UNIVERSIDADE FEEVALE

LEONARDO MOISÉS LEAL

IMPACTO DA DEFINIÇÃO DO NÍVEL DE GRANULARIDADE EM UM CÓDIGO COM ARQUITETURA DE MICROSERVIÇOS

(Título Provisório)

Anteprojeto de Trabalho de Conclusão

Novo Hamburgo

2022

LEONARDO MOISÉS LEAL

IMPACTO DA DEFINIÇÃO DO NÍVEL DE GRANULARIDADE EM UM CÓDIGO COM ARQUITETURA DE MICROSERVIÇOS

(Título Provisório)

Anteprojeto de Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientadora: Profa. Dra. Adriana Neves dos Reis

Novo Hamburgo

2022

RESUMO

Com o avanço tecnológico diversas novas ideias aparecem a cada momento visando melhorar o desenvolvimento de software. Exemplos como linguagens de programação, novos paradigmas, modelos de processos, designs de código e conceitos de arquitetura de software surgem como alternativas a serem aderidas. Com isso apareceram a arquitetura de *microservices* e o *domain-driven design (DDD)* e são cada vez mais discutidos dentro da comunidade de desenvolvimento de software. Sendo *microservices* uma arquitetura de software e DDD uma proposta de modelagem de código, é possível encontrar contextos em que os dois são utilizados em conjunto. Porém, muitas vezes não se avalia o impacto na utilização de cada uma dessas tecnologias dentro de uma aplicação, tornando a decisão de qual o nível ideal da granularidade do código algo não definido. Portanto, este trabalho tem como objetivo principal elaborar um modelo de definição de granularidade de código com o propósito de auxiliar na tomada de decisão, focado no desenvolvimento em uma aplicação com arquitetura de microserviços e DDD. Como metodologia de estudo será empregada a Engenharia de Software Baseada em Evidências (ESBE) para a avaliação da literatura encontrada e, dada as evidências, serão validadas e exibidas com a construção de códigos e perante testes.

Palavras-chave: Granularidade de código. arquitetura de microserviços. *domain-driven design*.

SUMÁRIO

1. MOTIVAÇÃO

Com o avanço no desenvolvimento de software, mudanças significativas ocorrem e com elas novas tecnologias são construídas. Com isso, as empresas de desenvolvimento buscam migrar os seus produtos e serviços utilizando novas tecnologias como, por exemplo, novas linguagens de programação, novos paradigmas, modelos de processos com novas perspectivas, designs de código e conceitos de arquitetura de software.

Arquitetura de software trata-se de uma representação que auxilia na compreensão de como um sistema irá se comportar (CMU, 2015), servindo como modelo para o sistema e projeto que está sendo construído. Cada arquitetura possui uma diretriz que deve ser seguida pela equipe de desenvolvimento. Neste sentido, novas propostas surgem e nem sempre é fácil tomar a decisão de qual estratégia seguir, pois existem vários padrões que podem ser adotados, tornando difícil a decisão de qual se adequa melhor ao propósito da aplicação.

A arquitetura monolítica é a mais difundida no mercado devido a sua popularidade (AMARAL; CARVALHO, 2017). A principal característica desta arquitetura é conter uma única aplicação responsável por todas as tarefas do sistema e possuir independência de outras aplicações (BECKER, 2019). Isso torna o sistema que utiliza essa arquitetura facilmente implementável pois contém apenas um projeto e um ponto de acesso aos dados. Por outro lado acaba criando códigos extensos, à medida que a aplicação cresce, gerando também apenas um único ponto de falha e comprometendo o acesso a toda a aplicação.

Uma outra arquitetura chamada de *Microservices*, ou microserviços, surgiu durante uma conferência de arquitetos de software (ICSAE) em maio de 2011 e vem sendo bastante difundida desde então na comunidade de arquitetos, engenheiros de softwares e desenvolvedores. microserviços são partes, específicas e independentes, de uma aplicação maior e possuem responsabilidade única (AMARAL; CARVALHO, 2017).

Diferentemente da monolítica, ela possui um estilo de arquitetura em que os aplicativos compreendem um conjunto de serviços *back-end* pequenos. Implantados de forma independente, cada um é executado em seu próprio processo ou servidor dedicado e normalmente interage por meio de *applications programming interface* (APIs), HTTP ou mensagens assíncronas (FOWLER; LEWIS, 2014). Sendo assim, a replicação de microserviços é mais eficiente do que a de aplicações monolíticas, pois seu escalonamento não necessita que a aplicação inteira seja replicada.

O uso da arquitetura de microserviços precisa ser bem avaliada no início do projeto, pois não é a solução para todos os cenários. As principais vantagens do uso de microserviços

estão relacionadas com a heterogeneidade tecnológica, resiliência, escalabilidade e facilidade de implantação. Entretanto, as desvantagens ocorrem em torno da complexidade de desenvolvimento, chamadas remotas e gerenciamento de múltiplos bancos de dados e transações (MOREIRA; BEDER, 2015). Apesar da produtividade inicial para uma aplicação monolítica ser maior que a de microserviços, no decorrer do desenvolvimento a produtividade para uma aplicação monolítica cai quase que 90%, enquanto a de microserviços permanece variando entre 5% e 10% a menos (AMARAL; CARVALHO, 2017).

