SUGESTÃO DE CRITICIDADE BASEADA EM MULTICRITÉRIO PARA ARQUITETURAS ESTABELECIDAS ORIENTADAS A MICROSSERVIÇOS

Eduardo Fernandes Mioto de Oliveira dos Santos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadora: Cláudia Maria Lima Werner

Rio de Janeiro
Abril de 2020

SUGESTÃO DE CRITICIDADE BASEADA EM MULTICRITÉRIO PARA ARQUITETURAS ESTABELECIDAS ORIENTADAS A MICROSSERVIÇOS

Eduardo Fernandes Mioto de Oliveira dos Santos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadora: Cláudia Maria Lima Werner

Aprovada por: Prof. Toacy Cavalcante de Oliveira
Prof.ª Cláudia Maria Lima Werner
Prof. Leonardo Gresta Paulino Murta

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 2020

Santos, Eduardo Fernandes Mioto de Oliveira dos

Sugestão de Criticidade Baseada em Multicritério para Arquiteturas Estabelecidas Orientadas a Microsserviços / Eduardo Fernandes Mioto de Oliveira dos Santos. – Rio de Janeiro: UFRJ/COPPE, 2020.

xvi, 180 p.: il.; 29,7 cm.

Orientadora: Cláudia Maria Lima Werner

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2020.

Referências Bibliográficas: p. 118-126.

1. Microsserviços. 2. Avaliação Multicritérios. 3. Criticidade. I. Werner, Cláudia Maria Lima. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

"Meus irmãos, considerem motivo de grande alegria o fato de passarem por diversas provações, pois vocês sabem que a prova da sua fé produz perseverança. E a perseverança deve ter ação completa, a fim de que vocês sejam maduros e íntegros, sem que falte a vocês coisa alguma."

Tiago 1:2-4

AGRADECIMENTOS

Dedico este trabalho a Deus, por proporcionar oportunidades e caminhos a trilhar, assim como meios para perseverar e seguir confiante na obtenção da vitória sob todas as adversidades apresentadas.

À minha esposa, Renata, pelo companheirismo, apoio e motivação desde o início desta jornada. Obrigado pela calma e compreensão durante este período.

À minha mãe, Fátima, pelas palavras sábias e direcionamento para a vida. Obrigado pelas conversas e por ser nosso porto-seguro.

Ao meu pai, Olavo, por sempre nos incentivar em todos os nossos sonhos e anseios, independentemente de o quão longe estivessem.

Ao meu irmão e irmãs, pelas conversas acerca da vida e seus diversos caminhos.

Aos meus demais parentes, que de certa forma me apoiaram nesta caminhada direta ou indiretamente. De forma especial gostaria de registrar um agradecimento às minhas tias Emília e Suely e ao meu tio Octacílio que sempre demonstraram apoio à família, mas que infelizmente os perdemos durante a realização deste trabalho.

Ao meu sogro e sogra, Mario e Jucy, por sempre estarem presentes em nossas vidas, auxiliando-nos e caminhando junto conosco, fazendo desta família uma unidade.

À minha orientadora, Cláudia Werner, pela oportunidade e direcionamento que recebi no mestrado. Obrigado pelos conselhos, confiança depositada ao longo dos trabalhos e ensinamentos que guiaram essa pesquisa. Espero ter contribuído face a confiança em mim depositada.

A orientação, mesmo que não oficial, do amigo Marcelo França, sou grato pela disponibilidade, discussões e questionamentos que auxiliaram nos desafios desta pesquisa.

Aos colegas do Programa de Engenharia de Sistemas e Computação (PESC) e todo o pessoal do grupo de Reutilização de Software (Diogo, Filipe, Susie, Diego, etc.), pela convivência e suporte sempre que necessário.

Aos professores Toacy Cavalcante e Leonardo Murta, por aceitarem participar da banca de defesa. A todos os professores da COPPE/UFRJ que tive o prazer de conhecer, e toda equipe técnico-administrativa pelo suporte necessário durante todo o mestrado.

Às empresas General Electric, Zoop Pagamentos, e por fim, a Philips Royal, na qual sou Arquiteto de Software Sênior, que em diferentes fases deste trabalho puderam

auxiliar no meu desenvolvimento pessoal e profissional servindo de base para esta pesquisa.

A todos que puderem participar dos estudos conduzidos durante a execução deste trabalho. A experiência prévia de vocês, bem como a satisfação em colaborar foi fundamental para o sucesso na condução desta pesquisa.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

SUGESTÃO DE CRITICIDADE BASEADA EM MULTICRITÉRIO PARA ARQUITETURAS ESTABELECIDAS ORIENTADAS A MICROSSERVIÇOS

Eduardo Fernandes Mioto de Oliveira dos Santos

Abril/2020

Orientadora: Cláudia Maria Lima Werner

Programa: Engenharia de Sistemas e Computação

A arquitetura de software orientada a microsserviços considera a separação de responsabilidades por componentes apartados, criando assim um conjunto de serviço interconectados. A fim de avaliar a criticidade em arquiteturas estabelecidas orientadas a microsserviços, a utilização de uma abordagem multicritérios é proposta por esta dissertação, possuindo por objetivo a sugestão dos microsserviços mais críticos para auxílio ao processo decisório de arquitetos de software e demais tomadores de decisão para investimento na manutenção ou evolução dos microsserviços em arquiteturas em uso. Vislumbrando maior estabilidade em arquiteturas estabelecidas, a abordagem proposta, com o apoio do método AHP, realiza uma avaliação dinâmica de múltiplos critérios com diferentes pesos de acordo com o perfil de usuário que fará uso da abordagem, tornando-a mais aderente a necessidade de diferentes perfis e agregando diferentes pontos de vista com o intuito de sugerir de forma mais assertiva os microsserviços mais críticos, delineando o foco para a evolução ou manutenção de microsserviços em arquiteturas estabelecidas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

MULTICRITERIA CRITICALITY SUGGESTION FOR ESTABLISHED MICROSERVICES ORIENTED ARCHITECTURES

Eduardo Fernandes Mioto de Oliveira dos Santos

April/2020

Advisors: Cláudia Maria Lima Werner

Department: Computer and Systems Engineering

The microservice-oriented software architecture considers the separation of responsibilities for separate components, thus creating a set of interconnected services. In order to assess the criticality in established architectures oriented to microservices, the use of a multicriteria approach is proposed by this dissertation, aiming at suggesting the most critical microservice to aid the decision process of software architects and other decision makers for investment maintenance or evolution of microservices in architectures in use. Intending to pursuit greater stability in established architectures, the proposed approach, supported by the AHP method, performs a dynamic evaluation of multiple criteria with different weights according to the user profile that will use the proposed approach, making it more adherent the need for different profiles and aggregating different points of view in order to more assertively suggest the most critical microservices, outlining the focus of the evolution or maintenance of microservices in established architectures.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ABREVIAÇÕES

AHP: *Analytic Hierarchy Process*
AMQP: *Advanced Message Queuing Protocol*
ANOVA: *Analysis of Variance*
API: *Application Programming Interface*
CEO: *Chief Executive Officer*
COTS: *Component Off-The-Shelf*
CPU: *Central Process Unit*
ESB: *Enterprise Service Bus*
FMECA: *Failure Modes Effects and Criticality Analysis*
HTTP: *Hypertext Transfer Protocol*
I/O: *Input/Output*
IEEE*: Institute of Electrical and Electronics Engineers*
ISO: *International Organization for Standardization*
K/V: *Key/Value*
M2CSEA: *Multicriteria Microservice Criticality Suggestion for Established Architectures*
MCDM: *Multi Criteria Decision Making*
MDS: *Multidimensional Scaling*
MSA: *Microservices Architecture*
NFP: *Non-functional Property*
QOC: *Question, Options, Criteria*
QoS: *Quality of Service*
RAM: *Random Access Memory*
RAML: *RESTful API Modeling Language*
REST: *Representational State Transfer*
RIM: *Reference Ideal Method*
RQF: Requisitos Funcionais
SOA: *Service Oriented Architecture*
SOAP: *Simple Object Access Protocol*
SOC: *Service Oriented Computing*
SQALE: *Software Quality Assessment based on Lifecycle Expectations*
TTM: Tempo de Transmissão da Mensagem
WSDL: *Web Services Description Language*

CAPÍTULO 1 – INTRODUÇÃO

1.1. Contexto

A arquitetura de software define os sistemas em termos de componentes e das interações entre estes, requerendo organização em termos de documentação e controle acerca das diversas escolhas a serem feitas, a fim de assegurar a correta implementação dos requisitos funcionais e não funcionais, verificando-se, por exemplo, protocolos de comunicação, sincronização e acesso a dados, atribuição de funcionalidades, distribuição física, dimensionamento e desempenho. A escolha arquitetural colabora em grande parte para o sucesso, ou não, de um sistema no que tange ao âmbito da tecnologia, devendo haver minucioso estudo a fim de averiguar necessidades e opções (GARLAN e SHAW, 1993).

Além de especificar a estrutura do sistema, a arquitetura demonstra a correspondência pretendida entre os requisitos do sistema e seus elementos após sua construção, podendo também conter propriedades a nível de sistema, como capacidade, taxa de transferência[1], consistência e compatibilidade entre componentes. Os modelos arquiteturais descrevem, por meio de documentação, as diferenças estruturais e semânticas entre componentes e suas interações. A definição de modelos arquiteturais pode ser utilizada para especificar um conjunto de sistemas, podendo os elementos serem definidos independentemente para que possam ser reutilizados em diferentes contextos (GARLAN e SHAW, 1993) (PERRY e WOLF, 1992).

O processo contínuo de evolução dos sistemas de software, com consequente aumento de tamanho e complexidade, torna a especificação arquitetural importante para a produção de software com qualidade. Desta forma, o papel da arquitetura de software é importante no desenvolvimento de software, pois serve como um plano de avaliação e implementação, tornando a escolha da correta arquitetura um problema crítico no domínio da engenharia de software (MOAVEN et al., 2008).

Durante as fases de concepção de um software é de grande importância o correto estudo e dimensionamento ao qual o sistema de propõe, com foco em promover a correta fundamentação acerca dos requisitos não funcionais, com a coerente definição da arquitetura frente a necessidade de negócio e adequação à estrutura que suportará a

[1] Taxa de transferência: Número médio de bits, caracteres ou blocos convertidos ou processados por unidade de tempo que passam entre equipamentos em um sistema de transmissão de dados.

utilização em larga escala e agregação de novas funcionalidades (PAPAZOGLOU et al., 2007).

Sistemas de larga escala demandam decisões arquiteturais que promovam a manutenibilidade de seus componentes. A utilização de arquiteturas monolíticas em sistemas deste porte pode promover um maior custo de manutenção por possuírem diversas responsabilidades contidas em si, com acúmulo de função, possuindo reduzida escalabilidade por ser necessário o escalonamento do sistema como um todo e não somente das funções mais utilizadas (NEWMAN, 2015).

Os sistemas monolíticos são comumente criados de forma a tornar a sua divisão complexa. Neste modelo arquitetural, as aplicações são implantadas em um único servidor ou replicadas por meio de um *cluster*[2] (STUBBS; MOREIRA; DOOLEY, 2015) (VILLAMIZAR et al., 2015).

O desenvolvimento de software orientado a microsserviços é uma maneira de projetar aplicações de software como grupos de serviços independentes, sendo estas aplicações coesas e de tamanho reduzido, podendo ser facilmente integradas com reduzido tempo de criação, validação e implantação. Os microsserviços podem ser atualizados e dimensionados individualmente, levando a uma maior estabilidade da arquitetura e maior resiliência (STUBBS; MOREIRA; DOOLEY, 2015).

1.2. Motivação

A necessidade de conhecer os microsserviços mais críticos advém da complexidade na tomada de decisão acerca da manutenção e evolução de arquiteturas baseadas neste modelo arquitetural. Após a implantação de uma arquitetura orientada a microsserviços, o processo de manutenção desta arquitetura tende a ser complexo, tendo em vista os múltiplos serviços a serem observados.

Diferentes partes interessadas, como arquitetos de software, gerentes de projeto, engenheiros de software, gerentes de infraestrutura, analistas de infraestrutura, dentre outros, possuem a necessidade de possuir dados de forma atualizada acerca de tais microsserviços, a fim de promover uma maior estabilidade da arquitetura, ou eventualmente a evolução dos microsserviços presentes nesta arquitetura.

A mudança para uma arquitetura de microsserviços é um assunto relevante atualmente, sendo possível observar diversas empresas motivadas e envolvidas em

[2] Cluster: conjunto de computadores interconectados que funcionam como um único sistema.

grandes refatorações de seus sistemas, quando aplicável, para acomodar as vantagens deste modelo arquitetural (DRAGONI et al., 2016).

1.3. Caracterização do Problema

Quando o sistema é construído como uma única aplicação, a compreensão do código é afetada na medida em que a aplicação possui mais linhas de código a serem interpretadas ou compiladas. A escolha por este modelo arquitetural causa maiores custos de manutenção e exige um maior tempo para especificar o sistema como um todo tendo em vista a quantidade de funções associadas a este, e por consequência haverá um maior tempo para codificar, gerando um aumento do *time-to-market*, ou seja, do lançamento do produto ao mercado consumidor (BALALAIE; HEYDARNOORI; JAMSHIDI, 2015).

O modelo de desenvolvimento de software orientado a microsserviços foi idealizado com o intuito de tratar a latente necessidade de entrega de valor ao negócio com redução de tempo de resposta ao mercado, associada à redução do custo de manutenção e evolução (BALALAIE; HEYDARNOORI; JAMSHIDI, 2015).

Para sistemas pouco complexos, a arquitetura monolítica pode ser a solução mais adequada, podendo ser escalada usando mecanismos simples de balanceamento de carga[3]. Entretanto, ao longo do tempo, problemas podem emergir, como por exemplo dificuldades no entendimento e manutenção do código, aumento do tempo de implantação e compromisso de longo prazo com as tecnologias utilizadas para desenvolvê-lo (BALALAIE; HEYDARNOORI; JAMSHIDI, 2015).

Embora o termo "decisão de *design* arquitetural" seja frequentemente usado (HOFMEISTER; NORD; SONI, 2000) (NORD et al., 2003), é difícil encontrar uma definição precisa. Portanto, neste trabalho, optou-se pela definição de *design* arquitetural segundo (JANSEN e BOSCH, 2005), que propõe que esta seja uma descrição do conjunto de adições arquiteturais, subtrações e modificações na arquitetura de software, considerando as regras e restrições de *design*, bem como os requisitos adicionais que, parcialmente, podem realizar um ou mais funções em uma determinada arquitetura.

Uma decisão de *design* arquitetural é, portanto, o resultado de um processo de *design* durante a construção inicial, correção ou a evolução de um sistema. As decisões

[3] Balanceamento de carga: técnica para distribuir a carga de trabalho entre dois ou mais computadores em rede.

de *design* arquitetural podem estar relacionadas ao domínio do sistema, aos estilos e padrões arquiteturais usados no sistema, componentes e outras seleções de infraestrutura, além de outros aspectos necessários para satisfazer os requisitos do sistema. Nesta perspectiva, a arquitetura de software é o resultado das decisões de *design* arquitetural feitas ao longo do tempo (JANSEN e BOSCH, 2005).

As decisões de *design* são muitas vezes interligadas entre si, pois trabalham em estreita relação. Além disso, comumente as decisões tomadas afetam por consequência outras partes da arquitetura. Isso leva à situação em que as informações de decisão de *design* são fragmentadas em várias partes do *design*, dificultando a localização e a alteração das decisões. Estes efeitos aumentam a complexidade da arquitetura de software à medida que são introduzidos numerosos relacionamentos aparentemente não interligados, por exemplo, as dependências entre entidades de uma arquitetura (JANSEN e BOSCH, 2005).

Arquiteturas estabelecidas são especificadas, desenvolvidas, implementadas e em uso, o que requer atenção especial em relação aos requisitos não funcionais, considerando que o sistema já está em uso e mudanças em sua estabilidade pode significar uma redução na qualidade do serviço. O conhecimento acerca da criticidade em uma arquitetura estabelecida pode possuir um impacto significativo na confiança dos tomadores de decisão para promover manutenção ou evolução (SANTOS e WERNER, 2019).

Para a contextualização acerca do conceito de criticidade, foram observados trabalhos de outras engenharias além da computação. Em engenharia, o princípio geral dos efeitos dos modos de falha e análises de criticidade (FMECA) é analisar o comportamento das partes do sistema na presença de falhas de seus vários componentes. FMECA é uma abordagem industrial padrão para gerenciar a confiabilidade de sistemas críticos durante seus estágios de *design* (CAYRAC; DUBOIS; PRADE, 1996).

Dado que os modos de falha e seus efeitos são identificados, uma análise de criticidade é usada para classificar cada modo de falha em potencial de acordo com a influência combinada de gravidade e probabilidade de ocorrência. (BATZEL e SWANSON, 2009). Esta abordagem é comumente utilizada para a avaliação de sistemas críticos em engenharia para aviação civil (BATZEL e SWANSON, 2009) e aeroespacial (CAYRAC; DUBOIS; PRADE, 1996).

De forma similar, o conceito de criticidade foi utilizado neste trabalho para avaliar os componentes, no contexto em questão sendo tratados como microsserviços,

com o intuito de avaliar a introdução de uma falha na arquitetura, gerando uma possível indisponibilidade ou degradação do serviço.

Dessa forma, com base na problemática descrita, esta dissertação de mestrado foca no problema de tomada de decisão observado pelos arquitetos de software e demais tomadores de decisão durante a manutenção e evolução de arquiteturas estabelecidas orientadas a microsserviços, visando apoiar a escolha do microsserviço a ser trabalhado. Neste ponto, surge a seguinte questão de pesquisa: *Como sugerir os microsserviços mais críticos em arquiteturas de software orientadas a microsserviços para apoiar a tomada de decisão para a manutenção e evolução de arquiteturas estabelecidas?*

1.4. Objetivo de Pesquisa

Este trabalho tem por objetivo o estudo sobre a sugestão dos microsserviços mais críticos por meio da avaliação multicritério de criticidade em arquiteturas orientadas a microsserviços, vislumbrando propor um método de avaliação de arquiteturas estabelecidas que sirva de apoio a decisões arquiteturais baseado em multicritérios.

De modo a alcançar o objetivo principal desta dissertação, foram estabelecidos os seguintes objetivos específicos:

- Identificar os atuais modelos de sugestão de microsserviços mais críticos dentro de uma arquitetura orientada a microsserviços. Este objetivo de pesquisa busca ilustrar os atuais modelos de sugestão de criticidade de microsserviços em arquiteturas estabelecidas, com o intuito de fornecer a base teórica para a fundamentação da proposta de uma abordagem de sugestão de criticidade em arquiteturas orientadas a microsserviços;
- Conduzir uma avaliação com especialistas da academia e indústria para validação da necessidade de possuir um modelo de avaliação de criticidade com a posterior sugestão dos microsserviços mais críticos em arquiteturas estabelecidas orientadas a microsserviços;
- Especificar uma abordagem para sugestão dos microsserviços mais críticos baseado em múltiplos critérios focados no suporte à tomada de decisão visando o auxílio a manutenção e evolução de arquiteturas estabelecidas;
- Especificar e implementar um ferramental de suporte que ofereça apoio para a sugestão dos microsserviços mais críticos;

- Avaliar o ferramental de suporte desenvolvido, visando caracterizar sua aceitação no cenário de sugestão de microsserviços críticos baseado em múltiplos critérios em arquiteturas estabelecidas.

O presente trabalho limita-se ao estudo da sugestão para auxílio à tomada de decisão em arquiteturas orientadas a microsserviços, não objetivando reduzir a importância das demais arquiteturas, ou mesmo evidenciar a melhor opção em casos gerais, havendo a necessidade da análise minuciosa caso a caso, a fim de uma tomada de decisão consciente quanto a melhor opção para cada problema apresentado.

1.5. Metodologia de Pesquisa

A metodologia de pesquisa aplicada no desenvolvimento deste trabalho é apresentada na Figura 1.1, e compõe-se por três fases: I. Concepção, focada na condução de uma revisão sistemática da literatura para obter insumos acerca do estado da arte de criticidade em arquiteturas orientadas a microsserviços, bem como na validação com especialistas acerca da avaliação de criticidade e na definição da abordagem M2CSEA proposta por este trabalho para colaborar com a sugestão de microsserviços mais críticos em arquiteturas estabelecidas; II. Implementação, que visa o desenvolvimento do ferramental de suporte por meio da implementação da abordagem proposta; e III. Avaliação, focada na realização de um estudo de caso para mensurar a eficiência do ferramental de suporte implementado. No total, cinco etapas foram realizadas nestas três fases, conforme descrito a seguir.

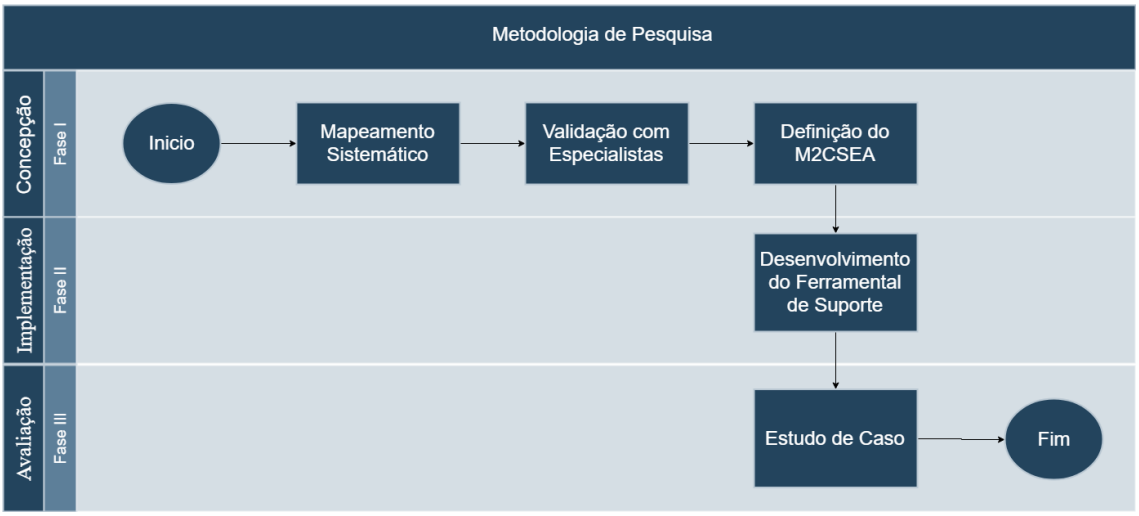


Figura 1.1 – Metodologia de pesquisa

1.6. Estrutura da Dissertação

Esta dissertação está organizada em sete capítulos. Esse capítulo apresentou o contexto e a motivação para realização desta pesquisa, bem como a caracterização do problema e a questão de pesquisa. Os demais capítulos estão organizados conforme a estrutura a seguir:

- **Capítulo 2 – Fundamentação Teórica:** neste capítulo, são tratados conceitos acerca da evolução da arquitetura de software, considerando a qualidade neste contexto, a avaliação baseada em multicritérios, os modelos de seleção, os modelos de avaliação multicritério e a criticidade em arquiteturas orientadas a microsserviços.

- **Capítulo 3 – Validação com Especialistas:** neste capítulo, são tratados pontos relacionados a validação da problemática em questão com especialistas, considerando os objetivos desta validação, as hipóteses, as definição do estudo, a seleção e arranjo dos participantes, a execução dos estudo, os resultados, as observações, a interpretação dos resultados, as ameaças a validade do estudo e conclusão.

- **Capítulo 4 – A Abordagem M2CSEA:** neste capítulo, são tratados pontos acerca da abordagem proposta, caracterização de criticidade em microsserviço, da qualidade em uso segundo ISO 25010:2011, seleção dos múltiplos critérios de seleção, utilização do método AHP, indicação e seleção de critérios de criticidade, análise da interdependência entre microsserviços, análise do tempo de resposta, análise da relevância para o negócio, análise da cobertura de testes unitários, análise do débito técnico, análise da utilização de recursos computacionais e agregação das análises.

- **Capítulo 5 - Implementação:** neste capítulo, serão tratados pontos acerca da implementação da proposta, vislumbrando requisitos, viabilidade tecnológica, a criação do protótipo e exemplo de uso do ferramental desenvolvido.

- **Capítulo 6 – Avaliação da Proposta:** neste capítulo, são tratados pontos da avaliação da abordagem delineada neste trabalho, considerando objetivos, hipóteses, as definição do estudo para avaliação da abordagem, a seleção e arranjo dos participantes, a execução dos estudo, os resultado obtidos, a interpretação dos resultados, as ameaças a validade e conclusão acerca do estudo.

- **Capítulo 7 - Conclusões:** neste capítulo, são tratadas as conclusões do presente trabalho com o epílogo, as contribuições do trabalho à literatura, limitações do trabalho, e os trabalhos futuros.

CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA

2.1. Introdução

A arquitetura de software (PERRY e WOLF, 1992) tornou-se um conceito amplamente aceito na academia e na indústria. A importância de documentar os componentes e os conectores de um sistema é reconhecida e promove um melhor controle sobre o *design*, o desenvolvimento e a evolução de sistemas (CLEMENTS et al., 2002) (JANSEN e BOSCH, 2005).

Embora as conquistas da arquitetura de software sejam de grande valor, ainda persistem alguns problemas. A complexidade, os altos custos de mudança e a degradação do *design* ao passar do tempo são alguns dos problemas fundamentais da arquitetura de software. Atualmente, grande parte do conhecimento e informações sobre as decisões de *design* nas quais a arquitetura é baseada, como por exemplo, resultados de análise de domínio, estilos arquiteturais usados, *trade-off* e outros, estão implicitamente incorporados na arquitetura (JANSEN e BOSCH, 2005).

Durante a evolução do sistema, os arquitetos podem violar regras de *design* e restrições decorrentes de decisões de projeto tomadas anteriormente. Violações dessas regras levam a um desvio arquitetural (PERRY e WOLF, 1992) e geram problemas associados, como por exemplo, o aumento dos custos de manutenção no decorrer do tempo (JANSEN e BOSCH, 2005). Como resultado desses problemas, os sistemas desenvolvidos têm um alto custo de mudança e tendem a se desgastar rapidamente, deixando de atender aos requisitos a que foram propostos (JANSEN e BOSCH, 2005).

Neste capítulo, será feita a fundamentação teórica acerca da evolução dos modelos de arquitetura de software associando-os aos modelos de qualidade de software. De forma contextualizada serão introduzidos os modelos de avaliação multicritério, os modelos de seleção e modelos de avaliação de criticidade.

A fim de promover esta contextualização teórica, além desta seção introdutória, este capítulo está organizado com a apresentação da evolução dos modelos de arquitetura de software na Seção 2.2. Na Seção 2.3, são detalhados pontos acerca da qualidade em arquiteturas de software. Na Seção 2.4 são expostos os modelos de avaliação multicritério. Na Seção 2.5, são apresentados os modelos de seleção. Por fim, as considerações finais são apresentadas na Seção 2.6.

2.2. A Evolução dos Modelos de Arquitetura de Software

Nesta seção, será apresentada uma breve introdução acerca da evolução dos modelos de arquitetura de software.

2.2.1. Arquitetura de Software Orientada a Serviços

Com a evolução das arquiteturas de software, após a consolidação e declínio das arquiteturas monolíticas, surgiu a arquitetura em N-camadas, na qual é possível delegar responsabilidades a serviços externos, e possuir um centralizador para controle do sistema como um todo. Este modelo promoveu a concepção de novos modelos arquiteturais, como sistemas baseados em componentes e, posteriormente, a arquitetura orientada a serviços (SOA) (LIU, 2011).

A computação orientada a serviços promove a ideia de construir componentes para aplicações em uma rede de serviços que pode ser fracamente acoplada para criar processos de negócios flexíveis e dinâmicos, com aplicações ágeis que abrangem organizações e plataformas de computação. O paradigma de computação orientada a serviços usa serviços para suportar o desenvolvimento de aplicações rapidamente, de baixo custo, interoperáveis, evoluídos e distribuídos. Os serviços são autônomos, independentes de plataforma, podendo ser descritos, publicados, descobertos e fracamente acoplados (PAPAZOGLOU et al., 2007).

A arquitetura orientada a serviços (SOA) entre os anos 2000 a 2010 foi uma das abordagens arquiteturais mais promissoras (BALALAIE; HEYDARNOORI; JAMSHIDI, 2016). A arquitetura orientada a serviços usa a rede como meio de comunicação e padrões abertos, incluindo o protocolo SOAP para transmissão de dados e a especificação WSDL para definir serviços (PAPAZOGLOU et al., 2007).

A arquitetura SOA possui diversos benefícios, tais como a capacidade de possuir um centralizador de comunicação com capacidade de garantia de entrega, fazendo uso de um barramento de comunicação (ESB), bem como a extração de funcionalidades para serviços definidos. Entretanto, há pontos de melhoria, como o custo em termos de consumo de recursos computacionais e tráfego de rede para o envio e recebimento de uma mensagem SOAP, tendo em vista as implementações mandatórias relativas ao protocolo, como por exemplo a necessidade de possuir o envelope SOAP e o tamanho do cabeçalho do protocolo.

Quanto ao tráfego de rede, este é de duas a três vezes menor quando se usa uma interface REST (BELQASMI et al., 2012). O REST é uma abordagem arquitetural para aplicações distribuídas, usada principalmente para criar serviços de fácil manutenção e escaláveis. A abordagem REST não depende de nenhum protocolo, mas quase todos os serviços REST usam HTTP como seu protocolo subjacente (DAYA, S. et al.: 2016).

A arquitetura de microsserviços surgiu como uma evolução da Arquitetura Orientada a Serviços (SOA) e, mesmo considerando alguns aspectos semelhantes, é necessária uma modificação cultural nas empresas para entregar os benefícios mais relevantes propostos pela arquitetura orientada a microsserviços, como por exemplo a redução *do time-to-market,* facilidade de manutenção e evolução, utilização das linguagens de programação mais adequadas de acordo com os requisitos e maior escalabilidade (MAHMOOD, 2008) (ALPERS et al., 2015).

2.2.2. Arquitetura de Software Orientada a Microsserviços

Microsserviços são serviços autônomos e independentes, de tamanho reduzido, com responsabilidades específicas e bem definidas que podem trabalhar em conjunto, mas não possuem interdependência. O uso desta abordagem proporciona liberdade de escolha tecnológica e velocidade na implementação de novas funcionalidades a fim de atender a novas demandas de negócio, estando alinhados às necessidades reais das empresas na economia atual (NEWMAN, 2015). Embora ainda exista pouca pesquisa no campo de microsserviços, este é um tópico de interesse para arquitetos, pois pode auxiliá-los na tomada de decisão (FRANÇA e WERNER, 2018).

Durante a fase de especificação do *design* arquitetural, é decidido o número e o tamanho de microsserviços de forma individuais, exigindo conhecimento de domínio e arquitetura. Um serviço com diferentes funcionalidades não associadas pode precisar ser dividido para melhorar sua testabilidade, manutenibilidade e implantação (DAYA et al., 2016).

De forma antagônica aos sistemas monolíticos, microsserviços possuem suas responsabilidades delineadas, restritas e são escaláveis, sendo também possível o uso de recursos computacionais de menor capacidade e melhor aproveitamento dos recursos computacionais disponíveis (NEWMAN, 2015).

A divisão de responsabilidades pode seguir diversas premissas, dentre as quais podemos destacar o princípio da única responsabilidade, no qual propõe-se manter

juntas unidades que sejam alteradas pela mesma razão, e manter separado unidades que sejam alteradas por razões diferentes (MARTIN, 2003).

O estudo de arquiteturas em concordância às necessidades econômicas sempre esteve em voga, entretanto, a escalabilidade é um dos principais fatores a serem considerados na implementação de um software para demandas atuais (SPINELLIS e GOUSIOS, 2009). Esta é uma característica de sistemas robustos o suficiente para suportar maiores taxas de acesso simultâneos, sem uso improdutivo de recursos, de modo linear, acompanhando o aumento de capacidade computacional, sem perda de eficiência e qualidade nos serviços (BONDI, 2000) (BRATAAS e HUGHES, 2004) (JOGALEKAR e WOODSIDE, 2000) (MESSERSCHMITT, 1996).

Desta forma, sistemas críticos necessitam de uma arquitetura escalável, mas sem prejuízo de manutenibilidade, pois há um grande risco associado ao custo futuro de manutenção por investimento indevido de tempo e/ou dinheiro na fase de estudo e criação do software, tornando-o complexo e necessitando de demasiado tempo para eventuais evoluções ou correções de não conformidades.

Vislumbrando uma implementação com alto nível da escalabilidade, confiabilidade, manutenibilidade, e por consequência disponibilidade, a abordagem de criação de microsserviços é interessante e sua implementação merece ser discutida mais a fundo.

Para a construção de um sistema escalável, não basta somente que hajam servidores capazes de suportar uma maior demanda de usuários, mas sim uma arquitetura de software capaz de suportar adequadamente e com o mínimo possível de falhas, um volume variável de acessos, havendo, portanto, um aumento da confiabilidade atribuída a este sistema (SPINELLIS e GOUSIOS, 2009) (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Por razões econômicas, um sistema escalável deve ser capaz de, havendo poucos usuários, demandar pouco recurso de hardware e, havendo uma maior demanda, ser capaz de possuir elasticidade suficiente para suportar a demanda e entregar ao usuário requerente a resposta adequada dentro de um tempo razoável, garantindo assim requisitos de desempenho.

Afirma-se que para a implantação adequada de uma arquitetura de software escalável é necessário avaliar o ponto com maior possibilidade de falhas ou enfileiramento, sendo este um possível causador de interrupção do serviço, dependendo da quantidade e frequência do enfileiramento. Desta forma, é crucial a verificação e

estudo minucioso de todos os serviços e servidores que compõem esta arquitetura, não somente focando em determinados aspectos isolados de um sistema.

A complexidade de um sistema aumenta progressivamente e com maior dificuldade de redução ao longo do tempo, pois novos módulos, funções e dependências são incluídas neste, tornando-o cada vez maior. Desta forma, com a intenção de atender a diferentes necessidades de negócio, por meio da implementação de novas funcionalidades, o sistema torna-se menos escalável com o passar do tempo.

Tendo em vista a necessidade de manutenção corretiva ou evolução de um sistema, é necessário que haja cautela quanto ao a escolha da arquitetura adequada nas etapas de estudo e implementação, pois caso a execução destas seja de modo inadequado novos problemas serão introduzidos, criando um ciclo danoso à qualidade do sistema (PRESSMAN, 2005).

Partindo do princípio da necessidade de cautela na manutenção dos sistemas, a redução de escopo promovido pela divisão de responsabilidades associada a arquitetura orientada a microsserviços propõe um menor campo de atuação para a solução de um problema, pois havendo serviços separados por responsabilidade haverá uma atuação em uma quantidade reduzida de códigos-fonte a serem afetados.

Possuindo um campo restrito de atuação, melhores práticas podem ser adotadas com menor esforço de implementação, como por exemplo, o aumento da coesão e redução do acoplamento, tendo em vista a relevância destas boas práticas para redução do tempo de manutenção e risco de criação de novas inconsistências por correções incoerentes (TRIPATHY e NAIK, 2014), bem como a redução da complexidade para que haja entregas frequentes de código-fonte, agregando valor ao negócio em curto prazo, pois as evoluções e correções serão feitas somente no escopo do serviço afetado pela demanda de negócio (HUMBLE e FARLEY, 2010).

2.3. Qualidade em Arquitetura de Software

Um dos propósitos do uso dos métodos de análise de arquiteturas de software é reduzir os custos causados por correções e aumentar a qualidade dos produtos, sendo qualidade e sua manutenção um dos principais problemas no desenvolvimento de sistemas de software. A ideia de prever a qualidade de um produto de software a partir de uma descrição arquitetural não é nova. Em 1972, PARNAS (1972) descreveu o uso de modularização como meio de decomposição de sistemas de alto nível para melhorar

a flexibilidade e a inteligibilidade. STEVENS, MYERS e CONSTANTINE (1974) introduziram as noções de acoplamento e coesão em módulos para avaliar alternativas para a decomposição do programa.

Nos últimos anos, a noção de arquitetura de software evoluiu para tornar-se fundamental no processo de definição e manutenção da qualidade do software. Esta evolução ocorreu, pois, as comunidades científica e industrial reconheceram que a arquitetura de software estabelece limites para a qualidade de software nos sistemas desenvolvidos (CLEMENTS et al., 2002) (DOBRICA e NIEMELA, 2002).

Esforços recentes para a sistematização das implicações do uso de padrões de projeto e estilos arquiteturais contribuem, frequentemente de maneira informal, para a garantia da qualidade de um projeto. Reconhece-se que não é possível medir os atributos de qualidade do sistema final com base somente na especificação da arquitetura de software. Isso implicaria em especificar, implementar e manter uma arquitetura de software de forma a não haver nenhum desvio, e ainda manter esta arquitetura em alinhamento com os requisitos não funcionais (BUSCHMANN, 1996) (GAMMA, 1995).

A análise de arquiteturas de softwares auxiliar a prever a qualidade de um sistema verificando os principais efeitos de mudança nesta arquitetura (KAZMAN et al., 1998). O objetivo da avaliação é verificar a arquitetura de software para identificar os riscos potenciais e validar se os requisitos de qualidade foram abordados no projeto (LI e HENRY, 1993) (DOBRICA e NIEMELA, 2002).

Um atributo de qualidade é uma característica não funcional de um componente ou sistema. A qualidade de software foi definida na IEEE 1061 (IEEE:1993) e representa o grau em que o software possui uma combinação desejada de atributos. A manutenção envolve um conjunto de atributos que influenciam o esforço para fazer modificações (NORMALIZACIÓN, 2011). As modificações podem incluir correções, melhorias ou adaptações de software para mudanças de contexto, requisitos e especificações funcionais.

2.3.1. Métricas de Qualidade em Arquitetura de Software

A especificação de métricas deve conter a medida selecionada para um atributo de qualidade, uma escala de medida e um conjunto de métodos para medição, desta

forma duas abordagens podem ser identificadas: adaptar métricas existentes ou definir novas métricas (BENGTSSON, 1998) (DUEÑAS; DE OLIVEIRA; JUAN, 1998).

Duas classes básicas de técnicas de avaliação, a técnica de questionamento e a técnica de medição, disponíveis no nível de arquitetura, são definidas em dois importantes relatórios de pesquisa, a saber: (ABOWD et al., 1997) e (CLEMENTS et al., 2002).

As técnicas de questionamento geram perguntas qualitativas a serem feitas sobre uma arquitetura e podem ser aplicadas para diversos atributos de qualidade. Esta técnica inclui cenários, questionários e listas de verificação.

As técnicas de medição sugerem medidas quantitativas a serem feitas em uma arquitetura, elas são usadas para responder perguntas específicas e abordam as qualidades de software específicas. Esta técnica inclui métricas, simulações, protótipos e experiências. A generalidade, o nível de detalhamento, a fase e o que é avaliado representam uma estrutura em quatro dimensões de comparação dessas técnicas (BASS; CLEMENTS; KAZMAN, 2003).

Quanto à generalidade, as técnicas podem ser gerais (questionário), específicas do domínio (listas de verificação, protótipo) ou específicas do sistema (cenários). O nível de detalhe (granularidade grossa, média ou fina) indica a quantidade de informações sobre a arquitetura que são necessárias para executar a avaliação.

Existem três fases de interesse para avaliação de arquitetura: início, meio e pós-uso. Essas fases ocorrem após as decisões iniciais de arquitetura de alto nível (questionário, protótipo), a qualquer momento após a elaboração do *design* da arquitetura (cenários, listas de verificação) e após o sistema ter sido completamente projetado, implementado e implantado (DOBRICA e NIEMELA, 2002).

Vários modelos de análise expressos em métodos formais são incluídos em técnicas quantitativas, de outra forma, as técnicas qualitativas ilustram avaliações de arquitetura de software usando cenários. Uma descrição das alterações necessárias para cada cenário representa um método qualitativo de avaliação (DOBRICA e NIEMELA, 2002).

Nessa perspectiva, os cenários são necessários, mas não suficientes para prever e controlar os atributos de qualidade, e devem ser complementados com outras técnicas de avaliação, particularmente, com interpretações quantitativas. Por exemplo, incluir perguntas sobre indicadores de qualidade nos cenários enriquece a avaliação da arquitetura. Interpretações quantitativas de avaliação de cenários podem ser

classificadas entre os efeitos dos cenários (*i.e.*, uma escala de cinco níveis de Likert) ou uma declaração, que estima o tamanho das modificações ou métricas diferentes, como linhas de código, pontos de função ou pontos de objeto (DOBRICA e NIEMELA, 2002).

As diferentes funções representam os interesses das partes interessadas relacionadas a um sistema, podendo as partes interessadas ser o usuário final, o administrador do sistema, o analista de negócios, ou qualquer interessado que de alguma forma possa influenciar a inclusão de novos requisitos à ao sistema em questão (DOBRICA e NIEMELA, 2002).

O cálculo da eficácia e satisfação de cada atributo de qualidade e a agregação destes podem nortear o problema de seleção do modelo de arquitetura ideal, pois a seleção de arquitetura de software é tipicamente um problema de tomada de decisão multicritério, no qual diferentes metas e objetivos devem ser levados em consideração, sendo apurados e elencados de acordo com critérios previamente estabelecidos, gerando sugestões por meio destes critérios, ordenado por um dado peso associado a cada um dos critérios.

A análise multicritério possibilita a comparação de critérios quantitativos e qualitativos (DYSON, 2003), sendo estes uma alternativa de estudo de um problema (MORAES e SANTALIESTRA, 2008), pois as questões propostas levam o tomador de decisão a confrontar diversos tipos de critérios, desde o custo de aquisição e risco, até a usabilidade, flexibilidade e confiabilidade.

