Refatoração de arquiteturas monolíticas em micro serviços no contexto de desenvolvimento de software global

Resumo

Nos últimos anos, o Desenvolvimento de Software Global tem sido adotado por diversas empresas, de modo que possam se beneficiar das vantagens atreladas ao uso da prática supracitada, que vão desde redução de custos à redução tempo necessário para um produto chegar ao mercado. Porém essa técnica também traz diversos desafios aos times que a utilizam. Este trabalho tem como objetivo mapear as estratégias de refatoração em aplicações alinhadas com as características arquiteturais exigidas por um ambiente de desenvolvimento de software global. Para isso, foi utilizado os benefícios provenientes do uso de uma arquitetura de software moderna, a arquitetura baseada em micro serviços. Também analisamos, que tipo de práticas podemos usar para evoluir as aplicações atuais que utilizam uma arquitetura ultrapassada, a arquitetura monolítica, para essa arquitetura baseada em micro serviços. Adotamos, como método de pesquisa, dois mapeamentos sistemáticos, com o objetivo de analisar o estado da arte em relação a arquitetura de software no contexto de desenvolvimento de software global e refatoração de aplicações monolíticas em micro serviço. Por fim, realizamos uma análise dos dados obtidos nos mapeamentos sistemáticos, com objetivo de identificar que práticas de refatoração e benefícios do uso de arquitetura em micro serviços podem contribuir para o sucesso de um projeto no ambiente de desenvolvimento de software global. Com isso, conseguimos concluir que todas as práticas encontradas durante esse trabalho tem algum impacto positivo dentro do ambiente GSD e relacionamos as necessidades do ambiente GSD com os benefícios oferecidos pelo uso da arquitetura em micro serviços.

Palavras-chave: Desenvolvimento de Software Global, Arquitetura de Software, Refatoração, Micro serviços, Monolítico, GSD.

Abstract

In the past few years, Global Software Development has been adopted by many companies, as they can benefit from the advantages linked to the use of this prac- tice, from the reduction of costs and time to market. However, this technique brings many challenges to the teams who use it. With this work, we aim to map the strategies of refactoring applications aligned with the architectural characteristics demanded by the distributed software development environment. For this, we used the benefits de- riving from using a modern software architecture, the microservices architecture. We also analyze what kind of practices we could apply to evolve the current applications that use an old architecture, the monolithic architecture, to this architecture based on microservices. We conduct two systematic mappings, to analyze the state-of-art of software architecture on the Global Software Development environment and refactoring of monolithic to microservices applications, respectively. By the end, we execute an analysis to identify the practices of refactoring, and the benefits of using the microservices architecture can contribute to the success of a project on the GSD environment. With this, we were able to conclude that all the practices found during this work have positive impacts inside of the GSD environment, and we relate to the necessities of the GSD environment with the offered benefits by using the microservice architecture.

Keywords: Global Software Development, Refactoring, Software Architecture, Microservices, Monolithic, GSD.

1. Introdução

Global Software Development (Desenvolvimento de Software Global - GSD) ou *Distributed Software Development* (Desenvolvimento de Software Distribuído - DSD) pode ser definido como atividades de software desenvolvidas em localizações geográficas separadas, ultrapassando fronteiras de nações, envolvendo interações em tempo real ou não (SAHAY; NICHOLSON; KRISHNA, 2003).

Nos últimos 10 anos, GSD tem se tornado um fenômeno utilizado por empresas de pequeno (NOLL et al., 2016) e de grande porte (SAHAY; NICHOLSON; KRISHNA, 2003). Os fatores chave que fazem aumentar o interesse das companhias pelo GSD, são principalmente acesso a uma maior variedade de habilidades, redução de tempo utilizado para um produto chegar ao mercado (*time to market*), proximidade do consumidor e redução de custos (SIEVI-KORTE; BEECHAM; RICHARDSON, 2019; SIEVI- KORTE; RICHARDSON; BEECHAM, 2019; MISHRA; MISHRA, 2013; BEECHAM; RI-

CHARDSON; NOLL, 2015). Porém, mesmo com esses benefícios, existe um desafio em comum entre todos os projetos distribuídos que tornam eles mais desafiadores do que os projetos co-localizados, é a distância global (SIEVI-KORTE; RICHARDSON; BEECHAM, 2019; MARINHO; NOLL; BEECHAM, 2018).

Essa distância global (HERBSLEB et al., 2001) possui três dimensões - temporal, geográfica e sociocultural - onde as duas primeiras são consequências de utilizar locais de desenvolvimento distantes uns dos outros. A distância sociocultural pode causar problemas de mal-entendimento e confiança entre os desenvolvedores distribuídos em diferentes localizações (DESHPANDE et al., 2010). Uma forma de mitigar esses problemas é minimizando a necessidade de comunicação entre os locais de desenvolvimento (SIEVI-KORTE; RICHARDSON; BEECHAM, 2019). Sievi-Korte et al. afirma que a necessidade de comunicação entre os locais está diretamente ligada as dependências entre os componentes de software que são regidas pela arquitetura de software utilizada (SIEVI-KORTE; RICHARDSON; BEECHAM, 2019).

Nos últimos anos, companhias como Google, Amazon, SoundCloud, Netflix, eBay, e outras, têm migrado suas aplicações de uma arquitetura monolítica (RICHARDSON, 2018) (MA) para uma arquitetura de micro serviços (MSA) (FAN; MA, 2017; ALSHUQAYRAN; ALI; EVANS, 2016). Essa transição tem ocorrido para tornar possível que seus serviços se beneficiem com as vantagens da arquitetura de micro serviços e o poder de escalar essas aplicações horizontalmente. Além disso, MSA proporciona outras vantagens como manutenibilidade, reusabilidade, disponibilidade e *deploy* automatizado (FRANCESCO; MALAVOLTA; LAGO, 2017; ALSHUQAYRAN; ALI; EVANS,2016).

Nos dias atuais existem diversas estratégias para evoluir de forma gradativa uma aplicação monolítica para uma estrutura baseada em micro serviços (BALALAIE et al., 2018; GOUIGOUX; TAMZALIT, 2017; KECSKEMETI; MAROSI; KERTESZ, 2016). Entretanto, não existe um consenso sobre que estratégias são melhores e em que cenários usá-las. Ocasionalmente, o processo de migração não é recomendado para algumas aplicações. Em alguns cenários, a evolução de uma aplicação não é benéfica economicamente, pode ser melhor reconstruí-la (KAZANAVIČIUS; MAŽEIKA, 2019).

Considerando que a arquitetura em micro serviços possui características que refletem a estrutura da organização a que se destinam, alinhado com a Lei de Conway (CONWAY, 1968), permitindo diminuir a quantidade de pessoas alterando o mesmo código (NEWMAN, 2015), ao contrário do modelo monolítico e que os projetos globais precisam de uma estratégia que otimize a alocação de atividades, como diminuir a comunicação entre os times (SIEVI-KORTE; RICHARDSON; BEECHAM, 2019). Ainda assim, não é

recomendado o uso de micro serviços desde a etapa de modelagem do sistema (NEWMAN, 2015), pois normalmente o time não possui uma compreensão ampla do domínio do projeto. Fowler (FOWLER, 2015), apresenta o conceito *Monolithic First*, onde um sistema deveria começar a ser construído a partir de uma arquitetura monolítica e evoluir gradativamente para uma arquitetura micro serviços. Fowler aponta ainda, baseado em suas experiências, que a maioria dos projetos de sucesso que utilizavam micro serviços, adotaram, um dia, a arquitetura monolítica. Portanto, torna-se necessário o uso da prática de refatoração para tal evolução. Com isso, assume-se que o uso de refatoração de arquitetura monolítica para micro serviços em projetos globais pode ser um fator de sucesso. Alinhado a essa suposição, assumimos como pergunta de pesquisa para esse trabalho: *"Como refatorar arquiteturas monolíticas em micro serviços no contexto de projetos globais?"*. Para ajudar a responder essa pergunta foram construídas mais três subperguntas de pesquisa.

RQ1 Quais são as principais estratégias existentes na literatura para refatorar aplicações de arquitetura monolítica para micro serviços?
RQ2 Quais características são demandadas por arquiteturas de software no contexto de desenvolvimento de software global?
RQ3 Existem necessidades do ambiente GSD, em relação a arquitetura de software, que podem ser mitigados com a utilização de arquitetura em micro serviços?

Este artigo tem como objetivo principal mapear estratégias de refatoração em aplicações alinhadas com as características arquiteturais exigidas por um ambiente de desenvolvimento de software distribuído. Este artigo contribui para o corpo de conhecimento do GSD ao apresentar um conjunto de práticas para uma arquitetura em micro serviços que atendem as necessidades de um ambiente GSD.

Este artigo está organizado da seguinte forma: na Seção 2, apresentamos o referencial bibliográfico. A seção 3 descreve o método que aplicamos. A seção 4, apresentam os resultados, suas implicações e limitações, respectivamente. Finalmente, a seção 5 apresenta as conclusões e direções de pesquisas futuras.

2. Referencial Teórico

Nesta Seção apresentaremos os principais conceitos necessários para compreender o tema discutido neste trabalho. Também apresentaremos uma discussão dos trabalhos relacionados ao tema deste trabalho, elencando os principais pontos defendidos por cada trabalho e as lacunas que identificamos.

2.1. Visão Geral

2.1.1. Arquiteturas Monolítica e Micro serviços

Arquitetura de software é definida pelo padrão ISO/IEC/IEEE 42010:2011 (MAY, 2011) como "conceitos ou propriedades fundamentais de um ambiente de sistema incorporado em seus elementos, relacionamentos e nos princípios de seu design e evolução". Design de arquitetura de software compreende a construção de elementos e a relação entre esses elementos de modo que a ligação entre esses elementos forneça uma descrição de alto nível. Essencialmente a modelagem de arquitetura documenta as decisões e os elementos que compõem a arquitetura do software (SIEVI-KORTE; BEECHAM; RICHARDSON, 2019).

Arquitetura monolítica é o conceito comumente usado para descrever as aplicações tradicionais existentes atualmente, que adotam uma estrutura onde existe uma única unidade de processamento com múltiplas responsabilidades de negócio. As aplicações monolíticas podem ser imensas e difíceis de gerenciar. (KAMIMURA et al., 2018)

A arquitetura baseada em micro serviços surgiu recentemente e que propõe que os sistemas sejam fragmentados em micro sistemas que funcionem de forma independente, de tal forma que melhorem sua disponibilidade, manutenibilidade e escalabilidade quando for necessário disponibilizar mais recursos para um melhor desempenho e experiência dos usuários do sistema. Newman (2015), ainda aponta duas características para arquitetura em micro serviços, a Tabela 1 lista essas características e quais os fundamentos por trás de cada uma. Newman (2015) também apresenta alguns benefícios, que estão listados Tabela 2.

