INTERFACE TECNOLÓGICA

CÓDIGO LIMPO: a importância da refatoração

CLEAN CODE: the importance of refactoring

Caio Augusto Miquelutti Porto – caioporto18@hotmail.com
Faculdade de Tecnologia de Taquaritinga (Fatec) – Taquaritinga – SP – Brasil

Oswaldo Lazaro Mendes – oswaldo.lazaro@fatectq.edu.br
Faculdade de Tecnologia de Taquaritinga (Fatec) – Taquaritinga – SP – Brasil

DOI: 10.31510/infa.v19i1.1373
Data de submissão: 09/03/2022
Data do aceite: 25/05/2022
Data da publicação: 30/06/2022

RESUMO

O ato de escrever código é muitas vezes subestimado, mas é a base que conduz cada movimento, positivo ou negativo, ao seu objetivo final. À medida que o software evolui, ele pode adquirir certos tipos de rigidez, permitindo que o software chegue ao fim de seu ciclo de vida. A chamada arquitetura limpa é uma das soluções propostas para minimizar esse sintoma. Código limpo tornou-se uma filosofia de desenvolvimento cujo objetivo principal é aplicar técnicas simples projetadas para facilitar a escrita e a leitura de código. No processo de limpeza ou escrita de código limpo, existe um processo chamado refatoração, que é um processo de melhoria do código sem criar novos recursos, o que auxilia na transformação do código. Neste trabalho, serão discutidos os conceitos de código limpo e refatoração, por meio de uma pesquisa bibliográfica descritiva, com o objetivo de demonstrar os processos de criação de um código limpo e transformação de um código por meio da refatoração. Como resultado foi possível observar que a criação de um código limpo é benéfico para o sistema por facilitar sua manutenção e agilizar seu desenvolvimento. Assim, pode-se concluir que a refatoração é um processo importante durante todo o processo de desenvolvimento, assim como a atenção a todos os detalhes é uma boa prática do desenvolvedor.

Palavras-chave: Código Limpo. Desenvolvimento. Refatoração.

ABSTRACT

The act of writing code is often overlooked, but it is the foundation that drives every move, positive or negative, to its ultimate goal. As software evolves, it can acquire certain types of rigidity, allowing the software to reach the end of its life cycle. The so-called clean architecture is one of the solutions proposed to minimize this symptom. Clean code has become a development philosophy whose primary goal is to apply simple techniques designed to make code easier to write and read. In the process of cleaning or writing clean code, there is a process called refactoring, which is a process of improving the code without creating new features, which helps in transforming the code. In this work, the concepts of clean code and refactoring will be discussed, through a descriptive bibliographic research, with the objective of demonstrating the processes of creating a clean code and transforming a code through

refactoring. As a result, it was possible to observe that the creation of a clean code is beneficial for the system because it makes it easier the maintenance and speeding up its development. Thus, it can be concluded that refactoring is an important process throughout the development process, just as attention to every detail is a good practice.

Keywords: Clean Code. Development. Refactoring.

1 INTRODUÇÃO

As tecnologias estão sempre em evolução, na atualidade as tecnologias digitais exigem constantes atualizações devido ao seu rápido desenvolvimento. A partir disso há a necessidade da criação de um conceito de desenvolvimento Ágil. As tecnologias ágeis auxiliam no desenvolvimento de novas tecnologias de maneira mais acelerada, porém, para que seja possível que os códigos sejam atualizados vários programadores trabalharão com o mesmo código, para que este código seja entendido por todos e para que o desenvolvimento continue acontecendo de forma rápida é necessário que o código seja entendido por todos os que trabalharem com ele, assim surge o conceito de código limpo.

Martin (2009, p. 13) defende que "Se quiser ser rápido, se quiser acabar logo, se quiser que seu código seja de fácil escrita, torne-o de fácil leitura."

O Conceito de código limpo auxilia na manutenção da funcionalidade do código, tornando possível e mais fácil a sua atualização conforme a necessidade.