2.3.2. Modelos de Qualidade de Software

De acordo com IZURIETA, GRIFFITH e HUVAERE (2017) os modelos de qualidade fornecem referências com as quais os componentes de software podem ser medidos. Modelos teóricos, como a especificação ISO/IEC 25010 e sua predecessora ISO/IEC 9126-1, oferecem diretrizes em muitas dimensões de qualidade que devem ser operacionalizadas para fornecer uma solução que possa ser usada pela comunidade de engenharia.

O escopo de aplicação dos modelos de qualidade inclui suporte à especificação e avaliação de software em diferentes perspectivas associadas à aquisição, requisitos, desenvolvimento, uso, avaliação, suporte, manutenção, controle e garantia de qualidade. Os modelos podem, por exemplo, ser usados por desenvolvedores, equipes de controle e garantia de qualidade e avaliadores independentes, particularmente aqueles responsáveis

por especificar e avaliar a qualidade do produto de software (NORMALIZACIÓN, 2011).

Os modelos de qualidade fornecem *benchmarks*[4] em relação aos quais os componentes de software podem ser medidos. Modelos teóricos, como as especificações ISO, oferecem diretrizes de dimensões variáveis de qualidade que devem ser operacionalizadas para fornecer uma solução funcional que possa ser usada pela comunidade de engenharia (IZURIETA; GRIFFITH; HUVAERE, 2017). No Apêndice A, são apresentados detalhes acerca das características e subcaracterísticas propostas pela norma ISO 25010:2011.

Para o presente trabalho, optou-se por seguir os modelos utilizados por IZURIETA, GRIFFITH e HUVAERE (2017), os quais propuseram a utilização dos modelos SQALE e Quamoco, pois estes se baseiam nos modelos teóricos da ISO e possuem consagração no meio industrial por sua implementação em ferramentas de análise estática de código-fonte.

Os modelos Quamoco e SQALE são modelos hierárquicos, estes vinculam não conformidades encontradas no software a aspectos de qualidade. Ambos os modelos usam essas informações como um meio para avaliar a qualidade de um componente de software.

O modelo SQALE foi desenvolvido em 2010 pela empresa Inspearit, anteriormente denominada DNV ITGS, para medir e gerenciar de forma objetiva a qualidade do código-fonte (LETOUZEY e ILKIEWICZ, 2012). O modelo SQALE foi projetado para ser o mais genérico possível, sendo possível aplicá-lo a diversas linguagens de programação. O método SQALE foi publicado sob a licença de código aberto e é gratuito (HEGEMAN, 2011).

A abordagem do modelo SQALE é baseada em oito características de código que são organizadas de forma piramidal. Na parte inferior da pirâmide está a testabilidade, seguida acima pela confiabilidade, capacidade de mudança, eficiência, segurança, facilidade de manutenção, portabilidade e, no topo da pirâmide, a reutilização. Se mais de uma característica for afetada por um requisito de qualidade, será formada uma associação com a característica mais baixa. As características na parte inferior da pirâmide representam dimensões de qualidade mais importantes e destinam-

[4] *Benchmark*: Em computação, *benchmark* é o ato de executar um programa de computador, um conjunto de programas ou outras operações, a fim de avaliar o desempenho relativo de um objeto, normalmente executando uma série de testes e ensaios.

se a auxiliar os profissionais na priorização de requisitos que precisam de refatoração na base de código. (IZURIETA; GRIFFITH; HUVAERE, 2017).

O modelo de qualidade Quamoco foi desenvolvido explicitamente como um metamodelo extensível. Seu objetivo era preencher uma lacuna entre conceitos abstratos e atributos mensuráveis. O conceito central do modelo é baseado em fatores, sendo estes destinados a representar um atributo ou propriedade de uma entidade. Existem dois tipos de fatores: aspectos de qualidade e fatores de produto. O primeiro representa as qualidades mais abstratas encontradas em modelos teóricos, como os padrões ISO. Este último representa as partes mensuráveis de um componente de software e tem impacto no aspecto de qualidade associado. Fatores formam hierarquias, onde podem refinar ainda mais algum aspecto da qualidade (IZURIETA; GRIFFITH; HUVAERE, 2017).

Para melhorar a modularização, o metamodelo é dividido em módulos, onde o módulo raiz contém hierarquias gerais de qualidade e fatores básicos do produto. Isso permite que os profissionais estendam o módulo raiz para fins ou tecnologias específicas e se concentrem nas qualidades relevantes para suas necessidades específicas. O modelo base do Quamoco é o resultado de muitos anos de colaboração de especialistas em qualidade da indústria e da academia e é composto por um conjunto abrangente de fatores e medidas que capturam a avaliação da qualidade do software (IZURIETA; GRIFFITH; HUVAERE, 2017).

2.3.2.1. Qualidade de Software e Débito Técnico

Na última década, a comunidade de pesquisa observou como o débito técnico se tornou uma abordagem popular para rastrear o progresso do desenvolvimento do código-fonte, apontando não conformidades que precisam ser refatoradas (MARINESCU, 2012) (DE GROOT et al., 2012) (FALESSI e VOEGELE, 2015).

Segundo MARINESCU (2012), as restrições de prazo de projeto para entrega podem causar problemas posteriores, levando a mudanças de *design* que geralmente causam falhas na estrutura de um sistema. Desta forma, os custos de manutenção aumentam significativamente. O impacto das decisões de projeto que fornecem benefícios de curto prazo às custas da integridade do sistema, é geralmente chamado de débito técnico.

A nova definição de débito técnico afirma explicitamente que “o débito técnico é um passivo contingente cujo impacto é limitado às qualidades internas do sistema,

principalmente a manutenção e a capacidade de evoluir" (AVGERIOU et al., 2016). Segundo IZURIETA, GRIFFITH e HUVAERE (2017), embora o foco da definição esteja em apenas um aspecto das muitas dimensões que compõem os modelos de qualidade baseados nas ISO/IEC 25010:2011 e ISO/IEC 9126-1, é importante mencionar que o método SQALE para gerenciar débitos técnicos associa custos de correção que afetam o índice de débitos técnicos de um sistema usando uma função de remediação que leva em consideração todas as dimensões da qualidade, não apenas a manutenção (LETOUZEY e ILKIEWICZ, 2012).

2.4. Modelos de Avaliação Multicritério

O método *Analytic Hierarchy Process* (AHP), desenvolvido por SAATY (1980), consiste num conjunto de passos no qual todas as combinações de critérios organizadas em uma matriz são avaliadas em comparações par a par. A meta é determinar a importância relativa de cada alternativa em relação aos critérios selecionados para a avaliação. Devemos lembrar que estas importâncias serão determinadas pelas pessoas envolvidas no processo de decisão, ou seja, elas vão usar conhecimento próprio para fazer os julgamentos.

O surgimento do AHP levou a uma melhoria considerável do processo de escolha da arquitetura e a posterior formalização para a automação dos processos de tomada de decisão (KHARCHENKO; HALAY; BODNARCHUK, 2016) (SAATY, 1980).

Motivado pela necessidade de se obter um maior conhecimento sobre a variedade de sistemas que compõem uma arquitetura, este trabalho propõe a utilização do uso de múltiplos critérios de análise para proposição dos microsserviços mais críticos em uma arquitetura. A proposta apresentada nesta dissertação converge diversos critérios de avaliação para orientar a sugestão por criticidade, e utiliza o método AHP para ponderação acerca do microsserviços de maior criticidade, dados os insumos oferecidos pelo modelo e suas verificações.

Quanto a utilização da abordagem de seleção e agregação baseado em multicritérios, os trabalhos de FAKHFAKH, VERJUS e POURRAZ (2011) e ZAYARAZ, VIJAYALAKSHMI e VIJAYALAKSHMI (2009) trabalham com métodos formais de avaliação com multicritérios. O presente trabalho e estes se assemelham no que tange a opção pelo uso de atributos de qualidade de software para avaliação e seleção. Entretanto, diferem desta dissertação na medida em que não estão diretamente

relacionados com a avaliação de criticidade em arquiteturas estabelecidas orientadas a microsserviço.

De acordo com CHYBOWSKI e GAWDZINSKA (2016), o método AHP pode ser aplicado na análise de importância de componentes para a estrutura de confiabilidade do sistema, usando-o em duas etapas:

1. Determinação dos critérios de importância dos componentes e suas relações em termos de capacidade de quantificação. Isso permite construir um modelo multicritério de importância dos componentes do sistema com coeficientes de peso calculados para os critérios de confiabilidade, segurança, custos de reparo do componente.
2. Determinação da relação de importância entre os componentes do sistema de acordo com todos os critérios analisados, o que permite obter a classificação final de importância dos componentes do sistema.

O princípio do AHP é separar os objetos complexos do sistema em um único elemento (critério) de acordo com os objetivos da pesquisa e, então, estabelecer uma estrutura de árvore ponderada de acordo com os relacionamentos entre os elementos, o que é chamado de árvore hierárquica. Em seguida, estabelecem-se prioridades entre os elementos (critério) da mesma hierarquia, fazendo uma série de julgamentos com base em comparações de pares dos elementos e sintetiza-se esses julgamentos para produzir um conjunto de prioridades gerais para a hierarquia. Finalmente, chega-se a uma decisão final, consolidando os pesos e avaliações dos especialistas.

2.5. Modelos de Seleção

HASELBÖCK, WEINREICH e BUCHGEHER (2017) investigaram o uso de modelos de decisão para arquitetura de microsserviço. Como primeiro passo, foram identificados os domínios dos microsserviços e foram criaram modelos de decisão para alguns destes domínios. Os resultados indicam que os profissionais perceberam que os modelos de decisão para microsserviços são úteis. Os desafios incluíram um grande número de áreas de conhecimento interligadas, a necessidade de adaptações específicas do contexto e a necessidade de processos para gerenciar a tomada de decisão ao longo do tempo. Para lidar com a complexidade da orientação para tomada de decisão, os

autores investigam o uso de modelos de decisão para apoiar a definição de uma arquitetura de microsserviço.

Os modelos de decisão são uma abordagem bem conhecida para explorar a modelagem de domínio, documentação e reutilização na arquitetura de software. Os autores desenvolveram modelos de decisão para uma arquitetura de microsserviços utilizando especificamente como base o metamodelo de LEWIS, LAGO e AVGERIOU (2016).

O metamodelo proposto por HASELBÖCK, WEINREICH e BUCHGEHER (2017) baseia seus questionamentos na notação de pergunta, opção e critério (*Question, Options, Criteria* ou QOC) (MACLEAN et al., 1991) e em requisitos de outros modelos de decisão, que também podem ser usados como critérios para avaliar opções de *design* (HASELBÖCK; WEINREICH; BUCHGEHER, 2017).

EDDIN e MAMMERI (2014) mencionam que é de grande valia utilizar as preferências do usuário de propriedades não-funcionais, como preço, desempenho, a fim de selecionar a configuração mais satisfatória. Em (EDDIN e MAMMERI, 2014), foi proposta uma abordagem para escolher a configuração ideal de um conjunto de alternativas, a fim de maximizar a satisfação do usuário final. Segundo os autores, as preferências do usuário sobre propriedades não funcionais podem ser usadas para guiar o processo de seleção, a fim de obter a configuração ideal buscando maximizar a satisfação do usuário.

SAYARA, TOWHID e HOSSAIN (2017) descrevem uma técnica probabilística para definir os microsserviços a serem gerados a partir de um sistema monolítico, considerando três aspectos principais - taxa de atualização, escalabilidade e a diversidade de tecnologia necessária para realizar cada serviço adequadamente. Os autores utilizaram uma matriz ponderada e a escala multidimensional (*Multidimensional scaling* ou MDS).

Essas técnicas ajudam a definir um número ideal de serviços para implementar o sistema na arquitetura de microsserviços. Para o uso do MDS, é necessário um grupo de serviços de acordo com similaridades, sendo estas calculadas com base em uma matriz ponderada. Essa matriz ponderada é gerada a partir do valor probabilístico da atualização ou dimensionamento dos recursos de negócios identificados nas primeiras etapas. De acordo com os autores, a abordagem de MDS foi escolhida em vez de *k*-

means[5] ou UPGMA[6], pois *k-means* não suporta matrizes de distância[7]. Quanto ao UPGMA, é usado para *clustering* de matriz não ponderada, entretanto, os autores fazem uso de uma matriz ponderada. Desta forma, concluiu-se que MDS é a melhor opção para o modelo proposto por SAYARA, TOWHID e HOSSAIN (2017).

HUANG et al. (2015) descrevem uma metodologia para combinar informações supervisionadas baseado na análise de Componentes principais (*Principal Component Analysis* ou PCA), selecionando seletivamente os componentes. A PCA é um algoritmo estatístico multivariado clássico para análise de dados, tendo por objetivo a extração dos principais recursos ou propriedades dos dados e a representação destes como um conjunto de novas variáveis, chamadas componentes principais. O método proposto é geral para todos os algoritmos da família PCA e ainda pode ser aplicado a outros algoritmos estatísticos multivariados não supervisionados (HUANG et al., 2015).

SERRAI et al. (2017) propuseram uma metodologia para usar uma combinação de dois métodos MCDM (*Multi Criteria Decision Making*), que são AHP e RIM (*Reference Ideal Method*). O AHP é usado para codificar as restrições de pesos de normalização atribuídas aos critérios de QoS (*Quality of Service*) no primeiro estágio. Em seguida, para lidar com as restrições de valor do usuário, considerou-se a RIM para classificar os serviços e às restrições de valor necessárias.

KAUR e TOMAR (2016) mencionam que a seleção de componentes de software continua sendo uma área desafiadora para obtenção dos componentes ótimos a partir de um repositório de componentes existente. As técnicas de agrupamento oferecem uma solução para problemas de seleção de componentes, mas com grandes deficiências como uma especificação prévia do número de *clusters*, especificando o raio dos *clusters*, localizando a representação apropriada para os vetores de recursos, escolhendo a métrica de distância correta, sobreposição de *clusters* e uso de julgamento subjetivo de desenvolvedores de aplicações.

O trabalho de KAUR e TOMAR (2016) apresenta uma técnica de seleção de componentes de software baseada em *clusters fuzzy*[8], utilizando o conceito de relações *fuzzy*. A principal contribuição do algoritmo é que os componentes que são similares

[5] *K-means*: Método de agrupamento no qual particiona-se n observações dentre k grupos, onde cada observação pertence ao grupo mais próximo da média.
[6] UPGMA: *Unweighted Pair Group Method using Arithmetic averages*. Método de agrupamento no qual ligam-se os grupos pela média de similaridade entre seus elementos
[7] Matriz de Distância: matriz contendo as distâncias, tomadas em pares, de um conjunto de pontos.
[8] *Cluster Fuzzy*: *cluster* difuso, sendo este uma forma de *cluster* na qual cada ponto de dados pode pertencer a mais de um *cluster*.

baseados em atributos múltiplos são mantidos juntos, ajudando o tomador de decisão a escolher o conjunto de componentes correto.

2.5.1. Modelos de Seleção e Atributos de Qualidade

FARIDI et al. (2015) propõem um mapeamento dos atributos de seleção de software com o Modelo de Qualidade ISO 25010:2011 (NORMALIZACIÓN, 2011) para análise de tomada de decisão. Tais atributos são frequentemente usados para definir a compra de software de prateleira (COTS), sendo a comparação dos atributos um modelo de grande relevância ao verificar diferentes opções de COTS, pois a comparação destes pode influenciar o custo de aquisição de software, seleção de fornecedores e análise de tomada de decisão.

De acordo com HASELBOCK e WEINREICH (2017), os modelos de orientação de decisão são um meio para a exploração e documentação de projeto. Estes autores apresentam os modelos de orientação de decisão para monitoramento de microsserviços com foco na tomada de decisão baseado no processamento de dados de monitoramento dos microsserviços.

A norma 25010:2011 (NORMALIZACIÓN, 2011) se destaca para este trabalho, pois o item referente à qualidade de software em uso foi observado por melhor representar arquiteturas estabelecidas e, baseado nesta norma, foram obtidas as características e subcaracterísticas a serem trabalhadas durante o processo de verificação e análise da abordagem sugerida no presente trabalho.

2.5.2. Modelos de Avaliação da Criticidade

Acerca do trabalho de SHAO, ZHANG e CAO (2018) foi utilizada uma abordagem de criticidade nomeada ACISM para análise da qualidade dos serviços. A abordagem ACISM usa as informações de peso adquiridas dinamicamente para obter o resultado e usa o resultado para selecionar as instâncias dos microsserviços.

Na abordagem ACISM, utiliza-se cinco parâmetros de avaliação, sendo estes:

- Utilização da CPU: referente ao servidor onde a instância do microsserviço está localizada.
- Número de conexões de solicitação no microsserviço: representando o número de usuários na instância de microsserviço que estão sendo acessados.

- Tempo esperado de resposta das instâncias de microsserviço: o tempo transcorrido para o usuário enviar uma solicitação ao serviço e receber a resposta da instância de microsserviço.
- Disponibilidade prevista das instâncias de microsserviço: a probabilidade de uma instância de microsserviço ser chamada com sucesso em um dado período de tempo.
- Confiabilidade esperada das instâncias de microsserviço: a probabilidade de uma instância do microsserviço responder corretamente as solicitações do usuário em um determinado período de tempo.

Esta dissertação de mestrado propõe a abordagem M2CSEA que se difere da abordagem ACISM, pois utiliza múltiplos critérios, diferente dos propostos pela abordagem ACISM, correlacionando-os com a opinião dos usuários e associada à norma ISO 25010:2011, podendo esta abordagem ser estendida em trabalhos futuros para contemplar novas verificações. Acerca das diferenças da abordagem M2CSEA em relação a abordagem ACISM (SHAO; ZHANG; CAO, 2018) estas poderão ser observadas na Seção 4.3.

2.6. Conclusão

Conforme observado nas seções anteriores, a adequada escolha da arquitetura relacionada aos requisitos propostos ao projeto é de fundamental importância. Entretanto, os desafios acerca da qualidade e da capacidade desta em atender aos requisitos que promoveram sua escolha não são cessados após a sua implementação.

Desta maneira, é de grande relevância que os microsserviços contidos em uma arquitetura estabelecida, mesmo que esta esteja aderente aos requisitos propostos, sejam acompanhados de maneira constante e dinâmica, observando os diferentes papéis desempenhados pelos envolvidos na tomada de decisão acerca da evolução ou manutenção desta arquitetura estabelecida.

Vislumbrando a maior estabilidade em arquiteturas estabelecidas, o presente trabalho, com o apoio da utilização do método AHP, propõe o uso de múltiplos critérios, a fim de avaliar os microsserviços mais críticos. Considerando tais critérios com diferentes pesos de acordo com o perfil de usuário que fará uso desta abordagem, tornando-a mais aderente a necessidade de diferentes perfis ou mesmo agregando diferentes visões, com o intuito de sugerir de forma mais guiada e eficiente o

microsserviços para crítico, a fim de delinear o foco da evolução ou manutenção em vista.

A abordagem proposta por esta dissertação, diferente das demais expostas neste capítulo, possui por objetivo proporcionar ao usuário final a capacidade de acompanhar em tempo real os microsserviços mais críticos em uma arquitetura estabelecida orientada a microsserviços. Baseado neste objetivo, a abordagem faz uso de múltiplos critérios de avaliação associados a opinião dos usuários, diferindo-se da abordagem de avaliação de microsserviços proposta por SHAO, ZHANG e CAO (2018), na qual são utilizados métodos de verificação não associados às opiniões dos usuários.

CAPÍTULO 3 – VALIDAÇÃO COM ESPECIALISTAS

3.1. Introdução

No capítulo anterior foi apresentada a fundamentação teórica para este trabalho tendo sido baseada em uma revisão da literatura, na qual observou-se a falta de trabalhos relacionados a criticidade em arquiteturas orientadas a microsserviços. Neste capítulo será apresentado uma pesquisa conduzida com especialistas a fim de validar a motivação para o tema.

Motivado pela necessidade de possuir um *feedback* da comunidade de especialista, foi executado durante o mês de agosto de 2019 uma pesquisa com 20 praticantes experientes de engenharia de software da indústria e da academia. A partir desta pesquisa foi constatada a falta de uma abordagem definida para medir a criticidade em arquiteturas estabelecida orientadas a microsserviços.

Além desta seção introdutória, este capítulo está organizado com a apresentação do objetivo da pesquisa na Seção 3.2. Na Seção 3.3, são apresentadas as hipóteses desta pesquisa. Na Seção 3.4, estão detalhadas a informações acerca da seleção e arranjo dos participantes. Na Seção 3.5, são apresentados os detalhes da execução da pesquisa. Na Seção 3.6, são descritos os resultados obtidos após a execução desta pesquisa. Na Seção 3.7, detalha-se a interpretação dos resultados obtidos. Na Seção 3.8, são descritas as ameaças à validade das hipóteses e do estudo. Por fim, a conclusão é apresentada na Seção 3.9.

3.2. Objetivo

Essa pesquisa teve como objetivo observar a necessidade de uma abordagem para medir a criticidade em arquiteturas orientadas a microsserviços. As informações sobre os microsserviços mais críticos são relevantes para arquitetos de software e outros tomadores de decisão, orientando assim a manutenção e a evolução da arquitetura de uma maneira mais assertiva e guiada.

RQ1 - Existe uma maneira de classificar os microsserviços por criticidade?

Nesta questão de pesquisa, a principal descoberta está associada à forma como os microsserviços estão sendo classificados atualmente em relação à criticidade. Esta pergunta pretende verificar quão comum é a classificação dos microsserviços em

relação à criticidade. A frequência do uso de determinados métodos pode indicar a utilidade dessa abordagem para as empresas que a utilizam.

RQ2 - Existe um consenso sobre uma abordagem de avaliação de criticidade para classificar microsserviços?

Considerando que existem vários atributos que podem ser associados à criticidade de microsserviços, contemplando diferentes pontos de vista das mais diversas partes interessadas e o uso desejado dessas informações em arquiteturas estabelecidas, esta questão de pesquisa pretende responder até que ponto existe um consenso acerca das abordagens atuais de classificação.

RQ3 - A abordagem de avaliação da criticidade baseada em múltiplos critérios pode auxiliar a maneira como a criticidade é verificada atualmente?

Considerando que um consenso não foi estabelecido até essa data, seria possível que uma abordagem de criticidade com múltiplos critérios pudesse melhorar a maneira como a criticidade é verificada e servir como uma diretriz para manutenção e melhorias.

3.3. Hipóteses

Nesta seção, são apresentadas as hipóteses e construto propostos nesta pesquisa.

***Hipótese Nula*:** A sugestão de criticidade de microsserviços com base em múltiplos critérios não agrega valor em arquiteturas estabelecidas.

Hipótese Alternativa: A sugestão de criticidade de microsserviços com base em múltiplos critérios pode agregar valor em arquiteturas estabelecidas.

***Construto*:** Uso de modelos estatísticos quantitativos para avaliar a capacidade da abordagem de avaliação de múltiplos critérios em arquiteturas de microsserviços para promover melhorias em arquiteturas estabelecidas associadas ao uso de dados orientados ao contexto para caracterização demográfica, com o objetivo de caracterizar os participantes mais relevantes para o aumento da validade interna do estudo.

3.4. Seleção e Arranjo dos Participantes

Os participantes foram selecionados com base na abordagem não probabilística e essa seleção ocorreu por conveniência. A seleção foi feita de acordo com o cargo e a função exercida nas empresas em que trabalham os participantes, associada ao tempo de experiência no desempenho dessa função, independentemente da empresa trabalhada. A experiência mínima necessária para a participação no estudo foi de 4 anos de experiência com desenvolvimento de software e pelo menos um ano de experiência no desenvolvimento de microsserviços.

A população foi estabelecida com base em experiências anteriores com desenvolvimento de software e microsserviços. Essa premissa permite procurar respostas mais fundamentadas, considerando a experiência de cada participante. Em relação às funções exercidas, estas foram focados em arquitetos de software, considerando arquitetos de solução e arquitetos de integração, engenheiros de software, considerando analistas de sistema, e por fim, gerentes com conhecimentos técnicos. No entanto, foram aceitos participantes não relacionados a esses papéis, mas que possuíam experiência suficiente que correspondesse ao mínimo mencionado para a seleção dos participantes.

3.4.1. Contato

Foi enviado por e-mail um questionário no qual todas as perguntas foram preparadas para serem respostas sem auxílio, considerando a experiência anterior de cada participante. A pesquisa foi aplicada virtualmente, utilizando a plataforma SoGoSuvey[9], contendo 32 perguntas e informações explicativas dispostas ao longo do questionário, exigindo aproximadamente 15 a 20 minutos para conclusão.

3.5. Execução do Estudo

Um resumo da pesquisa, bem como todas as informações expostas aos participantes durante o experimento, está disposto nos Apêndices D (Formulário de Consentimento do Survey) e E (Questionário do Survey). Durante o experimento, os participantes não receberam nenhuma informação adicional que não estivesse contida nestes documentos.

[9] SoGoSurvey: https://www.sogosurvey.com

A pesquisa foi criada usando perguntas de múltipla escolha, principalmente com base na escala Likert de cinco níveis, em virtude da facilidade de responder questões deste tipo e analisar as respostas estatisticamente. As respostas discrepantes (*outliers*) foram avaliadas com base na abordagem 3-sigma e analisadas caso a caso para remoção caso necessário, entretanto após análise nenhuma resposta foi removida manual ou automaticamente.

3.6. Resultados

3.6.1. Informações Demográficas

A maioria dos entrevistados se identificou como engenheiros de software ou arquitetos de software, ambos com 9 participantes cada, seguidos por gerente com habilidades técnicas e CEO de uma startup, ambos com 1 participante cada.

A disposição geográfica do participante, está concentrada principalmente no Brasil, com 16 participantes, seguida pelos EUA, com 2 participantes, e por fim Portugal e Espanha, ambos com 1 participante cada. Essa distribuição geográfica é relevante para esta pesquisa, pois pode servir como validação para a variação de contexto devido a multiculturalidade dentre estas regiões.

Em relação à cidade de residência, concentrou-se principalmente na cidade de Blumenau no estado de Santa Catarina no Brasil, com 9 participantes, seguido pela cidade do Rio de Janeiro no estado do Rio de Janeiro no Brasil, com 4 participantes, seguido da cidade de São Paulo no estado de São Paulo no Brasil e da cidade de San Ramon no estado da Califórnia nos EUA, ambos com 2 participantes cada, finalmente, a cidade de Barcelona da região da Catalunha na Espanha, a cidade de Braga na província de Minho em Portugal e a cidade de Pomerode no estado de Santa Catarina no Brasil, todos com 1 participante cada.

3.6.1.1. Experiência dos Participantes

O perfil dos respondentes atendeu às expectativas, considerando as experiências anteriores com desenvolvimento de software e microsserviços. A maioria dos participantes (13) possui mais de dez anos de experiência em desenvolvimento de software, baseado nos quais pode ser designada a estes participantes a atribuição de especialistas em sua área de atuação.

A experiência com componentes, referenciada nesta pesquisa como sendo o desenvolvimento de software com componentes reutilizáveis (por exemplo, DLLs[10], JARs[11], etc.) segue resultados semelhantes, com 9 participantes com mais de dez anos de experiência, caracterizando-se como uma situação positiva para proceder com comparações com o desenvolvimento de serviços, considerando SOA e microsserviços agrupados nesta categoria.

Em relação à experiência com SOA, os resultados indicam que a maioria dos participantes (10) tem mais de quatro anos de experiência com essa abordagem de arquitetura de software, sendo a quantidade de participantes mais experientes, com oito a dez anos ou mais de dez anos, igual a 5 participantes.

Considerando a experiência em microsserviços, a maioria dos participantes (14) tem pelo menos um ano e no máximo três anos, os demais (6 participantes) possuem mais de quatro anos de experiência. Considerando a maturidade recente dessa abordagem de arquitetura de software, esses números indicam um alto nível de experiência para participar desta pesquisa e ser considerado como resposta relevante.

3.6.2. Modelo de Qualidade de Software Baseado na ISO 25010

Nesta fase, os participantes foram solicitados a classificar a relevância dos atributos de Qualidade da ISO 25010:2011 relacionando-os à arquitetura de microsserviços, considerando suas experiências profissionais anteriores e considerando a descrição dos atributos de qualidade contida na especificação.

As propriedades mensuráveis relacionadas à qualidade de um sistema são chamadas de propriedades de qualidade, com medidas de qualidade associadas. Para chegar a medidas da característica ou subcaracterística da qualidade, a menos que estas possam ser medidas diretamente, é necessário identificar um conjunto de propriedades que juntas as abrangem, obtendo medidas de qualidade para cada uma e combinando-as computacionalmente para chegar a uma medida de qualidade derivada correspondente à característica ou subcaracterística de qualidade.

Nesta questão de classificação, os participantes foram solicitados a ranquear os atributos de qualidade de um a seis, considerando o primeiro sendo o mais relevante, atribuindo a ele um ponto, e o menos relevante, o sexto, atribuindo a ele seis pontos. Ao

[10] DLL: https://support.microsoft.com/pt-br/help/815065/what-is-a-dll
[11] JAR: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html

final, o atributo com menor quantidade de pontos é o mais relevante, assim como o com maior pontuação será o de menor relevância.

Conforme observado na Figura 3.1, os resultados mostraram dois grupos de semelhante relevância em relação aos atributos de qualidade propostos pela ISO 25010:2011. No primeiro grupo, os atributos mais relevantes são compostos por "Eficiência", com 50 pontos, e "Eficácia", com 51 pontos. No segundo grupo, os atributos menos relevantes são compostos por "Menor Risco" e "Cobertura de Contexto", com 65, e "Satisfação", com 69 pontos.

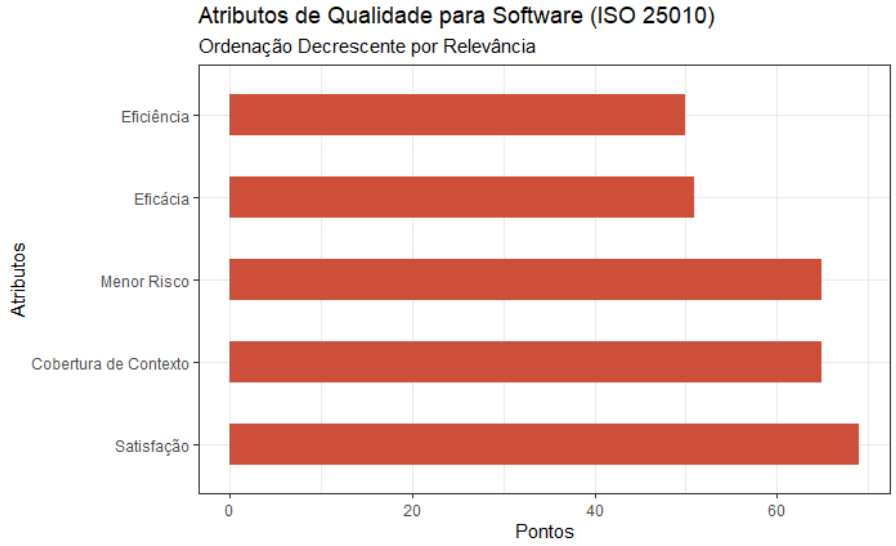


Figura 3.1 – Atributos de qualidade para software

3.6.3. Modelo de Qualidade de Software em Uso Baseado na ISO 25010

O modelo de qualidade de produto pode ser aplicado a um produto de software ou a um sistema de computador. De acordo com a ISO 25010:2011 (NORMALIZACIÓN, 2011), o sistema de computador é incluído em um sistema de informação que também pode incluir um ou mais sistemas de computador e sistemas de comunicação, como uma rede de área local e a Internet. O sistema de informações está dentro de um sistema humano-computador mais amplo, como um sistema corporativo ou sistema de larga escala, e pode incluir usuários e o âmbito técnico e físico.

Nessa fase, da mesma forma que a anterior, os participantes foram solicitados a classificar a relevância dos atributos de qualidade de produto de software em uso segundo a ISO 25010:2011, relacionando-os à arquitetura de microsserviços, considerando suas experiências profissionais anteriores e a descrição dos atributos de qualidade contida na especificação.

Nesta questão de classificação, mais uma vez, os participantes foram solicitados a ordenar os atributos de qualidade, considerando o primeiro como o mais relevante, atribuindo a ele um ponto, e o oitavo o menos relevante, atribuindo a ele oito pontos. Ao final, o menos votado é o atributo de maior relevância.

Conforme observado na Figura 3.2, os resultados mostraram dois grupos de relevância semelhante em relação aos atributos de qualidade para software em uso proposto pela ISO 25010:2011. No primeiro grupo, os atributos mais relevantes, são compostos por "Eficiência de Desempenho", com 63 pontos, "Resiliência", com 74 pontos, seguido de "Adequação Funcional" com 82 pontos, "Portabilidade" com 86 pontos, "Compatibilidade" com 88 pontos e "Manutenibilidade" com 93 pontos. No segundo grupo, os atributos menos relevantes são compostos por "Segurança" com 116 pontos e "Usabilidade" com 118 pontos.

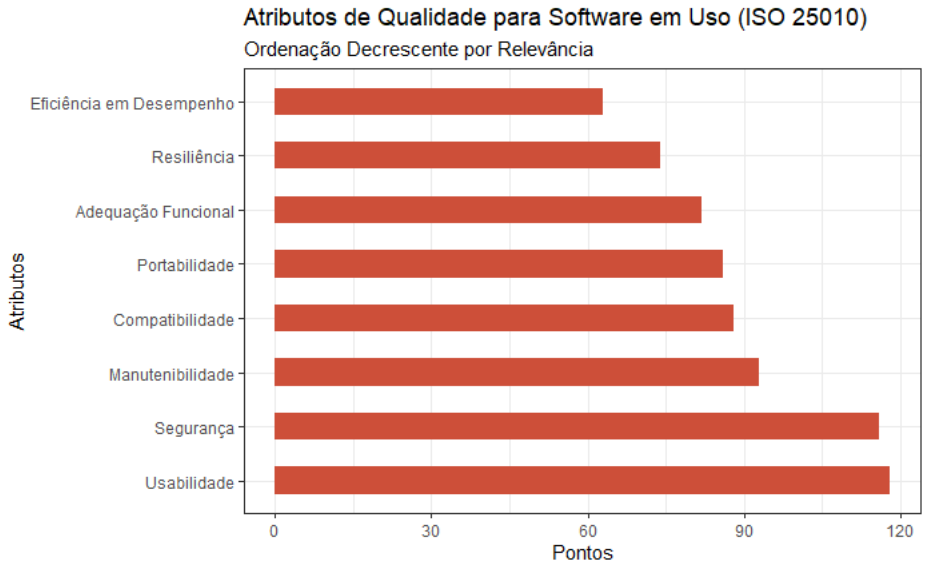


Figura 3.2 – Atributos de qualidade de produto em uso ordenados por relevância

3.6.3.1. Adoção de Microsserviços

Nesta fase, os participantes foram convidados a responder com base em suas experiências profissionais anteriores sobre os fatores mais relevantes para a escolha da arquitetura orientada a microsserviços e os desafios observados nessa adoção.

O fator mais relevante para a adoção de microsserviços foi a "Escalabilidade", com 90% de concordância, considerando a resposta "Concordo Fortemente" com 14 participantes e a resposta "Concordo" com 4 participantes. O segundo fator de maior relevância foi a capacidade de suportar várias linguagens de programação, representada na Figura 3.3 como "Ling. de Programação", com 75% de concordância, considerando a

resposta “Concordo Fortemente” com 6 participantes e a resposta “Concordo” com 9 participantes.

Os seguintes fatores tiveram concordância semelhante, com redução do tempo para o mercado, representada como “Tempo ao Mercado”, com 60% de concordância, seguida de capacidade para suporte a integração contínua, representada como “Integração Contínua”, com 55% e, redução da complexidade na manutenção da arquitetura, representado como "Manutenibilidade", com 50% de concordância. Considerando esses resultados, é possível afirmar que a adoção dos microsserviços tenha sido influenciada por estes fatores.

O último fator, com 35% de concordância e 25% de desacordo, foi o aumento da produtividade, representado na Figura 3.3 como “Produtividade”. Considerando esses resultados, não é possível concluir que a adoção dos microsserviços tenha sido influenciada por esse fator.

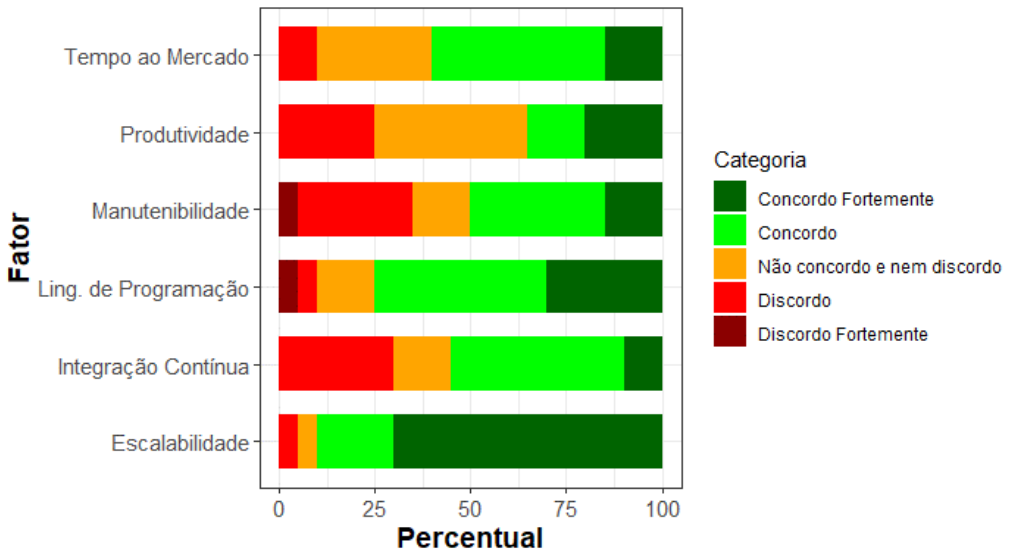


Figura 3.3 – Fatores para a adoção de microsserviços

A próxima pergunta estava relacionada aos desafios da adoção de microsserviços. Nesta questão os participantes classificaram os desafios observados considerando o primeiro como o maior desafio, atribuindo a ele um ponto, e o menor desafio sendo o sexto, atribuindo a ele seis pontos. Ao final, o menos votado é o maior desafio por tratar-se de um ranqueamento.

Como é possível observar na Figura 3.4, o desafio mais frequente observado pelos participantes é adaptar a cultura da empresa à arquitetura orientada a microsserviços, representado na Figura 3.4 como “Adaptar Cultura” e possuindo um total de 61 pontos. O próximo desafio de maior relevância foi a “Divisão de

Microsserviços", com 68 pontos, seguido de "Diagnóstico de Problemas" com 69 pontos, "Gestão da Escalabilidade" com 71 pontos, "Monitoramento de Custos" com 72 pontos e, finalmente, adaptar os microsserviços ao modelo de negócio, representado na Figura 3.4 como "Adaptação ao Modelo de Negócio", com 79 pontos.

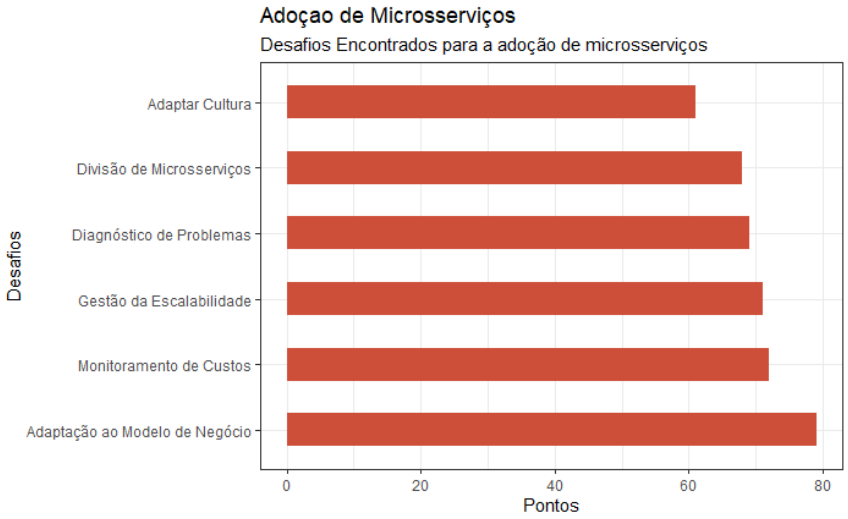


Figura 3.4 – Desafios na adoção de microsserviços

3.6.4. Métodos de Classificação de Criticidade

Nesta fase, os participantes foram solicitados a responder sobre os métodos de classificação de criticidade usados durante suas experiências profissionais anteriores com base em uma escala de Likert de cinco níveis. Esta fase pretendeu obter as experiências dos participantes com a criticidade em microsserviços e como esses microsserviços foram tratados durante sua última experiência com essa abordagem de arquitetura. As opções apresentadas aos participantes foram escolhidas de acordo com a experiência do autor em arquitetura e desenvolvimento de software, com as quais este elencou pontos comumente observados em sistemas críticos.

A maioria dos participantes, 75% destes, responderam que havia uma maneira de classificar o microsserviço em relação à criticidade. Não houve consenso quanto ao método de classificação da criticidade. No entanto, foi possível observar um alto nível de concordância quanto a classificação por "Número de Integrações" (70%), considerando "Concordo Fortemente" com 2 votos e "Concordo" com 12 votos.

Os outros fatores mais utilizados foram "Relevância para o Negócio" com 55% de concordância e "Opinião de Especialistas" com 45%. Por outro lado, o fator menos utilizado foi "Custo de Manutenção" com 50% de desacordo, seguido por "Falta de

Testes Unitários” com 45%. O fator “Tempo de Resposta” com 35% de concordância e 30% de discordância não foi conclusivo.