As Figuras 1 e 2 mostram respectivamente, a estrutura adotada por uma aplicação, monolítica e em micro serviços. Como apresentado na Figura 1, a arquitetura monolítica possui uma estrutura em que todas as solicitações enviadas pelo usuário através da interface são recebidas por um ponto central da aplicação. Já na mesma aplicação que adota uma arquitetura de micro serviços (2), uma solicitação do usuário pode ser recebida apenas por uma parte do sistema, diminuindo a carga de trabalho no sistema e a necessidade por um servidor com uma configuração muito robusta.

2.1.2. Refatoração de Aplicações

Segundo Fowler (2018), refatoração é um processo de alteração de sistemas visando melhorar seu funcionamento interno, porém sem impactar seu comportamento externo. Um dos principais objetivos da refatoração de aplicações é reestruturar o código de modo que melhore a manutenibilidade da aplicação e diminua os riscos de introdução de *bugs* na aplicação. Basicamente quando refatoramos, estamos melhorando o *design* do código após ele escrito (FOWLER, 2018).

O SWEBOK (BOURQUE; FAIRLEY et al., 2014), define o termo refatoração como uma técnica de reengenharia que tem como objetivo reorganizar uma aplicação sem alterar seu comportamento. O SWEBOK ainda diz que com o uso de refatoração, como uma técnica de manutenção, procura-se melhorar a estrutura de um software e sua manutenibilidade.

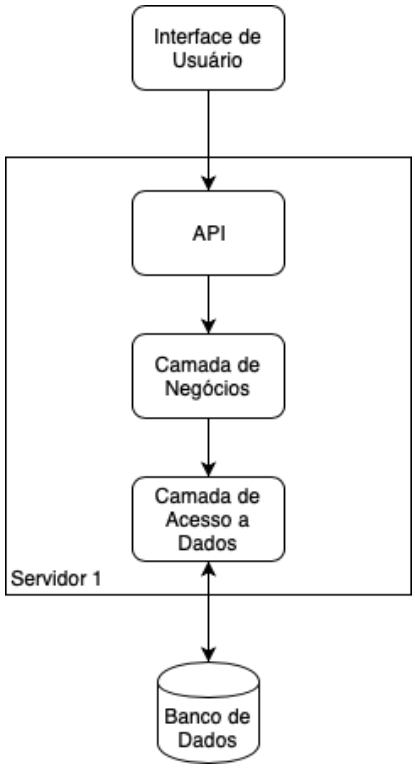
O desafio da refatoração, inclui identificar áreas potenciais que o arquiteto de software deve otimizar. Para refatoração de código, os autores Fowler (2018), introduziram o conceito de *"Code Smells"* para nomear áreas potenciais para melhoria, porém esse termo também pode ser utilizado para indicar problemas de arquitetura (STAL, 2014).

2.1.3. Desenvolvimento de Software Global

Global Software Development (Desenvolvimento de Software Global - GSD) é o termo utilizado para descrever as seguintes situações: organizações que deslocam toda ou parte do seu desenvolvimento de software para destinos de baixo custo ou para destinos em que as habilidades necessárias estão mais disponíveis; organizações que distribuíram seu desenvolvimento de software em vários países diferentes. O software pode ser desenvolvido globalmente por uma organização para usar-se, vender ou incorporar em um dispositivo que a empresa vende (in-sourcing). Uma empresa pode terceirizar desenvolvimento de software para um fornecedor, no mesmo país ou num país diferente, que, em seguida, desenvolve o software necessário para o cliente (terceirização) (VERNER et al., 2012).

Tabela 1 – Características arquitetura em micro serviços (NEWMAN, 2015)

Características	Descrição
Ser pequeno e focado em fazer uma única coisa de forma eficiente	Baseado no Princípio da Responsabilidade Única (MARTIN, 2009), definido por Robert C. Martin (também conhecido como "Uncle Bob"), a arquitetura de micro serviços aplica esse princípio no contexto de escopo de negócio, ou seja, cada micro serviço possui um conjunto de funcionalidades relacionadas a uma parte do negócio do sistema (NEWMAN, 2015).
Autônomo	Cada serviço funciona de forma isolada, onde a comunicação entre os serviços se comunicam entre si via chamadas na rede. Esses serviços precisam ser capazes de mudar independente uns dos outros. Cada serviço expõe uma API (*Application Programming Interface*) e eles devem se comunicar entre si através desta API (NEWMAN, 2015).



Fonte: O Autor
Figura 1 – Estrutura de uma aplicação com arquitetura monolítica

Tabela 2 – Benefícios chave da arquitetura em micro serviços (NEWMAN, 2015)

Benefícios	Fundamentos
Tecnologias Heterogêneas	Como os serviços são construídos de forma independente e utilizando um padrão de comunicação independente de linguagem, esses fatores tornam possível o uso de diferentes tecnologias dentro de cada serviço, ou seja, em uma estrutura com três serviços cada um pode ser implementado utilizando uma linguagem diferente.
Resiliência	Um conceito chave dentro da engenharia de resiliência é o *bulkhead*, que á uma proteção contra falhas para a aplicação. Caso um componente do sistema falhe, mas a falha não foi propagada, é possível isolar o problema do resto da aplicação. Na arquitetura monolítica, se um serviço falha, toda a aplicação para de trabalhar.
Escalabilidade	Quando falamos de escalabilidade na arquitetura monolítica, é necessário escalar toda a aplicação junta, o que torna muito custosa a escalabilidade de aplicações monolíticas. Já no contexto de micro serviços, como cada serviço funciona de forma independente, é possível escalar cada serviço de forma individual, facilitando o gerenciamento e diminuindo o custo de hospedagem da aplicação.
***Deploy* fácil**	Em arquiteturas monolíticas para executar o *deploy* é necessário subir toda a aplicação, já na arquitetura de micro serviços, o *deploy* de cada serviço é feito de forma independente do resto do sistema.
Alinhamento organizacional	Os micro serviços nos permitem alinhar nossa arquitetura a estrutura da organização, ajudando a minimizar o número de pessoas trabalhando no mesmo código fonte.
Modularidade	Uma das promessas de sistemas distribuídos e de arquiteturas orientadas à serviços é que existem diversas oportunidades para reusar funcionalidades. Micro serviços permite alcançar esse tipo de benefícios de forma aprofundada, pois utilizada uma distribuição de funcionalidades mais isolada.
Otimizado para substituição	Com o modelo de pequenos serviços individuais, o custo de substituí-los por uma implementação melhor ou apagá-los é muito menor que gerencia-los. Times que usam micro serviços, sentem-se completamente confortáveis em reescrever serviços quando necessário ou até mesmo eliminá-los quando não estão sendo utilizados.

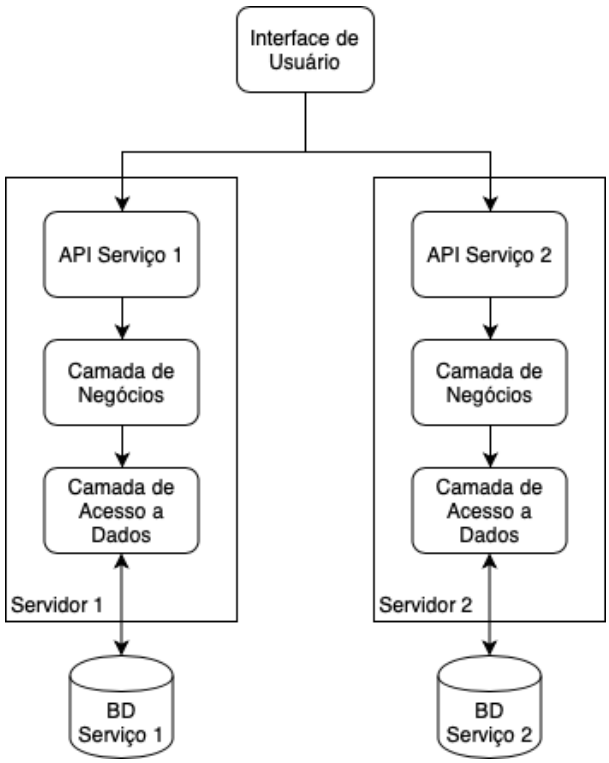
2.1.4. Arquitetura de Software e Desenvolvimento de Software Global

A modelagem de uma arquitetura de software inclui componentes e interfaces (PERRY; WOLF, 1992), que interligam múltiplas estruturas (ALI; BEECHAM; MISTRIK, 2010). A arquitetura de software é usada como meio para coordenar projetos (AVRITZER et al., 2010; HERBSLEB, 2007), não apenas projetos co-localizados, mas também como um mecanismo para alocar tarefas e coordenar times distribuídos (CLERC; LAGO; VLIET, 2007a; HERBSLEB; GRINTER, 1999).

Uma arquitetura de software bem definida, beneficia o processo de software global, garantindo à todos os membros dos times uma linguagem em comum para definir tarefas e atividades, permitindo uma melhor compreensão do domínio de negócio, apesar das diferenças culturais (VANZIN et al., 2005).

Uma arquitetura de software pode assumir várias formas, como por exemplo uma estrutura em camadas, um tipo de *pipeline* estruturado, e pode apresentar interações como por exemplo, baseado em mensagens (TAYLOR et al., 1996), ou baseado em serviços seguindo os princípios de Arquitetura Orientada a Serviços (SOA) (NEW- COMER; LOMOW, 2005), a abordagem RESTful (WHAT…,), ou aplicar uma das mais recentes tendências, denominada microsserviços (WOLFF, 2016; MALAVOLTA; CAPILLA, 2017). A abordagem de microsserviços evoluiu da demanda do uso crescente da computação em nuvem (QIAN et al., 2009; KULKARNI, 2012) e seguindo a abordagem *aaS (*as a Service*).

Ali et al. (ALI; BEECHAM; MISTRIK, 2010) sintetiza, através de uma revisão sistemática da literatura, conceitos que podem ser aplicados durante o processo de modelagem de uma arquitetura. Alguns outros estudos, tais como Fauzi, Bannerman e Staples (2010) consideram construção de software e configuração, entretanto eles só focam na perspectiva de processo. Isso sugere fortemente que existe uma lacuna em pesquisas relacionadas a modelagem de arquitetura de software no contexto de GSD.



Fonte: O Autor

Figura 2 – Estrutura de uma aplicação com arquitetura em micro serviços

3. Método de Pesquisa

Inicialmente foi executada uma pesquisa exploratória para identificar os principais autores e quais termos seriam utilizados para elaborar a *string* de busca utilizada em cada etapa subsequente. Para compreender melhor os conceitos básicos desse trabalho e contexto do problema abordado, optamos por executar dois mapeamentos sistemáticos e para consolidar as informações obtidas e relacioná-las executamos uma análise qualitativa, ambos os métodos serão descritos com seus respectivos protocolos de execução nas seções subsequentes. Dentro desse trabalho iremos focar especificamente em dois modelos arquiteturais, monolítico e micro serviços.

