Maturidade no processo de desenvolvimento de software Qualidade de código contínua

João Carlos Brasileiro Stefenon de Almeida[1]

[1] Universidade do Vale do Rio dos Sinos (UNISINOS)
São Leopoldo – RS – Brazil

jcbrasileiro@hotmail.com

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escritos em português. Artigos em inglês deverão apresentar apenas abstract. Nos dois casos, o autor deve tomar cuidado para que o resumo (e o abstract) não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do artigo.*

Part I

INTRODUÇÃO

Uma das mudanças mais impactantes no processo de desenvolvimento de software das ultimas décadas, foi a concepção e absorção das metodologias ágeis de software, seja SCRUM, Extreme Programming (XP), Test Driven Development (TDD), Lean Software Development, Kanban, etc, todas foram criadas a partir da filosofia AGILE [Agile 2001], surgido a partir do esforço de várias pessoas que lidavam com o processo de software na década de 1990, com o objetivo de definir uma abordagem mais efetiva e eficiente para o desenvolvimento de software. Apesar dos conceitos já existirem a quase 20 anos, a importância e a implementação dessas ideias ainda são muito pouco exploradas no Brasil, segundo Coleman Parkes Research [**?**] ; durante uma pesquisa; entrevistou cerca de 1.770 executivos de tecnologia da informação em 21 países, incluindo 76 brasileiros, mostra que somente 6% das empresas têm utilizado fortemente, "transformando toda a organização para abranger os princípios de agilidade".

A partir dessa nova filosofia, a importância da qualidade no desenvolvimento criou novos pontos de vistas, aumentando significativamente também tanto na concepção, quanto no decorrer das outras fases, incluindo durante o desenvolvimento, e não mais

apenas no final com um simples objetivo de execução. Um das mudanças foi a inclusão e a utilização do conceito de **MPV**, sigla de *Minimum Viable Product*, que significa produto mínimo viável - conceito popularizado por Eric Ries [**?**] - que trouxe a importância de entregar e garantir de modo incremental, funcionalidades importantes para o cliente/negocio, ressaltando não apenas garantir um software útil, mas também visando a maleabilidade do negocio, as possibilidades de mudanças, seja de prioridades de funções, ou mesmo do próprio objetivo fim. A adoção desses pequenos entregáveis , gerou e exigiu; finalmente; uma maturidade no desenvolvimento, criando, por exemplo, a necessidade de artefatos que evidenciassem e garantissem desdo inicio dois grande aspectos: (1) Qualidade e (2) Garantia que cada entregável operasse corretamente.

-

1º e 2º parágrafo: apresentação do tema dentro de um contexto. – Qualidade do código / garantia

3º parágrafo: delimitação do tema, apresentado através do problema de pesquisa do seu TCC. – Código ruim e "Ice cream" problem

4º parágrafo: apresente possíveis respostas para o problema de pesquisa levantado, ou seja, as hipóteses. Aplicação das principios OO e CLEAN CODE. Segregação das testes unitarios e testes de integração.

5º parágrafo: em poucas palavras, fale sobre o objetivo geral do trabalho e também dos específicos. Eles são ingredientes fundamentais para o trabalho.

6º parágrafo: apresente a relevância do seu trabalho acadêmico, identificando a importância dele para a sociedade ou comunidade científica. Isso é o que chamamos de justificativa.

7º parágrafo: descreva, em poucas palavras, qual metodologia foi utilizada. Foi pesquisa bibliográfica ou de campo? Você deve especificar o procedimento de forma concisa.

8º parágrafo: apresente a estrutura do trabalho, ou seja, como ele está dividido em capítulos. Lembre-se de falar, resumidamente, sobre o que se trata cada capitulo.

Part II

REVISÃO BIBLIOGRÁFICA

1. *Principles of Object-oriented programming (OOP)*

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

1.1. Encapsulation

1.2. Abstraction

1.3. Inheritance

1.4. Polymorphism

2. *Clean Code*

Learning to write clean code is hard work. It requires more than just the knowledge of principles and patterns. You must sweat over it. You must practice it yourself, and watch yourself fail. You must watch others practice it and fail. You must see them stumble and retrace their steps. You must see them agonize over decisions and see the price they pay for making those decisions the wrong way.

3. Design Principles

3.1. *Object-Oriented Design Principles (OODP)*

3.2. *General responsibility assignment software principles (GRASP)*

- Controller
- Creator
- Indirection
- Information Expert
- High Cohesion
- Low Coupling
- Polymorphism
- Protected Variations
- Pure Fabrication

3.3. *S.O.L.I.D principles*

- Single responsibility principle
- Open closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency Inversion principle

3.4. *Law of Demeter principles*

3.5. *Don't Repeat Yourself (DRY)*

3.6. *Favor Composition over Inheritance.*

3.7. *Keep it Simple, Stupid. (KISS)*

4. *Test-driven development (TDD)*

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.

The process can be defined as such:

Write a failing unit test Make the unit test pass Refactor

Repeat this process for every feature, as is necessary.

Part III

Trabalhos Relacionados

References

Agile (2001). Manifesto for agile software development. http://agilemanifesto.org/.