Proposta de uma abordagem para decompor sistemas monolíticos em microsserviços

Felipe Chateaubriand Brasil, Ricardo Terra

Departamento de Ciência da Computação
Universidade Federal de Lavras

felipe.brasil3@estudante.ufla.br,terra@ufla.br

Resumo. *A arquitetura de microsserviços vem sendo amplamente adotada no desenvolvimento de sistemas de software. Enquanto o desenvolvimento de novos sistemas ocorre de forma natural, adotar esse novo estilo arquitetural em sistemas legados – cuja tarefa de migração pode ser custosa, demorada e financeiramente cara para se realizar – é um problema desafiador. Diante desse cenário, este artigo propõe uma abordagem semi-automática para auxiliar arquitetos de software na tarefa de decomposição de sistemas monolíticos em microsserviços. Um exemplo motivador trouxe resultados promissores uma vez que, com poucas iterações, obteve um conjunto conciso de microsserviços com maior coesão (6,98%) e menor acoplamento (5,9%).*

1. Introdução

A arquitetura de microsserviços vem sendo adotada por diversas organizações de grande porte tais como Amazon, eBay e Netflix. O objetivo é promover a manutenibilidade e reusabilidade dos sistemas, uma vez que cada serviço é implantado de forma independente e potencialmente em plataformas diferentes [6]. Além das vantagens de microsserviços, o apoio de grandes empresas, como as supracitadas, fornece confiabilidade para que diversas fábricas de software optem por substituir sistemas monolíticos por microsserviços.

No entanto, realizar uma migração de uma arquitetura monolítica para microsserviços pode ser uma tarefa árdua, potencialmente demorada e financeiramente cara [3]. Árdua principalmente quando o sistema monolítico estiver altamente acoplado. Demorada pela complexidade em estimar o tempo do processo de migração. E cara, pois além de usualmente necessitar de vários recursos, não é qualquer membro da equipe de desenvolvimento que pode conduzir tal migração.

Diante disso, este artigo propõe uma solução semi-automática para auxiliar arquitetos de software na tarefa de decomposição de sistemas monolíticos em microsserviços. A abordagem consiste na (i) *compreensão* em que são utilizadas análises estática e dinâmica de código para prover um entendimento da dependência entre os elementos de código fonte e na (ii) *decomposição* em que um processo iterativo e interativo provê sugestões para a decomposição do sistema monolítico em microsserviços. Um exemplo motivador trouxe resultados promissores uma vez que, com poucas iterações, obteve um conjunto conciso de microsserviços com maior coesão (6,98%) e menor acoplamento (5,9%).

2. *Background*

2.1. Arquitetura de software

A arquitetura de um sistema pode ser definida como um conjunto de decisões de quais elementos devem ser considerados para que alcance escalabilidade, segurança, manutenibilidade, etc. [10]. É um conceito que mantém todas as fases do projeto em conjunto e influência na definição das atribuições que devem ser executadas pelas equipes de projeto e implementação [7].

2.2. Arquitetura de microsserviços

A arquitetura de microsserviços é uma abordagem para desenvolver um único sistema de software como um conjunto de pequenos serviços, cada um executando em seu próprio processo e comunicando-se com mecanismos leves, geralmente através de uma API de recurso HTTP [5]. Devido aos serviços serem menores e fracamente acoplados, facilita-se a manutenção, teste e escalonamento.

2.3. Migração de sistemas

Como a arquitetura de software não é representada de modo explícito no código [2], em processos de migração arquitetural, engenheiros de software frequentemente recorrem à descoberta arquitetural antes de realizar uma migração ou estender sua funcionalidade. Esse processo é uma tarefa complexa e onerosa que, na maioria das vezes, é realizada inteiramente de forma manual [11].

3. Abordagem Proposta

Esta seção descreve a abordagem proposta para auxiliar arquitetos de software na tarefa de decomposição de sistemas monolíticos em microsserviços. Conforme ilustrado na Figura 1, a abordagem consiste em duas etapas bem definidas: (i) *compreensão* em que são utilizadas análises estática e dinâmica de código para prover um grafo que condensa informações úteis ao processo de decomposição (Seção 3.1) e (ii) *decomposição* em que um processo iterativo e interativo provê recomendações para a decomposição do sistema monolítico em microsserviços (Seção 3.2).