3.1. Mapeamento Sistemático

Mapeamento Sistemático é uma revisão ampla dos estudos primários existentes em um tópico de pesquisa específico que visa identificar as evidências disponíveis. Assim, um mapeamento é um estudo secundário que tem como objetivo identificar e classificar a pesquisa relacionada a um tópico amplo de pesquisa (KEELE et al., 2007).

Segundo Keele et al. (2007). resultado de um mapeamento sistemático é a identificação de lacunas de uma determinada área. Assim, é possível sugerir pesquisas futuras e prover um guia para posicionar adequadamente novas atividades de pesquisa. Ou seja, um mapeamento sistemático visa prover uma visão geral de um tópico e identificar se há subtópicos nos quais mais estudos primários são necessários.

As subseções seguintes descrevem os processos de pesquisa utilizados em dois mapeamento sistemático, visando responder às seguintes perguntas:

RQ1 Quais são as principais estratégias existentes na literatura para refatorar aplicações de arquitetura monolítica para micro serviços?
RQ2 Quais características são demandadas por arquiteturas de software no contexto de desenvolvimento de software global?

3.1.1. Mapeamento Sistemático 1 (Refatoração de aplicações)

Primeiramente, definimos que pergunta de pesquisa norteará a execução dessa etapa e qual a motivação por trás dessa pergunta. A motivação para a pergunta de pesquisa foi revisada e melhorada. A pergunta de pesquisa que motivou essa etapa de pesquisa foi:

"RQ1: Quais são as principais estratégias existentes na literatura para refatorar aplicações de arquitetura monolítica para micro serviços?"

3.1.1.1. Elaboração da *String* de Busca

Assim, nós construímos um mapa mental visando estruturar a relação entre as palavras-chave e seus sinônimos.

Os termos *"Microservices"*, *"Monolithic"*, *"Refactoring"* e *"Modeling"* foram utilizados para construir as versões da *string* de busca, alguns sinônimos de cada termo também foram usados, com o objetivo de englobar artigos que possuam palavras chave similares.

3.1.1.2. Definição dos Critérios de Inclusão e Exclusão

Consequentemente, definimos quais seriam os critérios qualitativos adotados na análise dos artigos e o sistema de ranqueamento foi constantemente melhorado seguindo um método de síntese. Basicamente, esse mapeamento foi executado seguindo os passos a seguir.

Além disso, nós só consideramos artigos publicados em jornais, conferências e periódicos. Livros, teses, workshops, cursos, apresentações e postagem de blogs foram excluídos. Além disto, período da seleção foi restringido para conteúdos publicados após 2014, pois não existe um consenso sobre o termo *microservice* antes dessa data, segundo (PAHL; JAMSHIDI, 2016). Os critérios de inclusão e exclusão foram descritos na Tabela 4.

Foram utilizadas duas bases de dados como fonte de busca dos artigos, IEEExplore e *ACM Digital Library*. A Tabela 3 mostra a versão final das *strings* de busca construídas com base nas palavras chave utilizadas nessa etapa de pesquisa.

Antes da seleção, os artigos passaram por uma avaliação visando avaliar suas respectivas relevâncias em relação a pergunta de pesquisa desse mapeamento. As palavras-chave do autor, resumo, objetivos e resultados foram verificados para determinar a relevância dos artigos e se eles deveriam ser incluídos ou excluídos do estudo, baseado nos critérios de inclusão e exclusão descritos na Tabela 4.

Após a aplicação dos critérios de inclusão e exclusão nós obtivemos 14 artigos.

A Tabela 5 lista os artigos selecionados e onde foram publicados.

3.1.1.3. Classificação

Um vez que os artigos foram selecionados, nós iniciamos uma análise qualitativa, com a intenção de criar um modelo de qualidade desses artigos. O processo de classificação utilizado nessa etapa foi baseado em algumas facetas, de *Research Types* (Tabela 6) e *Contribution Types* (Tabela 7) (PETERSEN et al., 2008). Assim, nós olhamos de uma forma mais aprofundada os artigos selecionados e extraímos as técnicas propostas para execução de refatoração de aplicações com arquitetura monolítica para arquitetura em micro serviços.

3.1.2. Mapeamento Sistemático 2 (Arquitetura em GSD)

3.1.2.1. Pergunta de Pesquisa

Como primeira etapa deste processo definimos a pergunta que norteou e motivou essa etapa da pesquisa, de modo que pudesse contribuir com a pergunta geral deste trabalho (“Como refatorar arquiteturas monolíticas em micro serviços no contexto de projetos globais?”). A pergunta de pesquisa foi:

RQ2: “Quais características são demandadas por arquiteturas de software no contexto de desenvolvimento de software global?”.

Tabela 3 – *Strings* de busca (Mapeamento 1)

Strings de Busca	
IEEExplore	((((((((((((((Refactoring) OR (Decomposition)) OR (Fractionation)) OR (Transformation)) OR (Denormalization)) OR (Decoupling)) OR (Partition)) OR (Modernization)) OR (de-coupling)) OR (decoupled))) OR ((((((((Modeling) OR (UML)) OR (”model driven architecture”)) OR (”model-driven architecture”)) OR (”model driven engieneering”)) OR (”model-driven engieneering”)) OR (”monolithic first”)) OR (”unified modeling language”))))) AND (((((((Microservices) OR (”Micro services”)) OR (Micro-services)) OR (MSA)) OR (”Microservices architecture”)) OR (”microservice architecture engineering”)))) AND (((((Legacy) OR (monolithic)) OR (”legacy applications”)) OR (”monolithic architecture”)) OR (”monolithic first”))))
ACM Digital Library	+(Refactoring Decomposition Fractionation Transformation Denormalization Decoupling Partition Modernization Modeling UML ”model-driven architecturemodel-driven engieneeringmonolithic first unified modeling language”) +(”MicroservicesMicro-servicesMicro servicesMSAMicroservices architecturemicroservice architecture engineering”) +(”Legacymonolithiclegacy applicationsmonolithic architecturemonolithic first”)

Tabela 4 – Critérios de Seleção (Mapeamento 1)

	Critérios
Inclusão	• Estudos com foco em implementação de aplicações que adotam MSA. • Estudos que abordam refatoração de aplicações monolíticas em aplicações em micro serviços. • Artigos que as palavras chaves utilizadas na *string* de busca apareçam no *Abstract* ou nas palavras chaves do autor. • Esteja claro no *Abstract* que o artigo está relacionado a refatoração de aplicações.
Exclusão	• Revisões da literatura, blogs, apresentações e cursos serão excluídos. • Estudos que não tenha como foco refatoração de aplicações. • Estudos que o termo *microservice* fora usado apenas no *abstract*. • Estudos que *microservices* não seja foco de pesquisa. • Estudos com foco apenas em performance. • Estudos focados apenas em analisar ferramentas e plataformas que não tenha *microservices* como principal aplicação. • Estudos que não foram escritos em inglês.

3.1.2.2. Estratégia de Busca

Os termos *"Software Architecture"*, *"Develop"*, "GSD" e seus sinônimos foram utilizados para construir as versões da *string* de busca. Para o termo *"Software Architecture"*, foi utilizado um filtro para aparições apenas no Título e *Abstract*, pois as versões anteriores da *string* que não adotaram esse filtro apresentaram uma fuga em relação ao tema, ou seja, traziam resultados fora do escopo das palavras chave utilizadas.

Não foi definida nenhuma restrição em relação ao ano das publicações. Foram utilizadas duas bases de dados como fonte de busca para os artigos dessa etapa, *IEEExplore* e *ACM Digital Library*. A Tabela 8 apresenta a versão final da *string* de busca adaptada para cada uma das bases utilizadas.

Tabela 5 – Publicações Selecionadas (Mapeamento 1)

ID	Nome do Artigo	Formato
1	An Automatic Extraction Approach: Transition to Microservices Architecture from Monolithic Application (ESKI; BUZ- LUCA, 2018)	Conferência
2	Migrating Web Applications from Monolithic Structure to Microservices Architecture (REN et al., 2018)	Conferência
3	Publishing Linked Data Through Semantic Microservices Composition (SALVADORI et al., 2016)	Conferência
4	Towards the understanding and evolution of monolithic applications as microservices (ESCOBAR et al., 2016)	Conferência
5	Using Microservices for Legacy Software Modernization (KNOCHE; HASSELBRING, 2018)	Conferência
6	Migrating Enterprise Legacy Source Code to Microservices: On Multi Tenancy, Statefulness, and Data Consistency (FURDA et al., 2017)	Magazines
7	From Monolithic to Microservices: An Experience Report from the Banking Domain (BUCCHIARONE et al., 2018)	Magazines
8	The ENTICE approach to decompose monolithic services into microservices (KECSKEMETI; MAROSI; KERTESZ, 2016)	Conferência
9	Priority Order Determination Method for Extracting Services Stepwise from Monolithic System (SHIMODA; SUNADA, 2018)	Conferência
10	Migrating Legacy Software to Microservices Architecture (KAZANAVIČIUS; MAŽEIKA, 2019)	Conferência
11	Extraction of Microservices from Monolithic Software Architectures (MAZLAMI; CITO; LEITNER, 2017)	Conferência
12	From Monolith to Microservices: A Dataflow-Driven Approach (CHEN; LI; LI, 2017)	Conferência
13	Requirements Reconciliation for Scalable and Secure Microservice (De)composition (AHMADVAND; IBRAHIM, 2016)	Conferência
14	Extracting Candidates of Microservices from Monolithic Application Code (KAMIMURA et al., 2018)	Conferência

Tabela 6 – Tipo de Pesquisa - Faceta 1 (Mapeamento 1)

Categoria	Descrição
Proposta de Solução	Uma solução para um problema é proposta, a solução pode ser tanto original quanto uma extensão significativa de uma técnica existente. Os potenciais benefícios e a aplicabilidade da solução é mostrada por um pequeno exemplo ou uma boa linha de argumentos.
Artigos de Experiência	Artigos de Experiência explicam em que e como algo foi feito na prática. Isso deve ser uma experiência pessoal do autor.
Artigos Filosóficos	Esses artigos esboçam uma nova forma de olhar as coisas existentes estruturando a área em forma de uma taxonomia ou um conjunto de recursos conceituais.
Artigo de Opinião	Esses artigos expressam uma opinião pessoal de alguém se uma certa técnica é boa ou ruim, ou como as coisas devem ser feitas. Eles não confiam em trabalhos relacionados e metodologias de pesquisa.

