josif 2022
14ª JORNADA CIENTÍFICA E TECNOLÓGICA
11º SIMPÓSIO DE PÓS-GRADUAÇÃO
IFSULDEMINAS
REALIZAÇÃO INSTITUTO FEDERAL Sul de Minas Gerais
APOIO FAPEMIG CNPq

ISSN: 2319-0124

UM ESTUDO DE ARQUITETURAS, DESIGN PATTERNS E BOAS PRÁTICAS DE CODIFICAÇÃO: vantagens e desvantagens

Fabiano Reis FALEIROS[1]; Hiran Nonato Macedo FERREIRA[2]

RESUMO

Com o aumento crescente dos acessos de pessoas do mundo todo à internet, especialmente na última década, e o consequente avanço do número de usuários, surge a necessidade de desenvolver aplicações que atendam a demandas cada vez maiores. Expandindo as exigências e manutenções constantes de *software*, que devem acontecer de forma rápida e com desempenho satisfatório para o usuário. Este trabalho tem como propósito apresentar um relato de pesquisa sobre formas de se construir *softwares* escaláveis, com longo prazo de duração e facilidade de manutenção entre equipes, promovendo a redução do tempo gasto no desenvolvimento e facilitando a criação de novas funcionalidades. É apresentado um conjunto de arquiteturas, *design patterns* e boas práticas de desenvolvimento de *software*. Espera-se atingir por meio do uso combinado das técnicas apresentadas uma proposta de oferecer melhores soluções na construção de *softwares*.

Palavras-chave:
Arquitetura de software, engenharia de software, escalabilidade.

1. INTRODUÇÃO

O número de pessoas que acessam a internet vem crescendo a cada dia, especialmente na última década. Segundo pesquisas da Cisco (2018), em 2018 cerca de 3,9 bilhões de pessoas possuíam acesso à rede, com uma previsão de aumento para 5,3 bilhões em 2023, ou seja, uma diferença de aproximadamente 36% em um período de 5 anos. Com o crescente número de usuários, surge a necessidade de desenvolver aplicações que atendam a demandas cada vez maiores. Sendo exigidas melhorias e manutenções constantes de *software*, que devem acontecer de forma rápida e com desempenho satisfatório para o usuário.

Sommerville (2019) define que um bom *software* deve atender a algumas características básicas, como manutenibilidade, eficiência, confiança e aceitabilidade. Acrescentando ainda, que o desenvolvimento não acaba ao se concluir o sistema, pois ele ainda necessita de manutenções corretivas e evolutivas. Para Porto (2009), um sistema não escalável pode trazer vários problemas, pois as empresas podem não conseguir suprir adequadamente as necessidades de seus clientes, considerando que os usuários tendem a não utilizar um sistema que não os atende.

Tendo em vista este contexto, a proposta deste relato de pesquisa é apresentar formas de se construir *softwares* escaláveis, com longo prazo de duração e facilidade de manutenção entre equipes promovendo a redução do tempo gasto no desenvolvimento e facilitando a criação de novas funcionalidades. Expandindo a possibilidade de empresas oferecerem um serviço de alta qualidade aos seus clientes, ampliando lucros e adequando novos usuários.

Dessa forma, serão selecionadas arquiteturas, *design patterns* ou padrões de

[1]Aluno, IFSULDEMINAS – *Campus* Passos. E-mail: fabiano.reis@alunos.ifsuldeminas.edu.br.
[2]Orientador, IFSULDEMINAS – *Campus* Passos. E-mail: hiran.ferreira@ifsuldeminas.edu.br.

desenvolvimento de *software* que agregam valor entre si. Após, vão ser apresentadas suas vantagens e desvantagens, assim como os melhores cenários de uso.

2. FUNDAMENTAÇÃO TEÓRICA

A escalabilidade de *softwares* pode ser tratada como um conceito difícil de se definir, dado sua enorme amplitude e por se tratar de um atributo não-funcional, ou seja, denota um atributo de qualidade (CAVALCANTI, 2015).