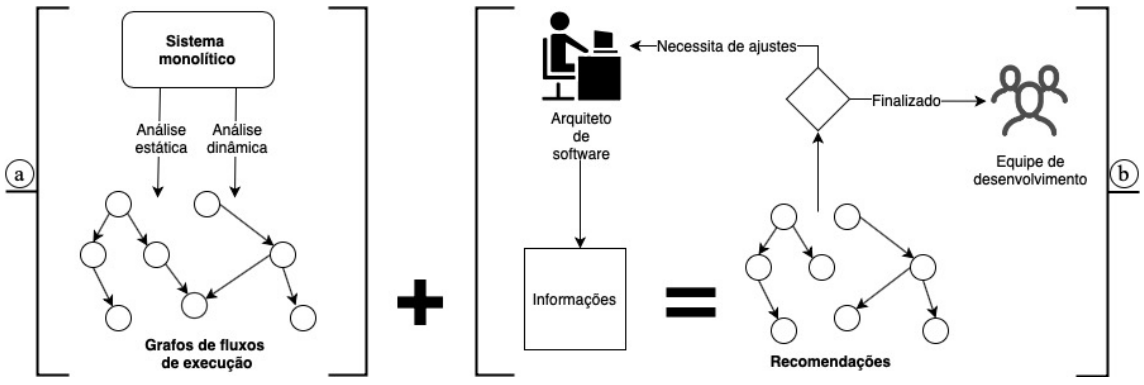

Figura 1. Abordagem proposta (compreensão + decomposição)

3.1. Compreensão

Esta etapa visa obter conhecimento sobre a arquitetura e o funcionamento do sistema monolítico a ser decomposto em microsserviços. Para isso, são necessários: (i) extrair dados estruturais a partir do código fonte e (ii) obter informações referentes às comunicações das classes durante sua execução. Conforme ilustrado na Figura 1a, foi desenvolvida uma análise híbrida que combina análises estática e dinâmica (ver Seção 2).

Por um lado, a análise estática tem como objetivo identificar as classes do sistema, seus métodos e atributos, além de identificar se as classes são entidades ou utilitárias, pois elas têm um alto índice de acoplamento e devem ser trabalhadas de modo distinto[1]. Por outro lado, a análise dinâmica requer que o mantenedor identifique e execute todos os fluxos de casos de uso do sistema, de modo a obter as comunicações entre os métodos das classes, além de associá-las aos fluxos de caso de uso que fazem parte. Essa é a tarefa mais "custosa" da nossa abordagem e tem como objetivo identificar os fluxos de caso de uso para criação de uma lógica de extração pelo negócio, já que o conceito de microsserviços não é apenas sobre a dependência estrutural das classes, mas sim, uma abstração sobre o negócio do sistema em pequenos serviços.

Por fim, cria-se um grafo que agrega todas as informações obtidas. Conforme descrito na Seção 2, os vértices representam classes e as arestas as dependências entre elas. Vértices possuem coloração distinta para diferenciar algumas categorias de classes – e.g., entidades e classes utilitárias – que são trabalhadas de modo distinto. As arestas são ponderadas cujo peso é calculado com base no valor da Equação 1.

$$peso(C_i, C_j) = \sum_{k=1}^{n} \mid métodos(uc_k, C_i, C_j) \mid \tag{1}$$

onde C_i e C_j representam respectivamente as classes de origem e destino, n representa o número de casos de uso e $métodos(uc_k, C_i, C_j)$ retorna o conjunto de métodos C_i que acessam métodos de C_j durante o caso de uc_k.

3.2. Decomposição

Esta etapa visa prover decomposições de um sistema monolítico em microsserviços por meio de um processo interativo com o arquiteto. Conforme ilustrado na Figura 1b, cada iteração gera uma recomendação de decomposição que o arquiteto pode aceitar ou refinar. O processo é iterativo, pois, caso o arquiteto refine o grafo, uma nova recomendação é gerada. De forma sucinta, o processo possui duas atividades bem definidas:

Ajuste do grafo: conforme mencionado, o arquiteto pode realizar ajustes, tais como (i) habilitar que classes sejam divididas ou (ii) ajustes de classes, como transformá-las em entidades, classes utilitárias ou comuns; e (iii) ajustes de comunicações, como obrigar que determinadas classes sejam agrupadas em um mesmo microsserviço ou modificar os pesos automaticamente calculados.