Tabela 7 – Tipo de Contribuição - Faceta 2 (Mapeamento 1)

Categoria	Descrição
Método de Desenvolvimento	Um metodologia de desenvolvimento descreve atividades ou tarefas e seus fluxos de trabalho. Isso também inclui métodos que descrevem regras ou guias de como coisas devem ser feitas, por exemplo, atividades para dar suporte a análise e modelagem de variabilidade dinâmica ou mesmo um modelo de desenvolvimento para lidar com reconfiguração em tempo de execução.
Ferramenta	Uma ferramenta de software desenvolvida para dar suporte a diferentes aspectos de variabilidade em tempo de execução e adaptação dinâmica.
Processo	Um processo é um grupo de passos comumente representada por um diagrama, com entradas e saídas que podem ser reproduzidas.
Discussão Conceitual	Taxonomias, identificação de problemas, relatório de experiência com caracterização de desafios e problemas são considerados nesse tipo de contribuição. Nessa categoria, comparação de resultados entre diferentes características de abordagens existentes são consideradas também.
Algoritmo	Um algoritmo para lidar com aspectos dinâmicos como uma seleção em tempo de execução de variantes, reconfiguração dinâmica de produtos ou mecanismos para tomar decisões.

Tabela 8 – *Strings* de busca (Mapeamento 2)

Strings de Busca	
IEEExplore	(“Abstract”: “software architect” OR “Document Title”: “software architect” OR “Abstract”: “Software Architecture”OR “Document Title”: “Software Architecture”OR ”Abstract”: “Software architectures”OR “Document Title”: “Software architectures” OR “Abstract”: ”Software architectural” OR “Document Title”: ”Software architectural” OR “Abstract”: “Software architecting” OR “Document Title”: “Software ar- chitecting”OR ”Abstract”: ”software architects”OR “Document Title”: “software architects” OR “Abstract”: “system architect”OR “Document Title”: ”system architect”OR ”Abstract”: “System Architecture” OR ”Document Title”: ”System Architecture”OR “Abstract”: “system architecting”OR ”Document Title”: ”system architecting” OR “Abstract”: “software product architect” OR “Document Title”: ”software product architect” OR “Abstract”: “Software Product Architecture” OR “Document Title”: “Software Product Architecture”OR “Abstract”: “Software Product architecting”OR “Document Title”: “Software Product architecting”) AND (design* OR develop* OR product OR products) AND (“global software engineering”OR “global software development” OR “distributed software engineering” OR “distributed software development” OR GSE OR GSD OR “distributed team”OR “global team” OR “dispersed team” OR “spread team” OR “virtual team” OR offshore OR outsource)
ACM Digital Library	+(“global software engineering global software development distributed software engineering distributed software development GSE GSD distributed teamglobal team dispersed team spread team virtual team offshore outsource”) +(“design*develop*product products”) +(“software architect Software Architecture Software architectures Software architectural Software architecting software architects system architect System Architecture Systems Architectures system architecting systems architecting software product architect Software Product Architecture Software Product architecting”)

3.1.2.3. Seleção dos estudos primários

Em um primeiro momento, os artigos passaram por uma avaliação para verificar sua relevância em relação a pergunta de pesquisa à ser respondida por esse mapeamento. As palavras-chave, resumo, objetivos e resultados foram analisados, utilizando a Tabela 9, com intuito de verificar a relevância de cada artigo para esse trabalho, a tabela supracitada define critérios de inclusão e exclusão.

Após a aplicação dos critérios de inclusão e exclusão nós obtivemos 13 artigos.

A Tabela 10 lista os artigos selecionados e onde foram publicados.

Tabela 9 – Critérios de Seleção (Mapeamento 2)

	Critérios
Inclusão	• Trabalhos com foco em modelagem de arquitetura de software • Trabalhos que foquem em times distribuídos • Trabalhos que as palavras chave apareçam no título ou *abstract*. • Esteja claro no *abstract* que o foco seja modelagem de arquitetura de software. • Artigos com mais de quatro páginas • Trabalhos *Peer-reviewed*
Exclusão	• Trabalhos que foquem apenas na perspectiva de processos de desenvolvimento e modelagem que não estejam relacionados com arquitetura de software. • Trabalhos que não foquem desenvolvimento de software global • Trabalhos que as palavras chave não apareçam no título ou *abstract* • Experimentos feitos com estudantes • Experimentos feitos com projetos *open-source* • Trabalhos não escritos em inglês • *Short papers* - artigos com quatro páginas ou menos • Trabalhos que o tópico principal não é arquitetura de software • Trabalhos que descrevam experiência sobre ensino de GSD

Tabela 10 – Publicações Selecionadas (Mapeamento 2)

ID	**Nome do Artigo**	**Formato**
1	A Tool Framework for Deriving the Application Architecture for Global Software Development Projects (YILDIZ; TEKINERDOGAN; CETIN, 2012)	Conferência
2	Architectural Viewpoints for Global Software Development (YILDIZ; TEKINERDOGAN, 2011)	Conferência
3	Activity Theory Applied to Global Software Engineering: Theoretical Foundations and Implications for Tool Builders (TELL; BABAR, 2012)	Conferência
4	Global Software Development: Are Architectural Rules the Answer? (CLERC; LAGO; VLIET, 2007b)	Conferência
5	Coordination Implications of Software Architecture in a Global Software Development Project (AVRITZER et al., 2010)	Jornal
6	Genetic Approach to Software Architecture Synthesis with Work Allocation Scheme (RAIHA; KOSKIMIES et al., 2010)	Conferência
7	Studying the deficiencies and problems of different architecture in developing distributed systems and analyze the existing solution (RAFIGHI; FARJAMI; MODIRI, 2015)	Conferência

8	Engineering of Quality Requirements As Perceived by Near-shore Development Centers' Architects in Eastern Europe: The Hole in the Whole (DANEVA; MARCZAK; HERRMANN, 2014)	Simpósio
9	A Quantitative, Evidence-based Approach for Recommending Software Modules (DANEVA; MARCZAK; HERRMANN, 2014)	Simpósio
10	Planning Global Software Development Projects Using Genetic Algorithms (VATHSAVAYI et al., 2013)	Simpósio
11	A Reference Architecture for Providing Tools As a Service to Support Global Software Development (CHAUHAN, 2014)	Conferência
12	Software Architecture Evaluation in Global Software Development Projects (SALGER, 2009)	Conferência
13	Service Oriented Evolutions and Analyses of Design Patterns (DONG et al., 2006)	Simpósio

3.1.2.4. Classificação

Nessa etapa utilizamos o mesmo método que foi empregado no mapeamento apresentado na Seção 3.1.1, classificamos cada um dos artigos analisando seu tipo de pesquisa e seu tipo de contribuição, baseando-se respectivamente nas Tabelas 6 e 7 respectivamente. Após isso, verificamos os artigos selecionados de uma forma mais aprofundada, com o objetivo de extraímos as características e boas práticas, apontadas como importantes pelos trabalhos analisados, para uma arquitetura de software considerados os aspectos exigidos no contexto de desenvolvimento de software global.

3.2. Análise Axial

Baseado nos conceitos de "Ground Theory" (GLASER; STRAUSS; STRUTZEL, 1968) esse estudo utilizou uma "análise axial" dos dados. Esta etapa envolve um conjunto de procedimentos na qual os dados são agrupados de novas formas, após a codificação aberta, por meio das conexões entre as categorias.

Para realizar a análise axial, começamos com o levantamento das características de arquiteturas GSDs. Em seguida, organizamos essas características de acordo com os benefícios encontrados em uma arquitetura micro serviço.

Depois de termos um catálogo de características da arquitetura GSD e benefícios das arquiteturas micro serviço relacionamos as técnicas de refatoração com os primeiros achados.

Dois pesquisadores participaram do exercício de análise axial. O pesquisador 1 realizou o exercício inicial de categorização e mapeamento; O pesquisador 2 verificou a concordância e sugeriu mudanças. Assim, os dois pesquisadores discutiram as mudanças, até chegarem a um consenso.

O resultado final é uma proposição de estratégias para refatorar arquiteturas GSD monolíticas em micro serviço.

4. Resultados

Nesta Seção, apresentaremos os resultados obtidos após a execução dos métodos de pesquisa descritos na Seção 3. Primeiramente relembraremos as perguntas de pesquisa, definidas no início desse trabalho, e a real por trás de cada uma delas. Para cada pergunta apresentaremos os resultados obtidos e posteriormente consolidamos as informações.

4.1. Revisitando as perguntas de pesquisa

4.1.1. *Quais são as principais estratégias existentes na literatura para refatorar aplicações de arquitetura monolítica para micro serviços?*

Com essa pergunta pretendemos identificar qual o estado da arte em relação às práticas adotadas no contexto de refatoração de aplicações com arquiteturas monolíticas para uma arquitetura em micro serviço, que fornecesse componentes menos acoplados.

A Figura 3, apresenta um gráfico que relaciona os tipos de pesquisas (dados na vertical - eixo y) e os tipos de contribuição (dados na horizontal - eixo x) de cada artigo analisado. Nesse gráfico consideramos as características predominantes em cada artigo, ou seja, se um artigo analisado possuía dos tipos de pesquisa, como proposta de solução e a descrição de uma experiência, e ele fala mais sobre a proposta de solução do que a experiência, no gráfico ele aparecerá categorizado como uma proposta de solução.

Analisando o gráfico podemos perceber que existe uma grande quantidade de propostas de solução focadas em métodos de desenvolvimento, porém em contraste a essa tipo de proposta. Não foram identificados artigos que tenham como principal contribuição, dentro do aspecto de proposta de soluções, o desenvolvimento de uma ferramenta.

Ainda sobre o gráfico da Figura 3, a pouca quantidade de artigos selecionados (ver Tabela 5) no contexto de refatoração de aplicações monolíticas para micro serviços, pode indicar uma falta de interesse na acadêmia em relação ao tema, gerando assim, várias lacunas de pesquisa.

A Figura 4, apresenta um recorte das propostas de soluções mapeadas dentro da amostra analisada. Foram desconsideradas desse recorte proposta de soluções que foram categorizadas como discussões teóricas, tais como processos de desenvolvimento. Podemos constatar que existe um interesse maior, no contexto de métodos e algoritmos para desacoplar sistemas, em técnicas que analisam as capacidade de negócio do sistema, comumente chamado de domínio do negócio.

Outros tipos de técnicas também são adotados, como por exemplo “técnicas baseadas em grafos, que analisam as relações entre os componentes do sistemas e gerando grupos com os componentes que possuem um maior número de interações entre si. Esses grupos de componentes, podem vir a ser refatorados/migrados em conjunto, se tornando um micro serviço. Similar a técnica que utiliza grafos, também surgiram técnicas que consideram “fluxo de interações entre camadas da aplicação”, elas identificam partes da aplicação que não interagem entre si, o que torna essas partes possíveis candidatas a se tornar um micro serviço.

Técnicas que avaliam o “nível de acoplamento entre componentes” também foram encontradas, essas técnicas também analisam o acoplamento entre os componentes, como as técnicas baseadas em grafos e fluxo de dados, porém de uma forma mais alto nível, sem olhar a fundo a relação entre os dados trafegados na aplicação ou o domínio do negócio. As seções a seguir, apresentam um desdobramento das técnicas identificadas nessa etapa de pesquisa.