Mesmo possuindo uma boa aceitação e do crescimento na utilização da arquitetura de microserviços (MSA) ainda é um desafio para a comunidade chegar a um consenso do tamanho ideal de cada serviço. Os microsserviços podem ser definidos com vários níveis de capacidade, e o tamanho dessa funcionalidade é normalmente referido como sua granularidade, ou seja, a complexidade funcional codificada em um serviço ou número de casos de uso implementados por um microsserviço (NEWMAN, 2021). Alcançar um nível ideal de granularidade é, portanto, de grande interesse para encontrar alternativas que o tornem menos complexos e que tenham baixa latência ou número de transações.

É comum que os desenvolvedores usem a própria funcionalidade para definir o escopo que determina o tamanho de um microsserviço. Porém, outra proposta tem aparecido quando abordado o desenvolvimento de software conhecido como *Domain-Driven Design* (DDD). O DDD é uma abordagem para softwares complexos onde busca concentrar-se no domínio principal, explorando modelos em uma colaboração criativa entre os profissionais de domínio e os profissionais de software. Formando uma linguagem onipresente dentro de um contexto limitado explicitamente (EVANS, 2003).

Em outras palavras o DDD é uma abordagem que agrupa técnicas e conceitos onde o foco está no domínio e na lógica para criar um *Domain Model* (modelo de domínio). Essa estratégia entrega vantagens como alinhamento de código com o negócio, favorece a reutilização, busca o mínimo de acoplamento e independe de tecnologia.

Dentro dos conceitos que compõem o DDD estão os *bounded context* (contexto delimitado ou BC) que visa facilitar e dar coerência no desenvolvimento de entidades para que possam ter significados e responsabilidades diferentes dependendo do contexto no qual estão inseridas. BCs são uma parte designada do software em que determinados termos, definições e regras se aplicam de forma consistente (EVANS, 2019). Com isso surgiram algumas propostas de criar microserviços com a modelagem DDD e seus BCs.

Diferentes BCs podem ajudar a mapear e auxiliar na utilização de microserviços (EVANS, 2016). O limite modular estabelecido por um BC no DDD torna microserviços uma

excelente alternativa (NEWMAN, 2021). Porém, a utilização desse método não consegue definir o melhor nível de granularidade, e o impacto dessa decisão ainda não está bem claro, deixando essa lacuna aberta dentro da comunidade de desenvolvedores.

2. OBJETIVOS

Objetivo geral

O objetivo deste trabalho é elaborar, utilizando uma abordagem de arquitetura de microserviços e *domain-driven design*, um modelo de definição de granularidade de código, com o propósito de auxiliar na tomada de decisão com as práticas que podem ser adotadas. A definição do modelo será baseada nas evidências encontradas dentro da comunidade de software através da bibliografia e os resultados obtidos em experimentos realizados com decodificação.

Objetivos específicos

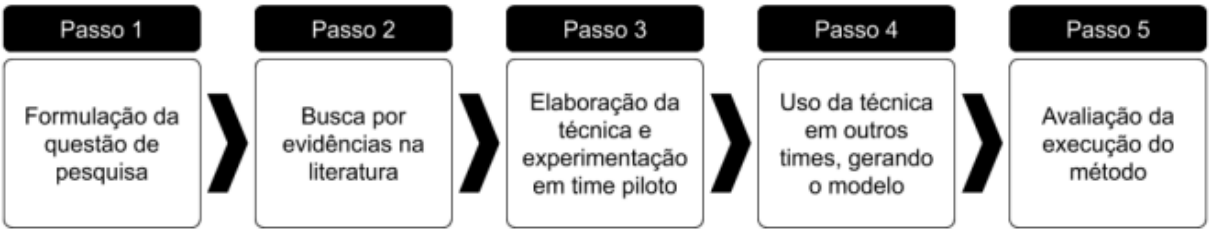
- Pesquisar trabalhos correlacionados entre microserviços e DDD;
- Mapear os princípios de arquitetura de software relacionados à granularidade de código;
- Construir um mapa mental com o que foi encontrado de relevante;
- Realizar experimentos em código a partir do mapeamento;
- Descrever os resultados encontrados através do código implementado;
- Propor um modelo consolidando recomendações para definição de granularidade em código.

3. METODOLOGIA

O objetivo deste trabalho possui um caráter exploratório, buscando evidências que possam comprovar ou não as questões da pesquisa propostas pelo estudo. Para isso será utilizada a Engenharia de Software Baseada em Evidências (ESBE). Essa abordagem tem como objetivo prover meios com os quais a melhor evidência da investigação pode ser integrada com a experiência prática e os valores humanos no processo de tomada de decisão a respeito da Engenharia de Software (DYBA; KITCHENHAM; JORGENSEN, 2005).

