Template de TCC-Pesquisa na Engenharia de Software: Formato da Sociedade Brasileira de Computação (SBC)

André Rodrigues de Freitas Faria[1]

[1]Instituto de Ciências Exatas e Informática - Pontifícia Universidade Católica de Minas Gerais (PUC MINAS) R. Dom José Gaspar, 500 - Coração Eucarístico, Belo Horizonte - MG, 30535-901

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 escrtos 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.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Danilo Boechat - daniloboechat@pucminas.br
Orientador de conteúdo (TCC I): Joana Souza - joanasouza@pucminas.br
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador do TCC II: (A ser definido no próximo semestre)

Belo Horizonte, DIA de MÊS de ANO.

1. Introdução

A arquitetura de software é uma área da engenharia de software responsável por definir e projetar a estrutura do sistema, definindo seus componentes e a relacao entre eles, sendo tal definição tanto numa camada macro quanto micro do software. Pode se fazer uma analogia entre a arquitetura de software e a arquitetura propriamente dita onde a materialidade do software se vê a partir dos componentes e módulos do sistema. Segundo Martin Fowler: "Arquitetura de software é o conjunto de estruturas necessárias para raciocinar sobre um sistema de software".

Por ser uma das partes mais cruciais de um software, uma arquitetura mal feita pode acarretar em vários problemas. Um dos modelos mais tradicionais de desenvolvimento é chamado de monolito, onde o sistema consiste em apenas um módulo responsável por resolver todos os problemas do software. O modelo monolitico pode levar o software

a ter vários problemas de escalabilidade, manutenção e desenvolvimento. Monolito distribuído foi um modelo arquitetural criado para tentar contornar os problemas do monolito tradicional, porém, tal prática levou apenas a mais problemas de dependencia entre os modulos do monolito. Essa pesquisa tem como objetivo entender o real impacto de escalabilidade e desenvolvimento que um modelo de monolito distribuído possui.

Monolitos distribuídos podem causar grandes problemas a um sistema de software, entender seu real impacto é de grande importância. Muitos softwares grandes no mercado ainda utilizam tecnologias antigas e principalmente arquiteturas antigas e ultrapassadas com o monolito distribuído. Entender o real impacto do monolito distribuído pode ser de grande ajuda para ajudar desenvolvedores a migrarem seu sistema para uma arquitetura de microsservicos visto que, muitas vezes, um monolito distribuído muitas vezes é a tentativa fracassada de desenvolver microsserviços

O objetivo geral desse trabalho é **analisar o desempenho de uma arquitetura de monolito distribuido em relacao a um microsservico**. Como objetivo especifico esse trabalho busca comparar o tempo de resposta entre uma arquitetura de microsservicos e uma arquitetura monolitica. Busca também analisar o consumo de recursos de uma mesma aplicacao utilizando as duas arquiteturas supracitadas. Por fim, essa pesquisa busca entender os impactos da escalabilidade horizontal em ambas as arquiteturas.

A arquitetura de microsservico veio para ser uma evolucao dos monolitos e entende-se que um monolito distribuído nasce da tentativa mal sucedida de um monolito. Os resultados esperados sao que o consumo de recursos de um monolito distribuido é maior que os de um microsservico. Também espera-se que o tempo de resposta de um monolito distribuido seja maior que o de um microsservico. Por fim também esperasae que a escalabilidade horizontal de uma arquitetura de microsservico seja muito mais escalavel e efetiva.

Durante a pesquisa será desenvolvido o backend de um projeto de teste nas arquiteturas de monolito distribuído e de microsservicos. Isso é de suma importancia para ter um ambiente controlado onde ambos os projetos terao a mesma funcionalidade porem em arquiteturas diferentes. Será documentado exatamente a estrutura de cada um dos projetos, assim como infraestrutura cloud e funcionalidade dos projetos. De tal forma será possível comparar claramente cada uma das arquiteturas.

2. Fundamentação Teórica

2.1. Images

All images and illustrations should be in black-and-white, or gray tones, excepting for the papers that will be electronically available (on CD-ROMs, internet, etc.). The image resolution on paper should be about 600 dpi for black-and-white images, and 150-300 dpi for grayscale images. Do not include images with excessive resolution, as they may take hours to print, without any visible difference in the result.

2.2. Figures and Captions

Figure and table captions should be centered if less than one line (Figure 1), otherwise justified and indented by 0.8cm on both margins, as shown in Figure 2. The caption font must be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

Figura 1. A typical figure

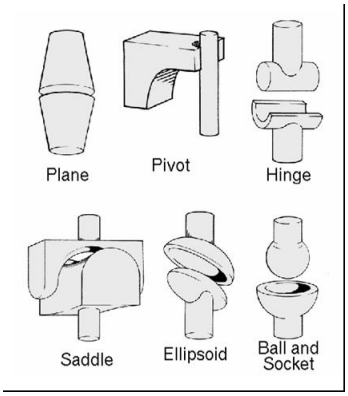

Figura 2. This figure is an example of a figure caption taking more than one line and justified considering margins mentioned in Section 2.2.