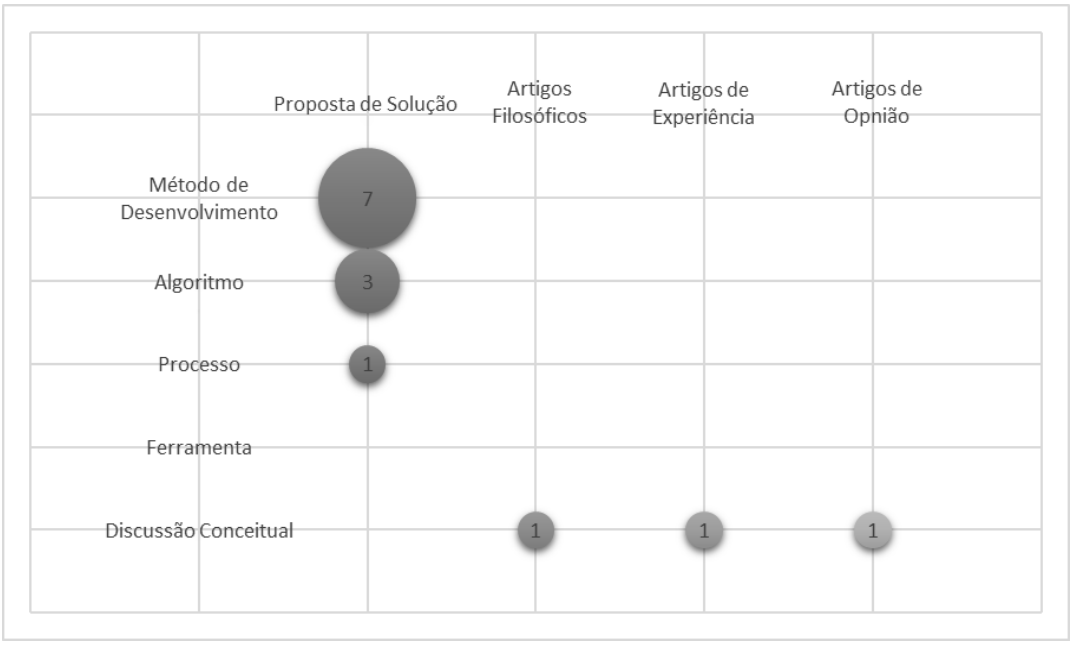


Figura 3 – Relação entre facetas de tipo de pesquisa e tipo de contribuição (Mapeamento 1)

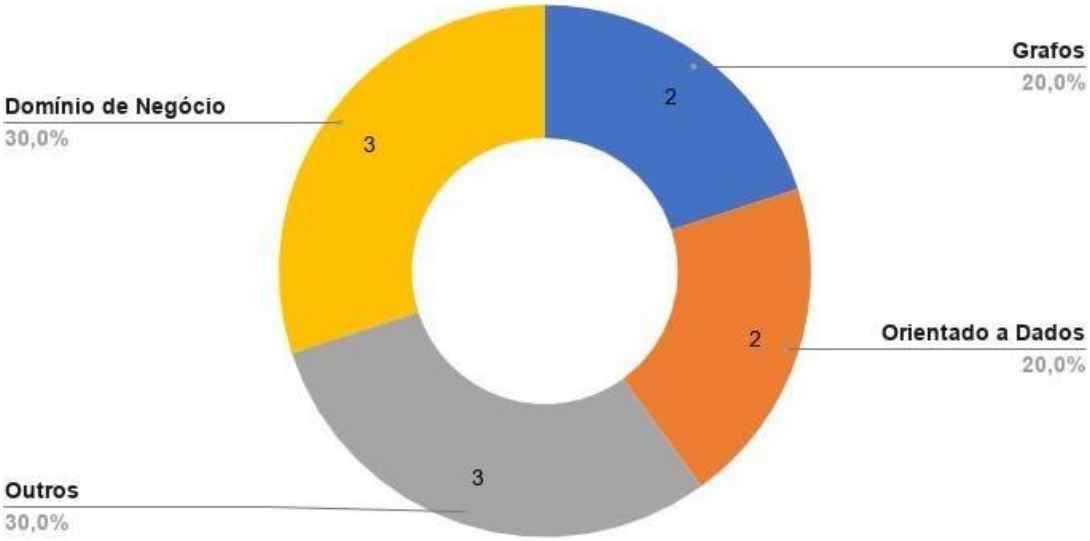


Figura 4 – Agrupamento das contribuições dos artigos analisados (Mapeamento 2)

4.1.1.1. Baseadas em Capacidade de Negócio

Kamimura et al. (KAMIMURA et al., 2018), propõe um método de refatoração que visa avaliar relação entre os pontos de entrada do sistema, ou seja, APIs, telas de *front-end* (interface acessada por usuário) e execuções de processos *batch* (componente que normalmente é responsável por processamento em lotes). Esses pontos de entrada são comumente agrupados por domínio do negócio. Para desenvolver o método, eles utilizam uma algoritmo para agrupamento de partes do sistema baseado acesso a dados e relações dos

dados e chamadas de sistema. Ao fim da execução exibem a estrutura gerada pelo algoritmo e a estrutura anterior do sistema. Kamimura et al. (2018) executaram dois estudos de caso para testar a viabilidade do método proposto.

Shimoda e Sunada (SHIMODA; SUNADA, 2018), define um método de priorização de funcionalidades baseando-se nos impactos positivos e negativos de se refatorar cada funcionalidade. O método proposto possui três etapas, elas são detalhadas a seguir:

1. Primeiro deve ser feita uma avaliação das funcionalidades e verificar os impactos positivos, caso a funcionalidade fosse refatorada, baseado no histórico de desenvolvimento da funcionalidade até o momento da avaliação. A avaliação dos impactos deve ser feita baseada em critérios como número de mudanças realizadas na funcionalidade, facilidade de escalar a funcionalidade, restrições de falhas e outras características por eles definidas.
2. Na segunda etapa do processo, seguindo o mesmo principio da etapa anterior, porém agora focado na avaliação dos impactos negativos da refatoração em cada funcionalidade. Cada aspecto é pontuado e na terceira etapa, é construído um gráfico com o somatório das pontuações, onde os pontos de impacto positivo se somam ao valor final e os pontos dos impactos negativos são subtraídos ao valor final.
3. Baseado no gráfico obtido na terceira etapa do método, tem-se uma lista de prioridade para refatoração das funcionalidades.

Ren et al. (REN et al., 2018), apresenta um método de refatoração de aplicações que diferentemente da maioria dos autores, que focam em análise estática de código e da aplicação em si, foca em análise de comportamento de código em tempo de execução. Nesse método proposto, o algoritmo analisa as interações entre as funcionalidades, relação entre o domínio das funcionalidades, as heranças existentes e as relações entre código executado, essa análise em tempo de execução é definida por eles como “Análise Dinâmica”.

4.1.1.2. Baseadas em Grafos

Eski e Buzluca (ESKI; BUZLUCA, 2018), propõem um método de refatoração de aplicações monolíticas que agrupam as classes e verifica acoplamento entre os componentes do sistema, utilizando algoritmos em grafos e por fim apresentam, como resultado do processamento, possíveis candidatos a micro serviços.

Malzami et al. (MAZLAMI; CITO; LEITNER, 2017), também propõem uma estratégia que utiliza um algoritmo baseado em grafo. Nessa estratégia, eles definem três estágios, o estágio monolítico, o estágio de grafo e o estágio de micro serviço. Existem duas etapas de transição, que são descritas a seguir.

1. Na primeira etapa a estrutura monolítica é convertida para uma estrutura de grafos, onde o peso das arestas entre os vértices definem o nível de acoplamento entre eles, quanto maior o peso, maior será o acoplamento.
2. Por fim, o grafo é separado em grupos e esses grupo serão utilizados para representar o serviços.

4.1.1.3. Orientado a Dados

Salvadori et al. (SALVADORI et al., 2016), propõem um método refatoração semântica baseado nos princípios de *Data Linking*, que é a tarefa de procurar recursos que possam representar de forma equivalente o mesmo recurso no mundo real.

Salvadori et al. (SALVADORI et al., 2016) desenvolveram um *framework* para auxiliar a execução do método proposto, em que é necessário informar como entrada, um conjunto de descrições sobre os micro serviços (que devem descrever os princípios a serem seguidos pelas sugestões de micro serviços geradas), um modelo ontológico e um conjunto de recursos que foram previamente definidos para auxiliar o algoritmo a compreender o mundo real.

Chen el al. (CHEN; LI; LI, 2017), propõem um algoritmo orientado a fluxo de dados, onde o objetivo é fornecer um mecanismo semi-automático para auxiliar no processo de refatoração de aplicações, quebrando as lógicas de negócio de uma aplicação monolítica em candidatos a micro serviço. O algoritmo possui três etapas, que são descritas a seguir.

1. Na primeira ser construído um diagrama da fluxo de dados purificado, para ilustrar de forma detalhada o fluxo de dados de acordo com as lógicas de negócio.
2. Na segunda etapa, é necessário transformar o diagrama de fluxo de dados em um modelo que pode ser decomposto.
3. Na terceira etapa, devem ser identificados os candidatos a micro serviço a partir do modelo de fluxo de dados decomposto.

4.1.1.4. Outras técnicas

Dentre essas outras técnicas existe uma focada em melhorar a análise do acoplamento, como é o caso da técnica proposta por Escobar et al. (ESCOBAR et al., 2016). A técnica por eles proposta, é dividida em três etapas.

1. A primeira etapa, consiste em obter um modelo que represente a estrutura da aplicação a ser refatorada e o resultado dessa etapa é utilizado como entrada na próxima etapa, que é a etapa de consulta.
2. Na segunda etapa, a etapa de consulta, tem-se como objetivo transformar as informações presentes no modelo, resultado da etapa anterior, após a transformação os componentes são agrupados para serem utilizados na etapa final.
3. A etapa final, denominada por eles de visualização, consiste na obtenção de vários diagramas que vão representar os dados obtidos após o processo de clusterização da etapa anterior.

Kecskemeti et al. (KECSKEMETI; MAROSI; KERTESZ, 2016), propõem uma metodologia para refatorar aplicações monolíticas em micro serviços baseada em técnicas de análise e síntese de imagens de máquinas virtuais. Além disso, Ahmadvand e Ibrahim (AHMADVAND; IBRAHIM, 2016), propõem uma metodologia de refatoração que adota como principais critérios os fatores de segurança e escalabilidade, com o intuito de fornecer um conhecimento básico para tomadas de decisões durante a etapa inicial de modelagem de arquitetura de sistemas.

4.1.2. *Quais características são demandadas por arquiteturas de software no contexto de desenvolvimento de software global?*

O Objetivo dessa pergunta era primeiramente entender o estado da arte de modelagem de arquitetura de software no contexto de desenvolvimento de software global. Posteriormente, extrair íamos as principais características que são exigidas por uma arquitetura de software no contexto de desenvolvimento de software global, para que pudesse aumentar a chance de sucesso do projeto.