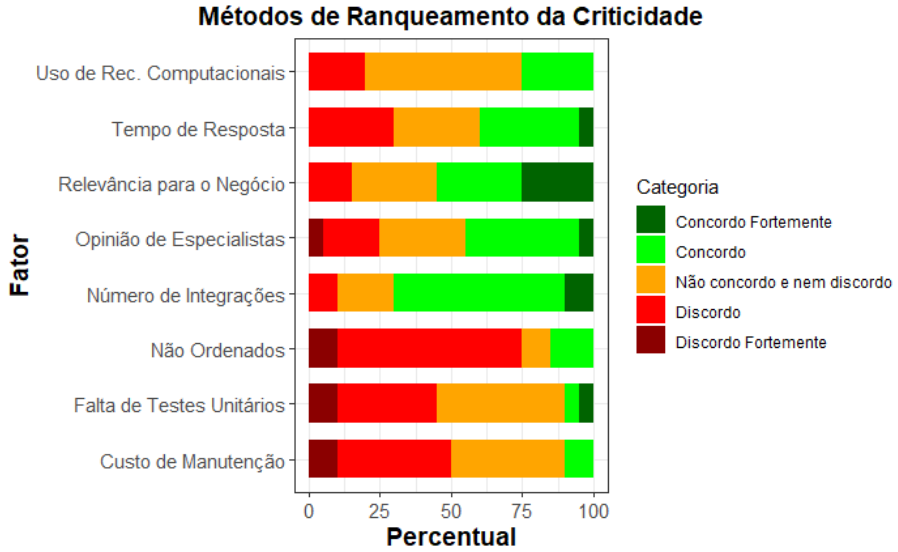


Figura 3.5 – Métodos de classificação de criticidade em microsserviços

3.6.5. Validação da Proposta

Nesta fase, os participantes foram convidados a considerar a existência de uma ferramenta baseada na opinião de várias partes interessadas, sugerindo o microsserviço mais crítico com base nos critérios estabelecidos abaixo.

- Tempo de resposta
- Opinião de especialistas
- Custo de manutenção (calculado por débito técnico)
- Cobertura de teste de unidade
- Número de integrações com outros microsserviços
- Relevância para os negócios

Para esta questão foram apresentados três possíveis benefícios: Redução de Defeitos, Redução de Custo, e Melhoria na Resiliência da Arquitetura. Todos os três benefícios foram considerados aptos a promover melhorias em arquiteturas estabelecidas orientadas a microsserviços, considerando uma ferramenta de auxílio para gerenciar e classificar os microsserviços a fim de sugerir o mais crítico, de acordo com múltiplos critérios.

O benefício com maior relevância foi a melhoria na resiliência da arquitetura de microsserviços, representada como “Melhorar a Resiliência da Arq.” na Figura 3.6, com 90% de concordância, onde 3 participantes escolhem “Concordo Fortemente” e 15 participantes escolheram “Concordo”.

O segundo benefício em termos de relevância foi “Redução de Custos”, com 85% de concordância, onde 2 participantes escolheram "Concordo Fortemente" e 15 participantes escolheram “Concordo”.

O último benefício que pôde ser observado foi a redução de defeitos no ambiente de produção, representada como “Redução de Defeitos” na Figura 3.6, com 75% de concordância, onde 4 participantes escolheram “Concordo Fortemente” e 11 participantes escolheram “Concordo”.

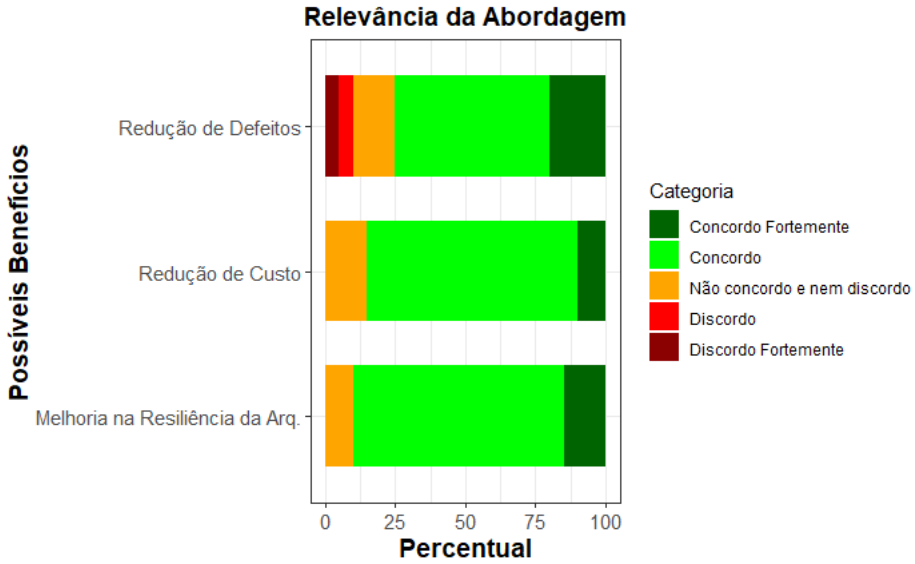


Figura 3.6 – Relevância da abordagem proposta

3.7. Interpretação dos Resultados

3.7.1. Resultados

Com base nos resultados obtidos nas seções anteriores e na agregação dos dados, as perguntas da pesquisa são respondidas a seguir.

RQ1 - Existe uma maneira de classificar os microsserviços por criticidade?

Considerando os resultados obtidos com as questões dos métodos de criticidade, foi possível observar que, atualmente, existem métodos para classificar os microsserviços por criticidade.

RQ2 - Existe um consenso sobre o método de avaliação de criticidade para classificar microsserviços?

Com base nos resultados obtidos, é possível afirmar que não há consenso sobre o método de criticidade para classificar microsserviços.

RQ3 - A abordagem de avaliação da criticidade baseada em múltiplos critérios pode auxiliar a maneira como a criticidade é verificada atualmente?

Considerando os resultados obtidos com as questões de validação da proposta, foi possível observar que atualmente existe uma oportunidade de melhorar a maneira como os microsserviços são classificados em relação a criticidade, considerando múltiplos critérios de acordo com a perspectiva de diferentes partes interessadas (por exemplo, arquitetos, engenheiros de software, gerentes etc.) e seguindo os requisitos de cada um.

3.7.2. Validação das Hipóteses

De acordo com DE WINTER e DODOU (2010), as escalas Likert de cinco níveis são comumente associadas a pesquisas e são usadas em uma ampla variedade de configurações. Ao responder a uma escala Likert, os participantes especificam seu nível de concordância com declarações com tipicamente cinco a sete níveis de resposta ordenada. Os dados em uma escala Likert têm características distintas, como valores discretos em vez de contínuos, números vinculados e intervalo restrito.

A literatura prova que o teste-t e Mann-Whitney-Wilcoxon[12] para a escala Likert de cinco níveis, geralmente, têm aplicação equivalente, exceto para distribuições discrepantes ou multimodais para as quais ocorreram significativas diferenças entre os dois testes (DE WINTER e DODOU, 2010). A taxa de erro do tipo I[13] de ambos os métodos não foi superior a 3% acima da taxa nominal de 5%, mesmo quando os tamanhos das amostras eram desiguais. Considerando essa suposição, a hipótese será validada com um teste-t de uma amostra (*one sample t-test*).

A hipótese alternativa considera que mais da metade dos participantes considera a abordagem útil. Prosseguindo com a pontuação da escala de Likert, atribuindo o valor crescente a cada nível de Likert, são geradas as seguintes pontuações que são atribuídas

[12] Teste Mann-Whitney-Wilcoxon: um teste não paramétrico aplicado para duas amostras independentes.
[13] Erro do Tipo I: Consiste em rejeitar a hipótese nula quando ela é verdadeira, ou seja, comete-se um erro ao afirmar que um resultado possui significância estatística quando na verdade este aconteceu por acaso.

a cada resposta do participante: “Concordo Fortemente” recebe 1 ponto, “Concordo” recebe 2 pontos, “Não concordo nem discordo” recebe 3 pontos, “Discordo” recebe 4 pontos e “Discordo Fortemente” recebe 5 pontos. Considerando a pontuação mencionada acima, o média populacional (μ[14]) no teste-t de uma amostra (*one sample t-test*) será 2,99, em relação a primeira posição relevante antes da pontuação neutra (3).

Tabela 3.1 – Teste de hipótese

Teste de Relevância para a Proposta de Abordagem	p-value
Melhoria na Resiliência da Arquitetura	> 0.01
Redução de Defeitos	> 0.01
Redução de Custo	0.02427

Com base no p-value, a hipóteses nula foi refutada para todos os três benefícios propostos. Desta forma, a hipótese alternativa é verdadeira, o que significa que a sugestão de criticidade de microsserviços baseada em múltiplos critérios pode agregar valor às arquiteturas estabelecidas.

3.8. Ameaças à Validade

Quanto à validade do construto, foi garantido o sigilo e o anonimato das respostas individuais, a fim de mitigar possíveis ameaças à coleta de informações imprecisas devido à apreensão da avaliação. O fornecimento de listas predefinidas com respostas pré-definidas no questionário é uma limitação deste estudo com base na literatura sobre engenharia de software e sistemas.

Em relação à validade interna do estudo, o grau de conclusão é considerável, pois o modelo de caracterização da população foi baseado na previsão de questões demográficas associadas às experiências prévias de participação em projeto de engenharia de software.

Em relação à validade externa do estudo, a seleção dos participantes foi uma amostragem geograficamente não delimitada, que pode representar uma parcela da população, porém possui limitações de generalização da população como um todo em função do número de participantes. Considerando a experiência da população escolhida e a dificuldade em localizar especialistas nesta área de conhecimento, este estudo considera uma representação suficiente de uma população qualificada e experiente em referência à experiência em desenvolvimento de software e microsserviços. Acerca do

[14] μ: letra grega que representa a média populacional ou mu.

conteúdo das perguntas, estas foram formuladas para permitir a repetição em diferentes contextos e, desta forma, favorecer a reprodutibilidade e ampliação da validação proposta neste estudo.

3.9. Conclusão

Arquiteturas orientadas a microsserviços têm diversas vantagens que são comumente observadas e almejadas por quem pretende usar ou faz uso desse modelo de arquitetura, no entanto, existem desafios acerca de uma implementação adequada. As arquiteturas estabelecidas orientadas a microsserviços requerem uma análise mais profunda para verificar consistentemente os microsserviços críticos, considerando múltiplos critérios (SANTOS e WERNER, 2019).

Com base nos resultados obtidos, é relevante prosseguir com a próxima fase na pesquisa de múltiplos critérios de criticidade de microsserviços. Motivado pela falta de um método para verificar e apoiar os tomadores de decisão na obtenção dos microsserviços mais críticos em uma arquitetura estabelecida, desta forma foi desenvolvida uma abordagem para ajudar os tomadores de decisão em seu processo decisório, a qual será apresentada no próximo capítulo.

CAPÍTULO 4 – A ABORDAGEM M2CSEA

4.1. Introdução

A abordagem proposta por esta dissertação é baseada em critérios não funcionais, que são trabalhados com os usuários de acordo com suas necessidades. Para melhor embasar a escolha desses critérios, foram observadas várias normas e padrões relacionados à qualidade de software, como as ISOs da família 25.000 e relatórios associados (VISSER et al., 2016).

Neste capítulo, a abordagem M2CSEA (*Multicriteria Microservice Criticality Suggestion for Established Architectures*) é apresentada e descrita em detalhes, possuindo por objetivo a sugestão dos microsserviços mais críticos em arquiteturas estabelecidas orientadas a microsserviços com base em múltiplos critérios. Além desta seção introdutória, este capítulo está organizado de forma a detalhar os pontos acerca das aplicações da metodologia, assim como, um cenário de exemplo para ilustrar o uso da abordagem na Seção 4.2. Por fim, as considerações finais são apresentadas na Seção 4.3.

4.2. Aplicação da Metodologia

A abordagem proposta por esta dissertação de mestrado é baseada em quatro fases, com foco em coletar os dados referente aos microsserviços (fase I), elencar as verificações de acordo com a necessidade do usuário (fase II), calcular a criticidade (fase III), e por fim, sugerir os microsserviços mais críticos (fase IV), conforme apresentado na Figura 4.1.

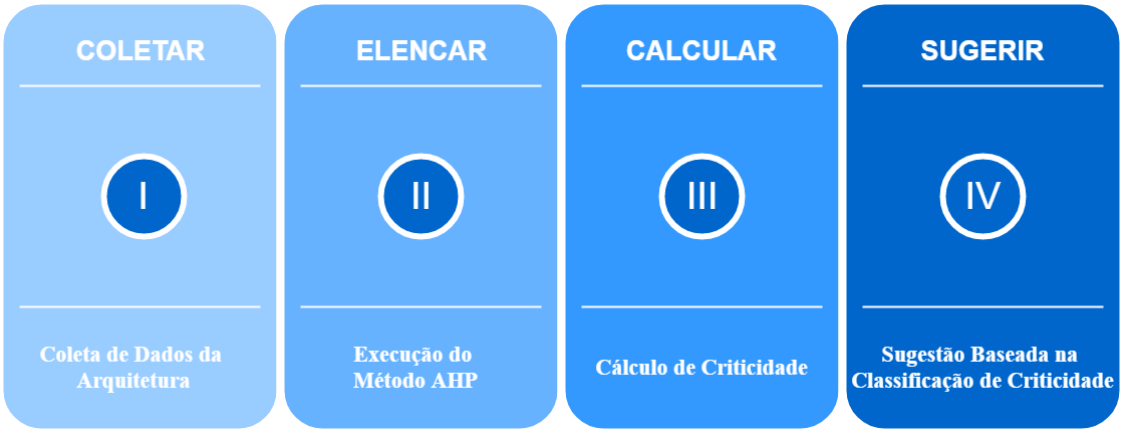


Figura 4.1 – Fases da abordagem M2CSEA

Prevendo a composição de uma sugestão mais orientada ao usuário de microsserviços críticos em arquiteturas estabelecidas, o usuário da abordagem poderá classificar os atributos mais relevantes e, com base no resultado da análise e na ordem das prioridades estipuladas, será mostrado o resultado da análise. A seguir cada uma das fases é brevemente introduzida.

- **Fase I - Coleta de dados da arquitetura:** Nesta fase inicial, as opções de verificação são constantemente analisadas e os dados obtidos são armazenados em bancos de dados para uso posterior.

- **Fase II – Execução do Método AHP:** Nesta segunda fase, o usuário fará uso do método AHP para elencar os critérios de análise mais relevantes segundo sua necessidade.

- **Fase III - Cálculo de criticidade:** Nesta terceira fase, os dados das opções de verificação geram uma pontuação de criticidade, promovendo o cálculo baseado nos dados obtidos nas fases I e II.

- **Fase IV - Sugestão baseada na classificação de criticidade:** Nesta última fase, a classificação de criticidade é apresentada em ordem decrescente e os microsserviços mais críticos são mostrados ao usuário final.

Nas próximas seções são detalhadas as etapas das fases mencionadas anteriormente.

4.2.1. Fase I - Coleta de Dados da Arquitetura

Nesta fase, os microsserviços são constantemente verificados para obtenção de informações relativas a cada uma das verificações da abordagem em questão, conforme ilustrado na Figura 4.2. As métricas expostas nesta seção foram obtidas correlacionando requisitos não-funcionais com ISO 25010 relativa a qualidade de produto de software. O contexto de cada opção de verificação, bem como sua relação com a ISO, será detalhado nas seções a seguir.

Figura 4.2 – Verificações da abordagem M2CSEA

4.2.1.1. Análise da Interdependência entre Microsserviços

O presente trabalho propõe a análise de interfaces de dependência entre componentes, onde neste caso são especificamente tratados como microsserviços. As interfaces de entrada são tidas como "*incoming*", ou seja, outro componente depende deste componente, e as interfaces de saída são tidas como "*outgoing*", onde este componente depende de outro componente. A relevância do uso desta análise é baseada no possível impacto gerado pela alteração das funcionalidades providas ou indisponibilidade do componente.

Uma equação simples é usada para promover essa classificação, como explicado na Fórmula (4.1) a seguir.

$$\begin{cases} a = i * 1.5 \\ b = j * 0.5 \end{cases} W = a + b \tag{4.1}$$

Nesta equação, *i* é uma representação para chamadas recebidas, e *j* é a representação para chamadas de saída. A estes dois valores estão associados diferentes multiplicadores, sendo calculado o fator de criticidade (W) com base no maior impacto da entrada chamada, em comparação com chamadas de saída em termos de dependência de informações. Para esta verificação é proposto que seja utilizado um grafo de chamada.

Um grafo de chamada é um grafo direcionado no qual um vértice representa uma função, um componente ou um método e uma aresta entre dois vértices A e B significa que A pode invocar B. Os arquitetos podem fazer uso de grafos de chamadas para entender os impactos potenciais que uma mudança pode causar em um software (CHAUMUN et al., 2002).

Considerando um serviço Y, representado por um vértice principal, possua N integrações, representadas por vértices ligados por arestas ao vértice principal, e façam uso das funcionalidades deste serviço Y, ao realizar uma alteração em funcionalidades associadas a estes integrações, o impacto pode se estender por qualquer uma das N integrações mencionadas. Desta forma, pode ser de grande utilidade o entendimento das integrações para mensurar o impacto sobre a manutenção em um serviço.

4.2.1.2. Análise do Tempo de Resposta

Para análise do tempo de resposta, são consideradas cada uma das interações e a taxa de falha das requisições entre microsserviços, sendo capaz de ilustrar, desta forma, quais microsserviços possuem integrações mais lentas e, por consequência, poderão resultar em um maior tempo de resposta ao usuário final, ou ainda verificar quais serviços merecem maior atenção por estarem com maiores taxa de falhas, necessitando de intervenção para verificar a causa raiz do problema.

4.2.1.3. Análise da Cobertura de Testes Unitários

Na análise de cobertura de testes unitários, são verificados todos os microsserviços, a fim de listá-los por ordem de menor cobertura, ou seja, nesta análise, o microsserviço mais crítico será o que possuir menor cobertura de testes unitários.

A análise da cobertura do código-fonte é o processo de encontrar áreas de um software não verificado por um conjunto de casos de teste, sendo necessário criar casos de teste adicionais para aumentar a cobertura e determinar uma medida quantitativa de cobertura de código, que é uma medida indireta de qualidade (CORNETT, 2002).

Esta análise torna-se relevante em função da possibilidade de inclusão de não conformidades no código-fonte alterado por não haver meios automatizados de verificado o trecho de código alterado. Assim sendo, o presente trabalho assume que quanto menor a cobertura de testes unitários, maior é o risco da inclusão de não-conformidades.

4.2.1.4. Análise do Débito Técnico

Conforme a literatura, é possível observar o débito técnico como uma abordagem popular para acompanhar o progresso do desenvolvimento do código-fonte, apontando os pontos de melhoria que precisam de refatoração (MARINESCU, 2012) (BROWN et al., 2010) . A sua reparação pode ser realizada imediatamente ou agendada para uma data posterior à custa de incorrer em dívida (BROWN et al., 2010) (KRUCHTEN; NORD; OZKAYA, 2012).

Para a análise estática do código-fonte, nesta dissertação foram escolhidos os modelos SQALE, considerando que esta é uma implementação operacional das especificações ISO para produtos de software com amplo uso na indústria (IZURIETA; GRIFFITH; HUVAERE, 2017).

A análise de débito técnico no presente trabalho é efetuada em cada um dos microsserviços. A abordagem utilizada para o cálculo do débito técnico utiliza o modelo de qualidade SQALE, conforme afirmado anteriormente. Este método é organizado em três níveis hierárquicos. O primeiro nível é composto de características, o segundo é composto das subcaracterísticas, e o terceiro nível é composto por requisitos que se relacionam com os atributos internos do código-fonte. Esses requisitos geralmente dependem do contexto e da linguagem de programação do software. Qualquer violação desses requisitos induz ao débito técnico (LETOUZEY e ILKIEWICZ, 2012).

4.2.1.5. Análise da Utilização de Recursos Computacionais

A análise do uso de recursos computacionais, mediante a obtenção dos dados de utilização de memória RAM e CPU do servidor no qual o microsserviço está em execução, é relevante para os tomadores de decisão para entender o custo associado à execução do microsserviço em diferentes situações, sob estresse ou sem carga (*footprint*). Um alto consumo de recursos pode destacar a necessidade de investigar possíveis melhorias. Para obtenção destes dados, é realizada uma solicitação ao sistema operacional do servidor no qual o microsserviço em questão está sendo executado.

4.2.1.6. Análise da Disponibilidade

A análise da disponibilidade, mediante a obtenção frequente dos dados do número de ocorrências de indisponibilidade de cada um dos microsserviços, busca informar a situação atual dos microsserviços e dos microsserviços dos quais estes

dependem, fornecendo assim insumos acerca das indisponibilidades que a arquitetura em questão pode enfrentar.

4.2.1.7. Análise e Qualidade de Software em Uso

De acordo com a ISO 25010:2011 (NORMALIZACIÓN, 2011), a especificação e avaliação abrangentes da qualidade do software é um fator essencial para garantir valor às partes interessadas. Isso pode ser alcançado através da definição das características de qualidade necessárias e desejadas, associadas às metas e objetivos das partes interessadas pelo sistema. Isso inclui características de qualidade relacionadas ao sistema de software e dados, bem como o impacto que o sistema tem sobre as partes interessadas. Faz-se necessário que as características da qualidade sejam especificadas, medidas e avaliadas sempre que possível, usando medidas e métodos de medição validados ou amplamente aceitos. Os modelos de qualidade da ISO 25010:2011 podem ser usados para identificar características relevantes da qualidade que podem ser utilizadas para estabelecer requisitos, seus critérios de satisfação e as medidas correspondentes.

O modelo de qualidade de produto em uso é composto por oito características, que são subdivididas em subcaracterísticas. O modelo é aplicável a sistemas de computador e produtos de software. As características e subcaracterísticas fornecem uma terminologia para especificar, medir e avaliar a qualidade do sistema e do produto de software, bem como um conjunto de características com as quais os requisitos de qualidade podem ser comparados quanto à cobertura e completude da implementação do modelo.

De acordo com a ISO 25010:2011, o processo para especificar e medir todas as subcaracterísticas para todas as partes de um sistema de computador ou produto de software não é simples e pode ser custoso. Da mesma forma, usualmente, não é prático especificar ou medir a qualidade em uso para todos os cenários possíveis dos casos de uso do usuário. A importância relativa das características de qualidade depende dos objetivos de alto nível e objetivos para o projeto. Portanto, o modelo pela ISO 25010:2011 deve ser adaptado antes do uso como parte da decomposição de requisitos para identificar essas características e subcaracterísticas que são mais importantes, e os recursos alocados entre os diferentes tipos de medidas, dependendo dos objetivos das partes interessadas e objetivos para o produto.

As Tabelas 4.1 e 4.2 expõem o relacionamento entre os atributos de qualidade, segundo a ISO 25010:2011, e as opções de verificação propostas, além de destacar as limitações da proposta. Cada uma das características e subcaracterísticas da norma contempladas pela abordagem M2CSEA estão detalhadas no Apêndice A.

Tabela 4.1 – Características de qualidade de produto segundo a ISO 25010:2011

Características de Qualidade – ISO 25010:2011	
Adequação funcional (*Functional Suitability*)	FS
Eficiência de desempenho (*Performance efficiency*)	PE
Compatibilidade (*Compatibility*)	CO
Usabilidade (*Usability*)	US
Confiabilidade (*Reliability*)	RE
Segurança (*Security*)	SE
Manutenibilidade (*Maintainability*)	MA
Portabilidade (*Portability*)	PO

Tabela 4.2 – Associações entre as verificações e qualidade de produto

Subcaracterísticas de Qualidade de Produto em Uso ISO-25010:2011	**Interdependência**	**Tempo de Resposta**	**Cobertura de Testes Unitários**	**Débito Técnico**	**Uso de Recursos Comp.**	**Disponibilidade**	**Fora do Escopo (Limitações)**
PE - Comportamento temporal *(Time behavior)*		■					
PE - Utilização de recursos *(Resource utilization)*					■		
PE - Capacidade *(Capacity)*							L
CO - Coexistência *(Co-existence)*							L
CO - Interoperabilidade *(Interoperability)*							L
RE - Maturidade *(Maturity)*							L
RE - Disponibilidade *(Availability)*						■	L
RE - Tolerância ao erro *(Fault tolerance)*							L
RE - Recuperabilidade *(Recoverability)*							L
SE - Confidencialidade *(Confidentiality)*							L
SE - Integridade *(Integrity)*							L
SE - Não repúdio *(Non-repudiation)*							L
SE - Rastreabilidade *(Accountability)*							L
SE - Autenticidade *(Authenticity)*							L
MA - Modularidade *(Modularity)*	■						L
MA - Reusabilidade *(Reusability)*							L
MA - Analisabilidade *(Analysability)*							L
MA - Modificabilidade *(Modifiability)*			■	■			
MA - Testabilidade *(Testability)*							F

As subcaracterísticas de adequação funcional, usabilidade e portabilidade não foram consideradas na Tabela 4.2 por não estarem associadas a requisitos não funcionais, ou não estarem relacionadas a arquiteturas estabelecidas. As subcaracterísticas marcadas com uma célula na cor preta foram contempladas na abordagem M2CSEA, as células contendo “L” são relativas às limitações da abordagem. Por fim, as células contendo “F” são relativas a requisitos funcionais e, por isso, estão fora do escopo.

4.2.2. Fase II – Execução do Método AHP

A composição das análises se dá de acordo com a obtenção dos resultados de cada uma das verificações efetuadas anteriormente, gerando insumos para a análise multicritérios com o método AHP, criando após esta análise a listagem de microsserviços mais críticos.

A partir desta listagem, será exposto ao utilizador da abordagem os resultados obtidos e os meios que levaram a tal resultado. Este resultado poderá ser alterado de acordo com os pesos atribuídos a cada um dos critérios utilizados, sendo possível, portanto, que o usuário faça uso da abordagem de acordo com a sua necessidade. A aplicação do método de análise multicritério com AHP se dá em três etapas, conforme descrito a seguir.

4.2.2.1. Etapa 1: Estabelecer a Árvore Hierárquica

Nesta primeira etapa da fase de execução do método AHP, o objetivo e os critérios para decisão são hierarquicamente configurados para a criação da árvore para a tomada de decisão, conforme apresentado na Figura 4.3. Na abordagem M2CSEA, o objetivo é localizar os microsserviços mais críticos, por meio dos critérios de verificação expostos na Seção 4.2.1.

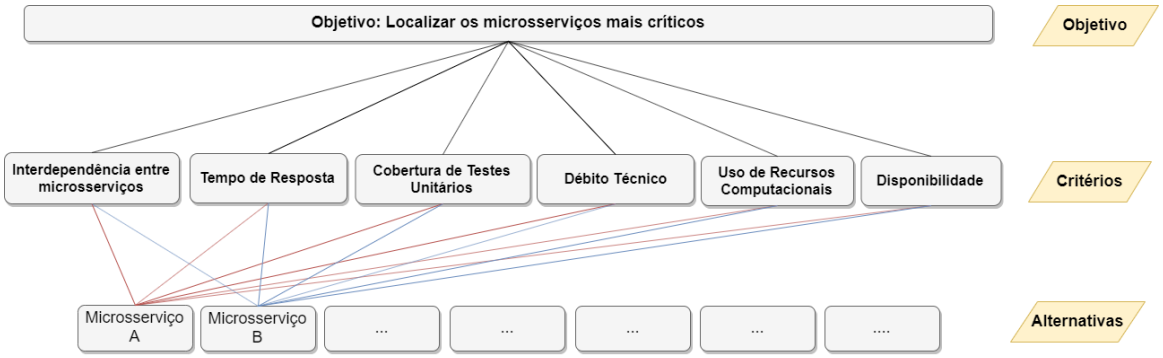


Figura 4.3 – Árvore hierárquica da abordagem M2CSEA

4.2.2.2. Etapa 2: Comparação das Características

Nesta segunda etapa da fase de execução do método AHP, é realizada a comparação de pares, sendo coletadas as opiniões dos usuários. O usuário escolhe o valor desejado para cada característica em comparação com as demais em uma escala que varia de um a nove, onde um significa que os dois elementos são igualmente importantes, e nove implica que um elemento é extremamente mais importante que o outro em uma matriz pareada. A escala em pares e o valor de importância atribuído a cada número estão ilustrados na Tabela 4.3.

Tabela 4.3 – Escala de importância segundo AHP (SAATY, 1980)

Escala de Importância	Definição da Escala de Importância
1	Igualmente importante
2	Igualmente a moderadamente mais importante
3	Moderadamente mais importante
4	Moderadamente a fortemente mais importante
5	Fortemente mais importante
6	Fortemente a significativamente mais Importante
7	Significativamente mais importante
8	Significativamente a extremamente mais importante
9	Extremamente mais importante

Segundo (TAHERDOOST, 2017), se a matriz for incompatível ou inconsistente, a matriz de comparações de pares não pode ser usada na coluna de normalização para obter o vetor de peso normalizado (Wi). Para uma matriz positiva e revertida, pode ser utilizada a técnica de autovetor (*eigenvector*) (JALALIYOON; ABU BAKAR; TAHERDOOST, 2012), utilizando-se as Fórmulas (4.2):

$$e^T = (1,1,\ldots,1)$$

$$W = \lim_{k\to\infty} \frac{A^k . e}{e^T . A^k . e} \quad (4.2)$$

Segundo JALALIYOON, ABU BAKAR e TAHERDOOST (2012), para a convergência entre o conjunto de respostas, o cálculo deve ser repetido, a fim de evitar a tomada de decisão diante de uma matriz incompatível. Em seguida, a Fórmula (4.3) é aplicada para transformar os dados brutos em valores absolutos significativos e com peso normalizado w = (w1, w2, w3 ... wn).

$$Aw = \lambda_{max}\, w, \quad \lambda_{max} \geq n$$

$$\lambda_{max} = \frac{\sum ajwj - n}{w1} \quad (4.3)$$

A={*aij*} com *aij* = 1 / *aij*

A: comparação de pares

w: vetor de peso normalizado

λmax: máximo autovetor (*eigenvector*) da matriz A

aij: comparação numérica entre os valores *i* e *j*

4.2.2.3. Etapa 3: Cálculo de Consistência

Nesta terceira e última etapa da fase de execução do método AHP, são realizados cálculos para validação dos resultados do AHP. Para isso, a razão de consistência (*consistency ratio* ou CR) é calculada usando a fórmula CR = CI / RI, na qual o índice de consistência (*consistency index* ou CI) é, por sua vez, medido pela Fórmula (4.4) (JALALIYOON; ABU BAKAR; TAHERDOOST, 2012).

$$CI = \frac{\lambda_{max} - n}{n-1} \quad (4.4)$$

O valor de RI (*random consistency index*) está relacionado à dimensão da matriz e é extraído da Tabela 4.4. Deve-se notar que a razão de consistência menor que 0,10 (10%) verifica se os resultados da comparação são aceitáveis (JALALIYOON; ABU BAKAR; TAHERDOOST, 2012).

Tabela 4.4 – Valores de índice de consistência aleatória (GOLDEN e WANG, 1989)

Dimensões	RI
1	0
2	0
3	0,5799
4	0,8921
5	1,1159
6	1,2358
7	1,3322
8	1,3952
9	1,4537
10	1,4882

De acordo com FREITAS, MARINS e SOUZA (2006), os procedimentos para o cálculo da CR e CI são:

- Para cada linha da matriz de comparação determinar a soma ponderada, com base na soma do produto de cada valor da mesma pela prioridade da alternativa correspondente;
- Os resultados obtidos deverão ser divididos pelos vetores da respectiva matriz;
- Fazendo uma média dos resultados de cada linha, obteremos λmax ;
- Do ponto de vista do AHP, é desejável que a CR de qualquer matriz de comparação seja menor ou igual a 0,10 (10%).

4.2.3. Fase III - Cálculo de Criticidade

A arquitetura em uso a ser analisada deverá fornecer os insumos para cada um dos critérios, vislumbrando que seja realizada a ponderação dos valores de acordo com as tabelas de conversão 4.5 a 4.10 expostas a seguir, com as quais os valores originais passarão a representar um fator de criticidade associado a cada um dos microsserviços e a cada uma das verificações.

Os microsserviços mais críticos podem ser alterados de acordo com os pesos atribuídos a partir da obtenção da opinião do usuário na Fase II e associado a cada um

dos critérios utilizados, para utilização da abordagem de acordo com a necessidade atual do usuário.

Tabelas de Conversão de Valores

As Tabelas 4.5 a 4.10 representam os intervalos para conversão dos valores, a fim de possuir uma definição base para comparação com as demais verificações. Estes valores são uma referência escolhida pelo autor do presente trabalho. Entretanto, para uso da abordagem M2CSEA, é sugerido que tais valores sejam escolhidos de acordo com a necessidade do utilizador e seus requisitos.

De acordo com (GHIROTTI; REILLY; RENTZ, 2018), com a evolução e crescimento dos sistemas com microsserviços interdependentes, o gerenciamento de dependências é uma parte crucial do *design* de sistemas, pois possuir o conhecimento acerca de novas dependências pode promover uma maior integridade dos dados e reduzir o risco de interrupções.

As dependências podem ser rastreadas observando o comportamento de um sistema, mas prevenir problemas de dependência antes que eles atinjam a produção requer uma estratégia mais ativa. A implementação do controle de dependência garante que cada nova dependência possa ser adicionada a um sistema antes que ele entre em uso (GHIROTTI; REILLY; RENTZ, 2018). A verificação de interdependências na abordagem M2CSEA serve como implementação para controle das dependências, sendo atualizada a cada inicialização de cada um dos microsserviços contidos na arquitetura a ser analisada.

Para a configuração base da abordagem M2CSEA, optou-se por criar um intervalo de dois em dois segundos para o fator de criticidade relativo a interdependência. Este intervalo significa que para esta arquitetura o número de microsserviços possui influência no risco associado a estabilidade do sistema, podendo ser observado a escala de criticidade na Tabela 4.5.

Tabela 4.5 – Escala de conversão de valores para interdependência

Escala de Criticidade	Intervalo Considerado	Definição da Escala de Criticidade
1	W = 0 a 2	Interdependência Muito Baixa
2	W= 2 a 4	Interdependência Baixa
3	W= 4 a 8	Interdependência Mediana
4	W= 8 a 10	Interdependência Alta
5	W= Mais de 10	Interdependência Muito Alta

Acerca do tempo de resposta para carregamento de páginas web, foi conduzido um experimento (GALLETTA et al., 2002) no qual foi examinado o tempo de resposta na escala de 0 a 12, com intervalos de dois em dois segundos. Este estudo sugere que, a redução do desempenho do sistema causa modificações nas intenções comportamentais, pois a satisfação e percepção positiva em relação ao serviço é alterada quando o tempo de resposta é superior 2 segundos, com significativa redução acima de 4 segundos. Para a configuração base da abordagem M2CSEA, optou-se por uma escala com intervalos de 0.5 segundos, pois trata-se de uma simulação de ambiente crítico, no qual a percepção do usuário acerca do desempenho das interfaces disponíveis não poder ser negativa. A Tabela 4.6 descreve os intervalos de tempo considerados e a escala de criticidade relativa a cada um destes.

Tabela 4.6 – Escala de conversão de valores para tempo de resposta

Escala de Criticidade	Intervalo Considerado	Definição da Escala de Criticidade
1	0.1 a 0.5 segundos	Tempo de Resposta Ótimo
2	0.5 a 1.0 segundos	Tempo de Resposta Bom
3	1.0 a 1.5 segundos	Tempo de Resposta Mediano
4	1.5 a 2.0 segundos	Tempo de Resposta Ruim
5	2.0 segundos ou maior	Tempo de Resposta Muito Ruim

Acerca da cobertura de código, de acordo com CORNETT (2002) um percentual aceitável de cobertura para testes unitários está em torno de 70 a 80% para a maior parte dos projetos. Para projetos com alto custo de falha, o valor mínimo aceitável necessita ser superior a este. A norma IEC 61508:2010 (COMMISSION, 2010), propõe que sistemas programáveis relacionados à segurança eletrônica possua 100% de cobertura de código. Para a configuração base da abordagem M2CSEA, optou-se por um intervalo de 20% por tratar-se de um ambiente crítico, mas não relacionado a sistemas similares aos propostos pela IEC 61508:2010, podendo ser observado a escala de criticidade na Tabela 4.7

Tabela 4.7 – Escala de conversão de valores para cobertura de testes unitários

Escala de Criticidade	Intervalo Considerado	Definição da Escala de Criticidade
1	De 80 a 100 %	Cobertura de Testes Muito Alta
2	De 60 a 80 %	Cobertura de Testes Alta
3	De 40 a 60 %	Cobertura de Testes Mediana
4	De 20 a 40 %	Cobertura de Testes Baixa
5	De 0 a 20 %	Cobertura de Testes Muito Baixa

De acordo com TOM, AURUM e VIDGEN (2013), embora o decaimento do código e a deterioração da arquitetura sejam geralmente reconhecidos como as principais dimensões do débito técnico, a obtenção precisa de sua extensão em um sistema representa um desafio significativo.

Estes autores evidenciam que o débito técnico, apesar de descrito por uma metáfora financeira, também está associado a um custo monetário real (TOM; AURUM; VIDGEN, 2013). O débito técnico ou dívida técnica também exibe muitos atributos da dívida financeira, pois pode ser vista como tendo juros e pagamentos, podendo ser usado como meio de alavancar um negócio. Entretanto, quando acumulada, a dívida técnica pode levar o negócio à falência, em função da incapacidade na correção de não-conformidades no software.

Usualmente, as não conformidades aparecem a longo prazo como resultado de maior complexidade do código-fonte do software. O débito técnico inadvertido não afeta a produtividade no curto prazo, pois normalmente não resulta em aumento ou diminuição da velocidade de entrega de código-fonte, pois este pode não ser descoberto por várias iterações ou mais (TOM; AURUM; VIDGEN, 2013).

Para a configuração base da abordagem M2CSEA, optou-se por um intervalo de 120 minutos a cada nível de criticidade, pois considerou-se um cenário de uma arquitetura crítica e uma pequena equipe de desenvolvimento, podendo ser observada a escala de criticidade na Tabela 4.8.

Tabela 4.8 – Escala de conversão de valores para débito técnico

Escala de Criticidade	Intervalo Considerado	Definição da Escala de Criticidade
1	De 0 a 120 minutos	Débito Técnico Muito Baixo
2	De 120 a 240 minutos	Débito Técnico Muito
3	De 240 a 480 minutos	Débito Técnico Mediano
4	De 480 a 960 minutos	Débito Técnico Alto
5	Maior que 960 minutos	Débito Técnico Muito Alto

De acordo com AVRITZER et al. (2018), determinar a melhor configuração de implantação de uma aplicação requer a avaliação sistemática de abordagens quantitativas da engenharia de desempenho. Os autores sugerem que adicionar maior capacidade computacional para CPU ou aumentar o número de réplicas de contêiner do Docker pode não resultar em melhoria no desempenho do sistema. Essas descobertas sugerem que a análise de gargalos deve ser executada antes que recursos adicionais sejam adicionados à configuração de implantação da arquitetura.

A verificação de consumo de recursos computacionais possui por objetivo informar ao usuário o percentual de consumo de CPU e memória RAM em tempo real, não sendo o objetivo desta verificação o cálculo baseado em dados históricos. Para esta verificação, foi utilizada uma escala intervalar igualmente dividida, considerando-se o percentual de consumo de CPU ou RAM e dividindo-o pelo número de posições na escala, resultando em um intervalo de 20% para cada um dos cinco itens da escala de 0 a 100%, conforme pode ser observado na Tabela 4.9.

Tabela 4.9 – Escala de conversão de valores para recursos computacionais

Escala de Criticidade	Intervalo Considerado	Definição da Escala de Criticidade
1	De 0 a 20% de CPU ou RAM	Consumo de Recursos Comp. Muito Baixo
2	De 20 a 40% de CPU ou RAM	Consumo de Recursos Comp. Baixo
3	De 40 a 60% de CPU ou RAM	Consumo de Recursos Comp. Mediano
4	De 60 a 80% de CPU ou RAM	Consumo de Recursos Comp. Alto
5	De 80 a 100% de CPU ou RAM	Consumo de Recursos Comp. Muito Alto

De acordo com COSTA (2019), a alta disponibilidade significa um sistema de software em execução e disponível. Assim, a indisponibilidade, um atributo negativo, é medida contando o número de verificações nas quais o microsserviço não estava pronto para ser usado. Quanto maior esse número, pior é a disponibilidade de um determinado microsserviço.

Na abordagem M2CSEA, optou-se por um intervalo de duas verificações de indisponibilidade por nível na escala de criticidade. Esta verificação contempla um período de 24 horas, na qual são somadas as verificações de indisponibilidade que ocorrem a cada requisição ao microsserviço. A escala de criticidade relativa a disponibilidade pode ser observada na Tabela 4.10.

Tabela 4.10 – Escala de conversão de valores para disponibilidade

Escala de Criticidade	Intervalo Considerado	Definição da Escala de Criticidade
1	De 0 a 2 indisponibilidades	Disponibilidade Muito Alta
2	De 2 a 4 indisponibilidades	Disponibilidade Alta
3	De 4 a 6 indisponibilidades	Disponibilidade Mediana
4	De 6 a 8 indisponibilidades	Disponibilidade Baixa
5	Mais de 8 indisponibilidades	Disponibilidade Muito Baixa

4.2.4. Fase IV - Sugestão Baseada na Classificação de Criticidade

Baseado nos valores obtidos nas Fases I, II e III, são feitos novos cálculos critério a critério em relação a cada um dos microsserviços. Estes cálculos consideram os valores obtidos a partir da verificação da arquitetura em uso (Fase I), os pesos referentes a cada uma das verificações, tendo sido estes calculados a partir da abordagem AHP (Fase II), bem como as tabelas de conversão de criticidade (Fase III). Possuindo estes valores, calcula-se a multiplicação dos valores dos pesos por cada fator de criticidade, chegando ao valor final de criticidade para cada um dos microsserviços em relação a cada uma das verificações.