> "A lógica deve ser direta 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" (STROUSTRUP apud MARTIN, 2009, p.7)

Para transformar um código sujo em um código limpo usa-se uma técnica chamada refatoração.

> "Surgida na década de 80, refatoração é um processo de melhoria de código sem, necessariamente, envolver a criação de novas *features*, para transformar um código mal feito ou bagunçado em código limpo e com design simples, melhorando sua legibilidade e eficiência." (CAVALCANTE, 2021. On-line)

Este trabalho tem como objetivo apresentar os conceitos de código limpo e refatoração, apontando a importância da refatoração para a escrita de um código limpo e também para realizar a limpeza transformando um código sujo em um código limpo.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Código Limpo

O Conceito de Código Limpo, apresentado por Robert Martin (2009) pode ser descrito como uma cultura de desenvolvimento que aplica técnicas simples para auxiliar na escrita e leitura de um código.

Para Martin (2009, p. 2) "O Código é a linguagem na qual expressamos nossos requisitos." Sendo assim, o código é um conjunto de instruções que define uma sequência de ações. O programador é responsável por escrever estas instruções que serão convertidas para a linguagem da máquina com o uso de um compilador. Um programador deve ser capaz de escrever e ler os códigos, para que possa compreender o que será realizado pelo programa quando o código for colocado em execução.

Zanette (2017, on-line) diz que "caso o software não esteja com um código limpo, o desenvolvimento e manutenção se tornarão cada vez mais complicados. Até chegar no ponto em que será mais simples começar o software do zero do que prosseguir com um código ruim". Ou seja, manter o código limpo é uma prática que facilita o processo de desenvolvimento e manutenção de um software.

Ao pensar em escrever um código limpo é necessário lembrar de algumas boas práticas. HORN (2021, on-line) defende que como os desenvolvedores de software devem também realizar manutenção dos sistemas será necessário que se faça a leitura e entendimento dos códigos desenvolvidos por outras pessoas, sendo assim, essa atividade se torna mais fácil de ser executada quando o trabalho é feito com base em boas práticas. Como boas práticas para criar um código limpo é possível ressaltar que deve ser simples e claro, para facilitar o entendimento por outros desenvolvedores. É necessário também que o código seja submetido e passe em todos os testes. Outras práticas importantes são, evitar duplicação e manter o código objetivo, curto e de fácil manutenção.

> Se todos deixássemos nosso código mais limpo do que quando o começamos, ele simplesmente não degradaria. A limpeza não precisa ser algo grande. Troque o nome de uma variável por um melhor, dividir uma função que esteja um pouco grande, eliminar um pouco de repetição de código, reduzir uma instrução *if* aninhada. (MARTIN, 2009 p. 14)

Ao elaborar o código de um software os desenvolvedores não devem ficar satisfeitos apenas com um código que funcione, é necessário considerar que o sistema deverá ser mantido, o código deverá ser alterado e outros desenvolvedores trabalharão com o mesmo código. É

necessário que as alterações e limpezas sejam feitas progressivamente ao longo de todo o projeto, para que não haja medo de continuar mantendo o código limpo. Para garantir que as alterações não trouxeram *bugs* para o sistema é necessário realizar testes com frequência. Além de ajudar nas mudanças, os testes fazem parte da expressividade, simplicidade e flexibilidade do código. Para que ele seja entendido por outros desenvolvedores além do autor, o teste pode mostrar a maneira correta de utilizar um elemento. Quando os testes respondem bem a simplicidade do código é possível concluir que não é necessário deixar o código mais complexo.