A Figura 5, apresenta o resultado de análise prévia realizada nos artigos selecionados no segundo mapeamento sistemático (ver Tabela 10), onde relacionamos os tipos de pesquisa com os tipos de contribuição de cada artigo. Nessa análise consideramos as facetas predominantes em cada artigo, ou seja, caso um artigo se enquadre em mais uma faceta ele seria considerado para a faceta predominante. Analisando o gráfico, vemos que a maior parte dos artigos, baseado no conjunto de dados analisado, se enquadrou na faceta de tipo de contribuição, como discussão conceitual, o que pode indicar que os estudos na área de arquitetura de software aplicada a desenvolvimento de software global, ainda estão em um estado teórico.

Percebemos também que existe uma grande quantidade de proposta de solução, porém distribuídas apenas entre algoritmos e ferramentas. Isso pode indicar uma lacuna na área em relação a propostas de solução que de fato possam contribuir no aspecto prático da modelagem de arquitetura de sistemas.

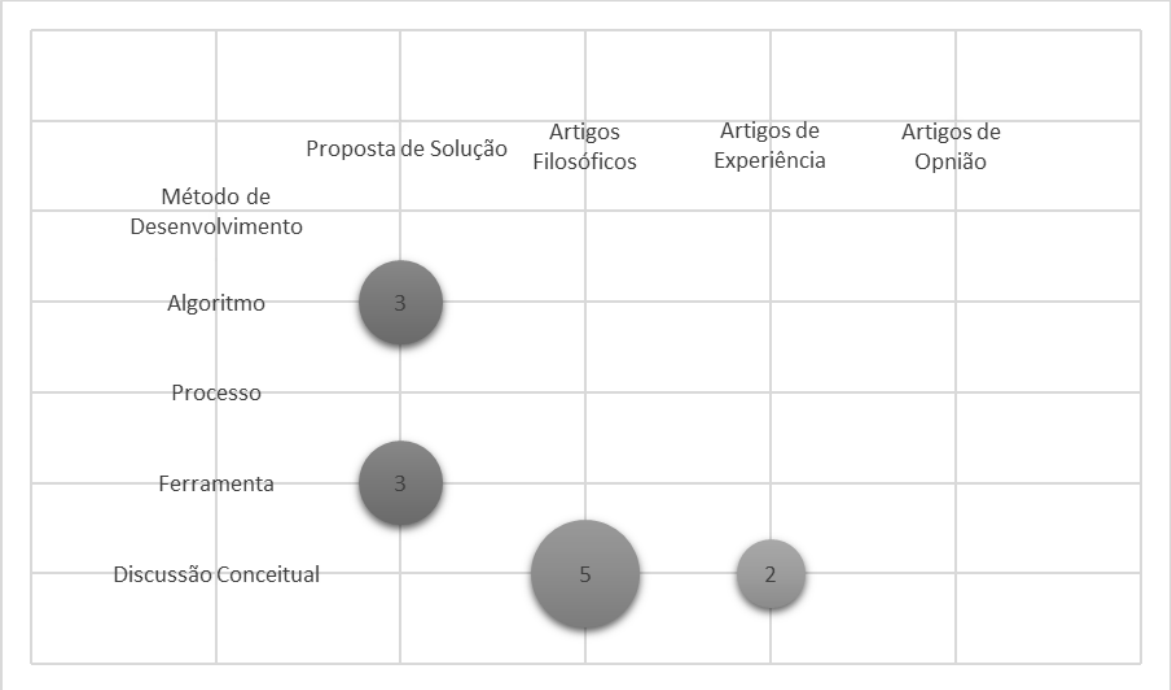


Figura 5 – Relação entre facetas de tipo de pesquisa e tipo de contribuição (Mapeamento 2)

A análise dos dados obtidos pode nos levar a crer, baseado na quantidade de artigos selecionados (ver Tabela 10), que existe uma lacuna de pesquisa em relação a modelagem de arquitetura de software no contexto de desenvolvimento global. Alguns artigos, obtidos nas primeiras etapas dessa pesquisa, acabaram por ser eliminados, pois focava apenas no

aspecto processual e não forneciam insumos a respeito do estágio de modelagem de sistemas do ponto de vista prático.

A Tabela 11, apresenta um levantamento, em relação às preocupações e características exigidas por uma arquitetura de software no contexto de desenvolvimento de software global, obtido a partir da análise dos artigos selecionados nesta etapa de pesquisa (ver Tabela 10). Os dados apresentados foram divididos em três diferentes aspectos que surgiram durante a análise, produto, implementação e comunicação. No aspecto "produto", estamos considerando as características exigidas por um produto finalizado, que podem contribuir para seus requisitos de qualidade e também para requisitos não-funcionais. Já no aspecto de "implementação", estamos tratando daqueles fatores que contribuem para a implementação propriamente dita, que é realizada por times em diferentes localidades, com suas responsabilidades. Por fim, dentro do aspecto de "comunicação", estão compreendidas todas as interações necessárias entre os times, desde etapa de concepção e levantamento de requisitos do projeto, passando pela etapa de implementação das tarefas e chegando na entrega do produto desenvolvido.

4.1.2.1. Disponibilidade

Rafighi et al. (RAFIGHI; FARJAMI; MODIRI, 2015), defende a disponibilidade como uma característica crucial que influencia desde a etapa de modelagem até a o momento em que o produto está em produção. Ela é indispensável em sistemas web de larga escala. Além disso, ele aponta que o tempo que um sistema permanece funcionando é um ponto crítico quando falamos de reputação e de impacto no funcionamento de muitas companhias.

4.1.2.2. Custo

O Custo, tem um impacto tanto no aspecto de desenvolvimento quando no as- pecto produto, pois o crescimento no custo de desenvolvimento é refletido no custo final de um produto. Rafighi et al. (RAFIGHI; FARJAMI; MODIRI, 2015), aponta o custo atrelado ao tempo que um desenvolvedor demora para fazer um *build* da aplicação, o esforço operacional necessário para executar o sistema e até mesmo o custo de treinamentos devem ser considerados.

4.1.2.3. Performance

Segundo Rafighi et al. (RAFIGHI; FARJAMI; MODIRI, 2015), a performance é um fator crucial, pois ela impacta diretamente na experiência do usuário ao utilizar um determinado sistema. Criar um sistema que forneça uma resposta rápida e possua uma baixa latência é crucial.

4.1.2.4. Escalabilidade

Rafighi et al. (RAFIGHI; FARJAMI; MODIRI, 2015), indica que quando falamos em sistemas em larga escala, tamanho não é apenas o único fator a ser considerado. O esforço necessário para aumentar a capacidade de um sistema também deve ser levado em consideração quando falamos de escalabilidade. A escalabilidade pode se referir a diferentes parâmetros de um sistema, como por exemplo, quanto de tráfego ele pode suportar ou até mesmo quantos transações mais ele pode processar.

4.1.2.5. Gerenciabilidade

Modelar um produto que seja fácil de gerenciar, inclusive quando falamos de código, é outra importante característica segundo Rafighi et al. (RAFIGHI; FARJAMI; MODIRI, 2015). Quando falamos de "gerenciabilidade", estamos considerando aspectos como manutenção e atualização.

4.1.2.6. Confiabilidade

Rafighi et al. (RAFIGHI; FARJAMI; MODIRI, 2015), defende que um sistema deve ser confiável, quando um dado é solicitado o sistema deve retornar o mesmo dado de forma consistente. Quando ocorre um evento em que um dado é modificado ou atualizado, então a requisição deve retornar o novo dado. Isso é uma preocupação tanto no aspecto de produto, quanto no aspecto de implementação.

Tabela 11 – Características necessárias em ambiente GSD

Aspecto	**Característica Necessária**	**Referências**
Produto	Disponibilidade	(RAFIGHI; FARJAMI; MODIRI, 2015)
	Custo	(RAFIGHI; FARJAMI; MODIRI, 2015)
	Performance	(RAFIGHI; FARJAMI; MODIRI, 2015)
	Escalabilidade	(RAFIGHI; FARJAMI; MODIRI, 2015)
	Gerenciabilidade	(RAFIGHI; FARJAMI; MODIRI, 2015)
	Confiabilidade	(RAFIGHI; FARJAMI; MODIRI, 2015)
Implementação	Modularidade	(VATHSAVAYI et al., 2013)
	Custo	(RAFIGHI; FARJAMI; MODIRI, 2015)
	Confiabilidade	(RAFIGHI; FARJAMI; MODIRI, 2015)
	Gerenciabilidade	(RAFIGHI; FARJAMI; MODIRI, 2015)
Comunicação	Considerar o ponto de vista do desenvolvedor	(SALGER, 2009)
	Arquitetos co-localizados durante a definição de arquitetura em alto nível	(SALGER, 2009)
	Padrões usados para comunicação	(YILDIZ;TEKINERDOGAN, 2011)
	Comunicação contínua	(CLERC;LAGO;VLIET, 2007b)

4.1.2.7. Modularidade

Vathsavayi et al. (VATHSAVAYI et al., 2013), defende que a modularidade ou desacoplamento é uma característica necessária que pode ser aplicada entre os componentes que compõem a arquitetura. Vathsavayi et al. (VATHSAVAYI et al., 2013), também afirma que existem diferentes soluções para modularizar aplicações ou desacoplar uma aplicação, como por exemplo técnicas de comunicação baseada em mensagem, interfaces REST, *web services* e outras técnicas.

4.1.2.8. Considerar o ponto de vista do desenvolvedor

Salger (SALGER, 2009), defende que é importante levar em consideração o ponto de vista do desenvolvedor, principalmente quando se está definindo a estrutura arquitetural que será adotada no projeto, pois a linguagem adotada pelos desenvolvedores é mais acessível aos próprios desenvolvedores quando forem consumir a documentação, que define a arquitetura, durante o desenvolvimento do sistema.

4.1.2.9. Arquitetos co-localizados durante a definição de arquitetura em alto nível

Salger (SALGER, 2009), aponta que é importante, durante a definição da arquitetura em alto nível, que existam arquitetos trabalhando tanto distribuidamente quanto co-localizados, ou seja, deveria existir mais de um arquiteto trabalhando para dar suporte aos times de desenvolvimento que estão distribuídos.

4.1.2.10. Padrões usados para comunicação

Yildiz e Tekinerdogan (YILDIZ; TEKINERDOGAN, 2011), defendre que exista um meta-modelo que define como as arquiteturas serão definidas, ou seja, deve existir um modelo que estabelece um padrão de linguagem para a especificação da arquitetura, o que pode auxiliar a mitigar os desafios relacionados a comunicação no ambiente de software distribuído.