4.2.5. Exemplo de Uso da Abordagem M2CSEA

Nesta seção, é demonstrado um exemplo prático da abordagem M2CSEA, ilustrando de forma detalhada o passo a passo de cada fase e os resultados obtidos. Para este exemplo são considerados 5 microsserviços fictícios nomeados como MS1, MS2, MS3, MS4 e MS5.

Fase I - Coleta de dados da arquitetura

Nesta fase, são obtidos os dados referentes as opções de verificação mencionadas no item 4.3.1, sendo estes analisados e armazenados nos bancos de dados. Para o exemplo em questão são apresentados na Tabela 4.11 valores fictícios para cada uma das verificações, considerando hipoteticamente que os microsserviços foram avaliados e os valores foram obtidos em tempo de execução.

Tabela 4.11 – Coleta de dados da arquitetura em uso

Microsserviço	Verificação	Valor Obtido	Fator de Criticidade
MS1	Interdependência	Incoming: 2, Outgoing: 0, Criticality: 3	2
MS2	Interdependência	Incoming: 1, Outgoing: 0, Criticality: 1.5	1
MS3	Interdependência	Incoming: 0, Outgoing: 3, Criticality: 1.5	1
MS4	Interdependência	Incoming: 1, Outgoing: 1, Criticality: 2	2
MS5	Interdependência	Incoming: 4, Outgoing: 0, Criticality: 6	3
MS1	Tempo de Resposta	42 segundos	1
MS2	Tempo de Resposta	38 segundos	1
MS3	Tempo de Resposta	1387 segundos	3
MS4	Tempo de Resposta	2143 segundos	5
MS5	Tempo de Resposta	2130 segundos	5
MS1	Cobertura de Testes Unit.	0 % de código coberto	5
MS2	Cobertura de Testes Unit.	21 % de código coberto	4
MS3	Cobertura de Testes Unit.	40 % de código coberto	4
MS4	Cobertura de Testes Unit.	55 % de código coberto	3
MS5	Cobertura de Testes Unit.	30 % de código coberto	4
MS1	Débito Técnico	432 minutos	3
MS2	Débito Técnico	200 minutos	2
MS3	Débito Técnico	300 minutos	3
MS4	Débito Técnico	50 minutos	1
MS5	Débito Técnico	315 minutos	3
MS1	Cons. de Recursos Comp.	CPU: 40 % | RAM: 30 %	2
MS2	Cons. de Recursos Comp.	CPU: 5 % | RAM: 15 %	1
MS3	Cons. de Recursos Comp.	CPU: 10 % | RAM: 20 %	2
MS4	Cons. de Recursos Comp.	CPU: 25 % | RAM: 10 %	2
MS5	Cons. de Recursos Comp.	CPU: 62 % | RAM: 25 %	4
MS1	Disponibilidade	3 verificações com falha	2
MS2	Disponibilidade	1 verificações com falha	1
MS3	Disponibilidade	0 verificações com falha	1
MS4	Disponibilidade	0 verificações com falha	1
MS5	Disponibilidade	0 verificações com falha	1

Fase II – Execução do Método AHP

Nesta segunda fase, o usuário da abordagem M2CSEA realiza a seleção da comparação das verificações em pares, sendo na sequência realizado o ranqueamento da verificação mais importante em sua opinião, tendo sido atribuído o peso de acordo com a Seção 4.2.3. No exemplo em questão, o usuário realizou a comparação dos pares de verificação e obteve os resultados apresentados na Tabela 4.12, tendo sido obtidos os valores para validação mostrados na Tabela 4.13 e o resultado do ranqueamento mostrado na Tabela 4.14.

O valor obtido para o racional de consistência é então avaliado a fim de assegurar a consistência das escolhas, sendo necessário este ser inferior a 10%, conforme descrito na Seção 4.2.2.3.

Tabela 4.12 – Escolha do usuário para comparação de pares da verificação

	Interdependência	Tempo de Resposta	Cobertura de Testes Unitários	Débito Técnico	Cons. Recursos Comp.	Disponibilidade
Interdependência	1	3	4	5	2	0,250
Tempo de Resposta	0,330	1	3	4	2	0,166
Cobertura de Testes Unitários	0,250	0,330	1	0,500	0,200	0,142
Débito Técnico	0,200	0,250	2	1	0,200	0,125
Consumo de Recursos Comp.	0,500	0,500	5	5	1	0,200
Disponibilidade	4	6	7	8	5,000	1

Tabela 4.13 – Valores de validação para comparação de pares da verificação

Critério de Validação	Valor
Índice de Consistência	0,098223094
Racional de Consistência	7,44%

Tabela 4.14 – Resultado do ranqueamento dos pesos das verificações

Verificações	Peso Calculados
Disponibilidade	0,488304874
Interdependência	0,195394658
Tempo de Resposta	0,120004519
Consumo de Recursos Computacionais	0,11789407
Cobertura de Testes Unitários	0,036952678
Débito Técnico	0,0414492
Soma (Prova Real)	1

Fase III - Cálculo de criticidade

Nesta terceira fase, os dados das opções de verificação geram uma pontuação de criticidade de acordo com os dados obtidos nas fases I e II. Na Tabela 4.15, são apresentadas nas colunas “Fator de Criticidade” e “Peso Calculado”, respectivamente, os valores obtidos das Tabelas 4.11 e 4.14. Na coluna “Resultado”, é representada a multiplicação do valor de fator de criticidade pelo peso calculado, resultando assim no valor referente a criticidade referente a uma verificação para um determinado microsserviço.

Tabela 4.15 – Resultado do cálculo final

Microsserviço	Verificação	Fator de Criticidade	Peso Calculado	Resultado
MS1	Interdependência	2	0,195394658	0,39078932
MS2	Interdependência	1	0,195394658	0,19539466
MS3	Interdependência	1	0,195394658	0,19539466
MS4	Interdependência	2	0,195394658	0,39078932
MS5	Interdependência	3	0,195394658	0,58618397
MS1	Tempo de Resposta	1	0,120004519	0,12000452
MS2	Tempo de Resposta	1	0,120004519	0,12000452
MS3	Tempo de Resposta	3	0,120004519	0,36001356
MS4	Tempo de Resposta	5	0,120004519	0,6000226
MS5	Tempo de Resposta	5	0,120004519	0,6000226
MS1	Cob. de Testes Unit.	5	0,036952678	0,18476339
MS2	Cob. de Testes Unit.	4	0,036952678	0,14781071
MS3	Cob. de Testes Unit.	4	0,036952678	0,14781071
MS4	Cob. de Testes Unit.	3	0,036952678	0,11085803
MS5	Cob. de Testes Unit.	4	0,036952678	0,14781071
MS1	Débito Técnico	3	0,0414492	0,1243476
MS2	Débito Técnico	2	0,0414492	0,0828984
MS3	Débito Técnico	3	0,0414492	0,1243476
MS4	Débito Técnico	1	0,0414492	0,0414492
MS5	Débito Técnico	3	0,0414492	0,1243476
MS1	Cons. de Recursos Comp.	2	0,11789407	0,23578814
MS2	Cons. de Recursos Comp	1	0,11789407	0,11789407
MS3	Cons. de Recursos Comp	2	0,11789407	0,23578814
MS4	Cons. de Recursos Comp	2	0,11789407	0,23578814
MS5	Cons. de Recursos Comp	4	0,11789407	0,47157628
MS1	Disponibilidade	2	0,488304874	0,97660975
MS2	Disponibilidade	1	0,488304874	0,48830487
MS3	Disponibilidade	1	0,488304874	0,48830487
MS4	Disponibilidade	1	0,488304874	0,48830487
MS5	Disponibilidade	1	0,488304874	0,48830487

Fase IV - Sugestão baseada na classificação de criticidade

Nesta última fase, a classificação de criticidade é apresentada baseado na ordenação decrescente dos resultados obtidos na Tabela 4.15. Desta forma, os microsserviços mais críticos são mostrados ao usuário da abordagem M2CSEA. No exemplo em questão, o microsserviço mais crítico é o MS1 em função da verificação de disponibilidade, pois esta apresentou um fator de criticidade igual a 2 e peso calculado da verificação igual a 0,488304874, conforme apresentado na Tabela 4.16.

O fator de criticidade foi obtido dos dados apurados na Fase I e apresentados na Tabela 4.11. O peso calculado de verificação foi obtido baseado nas escolhas do usuário apuradas na Fase II e apresentados na Tabela 4.12, tendo sido efetuado o cálculo para peso da verificação e apresentado na Tabela 4.14.

Tabela 4.16 – Resultado final do ranqueamento dos microsserviços mais críticos

Microsserviço	Verificação	Fator de Criticidade	Peso Calculado da Verificação	Resultado
MS1	Disponibilidade	2	0,488304874	0,97660975
MS4	Tempo de Resposta	5	0,120004519	0,6000226
MS5	Tempo de Resposta	5	0,120004519	0,6000226
MS5	Interdependência	3	0,195394658	0,58618397
MS2	Disponibilidade	1	0,488304874	0,48830487
MS3	Disponibilidade	1	0,488304874	0,48830487
MS4	Disponibilidade	1	0,488304874	0,48830487
MS5	Disponibilidade	1	0,488304874	0,48830487
MS5	Cons. de Recursos Comp.	4	0,11789407	0,47157628
MS1	Interdependência	2	0,195394658	0,39078932
MS4	Interdependência	2	0,195394658	0,39078932
MS3	Tempo de Resposta	3	0,120004519	0,36001356
MS1	Cons. de Recursos Comp.	2	0,11789407	0,23578814
MS3	Cons. de Recursos Comp.	2	0,11789407	0,23578814
MS4	Cons. de Recursos Comp.	2	0,11789407	0,23578814
MS2	Interdependência	1	0,195394658	0,19539466
MS3	Interdependência	1	0,195394658	0,19539466
MS1	Cob. de Testes Unit.	5	0,036952678	0,18476339
MS2	Cob. de Testes Unit.	4	0,036952678	0,14781071
MS3	Cob. de Testes Unit.	4	0,036952678	0,14781071
MS5	Cob. de Testes Unit.	4	0,036952678	0,14781071
MS1	Débito Técnico	3	0,0414492	0,1243476
MS3	Débito Técnico	3	0,0414492	0,1243476
MS5	Débito Técnico	3	0,0414492	0,1243476
MS1	Tempo de Resposta	1	0,120004519	0,12000452
MS2	Tempo de Resposta	1	0,120004519	0,12000452
MS2	Cons. de Recursos Comp	1	0,11789407	0,11789407
MS4	Cobertura de Testes Unit	3	0,036952678	0,11085803
MS2	Débito Técnico	2	0,0414492	0,0828984
MS4	Débito Técnico	1	0,0414492	0,0414492

4.3. Conclusão

A abordagem proposta avalia, de acordo com os pesos atribuídos a cada um dos critérios, a relevância destes em relação aos demais. Os dados acerca da arquitetura estabelecida são coletados considerando as diferentes verificações trabalhadas, a saber: Interdependência entre microsserviços, tempo de resposta, cobertura de testes unitários, débito técnico, utilização de recursos computacionais e disponibilidade. Baseado nas informações obtidas, é realizado o cálculo de uma matriz ponderada, a fim de listar os microsserviços mais críticos e sugeri-los aos utilizadores do método.

Conforme descrito no Capítulo 2, após a realização de uma revisão da literatura, foi observado que o campo de pesquisa relativo à avaliação de criticidade em arquiteturas orientadas a microsserviços ainda foi pouco explorado, tendo sido encontrado somente o trabalho de SHAO, ZHANG e CAO (2018), no qual foi utilizada uma abordagem nomeada ACISM para análise da qualidade dos serviços.

A abordagem ACISM usa as informações de peso adquiridas dinamicamente para obter o resultado, usando-o para selecionar as instâncias dos microsserviços. A abordagem proposta por esta dissertação proporciona ao usuário final a capacidade de acompanhar em tempo real os microsserviços mais críticos em uma arquitetura estabelecida orientada a microsserviços, baseado no uso de múltiplos critérios de avaliação associados a opinião dos usuários, diferindo-se da abordagem de avaliação de microsserviços proposta por SHAO, ZHANG e CAO (2018), na qual são utilizados métodos de verificação não associados às opiniões dos usuários.

Diferenças adicionais da abordagem M2CSEA em relação a abordagem ACISM (SHAO; ZHANG; CAO, 2018), podem ser observadas a seguir:

- **Maior completude para consumo de recursos computacionais**: a abordagem M2CSEA utiliza o consumo de memória RAM e CPU, ao invés de somente CPU como a abordagem ACISM para avaliação de recursos computacionais.

- **Maior completude para tempo de resposta e disponibilidade**: a abordagem M2CSEA utiliza os dados obtidos em tempo real e também dados históricos com limite para dados obtidos até um ano antes da análise em questão. A abordagem ACISM não utiliza dados obtidos em tempo real para a composição do cálculo relativo ao tempo de resposta.

- **Inclusão da verificação de débito técnico**: a abordagem M2CSEA fornece ao usuário a avaliação de débito técnico, pois considera que este é um insumo relevante para a redução de possíveis não conformidades no código-fonte, que eventualmente podem gerar degradação ou indisponibilidade do serviço.

- **Inclusão da verificação de interdependência entre microsserviços**: a abordagem M2CSEA fornece ao usuário a avaliação de interdependência entre microsserviços, pois a falha de um componente com múltiplas dependências pode vir a gerar um impacto significativo na arquitetura em uso, podendo ocasionar degradação ou indisponibilidade do serviço.

- **Inclusão da verificação de cobertura de testes unitários**: a abordagem M2CSEA fornece ao usuário a avaliação de cobertura de testes unitários, pois a avaliação do código-fonte por meio de testes unitários pode indicar possíveis problemas e, eventualmente, um trecho de código não coberto com testes pode resultar em falhas em tempo de execução.

Apesar da diversidade de verificações a serem trabalhadas pela abordagem M2CSEA, esta não cobre todas as características e subcaracterísticas da ISO 25010:2011. Entretanto, esta é extensível pois fornece a base para cálculo da criticidade mediante a obtenção de dados complementares para que sejam introduzidas novas verificações e, por consequência, uma maior cobertura das características de qualidade propostas pelo ISO-25010:2011.

CAPÍTULO 5 – IMPLEMENTAÇÃO

5.1. Introdução

No capítulo anterior, a sistemática da abordagem *M2CSEA* foi apresentada em detalhes. O objetivo desta abordagem de sugestão dos microsserviços mais críticos em arquiteturas estabelecidas é apoiar a tomada de decisão, durante a utilização da arquitetura de software, com foco em auxiliar na escolha do investimento para manutenção ou evolução dos microsserviços.

Neste cenário, para ofertar meios para utilização prática da abordagem, um ferramental de suporte foi implementado. Este capítulo descreve o processo de implementação do ferramental elaborado. Além desta seção introdutória, este capítulo está organizado de forma que os requisitos iniciais para implementação do ferramental são apresentados na Seção 5.2. Na Seção 5.3, são detalhados pontos acerca das necessidades tecnológicas para a implementação do ferramental. Na Seção 5.4, uma visão geral dos componentes do ferramental de suporte é detalhada. Na Seção 5.5, um cenário de exemplo é descrito para ilustrar o uso da abordagem por meio do ferramental de suporte implementado. Por fim, as considerações finais são apresentadas na Seção 5.6.

5.2. Requisitos

Com base nas observações realizadas ao conduzir a fundamentação teórica, apresentadas no Capítulo 2, a validação com especialistas no Capítulo 3, e a abordagem M2CSEA apresentada no Capítulo 4, requisitos foram identificados para implementação do ferramental de suporte da abordagem. Nos itens a seguir são detalhados os requisitos funcionais para a implementação da abordagem:

- **A ferramenta proposta deve ser capaz de coletar e processar os dados referentes a interdependência entre os microsserviços (RQF 1)**

  A ferramenta proposta deve ser capaz de receber informações acerca da interdependência existente entre os microsserviços de forma atualizada para que seja efetuado o cálculo baseado na fórmula exposta na Seção 4.2.1.1. Para este requisito, a ferramenta também deve calcular o fator de criticidade associado a verificação de interdependência entre microsserviços e considerar a Tabela 4.5 exposta na Seção 4.2.3, para normalização dos valores.

- **A ferramenta proposta deve ser capaz de coletar e processar os dados referentes ao consumo de recursos computacionais (RQF 2)**

  A ferramenta proposta deve ser capaz de obter de forma individualizada o consumo de recursos computacionais de cada um dos microsserviços contidos em uma arquitetura em uso. Para este requisito, a ferramenta deve calcular o fator de criticidade associado a verificação de consumo de recursos computacionais e considerar a Tabela 4.9 exposta na Seção 4.2.3, para normalização dos valores.

- **A ferramenta proposta deve ser capaz de coletar e processar os dados referentes a disponibilidade dos microsserviços (RQF 3)**

  A ferramenta proposta deve ser capaz de obter os dados acerca da indisponibilidade dos microsserviços de uma arquitetura em uso. Para este requisito, a ferramenta também deve calcular o fator de criticidade associado a disponibilidade dos microsserviços e considerar a Tabela 4.10 exposta na Seção 4.2.3, para normalização dos valores.

- **A ferramenta proposta deve ser capaz de coletar e processar os dados referentes ao tempo de resposta dos microsserviços (RQF 4)**

  A ferramenta proposta deve ser capaz de obter os dados referentes ao tempo de resposta das interfaces dos microsserviços de uma arquitetura em uso e ser capaz de agregá-los. Para este requisito, a ferramenta também deve calcular o fator de criticidade associado ao tempo de resposta dos microsserviços e considerar a Tabela 4.6 exposta na Seção 4.2.3, para normalização dos valores.

- **A ferramenta proposta deve ser capaz de coletar e processar os dados referente ao débito técnico dos microsserviços (RQF 5)**

  A ferramenta proposta deve ser capaz de obter os dados referentes ao débito técnico calculado em minutos a partir do método SQALE e ser capaz de agregá-los. Para este requisito, a ferramenta também deve calcular o fator de criticidade associado ao débito técnico dos microsserviços e considerar a Tabela 4.8 exposta na Seção 4.2.3, para normalização dos valores.

- **A ferramenta proposta deve ser capaz de coletar e processar os dados referentes a cobertura de testes unitários dos microsserviços (RQF 6)**

  A ferramenta proposta deve ser capaz de obter os dados referentes a cobertura de testes unitários utilizando os valores em termos percentuais. Para este requisito, a ferramenta também deve calcular o fator de criticidade associado a cobertura de testes unitários dos microsserviços e considerar a Tabela 4.7 exposta na Seção 4.2.3, para normalização dos valores.

- **A ferramenta proposta deve suportar, de acordo com a preferência do usuário, a escolha da importância de forma comparativa em relação a interdependência, consumo de recursos computacionais, disponibilidade, tempo de resposta, débito técnico e cobertura de testes unitários dos microsserviços (RQF 7)**

  A ferramenta proposta deve ser capaz de obter o ranqueamento para todas as verificações de acordo com a preferência e necessidade do usuário, sendo também capaz de armazená-las para posterior uso.

- **A ferramenta proposta deve identificar, baseado na coleta dos dados obtidos nos RQF 1 a 6 e na importância comparativa obtida no item RQF 7, os microsserviços mais críticos (RQF 8)**

  A ferramenta proposta deve ser capaz de obter todos os fatores de criticidade calculados pelos requisitos RQF1 a RQF6, e realizar um novo cálculo relacionando-os com o requisito RQF7, no qual são obtidas as preferências do usuário para cada uma das verificações. A partir deste novo cálculo são obtidos os microsserviços mais críticos de forma orientada às preferências do usuário.

- **A ferramenta proposta deve exibir os microsserviços mais críticos baseado nos dados coletados e processados pelos requisitos RQF 1 a 6 e baseado no cálculo efetuado no RQF 8 (RQF 9)**

  A ferramenta proposta deve ser capaz de exibir os dados coletados e os microsserviços mais críticos por cada uma das verificações a partir do fator de criticidade, conforme os cálculos realizados nos requisitos RQF 1 a RQF 6, também sendo capaz de exibir os microsserviços mais críticos de forma agrupada e associada às preferência dos usuários, conforme o cálculo realizado no RQF 8.

5.3. Viabilidade Tecnológica

A fim de criar um ferramental para implementação da abordagem M2CSEA, foi necessário estabelecer premissas acerca da viabilidade tecnológica associada a cada uma das verificações expostas na Seção 4.2.1. A seguir serão detalhados os tópicos associados às verificações e necessidades relativas a cada um destes.

5.3.1. Cálculo de Débito Técnico com o Método SQALE e SonarQube

Para a implementação da verificação de débito técnico, foi escolhido o método SQALE, pois este possui grande aceitação na literatura e diversos trabalhos publicados utilizando-o. Para a verificação de débito técnico na abordagem M2CSEA, foi utilizado o plug-in[15] do método SQALE para a ferramenta SonarQube, pois segundo HEGEMAN (2011) muitas organizações em todo o mundo frequentemente o utilizam para monitorar o débito técnico.

O cálculo para a verificação de débito técnico se baseia na definição de índices e indicadores definidos pelo método SQALE. Para isso, o método organiza e agrupa os requisitos de acordo com uma cronologia específica, sendo possível rastreá-los em verificações anteriores (LETOUZEY e ILKIEWICZ, 2012).

O método SQALE usa um indicador para representar a distribuição específica do débito técnico para cada característica selecionada. Esse indicador chamado de pirâmide, pode ser utilizado de duas maneiras, a primeira através da visão analítica, sendo a distribuição do débito por característica, a segunda através da visão consolidada, sendo a soma do débito para uma dada característica e todas as características listadas na Figura 5.1 (LETOUZEY e ILKIEWICZ, 2012).

Para esta dissertação de mestrado, a visão analítica com a posterior agregação dos valores por microsserviços é utilizada independente da característica que originou o débito técnico.

[15] Plug-in: programa de computador usado para adicionar funções a outros programas maiores, provendo alguma funcionalidade especial ou muito específica

Figura 5.1 – Características do SQALE (LETOUZEY e ILKIEWICZ, 2012)

Na Figura 5.1, são listadas as características apresentadas pelo método SQALE, as quais são associadas a requisitos. Cada um dos requisitos criados pelo método possui um tempo de solução da não-conformidade associada ao não cumprimento do requisito. Após a execução da análise do código-fonte a ferramenta SonarQube, utilizando-se do método SQALE lista todas as não-conformidades encontradas e o tempo de solução associado a cada uma destas, conforme exemplo contido na Tabela 5.1. À soma de todos os tempos de solução das não-conformidades é atribuído o nome de débito técnico ou dívida técnica.

Tabela 5.1 – Exemplos de requisito e tempo de solução baseado no SQALE. Adaptada de LETOUZEY e ILKIEWICZ (2012)

Característica	Requisito	Tempo de Solução
Manutenibilidade	Não há nenhum bloco de instruções comentado	2 minutos por ocorrência
	A identação do código deve seguir uma regra consistente	2 minutos por arquivo, independente do número de violações
Modificabilidade	Não há dependência cíclica entre pacotes	1 hora por arquivo dependente a ser removido
Confiabilidade	O tratamento de exceções não deve capturar exceção de ponteiro nulo	40 minutos por ocorrência
	O código substituirá "*equals*" e "*hashcode*"	1 hora por ocorrência
	Não há comparação entre pontos flutuantes	40 minutos por ocorrência
	Nenhuma variável de iteração é modificada no corpo de um loop	40 minutos por ocorrência
Testabilidade	Não existe método com complexidade ciclomática acima de 12	1 hora por ocorrência se a medida é < 24; 2 horas por ocorrência se a medida é > 24
	Não há partes duplicadas de 100 caracteres ou mais	20 minutos por ocorrência

5.3.2. Cobertura de Testes Unitários

Para a implementação da verificação da cobertura de testes unitários, é utilizada a abordagem de acordo com a linguagem utilizada na criação dos microsserviços a serem analisados. Para a obtenção dos percentuais de cobertura, é utilizada a ferramenta SonarQube[16], na qual diferentes implementações são utilizadas para realizar o cálculo de cobertura, sendo sempre associados à linguagem de programação na qual foi escrito o microsserviço em questão. Na Tabela 5.2, são exibidas as linguagens e algoritmos utilizados para cada uma destas linguagens.

Tabela 5.2 – Implementações de análise de cobertura de testes unitários

Linguagem de Programação	Implementação
Apex	Baseado na ferramenta SonarApex[17]
C, C++, Objective-C	Baseado na ferramenta SonarCFamily[18]
C#	Baseado na ferramenta SonarC#[19]
Flex	Baseado na ferramenta SonarFlex[20]
Go	Baseado na ferramenta SonarGo[21]
Java / Kotlin / Scala	Baseado na ferramenta JaCoCo[22]
Javascript	Baseado na ferramenta SonarJS[23]
PHP	Baseado na ferramenta SonarPHP[24]
Python	Baseado na ferramenta SonarPython[25]

5.3.3. Conteinerização e Consumo de Recursos Computacionais

Para a adequada separação dos microsserviços em servidores apartados com o objetivo de possuir a obtenção mais fiel possível e gerenciamento dos recursos computacionais disponíveis e alocados, esta dissertação de mestrado utiliza contêineres.

De acordo com BURNS et al. (2016), os primeiros contêineres apenas forneciam isolamento do sistema de arquivos (*file system*). O isolamento de recursos fornecido pelos contêineres permite, por exemplo, designar lotes de tarefas ao sistema operacional de forma independente dos demais contêineres em execução. Os contêineres fornecem as ferramentas de gerenciamento de recursos que tornam isso possível, bem como o

[16] SonarQube: https://www.sonarqube.org
[17] SonarApex: https://www.sonarsource.com/products/codeanalyzers/sonarapex.html
[18] SonarCFamily: https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforc.html
[19] SonarC#: https://www.sonarsource.com/products/codeanalyzers/sonarcsharp.html
[20] SonarFlex: https://www.sonarsource.com/products/codeanalyzers/sonarflex.html
[21] SonarGo: https://www.sonarsource.com/products/codeanalyzers/sonargo.html
[22] JaCoCo: https://www.eclemma.org/jacoco
[23] SonarJS: https://www.sonarsource.com/products/codeanalyzers/sonarjs.html
[24] SonarPHP: https://www.sonarsource.com/products/codeanalyzers/sonarphp.html
[25] SonarPython: https://www.sonarsource.com/products/codeanalyzers/sonarpython.html

isolamento de recursos no nível do *kernel*[26] para impedir que os processos interfiram entre si (BURNS et al., 2016).

O ferramental criado para implementar a abordagem M2CSEA utiliza a ferramenta Docker[27] e sua extensão Docker-Compose[28] como gerenciador de contêineres, a fim de possuir a separação de recursos necessários para os microsserviços.

5.3.4. Tempo de Resposta e a Influência da Latência

Para a verificação do tempo de resposta, é importante estabelecer a definição dos termos acerca dos itens a serem medidos. Para esta verificação, a abordagem M2CSEA utiliza o tempo de resposta de cada uma das interfaces de cada microsserviço contido em uma arquitetura em uso, mas desconsiderando a latência de rede.

A latência é o atraso que ocorre após a execução de uma operação de envio e antes que os dados comecem a chegar ao computador de destino. A latência pode ser medida como o tempo necessário para transferir uma mensagem vazia de um computador a outro, ou mesmo de um conector de rede a outro. A taxa de transferência de dados é a velocidade na qual os dados podem ser transferidos entre dois computadores na rede após o início da transmissão, geralmente citados em bits por segundo (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Após essas definições, o tempo necessário para uma rede transferir uma mensagem entre dois computadores é:

$$TTM^{29} = latência + tamanho\ do\ pacote / taxa\ de\ transferência\ de\ dados$$

(COULOURIS; DOLLIMORE; KINDBERG, 2005)

A equação acima é válida para mensagens cujo tamanho não exceda o máximo determinado pela tecnologia de rede utilizado, pois mensagens mais longas precisam ser segmentadas e o tempo de transmissão é a soma dos tempos para os segmentos. A taxa de transferência de uma rede é determinada, principalmente, por suas características físicas, enquanto a latência é determinada principalmente por sobrecargas de software, atrasos de roteamento e um elemento estatístico dependente da carga resultante de

[26] Kernel: Núcleo do sistema operacional
[27] Docker: https://www.docker.com
[28] Docker-Compose: https://docs.docker.com/compose
[29] TTM: Tempo de Transmissão da Mensagem

demandas conflitantes por acesso aos canais de transmissão. Muitas das mensagens transferidas entre processos em sistemas distribuídos são pequenas, portanto, a latência costuma ter significado igual ou maior que a taxa de transferência na determinação do desempenho (COULOURIS; DOLLIMORE; KINDBERG, 2005).

A largura de banda total do sistema de uma rede é uma medida da taxa de transferência - o volume total de tráfego que pode ser transferido pela rede em um determinado momento. Em muitas tecnologias de rede local, como Ethernet, a capacidade total de transmissão da rede é usada para cada transmissão e a largura de banda do sistema é a mesma que a taxa de transferência de dados. Porém, na maioria das redes de área ampla (*wide area networks*), as mensagens podem ser transferidas em vários canais diferentes, simultaneamente, e a largura de banda total do sistema não tem relação direta com a taxa de transferência (COULOURIS; DOLLIMORE; KINDBERG, 2005).

O desempenho das redes se deteriora em condições de sobrecarga - quando há muitas mensagens na rede ao mesmo tempo. O efeito da sobrecarga na latência, taxa de transferência de dados e largura de banda total do sistema de uma rede depende fortemente da tecnologia da rede (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Acerca da virtualização de rede (PETERSEN et al., 2005), esta possui relação com a construção de redes virtuais diferentes em uma rede existente, como a Internet. Cada rede virtual pode ser projetada para suportar múltiplas aplicações de forma distribuída, sem que haja interferência entre as múltiplas redes virtuais. Isso sugere uma resposta ao dilema levantado pelo argumento de ponta a ponta (SALTZER; REED; CLARK, 1984). Uma rede virtual específica de uma aplicação pode ser construída acima de uma rede existente e otimizada para essa aplicação em particular, sem alterar as características da rede subjacente (COULOURIS; DOLLIMORE; KINDBERG, 2005).

Para a abordagem M2CSEA, é utilizada a abordagem de virtualização de rede, com uso de baixa latência por não haver tráfego em meios físicos, tendo em vista que todos os contêineres da abordagem estão localizados no mesmo servidor. Desta forma, não há impacto da latência sobre o tempo de respostas das APIs medidas.

5.3.5. Service Discovery

Os sistemas distribuídos de microsserviços podem superar as deficiências das arquiteturas monolíticas tradicionais, permitindo independência de desenvolvimento, implantação, atualização e dimensionamento de componentes, mas a engenharia desses sistemas não é isenta de desafios. A crescente adoção de microsserviços na nuvem é motivada pela facilidade de implantação e atualização do software, bem como pelo acoplamento flexível provisionado pelo dinamismo do *service discovery* (STUBBS; MOREIRA; DOOLEY, 2015) (ESPOSITO; CASTIGLIONE; CHOO, 2016).

O *service discovery* garante que os microsserviços possam ser localizados. Para que uma arquitetura em uso seja auto gerenciada em relação a escalabilidade, seus microsserviços precisam possuir a localização dos microsserviços dos quais este depende, para que sejam capazes de se adaptar a um ambiente em mudança, por exemplo, novas instâncias de microsserviços sendo adicionados ou instâncias existentes sendo removidas.

As arquiteturas de microsserviços devem ser dinâmicas, pois novas instâncias de microsserviço são criadas para atender à demanda e são removidas tão logo haja a redução da carga de trabalho, sendo desta forma necessário alto dinamismo. Portanto, uma configuração fixa acerca da localização dos serviços não será suficiente para lidar com esses cenários (WOLFF, 2016) (TOFFETTI et al., 2015) (LE et al., 2015).

O ferramental criado para implementar a abordagem M2CSEA utiliza a ferramenta Consul[30] como *service discovery*, a fim de possuir de forma atualizada a localização dos serviços.

5.3.6. Service Registry

Quando todos os microsserviços possuem uma abordagem comum para a utilização de *service discovery*, um *service registry* centralizado de todos os microsserviços se faz necessário. A utilização de *service registry* pode ser útil para uma visão geral da arquitetura e para obter informações de monitoramento acerca dos microsserviços registrados neste (WOLFF, 2016).

O *service registry* é um banco de dados de serviços, no qual as instâncias dos serviços são registradas em sua inicialização e removidas na finalização do serviço, em

[30] Consul: https://www.consul.io

caso de uma finalização esperada (*friendly shutdown*). Os clientes do *service discovery* o consultam para localizar as instâncias disponíveis de um determinado serviço.

Um *service registry* está disponível para registrar endereços de rede e portas de serviço durante a execução do serviço e é usado em conjunto com o *service discovery* de tempo de execução. Além das funções citadas, o *service discovery* pode ser usado para recuperar informações de interação entre os componentes (MONTESI e WEBER, 2016) (BERG; SIEGEL; CRAMP, 2017).

O ferramental criado para implementar a abordagem M2CSEA utiliza a ferramenta Consul como *service registry*, a fim de possuir de forma atualizada os dados de rede dos microsserviços, como endereço de rede e as portas dos serviços disponíveis.

5.3.7. Disponibilidade e *Healthcheck*

Comumente, o *service discovery* está fortemente alinhada com o monitoramento de serviço, de modo que uma instância de serviço apenas obtenha e mantenha somente um registro de serviço após passar com êxito em seus testes de monitoramento, também conhecidos como verificação de integridade ou verificação de saúde (*healthcheck*) (STUBBS; MOREIRA; DOOLEY, 2015).

Para a verificação de saúde, a ferramenta de *service discovery* faz consultas a API de verificação de integridade de diversas instâncias dos serviços para verificar se esta é capaz de lidar com as solicitações (DAYA et al., 2016) (STUBBS; MOREIRA; DOOLEY, 2015).

O ferramental criado para implementar a abordagem M2CSEA utiliza a ferramenta Consul como fornecedor das informações acerca da disponibilidade baseado na consulta que esta ferramenta faz às APIs de verificação de saúde (*healthcheck*) contidas em cada um dos microsserviços a serem analisados.

5.3.8. K/V Store

Um *K/V Store* (armazenador de chave/valor), ou banco de dados de chave-valor, é um modelo de armazenamento de dados projetado para armazenar, recuperar e gerenciar matrizes associativas, sendo uma estrutura de dados mais conhecida como dicionário ou tabela de *hash*[31]. Os dicionários contêm uma coleção de objetos ou registros que, por sua vez, podem conter múltiplos campos diferentes. Os registros são

[31] *Hash*: Mapeamento de dados de comprimento variável para dados de comprimento fixo

armazenados e recuperados usando uma chave que identifica exclusivamente o registro e são usados para encontrar rapidamente os dados no banco de dados (TWEED e JAMES, 2010).

O ferramental criado para implementar a abordagem M2CSEA utiliza a ferramenta Consul como *K/V Store*, a fim de possuir de forma atualizada a lista de dependências de cada um dos microsserviços, sendo esta lista inserida em um banco de dados orientado a grafos para posterior consulta.

5.3.9. Interdependência e o uso de Banco de Dados Orientado a Grafos

O banco de dados de grafos pode utilizar efetivamente o modelo de grafos para armazenar, gerenciar, atualizar dados e seus relacionamentos, fazendo uso de esquemas flexíveis (*schemaless*). Como evidenciado por JOUILI e VANSTEENBERGHE (2013), o banco de dados Neo4J[32] possui os melhores resultados em testes de *benchmark*[33], superando os bancos BerkeleyDB[34], Orient[35] e DEX[36], independentemente da carga de trabalho ou dos parâmetros utilizados. Com relação às cargas de trabalho intensivas somente leitura, Neo4j, BerkeleyDB, Orient e DEX obtiveram desempenhos semelhantes.

O Neo4j é um robusto banco de dados de grafos transacional, de código aberto e escrito pela linguagem de programação Java. No Neo4j, os grafos são compostos por um grande número de vértices e relacionamentos. Os vértices e os relacionamentos podem ter propriedades expressas na forma de pares de chave-valor. Os métodos de consulta do Neo4j incluem Java Core API[37], Traversal Framework[38] e Cypher Query Language[39] (HUANG e DONG, 2013).

O ferramental criado para implementar a abordagem M2CSEA utiliza a ferramenta Neo4J como banco de dados orientado a grafos, a fim de consultar a relação de interdependência entre os microsserviços de forma eficiente, com reduzido tempo de resposta.

[32] NEO4J: https://neo4j.com
[33] Benchmark: execução de programas de computador para comparação de desempenho
[34] BerkeleyDB: https://www.oracle.com/database/technologies/related/berkeleydb-downloads.html
[35] Orient: https://orientdb.com
[36] DEX: http://sparsity-technologies.com/#sparksee
[37] Java Core API: https://docs.oracle.com/javase/10/core/java-core-libraries1.htm
[38] Transversal Framework: https://neo4j.com/docs/java-reference/current/tutorial-traversal
[39] Cypher Query Language: https://neo4j.com/developer/cypher-query-language

5.3.10. Descrição das APIs e OpenAPI

As linguagens de descrição de APIs REST, como Swagger[40] e RAML[41], estão se tornando um padrão na indústria e na academia, resultando na criação da Open API Initiative[42] como uma abordagem de padronização para descrições de API (HAUPT et al., 2017). Para a descrição das APIs contidas neste capítulo, é utilizada a linguagem Swagger.

5.4. Protótipo

A ferramenta de avaliação da criticidade dos microsserviços, chamado neste trabalho de M2CSEA (*Multicriteria Microservice Criticality Suggestion for Established Architectures*), é composta por componentes com função específica, vislumbrando a entrega de valor por meio da descoberta e sugestão dos microsserviços mais críticos de uma arquitetura seguindo o modelo proposto nas seções anteriores.

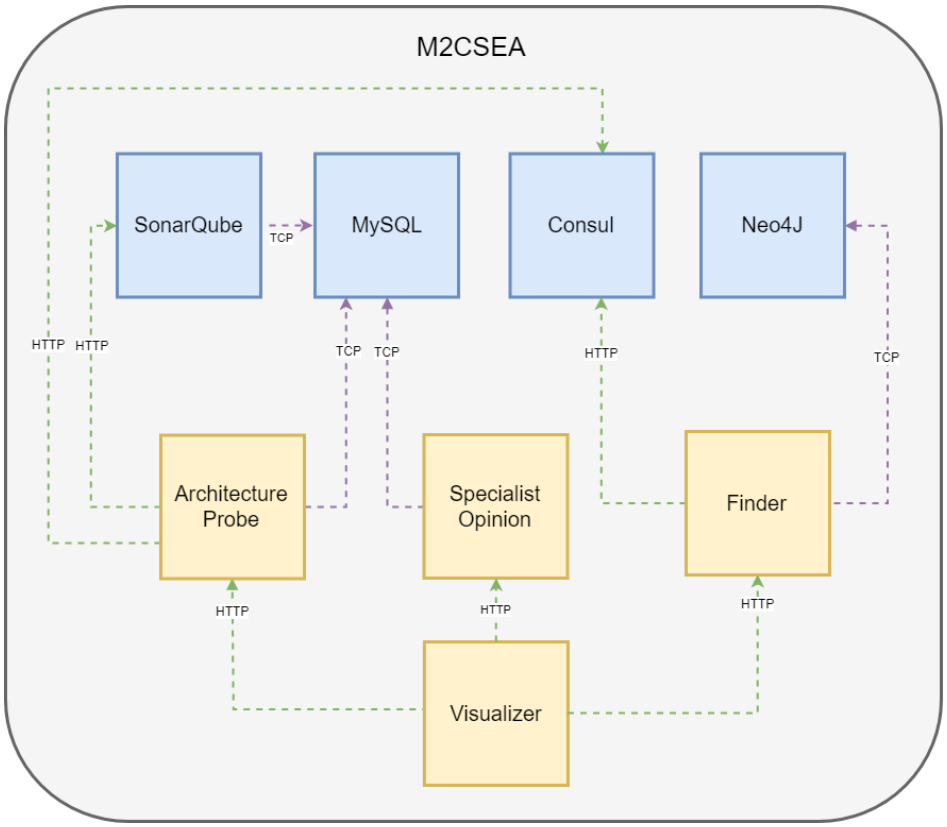


Figura 5.2 – Arquitetura da ferramenta baseada na abordagem M2CSEA

[40] Swagger: https://swagger.io
[41] RAML: https://raml.org
[42] Open API Initiative: https://www.openapis.org

Na Figura 5.2, são mostrados em amarelo os microsserviços criados para a abordagem M2CSEA, e em azul são exibidas as ferramentas utilizadas para suportar a necessidade de obtenção e armazenamento de dados.

Nas seções seguintes, são detalhados todos os itens desta arquitetura, e para uma melhor contextualização acerca da rastreabilidade dos requisitos e suas implementações, é exibida na Figura 5.3 a relação entre as verificações da abordagem M2CSEA e os requisitos expostos na Seção 5.2.