HORN (2021, on-line) estabelece 7 principais regras para o código limpo.

a) Utilizar nomenclatura clara e intuitiva

O nome de uma variável, função ou classe deve responder a todas as grandes questões. Ele deve lhe dizer porque existe, o que faz e como é usado. (MARTIN, 2009, p. 18)

b) Seguir os padrões utilizados no código

Ao definir os padrões de um código eles devem ser mantidos, assim como alinhamentos e padrão de espaçamento.

c) Manter os dados de configuração separados do código fonte

“Os dados de configuração, como strings de conexão com o banco de dados, devem ser adicionados em um arquivo separado do código fonte” (HORN, 2021, on-line)

d) Evitar repetições excessivas

“Cada parte de conhecimento do sistema deve possuir apenas uma representação. Desta forma, evitando a ambiguidade do código. Em outras palavras, não deve existir duas partes do programa que desempenham a mesma função” (ZANETTE, 2017, on-line)

e) Ter cuidado com o uso de comentários no código

“Nada pode ser tão útil quanto um comentário bem colocado. Nada consegue amontoar um módulo mais do que comentários dogmáticos e supérfluos. Nada pode ser tão prejudicial quanto um velho comentário mal feito que dissemina mentiras e informações incorretas.” (MARTIN, 2009, p. 53)

f) Realizar o tratamento de erros

Eventualmente alguns erros podem ocorrer, assim, o código deve estar preparado para lidar com eles, apresentando mensagens adequadas e esclarecedoras.

g) Executar testes limpos

Um código limpo deve ser validado com testes limpos. Os testes devem ser realizados em blocos, com clareza, simplicidade e consistência.

É indispensável ressaltar que um código ruim pode funcionar, mas se ele for mal planejado e não for limpo pode arruinar o projeto. Deve-se programar pensando que outras pessoas devem entender o código, ao escrever um código limpo fica mais fácil para que outros programadores o entenda, portanto é importante considerar o trabalho em equipe e respeitar os desenvolvedores que trabalharão com o mesmo código futuramente.

Robert Martin (2009) também destaca a importância de se preocupar e dedicar tempo e atenção a cada elemento, dessa forma o processo de refatoração também é facilitado.

2.2 Refatoração

Refatoração é o nome dado à técnica de transformar um código sujo em limpo, ou até mesmo o processo de melhorar o código, tornando-o mais claro, simples e eficiente.

> Refatoração é a prática de fazer uma série de pequenas transformações que melhoram a estrutura do sistema, sem afetar seu comportamento. Uma transformação sozinha é insignificante, mas, juntas, elas se combinam em transformações significativas do projeto e da arquitetura do sistema. (MARTIN; MARTIN, p. 46)

A Refatoração é um processo de melhoria do código que não cria novas funcionalidades, tem como objetivo organizar o código, deixando o design mais simples. “A refatoração altera os programas em pequenos passos. Se você cometer um erro, é fácil encontrar a falha” (FOWLER, 2000, p. 28).

Ao desenvolver projetos é normal que o código se deteriore, a refatoração também pode auxiliar neste processo evitando a deterioração, auxilia também no entendimento do código, facilitando a manutenção e a eliminação de bugs.

O código tende a se deteriorar. À medida que adicionamos recursos e lidamos com erros, a estrutura do código degrada. Se não for controlada, essa degradação levará a uma bagunça complicada e impossível de organizar. (MARTIN; MARTIN, p. 45)

Para tornar um código limpo a refatoração é um processo importante. Com este processo é possível realizar mudanças pequenas sem prejudicar a funcionalidade do sistema.

Segundo Fowler (apud MARTIN; MARTIN, p. 63) a refatoração é o processo de alterar um sistema de software de tal maneira que não mude o comportamento externo do código, embora melhore sua estrutura interna.

Para refatorar é possível levar em conta as regras para um código limpo, fazendo as alterações necessárias para excluir duplicações, melhorar os nomes, rever comentários e os padrões.

Durante o processo de desenvolvimento, em alguns momentos pode-se identificar a necessidade de refatorar, como quando há muita repetição, para remover classes desnecessárias, quando algo está sendo implementado, neste caso há a necessidade de verificar o código para avaliar a existência de *bugs*. Ao tentar se tornar um profissional melhor, aprender a escrever códigos limpos e refatorar sempre que possível e necessário faz parte do processo.