[1]De forma sucinta devido à limitação de espaço, entidades possuem a anotação **`@Entity`** ou são classes com apenas atributos privados (e pelo menos um) em que os métodos públicos sejam apenas acessores (*getters* e *setters*) e os métodos **`equals`** e **`hashcode`**. Classes utilitárias devem ser classes com modificar **`final`** e construtor privado e que tenha todos seus métodos e atributos públicos definidos como estáticos.

Processamento automático: foi desenvolvido um algoritmo capaz de criar grupos a partir do grafo com quatro etapas sequenciais:

1. Considerando apenas classes comuns, o algoritmo associa o nó filho ao nó pai, onde o nó pai de cada vértice é o nó que o referencia e tem o maior peso atribuído na aresta. Em caso de um filho ter mais de um nó pai, mesmo peso em duas arestas, todos são adicionados ao mesmo grupo;
2. Entidades são adicionadas aos grupos que a referenciam. Se uma entidade é utilizada em mais de um grupo, ela será replicada nos microsserviços;
3. Classes utilitárias são adicionadas aos grupos que a referenciam. Caso mais de um grupo a referencie, elas são adicionadas em um novo projeto para serem utilizadas como uma biblioteca externa nos microsserviços que a referenciam; e
4. Se o arquiteto habilitou a separação de classes, o algoritmo analisa se dividir uma classe traz vantagens [4]. Por exemplo, microsserviços MS_1 e MS_2 dependem da classe C, mas MS_1 devido ao método $foo()$ e MS_2 devido ao método $bar()$.

4. Exemplo Motivador

Esta seção ilustra um exemplo motivador de uso da abordagem proposta. A partir de um sistema monolítico descrito na Seção 4.1, esta seção descreve todo o processo realizado por meio da aplicação da ferramenta que implementa a abordagem proposta (Seção 4.2) para se obter a sua decomposição em microsserviços (Seções 4.3 e 4.4).

4.1. Sistema alvo

Foi escolhido um sistema monolítico, de pequeno porte, responsável pela gerência de venda de produtos, o qual foi desenvolvido por terceiros. Ele contém cinco fluxos de caso de uso que são responsáveis pela autenticação, pelas telas de gerenciamento de clientes, produtos e funcionários e pelo fluxo da realização de vendas. O sistema foi implementado com a linguagem de programação Java e utiliza das tecnologias SpringBoot e React.

4.2. Ferramenta

A ferramenta *MicroservicesDecomposer* é um *plug-in* para o IDE Eclipse que implementa a abordagem proposta para a linguagem Java. As análises estáticas são realizadas por meio do projeto Eclipse JDT[2] manipulando *Abstract Syntax Trees (ASTs)*. Por outro lado, as análises dinâmicas são realizadas com aspectos por meio da extensão AspectJ[3]. Todos os dados obtidos foram agrupados e abstraídos para uma base de dados relacional, de modo a gerar um modelo de relacionamento entre as classes do sistema utilizando estrutura de grafos, além de calcular as métricas de acoplamento e coesão para servir de base de comparativo e entendimento.

4.3. Compreensão

A ferramenta *MicroservicesDecomposer* executou a etapa de compreensão. Embora as informações das classes e suas dependências sejam automaticamente obtidas por meio

[2]https://www.eclipse.org/jdt/
[3]https://www.eclipse.org/aspectj/

de análise estática, o primeiro autor deste artigo executou cinco fluxos básicos cujas execuções foram monitoradas pela ferramenta.

A Figura 2 ilustra o grafo de comunicações das classes do sistema avaliado[4]. É possível observar as entidades (cor amarela), as classes utilitárias (cor roxa) e as associações entre as classes que receberam um peso. As dependências mais fracas foram representadas com linha tracejada e as mais fortes com linha inteira.

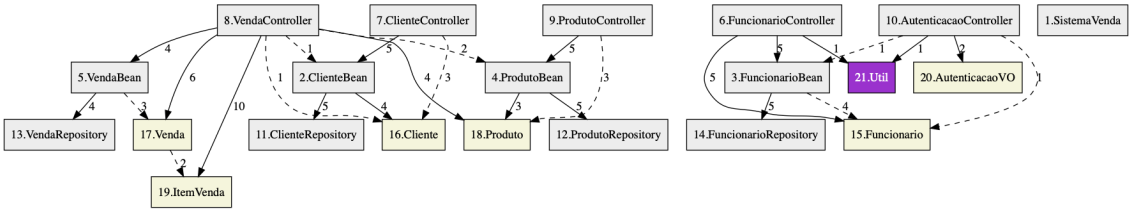

Figura 2. Representação do sistema monolítico (versão 0)