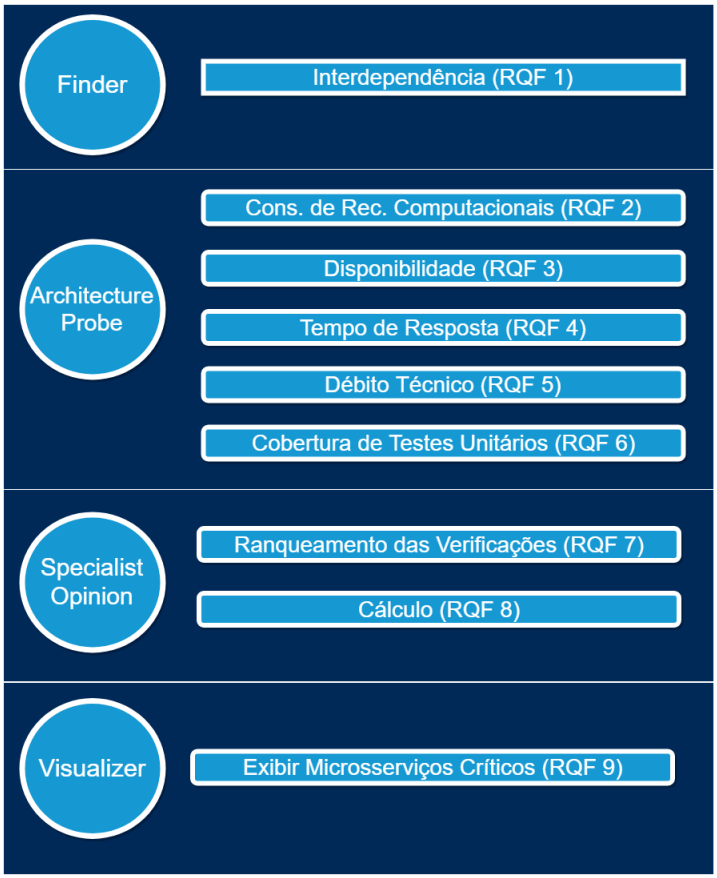


Figura 5.3 – Correlação entre os requisitos e ferramental da abordagem M2CSEA

No Apêndice B, estão listados os diagramas de sequência referentes a cada um dos serviços e suas funcionalidades, ilustrando a interação entre os componentes da implementação realizada com base na abordagem M2CSEA.

5.4.1. Serviço *Architecture Probe*

No serviço, *Architecture Probe* são implementados os requisitos funcionais RQF 2 a 6, referentes a obtenção e tratamento dos dados relativos às verificações de consumo

de recursos computacionais, disponibilidade, tempo de resposta, débito técnico e cobertura de testes unitários.

O serviço *Architecture Probe* possui acesso a dados a partir do banco de dado relacional (MySQL[43]), ao analisador estático com a implementação do SQALE (SonarQube), e ao *service discovery*, *service registry* e *k/v store* (Consul), também expondo interfaces REST para serem consumidas pelo serviço de visualização que será detalhado na Seção 5.4.4. A documentação das interfaces expostas seguindo o padrão *OpenAPI*, podem ser encontradas no próprio serviço em uma *url* específica (/swagger-api.html), como mostrado no Item I do Apêndice C.

A seguir são apresentados detalhes acerca da implementação dos requisitos RQF 2 e 6 referentes à obtenção e tratamento dos dados relativos às verificações de consumo de recursos computacionais na Seção 5.4.1.1, disponibilidade na Seção 5.4.1.2, tempo de resposta na Seção 5.4.1.3, débito técnico na Seção 5.4.1.4 e cobertura de testes unitários na Seção 5.4.1.5.

5.4.1.1. Verificação de Consumo de Recursos Computacionais

Para o processo referente à verificação de consumo de recursos computacionais, o serviço *Architecture Probe*, comunica-se com o gerenciador de contêineres (Docker), e a partir deste obtém os dados relativos ao consumo de memória RAM e CPU para cada um dos microsserviços que estiverem em execução no momento.

Após a obtenção dos dados relativos ao consumo de recursos computacionais, o serviço *Architecture Probe* realiza a inserção destes dados no banco de dados relacional (MySQL). Na sequência, realiza-se o cálculo do fator de criticidade de acordo com a Tabela 4.9 presente na Seção 4.2.3, e o armazena no banco de dados relacional para posterior consulta.

5.4.1.2. Verificação da Disponibilidade

Para o processo referente à verificação de consumo de recursos computacionais, o serviço *Architecture Probe*, comunica-se com o sistema de *service discovery* (Consul) para verificar o status dos últimos *healthchecks* de cada um dos microsserviços, e a partir destes obtém a lista dos microsserviços indisponíveis no momento da verificação.

[43] MySQL: https://www.mysql.com

Após a obtenção dos dados relativos à indisponibilidade dos microsserviços, o serviço *Architecture Probe* realiza a inserção destes dados no banco de dados relacional (MySQL). Na sequência, realiza o cálculo do fator de criticidade de acordo com a Tabela 4.10 presente na Seção 4.2.3, e o armazena no banco de dados relacional para posterior consulta.

5.4.1.3. Verificação do Tempo de Resposta

Para o processo referente à verificação do tempo de resposta, cada microsserviço faz a inserção no banco de dados relacional (MySQL) após cada requisição a qualquer uma das APIs expostas por ele. Posteriormente, de forma assíncrona, o serviço *Architecture Probe,* comunica-se com o banco de dados relacional (MySQL) para verificar os tempos de respostas dos microsserviços de acordo com um período definido dentre as opções – um dia, sete dias ou trinta dias.

Na sequência, realiza-se o cálculo do fator de criticidade de acordo com a Tabela 4.6, presente na Seção 4.2.3, e o armazena no banco de dados relacional para posterior consulta.

5.4.1.4. Verificação do Débito Técnico

Para o processo referente à verificação do débito técnico, cada microsserviço faz o envio do relatório de análise de débito técnico para a ferramenta SonarQube após cada modificação de versão do serviço contido na arquitetura em uso. Posteriormente, de forma assíncrona, o serviço *Architecture Probe,* comunica-se com a ferramenta SonarQube para verificar todas as não conformidades referentes a todos os microsserviços na arquitetura em uso e que possuam os dados armazenados na ferramenta SonarQube.

Na sequência, realiza-se a soma de todos os débitos técnicos na escala de minutos e realiza o cálculo do fator de criticidade de acordo com a Tabela 4.8, presente na Seção 4.2.3, armazenando-os na sequência no banco de dados relacional para posterior consulta.

5.4.1.5. Verificação de Cobertura de Testes Unitários

Para o processo referente à verificação da cobertura de testes unitários, cada microsserviço faz o envio do relatório de análise de cobertura de testes unitários, de

acordo com a linguagem que foi construído, para a ferramenta SonarQube após cada modificação de versão do serviço contido na arquitetura em uso. Posteriormente, de forma assíncrona, o serviço *Architecture Probe*, comunica-se com a ferramenta SonarQube para verificar o percentual de cobertura de todos os microsserviços na arquitetura em uso e que possuam os dados armazenados na ferramenta SonarQube.

Na sequência, realiza-se o cálculo do fator de criticidade de acordo com a Tabela 4.7, presente na Seção 4.2.3, armazenando-os na sequência no banco de dados relacional para posterior consulta.

5.4.2. Serviço *Finder*

No serviço *Finder*, é implementado o requisito funcional RQF 1, referente à capacidade de coletar e processar os dados referentes a interdependência entre os microsserviços. Neste serviço, foi implementado um algoritmo de classificação para elencar os microsserviços de acordo com a interdependência entre eles. Nesta análise, são compiladas as dependências dos microsserviços e utiliza-se um banco de dados orientado a grafos (Neo4J) para armazenar a relação de dependência entre os microsserviços.

Para o processo de descoberta dos microsserviços, é utilizado um sistema (Consul) capaz de atuar como registrador de serviço (*service registry)* e localizador de serviços (*service discovery*) e armazenador de chave/valor (*key-value Store*), no qual é armazenado um dado contendo uma chave e um valor, sendo possível resgatar tais valores posteriormente por meio das chaves armazenadas.

O serviço *Finder* possui acesso a dados a partir do banco de dado orientado a grafos (Neo4J), e ao serviço de *service discovery*, *service registry* e *K/V Store* (Consul). O serviço *Finder* também expõe interfaces REST para serem consumidas pelo serviço de visualização que será detalhado na Seção 5.4.4. A documentação das interfaces, expostas seguindo o padrão OpenAPI, podem ser encontradas no próprio serviço em uma *url* específica (/swagger-api.html), como mostrado no Item II do Apêndice C.

A execução deste serviço pode ser dividida em dois processos diferentes para implementação do requisito RQF 1, a obtenção dos dados da interdependência entre os microsserviços e a análise dos dados obtidos. A seguir, na Seção 5.4.2.1, são apresentados detalhes acerca da obtenção dos dados de interdependência e na Seção

5.4.2.2 expõe-se detalhes do processamento dos dados referente a interdependência entre os microsserviços.

5.4.2.1. Processo de Obtenção dos Dados de Interdependência

No processo de inicialização, cada microsserviços realiza uma chamada para o processo de *service register* e *service Discovery*, na sequência, procura internamente dependências, seguido de outra chamada para o *service discovery* com a intenção de preencher o *K/V Store*, adicionando ou atualizando suas dependências.

Considerando o sistema de *Service Discovery* com um *KV Store* contendo dados de cada microsserviços envolvido em uma arquitetura, uma ampla análise de dependências pode ser realizada. O projeto do *Finder* é responsável por registrar-se no *Service Discovery*, consultar o *KV Store* para informações de dependências dos microsserviços, analisar essas informações e preencher o banco de dados orientado a grafos (Neo4J).

5.4.2.2. Processo de Análise dos Dados Obtidos

Para o processo de análise dos dados, considera-se que o banco de dados orientado a grafos (Neo4j) conterá previamente os dados referentes as interdependências dos microsserviços, tendo em vista que este atual processo depende da execução do anterior, responsável pela obtenção dos dados.

O processamento de armazenamento segue a fluxograma exposto na Figura 5.4, onde os microsserviços, representados por um vértice no banco de dados de grafos (Neo4J) serão localizados, bem como suas dependências. Esse processo é necessário para recuperar o identificador único do vértice e reutilizar as informações já armazenadas no banco de dados para evitar a duplicação.

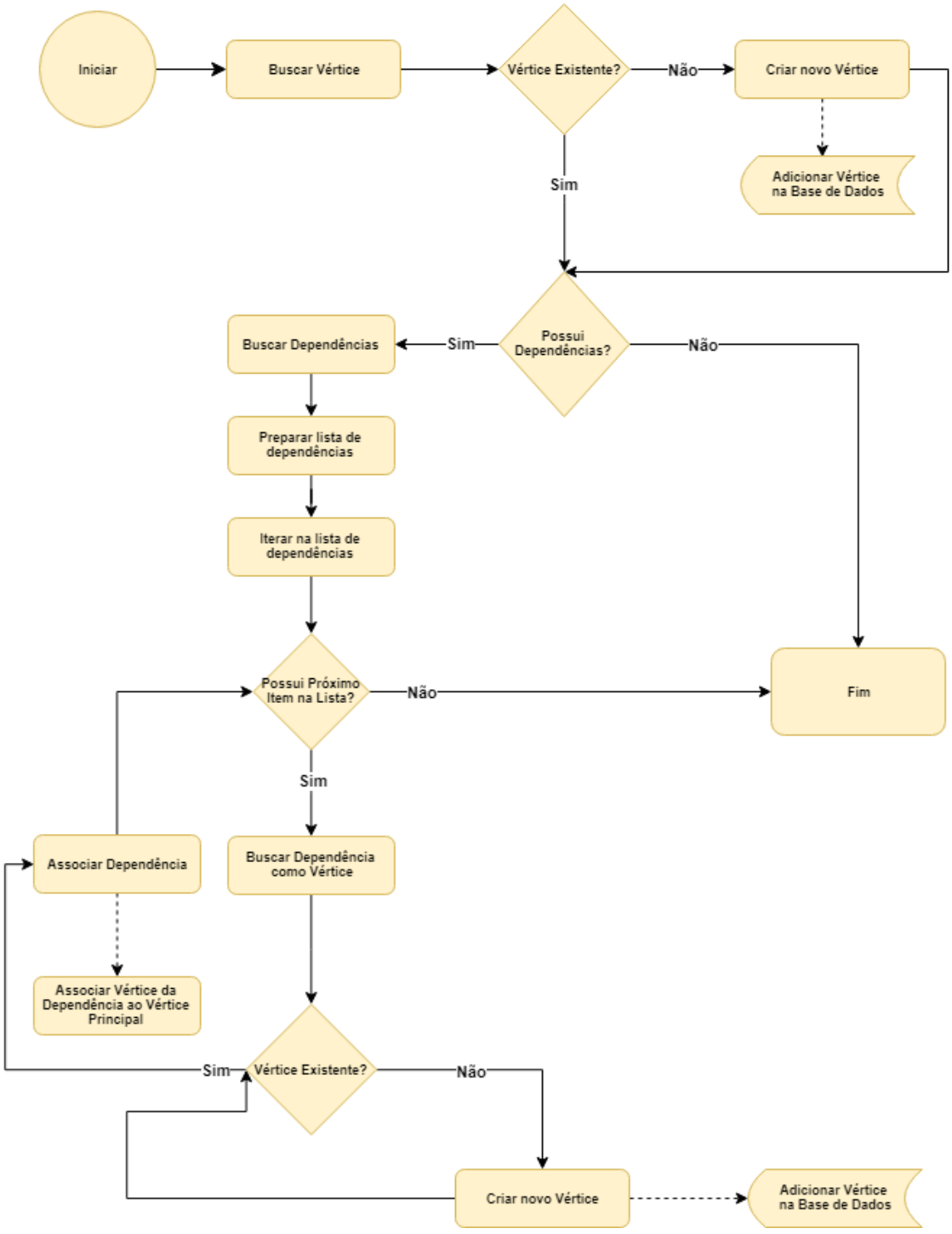


Figura 5.4 – Fluxograma do processo de inclusão de informações no banco de dados

5.4.3. Serviço *Specialist Opinion*

No serviço Specialist Opinion são implementados os requisitos funcionais RQF 7 e 8, referentes à obtenção das preferências do usuário com a escolha da importância de forma comparativa entre as verificações e o cálculo para sugestão dos microsserviços mais crítico. Este serviço possui acesso a dados somente a partir do banco de dados relacional (MySQL), e expõe interfaces REST para serem consumidas pelo serviço de visualização que será detalhado na Seção 5.4.4. A documentação das interfaces expostas seguindo o padrão OpenAPI, podem ser encontradas no próprio serviço em uma url específica (/swagger-api.html) como apresentado no Item III do Apêndice C.

A seguir serão apresentados detalhes acerca da implementação dos requisitos RQF 7 e 8 referentes a obtenção das preferências dos usuários para o ranqueamento das verificações na Seção 5.4.3.1 e o cálculo dos microsserviços mais críticos na Seção 5.4.3.2.

5.4.3.1. Ranqueamento das Verificações

Para o processo de ranqueamento de verificações, que será base para o cálculo a ser realizado com orientação do AHP, é necessário que o usuário escolha um valor entre 1/9 (0.111) e 9, de acordo com a escala proposta por (SAATY, 1980) e representada na Tabela 4.3, na Seção 4.2.2.2. O processo iniciado pelo serviço *Visualizer* realiza o armazenamento dos valores referentes às preferências do usuário tendo sido recuperados na Seção 5.4.2.2, a fim de realizar o cálculo dos microsserviços mais críticos.

5.4.3.2. Cálculo dos Microsserviços mais Críticos

Para o processo referente ao cálculo dos microsserviços mais críticos, o serviço *Specilist Opinion* obtém os fatores de criticidade calculados para cada uma das verificações e os armazena em memória para posterior correlação com os valores referentes aos pesos de cada uma das verificações.

Na sequência, o serviço *Specialist Opinion* recupera os valores cadastrados na Seção 5.4.3.1 acerca das preferências dos usuários em relação a comparação par a par das verificações, e de acordo com este, promove o cálculo baseado no AHP utilizando-se a Tabela 4.4 para validação da consistência aleatória, com isso, é obtida a listagem de pesos relativos a cada uma das verificações.

Com base nos valores obtidos referentes aos fatores de criticidade de cada um dos microsserviços e da listagem de pesos das verificações é feito um novo cálculo correlacionando-os, a fim de obter o valor referente a criticidade de cada um dos microsserviços de forma associada as preferências do usuário.

5.4.4. Serviço *Visualizer*

No serviço *Visualizer*, é implementado o requisito funcional RQF 9, referente à exibição dos microsserviços mais críticos baseado nos dados coletados e processados pelos requisitos RQF 1 a 6 e baseado no cálculo efetuado no RQF 8. A fim de melhor detalhar como este serviço atua na implementação do requisito RQF 9, esta seção é dividida em nove partes.

Na Seção 5.4.4.1, é apresenta a visualização dos dados pós-processados referentes a interdependência entre os microsserviços. Na Seção 5.4.4.2, é detalhada a visualização dos dados pós-processados referentes ao consumo de recursos computacionais. A Seção 5.4.4.3 descreve a visualização dos dados pós-processados referentes a disponibilidade dos microsserviços. Na Seção 5.4.4.4, é delineada a visualização dos dados pós-processados referentes ao tempo de resposta dos microsserviços. Na Seção 5.4.4.5, descreve-se a visualização dos dados pós-processados referentes ao débito técnico dos microsserviços. A Seção 5.4.4.6, expõe a visualização dos dados pós-processados referentes a cobertura de testes unitários dos microsserviços. A Seção 5.4.4.7, discorre sobre a visualização para obtenção das preferências dos usuários para as verificações. A visualização do cálculo baseado nas preferências dos usuários é compreendida na Seção 5.4.4.8. Por fim, a Seção 5.4.4.9, detalha a visualização dos dados coletados referentes aos dados obtidos a partir de cada uma das verificações e a sugestão dos microsserviços mais críticos.

5.4.4.1. Visualização da Interdependência entre os Microsserviços

Nesta seção, são apresentados detalhes acerca da visualização dos dados pós-processados referentes à interdependência entre os microsserviços. O serviço *Visualizer* expor graficamente os diferentes critérios de avaliação e a configuração realizada pelo usuário.

Para a verificação referente à interdependência entre os microsserviços, um grafo é criado, onde cada vértice representa um microsserviços, e as arestas representam

as interfaces de entrada e de saída. Baseado no cálculo de criticidade mencionado na Seção 4.2.1.1 - Fase I, os vértices mais críticos serão identificados e expostos em um grafo criado dinamicamente, esta identificação é feita colorindo o vértice mais crítico com uma cor de alerta (vermelho), como pode ser observado nas Figuras 5.5, 5.6 e 5.7.

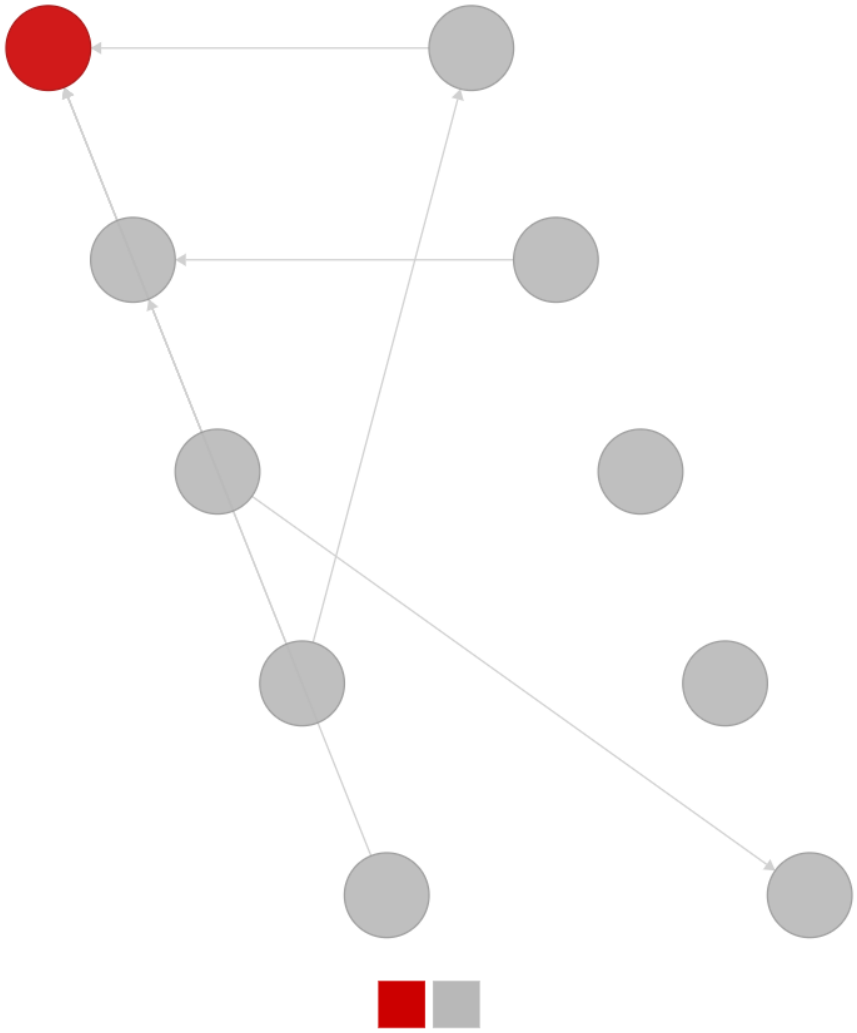


Figura 5.5 – Tela de interdependências sem interação do mouse (sem foco)

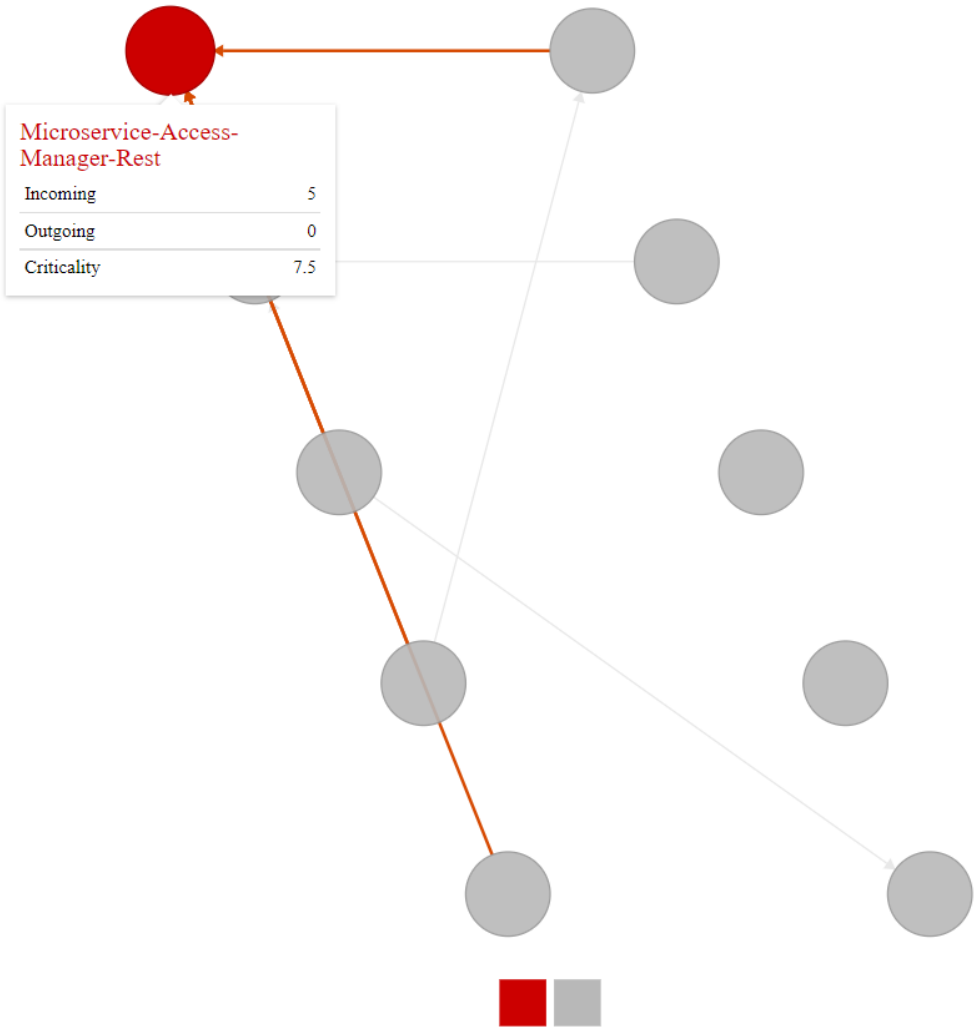


Figura 5.6 – Tela de interdependências com interação de mouseover (com Foco)

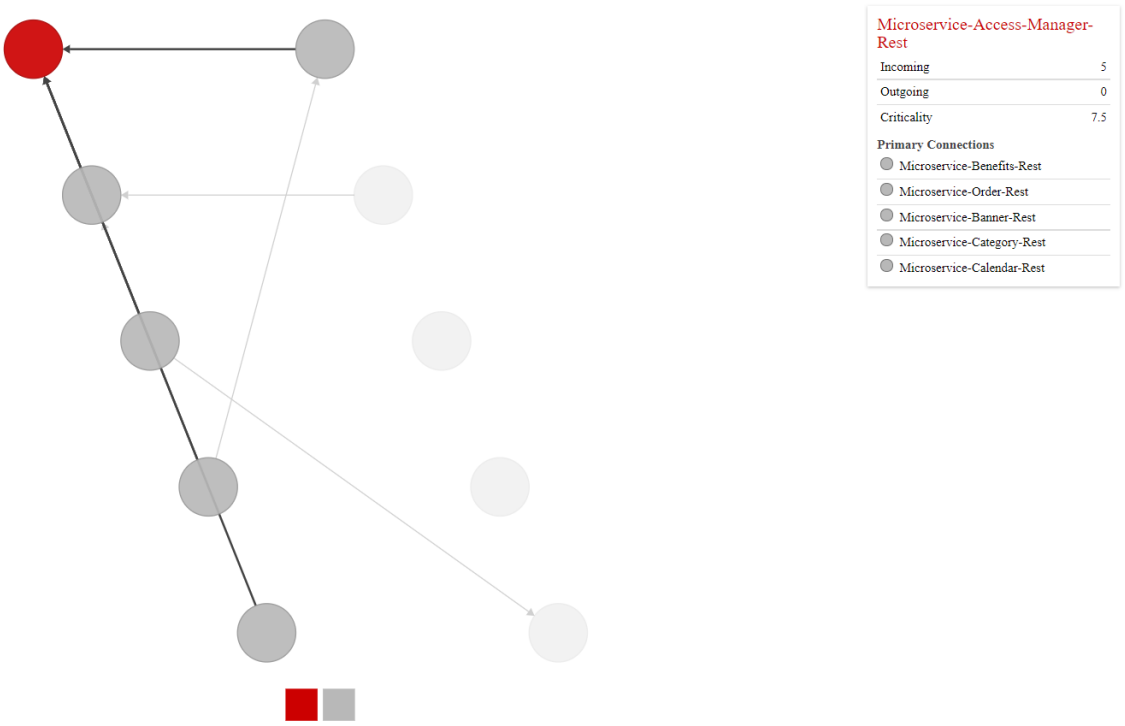


Figura 5.7 – Tela de interdependências com a seleção de microsserviços (clique)

5.4.4.2. Visualização do Consumo de Recursos Computacionais

Nesta seção, são descritos detalhes acerca da visualização dos dados pós-processados referentes ao consumo de recursos computacionais. São apresentadas nesta tela duas barras com valores de 0% a 100%, referente ao percentual de consumo de CPU e memória RAM para cada um dos microsserviços analisados na arquitetura em uso, conforme a Figura 5.8. Para um melhor entendimento do usuário acerca da criticidade referente aos percentuais mostrados, foram utilizadas diferentes cores que podem variar entre verde (0% a 50%), amarelo (maior que 50% e menor que 80%) e vermelho (maior que 80%), de acordo com os percentuais de consumo.

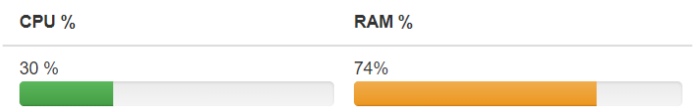
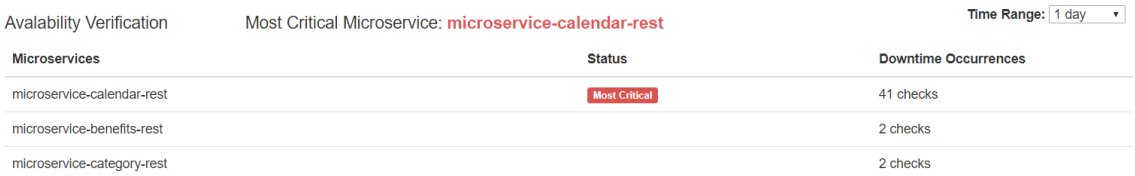


Figura 5.8 – Exibição do consumo de CPU e memória RAM

5.4.4.3. Visualização da Disponibilidade dos Microsserviços

Nesta seção, são apresentados detalhes acerca da visualização dos dados pós-processados referentes à disponibilidade dos microsserviços. Para a listagem da

verificação de disponibilidade, utilizou-se o número de validações nas quais foi observada indisponibilidade do microsserviço, conforme a Figura 5.9. A essa busca também foi adiciona um filtro de tempo a ser considerado, possuindo as opções referentes a 1 dia, 7 dias, 30 dias, 60 dias, 90 dias, 120 dias, 150 dias, 180 dias e 365 dias. A listagem possui ordem decrescente e informa o microsserviço mais crítico relativo a esta verificação.

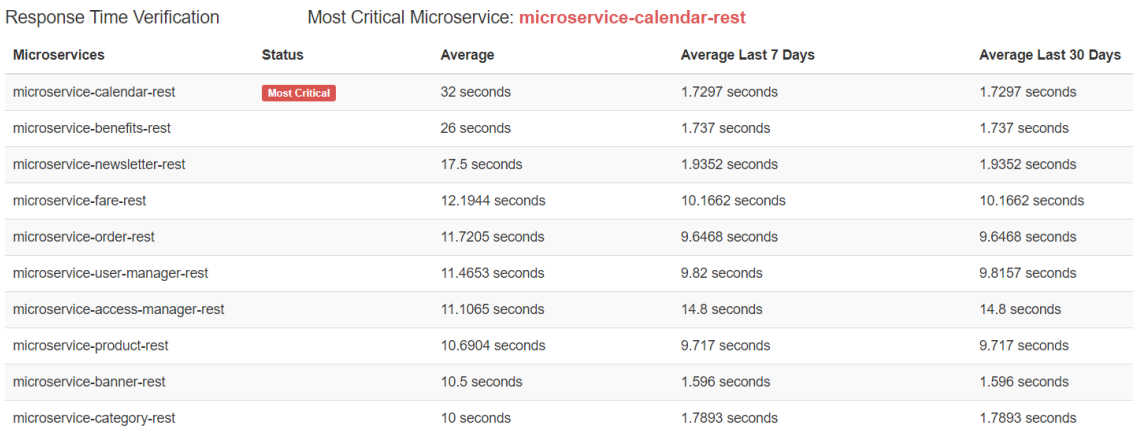

Avalability Verification Most Critical Microservice: microservice-calendar-rest Time Range: 1 day

Microservices	Status	Downtime Occurrences
microservice-calendar-rest	Most Critical	41 checks
microservice-benefits-rest		2 checks
microservice-category-rest		2 checks

Figura 5.9 – Tela para exibição dos dados referentes à disponibilidade

5.4.4.4. Visualização do Tempo de Resposta dos Microsserviços

Nesta seção, são descritos detalhes acerca da visualização dos dados pós-processados referentes ao tempo de resposta dos microsserviços. Para a listagem da verificação de tempo de resposta, utilizou-se o tempo médio relativo a 1 dia, 7 dias e 30 dias, conforme a Figura 5.10. A listagem possui ordem decrescente e informa o microsserviço mais crítico relativo a esta verificação.


Response Time Verification Most Critical Microservice: microservice-calendar-rest

Microservices	Status	Average	Average Last 7 Days	Average Last 30 Days
microservice-calendar-rest	Most Critical	32 seconds	1.7297 seconds	1.7297 seconds
microservice-benefits-rest		26 seconds	1.737 seconds	1.737 seconds
microservice-newsletter-rest		17.5 seconds	1.9352 seconds	1.9352 seconds
microservice-fare-rest		12.1944 seconds	10.1662 seconds	10.1662 seconds
microservice-order-rest		11.7205 seconds	9.6468 seconds	9.6468 seconds
microservice-user-manager-rest		11.4653 seconds	9.82 seconds	9.8157 seconds
microservice-access-manager-rest		11.1065 seconds	14.8 seconds	14.8 seconds
microservice-product-rest		10.6904 seconds	9.717 seconds	9.717 seconds
microservice-banner-rest		10.5 seconds	1.596 seconds	1.596 seconds
microservice-category-rest		10 seconds	1.7893 seconds	1.7893 seconds

Figura 5.10 – Tela para exibição dos dados referentes ao tempo de resposta

5.4.4.5. Visualização do Débito Técnico dos Microsserviços

Nesta seção, são apresentados detalhes acerca da visualização dos dados pós-processados referentes ao débito técnico dos microsserviços. Para a listagem da verificação de tempo de débito técnico, utilizou-se o tempo relativo à soma das não conformidades de cada um dos microsserviços, conforme a Figura 5.11. A listagem

possui ordem decrescente e informa o microsserviço mais crítico relativo a esta verificação.

Tech Debit Verification — Most Critical Microservice: **microservice-product-rest**

Microservices	Status	Technical Debit
microservice-product-rest	Most Critical	540 minutes
microservice-order-rest		480 minutes
microservice-delivery-rest		420 minutes
microservice-fare-rest		300 minutes
microservice-billing-rest		240 minutes
microservice-basket-rest		180 minutes
microservice-access-manager-rest		180 minutes
microservice-user-manager-rest		180 minutes
microservice-inventory-rest		180 minutes

Figura 5.11 – Tela para exibição dos dados referentes ao débito técnico

5.4.4.6. Visualização da Cobertura de Testes Unitários dos Microsserviços

Nesta seção, são descritos detalhes acerca da visualização dos dados pós-processados referentes à cobertura de testes unitários dos microsserviços. Para a listagem da verificação de cobertura de testes unitários, utilizou-se o percentual de cobertura para cada um dos microsserviços analisados, conforme a Figura 5.12. A listagem possui ordem decrescente e informa o microsserviço mais crítico relativo a esta verificação.

Unit Test Coverage Vision — Most Critical Microservice: **microservice-product-rest**

Microservices	Status	Coverage %
microservice-product-rest	Most Critical	0 %
microservice-order-rest		10 %
microservice-category-rest		12 %
microservice-access-manager-rest		12 %
microservice-calendar-rest		25 %
microservice-banner-rest		40 %
microservice-newsletter-rest		42 %
microservice-fare-rest		51 %
microservice-user-manager-rest		67 %
microservice-benefits-rest		81 %

Figura 5.12 – Tela para exibição dos dados referentes a cobertura de testes unitários

5.4.4.7. Visualização para Obtenção das Preferências dos Usuários

Nesta seção, são apresentados detalhes acerca da visualização referente à obtenção das preferências do usuário. Para a obtenção das preferências do usuário, utilizou-se os valores referentes à escala definida por (SAATY, 1980), exposta na Tabela 4.3 do Capítulo 4. A listagem dos valores a serem selecionados possui ordem crescente, sendo ordenados da esquerda para a direita conforme a Figura 5.13. Ao selecionar um valor para o par em comparação, este é salvo automaticamente, não sendo necessário o clique em qualquer botão.

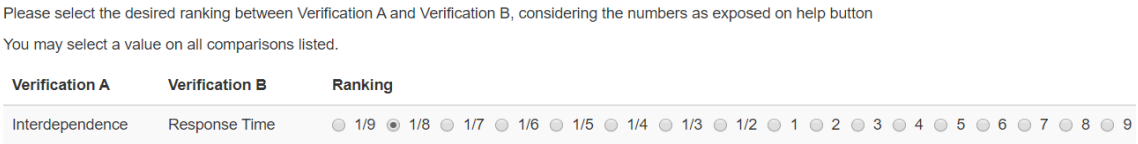


Figura 5.13 – Seleção dos pares de verificações para cálculo do AHP

A fim de melhor contextualizar o usuário acerca do significado dos valores, um botão de ajuda foi adicionado, e ao clique deste botão, uma tela sobreposta (modal) é exibida, escurecendo o entorno e promovendo a atenção ao centro da tela, na qual serão exibidos os valores da escala proposta por (SAATY, 1980) conforme a Figura 5.14.

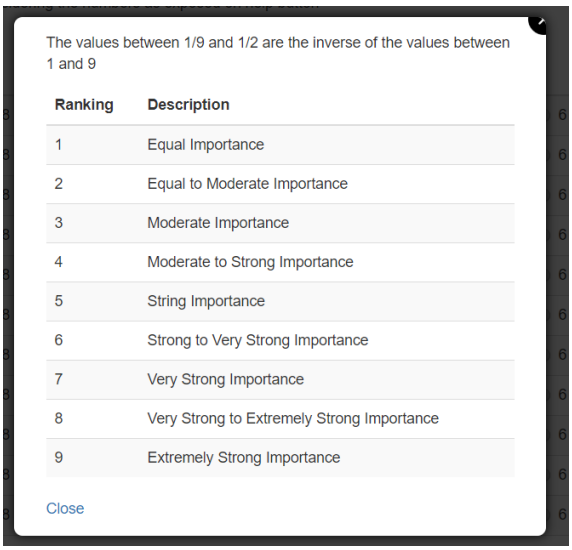


Figura 5.14 – Exibição da listagem dos valores. Adaptada de (SAATY, 1980)

Para um melhor entendimento acerca da listagem de pares de verificação a serem comparados, a Figura 5.15 expõe a tela a ser utilizada pelo usuário para escolha de suas preferências.

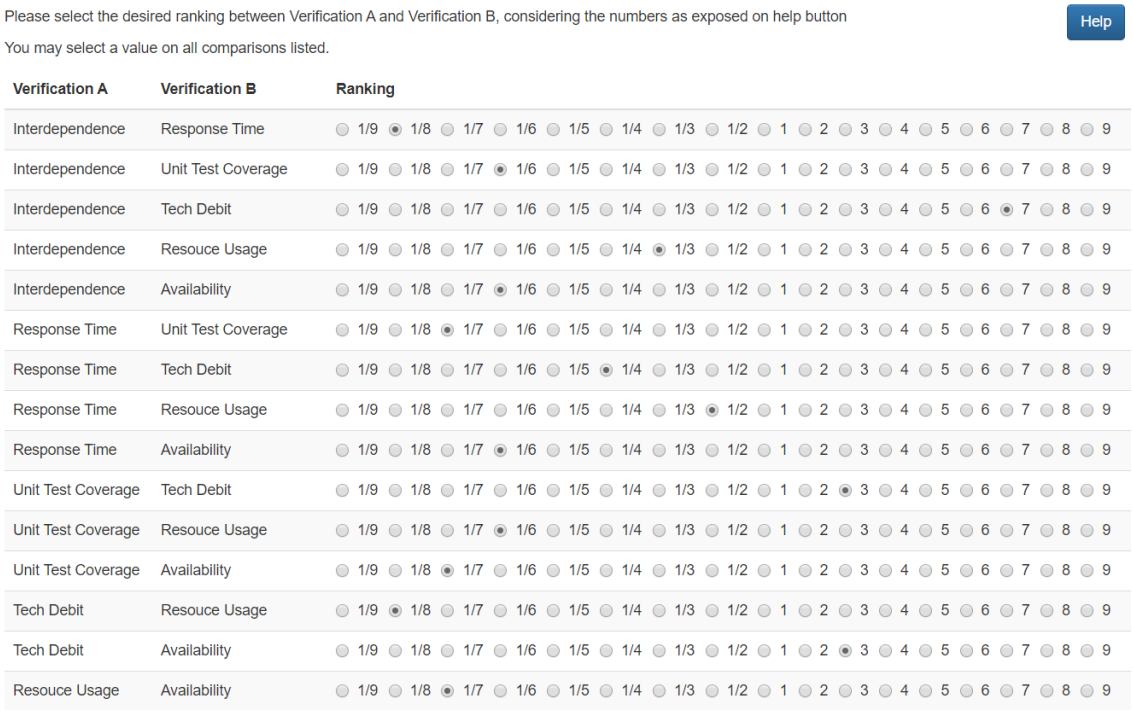


Figura 5.15 – Tela para seleção dos pares de verificações para cálculo do AHP

5.4.4.8. Visualização da Sugestão dos Microsserviços mais Críticos

Nesta seção, são descritos detalhes acerca da visualização referente aos dados obtidos a partir de cada uma das verificações e a sugestão dos microsserviços mais críticos. Para a listagem dos microsserviços mais críticos, utilizou-se o cálculo mencionado na Seção 4.2.3. Para cada microsserviço, é exibido o fator de criticidade associado a este para uma verificação específica, bem como o peso calculado para aquela verificação, conforme pode ser observado na Figura 5.16.

À listagem de microsserviços mais críticos também foi adicionado um filtro de tempo a ser considerado, possuindo as opções referentes a 1 dia, 7 dias, 30 dias, 60 dias, 90 dias, 120 dias, 150 dias, 180 dias e 365 dias. A listagem dos microsserviços mais críticos possui ordem decrescente e informa o microsserviço mais crítico relativo àquela verificação.