Porto (2009) caracteriza a escalabilidade como uma propriedade que traz a habilidade de "crescer" ou "diminuir" ao sistema de acordo com a necessidade imposta. Ou seja, se o sistema tiver uma variância em sua carga de trabalho, seu desempenho não deve ser prejudicado, ou pelo menos ter alguma consequência reduzida. O conceito de manutenibilidade tem relação direta com a "capacidade de manutenção" de um *software* e pode ser definida pelas seguintes características: modularidade, reusabilidade, analisabilidade, modificabilidade e testabilidade. (WAZLAWICK, 2013)

A arquitetura de *software* pode ser definida como um "projeto em mais alto nível", na qual podemos considerar que seu maior foco está em unidades de maior tamanho, ou seja, pacotes, componentes, camadas, serviços e subsistemas, deixando de dar foco principal na organização de interfaces e classes individuais (VALENTE, 2020). Valente (2020) complementa ainda que a arquitetura de *software* pode também ser definida como as "decisões mais importantes de um sistema". Decisões que possuem tamanha importância que após terem sido tomadas, dificilmente podem ser revertidas, ou seja, a arquitetura trata-se não apenas de um conjunto de módulos, mas de todo um conjunto de decisões.

Domain Driven Design (DDD) ou projeto orientado a domínio se trata de uma abordagem de desenvolvimento de *software* que define que cada aplicação que se deseja construir deve pertencer a um domínio. O domínio pode ser definido pelo assunto que é tratado na aplicação, chamada de lógica de negócio, ou seja, todo o conhecimento ou atividade que resulte em lógica na aplicação define o domínio. Ademais o domínio deve ser o "coração do *software*", em uma arquitetura de camadas que é composta por interface com usuário, aplicação, domínio e infraestrutura. (EVANS, 2016).

Na engenharia de *software*, padrões de projeto ou *design patterns* podem ser denominados como uma solução geral para um problema que ocorre com alguma frequência em um determinado projeto ou contexto de um projeto de *software*. A criação de padrões é importante para capitalizar experiências e deixar de repetir erros que possuem soluções conhecidas (WAZLAWICK, 2013). Dependency Injection ou injeção de dependências se trata de um padrão de desenvolvimento de *software* na qual se têm um objeto chamado *assembler* que possui por função injetar instâncias de variáveis sempre que um novo objeto é criado. Repository e Service ou Repositórios e Serviços se trata de um padrão que, segundo Jones (2019), divide a camada de negócios da aplicação em duas camadas distintas e dependem de a Injeção de Dependência funcionar corretamente.

Não há uma definição tão clara a respeito do código limpo, Almeida e Miranda (2010) citam que a escrita de códigos pode ser comparada com a arte, pois não é possível definir parâmetros lógicos e mensuráveis que definam a diferença de qualidade entre códigos-fonte. Segundo Martin (2011) códigos-fonte não são apenas palavras escritas em algum editor de código, eles necessitam de organização de forma que programadores consigam ler e entender com facilidade.

O código limpo se trata de uma filosofia com objetivo de tornar as tarefas mais simples, fáceis e intuitivas, sempre deixando clara sua intenção. Para Almeida e Miranda (2010) e Martin (2011) um código limpo se trata de um estilo de programação próximo a três valores: Expressividade, Simplicidade e Flexibilidade.

A partir da definição de um código limpo, um próximo passo é entender como desenvolver este na prática. Segundo Martin (2011) é importante reconhecer que nem sempre na primeira tentativa se obtém sucesso na escrita de um método ou classe da melhor forma possível. É

necessário todo um cuidado e tempo dedicados, desde algum nome de variável a uma construção de hierarquia de classes.

Almeida e Miranda (2010) destacam ainda que um código limpo não pode causar receio aos programadores no momento que são feitas adições de código. Deste modo surge a necessidade de se utilizar de Testes Unitários como uma fundação para esta metodologia, dando a segurança que todas novas alterações não trarão impacto no que já foi desenvolvido.

3. MATERIAL E MÉTODOS

Com objetivo de colocar em prática os conhecimentos explanados no presente trabalho, será desenvolvido uma plataforma de avaliação e orientação para medidas terapêuticas relacionadas à saúde mental de pacientes. Nesta plataforma serão disponibilizadas uma série de perguntas, que, de acordo com sua resposta, irá disponibilizar várias recomendações para prevenir ou confortar pessoas com problemas mentais.