A ESBE dispõe de um mecanismo para suportar e melhorar as decisões relacionadas à adoção de tecnologias, fornecendo um objetivo comum aos pesquisadores, garantindo que a pesquisa esteja relacionada com as necessidades do mercado (KITCHENHAM; DYBA; JORGENSEN, 2004). Essa metodologia é dividida em cinco passos: 1) formular as questões que devem ser investigada pela pesquisa; 2) buscar evidências que respondem essas questões; 3) criticar a evidência em relação a sua validade, impacto e aplicabilidade; 4) aplicar as evidências no contexto do estudo; 5) avaliar a efetividade e eficiência do método (KITCHENHAM; DYBA; JORGENSEN, 2004).

Figura 1 – Execução da ESBE no presente trabalho



Fonte: Laux (2019, p. 20)

A ESBE e os seguintes procedimentos para a pesquisa serão adotados: Realização de uma revisão e mapeamento da bibliografia, na busca por evidências teóricas sobre o impacto da definição de granularidade de código em engenharia de software. As evidências serão oriundas da pesquisa realizada na literatura para que seja aplicada em um contexto prático com o desenvolvimento em código. Após isso essas evidências serão validadas para que posteriormente sejam efetivamente demonstrados os resultados.

4. CRONOGRAMA

Trabalho de Conclusão I

Etapas	Março	Abril	Maio	Junho
Escrita anteprojeto	X			
Revisão anteprojeto	X	X		
Revisão da literatura: Microserviços		X	X	
Revisão da literatura: DDD		X	X	
Revisão da literatura: Granularidade utilizando os dois modelos		X	X	
Desenvolvimento de código dada as evidências			X	X
Escrita TCC 1		X	X	X
Revisão TCC 1		X	X	X
Entrega TCC 1				X

Trabalho de Conclusão II

Etapas	Agosto	Setembro	Outubro	Novembro
Desenvolvimento de código dada as evidências	X	X	X	
Validação dos resultados do desenvolvimento	X	X	X	
Descrição da percepção dada a validação		X	X	X
Escrita TCC II	X	X	X	X
Revisão TCC II	X	X	X	X
Entrega TCC II				X

BIBLIOGRAFIA

AMARAL, Odravison. CARVALHO, Marcus. **Arquitetura de Micro Serviços: uma Comparação com Sistemas Monolíticos**. 2017. Universidade Federal da Paraíba (UFPB), Rio Tinto, PB, 2017.

BECKER, Alex Malmann. **O Impacto do Uso de Micro Serviços na Evolução de uma Linha de Produção de Software**. 2019. Universidade Federal de São Carlos (UFSCar), São Carlos, SP, 2019.

CMU - SOFTWARE ENGINEERING INSTITUTE. **Software Architecture**. 2015. Disponível em: <https://www.sei.cmu.edu/our-work/software-architecture/>. Acesso em: 31 mar. 2022.

DYBA, Tore. KITCHENHAM, Barbara A. JORGENSEN, Magne. **Evidence-based software engineering for practitioners**. IEEE Software, v. 22, n. 1, p. 58-65, 2005.

EVANS, Eric. **DDD and Microservices: At Last, Some Boundaries**. 2016. Disponível em: <https://www.infoq.com/presentations/ddd-microservices-2016/>. Acesso em 31 mar. 2022.

EVANS, Eric. **Defining Bounded Contexts – Eric Evans at DDD Europa**. 2019. Disponível em: <https://www.infoq.com/news/2019/06/bounded-context-eric-evans/>. Acesso em 31 mar. 2022.

EVANS, Eric. **Domain-Driven Design: Tackling Complecity in the Heart of Software**. 2003. Addison-Wesley Professional, 1 ed. 2003.

FOWLER, Martin. LEWIS, James. **Microservices a definition of this new architectural term**. 2014. Disponível em <https://martinfowler.com/articles/microservices.html>. Acesso em: 31 mar. 2022.

KITCHENHAM, Barbara A. DYBA, Tore. JORGENSEN, Magne. **Evidence-based software engineering**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 26., 2004, Edinburgh. Anais... Washington: IEEE Computer Society, 2004.

LAUX, Marina Letícia. **Técnica de Apoio à Construção do Modelo de Maturidade do Time Scrum**. 2019. Universidade Feevale, Novo Hamburgo, RS, 2019.

MOREIRA, Pedro Felipe Marques. BEDER, Delano Medeiros. **Desenvolvimento de Aplicações e Micro Serviços: Um estudo de caso**. 2015. Revista T.I.S. São Carlos, v. 4, n. 3, p. 209-215, set-dez 2015.

NEWMAN, Sam. **Building Microservices: Designing Fine-Grained Systems**. 2021. O'Reilly Media; 2nd ed. 2021.11