriam no projeto.

Já a revisão literária foi utilizada a fim de reunir informações já existente sobre o problema tratado, a fim de obter uma visão crítica sobre ele, possibilitando dessa forma uma melhor discussão sobre o assunto visando a criação de ações para solucioná-lo.

4. Abordagem proposta

Tendo em vista todos os problemas que já foram e ainda são ocasionados pela falta de estruturação adequada de código, tanto na Fábrica de Tecnologias Turing (FTT) quanto em outras fábricas, torna-se necessário a adoção de medidas que tenham por objetivo resolver ou ao menos amenizar os problemas presentes.

Dessa forma, a abordagem proposta para a implementação de medidas para tratar esse atual problema presente na FTT, se baseia na adoção na adoção de padrões e novas etapas nos processos a serem seguidos durante o desenvolvimento do código, possibilitando dessa forma uma maior fiscalização sobre o nível de qualidade do que está sendo produzido.

Dentro da Fábrica de Tecnologias Turing algumas medidas já foram implementadas a fim de auxiliar na produção de um bom *clean code*. Uma dessas medidas se baseia na utilização de uma nova etapa dentro do processo de desenvolvimento de código, denominada "*Code Review"*.

Code review ou revisão de código é uma etapa dentro do processo de desenvolvimento na qual se encontra logo após o desenvolvimento funcional do requisito e que antecede o envio do requisito para o teste. A função dessa etapa é a criação de uma revisão de código que é realizada por outro membro da equipe, visando dessa forma encontrar os erros já citados anteriormente (nomes irregulares, funções grandes, comentários inúteis, etc).

Com a utilização da *code review* foi possível constatar uma melhora significativa sobre o nível de qualidade do código produzido, já que o mesmo era revisado por mais de um membro antes de ser entregue, possibilitando uma maior localização e correção dos erros relacionados ao *clean code*.

Tendo em vista essa os benefícios ocasionados por essa nova etapa no processo, uma proposta de melhoria se baseada na aprimoração da *code review*. Atualmente a *code review* só ocorre antes do primeiro envio do requisito ao teste, dessa forma o envio de correções solicitadas pela equipe de teste já não passa por essa fase.

Uma possível melhoria seria a realização da *code review* sempre que um requisito tiver seu código modificado, entretanto ela sempre deveria ser realizada por um membro diferente da equipe, assim quando as variações de membros acabarem ela comece a ser realizada pelo primeiro membro até chegar no último novamente, visando uma revisão de código pelo maior número de pessoas possíveis.

Um benefício da implementação dessa prática repetitiva, seria o constante refinamento do código, melhorando-o a cada nova modificação.

Outra proposta de melhoria seria a implementação de configurações padrões nos editores de código, auxiliando dessa forma na formatação dos códigos produzidos. Tal proposta de melhoria já foi introduzida anteriormente como uma forma de experimento, entretanto não foi implementada de forma profunda.

Com o experimento da proposta anterior foi possível notar uma pequena melhora na produção de código, dessa forma com um maior estudo para descobrir formas de configurações mais avançadas e adotar a utilização de melhores padrões para os editores de código, seria possível a obtenção de uma melhoria significativa na qualidade dos códigos produzidos, e reduziria grande parte dos esforços utilizados para a padronização básica dele.

Portanto, tanto a *code review* quanto a utilização de configurações dos editores de código poderiam ser objetivos a serem aprimorados e utilizados com maior frequência dentro do projeto, visando melhorias constantes ligadas a qualidade e produtividade de código.

5. Considerações finais

Após analisarmos todos os problemas relacionados a má estruturação de código, torna-se de fácil percepção a importância da utilização das boas práticas relacionada ao *clean code*, visando dessa forma a criação de um código de alta qualidade que proporcione um bom nível de produtividade

Códigos que são criados através do pensamento crítico exigido pelo *clean code*, apresentam uma estrutura de fácil compreensão e interpretação, dessa forma é notório relembrar que esses são conceitos fundamentais para a realização de possíveis manutenções, tanto evolutivas quanto corretivas.

Portanto, a utilização dos conceitos de *clean code* torna-se um fator de suma importância para a melhoria no nível de produtividade em uma empresa ou fábrica, sendo dessa forma indispensável para seu sucesso no atual cenário de desenvolvimento de *software*.

6. Referência

Borges, João Roberto P. **“Clean Code: O que é? Porque usar?”.** Clean Code, [*S. l.*], 2018. Disponível em: https://medium.com/joaorobertopb/1-clean-code-o-que-%C3%A9-porque-usar-1e4f9f4454c6. Acesso em: 26 jun. 2019.

Celestino, André. **“A prática do Clean Code”**. Clean Code, [*S. l.*], 2014. Disponível em: https://www.andrecelestino.com/pratica-clean-code/. Acesso em: 26 jun. 2019.

Celestino, André Luis. **“Desenvolvimento de Software – A Prática do Clean Code. Clean Code”**, [*S. l.*], 2014. Disponível em: https://www.profissionaisti.com.br/2014/04/desenvolvimento-de-software-a-pratica-do-clean-code/. Acesso em: 26 jun. 2019.

Dantas, Cristine. **“Gerência de Configuração de Software”.** Configuração de Software, [*S. l.*], 2009. Disponível em: https://www.devmedia.com.br/gerencia-de-configuracao-de-software/9145. Acesso em: 28 jun. 2019.

Oliveira, Marcio de S. Justo. **“Clean Code: Desenvolvimento com qualidade”**. Clean Code, [*S. l.*], 2014. Disponível em: https://www.devmedia.com.br/clean-code-desenvolvimento-com-qualidade/30717. Acesso em: 25 jun. 2019.

Martin, Robert C. **“Código Limpo”**. [*S. l.*]: Elsevier/alta Books, 2008.

Neitzke, Neri Aldoir. **“Boas práticas de Programação”**. Clean Code, [*S. l.*], 2011. Disponível em: https://www.devmedia.com.br/boas-praticas-de-programacao/21137. Acesso em: 21 jun. 2019.

Pedralho, André. **“Aplicando boas práticas em todo o processo de desenvolvimento”**. Boas práticas no processo de desenvolvimento, [*S. l.*], 2016. Disponível em: https://www.devmedia.com.br/aplicando-boas-praticas-em-todo-o-processo-de-desenvolvimento/34407. Acesso em: 25 jun. 2019.