Neste passo será desenvolvido todo o código fonte da aplicação, pelo lado do servidor será construída uma API REST utilizando a linguagem JavaScript. Para a aplicação cliente, as interfaces da mesma serão desenvolvidas em conjunto com a biblioteca React também fazendo o uso da linguagem JavaScript. Ao final do desenvolvimento a aplicação será disponibilizada em servidores *on-line* utilizando de práticas de Continuous Integration/Continuous Delivery ou Integração Contínua/Entrega contínua, técnicas que garantem entregas e manutenções eficientes de código.

Ao fim do desenvolvimento da aplicação terá início a validação do sistema, neste momento serão avaliadas questões de usabilidade por parte de usuários, e temas a respeito da escalabilidade e manutenibilidade do *software* criado, ponto chave da pesquisa. Os usuários do sistema irão apresentar opções de melhoria visual ou experiência de uso por meio de pesquisas na fase de teste da aplicação.

4. RESULTADOS ESPERADOS E DISCUSSÕES

Espera-se atingir por meio do uso combinado de técnicas de desenvolvimento, sendo elas, arquiteturas, *design patterns* e boas práticas condizentes entre si uma proposta de oferecer melhores soluções na construção de *softwares*. Com essa proposta um novo passo será dado em direção a evolução de aplicações cada vez mais escaláveis e com facilidade de manutenção entre times de desenvolvimento.

Busca-se contribuir em uma nova abordagem na construção de um *software* combinando técnicas e conhecimentos que também podem contribuir para trabalhos futuros. Ademais será oferecida uma plataforma para auxílio na terapia mental, facilitando o trabalho de profissionais da saúde.

5. CONCLUSÕES

Este trabalho apresentou uma proposta de estudo sobre arquiteturas, *design patterns* e boas práticas de desenvolvimento de *software* aplicadas em um estudo de caso. Até o momento as etapas de prototipação e levantamento de requisitos da aplicação proposta foram concluídas. A fase de desenvolvimento está em progresso e assim que finalizada serão feitas as validações e pesquisas com usuários.

Espera-se que após o desenvolvimento os autores contribuam para o entendimento na relação teórico prática sobre a construção de *softwares* escaláveis, com longo prazo de duração e facilidade de manutenção entre equipes. Desta forma alcançando a possibilidade de oferecer serviços com qualidade aos usuários, além de escalabilidade e velocidade na entrega de novas funcionalidades.

REFERÊNCIAS

ALMEIDA, Lucianna Thomaz; DE MIRANDA, João Machini. Código Limpo e seu Mapeamento para Métricas de Código Fonte. Universidade de São Paulo. Disponível em: http://www.ime.usp.br/~ cef/mac499-10/monografias/luciannajoao/arquivos/monografia.pdf, Maio, 2010.

CAVALCANTI, Paulo de Lima. MAPEAMENTO SISTEMÁTICO SOBRE ESCALABILIDADE DO I* (ISTAR). 2015. 150 f. Dissertação (Mestrado) - Curso de Pós-Graduação em Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2015.

CISCO. Cisco Annual Internet Report (2018–2023) White Paper. 2018. Disponível em: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. Acesso em: 19 maio 2021.

EVANS, Eric. Domain-Driven Design: Atacando as complexidades no coração do software. Rio de Janeiro: Alta Book, 2016. 528 p.

JONES, Matthew. The Repository-Service Pattern with DI and ASP.NET 5.0. 2019. Disponível em: https://exceptionnotfound.net/the-repository-service-pattern-with-dependency-injection-and-asp-net-core/. Acesso em: 08 jun. 2021.

MARTIN, Robert C.; MARTIN, Micah. Agile Principles, Patterns, and Practices in C#. Londres: Pearson, 2011. 691 p.

PORTO, Ivens Oliveira. Padrões e Diretrizes Arquiteturais para Escalabilidade de Sistemas. 2009. 161 f. Dissertação (Mestrado) - Curso de Pós-Graduação em Ciência da Computação, Universidade Federal de Uberlândia, Uberlândia, 2009.

SOMMERVILLE, Ian. Engenharia de Software. 10. ed. São Paulo: Pearson, 2019. 768 p.

VALENTE, Marco Tulio. Engenharia de Software Moderna. [S. I]: Moderna, 2020. 408 p.

WAZLAWICK, Raul Sidnei. Engenharia de Software Conceitos e Práticas. São Paulo: Elsevier Editora Ltda., 2013. 363 p.