> Ao aprender a evoluir seus projetos, você pode tornar-se um projetista de software melhor e reduzir a quantidade de trabalho que projeta em excesso ou escassez. TDD e refatoração contínua são as práticas-chave para projeto evolutivo. Introduza gradualmente refatorações direcionadas para padrões no seu conhecimento de como refatorar e encontrar-se-á mais bem equipado para evoluir grandes projetos. (KERIEVSKY, 2008, p. 33)

Segundo Kent Beck (2004) existem 4 regras para a criação de um projeto simples. Estas regras são: Efetuar todos os testes, não duplicar códigos, expressividade e reduzir o número de classes e métodos. As três últimas regras estão diretamente relacionadas a refatoração, são processos feitos durante o processo de refatoração.

Todo desenvolvedor entende a importância da realização de testes, ao longo do processo de escrita de um código limpo ou de limpeza de um código é necessário sempre refatorar, para isso ao longo de todo o processo de refatoração, ou seja, sempre que realizada uma alteração no código, seja buscando evitar a duplicação, para corrigir um *bug* ou para qualquer alteração que seja feita, novos testes devem ser realizados para garantir o funcionamento do código. Por isso é importante ressaltar que os testes também fazem parte do processo de refatoração, pois além da limpeza e clareza do código, é essencial que ele esteja funcionando.

Alguns pontos devem ser observados durante a refatoração, como: A composição de métodos, a organização dos dados e a simplificação de condicionais.

Durante a refatoração uma parte importante do trabalho é combinar os métodos corretamente. Muitas vezes um método longo pode se tornar confuso em relação a sua lógica de execução, o que pode tornar o código difícil de entender e até mesmo de alterar. Existem algumas técnicas que simplificam os métodos, eliminam a duplicação, tornando o código mais fácil de ser compreendido e desenvolvido. Também é importante simplificar as chamadas de métodos.

Quanto a organização dos dados, é importante tomar cuidado com o manuseio de dados ao organizar as classes, podendo-se criar novas classes, e torná-las mais maleáveis e reutilizáveis. A simplificação de condicionais é importante porque as condicionais tendem a terem uma lógica mais complicada, portanto torná-las simples e de fácil entendimento é essencial.

3 PROCEDIMENTOS METODOLÓGICOS

De acordo com Andrade (2006), Cervo et. al. (2010) e Gil (2008), as pesquisas podem ser classificadas de várias formas. Seguindo as classificações indicadas pelos autores, este trabalho foi classificado como uma pesquisa Qualitativa aplicada, desenvolvido por meio de uma pesquisa bibliográfica descritiva.

Neste trabalho, por meio de uma pesquisa qualitativa descritiva, serão descritos os dados encontrados durante a pesquisa para que seja possível estabelecer as relações necessárias entre a refatoração e o código limpo.

De acordo com a finalidade, a pesquisa desenvolvida é classificada como uma pesquisa aplicada, pois está relacionada ao tópico de desenvolvimento de software, e busca apresentar uma maneira de criar códigos de forma limpa, para facilitar sua manutenção e aumentar a produtividade.

A pesquisa foi realizada por meio de livros, revistas, artigos e teses, com uma pesquisa bibliográfica em busca de explicações dos conceitos de código limpo e refatoração, para que seja possível entender e explicar a importâncias do processo da refatoração na criação de um código limpo.

Inicialmente foi realizada uma pesquisa sobre Código limpo, com base no entendimento de código limpo foi realizada a pesquisa sobre refatoração para que fosse possível discutir este processo dentro dos procedimentos para a criação de códigos limpos.

4 RESULTADOS E DISCUSSÃO

Ao escrever um código é possível que ele funcione mesmo estando ilegível ou bagunçado, mas esta atitude pode acarretar vários problemas tanto para o programador quanto para a empresa. A baixa qualidade do código pode apresentar diversos problemas, como a baixa produtividade ou motivação do profissional, assim como a dificuldade de manter o projeto e seus custos, pois acarreta custos maiores para manutenção do programa, demanda mais tempo para realizar as alterações necessárias e para criar funcionalidades.