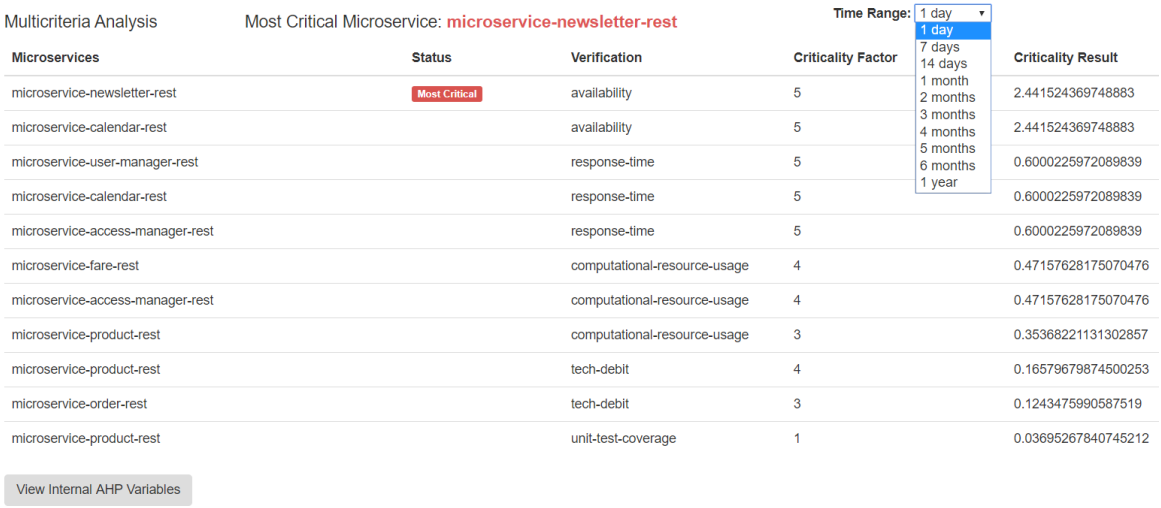


Figura 5.16 – Tela para exibição da listagem dos microsserviços mais críticos

5.4.4.9. Visualização do Cálculo Baseado nas Preferências do Usuário

Nesta seção, são apresentados detalhes acerca da visualização referente ao cálculo baseado nas preferências do usuário. Para um melhor entendimento acerca dos dados resultantes do cálculo realizado baseado no método AHP, foi adicionado um botão de visualização das variáveis utilizadas pelo método e ao clicar neste botão, são exibidas as informações acerca dos valores calculados, conforme as Figuras 5.17, 5.18 e 5.19

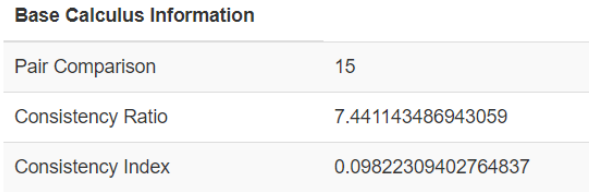


Figura 5.17 – Exibição dos valores calculados referentes ao AHP

Comparisons between Verifications	
Response Time in comparison with Unit Test Coverage	3
Unit Test Coverage in comparison with Comp. Resource Usage	0.2
Response Time in comparison with Comp. Resource Usage	2
Technical Debt in comparison with Comp. Resource Usage	0.2
Interdependence in comparison with Comp. Resource Usage	2
Response Time in comparison with Availability	0.16666666666666666
Interdependence in comparison with Availability	0.25
Comp. Resource Usage in comparison with Availability	0.2
Interdependence in comparison with Unit Test Coverage	4
Unit Test Coverage in comparison with Technical Debt	0.5
Technical Debt in comparison with Availability	0.125
Interdependence in comparison with Response Time	3
Interdependence in comparison with Technical Debt	5
Unit Test Coverage in comparison with Availability	0.14285714285714285
Response Time in comparison with Technical Debt	4

Figura 5.18 – Exibição dos pares de verificações após cálculo do AHP

Verifications Ranking	
Availability	0.48830487394977656
Unit Test Coverage	0.03695267840745212
Response Time	0.12000451944179678
Resouce Usage	0.11789407043767619
Interdependence	0.19539465807704773
Tech Debit	0.04144919968625063

Figura 5.19 – Exibição dos pesos das verificações após cálculo do AHP

5.5. Exemplo de Uso

Para facilitar a compreensão dos detalhes da abordagem *M2CSEA* e seu ferramental de suporte, esta seção apresenta um cenário de exemplo que acompanha uma organização fictícia com foco em *e-commerce* e que possui uma arquitetura estabelecida para seu portal de compras. A esta empresa foi dado o nome de EComm. Para este exemplo, são analisados os microsserviços da arquitetura da empresa EComm para que haja a orientação acerca dos microsserviços mais críticos, e por consequência onde a EComm necessitará de maior investimento em manutenção ou evolução.

A arquitetura dos microsserviços da EComm foi criada conforme o diagrama contido na Figura 5.20. Todos os microsserviços contidos nesta arquitetura se comunicam com as ferramentas Consul, MySQL e SonarQube por meio de uma rede virtual criada a partir do gerenciador de contêineres (Docker).

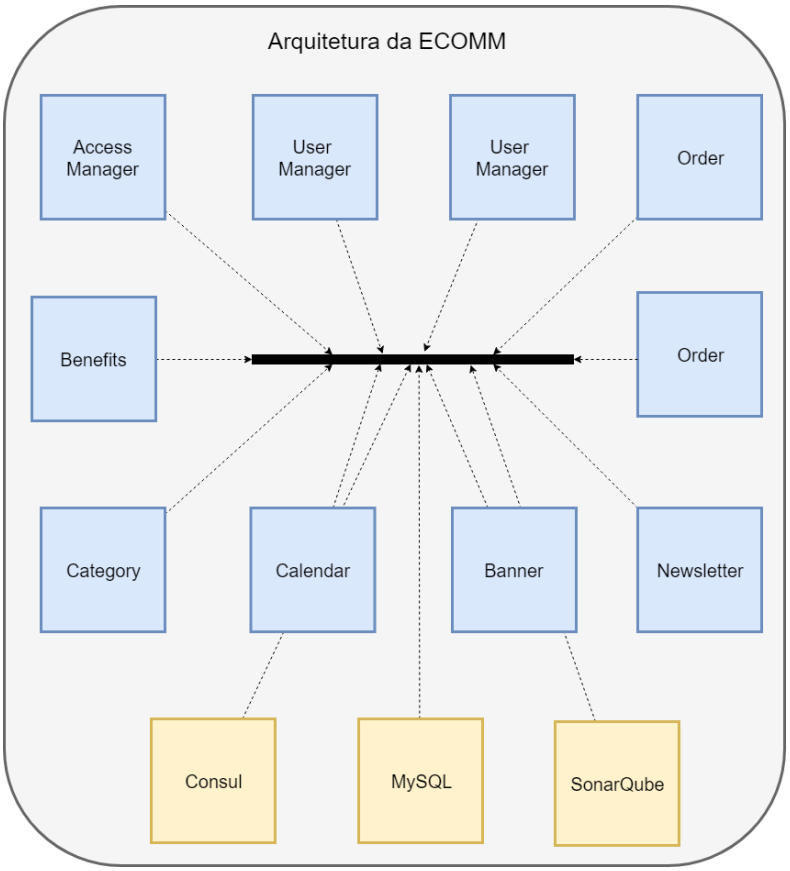


Figura 5.20 – Diagrama da arquitetura da empresa Ecomm

Os microsserviços da arquitetura em uso da empresa Ecomm estão utilizando a abordagem M2CSEA, e com isso fornecem as informações de tempo de resposta para cada uma das suas interfaces REST por meio do banco MySQL, bem como o relatório de cobertura de testes unitários e a análise de débito técnico para a ferramenta SonarQube. Quanto a disponibilidade e a interdependência, cada serviço, ao realizar seu processo de inicialização, também registra-se no Consul e informa a este sistema todos os microsserviços dos quais este depende, sendo possível a obtenção dos dados referente à disponibilidade e dependências a partir desta ferramenta.

Considerando que as ferramentas Consul, MySQL e SonarQube estão atualizadas, os serviços da abordagem M2CSEA podem extrair a partir destes as informações necessárias para o cálculo do fator de criticidade referente a cada verificação para cada um dos microsserviços.

Quanto às preferências do usuário, a tela de obtenção destes dados fica disponível para o usuário. Desta forma, a comparação par a par é realizada, e ao fim são obtidos os pesos referentes a cada uma das verificações, sendo possível ordená-las quanto a sua prioridade do ponto de vista do usuário.

Após a obtenção dos fatores de criticidade para cada um dos microsserviços e cada uma das verificações, é feita a obtenção dos pesos das verificações, e com isso feita a correlação para se chegar à lista de microsserviços mais críticos. A partir da conclusão do cálculo desta lista, é exibido ao usuário a lista de microsserviços mais críticos.

5.6. Considerações Finais

Este capítulo apresentou o ferramental de suporte implementado para apoiar a sistemática da abordagem *M2CSEA* (Capítulo 4), sendo este composto pelas ferramentas *Finder*, *Architecture Probe*, *Specialist Opinion*, *Visualizer* e ferramentas associadas. Este ferramental oferece uma infraestrutura para sugestão dos microsserviços mais críticos em arquiteturas estabelecidas com o intuito de sugerir aos tomadores de decisão os microsserviços que mais necessitam de atenção para manutenção ou evolução.

Algumas limitações podem ser observadas no ferramental de suporte da abordagem *M2CSEA*. No ferramental, foram analisadas somente comunicações via HTTP, não havendo uma implementação com outros protocolos de comunicação comumente utilizados em arquiteturas orientadas a microsserviços, como AMQP por exemplo.

Quanto ao número de instâncias dos microsserviços, o ferramental somente considerou uma instância de cada microsserviços enviando informações acerca das verificações, entretanto este cenário é incomum em um ambiente produtivo de sistemas de grande porte.

Acerca dos dados de consumo de recursos computacionais, foram considerados somente a memória RAM e CPU, não tendo sido considerados outros insumos de menor importância, mas que em um contexto de um ambiente de produção podem ser úteis, como I/O de disco, I/O de rede e memória *swap*.

Por fim, melhorias podem ser realizadas para inserir recursos com o objetivo de apoiar a obtenção de opiniões de usuários de múltiplas áreas da mesma empresa, para servir de base para auxiliar a tomada de decisão entre equipes.

CAPÍTULO 6 – AVALIAÇÃO DA PROPOSTA

6.1. Introdução

Conforme observado no capítulo anterior, um ferramental de suporte foi implementado para ofertar meios para utilização prática da abordagem *M2CSEA*, entretanto, de acordo com a literatura de engenharia de software experimental, a validade de qualquer corpo de conhecimento deve ser avaliada para que esse conhecimento possa ser considerado científico (JURISTO e MORENO, 2013).

Neste capítulo, são apresentados os estudos experimentais conduzidos para validação da proposta e da aplicabilidade da abordagem M2CSEA. De acordo com WOHLIN et al. (2012) e ROBSON (2002), existem três tipos diferentes de estratégias de investigação que podem ser consideradas para um estudo experimental: pesquisa (*survey*), experimento controlado ou estudo de caso.

WOHLIN et al. (2012) e FINK (2003) mencionam que uma pesquisa é um sistema para coletar informações de ou sobre pessoas para descrever, comparar ou explicar seus conhecimentos, atitudes e comportamento. Uma pesquisa é frequentemente uma investigação realizada em retrospecto (PFLEEGER, 1994). O principal meio de coleta de dados qualitativos ou quantitativos são entrevistas ou questionários. Isso é feito com uma amostra representativa da população a ser estudada. Os resultados da pesquisa são então analisados para derivar conclusões descritivas e explicativas. Eles são generalizados para a população da qual a amostra foi coletada.

O experimento controlado em engenharia de software é uma investigação empírica que manipula um fator ou variável da configuração estudada. Com base na randomização, diferentes tratamentos são aplicados, mantendo outras variáveis constantes e medindo os efeitos nas variáveis de resultado. Em experimentos orientados a humanos, esse método pode aplicar diferentes tratamentos a objetos, enquanto em experimentos orientados à tecnologia diferentes tratamentos técnicos são aplicados a diferentes objetos (WOHLIN et al., 2012).

Em experimentos controlados, as experiências são feitas principalmente em um ambiente de laboratório, que fornece controle de alto nível. Ao experimentar, os sujeitos são atribuídos a diferentes tratamentos aleatoriamente. O objetivo é manipular uma ou mais variáveis e controlar todas as outras variáveis em níveis fixos. O efeito da manipulação é medido e, com base nisso, uma análise estatística pode ser realizada. Nos casos em que é

impossível atribuir tratamentos aleatoriamente aos indivíduos, pode-se usar quase-experimentos (WOHLIN et al., 2012).

De acordo com WOHLIN et al. (2012), o estudo de caso em engenharia de software é uma investigação empírica que se baseia em várias fontes de evidência para investigar uma instância, ou um pequeno número de instâncias, de um fenômeno contemporâneo de engenharia de software em seu contexto da vida real, especialmente quando a fronteira entre fenômeno e contexto não podem ser claramente especificados (RUNESON et al., 2012) (PETERSEN e WOHLIN, 2009).

Os estudos de caso são usados para pesquisar projetos, atividades ou atribuições. Os dados são coletados para uma finalidade específica ao longo do estudo. Com base na coleta de dados, análises estatísticas podem ser realizadas. O estudo de caso, normalmente, visa rastrear um atributo específico ou estabelecer relacionamentos entre diferentes atributos.

O nível de controle é mais baixo em um estudo de caso do que em um experimento. Um estudo de caso é um estudo observacional enquanto o experimento é um estudo controlado (ZELKOWITZ e WALLACE, 1998). A análise estatística multivariada é, frequentemente, aplicada neste tipo de estudo.

Além desta seção introdutória, este capítulo está organizado de forma que o objetivo referente à avaliação da proposta é apresentado na Seção 6.2. Na Seção 6.3, são apresentados detalhes acerca da caracterização da população, incluindo os critérios de elegibilidade bem como a seleção e arranjo dos participantes. As questões de pesquisa são descritas na Seção 6.4. A Seção 6.5 possui o detalhamento das hipóteses. O planejamento do estudo é apresentado na Seção 6.6. Os detalhes acerca da execução do estudo são apresentados na Seção 6.7. Os resultados obtidos são detalhados na Seção 6.8, contemplando a interpretação dos resultados, as ameaças a validade, a validação das hipóteses e a validade do estudo. Por fim, a conclusão é apresentada na Seção 6.9.

6.2. Objetivo

O objetivo deste estudo foi determinar se a abordagem proposta relacionada a múltiplos critérios é eficiente e assertiva ao sugerir os microsserviços mais críticos em arquiteturas estabelecidas.

6.3. Caracterização da População

A população do estudo são profissionais de engenharia de software com pelo menos quatro anos de experiência em desenvolvimento de software. Um subgrupo desta população foi definido com base na experiência em microsserviços, sendo necessário que o participante possuísse ao menos dois anos de experiência no desenvolvimento de software com microsserviços.

6.3.1. Critérios de Elegibilidade

Critérios de inclusão:

- 4 anos de experiência em desenvolvimento de software.
- Graduação acadêmica completa.

Critério de exclusão

- O participante não possui domínio dos idiomas português ou inglês.

6.3.2. Seleção e Arranjo dos Participantes

Os participantes foram selecionados de forma não aleatória, sendo escolhidos por conveniência. O período de recrutamento dos participantes foi de 21 de outubro de 2019 e a 1º de dezembro de 2019. Foram contatados 80 participantes por e-mail e telefone, sendo sido a taxa de participação igual a 15% (12 participantes).

A execução e o recrutamento para o estudo ocorreram em paralelo, havendo um acompanhamento individual durante o experimento. O consentimento da privacidade e a não divulgação dos dados foram explicados e assinados eletronicamente.

Os participantes foram divididos aleatoriamente em quatro grupos:

- **Grupo A (Tratamento):** participantes com experiência em arquitetura de microsserviços, utilizando a implementação da proposta de estudo baseada na abordagem M2CSEA.
- **Grupo B (Controle):** participantes com experiência em arquitetura de microsserviços, que não utilizam a implementação da proposta de estudo baseada na abordagem M2CSEA.

- **Grupo C (Tratamento):** participantes sem experiência em arquitetura de microsserviços, utilizando a implementação da proposta de estudo baseada na abordagem M2CSEA.
- **Grupo D (Controle):** participantes sem experiência em arquitetura de microsserviços, que não utilizam a implementação da proposta de estudo com base na abordagem M2CSEA.

A divisão dos participantes em grupos com experiência em microsserviços e sem experiência em microsserviços utilizando ou não a ferramenta visa demonstrar se o conhecimento acerca desta arquitetura seria um impedimento para o uso do ferramental.

6.4. Questões de Pesquisa

RQ1 - A abordagem proposta é assertiva ao sugerir os microsserviços mais críticos em arquiteturas estabelecidas?

RQ2 - A abordagem proposta é eficiente em sugerir os microsserviços mais críticos em arquiteturas estabelecidas?

RQ3 - A abordagem proposta pode ajudar os profissionais de engenharia de software no processo de tomada de decisão?

6.5. Hipóteses

Hipótese Nula: A abordagem proposta não é eficiente nem assertiva para sugerir os microsserviços mais críticos em arquiteturas estabelecidas.

Hipótese Alternativa 1: A abordagem proposta é eficiente e assertiva para sugerir os microsserviços mais críticos em arquiteturas estabelecidas.

Hipótese Alternativa 2: A abordagem proposta é eficiente, mas não assertiva, para sugerir os microsserviços mais críticos em arquiteturas estabelecidas.

Hipótese Alternativa 3: A abordagem proposta não é eficiente, mas sim assertiva, para sugerir os microsserviços mais críticos em arquiteturas estabelecidas.

6.6. Planejamento do Estudo

Conforme dito anteriormente, o estudo de caso foi escolhido como estratégia de investigação motivada pelo objetivo de verificar um fenômeno real em um cenário semelhante à vida real, sendo difícil randomizar as diversas variáveis associadas a este estudo e extrair resultados conclusivos.

Para execução do estudo de caso, considerou-se uma seleção pré-estabelecida de preferências do usuário, pois esta hipótese de trabalho está associada à eficiência e assertividade na orientação de um processo de tomada de decisão. A seleção de preferências do usuário fará parte de trabalhos futuros, associados à forma como diversos tomadores de decisão podem colaborar entre si para uma decisão com múltiplos atores, considerando diferentes funções e requisitos.

Os questionários aplicados a cada um dos grupos de controle e tratamento, bem como as informações auxiliares ao grupo de controle relacionadas às ferramentas disponíveis para uso, assim como o formulário de não divulgação e às especificações técnicas estão disponíveis nos Apêndices F (Formulário de Consentimento do Estudo de Caso), G (Questionário do Estudo de Caso) e H (Informações Complementares para o Grupo Controle do Estudo de Caso).

6.7. Execução do Estudo

Este estudo foi realizado de forma geograficamente distribuída, tendo sido realizado em mais de um local. A maioria dos participantes participou de forma on-line por meio de videoconferência, tendo realizado o estudo por meio de orientação por voz e compartilhamento de tela. Os participantes que realizaram o estudo de forma presencial, estavam localizados na cidade de Blumenau, Santa Catarina / Brasil. A diferença entre participantes locais e remotos não foi relatada por nenhum destes como causa de aumento ou redução de seu desempenho.

Para participantes remotos, a todo momento foi utilizada uma conexão à Internet de alta velocidade. Para todos os participantes, não foi oferecida a oportunidade de executar o estudo com interrupções. Portanto, uma vez iniciada a sessão de estudo, esta deveria ser realizada até o final ou os dados deveriam ser descartados e o participante não seria elegível a uma segunda tentativa.

As atividades solicitadas aos participantes foram cronometradas durante todo o estudo. Essas medidas foram a base da análise estatística para a coleta de dados associada às medidas de eficiência.

Durante o estudo, os participantes responderam a um questionário contendo um formulário para consentimento do estudo de caso (Apêndice F) e um questionário com dados demográficos e questões acerca das tarefas do estudo (Apêndice G). Os participantes foram informados que não era permitido fornecer nenhuma informação que fosse capaz de identificá-los, considerando que todas as respostas não conteriam um viés associado a restrições pessoais, tendo em vista que o desempenho deste não estaria associado a um participante identificável.

6.8. Resultados

Esta seção expõe os resultados obtidos após a execução do estudo de caso e os resultados extraídos dos dados compilados. Com relação à análise dos dados, as discrepâncias (*outliers*) foram avaliadas com uma avaliação 3-sigma de forma individual, não tendo ocorrido nenhuma remoção automática ou manual. Toda a análise estatística realizada neste estudo foi realizada com base no software JASP[44].

A seguir são detalhadas as medidas obtidas de acordo com as respostas ao questionário após a execução das tarefas solicitadas. Para as medidas 1 e 2, foram realizadas atividades em dois diferentes estados, o inicial e o modificado, sendo estes dois estados diferentes de avaliações dos microsserviços mais críticos possuindo diferentes resultados, com o intuito de reduzir o viés da localização correta por eventual coincidência.

A. Medida 1 - Assertividade no estado inicial e no estado modificado (RQ1)

Esta medida pretende avaliar a assertividade do participante ao localizar o microsserviço mais crítico em diferentes perspectivas. A assertividade neste contexto se dá com a comparação de um gabarito estipulado antes da execução da sessão do estudo e que será comparado a resposta do participante para terminar se esta foi ou não correta.

Para analisar os resultados da medida 1 relacionada à assertividade no estado inicial e no estado modificado, foi utilizado o teste do qui-quadrado (distribuição χ^2) para comparar duas distribuições de frequências, considerando os dados como frequências absolutas (contagem de respostas assertivas).

[44] JASP: https://jasp-stats.org

Em estatística, existem dois tipos de variáveis: variáveis numéricas (contáveis) e variáveis não numéricas (categóricas). O método qui-quadrado utiliza um número único que verifica a diferença existente entre as contagens observadas e as que se esperaria se não houvesse nenhum relacionamento na população. A estatística com base no método qui-quadrado é uma maneira de mostrar uma relação entre duas variáveis categóricas, como mostra a Fórmula (6.1).

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i} \quad (6.1)$$

As porcentagens de assertividade foram calculadas para cada grupo (A, B, C e D) considerando as respostas certas para cada pergunta associada à localização do microsserviços mais crítico. Nesta medida, o test-t de amostras independentes possui t = 0,000, df = 10.000 e p = 1.000, não sendo possível afirmar que os grupos de tratamento (A e C) foram mais assertivos que os grupos controle (B e D), considerando que apenas uma resposta em quinze perguntas de dois participantes em um total de 6 participantes nos grupos de controle foram respondidas erroneamente. Todas as perguntas dos grupos de tratamento foram respondidas adequadamente, sem nenhum erro.

Os resultados referentes a validação de assertividade podem ser observados na Tabela C presente no Apêndice J (Resultados Individuais Agrupados no Estudo de Caso).

B. Medida 2 - Eficiência no estado inicial e no estado modificado

Esta medida pretende avaliar a rapidez com que o participante é capaz de localizar assertivamente o microsserviço mais crítico em diferentes perspectivas, comparando todos os grupos. Os grupos A e C apenas usaram a ferramenta proposta criada com base no modelo M2CSEA, os grupos B e D possuíam algumas ferramentas específicas, que detalhadas no Apêndice H (Informações Complementares para Grupo Controle), para orientar a atividade de localização.

Para analisar os resultados dessa medida, relacionados à eficiência no estado inicial e no estado modificado, é utilizado o método de Análise de Variância Unidirecional (ANOVA). Segundo SELTMAN (2009), o método one-way ANOVA examina a qualidade da média da população para obter um resultado quantitativo e uma única variável categórica. O método ANOVA pode ser usado no caso de um resultado

quantitativo com uma variável explicativa categórica que possui dois ou mais níveis de tratamento.

O termo unidirecional ou *one-way*, também chamado de fator único, indica que existe uma única variável explicativa, também conhecida como tratamento, com dois ou mais níveis, e apenas um nível de tratamento. O método ANOVA é semelhante ao teste-t, no entanto, o método ANOVA pode ser usado para comparar as médias de três ou mais grupos; por outro lado, os testes-t podem comparar apenas dois grupos por vez (SELTMAN, 2009).

Todos os testes one-way ANOVA neste trabalho foram conduzidos com um nível de significância de 0,05. Um nível de significância (como α ou alfa) de 0,05 indica um risco de 5% para falso-positivo, ou seja, a possibilidade de concluir que haja uma diferença quando não há diferença real. Se o p-value for menor ou igual ao nível de significância, a hipótese nula é rejeitada e é possível concluir que nem todas as médias da população são iguais.

Em cada verificação exposta nesta medida, duas perguntas relacionadas a cada verificação foram usadas, uma relativa ao estado inicial e outra relativa ao estado modificado, tendo sido solicitado ao participante que localizasse os microsserviços mais críticos usando cada verificação. Afinal, as duas respostas de cada verificação foram agrupadas e calculada a média para criar um único valor para servir de base a cada verificação desta medida.

Os resultados referentes a cada uma das questões relativas à eficiência podem ser observados nas Tabelas A, B e C presentes no Apêndice I (Resultados Individuais Obtidos no Estudo de Caso), e o agrupamento dos resultados conforme expostos no parágrafo anterior, pode ser observado nas Tabelas A e B presentes no Apêndice J (Resultados Individuais Agrupados no Estudo de Caso).

Eficiência na verificação de interdependência

Analisando as respostas relacionadas à medida associada à verificação da interdependência, não houve diferença estatisticamente significante entre os grupos, conforme demonstrado pela one-way ANOVA ($F = 2,733$, $p = 0,114$) e mostrada na Tabela 6.1, entretanto, optou-se por prosseguir com o teste H de Kruskal-Wallis para prova.

O teste teste H de Kruskal-Wallis mostrou que há uma diferença estatisticamente significante entre os grupos, $\chi2(2) = 7.308$, $p = 0.063$, o que significa um ranqueamento

médio de 13.420 para o Grupo A, 25.855 para o Grupo B, 5.382 para o Grupo C e por fim 20.182 para o Grupo D.

Desta forma, como análise complementar, foi conduzido um teste post hoc de Tukey, como mostrado na Tabela 6.2, com um intervalo de confiança de 95%, de que a eficiência da verificação da interdependência foi estatisticamente significante relacionada ao grupo experimental A e grupo controle B (p = 0,408) e grupo experimental C e grupo controle D (p = 0,278)

Tabela 6.1 – Teste ANOVA para interdependência

Casos	Soma dos Quadrados	df	Média dos Quadrados	F	p
Grupo	701.511	3.000	233.837	2.733	0.114
Residual	684.362	8.000	85.545		

Nota. Tipo III - Soma dos Quadrados

Tabela 6.2 – Teste post hoc de Tukey para interdependência

			95% CI para Diferença Média					
		Diferença Média	Mais Baixo	Mais Alto	SE	t	Cohen's d	p_{tukey}
A	B	-12.435	-36.619	11.749	7.552	-1.647	-1.062	0.408
	C	8.038	-16.145	32.222	7.552	1.064	1.127	0.719
	D	-6.762	-30.945	17.422	7.552	-0.895	-0.735	0.808
B	C	20.473	-3.710	44.657	7.552	2.711	2.202	0.100
	D	5.673	-18.510	29.857	7.552	0.751	0.517	0.874
C	D	-14.800	-38.984	9.384	7.552	-1.960	-2.540	0.278

Nota. Ajuste do intervalo de confiança: método tukey para comparar uma família de 4 estimativas

Eficiência na verificação do tempo de resposta

Analisando as respostas relacionadas à medida associada à verificação do tempo de resposta, houve diferença estatisticamente significante entre os grupos, conforme demonstrado pelo método one-way ANOVA (F = 14,632, p = 0,001) e mostrada na Tabela 6.3.

Um teste post hoc de Tukey mostrado na Tabela 6.4, com um intervalo de confiança de 95%, indica que a eficiência da verificação do tempo de resposta está estatisticamente significante relacionada ao grupo experimental A e ao grupo controle B (p = 0,040) e ao grupo experimental C e grupo de controle D (p = 0,002)

Tabela 6.3 – Teste ANOVA para tempo de resposta

Casos	Soma dos Quadrados	df	Média dos Quadrados	F	p
Grupo	11114.809	3.000	3704.936	14.632	0.001
Residual	2025.728	8.000	253.216		

Nota. Tipo III - Soma dos Quadrados

Tabela 6.4 – Teste post hoc de Tukey para tempo de resposta

			95% CI para Diferença Média					
		Diferença Média	Mais Baixo	Mais Alto	SE	t	Cohen's d	p_{tukey}
A	B	-43.768	-85.376	-2.161	12.993	-3.369	-2.065	0.040
	C	6.147	-35.461	47.754	12.993	0.473	0.936	0.963
	D	-66.987	-108.594	-25.379	12.993	-5.156	-6.721	0.004
B	C	49.915	8.308	91.522	12.993	3.842	2.474	0.021
	D	-23.218	-64.826	18.389	12.993	-1.787	-1.079	0.345
C	D	-73.133	-114.741	-31.526	12.993	-5.629	-9.671	0.002

Nota. Ajuste do intervalo de confiança: método tukey para comparar uma família de 4 estimativas

Eficiência na verificação de débito técnico

Analisando as respostas relacionadas à medida associada à verificação do débito técnico, houve diferença estatisticamente significante entre os grupos, conforme demonstrado pela One-way ANOVA (F = 21,333, p = < 0,001) e mostrada na Tabela 6.5.

Um teste post hoc de Tukey mostrado na Tabela 6.6, com um intervalo de confiança de 95%, indica que a eficiência da verificação do débito técnico é estatisticamente significante relacionada ao grupo experimental A e ao grupo controle B (p = 0,002) e ao grupo experimental C e grupo controle D (p = 0,003).

Tabela 6.5 – Teste ANOVA para débito técnico

Casos	Soma dos Quadrados	df	Média dos Quadrados	F	p
Grupo	2936.707	3.000	978.902	21.333	< 0.001
Residual	367.102	8.000	45.888		

Nota. Tipo III - Soma dos Quadrados

Tabela 6.6 – Teste post hoc de Tukey para débito técnico

			95% CI para Diferença Média					
		Diferença Média	Mais Baixo	Mais Alto	SE	t	Cohen's d	p_{tukey}
A	B	-32.095	-49.807	-14.383	5.531	-5.803	-3.727	0.002
	C	2.040	-15.672	19.752	5.531	0.369	2.211	0.982
	D	-28.120	-45.832	-10.408	5.531	-5.084	-6.597	0.004
B	C	34.135	16.423	51.847	5.531	6.172	3.979	0.001
	D	3.975	-13.737	21.687	5.531	0.719	0.417	0.887
C	D	-30.160	-47.872	-12.448	5.531	-5.453	-7.188	0.003

Nota. Ajuste do intervalo de confiança: método tukey para comparar uma família de 4 estimativas

Eficiência na verificação de cobertura de testes unitários

Analisando as respostas relacionadas à medida associada à verificação da cobertura do teste unitário, houve diferença estatisticamente significante entre os grupos, conforme demonstrado pela one-way ANOVA (F = 7,636, p = 0,010) e mostrada na Tabela 6.7.

Um teste post hoc de Tukey, mostrado na Tabela 6.8, com um intervalo de confiança de 95%, indica que a eficiência da verificação da cobertura do teste de unidade é estatisticamente significante em relação ao grupo experimental A e ao grupo controle B (p = 0,118) e ao grupo experimental C e grupo controle D (p = 0,018).

Tabela 6.7 – Teste ANOVA para cobertura de testes unitários

Casos	Soma dos Quadrados	df	Média dos Quadrados	F	p
Grupo	4642.558	3.000	1547.519	7.636	0.010
Residual	1621.190	8.000	202.649		

Nota. Tipo III - Soma dos Quadrados

Tabela 6.8 – Teste post hoc de Tukey para cobertura de testes unitários

			95% CI para Diferença Média					
		Diferença Média	Mais Baixo	Mais Alto	SE	t	Cohen's d	p_{tukey}
A	B	-30.142	-67.363	7.080	11.623	-2.593	-3.873	0.118
	C	1.733	-35.488	38.955	11.623	0.149	2.587	0.999
	D	-44.207	-81.428	-6.985	11.623	-3.803	-2.381	0.022
B	C	31.875	-5.347	69.097	11.623	2.742	4.099	0.096
	D	-14.065	-51.287	23.157	11.623	-1.210	-0.699	0.638
C	D	-45.940	-83.162	-8.718	11.623	-3.952	-2.474	0.018

Nota. Ajuste do intervalo de confiança: método tukey para comparar uma família de 4 estimativas

Eficiência na verificação da utilização de recursos computacionais

Analisando as respostas relacionadas à medida associada à verificação do uso de recursos computacionais, houve diferença estatisticamente significante entre os grupos, conforme demonstrado pela one-way ANOVA (F = 16,882, p = < 0,001) e mostrada na Tabela 6.9.

Um teste post hoc de Tukey, mostrado na Tabela 6.10, com um intervalo de confiança de 95%, indica que a eficiência da verificação do uso de recursos computacionais está estatisticamente significante relacionada ao grupo experimental A e ao grupo controle B (p = 0,005) e ao grupo experimental C e grupo controle D (p = 0,004).

Tabela 6.9 – Teste ANOVA para consumo de recursos computacionais

Casos	Soma dos Quadrados	df	Média dos Quadrados	F	p
Grupo	3496.509	3.000	1165.503	16.882	< 0.001
Residual	552.295	8.000	69.037		

Nota. Tipo III - Soma dos Quadrados

Tabela 6.10 – Teste post hoc de Tukey para consumo de recursos computacionais

			95% CI para Diferença Média					
		Diferença Média	Mais Baixo	Mais Alto	SE	t	Cohen's d	p_{tukey}
A	B	-34.022	-55.747	-12.296	6.784	-5.015	-3.607	0.005
	C	2.455	-19.270	24.180	6.784	0.362	1.415	0.983
	D	-31.630	-53.355	-9.905	6.784	-4.662	-4.447	0.007
B	C	36.477	14.751	58.202	6.784	5.377	3.900	0.003
	D	2.392	-19.334	24.117	6.784	0.353	0.206	0.984
C	D	-34.085	-55.810	-12.360	6.784	-5.024	-4.864	0.004

Nota. Ajuste do intervalo de confiança: método tukey para comparar uma família de 4 estimativas

Eficiência na verificação da disponibilidade

Analisando as respostas relacionadas à medida associada à verificação do uso de recursos computacionais, houve diferença estatisticamente significante entre os grupos, conforme demonstrado pela one-way ANOVA (F = 14,717, p = 0,001) e mostrada na Tabela 6.11.

Um teste post hoc de Tukey, mostrado na Tabela 6.12, com um intervalo de confiança de 95%, indica que a eficiência da verificação da disponibilidade está estatisticamente significante relacionada ao grupo experimental A e grupo controle B (p = 0,003) e grupo experimental C e controle grupo D (p = 0,019).

Tabela 6.11 – Teste ANOVA para disponibilidade

Casos	Soma dos Quadrados	df	Média dos Quadrados	F	p
Grupo	43330.656	3.000	14443.552	14.717	0.001
Residual	7851.293	8.000	981.412		

Nota. Tipo III - Soma dos Quadrados

Tabela 6.12 – Teste post hoc de Tukey para disponibilidade

			95% CI para Diferença Média					
		Diferença Média	Mais Baixo	Mais Alto	SE	t	Cohen's d	p_{tukey}
A	B	-133.985	-215.897	-52.073	25.579	-5.238	-9.368	0.003
	C	4.365	-77.547	86.277	25.579	0.171	1.012	0.998
	D	-95.780	-177.692	-13.868	25.579	-3.745	-2.272	0.024
B	C	138.350	56.438	220.262	25.579	5.409	10.144	0.003
	D	38.205	-43.707	120.117	25.579	1.494	0.866	0.483
C	D	-100.145	-182.057	-18.233	25.579	-3.915	-2.388	0.019

Nota. Ajuste do intervalo de confiança: método tukey para comparar uma família de 4 estimativas

C. *Medida 3 - Eficiência usando a sugestão de vários critérios*

Esta medida pretende avaliar a rapidez com que o participante é capaz de localizar o microsserviço mais crítico com vários critérios usando apenas grupos experimentais (A e C). Os grupos A e C apenas usaram a ferramenta implementada com base na abordagem M2CSEA para orientar na atividade de localização. Para analisar os resultados dessa medida, no que diz respeito à eficiência no estado inicial e no estado modificado, também foi utilizado o método one-way ANOVA.

Nesta medida, foram utilizadas 5 questões relacionadas a múltiplos critérios, utilizando uma seleção predefinida associada às preferências do usuário em relação à classificação de verificação. Solicitou-se ao participante que localizasse os microsserviços mais críticos usando sugestões de vários critérios em todas essas cinco perguntas. Afinal, essas questões foram agrupadas para criar um único meio para servir de base a essa medida.

Analisando as respostas relacionadas à medida associada à eficiência por sugestão de múltiplos critérios, não houve diferença estatisticamente significante entre os grupos, conforme demonstrado pela one-way ANOVA (F = 3,5, p = 0,470) e mostrada na Tabela 6.13.

Um teste post hoc de Tukey mostrado na Tabela 6.14 indica que não há diferença estatisticamente significante (p = 0,470) entre os grupos experimentais com experiência anterior em microsserviços (A) e sem experiência anterior (C).

Tabela 6.13 – Teste ANOVA para múltiplos critérios

Casos	Soma dos Quadrados	df	Média dos Quadrados	F	p
Grupo	11.426	1.000	11.426	0.637	0.470
Residual	71.761	4.000	17.940		

Nota. Tipo III - Soma dos Quadrados

Tabela 6.14 – Teste post hoc de Tukey para múltiplos critérios

			95% CI para Diferença Média					
		Diferença Média	Mais Baixo	Mais Alto	SE	t	Cohen's d	p_{tukey}
A	C	2.760	-6.842	12.362	3.458	0.798	0.652	0.470

Com base na Tabela 6.15, considerando o desvio padrão e o valor médio, é possível concluir que a abordagem de múltiplos critérios na abordagem M2CSEA é um método eficiente para localizar os microsserviços críticos.

Tabela 6.15 – Estatística descritiva para múltiplos critérios

	Grupos	
	A	C
Participantes Válidos	3	3
Participantes Inválidos	0	0
Média	6.413	3.653
Desvio Padrão	5.940	0.775
Mínimo	2.110	2.870
Máximo	13.190	4.420

D. *Medida 4 - Influência da proposta na tomada de decisão*

Essa medida pretende avaliar como os participantes dos grupos experimentais A e C consideram a ferramenta proposta útil para a tomada de decisões em cenários reais, considerando uma empresa com vários microsserviços já implantados em uma arquitetura estabelecida e com a intenção de investir recursos em manutenção ou evolução.

Esta questão foi projetada como escala Likert de cinco níveis para analisar os resultados dessa medida. Foi utilizado o teste t de uma amostra. De acordo com DE WINTER e DODOU (2010), as escalas Likert de cinco pontos são comumente associadas a pesquisas e são usadas em uma ampla variedade de configurações. Ao responder a uma escala Likert, os participantes especificam seu nível de concordância com declarações com, tipicamente, cinco ou sete níveis de resposta ordenada. Os dados

do item Likert têm características distintas: valores discretos em vez de contínuos, números vinculados e intervalo restrito.

A literatura afirma que o teste t e Mann-Whitney-Wilcoxon para a escala Likert de cinco níveis, geralmente, têm aplicação equivalente (DE WINTER e DODOU, 2010). O erro tipo I é a rejeição de uma hipótese nula verdadeira, enquanto um erro tipo II é a não rejeição de uma hipótese nula falsa. A taxa de erro tipo I de ambos os métodos nunca foi superior a 3% acima da taxa nominal de 5%, nem mesmo quando o tamanho da amostra era desigual. Considerando essa hipótese, a hipótese é validada com um teste t de uma amostra.

A medida considera que mais da metade dos participantes considera a abordagem útil. Prosseguindo com a pontuação na escala de Likert, atribuindo o valor ascendente a cada nível de Likert, uma pontuação é atribuída a cada resposta da seguinte forma: “Concordo Fortemente” recebe 1 ponto, “Concordo” recebe 2 pontos, “Nem concordo nem discordo” recebe 3 pontos, “Discordo” recebe 4 pontos e “Discordo Fortemente” recebe 5 pontos. Considerando o escore mencionado acima, a média de uma população (mu) no teste t de uma amostra foi de 2,99, em relação à primeira posição relevante antes de um escore neutro (3).

Com base no valor-p (p <0,000), a proposta influencia positivamente o processo de tomada de decisão em relação aos microsserviços mais críticos em arquiteturas estabelecidas.

E. Ferramentas similares (S5Q1)

Esta pergunta pretende obter informações relacionadas à experiência passada do participante com ferramentas semelhantes. Esta pergunta propõe listar as ferramentas capazes de tratar todas as verificações tratadas pela abordagem M2CSEA. Essas verificações, conforme expostas na Seção 4.2.1, são análise de interdependência de microsserviços, análise de tempo de resposta, análise de cobertura de teste de unidade, análise do débito técnico, análise de uso de recursos computacionais e análise de disponibilidade.

Para esta pergunta, 4 de 6 participantes de grupos que usavam a ferramenta baseada na abordagem M2CSEA declararam que não usavam anteriormente uma ferramenta semelhante que considera toda a verificação, um participante declarou que

usava o AWS X-Ray[45] e um participante declarou que usava o Elasticsearch com Kibana e Logstash (ELK[46]) para analisar os dados de microsserviços e extrair as informações de verificação.

F. *Comparação de ferramentas em relação à assertividade (S5Q2)*

Esta pergunta pretende obter informações relacionadas à experiência passada do participante com uma ferramenta semelhante em comparação com a ferramenta proposta pela abordagem M2CSEA em termos de capacidade de ferramentas para auxiliar uma tomada de decisão assertiva.

Considerando que apenas 2 dos 6 participantes, que estavam nos grupos A e C que usaram a ferramenta baseada na abordagem M2CSEA, apenas esses participantes foram capazes de responder a essa pergunta comparando as ferramentas, conforme observado na Tabela E, no Apêndice I.

