A IMPORTÂNCIA DO CÓDIGO LIMPO NA PERSPECTIVA DOS DESENVOLVEDORES E EMPRESAS DE SOFTWARE

*Joberto Diniz Junior1, Djalma Domingos da Silva2*

*1, 2 FATEC – Faculdade de Tecnologia de São José do Rio Preto*

[*me@jobertodiniz.com*](mailto:me@jobertodiniz.com)*1* [*djalma@fatecriopreto.edu.br*](mailto:djalma@fatecriopreto.edu.br)*2*

***1. Introdução***

Desde os anos noventa uma tarefa não parece ter mudado muito no desenvolvimento de sistemas: a manutenção do código-fonte. Diferentemente do senso comum, programas são lidos mais frequentemente do que eles são escritos [1]. Constantemente lemos código antigo como parte do esforço para criar um novo. Isso se deve, principalmente, ao atraso que o código ruim proporciona [2]. Dois grandes problemas emergem do código ruim: bugs e baixa produtividade dos desenvolvedores. Mas será que precisa ser sempre assim? Será que não existem melhores formas de escrever um código que facilite o entendimento dos desenvolvedores atuais e futuros, que minimize os bugs e aumente a produtividade?

***2. Justificativa***

Do ponto de vista do desenvolvedor, conhecer as técnicas do Código Limpo trará uma mudança de paradigma de como escrever código e melhorar profissionalmente. Da perspectiva da empresa, contratar profissionais que conheçam as técnicas diminuirá os bugs e aumentará a qualidade do código.

***3. Fundamentação Teórica***

Código ruim custa caro e não pode ser negligenciado. De acordo com [2] o código-fonte ruim influencia tanto uma empresa que pode leva-la à falência.

Segundo [2], Código Limpo é o uso disciplinado de uma miríade de pequenas técnicas aplicadas disciplinadamente. Um Código Limpo faz o leitor sorrir, possui elegância, é agradável, e talvez o mais importante, parece que foi escrito por alguém que se importava.

Algumas características mais específicas se destacam, tais como nomes significativos que comuniquem seu real propósito, classes enxutas e métodos pequenos de no máximo 5 linhas.

Outra grande característica são os princípios SOLID de programação orientada a objetos: Princípio da Responsabilidade Única, Principio Aberto-Fechado, Principio da Substituição de Liskov, Principio da Segregação de Interface e Princípio da Inversão de Dependência. Seguindo esses princípios, o índice de manutenibilidade, a testabilidade e compreensão do código aumentam acentuadamente.

Pode-se citar ainda o uso do padrão de projeto Objeto Nulo para evitar os fatídicos erros de *NullPointerException* (Java) e *NullReferenceException* (C#).

***3. Objetivos***

Este estudo tem por objetivo apresentar e colocar à prova por meio de um pequeno experimento a importância do Código Limpo tanto para os desenvolvedores quanto para as empresas de software.

***4. Metodologia***

Nove voluntários participaram do experimento, que consistia na resolução do cálculo do desconto de Instituto Nacional de Seguro Social (INSS), que foram separados em dois grupos: um com técnicas do Código Limpo e outro com código convencional. Os resultados, como índice de manutenibilidade e tempo gasto, foram analisados estatisticamente por meio do teste *t* independente.

***5. Resultados***

Estatisticamente, como mostra a Figura 1, observa-se que a hipótese nula foi rejeitada para a métrica de índice de manutenibilidade, visto que o valor-p (3,6%) é menor que 5%, revelando que se as técnicas do Código Limpo forem seguidas na construção de um código, ele certamente será mais fácil de se modificar.

****

Figura 1 – Valores índice de manutenibilidade.

Diante desse alto índice, o tempo gasto para modificar o Código Limpo foi menor (16,50±2,25 vs 49,00±10,97), o que mostra que o código estava de fácil compreensão e bem estruturado para receber novas funcionalidades.

***6. Conclusão***

Os resultados sugerem que as técnicas do Código Limpo podem aumentar a produtividade dos desenvolvedores, visto que o índice de manutenibilidade e o tempo de manutenção são melhores que de um código convencional. A importância do Código Limpo é notória, e as empresas de software podem investir em estratégias que visem o treinamento de seus desenvolvedores.

***7. Referências***

[1] Beck K., Implementation Patterns, Addison Wesley, 2007

[2] Martin R., Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2009