4.4. Decomposição

Versão 1: Em uma primeira iteração de sugestões de microsserviços, o arquiteto não forneceu instruções, o que resultou em uma decomposição utilizando apenas a lógica do processamento automático de recomendação, proposta na Seção 3.2. Essa abordagem resultou em cinco sugestões de microsserviços, além de um grupo utilitário, conforme Figura 3. É possível observar que as divisões foram indicadas respeitando as associações mais fortes. As classes **`Produto`**, **`Cliente`** e **`Funcionario`** que são entidades e eram associadas a mais de um microsserviço foram replicadas. A classe utilitária **`Util`** foi separada por ser referenciada em dois microsserviços distintos e com isso poderá se tornar apenas uma biblioteca a ser utilizada por ambos os microsserviços.

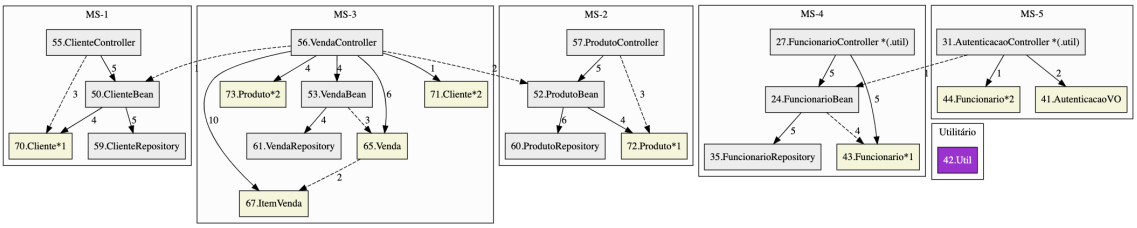

Figura 3. Sugestão de separação de classes do sistema monolítico (versão 1)

Versão 2: Como o processo é interativo, o resultado foi analisado e percebeu-se que as classes **`AutenticacaoController`** e **`FuncionarioBean`** poderiam estar em um mesmo microsserviço, dado que no sistema analisado apenas os funcionários que realizam autenticação. Portanto, refinou o grafo indicando que essas duas classes devem ser agrupadas em um mesmo microsserviço, o que resultou em uma nova versão que foi gerada sobre o resultado anterior. A Figura 4 traz o resultado obtido pela nova iteração, e ao compará-la com a versão 1 é possível observar a união dos grupos **`MS-4`** e **`MS-5`**, a entidade **`Funcionario`** deixou de ser replicada e a classe utilitária **`Util`** saiu do grupo **`Utilitario`** que havia sido criado, pelo fato de passar a ter um único microsserviço que a referencia.

[4]Por questões de legibilidade, todas as figuras estão disponíveis em alta resolução em: https://github.com/anonymoussoftwareengineering/2020_vem_microservices_decomposer/tree/master/AvaliacaoVEM/Imagens

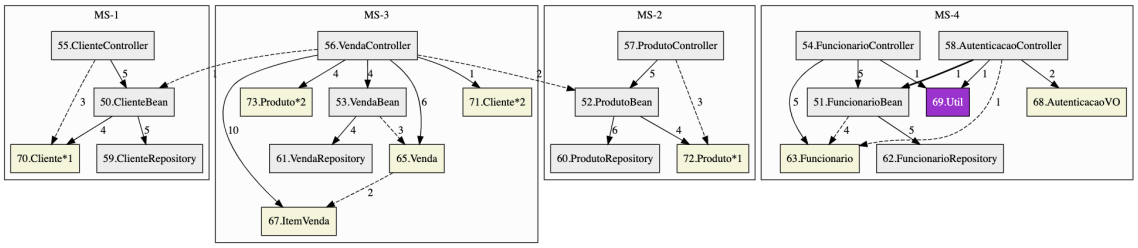

Figura 4. Sugestão de separação de classes do sistema monolítico (versão 2)

Versão 3: Ainda no processo iterativo, o resultado foi reavaliado e observou que havia microsserviços acoplados e, portanto, foi gerada uma nova rodada de recomendação com base no resultado anterior, indicando que poderia separar classes que fazem a união entre dois microsserviços, de modo a tentar deixá-los independentes. Na versão 3, representada pela Figura 5, é possível observar que a classe **`VendaController`** foi desmembrada em duas, deixando independente o microsserviço do **`ClienteBean`** e reduzindo a quantidade de microsserviços acoplados. Essa separação pode ser realizada, pois o método que realizava a chamada era responsável unicamente pela verificação da existência do cliente através do CPF. No caso da dependência da classe **`VendaController`** e **`ProdutoBean`** não foi possível realizar a separação, pois os métodos origem são invocados por outros métodos na própria classe.