Houve um consenso entre os participantes de que a ferramenta proposta pode guiar mais do que as ferramentas usadas anteriormente (AWS X-Ray e ELK) para um processo de tomada de decisão.

A ferramenta AWS X-Ray possui a capacidade de verificar microsserviços em execução na infraestrutura da AWS de acordo com o tempo de resposta, disponibilidade e interdependências, entretanto, não é capaz de avaliar sem o auxílio de outros serviços a utilização de recursos computacionais, cobertura de testes unitários e débito técnico. Desta forma, não atende os requisitos de ser uma ferramenta de semelhante completude em relação a ferramenta desenvolvida nesta dissertação e baseada na abordagem M2CSEA.

A ferramenta ELK possui a capacidade de tratar, processar, indexar e buscar *logs*, não possuindo diretamente nenhum meio para verificar os microsserviços de acordo com o tempo de resposta, disponibilidade e interdependências, utilização de recursos computacionais, cobertura de testes unitários ou débito técnico. A justificativa para a citação desta ferramenta foi em virtude de sua capacidade de processar, por meio de logs, os dados relativos as verificações mencionadas, entretanto estes dados obrigatoriamente deveriam ser previamente coletados e tratados para serem enviados aos arquivos de *logs* para posterior análise.

[45] AWS X-Ray: https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html
[46] ELK: https://www.elastic.co/pt/what-is/elk-stack

G. *Comparação de ferramentas em relação à eficiência (S5Q3)*

Esta pergunta pretende obter informações relacionadas à experiência passada do participante com ferramentas semelhantes em comparação com a ferramenta proposta baseada na abordagem M2CSEA em termos de eficiência.

Considerando que apenas 2 dos 6 participantes, que estavam nos grupos A e C que usaram a ferramenta, responderam que usavam uma ferramenta semelhante, apenas esses participantes foram capazes de responder a essa pergunta comparando as ferramentas. Foi consenso entre os participantes que a ferramenta proposta é mais eficiente do que as ferramentas usadas anteriormente (AWS X-Ray e ELK) para localizar o microsserviço mais crítico.

6.8.1. Interpretação dos Resultados

Com base nos resultados obtidos e na agregação de dados executados, nas seções anteriores, as perguntas da pesquisa são respondidas.

RQ1 - A abordagem proposta é assertiva ao sugerir os microsserviços mais críticos em arquiteturas estabelecidas?

Considerando os resultados obtidos na medida 1, é possível afirmar que a abordagem proposta é assertiva, sugerindo o microsserviço mais crítico em uma arquitetura estabelecida.

RQ2 - A abordagem proposta é eficiente em sugerir os microsserviços mais críticos em arquiteturas estabelecidas?

Considerando os resultados obtidos na medida 2, na validação do teste Tukey Post Hoc para a Verificação de Interdependência AB ($p = 0,408$) e CD ($p = 0,278$), Verificação do Tempo de Resposta AB ($p = 0,040$) e CD ($p =. 002$), Verificação técnica de débito AB ($p = 0,002$) e CD ($p = 0,003$), Verificação de cobertura de teste unitário AB ($p = 0,118$) e CD ($p = 0,018$), Verificação de uso de recursos computacionais AB ($p = 0,005$) e CD ($p = 0,004$) e Verificação de disponibilidade com AB ($p = 0,003$) e CD ($p = 0,019$), houve diferença estatisticamente significante entre os grupos.

Considerando os resultados obtidos na medida 3, é possível afirmar que a abordagem proposta é eficiente, sugerindo o microsserviço mais crítico em uma arquitetura estabelecida.

RQ3 - A abordagem proposta pode ajudar os profissionais de engenharia de software no processo de tomada de decisão?

Considerando os resultados obtidos na medida 4 (p <0,000), é possível afirmar que a abordagem proposta pode auxiliar os profissionais de engenharia ao sugerir o microsserviço mais crítico em uma arquitetura estabelecida.

6.8.2. Ameaças à Validade

6.8.2.1. Validação das Hipóteses

Com base no RQ1 e no RQ2, a hipótese nula foi refutada, na qual afirma-se que "a abordagem proposta não é eficiente nem assertiva para sugerir os microsserviços mais críticos em arquiteturas estabelecidas.", desta forma, uma das hipóteses alternativas é verdade.

Considerando RQ1 e RQ2, é possível afirmar que M2CSEA é uma abordagem assertiva e eficiente, refutando a hipótese alternativa 2 e a hipótese alternativa 3, que afirmam respectivamente que "A abordagem proposta é eficiente, mas não assertiva, para sugerir os microsserviços mais críticos em arquiteturas estabelecidas" e "A abordagem proposta não é eficiente, mas assertiva para sugerir os microsserviços mais críticos em arquiteturas estabelecidas".

Finalmente, foi possível concluir que a hipótese alternativa 1 é verdadeira, sendo capaz de afirmar que "a abordagem proposta é eficiente e assertiva para sugerir os microsserviços mais críticos em arquiteturas estabelecidas".

6.8.2.2. Validade do Estudo

Quanto à validade do construto, foi garantido o sigilo e o anonimato das respostas individuais, a fim de mitigar possíveis ameaças à coleta de informações imprecisas devido à apreensão da avaliação.

Em relação à Validade Interna do estudo, o grau de conclusão é considerável, pois o modelo de caracterização da população foi baseado na previsão de questões demográficas associadas à experiência e em relatórios reais de participação no projeto.

Quanto à validade externa do estudo, a seleção dos participantes foi uma amostra sem delimitação geográfica, o que pode representar uma parcela da população.

No entanto, possui grandes limitações de generalização da população como um todo. E as perguntas foram formuladas para permitir a repetição em diferentes contextos.

6.9. Conclusão

Com base nos resultados, foi possível concluir a eficiência, assertividade e relevância da abordagem de múltiplos critérios proposta na sugestão dos microsserviço mais crítico em arquiteturas estabelecidas.

Este estudo limita-se ao estudo de microsserviços, não possuindo por objetivo reduzir a importância de outras arquiteturas ou até destacar a melhor opção em casos gerais, sendo necessário uma análise minuciosa dos requisitos e uma avaliação caso a caso, a fim de tomar uma decisão consciente sobre a melhor opção para o problema apresentado.

O estudo possui limitações em relação a alguns parâmetros não funcionais que podem gerar processamento extra para sistemas sob validação, por exemplo, segurança (por exemplo, teste de penetração e desempenho como teste de carga) ou aquele que requer dependências de infraestrutura, por exemplo, escalabilidade e tolerância ao erro.

Em trabalhos futuros, a ferramenta baseada na abordagem M2CSEA será comparada com outras ferramentas, com o objetivo de validar as melhorias a serem aplicadas. Trabalhos futuros também considerarão a implementação de novos métodos de verificação, bem como a ampliação da cobertura da ISO 25010, conforme exposto nas Tabelas 4.1 e 4.2, localizadas na Seção 4.2.1.7.

A abordagem M2CSEA também será considerada em implementações baseadas em um ambiente estável e de uso contínuo (produção), em vez de uma arquitetura simulada, bem como a coleta de preferências do usuário sendo executada com usuários reais que pretendem promover o processo de tomada de decisão baseado em múltiplos critérios mais assertivo de uma arquitetura estabelecida orientadas a microsserviços.

CAPÍTULO 7 – CONCLUSÕES

7.1. Epílogo

Após a implementação de uma arquitetura de microsserviços, o processo de manutenção tende a ser complexo, considerando os vários serviços a serem observados. A necessidade de conhecer os microsserviços mais críticos vem dessa complexidade de manutenção. Diferentes partes interessadas precisam conhecer esses serviços para promover maior estabilidade arquitetural ou, eventualmente, a evolução dos serviços nessa arquitetura. A verificação de criticidade dos microsserviços em arquiteturas estabelecidas pode orientar os profissionais de engenharia de software para um processo de tomada de decisão mais fundamentado.

Esta dissertação de mestrado apresentou uma abordagem denominada M2CSEA para a sugestão dos microsserviços mais críticos em arquiteturas estabelecidas com base em múltiplos critérios. O modelo proposto por esta dissertação é baseado em critérios não funcionais que são elencados pelos usuários de acordo com suas necessidades. As informações sobre os microsserviços mais críticos são relevantes para arquitetos de software e outros tomadores de decisão, orientando assim a manutenção e a evolução da arquitetura de uma maneira mais assertiva.

Para o desenvolvimento deste trabalho, uma revisão da literatura foi realizada (Capítulo 2), uma validação com especialistas foi conduzida (Capítulo 3), influenciando na sistemática da abordagem (Capítulo 4), bem como no ferramental de suporte implementado (Capítulo 5). Para validação da abordagem foi um estudo de caso (Capítulo 6). A partir dos resultados, verificou-se indícios de aceitação do ferramental de suporte no contexto da sugestão dos microsserviços mais críticos em arquiteturas estabelecidas, sendo sido possível concluir que a abordagem é mais eficiente em localizar os microsserviços mais críticos em arquiteturas estabelecidas que outras ferramentas, assim como é mais assertiva em guiar o processo de tomada de decisão.

Neste capítulo, além desta seção, as contribuições desta dissertação são apresentadas na Seção 7.2. Na Seção 7.3, as limitações da pesquisa são discutidas. Na Seção 7.4, as publicações geradas no escopo desta dissertação de mestrado são apresentadas. Por fim, na Seção 7.5, algumas possibilidades de trabalhos futuros são descritas.

7.2. Contribuições

Esta dissertação de mestrado apresentou os resultados de uma pesquisa que estabeleceu uma abordagem para apoiar a tomada de decisão na definição dos microsserviços mais críticos em arquiteturas estabelecidas por meio de múltiplos critérios. Como contribuições alcançadas, incluem-se:

- **Caracterização de Criticidade em Arquiteturas Estabelecidas orientadas a Microsserviços**, por meio da condução da Revisão Sistemática da Literatura e resultados observados ao longo do Capítulo 2;

- **Validação com Especialistas,** por meio da condução de um *survey*, no contexto de especialistas da área de desenvolvimento de software orientados a microsserviços, tendo sido possível concluir a necessidade da definição de uma abordagem para auxiliar a tomada de decisão para escolha dos microsserviços mais críticos, tendo sido detalhado ao longo do Capítulo 3.

- **Definição de uma abordagem para Sugestão dos Microsserviços mais críticos em Arquiteturas Estabelecidas orientadas a Microsserviços**, com base no AHP e utilização de múltiplos critérios aliados à escolha das preferências dos usuários descrita no Capítulo 4;

- **Implementação do ferramental de suporte**, oferecendo apoio para a validação da abordagem e sugestão de utilização, descrito no Capítulo 5.

- **Avaliação preliminar do ferramental de suporte,** por meio de um estudo de caso, no contexto de praticantes experientes de engenharia de software, descrito no Capítulo 6.

A questão de pesquisa apresentada no Capítulo 1 foi utilizada para direcionar essa dissertação de mestrado. Neste ponto, os resultados da revisão da literatura apresentados no Capítulo 2, colaboram para compreender como as diferentes abordagens para seleção e utilização de múltiplos critérios, identificando também a falta de um estudo acerca da criticidade em microsserviços.

Nesta dissertação, para responder esta questão, uma abordagem para sugestão dos microsserviços mais críticos foi estabelecida com base em múltiplos critérios. Dessa forma, a validação com especialistas (Capítulo 3), a sistemática da abordagem M2CSEA (Capítulo 4), seu ferramental de suporte (Capítulo 5) e a validação da abordagem considerando aceitação do ferramental implementado (Capítulo 6), contribuem para solucionar esta questão ao trabalhar com múltiplas verificações em arquiteturas estabelecidas com base nas preferências dos usuários e obter indícios de aceitação do ferramental de suporte no contexto de praticantes de engenharia de software em busca dos microsserviços mais críticos em arquiteturas estabelecidas.

No entanto, criticidade em arquiteturas estabelecidas orientadas a microsserviços é um tópico de pesquisa recente e estudos experimentais realizados por diferentes grupos de pesquisa são necessários para responder plenamente essa questão de pesquisa.

7.3. Limitações

Algumas limitações podem ser identificadas nesta pesquisa ao realizar uma análise crítica, tanto na abordagem apresentada quanto no seu ferramental de suporte. Dentre as principais limitações, estão:

- **Maior cobertura da ISO 25010:** a abordagem M2CSEA possui limitações quanto a cobertura das características propostas pelo ISO 25010:2011, conforme exposto nas Tabelas 4.1 e 4.2, localizadas na Seção 4.2.1.7. Para uma maior cobertura se faz necessário incluir verificações complementares, a fim de compor uma maior variedade de dados coletados e analisados.

- **Parâmetros não funcionais sem processamento adicional:** a abordagem M2CSEA possui limitações em relação a parâmetros não funcionais que possam gerar processamento extra para sistemas sob validação, por exemplo, segurança (por exemplo, teste de penetração) e desempenho (por exemplo, teste de carga) ou demais testes que possam requerer dependências de infraestrutura, por exemplo, escalabilidade e tolerância a falhas.

- **Avaliação de dados históricos:** as verificações da abordagem M2CSEA consideram o estado atual dos microsserviços, desconsiderando os dados históricos para o cálculo da criticidade, entretanto a utilização destes dados pode vir a ser útil

para servir como base para uma análise de tendência para validação de melhoria ou piora de cada uma das verificações ao longo do tempo.

- **Avaliação mais frequente dos dados:** no ferramental de suporte, foi utilizada a frequência mínima de 1 dia para agregação dos dados e verificação da criticidade. Esta é uma limitação do ferramental que pode ser ampliada para que o usuário selecione o tempo em que a arquitetura deve ser verificada, de acordo com sua necessidade.

- **Diferentes tipos de protocolos de comunicação**: no ferramental de suporte da abordagem M2CSEA, foram analisadas somente comunicações via HTTP, não havendo uma implementação com outros protocolos de comunicação comumente utilizados em arquiteturas orientadas a microsserviços, como AMQP, por exemplo.

- **Múltiplas instâncias do mesmo microsserviços em execução**: quanto ao número de instâncias dos microsserviços, o ferramental de suporte da abordagem M2CSEA somente considerou uma instância de cada microsserviços enviando informações acerca das verificações, entretanto este cenário é incomum em um ambiente produtivo de sistemas de grande porte.

- **Dados adicionais de consumo de recursos computacionais:** acerca dos dados de consumo de recursos computacionais, foram considerados na abordagem M2CSEA somente os dados referente a memória RAM e CPU, não tendo sido considerados outros insumos de menor importância, mas que em um contexto de um ambiente de produção podem ser úteis, como I/O de disco, I/O de rede e memória swap.

- **Suporte a múltiplas áreas de negócio:** melhorias podem ser realizadas para inserir recursos com o objetivo de apoiar a obtenção de opiniões de usuários de múltiplas áreas da mesma empresa, para servir de base para auxiliar a tomada de decisão entre equipes.

- **Verificação ativa**: a fim de possuir os dados referente as verificações dos microsserviços de forma atualizada e constante, o ferramental de implementação da abordagem M2CSEA pode contemplar a criação de um novo componente a ser executado de maneira ininterrupta. Este novo componente possuirá por objetivo a execução das verificações da abordagem sem que seja necessário aguardar uma requisição ao serviço para coleta das métricas.

- **Agregação das Análises por microsserviço**: a listagem de microsserviços de acordo com sua criticidade se dá baseado no cálculo individual das verificações. Uma melhoria pode ser implementada para que todas as verificações de um microsserviço sejam agrupadas para gerar um valor único referente ao microsserviço em questão. Esta análise pode servir como um complemento à análise individual, caso o usuário do ferramental opte por esta agregação.

7.4. Publicações

No escopo desta dissertação de mestrado, três artigos foram elaborados e submetidos para conferências internacionais, resultando nas seguintes publicações:

- A revisão da literatura foi aceita na XXIII Conferência Ibero-Americana em Engenharia de Software (CibSE 2020), a ser realizada no dia 16 de novembro de 2020, com o artigo intitulado "*Selection Methods for Criticality in Microservices Architectures: A Systematic Literature Review*".

- O *survey* para validação da proposta com especialistas foi apresentado na I Conferência Internacional sobre Sistemas de Informação e Tecnologias de Software, no dia 15 de novembro de 2019 em Quito / Equador, com o artigo "*A Survey on Microservices Criticality Attributes on Established Architectures*", tendo sido agraciado com o prêmio de melhor artigo nesta edição da conferência (SANTOS e WERNER, 2019).

- O estudo de caso para avaliação da abordagem será submetido ao XIV Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software, a ser realizado em outubro de 2020, com o artigo intitulado "*Multiple Criteria Approach on Criticality Suggestion to Microservices Architectures*".

7.5. Trabalhos Futuros

O tópico de criticidade em arquiteturas orientadas a microsserviços ainda apresenta diversos desafios. A partir da abordagem apresentada e ferramental de suporte implementado ao longo desta pesquisa, é possível identificar oportunidades de melhoria. Entre as perspectivas de trabalhos futuros, se destacam:

- Comparar a abordagem M2CSEA com outras ferramentas, com o objetivo de validar as melhorias a serem aplicadas na abordagem em questão.

- Implementar novos métodos de verificação, bem como a ampliação da cobertura da ISO 25010, conforme exposto nas Tabelas 4.1 e 4.2, localizadas na Seção 4.2.1.7, referente a Análise e Qualidade de Software em Uso.

- Correlacionar e implementar novos métodos de verificação de acordo com o modelo QPS (ROCHA et al., 2018), o qual avalia produtos de software de acordo com quatro dimensões (Organizacional, Engenharia de Software, Serviço e Qualidade do Produto), permitindo uma avaliação multidimensional da qualidade do produto associado aos métodos de verificação utilizados.

REFERÊNCIAS BIBLIOGRÁFICAS

ABOWD, G.; BASS, L.; CLEMENTS, P.; KAZMAN, R.; NORTHROP, L. *Recommended Best Industrial Practice for Software Architecture Evaluation***.** [s.l.] Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1997.

ALPERS, S.; BECKER, C.; OBERWEIS, A.; SCHUSTER, T. *Microservice Based Tool Support for Business Process Modelling*. Proceedings of the 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop. *Anais*... In: EDOCW. p. 71–78, Adelaide, Australia: IEEE, 2015.

AVGERIOU, P.; KRUCHTEN, P.; OZKAYA, I.; SEAMAN, C. Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). *Dagstuhl Reports*, v. 6, n. 4, p. 110–138, 2016.

AVRITZER, A.; FERME, V.; JANES, A.; RUSSO, B.; SCHULZ, H.; VAN HOORN, A. *A Quantitative Approach for the Assessment of Microservice Architecture Deployment Alternatives by Automated Performance Testing*. (C. E. Cuesta, D. Garlan, J. Pérez, Eds.)Software Architecture. *Anais*... In: ECSA. p. 159–174, Madrid, Spain: Springer International Publishing, set. 2018.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. *CoRR*, v. abs/1507.08217, 2015.

BATZEL, T. D.; SWANSON, D. C. Prognostic Health Management of Aircraft Power Generators. IEEE Transactions on Aerospace and Electronic Systems, v. 45, n. 2, p. 473–482, 2009.

BELQASMI, F.; SINGH, J.; MELHEM, S. Y. B.; GLITHO, R. H. Soap-based vs. restful web services: A case study for multimedia conferencing. *IEEE*, v. 16, n. 4, p. 54–63, 2012.

BENGTSSON, P. *Towards Maintainability Metrics on Software Architecture: An Adaptation of Object-Oriented Metrics*. First Nordic Workshop on Software Architecture. *Anais*... In: NOSA. Ronneby, Sweden: ago. 1998.

BERG, T. VAN DEN; SIEGEL, B.; CRAMP, A. Containerization of high level architecture-based simulations: A case study. *The Journal of Defense Modeling and Simulation*, v. 14, n. 2, p. 115–138, 2017.

BONDI, A. B. *Characteristics of Scalability and Their Impact on Performance*. Proceedings of the 2nd International Workshop on Software and Performance. *Anais*... In: WOSP. p. 195–203, Ottawa, Canada: ACM, 2000.

BRATAAS, G.; HUGHES, P. Exploring architectural scalability. *ACM SIGSOFT Software Engineering Notes*, v. 29, n. 1, p. 125–129, 2004.

BROWN, N.; CAI, Y.; GUO, Y.; KAZMAN, R.; KIM, M.; KRUCHTEN, P.; LIM, E.; MACCORMACK, A.; NORD, R.; OZKAYA, I.;. *Managing Technical Debt in Software-Reliant Systems***.** Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. Anais... In: FOSER. p. 47–52, Santa Fe, USA: ACM, 2010.

BURNS, B. et al. Borg, Omega, and Kubernetes. *Queue*, v. 14, n. 1, p. 70–93, jan. 2016.

BUSCHMANN, F. *Pattern Oriented Software Architecture A System of Patterns*. [s.l.] Ashish Raut, 1996. v. 1.

CAYRAC, D.; DUBOIS, D.; PRADE, H. Handling uncertainty with possibility theory and fuzzy sets in a satellite fault diagnosis application. IEEE Transactions on Fuzzy Systems, v. 4, n. 3, p. 251–269, 1996.

CHAUMUN, M. A.; KABAILI, H.; KELLER, R. K.; LUSTMAN, F. A change impact model for changeability assessment in object-oriented software systems. *Science of Computer Programming*, v. 45, n. 2–3, p. 155–174, 2002.

CHYBOWSKI, L.; GAWDZINSKA, K. *On the possibilities of applying the AHP method to a multi-criteria component importance analysis of complex technical objects*. (Rocha, A and Correia, AM and Adeli, H and Reis, LP and Teixeira, MM, Ed.)New Advances in Information Systems and Technologies,. *Anais*...: Advances in Intelligent Systems and Computing. p. 701–710, Berlin, Germany: Springer, 2016.

CLEMENTS, P., GARLAN, D., BASS, L., STAFFORD, J., NORD, R., IVERS, J., LITTLE, R. *Documenting software architectures: views and beyond*. [s.l.] Pearson Education, 2002.

COMMISSION, I.E. IEC 61508:2010. *Functional safety of electrical, electronic, programmable electronic safety-related systems*, 2010.

CORNETT, S. Code coverage analysis. *Bullseye Testing Technology*, 2002.

COSTA, M. F. *Director: A Cloud Microservice Selection Framework*. PhD Thesis—[s.l.] Universidade Federal do Rio de Janeiro, 2019.

COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. [s.l.] Pearson Education, 2005.

DAYA, SHAHIR, NGUYEN VAN DUY, KAMESWARA EATI, CARLOS M. FERREIRA, DEJAN GLOZIC, VASFI GUCER, MANAV GUPTA, SUNIL JOSHI, VALERIE LAMPKIN, MARCELO MARTINS. *Microservices from*

Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach. 1. ed. [s.l.] IBM Redbooks, 2016.

DE GROOT, J. DE; NUGROHO, A.; BÄCK, T.; VISSER, J. *What is the value of your software?* Third International Workshop on Managing Technical Debt (MTD). *Anais...*: MTD '12.Zurich, Switzerland: IEEE Press, p. 37-44, jun. 2012.

DE WINTER, J. C.; DODOU, D. Five-point Likert items: t test versus Mann-Whitney-Wilcoxon. *Practical Assessment, Research & Evaluation*, v. 15, n. 11, p. 1–12, 2010.

DOBRICA, L.; NIEMELA, E. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, v. 28, n. 7, p. 638–653, jul. 2002.

DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. Microservices: Yesterday, Today, and Tomorrow. In: MAZZARA, M.; MEYER, B. (Eds.). *Present and Ulterior Software Engineering*. Cham: Springer International Publishing, 2017. p. 195–216.

DUEÑAS, J. C.; DE OLIVEIRA, W. L.; JUAN, A. *A software architecture evaluation model*. Lecture Notes in Computer Science. *Anais*... In: ARES. p. 148–157 Berlin, Heidelberg: Springer, 1998.

DYSON, P. AHP and expert choicoggo a step beyond the spreadsheet. *Seybold Report: Analyzing Publishing Technologies*, v. 3, n. 4, p. 20–21, 2003.

EDDIN, M. C.; MAMMERI, Z. *Non-functional properties aware configuration selection in component-based systems*. 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. *Anais*... In: SNPD. p. 1-7, Las Vegas, NV, USA: IEEE, jun. 2014.

ESPOSITO, C.; CASTIGLIONE, A.; CHOO, K. K. R. Challenges in Delivering Software in the Cloud as Microservices. *IEEE Cloud Computing*, v. 3, n. 5, p. 10–14, set. 2016.

FAKHFAKH, N.; VERJUS, H.; POURRAZ, F. *Multi-criteria Decision Making Method for Quality of Service Aggregation*. 2011 IEEE 15th International Enterprise Distributed Object Computing Conference. *Anais*... In: EDOC. p. 203-212, Helsinki, Finland: Agosto 2011.

FALESSI, D.; VOEGELE, A. *Validating and prioritizing quality rules for managing technical debt: An industrial case study*. 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD). *Anais*...Bremen, Germany: IEEE, p. 41-48, out. 2015.

FARIDI, M. S.; JAVED, Z.; ABID, M. H.; MUDASSAR, A.; NGADI, A. *IROTS: A Proposed COTS Evaluation & Selection Methodology for Component Based Software Engineering in Under-Development Countries*. 2nd International Conference on Advances in Computer Science and Engineering (CSE 2013). p. 1-4, *Anais*...Atlantis Press, jul. 2013.

FINK, A. *The Survey Handbook*. 2. ed. Thousand Oaks, California: [s.n.].

FRANÇA, M.; WERNER, C. *Perspectives for Selecting Cloud Microservices*. 2018 IEEE International Conference on Software Architecture Companion (ICSA-C). Anais. p. 56-59, Seattle, USA: IEEE, maio 2018.

FREITAS, A. L. P.; MARINS, C. S.; DE OLIVEIRA SOUZA, D. A metodologia de multicritério como ferramenta para a tomada de decisões gerenciais: um estudo de caso. *Revista GEPROS*, n. 2, p. 51, 2006.

GALLETTA, D.; HENRY, R.; MCCOY, S.; POLAK, P. Web site delays: How slow can you go. *Journal of Association of Information Systems*, v. 5, n. 1, p. 1–28, 2004.

GAMMA, E. *Design patterns: elements of reusable object-oriented software*. [s.l.] Pearson Education India, 1995.

GARLAN, D.; SHAW, M. *Advances in Software Engineering and Knowledge Engineering, volume I, chapter An Introduction to Software Architecture*. [s.l.] World Scientific Publishing Company, 1993a.

GARLAN, D.; SHAW, M. An introduction to software architecture. In: *Advances in software engineering and knowledge engineering.* [s.l.] World Scientific, 1993b. p. 1–39.

GHIROTTI, S. E.; REILLY, T.; RENTZ, A. Tracking and Controlling Microservice Dependencies. *Communications of the ACM*, v. 61, n. 11, p. 98–104, out. 2018.

GOLDEN, B. L.; WANG, Q. An alternate measure of consistency. In: *The Analytic Hierarchy Process*. [s.l.] Springer, 1989. p. 68–81.

HASELBOCK, S.; WEINREICH, R. *Decision Guidance Models for Microservice Monitoring*. 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). *Anais*... In: ICSAW. p.54-61, Gothenburg, Sweden: IEEE, abr. 2017.

HASELBÖCK, S.; WEINREICH, R.; BUCHGEHER, G. *Decision Models for Microservices: Design Areas, Stakeholders, Use Cases, and Requirements*. (A. Lopes, R. de Lemos, Eds.)Software Architecture. *Anais*... In: ECSA 2017. Canterbury, UK: Springer International Publishing, p. 155-170, ago. 2017.

HAUPT, F.; LEYMANN, F.; SCHERER, A.; VUKOJEVIC-HAUPT, K. *A framework for the structural analysis of REST APIs*. 2017 IEEE International Conference on Software Architecture (ICSA). *Anais*... In: ICSA. Gothenburg, Sweden: IEEE, p. 55-58, abr. 2017.

HEGEMAN, E. InfoSupport on the quality of quality models. Master's Thesis—[s.l.] University of Twente, 2011.

HOFMEISTER, C.; NORD, R.; SONI, D. *Applied software architecture*. [s.l.] Addison-Wesley Professional, 2000.

HUANG, H.; DONG, Z. *Research on architecture and query performance based on distributed graph database neo4j*. 2013 3rd International Conference on Consumer Electronics, Communications and Networks. *Anais*... In: CECNET. Xianning, China: IEEE, p. 533-536, nov. 2013.

HUMBLE, J.; FARLEY, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. [s.l.] Pearson Education, 2010.

IEEE. IEEE Standard for a Software Quality Metrics Methodology. *IEEE Std 1061-1992*, p. 1–96, mar. 1993.

IZURIETA, C.; GRIFFITH, I.; HUVAERE, C. *An Industry Perspective to Comparing the SQALE and Quamoco Software Quality Models*. 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). *Anais*...: ESEM '17.Toronto, Canada: IEEE Press, p. 287-296, nov. 2017.

JALALIYOON, N.; ABU BAKAR, N.; TAHERDOOST, H. Accomplishment of critical success factor in organization; using analytic hierarchy process. *International Journal of Academic Research in Management (IJARM)*, v. 1, n. 1, 2012.

JANSEN, A.; BOSCH, J. *Software Architecture as a Set of Architectural Design Decisions*. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05). *Anais*... In: WICSA. Pittsburgh, USA: IEEE, p. 109-120, nov. 2005.

JOGALEKAR, P.; WOODSIDE, M. Evaluating the scalability of distributed systems. *IEEE Transactions on parallel and distributed systems*, v. 11, n. 6, p. 589–603, 2000.

JOUILI, S.; VANSTEENBERGHE, V. *An empirical comparison of graph databases*. 2013 International Conference on Social Computing. *Anais*... In: SOCIALCOM. Alexandria, USA: IEEE, p. 708–715, set. 2013.

JURISTO, N.; MORENO, A. M. *Basics of software engineering experimentation*. 1. ed. [s.l.] Springer Science & Business Media, 2013.

KAUR, J.; TOMAR, P. *A Software Component Selection Technique based on Fuzzy Clustering.* 2016 1st India International Conference on Information Processing. *Anais*... In: IICIP. p. 1-5, Delhi, India: IEEE, ago. 2016.

KHARCHENKO, A.; HALAY, I.; BODNARCHUK, I. *Multicriteria architecture choice of software system under design and reengineering*. 2016 XIth International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT). *Anais*... In: CSIT. Lviv, Ukraine: IEEE, 4-8, set. 2016.

KRUCHTEN, P.; NORD, R. L.; OZKAYA, I. Technical Debt: From Metaphor to Theory and Practice. *IEEE Software*, v. 29, n. 6, p. 18–21, nov. 2012.

LETOUZEY, J.-L.; ILKIEWICZ, M. Managing technical debt with the sqale method. *IEEE software*, v. 29, n. 6, p. 44–51, 2012.

LEWIS, G. A.; LAGO, P.; AVGERIOU, P. *A decision model for cyber-foraging systems*. 13th Working IEEE/IFIP Conference on Software Architecture. *Anais*... In: WICSA. Venice, Italy: IEEE, p. 51–60, abr. 2016.

LIU, H. H. *Software performance and scalability: a quantitative approach*. [s.l.] John Wiley & Sons, v. 7, 2011.

MACLEAN, A.; YOUNG, R. M.; BELLOTTI, V. M.; MORAN, T. P. Questions, options, and criteria: Elements of design space analysis. *Human–computer interaction*, v. 6, n. 3–4, p. 201–250, 1991.

MAHMOOD, Z. Software Products and Technologies for the Development and Implementation of SOA. *WSEAS Transactions on Computer Research*, v. 3, n. 1, p. 28–34, jan. 2008.

MARINESCU, R. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, v. 56, n. 5, p. 9–1, 2012.

MARTIN, R. C. *The principles, patterns, and practices of agile software development*. [s.l.] Prentice-Hall, 2003.

MESSERSCHMITT, D. G. The convergence of telecommunications and computing: What are the implications today? *Proceedings of the IEEE*, v. 84, n. 8, p. 1167–1186, 1996.

MOAVEN, S. et al. *A fuzzy model for solving architecture styles selection multi-criteria problem*. 2008 Second UKSIM European Symposium on Computer Modeling and Simulation. *Anais*... In: UKSIM. p. 388–393, Liverpool, UK: IEEE, 2008.

MONTESI, F.; WEBER, J. Circuit Breakers, Discovery, and API Gateways in Microservices. *CoRR*, v. abs/1609.05830, 2016.

MORAES, E. A.; SANTALIESTRA, R. Modelo de decisão com múltiplos critérios para escolha de software de código aberto e software de código fechado. *Revista Organizações em Contexto*, v. 4, p. 59–83, 2008.

NEWMAN, S. *Building microservices: designing fine-grained systems*. 1. ed. [s.l.] O'Reilly Media, Inc., 2015.

NORMALIZACIÓN, O. I. DE. *ISO/IEC 25010:2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models*. [s.l.] ISO, 2011.

PAPAZOGLOU, M. P.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, v. 40, n. 11, p. 38–45, nov. 2007.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, v. 15, n. 12, p. 1053–1058, 1972.

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

PETERSEN, K.; WOHLIN, C. *Context in industrial software engineering research*. 2009 3rd International Symposium on Empirical Software Engineering and Measurement. *Anais*... In: ESEM. Lake Buena Vista, USA: IEEE, p. 401-404, out. 2009.

PFLEEGER, S. L. Design and analysis in software engineering: the language of case studies and formal experiments. *ACM SIGSOFT Software Engineering Notes*, v. 19, n. 4, p. 16–20, 1994.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. 6. ed. [s.l.] McGraw-Hill Higher Education, 2005.

ROBSON, C. Real World Research: A Resource for Social Scientists and Practitioner-Researchers., 2nd edn.(Blackwell Publishing: Oxford, UK.). 2002.

ROCHA, A.; TRAVASSOS, G.; SANTOS, G.; REINEHR, S. QPS-Modelo para Avaliação da Qualidade de Produtos de Software: Resultados Iniciais. 2018.

RUNESON, P.; HOST, M.; RAINER, A.; REGNELL, B. *Case study research in software engineering: Guidelines and examples*. [s.l.] John Wiley & Sons, 2012.

SAATY, T. L. The analytic hierarchy process McGraw-Hill. *New York*, v. 324, 1980.

SANTOS, E. F. M. DE O. DOS; WERNER, C. M. L. *A Survey on Microservices Criticality Attributes on Established Architectures*. 2019 International Conference on Information Systems and Software Technologies. *Anais*... In: ICI2ST. Quito, Equador: nov. 2019.

SAYARA, A.; TOWHID, MD. S.; HOSSAIN, MD. S. *A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling*. 2017 20th International Conference of Computer and Information Technology. *Anais*... In: ICCIT. p. 1-6, Dhaka, Bangladesh: IEEE, dez. 2017.

SELTMAN, H. J. Experimental design and analysis. *Department of Statistics at Carnegie Mellon (Online Only)*, 2009.

SHAO, J.; ZHANG, X.; CAO, Z. *Research on Context-based Instances Selection of Microservice*. Proceedings of the 2nd International Conference on Computer Science and Application Engineering. *Anais*... In: CSAE. p. 1-5, Hohhot, China: ACM, 2018.

SPINELLIS, D.; GOUSIOS, G. *Beautiful architecture: leading thinkers reveal the hidden beauty in software design*. 1. ed. [s.l.] O'Reilly Media, Inc., 2009.

STEVENS, W. P.; MYERS, G. J.; CONSTANTINE, L. L. Structured design. *IBM Systems Journal*, v. 13, n. 2, p. 115–139, 1974.

STUBBS, J.; MOREIRA, W.; DOOLEY, R. *Distributed Systems of Microservices Using Docker and Serfnode*. 2015 7th International Workshop on Science Gateways. *Anais*... In: IWSG. p. 34-39, Budapest, Hungary: IEEE, jun. 2015.

TAHERDOOST, H. Decision Making Using the Analytic Hierarchy Process (AHP); A Step by Step Approach. *International Journal of Economics and Management Systems*, v. 2, p. 244–246, 2017.

TOFFETTI, G.; BRUNNER, S.; BLÖCHLINGER, M.; DUDOUET, F.; EDMONDS, A. *An Architecture for Self-managing Microservices*. Proceedings of the 1st International Workshop on Automated Incident Management in Cloud. *Anais*... In: AIMC. p. 19–24, New York, NY, USA: ACM, 2015.

TOM, E.; AURUM, A.; VIDGEN, R. An exploration of technical debt. *Journal of Systems and Software*, v. 86, n. 6, p. 1498–1516, 2013.

TRIPATHY, P.; NAIK, K. *Software evolution and maintenance*. [s.l.] John Wiley & Sons, 2014.

TWEED, R.; JAMES, G. A universal nosql engine, using a tried and tested technology. *White Paper, Creative Commons Attribution CC-BY*, 2010.

VILLAMIZAR, M.; GARCÉS, O.; CASTRO, H.; VERANO, M.; SALAMANCA, L.; CASALLAS, R.; GIL, S. *Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud*. 10th Computing

Colombian Conference. *Anais*... In: 10CCC. p. 583-590, Bogota, Colombia: IEEE, set. 2015.

VISSER, J.; RIGAL, S.; WIJNHOLDS, G.; VAN ECK, P.; VAN DER LEEK, R. Building Maintainable Software, Java Edition: Ten Guidelines for Future-Proof Code; O'Reilly Media, Inc., 2016.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in Software Engineering*. [s.l.] Springer Berlin Heidelberg, 2012.

WOLFF, E. *Microservices: Flexible Software Architecture*. [s.l.] Addison-Wesley Professional, 2016.

ZAYARAZ, G.; VIJAYALAKSHMI, S.; VIJAYALAKSHMI, V. *Evaluation of software architectures using multicriteria fuzzy decision making technique*. 2009 International Conference on Intelligent Agent & Multi-Agent Systems. *Anais*... In: IAMA. p. 1–5, Chennai, India: IEEE, 2009.

ZELKOWITZ, M. V.; WALLACE, D. R. Experimental models for validating technology. *Computer*, v. 31, n. 5, p. 23–31, 1998.

ZIMMERMANN, O.; MIKSOVIC, C. Decisions required vs. decisions made: connecting enterprise architects and solution architects via guidance models. In: *Aligning Enterprise, System, and Software Architectures*. [s.l.] IGI Global, 2013. p. 176–208.

ZIMMERMANN, O.; WEGMANN, L.; KOZIOLEK, H.; GOLDSCHMIDT, T. *Architectural decision guidance across projects-problem space modeling, decision backlog management and cloud computing knowledge*. 12th Working IEEE/IFIP Conference on Software Architecture. *Anais*... In: WICSA. p. 85–94, Montreal, Canada: IEEE, maio 2015.

Apêndice A – Características e Subcaracterísticas de Qualidade em Uso segundo a ISO 25010:2011

A seguir são apresentados detalhes acerca das características e subcaracterísticas propostas pela norma ISO 25010:2011 e o relacionamento com a abordagem proposta pelo presente trabalho. As subcaracterísticas de adequação funcional, usabilidade e portabilidade não foram consideradas neste apêndice por não estarem associadas a requisitos não funcionais ou não estarem relacionadas a arquiteturas estabelecidas.

Esta dissertação utiliza as seguintes características e subcaracterísticas do modelo de qualidade em uso da ISO 25010:2011 (NORMALIZACIÓN, 2011):

Característica: Adequação funcional *(Functional Suitability)*

Essa característica representa o grau em que um produto ou sistema fornece funções que atendem às necessidades declaradas e implícitas quando usadas em condições especificadas. Essa característica é composta pelas seguintes subcaracterísticas:

- Completude funcional (*Functional completeness*): Grau em que o conjunto de funções cobre todas as tarefas e objetivos do usuário especificados.
- Corretude funcional (*Functional correctness*): Grau em que um produto ou sistema fornece os resultados corretos com o grau de precisão necessário.
- Adequação funcional (*Functional appropriateness*): Grau em que as funções facilitam a realização de tarefas e objetivos especificados.

Característica: Eficiência de desempenho *(Performance efficiency)*

Essa característica representa o desempenho em relação à quantidade de recursos utilizados nas condições estabelecidas. Essa característica é composta pelas seguintes subcaracterísticas:

- Comportamento temporal (*Time behavior*): grau em que a resposta e o tempo de processamento e as taxas de transferência de um produto ou sistema, ao executar suas funções, atendem aos requisitos.
- Utilização de recursos (*Resource utilization*): Grau em que as quantidades e os tipos de recursos utilizados por um produto ou sistema, ao desempenhar suas funções, atendem aos requisitos.

- Capacidade (*Capacity*): Grau em que os limites máximos de um parâmetro de produto ou sistema atendem aos requisitos.

Característica: Compatibilidade (*Compatibility*)

Grau em que um produto, sistema ou componente pode trocar informações com outros produtos, sistemas ou componentes e / ou executar as funções necessárias, enquanto compartilha o mesmo ambiente de hardware ou software. Essa característica é composta pelas seguintes subcaracterísticas:

- Coexistência (*Co-existence*): Grau em que um produto pode executar suas funções necessárias com eficiência, compartilhando um ambiente e recursos comuns com outros produtos, sem impacto negativo em qualquer outro produto.
- Interoperabilidade (*Interoperability*): Grau em que dois ou mais sistemas, produtos ou componentes podem trocar informações e usar as informações que foram trocadas.

Característica: Usabilidade (*Usability*)

Essa característica representa o grau em que um produto ou sistema pode ser usado por usuários especificados para atingir metas especificadas com eficácia, eficiência e satisfação em um contexto de uso determinado. Essa característica é composta pelas seguintes subcaracterísticas:

- Reconhecimento de adequação (*Appropriateness recognizability*): grau em que os usuários podem reconhecer se um produto ou sistema é apropriado para suas necessidades.
- Aprendizagem (*Learnability*): grau em que um produto ou sistema pode ser usado por usuários especificados para atingir objetivos específicos de aprender a usar o produto ou sistema com eficácia, eficiência, liberdade de risco e satisfação em um contexto determinado de uso.
- Operabilidade (*Operability*): grau em que um produto ou sistema possui atributos que facilitam a operação e o controle.
- Proteção contra erros do usuário *(User error protection)*: grau em que um sistema protege os usuários contra erros.