3. Trabalhos Relacionados

3.1. Sections and Paragraphs

Nas últimas décadas, arquiteturas de software vêm evoluindo com o intuito de atender às crescentes demandas por escalabilidade, manutenção e desempenho em aplicações web. Nesse contexto, diversos estudos comparativos entre monólitos e microsserviços foram conduzidos com o objetivo de identificar vantagens, limitações e cenários mais adequados para cada abordagem.

Khant Hmue et al. (2024) realizaram um estudo prático e comparativo entre arquiteturas monolíticas e baseadas em microsserviços aplicadas a um sistema de e-commerce. Ambas as versões foram desenvolvidas com a stack MERN (MongoDB, Express, React e Node.js) e submetidas a testes de carga utilizando o Apache JMeter. Os resultados demonstraram que a arquitetura de microsserviços apresentou menor latência (777 ms contra 1488 ms no monólito, sob 200 usuários concorrentes), menor taxa de erro (19,93

Em outro estudo, Hasan et al. (2023) focam especificamente na métrica de manutenibilidade de arquiteturas de software durante a migração de sistemas legados mo-

nolíticos para microsserviços. Os autores propõem um modelo baseado em métricas estruturais, como acoplamento, coesão e complexidade, para avaliar a qualidade arquitetural de sistemas que adotam microsserviços. A pesquisa é fundamentada em modelos de qualidade reconhecidos, como o ISO/IEC 25010, e propõe a utilização dessas métricas ainda na fase de projeto, como forma de reduzir a dívida técnica e evitar falhas estruturais durante ou após a migração. Embora não realizem testes empíricos, os autores sugerem que tais métricas podem ser integradas a ferramentas de análise estática para apoiar decisões arquiteturais fundamentadas (Hasan et al., 2023).

Por fim, Kamisetty et al. (2023) apresentam uma análise comparativa ampla entre monólitos e microsserviços com foco em escalabilidade e desempenho, com base em revisões sistemáticas da literatura e estudos de caso reais. O artigo destaca que, embora os monólitos ofereçam simplicidade e facilidade de desenvolvimento inicial, eles se tornam progressivamente mais difíceis de escalar e manter. Microsserviços, por outro lado, promovem isolamento de falhas, escalabilidade independente de componentes e ciclos de desenvolvimento mais ágeis. Contudo, a adoção de microsserviços impõe novos desafios relacionados à comunicação interserviços, consistência de dados e governança arquitetural. O estudo conclui que não há uma abordagem universalmente superior, e a escolha entre monólito e microsserviços deve considerar o contexto específico da aplicação e as capacidades da equipe de desenvolvimento (Kamisetty et al., 2023).

Esses trabalhos evidenciam que, embora a arquitetura de microsserviços apresente benefícios claros em relação a desempenho e escalabilidade, especialmente em cenários de alta concorrência, há uma complexidade inerente que deve ser cuidadosamente avaliada. A análise comparativa proposta neste trabalho contribui com essa discussão ao explorar uma alternativa intermediária, o monólito distribuído, que busca unir benefícios de modularidade e simplicidade.

4. Materiais e Métodos

4.1. Exemplo de Listing

```
public class Phone {
  private final String unformattedNumber;

  public String getNumber() {
    return unformattedNumber.substring(6,10);
  }
}
```

Listing 1. Code Example

4.2. Exemplo de Algorithm

4.3. Exemplo de Cronograma

Esta seção apresenta um exemplo de tabela para construção de cronograma.

4.4. References

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

Algorithm 1: Código fonte em Java

```
void printOwing (double amount){
  print Banner();
  //print details
    System.out.println("name: " + _name);
    System.out.println("amout: " + amount);
}
```

Tabela 1. Distribuição de tarefas por datas

	2025						
	fevereiro		**março**		**abril**		**maio**
Tarefas	1ªQ	2ªQ	1ªQ	2ªQ	1ªQ	2ªQ	1ªQ
Desenvolvimento do *script* para coleta dos repositórios	X						
Coleta de dados dos repositórios		X					
Desenvolvimento da solução		X					
Coleta de dados			X	X			
Aplicação do Método			X	X	X		
Geração dos painéis						X	
Discussão e avaliação dos resultados							X

Cormen et al. (2016) representa uma citação direta.

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.

Referências

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2002). Algoritmos: teoria e prática. *Editora Campus*, 2:296.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.

Apêndice

A. Exemplo de Seção em um Apêndice