4.1.2.11. Comunicação contínua

Clerc et al. (CLERC; LAGO; VLIET, 2007b), defende que a comunicação contínua é uma prática necessária para lidar e mitigar os desafios relacionados a diferentes aspectos intrínsecos do ambiente de software distribuído, mas ajuda a mitigar principalmente os desafios relacionados à atividades ligadas a arquitetura de software.

4.1.3. *Existem necessidades do ambiente GSD, em relação a arquitetura de software, que podem ser mitigados com a utilização de arquitetura em micro serviços?*

Com essa pergunta de pesquisa, pretendíamos entender como os benefícios da arquitetura em micro serviços poderiam auxiliar a mitigar os desafios atrelados ao ambiente de desenvolvimento de software global. Também esperávamos compreender a necessidade da refatoração para atingir esses benefícios.

Essa pergunta foi respondida através de uma análise axial, um tipo de análise qualitativa utilizando como dados de entrada os resultados apresentados como resposta para as duas perguntas anteriores.

Para demonstrar a análise realizada, foi elaborado a Tabela 12, que relaciona as preocupações e necessidades exigidas (ver Tabela 11) pelo ambiente de desenvolvimento de software global, que podem contribuir para o sucesso do projeto, com os benefícios atrelados a arquitetura em micro serviços (ver Tabela 2) e as técnicas de refatoração que pode auxiliar no processo de evolução de aplicações

A Tabela 12, apresenta apenas as características e preocupações que podem ser atendidas por algum benefício oriundo da utilização de arquitetura em micro serviços.

Necessidade	Benefício	Técnica
Disponibilidade	Resiliência	Baseado em Grafos (4.1.1.2), Baseado em Capacidade de negócio (4.1.1.1), Orientado a Dados (4.1.1.3), Outros métodos (4.1.1.4)
Custo	Tecnologias Heterogêneas, *Deploy* Fácil, Otimizado para substituição	
Performance	Autonomia e Modularidade	
Escalabilidade	Escalabilidade	
Modularidade	Modularidade	

Tabela 12 – Necessidades x Benefícios x Técnicas

4.1.3.1. Necessidade diretamente ligada a benefício

Baseado nos dados apresentados na Tabela 12, é possível perceber em um primeiro momento, que existem necessidades e preocupações que estão diretamente ligadas aos benefícios de se utilizar uma arquitetura de software em micro serviços. Também concluímos que qualquer uma das técnicas, encontradas como resultado da primeira etapa deste trabalho, podem contribuir até certo ponto para atender as necessidades exigidas pelo ambiente GSD.

As necessidades que se relacionam diretamente aos benefícios são “Escalabilidade” e “Modularidade”. A modularidade é uma necessidade no GSD, pois está diretamente ligada aos desafios de comunicação, com sistema modularizados é possível alocar as atividades de modo que os times possam desenvolver suas atividades de forma independente, e consequentemente diminuir a necessidade por comunicação constante. Já a escalabilidade é uma necessidade que melhora a qualidade do produto desenvolvido e exige uma melhor estruturação do código o que facilita o entendimento dos artefatos de código produzidos durante o processo de desenvolvimento.

4.1.3.2. Necessidades relacionadas aos benefícios

A maioria das necessidades e preocupações são contempladas indiretamente por alguns benefícios de se utilizar uma arquitetura de software em micro serviços. Porém, ainda assim, qualquer técnica de refatoração encontrada, durante a execução desse trabalho, pode atender parcial ou totalmente as necessidades e preocupações do ambiente de desenvolvimento de software global.

No aspecto disponibilidade, podemos facilmente relacionar com a resiliência oferecida pela arquitetura em micro serviços, pois a resiliência é definida pela capacidade que algo ou alguma coisa tem de resistir a adversidades, ou seja, se um sistema é resiliente/resistente ele promove a disponibilidade.

Quando falamos de custo em ambientes GSD, estamos preocupados em controlar os custo tanto no aspecto de implementação, quanto no de produto como um todo. O custo pode ser controlado quando nos aproveitamos de benefícios de micro serviço, como por exemplo o uso de tecnologias heterogêneas, o que permite escolher tecnologias que possibilitem contratar mão de obra com um melhor custo benefício, *deploy* fácil e otimização para substituição seguem o mesmo princípio, porém diminuindo o tempo empregado pelos desenvolvedores para colocar uma versão no ar ou para substituir a versão atual por uma nova.

O aspecto de performance, pode ser atendido até certo ponto quando exploramos os benefícios de autonomia e modularidade, pois os micro serviços são tão independentes e autônomos que possibilitam o uso de estruturas dedicadas a sua hospedagem e utilizam desses recursos para a performance ser transparente ao usuário, mesmo que o sistema esteja sobrecarregado por chamadas simultâneas de diversos usuários.

5. Discussões e Conclusão

O Desenvolvimento de Software Global traz diversos benefícios às empresas que adotam esse tipo de estrutura de desenvolvimento, porém mesmo com todos os benefícios existem desafios atrelados ao ambiente GSD. Muitos dos desafios estão ligados a problemas de comunicação e a complexidade atrelada a alocação de atividade entre os times distribuídos.

Um dos desafios que tem um grande impacto no processo de desenvolvimento de software são os mal-entendimentos causados pelas diferenças culturais dos indivíduos dos times. Em alguns casos, é possível mitigar esse desafio diminuindo a dependência entre os componentes a serem desenvolvidos pelas diferentes localidades do ambiente de desenvolvimento.

Esse trabalho teve como objetivo, fornecer insumos iniciais para dar suporte a novas pesquisas, que possam estudar melhor a implementação de uma arquitetura em micro serviços no ambiente GSD.

5.1. Contribuições

Assim sendo, as contribuições deste trabalho estão descritas abaixo:

- **Benefícios.** Fornecemos uma maneira de relacionar a adoção de uma arquitetura em micro serviços e que necessidades do ambiente GSD elas podem ajudar a atender e que desafios elas podem ajudar a mitigar.
- **Insumos.** Fornecemos insumos que podem auxiliar os times distribuídos a tomar decisões relacionadas a adoção de um arquitetura de software, descritos na Tabela 13.
- **Relação de necessidades e benefícios.** Fornecemos um mapeamento e descrição de como uma arquitetura de micro serviços pode ajudar a mitigar desafios gerados pelo uso de desenvolvimento de software distribuído.

Insumos Obtidos
• Estado da arte em relação a técnicas adotadas atualmente para refatorar aplicações, que podem ajudar um time a decidir qual melhor abordagem para ser utilizada para cada contexto. • Compreender, como técnicas de refatoração de aplicações monolíticas para micro serviços estão ligadas aos benefícios de se utilizar uma arquitetura de micro serviço. • Compreender, como podemos relacionar os benefícios da adoção de uma arquitetura em micro serviços pode ajudar a aproveitar um ambiente de desenvolvimento global em alguns aspectos.

Tabela 13 – Insumos Obtidos

5.2. Ameaças e Limitações

As limitações deste trabalho são expostas na sequência.

- A utilização de apenas duas bases de dados, durante as duas primeiras etapas de pesquisa deste trabalho, foram considerados artigos presentes apenas no *IEEExplore* e na *ACM Digital Library*, pode ameaçar a validade dessa pesquisa.
- O fato deste trabalho utilizar uma abordagem teórica, através de uma discussão conceitual, pode ser considerada uma ameaça a validade dos resultados apresentados.

5.3. Trabalhos Futuros

No decorrer da execução desse trabalho conseguimos identificar lacunas e novas possibilidades de trabalhos que podem utilizar os resultados desse trabalho como insumo. Listamos abaixo os pontos que são possibilidades para trabalhos futuros.

1. Propor uma metodologia de desenvolvimento com foco na evolução da arquitetura de software em ambientes globais, de modo que possamos explorar os pontos levantados por esse trabalho.
2. Realizar um estudo de caso, empregando um método que utilize domínio de negócio para refatorar uma aplicação monolítica para micro serviços em ambientes de software globais.
3. Executar um estudo de caso para validação dos pontos que foram levantados nesse trabalho, de preferência dentro de alguma empresa que já adote desenvolvimento de software através de times globais.
4. Executar um experimento controlado, para avaliar os benefícios e desempenho de cada uma das técnicas de refatoração mapeadas por esse trabalho.
5. Construir uma ferramenta que incorpore algoritmos de avaliação de arquitetura monolítica através de metadados e identifique candidatos a micro serviços.
6. Construir uma ferramenta que auxilie no processo de refatoração dentro da um ambiente de software global e que ajude na gestão de conhecimentos relacionados a arquitetura do sistema.
7. Construir uma ferramenta que auxilie na etapa de modelagem de arquitetura e possibilita distribuir atividades entre os times distribuídos isolando os componentes.
8. Entender o impacto de práticas dentro do contexto de modelagem de arquitetura de sistemas e durante a execução e acompanhamento das atividades de um time global quando adotamos uma arquitetura em micro serviços.

Referências

AHMADVAND, M.; IBRAHIM, A. Requirements reconciliation for scalable and secure microservice (de) composition. In: IEEE. *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. [S.l.], 2016. p. 68–73.

ALI, N.; BEECHAM, S.; MISTRIK, I. Architectural knowledge management in global software development: a review. In: IEEE. *2010 5th IEEE International Conference on Global Software Engineering*. [S.l.], 2010. p. 347–352.

ALSHUQAYRAN, N.; ALI, N.; EVANS, R. A systematic mapping study in microservice architecture. In: IEEE. *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. [S.l.], 2016. p. 44–51.

AVRITZER, A.; PAULISH, D.; CAI, Y.; SETHI, K. Coordination implications of software architecture in a global software development project. *Journal of Systems and Software*, Elsevier, v. 83, n. 10, p. 1881–1895, 2010.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P.; TAMBURRI, D. A.; LYNN, T. Microservices migration patterns. *Software: Practice and Experience*, Wiley Online Library, v. 48, n. 11, p. 2019–2042, 2018.

BEECHAM, S.; RICHARDSON, I.; NOLL, J. Assessing the strength of global teaming practices: A pilot study. In: IEEE. *2015 IEEE 10th International Conference on Global Software Engineering*. [S.l.], 2015. p. 110–114.

BOURQUE, P.; FAIRLEY, R. E. et al. *Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 3.0*. [S.l.]: IEEE Computer Society Press, 2014.

BUCCHIARONE, A.; DRAGONI, N.; DUSTDAR, S.; LARSEN, S. T.; MAZZARA, M. From monolithic to microservices: An experience report from the banking domain. *Ieee Software*, IEEE, v. 35, n. 3, p. 50–55, 2018.

CHAUHAN, M. A. A reference architecture for providing tools as a service to support global software development. In: ACM. *Proceedings of the WICSA 2014 Companion Volume*. [S.l.], 2014. p. 16.

CHEN, R.; LI, S.; LI, Z. From monolith to microservices: a dataflow-driven approach. In: IEEE. *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. [S.l.], 2017. p. 466–475.

CLERC, V.; LAGO, P.; VLIET, H. V. Assessing a multi-site development organization for architectural compliance. In: IEEE. *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. [S.l.], 2007. p. 10–10.