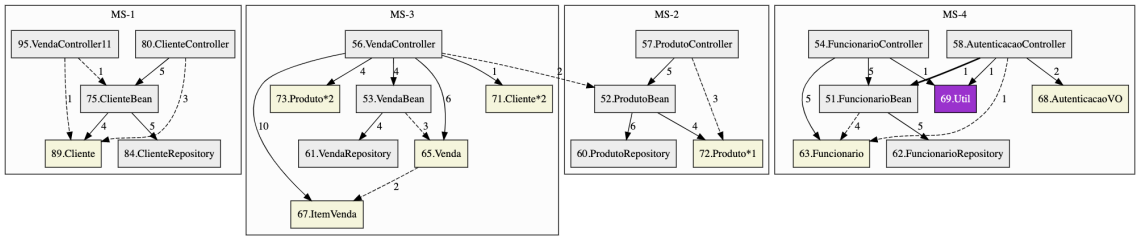

Figura 5. Sugestão de separação de classes do sistema monolítico (versão 3).

4.5. Discussão

Esta seção conduz uma análise de coesão e acoplamento de cada uma das versões propostas. A ferramenta *MicroservicesDecomposer* calcula métricas de cada decomposição proposta de modo a facilitar o entendimento e comparativo do sistema inicial na versão monolítica com as versões geradas. A Tabela 1 reporta os valores para as versões do exemplo motivador.

Tabela 1. Métricas obtidas através do agrupamento de classes em cada versão

Versão	CBO	RFC	LCOM4
0 (monolítica)	1,29	7,90	1,29
1	1,13	8,13	1,29
2	1,17	8,00	1,29
3	1,17	7,70	1,20

Para o acoplamento de *CBO* e *RFC*, os resultados oscilaram nas versões 1 e 2, i.e., quando um melhorava o outro piorava. Isso ocorreu pelo fato de que ao replicar uma

entidade aumentou o número de classes, manteve o número de associações, diminuindo o *CBO*, porém, como entidade costuma ter *RFC* alto, por conta de *getters* e *setters*, o valor de *RFC* na média geral cresceu. Esse fato fez com que fosse necessário manter as duas métricas para acoplamento, de modo a buscar um equilíbrio. Na versão 3 onde ocorreram divisões de classes, ambos os valores melhoraram comparado à versão inicial.

No cálculo da métrica de coesão com *LCOM4*, foram desconsideradas as entidades, pois tendem a ter como valor o número de atributos caso não tenha um construtor que os associe, e as classes utilitárias, que tendem a ter o número de métodos, pois independente do valor apresentado nesses casos a estrutura deve ser mantida, sendo consideradas como ruído. Portanto, os valores da versão 0, 1 e 2 não sofreram alterações e na versão final 3, onde ocorreu a divisão de classes, o valor melhorou.

Embora trata-se de um exemplo motivador, os resultados foram satisfatórios, pois além de realizar a decomposição arquitetural que atenderam as necessidades, os valores das métricas tiveram melhoras na versão final de microsserviço sugerida. Para a métrica de coesão, teve-se uma melhora de 6,98% para *LCOM4* e para a métrica de acoplamento a melhora foi de 5,9%, sendo de 9,3% para *CBO* e 2,53% para *RFC*.

5. Trabalhos relacionados

Kalske et al. [8] identificaram que os desafios na migração da arquitetura monolítica para a de microsserviços vão além das dificuldades técnicas, mas também problemas em níveis organizacionais. As dificuldades técnicas encontradas ocorrem por fatores como o tempo gasto para identificação da separação dos serviços, a chance de gerar novos *bugs*, a possibilidade de aumentar o tráfego de rede através da comunicação entre microsserviços e o problema para se definir granularidade. O artigo evidenciou o alto tempo gasto na identificação das classes de cada serviço, o que motiva estudos que atuem nessa atividade como a proposta neste artigo.

Dehghani [1] apresentou estratégias de divisão de um sistema monolítico para a arquitetura de microsserviços. Foram apresentadas técnicas a serem adotadas pela equipe de desenvolvimento para que a extração não afete as funcionalidades já existentes, como inicializar a decomposição por serviços menores e menos acoplados. Apesar de a autora fornecer uma estratégia para decompor sistemas monolíticos, todo processo é realizado de forma manual, ao contrário da abordagem proposta neste artigo que propõe uma abordagem semi-automática.