Assim pode-se dizer que manter uma ética e boas práticas de desenvolvimento, criando códigos limpos auxilia na produtividade da equipe, facilita a manutenção e pode motivar os profissionais.

É importante ressaltar que a refatoração é parte essencial na criação de códigos limpos, tanto para o processo de escrita do código quanto para a manutenção. A refatoração deve ser feita ao longo de todo o projeto, o desenvolvedor deve criar o hábito de sempre melhorar o código que ele estiver trabalhando, para que aqueles que tiverem que trabalhar com o código no futuro possa sempre entender o código com clareza, evitando grandes erros ao realizar atualizações.

Portanto a utilização de nomenclaturas adequadas de variáveis, métodos e classes, assim como comentários pontuais e bem desenvolvidos, são benéficos para o rendimento e desempenho de outros profissionais que farão a leitura do código, auxiliando no entendimento para que se possa realizar alterações ou correções em um código.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foram apresentados conceitos referentes ao código limpo, buscando apresentar aos desenvolvedores algumas formas de melhorar um código. Foi possível identificar que por meio da refatoração é possível realizar melhorias constantes no código, desde o início de sua criação. Portanto a refatoração é um processo que pode ser realizado para melhorar um código muito ruim, mas também é um método usado para realizar pequenas melhorias ao longo do processo de desenvolvimento.

Também foi identificado que com o uso de boas práticas de escrita de códigos é possível gerar valor ao produto e para o cliente, pois facilita e agiliza o processo de desenvolvimento em equipe, bem como a manutenção do código.

Por fim, pode-se também considerar a importância da atenção do desenvolvedor a todos os aspectos do código, porque o desenvolvedor é um escritor e criador, ao tratar o código com atenção se demostra o profissionalismo e dedicação ao trabalho, assim como o respeito a todos os colegas de profissão que poderão trabalhar com um código bem escrito.

REFERÊNCIAS

ANDRADE, Maria M. **Introdução à metodologia do Trabalho Científico**. 7. ed. São Paulo. Editora Atlas S.A. 2006.

BECK, Kent. **Extreme Programming Explained: Embrace Change.** 2. ed. revisada. Addison-Wesley Professional, 2004.

CAVALCANTE, Pablo H. A.; **Entenda o que é refatoração e suas principais técnicas.** GeekHunter. Jan. 2021. Disponível em: <https://blog.geekhunter.com.br/refatoracao/>. Acesso em: 02 Fev. 2022.

CERVO, A.; BERVIAN, P.; SILVA, R. **Metodologia Científica**. 6. ed. São Paulo. Editora Pearson. 2010.

FOWLER, Martin. **Refatoração: Aperfeiçoando o design de códigos existentes.** 1. ed. Porto Alegre: Artmed Editora SA, 2000.

GIL, Antonio C. **Métodos e Técnicas e Pesquisa Social**. 6. ed. São Paulo. Editora Atlas S.A. 2008.

HORN, Michelle. **Clean Code: o que é, porque usar e principais regras!**. Blog BeTrybe. Fev. 2021. Disponível em: <https://blog.betrybe.com/tecnologia/clean-code/>. Acesso em: 02 Fev. 2022.

KERIEVSKY, Joshua. **Refatoração para padrões.** 1. ed.Porto Alegre: Bookman Editora LTDA, 2008.

MARTIN, Robert C. **Código Limpo: Habilidades Práticas do Agile Software.** 1. ed. Atlas Book, 2009.

MARTIN, Robert C; MARTIN, Micah. **Princípios, Padrões e Práticas Ágeis em C#.** 1. ed. Porto Alegre: Bookman, 2011. Kindle.

ZANETTE, Alysson. **Clean Code: Boas práticas para manter seu código limpo!.** BeCode. 2017. Disponível em: <https://becode.com.br/clean-code/> Acesso em: 02 Fev. 2022.