CLERC, V.; LAGO, P.; VLIET, H. V. Global software development: are architectural rules the answer? In: IEEE. *International Conference on Global Software Engineering (ICGSE 2007)*. [S.l.], 2007. p. 225–234.

CONWAY, M. E. How do committees invent. *Datamation*, v. 14, n. 4, p. 28–31, 1968.

DANEVA, M.; MARCZAK, S.; HERRMANN, A. Engineering of quality requirements as perceived by near-shore development centers' architects in eastern europe: the hole in the whole. In: ACM. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. [S.l.], 2014. p. 19.

DESHPANDE, S.; RICHARDSON, I.; CASEY, V.; BEECHAM, S. Culture in global software development-a weakness or strength? In: IEEE. *2010 5th IEEE International Conference on Global Software Engineering*. [S.l.], 2010. p. 67–76.

DONG, J.; YANG, S.; LAD, D. S.; SUN, Y. Service oriented evolutions and analyses of design patterns. In: IEEE. *2006 Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*. [S.l.], 2006. p. 11–18.

ESCOBAR, D.; CÁRDENAS, D.; AMARILLO, R.; CASTRO, E.; GARCÉS, K.; PARRA,

C.; CASALLAS, R. Towards the understanding and evolution of monolithic applications as microservices. In: IEEE. *2016 XLII Latin American Computing Conference (CLEI)*. [S.l.], 2016. p. 1–11.
ESKI, S.; BUZLUCA, F. An automatic extraction approach: transition to microservices architecture from monolithic application. In: ACM. *Proceedings of the 19th International Conference on Agile Software Development: Companion*. [S.l.], 2018. p. 25.
FAN, C.-Y.; MA, S.-P. Migrating monolithic mobile application to microservice architecture: An experiment report. In: IEEE. *2017 IEEE International Conference on AI & Mobile Services (AIMS)*. [S.l.], 2017. p. 109–112.
FAUZI, S. S. M.; BANNERMAN, P. L.; STAPLES, M. Software configuration management in global software development: A systematic map. In: IEEE. *2010 Asia Pacific Software Engineering Conference*. [S.l.], 2010. p. 404–413. Citado na página 19.
FOWLER, M. *Monolith First*. 2015. Disponível em: <https://martinfowler.com/bliki/MonolithFirst.html>.
FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 2018.
FRANCESCO, P. D.; MALAVOLTA, I.; LAGO, P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In: IEEE. *2017 IEEE International Conference on Software Architecture (ICSA)*. [S.l.], 2017. p. 21–30.
FURDA, A.; FIDGE, C.; ZIMMERMANN, O.; KELLY, W.; BARROS, A. Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency. *IEEE Software*, IEEE, v. 35, n. 3, p. 63–72, 2017.
GLASER, B. G.; STRAUSS, A. L.; STRUTZEL, E. The discovery of grounded theory; strategies for qualitative research. *Nursing research*, LWW, v. 17, n. 4, p. 364, 1968.
GOUIGOUX, J.-P.; TAMZALIT, D. From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. In: IEEE. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. [S.l.], 2017.
HERBSLEB, J. D. Global software engineering: The future of socio-technical coordination. In: IEEE. *Future of Software Engineering (FOSE'07)*. [S.l.], 2007. p. 188–198.
HERBSLEB, J. D.; GRINTER, R. E. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, IEEE, v. 16, n. 5, p. 63–70, 1999.
HERBSLEB, J. D.; MOCKUS, A.; FINHOLT, T. A.; GRINTER, R. E. An empirical study of global software development: distance and speed. In: IEEE COMPUTER SOCIETY. *Proceedings of the 23rd international conference on software engineering*. [S.l.], 2001. p. 81–90.
KAMIMURA, M.; YANO, K.; HATANO, T.; MATSUO, A. Extracting candidates of microservices from monolithic application code. In: IEEE. *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. [S.l.], 2018. p. 571–580.
KAZANAVIČIUS, J.; MAŽEIKA, D. Migrating legacy software to microservices architecture. In: IEEE. *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*. [S.l.], 2019. p. 1–5.
KECSKEMETI, G.; MAROSI, A. C.; KERTESZ, A. The entice approach to decompose monolithic services into microservices. In: IEEE. *2016 International Conference on High Performance Computing & Simulation (HPCS)*. [S.l.], 2016. p. 591–596.
KEELE, S. et al. *Guidelines for performing systematic literature reviews in software engineering*. [S.l.], 2007.
KNOCHE, H.; HASSELBRING, W. Using microservices for legacy software modernization. *IEEE Software*, IEEE, v. 35, n. 3, p. 44–49, 2018.

KULKARNI, G. Cloud computing-software as service. *International Journal of Cloud Computing And Services Science*, IAES Institute of Advanced Engineering and Science, v. 1, n. 1, p. 11, 2012..

MALAVOLTA, I.; CAPILLA, R. Current research topics and trends in the software architecture community: Icsa 2017 workshops summary. In: IEEE. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. [S.l.], 2017.

MARINHO, M.; NOLL, J.; BEECHAM, S. Uncertainty management for global software development teams. In: IEEE. *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. [S.l.], 2018. p. 238–246.

MARTIN, R. C. *Clean code: a handbook of agile software craftsmanship*. [S.l.]: Pearson Education, 2009.

MAY, I. *Systems and software engineering–architecture description*. [S.l.], 2011.

MAZLAMI, G.; CITO, J.; LEITNER, P. Extraction of microservices from monolithic software architectures. In: IEEE. *2017 IEEE International Conference on Web Services (ICWS)*. [S.l.], 2017. p. 524–531.

MISHRA, A.; MISHRA, D. Software architecture in distributed software development: A review. In: SPRINGER. *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*. [S.l.], 2013. p. 284–291.

NEWCOMER, E.; LOMOW, G. *Understanding SOA with Web services*. [S.l.]: Addison-Wesley, 2005.

NEWMAN, S. *Building microservices: designing fine-grained systems*. [S.l.]: "O'Reilly Media, Inc.", 2015.

NOLL, J.; BEECHAM, S.; RICHARDSON, I.; CANNA, C. N. A global teaming model for global software development governance: A case study. In: IEEE. *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*. [S.l.], 2016. p.179–188.

PAHL, C.; JAMSHIDI, P. Microservices: A systematic mapping study. In: *CLOSER (1)*. [S.l.: s.n.], 2016. p. 137–146.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, ACM, v. 17, n. 4, p. 40–52, 1992.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: *Ease*. [S.l.: s.n.], 2008. v. 8, p. 68–77.

QIAN, L.; LUO, Z.; DU, Y.; GUO, L. Cloud computing: An overview. In: SPRINGER. *IEEE International Conference on Cloud Computing*. [S.l.], 2009. p. 626–631.

RAFIGHI, M.; FARJAMI, Y.; MODIRI, N. Studying the deficiencies and problems of different architecture in developing distributed systems and analyze the existing solution. In: IEEE. *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. [S.l.], 2015. p. 826–834.

RAIHA, O.; KOSKIMIES, K. et al. Genetic approach to software architecture synthesis with work allocation scheme. In: IEEE. *2010 Asia Pacific Software Engineering Conference*. [S.l.], 2010. p. 70–79.

REN, Z.; WANG, W.; WU, G.; GAO, C.; CHEN, W.; WEI, J.; HUANG, T. Migrating web applications from monolithic structure to microservices architecture. In: ACM. *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*. [S.l.], 2018. p. 7.

RICHARDSON, C. Pattern: monolithic architecture. *Microservices. io*, 2018.

SAHAY, S.; NICHOLSON, B.; KRISHNA, S. *Global IT outsourcing: software development across borders*. [S.l.]: Cambridge University Press, 2003.

SALGER, F. Software architecture evaluation in global software development projects. In: SPRINGER. *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*. [S.l.], 2009. p. 391–400.
SALVADORI, I.; HUF, A.; MELLO, R. d. S.; SIQUEIRA, F. Publishing linked data through semantic microservices composition. In: ACM. *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*. [S.l.], 2016. p. 443–452.
SHIMODA, A.; SUNADA, T. Priority order determination method for extracting services stepwise from monolithic system. In: IEEE. *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*. [S.l.], 2018. p. 805–810.
SIEVI-KORTE, O.; BEECHAM, S.; RICHARDSON, I. Challenges and recommended practices for software architecting in global software development. *Information and Software Technology*, Elsevier, v. 106, p. 234–253, 2019.
SIEVI-KORTE, O.; RICHARDSON, I.; BEECHAM, S. Software architecture design in global software development: An empirical study. *Journal of Systems and Software*, Elsevier, v. 158, p. 110400, 2019.
STAL, M. Refactoring software architectures. In: *Agile Software Architecture*. [S.l.]: Elsevier, 2014. p. 63–82.
TAYLOR, R. N.; MEDVIDOVIC, N.; ANDERSON, K. M.; WHITEHEAD, E. J.; ROBBINS,J. E.; NIES, K. A.; OREIZY, P.; DUBROW, D. L. A component-and message-based architectural style for gui software. *IEEE Transactions on Software Engineering*, IEEE, v. 22, n. 6, p. 390–406, 1996.
TELL, P.; BABAR, M. A. Activity theory applied to global software engineering: Theoretical foundations and implications for tool builders. In: IEEE. *2012 IEEE Seventh International Conference on Global Software Engineering*. [S.l.], 2012. p. 21–30.
VANZIN, M.-A.; RIBEIRO, M. B.; PRIKLADNICKI, R.; CECCATO, I.; ANTUNES, D. Global software processes definition in a distributed environment. In: IEEE. *29th Annual IEEE/NASA Software Engineering Workshop*. [S.l.], 2005. p. 57–65.
VATHSAVAYI, S.; SIEVI-KORTE, O.; KOSKIMIES, K.; SYSTÄ, K. Planning global software development projects using genetic algorithms. In: SPRINGER. *International Symposium on Search Based Software Engineering*. [S.l.], 2013. p. 269–274.
VERNER, J. M.; BRERETON, O. P.; KITCHENHAM, B.; TURNER, M.; NIAZI, M. Systematic literature reviews in global software development: A tertiary study. IET, 2012.
WHAT is Rest ? <https://www.restapitutorial.com/lessons/whatisrest.html>. Accessed: 2019-10-20.
WOLFF, E. *Microservices: flexible software architecture*. [S.l.]: Addison-Wesley Professional, 2016.
YILDIZ, B. M.; TEKINERDOGAN, B. Architectural viewpoints for global software development. In: IEEE. *2011 IEEE Sixth International Conference on Global Software Engineering Workshop*. [S.l.], 2011. p. 9–16.
YILDIZ, B. M.; TEKINERDOGAN, B.; CETIN, S. A tool framework for deriving the application architecture for global software development projects. In: IEEE. *2012 IEEE Seventh International Conference on Global Software Engineering*. [S.l.], 2012. p. 94–103.