Levcovitz et al. [9] reportam uma decomposição de um sistema bancário de arquitetura monolítica para a arquitetura de microsserviços e propõem um processo de extração. Foram identificados cenários que necessitaram de esforços extras, tais como os microsserviços que compartilham a mesma tabela do banco de dados ou que estão no meio de outra operação. Esse trabalho utilizou estrutura de grafos para representação das comunicações do sistema similarmente como a abordagem proposta neste artigo, embora análises e recomendações foram realizadas de forma manual.

6. Considerações finais

Com o decorrer do tempo, fábricas de software precisam aperfeiçoar as arquiteturas dos sistemas por questões de segurança, modernização ou até mesmo para atender a alguma

nova necessidade. Como os processos de reestruturação são normalmente demorados e financeiramente caros, algumas empresas podem optar por ficar com código defasado tecnologicamente ou começar porém não concluir um processo de remodularização.

Diante desse cenário, este artigo propõe uma abordagem semi-automática para auxiliar arquitetos de software na tarefa de decomposição de sistemas monolíticos em microsserviços. A abordagem recebe como entrada um sistema monolítico, (i) realiza uma etapa de *compreensão* (análises estática e dinâmica) para se obter um grafo e (ii) propõe uma potencial decomposição. Como o processo é interativo e iterativo, a cada ajuste no grafo realizado pelo arquiteto, uma nova potencial decomposição é proposta. Um exemplo motivador alcançou resultados promissores uma vez que, com poucas iterações, obteve um conjunto conciso de microsserviços com maior coesão (6,98%) e menor acoplamento (5,9%) do que o sistema na arquitetura monolítica.

A ferramenta *MicroservicesDecomposer* e todos os fontes da avaliação estão publicamente disponíveis em:

https://github.com/PqES/MicroservicesDecomposer

A evolução da abordagem proposta inclui: (i) avaliações reais com sistemas monolíticos de grande porte, (ii) comparativo com outras técnicas de mesmo propósito, (iii) proposta de uma heurística para estabelecer o tamanho (em número de classes) de cada microsserviço e (iv) avaliações com equipes reais de desenvolvimento.

Agradecimentos

Esta pesquisa é apoiada pela FAPEMIG (APQ-03513-18) e CNPq (305829/2018-1).

Referências

[1] Zhamak Dehghani. How to break a monolith into microservices, 2018. Acesso em: 14 maio 2020. Disponível em: https://martinfowler.com/articles/break-monolith-into-microservices.htm.

[2] Stephane Ducasse and Damien Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, 2009.

[3] Daniel Escobar, Diana Cárdenas, Rolando Amarillo, Eddie Castro, Kelly Garcés, Carlos Parra, and Rubby Casallas. Towards the understanding and evolution of monolithic applications as microservices. In *XLII Latin American on Computing Conference (CLEI)*, pages 1–11, 2016.

[4] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley, Boston, 1999.

[5] Martin Fowler and James Lewis. Microservices a definition of this new architectural term, March 2014. Acesso em: 23 março 2020. Disponível em: https://martinfowler.com/articles/microservices.html.

[6] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. Research on architecting microservices: trends, focus, and potential for industrial adoption. In *1st International Conference on Software Architecture (ICSA)*, pages 21–30, 2017.

[7] David Garlan, Felix Bachmann, James Ivers, Judith Stafford, Len Bass, Paul Clements, and Paulo Merson. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2010.

[8] Miika Kalske, Niko Mäkitalo, and Tommi Mikkonen. Challenges when moving from monolith to microservice architecture. In *18th International Conference on Web Engineering (ICWE)*, pages 32–47, 2018.

[9] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. Towards a technique for extracting microservices from monolithic enterprise systems. In *III Workshop de Visualização, Evolução e Manutenção de Software (VEM)*, pages 97–104, 2015.

[10] Leonardo Passos, Ricardo Terra, Marco Tulio Valente, Renato Diniz, and Nabor das Chagas Mendonca. Static architecture-conformance checking: An illustrative overview. *IEEE Software*, 27(5):82–89, 2010.

[11] Aurora Ramírez, José Raúl Romero, and Sebastián Ventura. An approach for the evolutionary discovery of software architectures. *Information Sciences*, 305:234–255, 2015.