- Estética da interface do usuário (*User interface aesthetics*): grau em que uma interface do usuário permite uma interação agradável e satisfatória para o usuário.
- Acessibilidade (*Accessibility*): grau em que um produto ou sistema pode ser usado por pessoas com a mais ampla variedade de características e capacidades para atingir uma meta especificada em um contexto de uso determinado.

Característica: Confiabilidade (*Reliability*)

Essa característica representa o grau em que um sistema, produto ou componente executa funções específicas em condições determinadas por um dado período. Essa característica é composta pelas seguintes subcaracterísticas:

- Maturidade (*Maturity*): Grau em que um sistema, produto ou componente atende às necessidades de confiabilidade em operação normal.
- Disponibilidade (*Availability*): Grau em que um sistema, produto ou componente está operacional e acessível quando necessário para uso.
- Tolerância ao erro (*Fault tolerance*): Grau em que um sistema, produto ou componente opera como pretendido, apesar da presença de falhas de hardware ou software.
- Recuperabilidade (*Recoverability*). Grau em que, no caso de uma interrupção ou falha, um produto ou sistema pode recuperar os dados diretamente afetados e restabelecer o estado desejado do sistema.

Característica: Segurança (*Security*)

Essa característica representa o grau em que um produto ou sistema protege informações e dados para que pessoas ou outros produtos ou sistemas tenham o grau de acesso a dados adequado a seus tipos e níveis de autorização. Essa característica é composta pelas seguintes subcaracterísticas:

- Confidencialidade (*Confidentiality*): Grau em que um produto ou sistema garante que os dados sejam acessíveis apenas àqueles autorizados a ter acesso.
- Integridade (*Integrity*): Grau em que um sistema, produto ou componente impede o acesso não autorizado ou a modificação de programas ou dados de computador.

- Não repúdio (*Non-repudiation*): Grau em que se pode provar que as ações ou eventos ocorreram, para que os eventos ou ações não possam ser repudiados posteriormente.
- Rastreabilidade (*Accountability*): Grau em que as ações de uma entidade podem ser rastreadas exclusivamente para a entidade.
- Autenticidade (*Authenticity*): Grau em que a identidade de um sujeito ou recurso pode ser provada como a reivindicada.

Característica: Manutenibilidade (*Maintainability*)

Essa característica representa o grau de eficácia e eficiência com que um produto ou sistema pode ser modificado para melhorá-lo, corrigi-lo ou adaptá-lo às mudanças no ambiente e nos requisitos. Essa característica é composta pelas seguintes subcaracterísticas:

- Modularidade (*Modularity*): Grau em que um sistema ou programa de computador é composto de componentes discretos, de forma que uma alteração em um componente tenha um impacto mínimo em outros componentes.
- Reusabilidade (*Reusability*): Grau em que um ativo pode ser usado em mais de um sistema ou na construção de outros ativos.
- Analisabilidade (*Analysability*): Grau de eficácia e eficiência com as quais é possível avaliar o impacto em um produto ou sistema de uma alteração pretendida em uma ou mais de suas partes, ou diagnosticar um produto quanto a deficiências ou causas de falhas, ou identificar peças a serem modificadas.
- Modificabilidade (*Modifiability*). Grau em que um produto ou sistema pode ser efetivamente e eficientemente modificado sem a introdução de defeitos ou degradação da qualidade do produto existente.
- Testabilidade (*Testability*): Grau de eficácia e eficiência com o qual os critérios de teste podem ser estabelecidos para um sistema, produto ou componente e testes podem ser realizados para determinar se esses critérios foram atendidos.

Característica: Portabilidade (*Portability*):

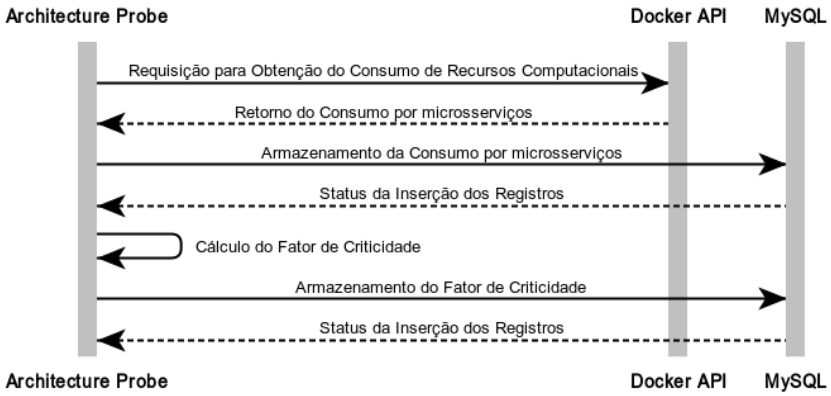
Essa característica representa o grau de eficácia e eficiência com as quais um sistema, produto ou componente pode ser transferido de um hardware, software ou

outro ambiente operacional ou de uso para outro. Essa característica é composta pelas seguintes subcaracterísticas:

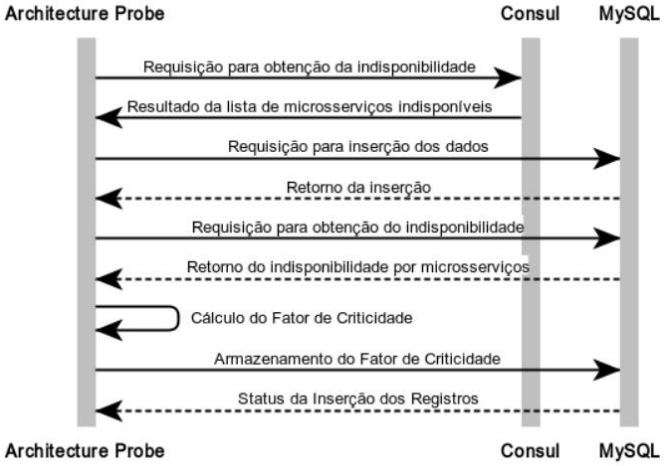
- Adaptabilidade (*Adaptability*): Grau em que um produto ou sistema pode ser efetivamente e eficientemente adaptado para hardware, software ou outros ambientes operacionais ou de uso diferentes ou em evolução.
- Instalabilidade (*Installability*): grau de eficácia e eficiência com as quais um produto ou sistema pode ser instalado e / ou desinstalado com sucesso em um ambiente especificado.
- Substituibilidade (Replaceability): grau em que um produto pode substituir outro produto de software especificado para a mesma finalidade no mesmo ambiente.

Apêndice B – Diagramas de Sequência da abordagem M2CSEA

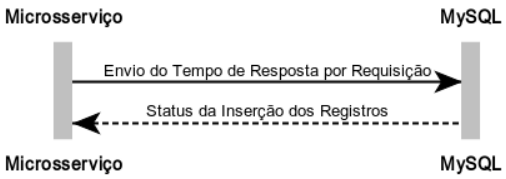
Item I - Diagrama de sequência do processo de verificação do consumo de recursos computacionais



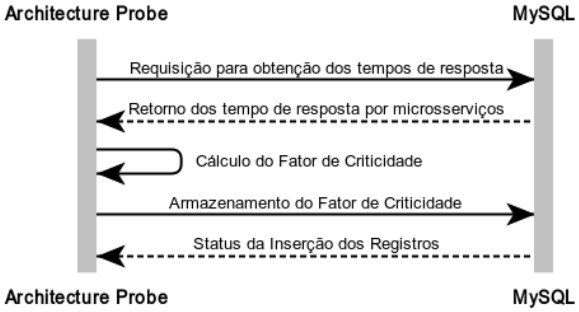
Item II – Diagrama de sequência do processo de verificação de disponibilidade



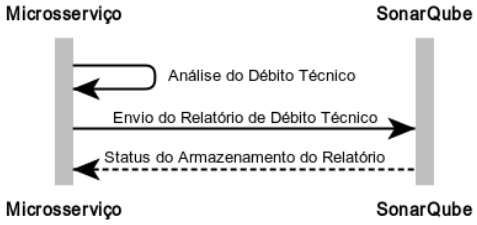
Item III – Diagrama de sequência do processo de cadastro do tempo de resposta



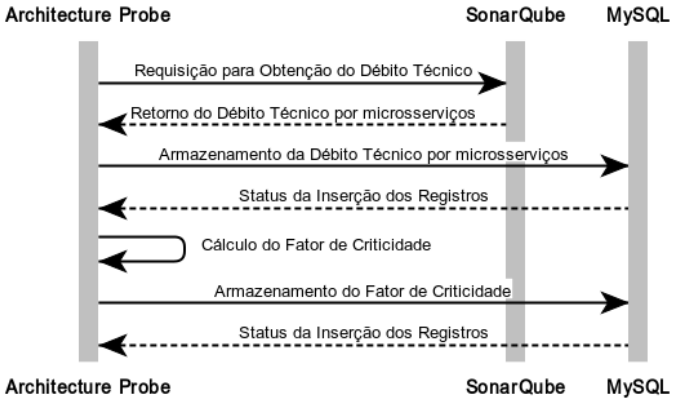
Item IV – Diagrama de sequência do processo de verificação do tempo de resposta



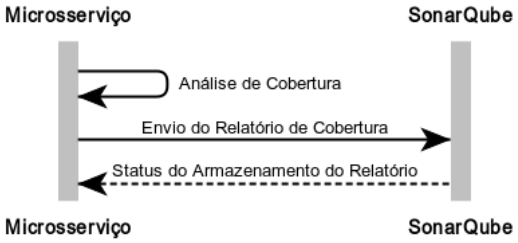
Item V – Diagrama de sequência do processo de cadastro do débito técnico no SonarQube



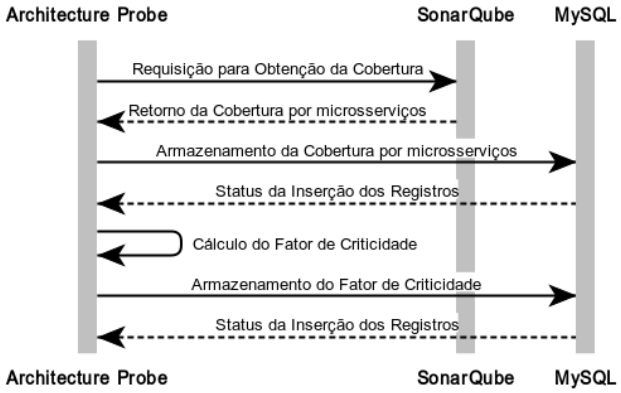
Item VI – Diagrama de sequência do processo de verificação do débito técnico



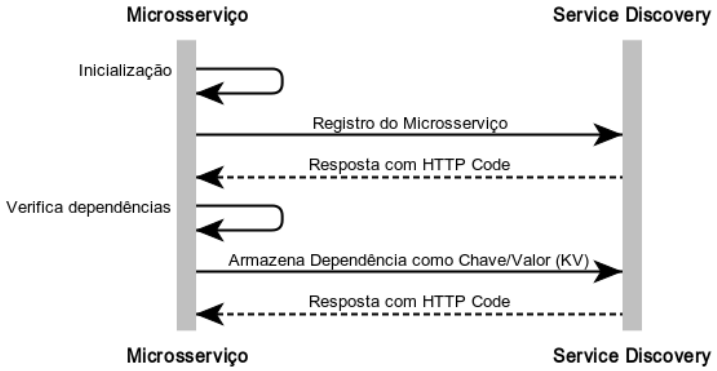
Item VII – Diagrama de sequência do processo de cadastro do relatório de cobertura no SonarQube



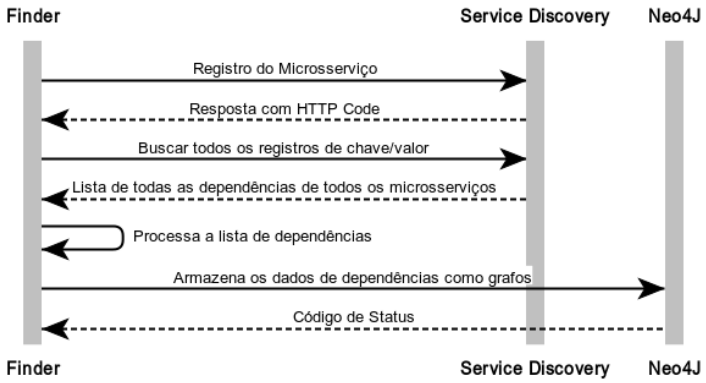
Item VIII – Diagrama de sequência do processo de verificação de cobertura de testes unitários



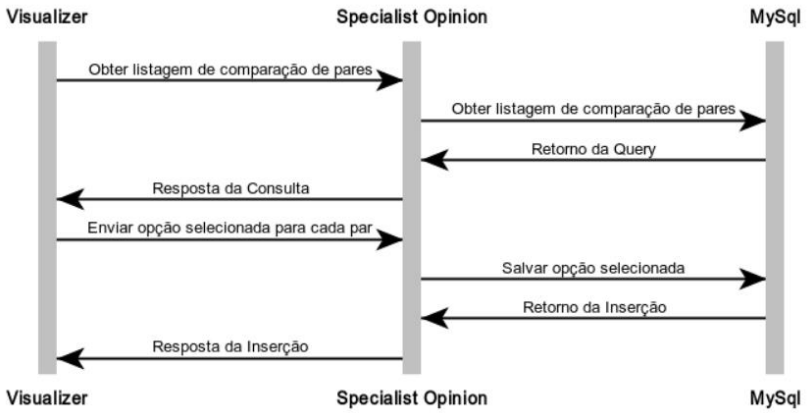
Item IX – Diagrama de Sequência do Processo de Registro de Microsserviços e Dependências



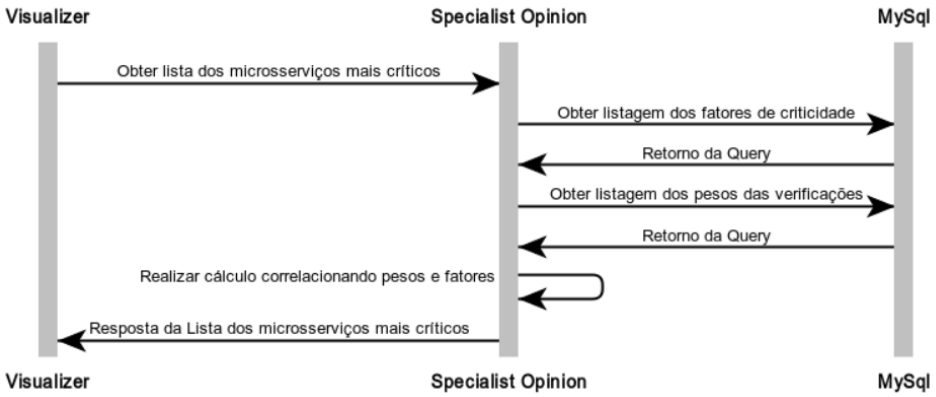
Item X – Diagrama de Sequência do Processo de Verificação de Interdependências entre microsserviços



Item XI – Diagrama de sequência do processo de obtenção da opinião do usuário

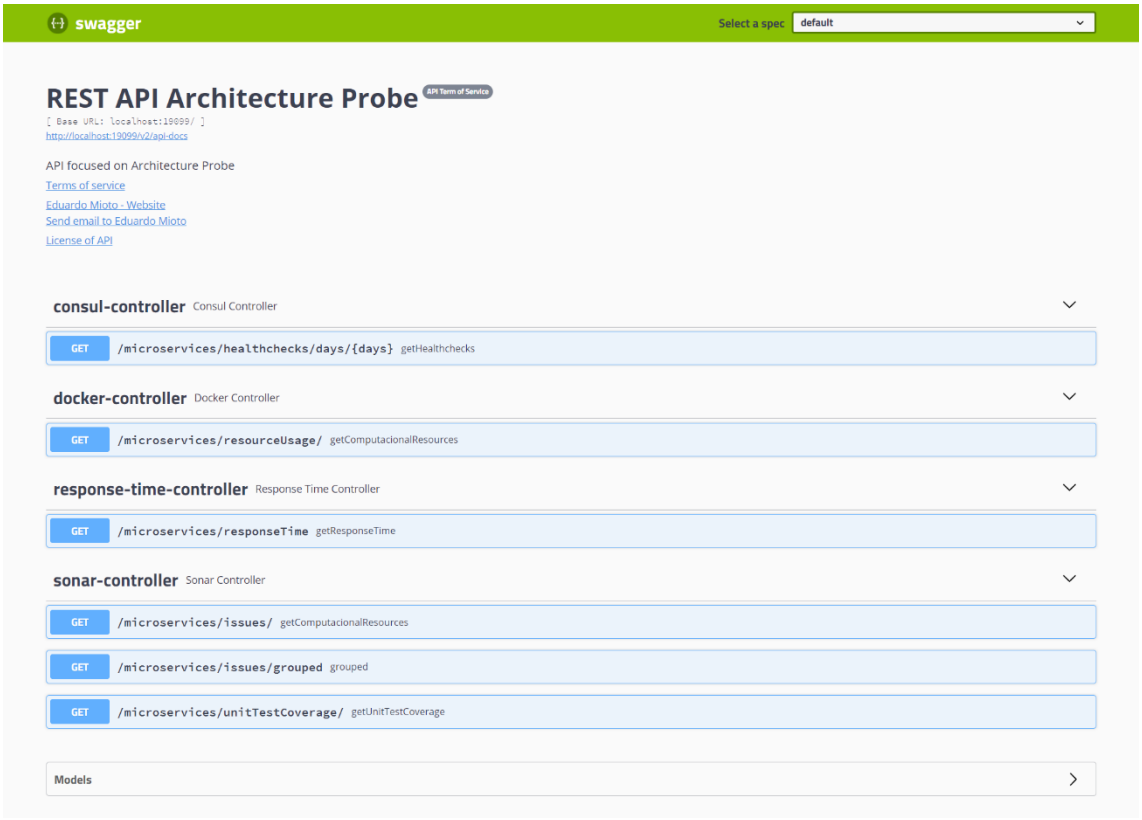


Item XII – Diagrama de sequência do processo de cálculo dos microsserviços mais críticos

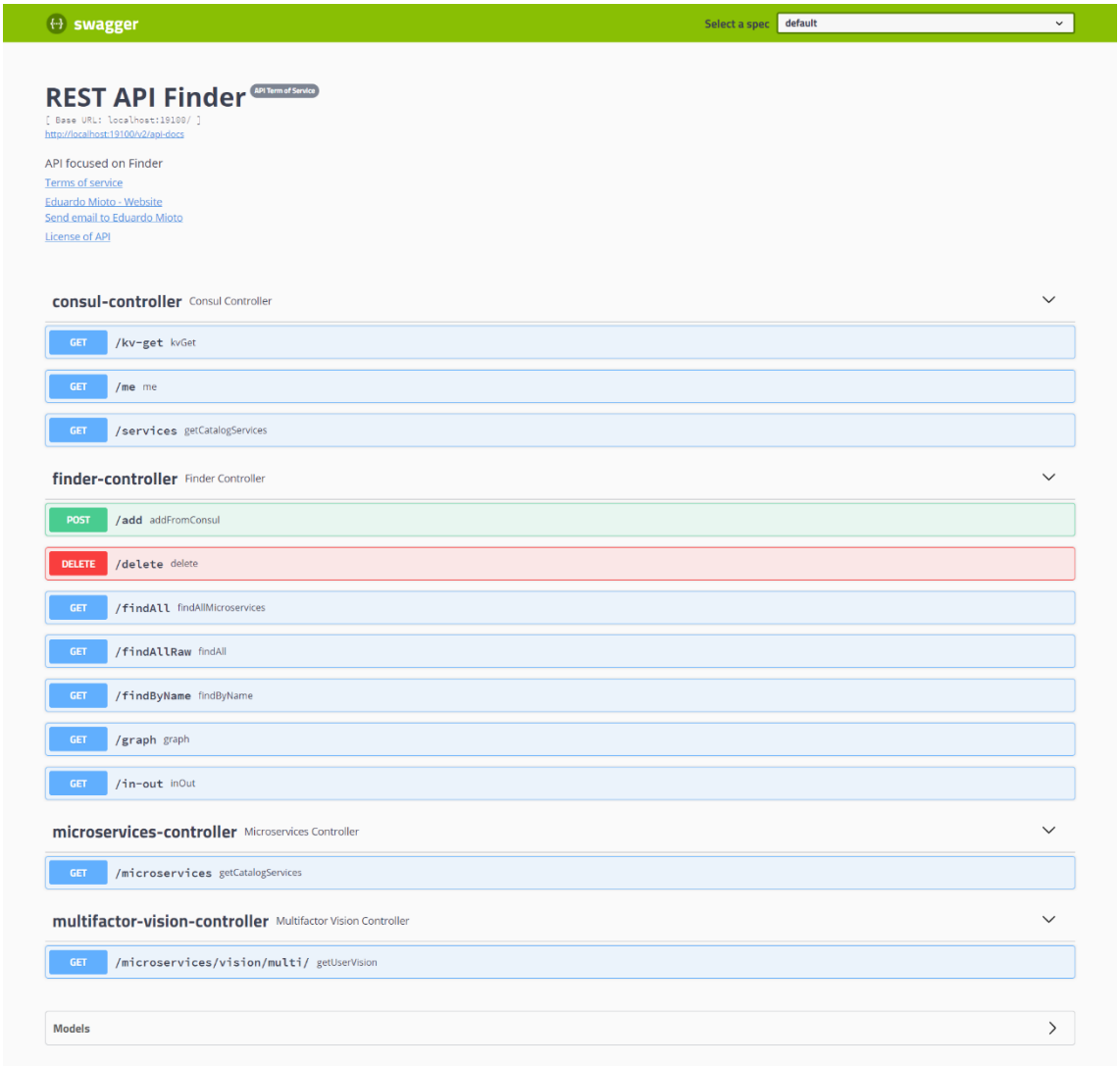


Apêndice C – Definição das APIs

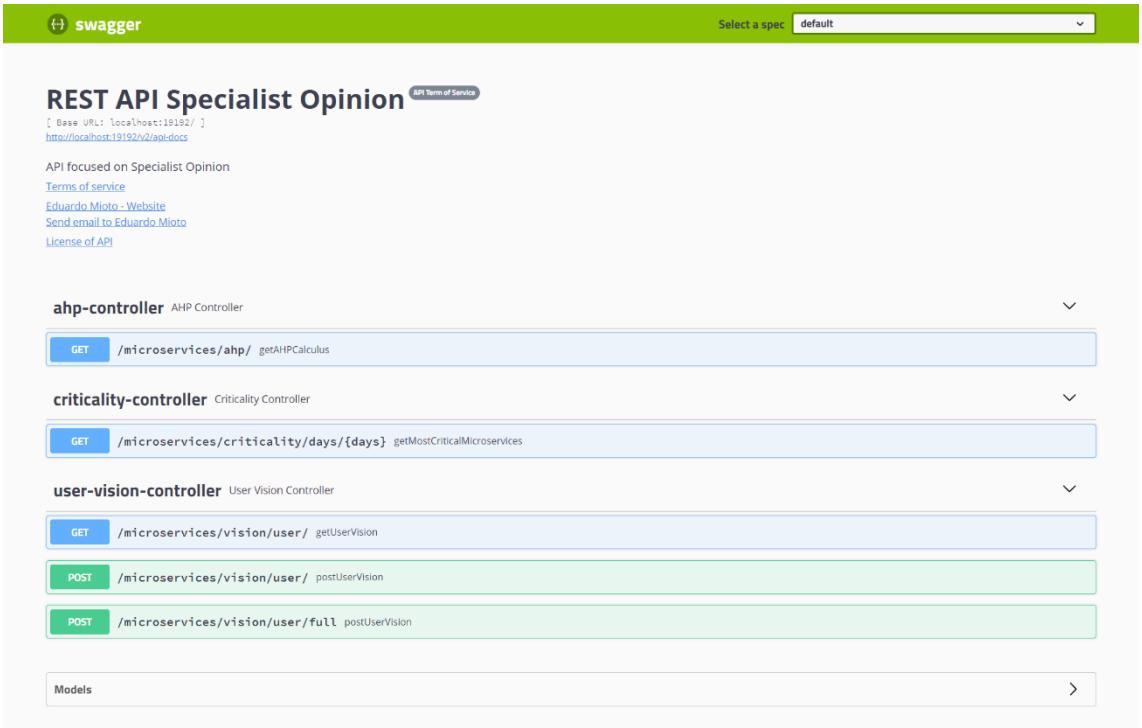
Item I – Definição das APIs de serviço Architecture Probe no padrão OpenAPI



Item II – Definição das APIs de serviço Finder no padrão OpenAPI



Item III – Definição das APIs de serviço Specialist Opinion no padrão OpenAPI



Apêndice D – Formulário do Consentimento do Survey

Introdução

Uma aplicação em nuvem geralmente deve combinar vários microsserviços interdependentes que fornecem diferentes funcionalidades. Os desafios ao lidar com configurações heterogêneas de microsserviços e recursos de datacenter na nuvem impulsionados por requisitos de desempenho heterogêneos são grandes.

A arquitetura de software orientada a microsserviços considera a separação de responsabilidades por componentes apartados, criando assim um conjunto de serviço interconectados, mas que possuem independência.

Arquiteturas estabelecidas são arquiteturas desenvolvidas, implementadas e em uso, o que requer atenção especial em relação aos requisitos não funcionais, considerando que a arquitetura está em uso e sua falha em sua estabilidade pode significar uma redução na qualidade do serviço.

Formulário de Consentimento

Este estudo tem como objetivo avaliar a relevância da sugestão de criticidade em arquiteturas estabelecidas orientadas a microsserviços.

IDADE

Eu declaro ter mais de 18 anos de idade e concordo em participar de um estudo experimental conduzido por Eduardo Fernandes Mioto de Oliveira dos Santos.

PROCEDIMENTO

Eu entendo que, uma vez o experimento tenha terminado, o questionário que respondi será estudado visando entender a eficiência dos procedimentos e serão realizadas análises nas respostas obtidas.

CONFIDENCIALIDADE

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não

terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e ensinado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

PESQUISADOR RESPONSÁVEL

Eduardo Fernandes Mioto de Oliveira dos Santos

Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

PROFESSOR RESPONSÁVEL

Profª. Cláudia M.L. Werner

Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Apêndice E – Questionário do Survey

Código do Participante: (obtido automaticamente) Data: (obtida automaticamente)

1) Você leu o formulário e termos e deseja participar?

( ) Sim, desejo participar

( ) Não, prefiro não participar

2) Nome Completo

_____

3) Atual Empresa/Instituição em que trabalha

_____

4) Formação Acadêmica:

( ) Doutorado

( ) Doutorado em Andamento

( ) Mestrado

( ) Mestrado em Andamento

( ) Especialização

( ) Especialização em Andamento

( ) Graduação

( ) Graduação em Andamento

( ) Outro: _____

5) Quanto à escolaridade, qual o ano de ingresso na universidade na mais recente formação?

_____

6) Quanto à escolaridade, qual o ano de finalização (ou previsão) na mais recente formação?

_____

7) Qual o seu atual cargo?

( ) Analista de Sistemas / Engenheiro de Software

( ) Arquiteto de Software / Arquiteto de Soluções / Arquiteto de Integração

( ) Pesquisador

( ) Professor

( ) Gestor (sem perfil técnico)

( ) Gestor (com perfil técnico)

( ) Estudante

( ) Outro: _____

8) Qual a sua atual cidade de residência?

( ) Rio de Janeiro – RJ/Brasil

( ) Niterói – RJ/Brasil

( ) Blumenau – SC/Brasil

( ) Florianópolis – SC/Brasil

( ) São Paulo – SP/Brasil

( ) San Francisco – CA/EUA

( ) San Ramon – CA/EUA

( ) Outro: _____

9) Qual seu tempo de experiência com desenvolvimento de software?

( ) Não possuo experiência

( ) 1 a 3 anos

( ) 4 a 7 anos

( ) 7 a 10 anos

( ) Mais de 10 anos

10) Qual seu tempo de experiência com desenvolvimento de software orientado a componentes (libs, .jar, .dll, etc)?

( ) Não possuo experiência

( ) 1 a 3 anos

( ) 4 a 7 anos

( ) 7 a 10 anos

( ) Mais de 10 anos

11) Qual seu tempo de experiência com desenvolvimento de software orientado a serviços (SOA)?

( ) Não possuo experiência

( ) 1 a 3 anos

( ) 4 a 7 anos

( ) 7 a 10 anos

( ) Mais de 10 anos

12) Qual seu tempo de experiência com desenvolvimento de software com Arquitetura orientada a microsserviços?

( ) Não possuo experiência

( ) 1 a 3 anos

( ) 4 a 7 anos

( ) 7 a 10 anos

( ) Mais de 10 anos

Para as respostas abaixo, considere sua última experiência com microsserviços.

Acerca dos fatores de influência para a opção pela arquitetura de microsserviços, assinale a alternativa que melhor está relacionada a sua opinião

13) Um fator de grande influência para a opção pela arquitetura de microsserviços foi a "Integração Contínua / Entrega Contínua"

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

14) Um fator de grande influência para a opção pela arquitetura de microsserviços foi o "Menor Time-to-Market¨

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

15) Um fator de grande influência para a opção pela arquitetura de microsserviços foi a "Maior Escalabilidade¨

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

16) Um fator de grande influência para a opção pela arquitetura de microsserviços foi a "Maior produtividade no desenvolvimento¨

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( )Concordo Fortemente

17) Um fator de grande influência para a opção pela arquitetura de microsserviços foi a "Maior facilidade de depuração e manutenibilidade¨

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

18) Um fator de grande influência para a opção pela arquitetura de microsserviços foi a "Capacidade para Suportar Múltiplas Linguagens na mesma Arquitetura¨

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

Para as respostas abaixo, considere sua última experiência com microsserviços

19) **Quais foram os maiores desafios na implementação deste modelo de arquitetura? Enumere os itens abaixo de 1 a 6. Os valores selecionados não podem se repetir.**

Adaptar a Cultura Corporativa à Arquitetura de Microsserviços

(1) (2) (3) (4) (5) (6)

Gestão da Escalabilidade dos Microsserviços

(1) (2) (3) (4) (5) (6)

Diagnóstico de Problemas

(1) (2) (3) (4) (5) (6)

Monitoramento dos Custos

(1) (2) (3) (4) (5) (6)

Adaptar o Arquitetura de Microsserviços ao Modelo de Negócio da Empresa

(1) (2) (3) (4) (5) (6)

Separação do Software Legado em Microsserviços

(1) (2) (3) (4) (5) (6)

Instrução - Leia a orientação abaixo

Para a próxima seção é necessário que você esteja familiarizado com os conceitos propostos pela ISO/IEC 25010:2011 acerca da avaliação de sistemas e softwares. Abaixo segue o detalhamento dos conceitos necessários para melhor contextualização acerca da norma.

Modelos de Qualidade

As propriedades mensuráveis relacionadas à qualidade de um sistema são chamadas de propriedades de qualidade, com medidas de qualidade associadas. Para chegar a medidas da característica ou subcaracterística de qualidade, a menos que a característica ou subcaracterística possa ser medida diretamente, será necessário identificar um conjunto de propriedades que juntas abranjam a característica ou subcaracterística, obtenha medidas de qualidade para cada uma e combine computacionalmente para chegar a uma medida de qualidade derivada correspondente à característica ou sub característica de qualidade.

Eficácia: precisão e integridade com as quais os usuários alcançam objetivos especificados

Eficiência: Recursos gastos em relação à precisão e integridade com que os usuários alcançam objetivos

Satisfação: grau em que as necessidades do usuário são satisfeitas quando um produto ou sistema é usado em um contexto de uso especificado

Menor Risco: grau em que um produto ou sistema mitiga o risco potencial ao status econômico, vida humana, saúde ou meio ambiente

Cobertura de Contexto: grau em que um produto ou sistema pode ser usado com eficácia, eficiência, isenção de riscos e satisfação nos contextos especificados de uso e em contextos além daqueles inicialmente identificados explicitamente

Modelo de Qualidade de Software em Uso

Considere o modelo de qualidade de software em uso exposto acima, e marque a resposta segundo sua última experiência com microsserviços.

20) Ordene os atributos de qualidade de software quanto a sua relevância para a escolha da arquitetura orientada a microsserviços. Para a atividade abaixo, ordene os atributos do mais mais relevante (1) para o menos relevante (6).

Eficácia

(1) (2) (3) (4) (5)

Eficiência

(1) (2) (3) (4) (5)

Satisfação

(1) (2) (3) (4) (5)

Menor Risco

(1) (2) (3) (4) (5)

Cobertura de Contexto

(1) (2) (3) (4) (5)

Para a próxima seção é necessário que você esteja familiarizado com os conceitos propostos pela ISO/IEC 25010:2011 acerca da avaliação de sistemas e softwares. Abaixo segue o detalhamento dos conceitos necessários para melhor contextualização acerca da norma.

Qualidade de Produto

O modelo de qualidade do produto categoriza as propriedades de qualidade do sistema / software em oito características: adequação funcional, eficiência de desempenho, compatibilidade, usabilidade, confiabilidade, segurança, manutenção e portabilidade. Cada característica é composta por um conjunto de subcaracterísticas relacionadas.

O modelo de qualidade do produto pode ser aplicado apenas a um produto de software ou a um sistema de computador que inclui software, pois a maioria das sub características é relevante para o software e os sistemas.

O modelo de qualidade do produto se concentra no sistema de computador de destino que inclui o produto de software de destino e o modelo de qualidade em uso se concentra em todo o sistema humano-computador que inclui o sistema de computador de destino e o produto de software de destino. O sistema de computador de destino também inclui hardware de computador, produtos de software não-alvo, dados não-alvo e dados de destino, que são o assunto do modelo de qualidade dos dados.

O sistema de computador de destino está incluído em um sistema de informação que também pode incluir um ou mais sistemas de computadores e sistemas de comunicação, como uma rede de área local e a Internet. O sistema de informações está dentro de um sistema humano-computador mais amplo (como um sistema corporativo, sistema incorporado ou sistema de controle em larga escala) e pode incluir usuários e o ambiente de uso técnico e físico. O local em que o limite do sistema é considerado depende do escopo dos requisitos ou da avaliação e de quem são os usuários.

Adequação Funcional: grau em que um produto ou sistema fornece funções que atendem às necessidades declaradas e implícitas quando usadas em condições especificadas

Eficiência de Desempenho: desempenho relativo à quantidade de recursos utilizados sob condições estabelecidas

Compatibilidade; Grau em que um produto, sistema ou componente pode trocar informações com outros produtos, sistemas ou componentes e / ou executar as funções necessárias, enquanto compartilha o mesmo ambiente de hardware ou software

Usabilidade: grau em que um produto ou sistema pode ser usado por usuários especificados para atingir objetivos especificados com eficácia, eficiência e satisfação em um contexto especificado de uso

Resiliência: grau em que um sistema, produto ou componente executa funções especificadas em condições especificadas por um período especificado

Segurança: grau em que um produto ou sistema protege informações e dados para que pessoas ou outros produtos ou sistemas tenham o grau de acesso a dados adequado a seus tipos e níveis de autorização

Manutenibilidade: grau de eficácia e eficiência com as quais um produto ou sistema pode ser modificado pelos mantenedores pretendidos

Portabilidade: grau de eficácia e eficiência com as quais um sistema, produto ou componente pode ser transferido de um hardware, software ou outro ambiente operacional ou de uso para outro

Para a atividade abaixo, ordene os atributos do mais mais relevante (1) para o menos relevante (8).

21) Ordene os atributo de qualidade de produto de software quanto a sua relevância para a escolha da arquitetura orientada a microsserviços?

Adequação Funcional

(1) (2) (3) (4) (5) (6) (7) (8)

Eficiência de Desempenho

(1) (2) (3) (4) (5) (6) (7) (8)

Compatibilidade

(1) (2) (3) (4) (5) (6) (7) (8)

Usabilidade

(1) (2) (3) (4) (5) (6) (7) (8)

Resiliência

(1) (2) (3) (4) (5) (6) (7) (8)

Segurança

(1) (2) (3) (4) (5) (6) (7) (8)

Manutenibilidade

(1) (2) (3) (4) (5) (6) (7) (8)

Portabilidade

(1) (2) (3) (4) (5) (6) (7) (8)

Para as respostas abaixo, considere sua última experiência com microsserviços

Critérios para obtenção dos microsserviços mais críticos em arquiteturas de software em uso

22) Quanto à criticidade, não havia uma forma de elencar os microsserviços mais críticos

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

23) Quanto à criticidade, os microsserviços eram ordenados por "Relevância para o Negócio"

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

24) Quanto à criticidade, os microsserviços eram ordenados pelo "Número de integrações com outros microsserviços"

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

25) Quanto à criticidade, os microsserviços eram ordenados pelo "Custo de Manutenção"

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

26) Quanto à criticidade, os microsserviços eram ordenados pela "Opinião de Especialistas"

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

27) Quanto à criticidade, os microsserviços eram ordenados por "Tempo de Resposta"

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

28) Quanto à criticidade, os microsserviços eram ordenados por "Utilização de Recursos Computacionais"

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

29) Quanto à criticidade, os microsserviços eram ordenados por "Cobertura de Testes Unitários"

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

Aplicabilidade da Metodologia Proposta

Para as respostas abaixo, considere a existência de uma ferramenta que baseado na opinião de diversos interessados/stakeholders, sugira o microsserviço mais crítico baseado nos critérios abaixo expostos.

- Tempo de Resposta
- Opinião de Especialistas
- Custo de Manutenção (calculado por débito técnico)
- Cobertura de Testes Unitários
- Número de integrações com outros microsserviços
- Relevância para o Negócio
- Utilização de Recursos Computacionais

30) A utilização de uma ferramenta como a proposta acima, poderia auxiliar na redução no custo associado a manutenção em um ou mais projetos que trabalhei

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

31) A utilização de uma ferramenta como a proposta acima, poderia auxiliar na geração do aumento da resiliência na arquitetura em um ou mais projetos que trabalhei.

( ) Discordo Fortemente
( ) Discordo
( ) Não concordo e nem discordo
( ) Concordo
( ) Concordo Fortemente

32) A utilização de uma ferramenta como a proposta, poderia auxiliar na redução do número de defeitos no ambiente produtivo

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

Apêndice F – Formulário de Consentimento do Estudo de Caso

Introdução à Abordagem

O processo contínuo de evolução da arquitetura de software com um consequente aumento de tamanho e complexidade torna o *design* do sistema extremamente importante para a produção de softwares de qualidade. Assim, o papel da arquitetura de software é significativamente importante no desenvolvimento de software, pois serve como um plano de avaliação e implementação, tornando a escolha da arquitetura correta um problema crítico no campo da engenharia de software.

Após a implementação de uma arquitetura orientada a microsserviços, o processo de manutenção tende a ser complexo, considerando os múltiplos serviços a serem observados. A necessidade de conhecer os microsserviços mais críticos advém dessa complexidade de manutenção. Diferentes partes interessadas, como arquitetos de sistemas, gerentes de projeto, engenheiros de software, gerentes de infraestrutura, analistas de infraestrutura, entre outros, precisam conhecer esses serviços para promover maior estabilidade arquitetural ou, eventualmente, a evolução dos serviços nessa arquitetura.

A verificação de criticidade dos microsserviços na arquitetura estabelecida pode guiar os particionadores de engenharia de software para um processo de tomada de decisão mais fundamentado. Este estudo tem como objetivo estudar a avaliação de criticidade de múltiplos critérios em arquiteturas orientadas a microsserviços, visando a proposta de um método de avaliação de arquiteturas estabelecidas que suportará decisões arquitetônicas baseadas em múltiplos critérios.

Formulário de Consentimento

Este estudo tem como objetivo avaliar a abordagem M2CSEA (*Multicriteria Microservice Criticality Suggestion for Established Architectures*), considerando seu apoio à análise e verificação dos microsserviços mais críticos em arquiteturas estabelecidas para sistemas em execução.

IDADE

Eu declaro ter mais de 18 anos de idade e concordo em participar de um estudo experimental conduzido por Eduardo Fernandes Mioto de Oliveira dos Santos.

PROCEDIMENTO

Eu entendo que, uma vez o experimento tenha terminado, o questionário que respondi será estudado visando entender a eficiência dos procedimentos e serão realizadas análises nas respostas obtidas.

CONFIDENCIALIDADE

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e ensinado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

PESQUISADOR RESPONSÁVEL

Eduardo Fernandes Mioto de Oliveira dos Santos

Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

PROFESSOR RESPONSÁVEL

Profª. Cláudia M.L. Werner

Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Apêndice G – Questionário do Estudo de Caso

D1 - Declaro que possuo conhecimento que nenhuma das informações contidas neste experimento me identificarão e que devo manter sigilo das perguntas e protocolo utilizado durante este experimento.

( ) Estou ciente e desejo participar.

( ) Não desejo participar.

D2 - Possuo graduação completa?

( ) Sim

( ) Não

D3 - Possuo mais de 4 anos em desenvolvimento de software?

( ) Sim

( ) Não

D4 - Possuo mais de 2 anos com desenvolvimento de software com microsserviços?

( ) Sim

( ) Não

Seção 1 - Para as respostas abaixo, considere sua experiência durante a execução das tarefas solicitadas neste experimento.

S1_Q1 - Localize o microsserviço mais crítico segundo o nível de dependência entre os microsserviços

Microsserviço:

Tempo:

S1_Q2- Localize o microsserviço mais crítico segundo o tempo de resposta

Microsserviço:

Tempo:

S1_Q3 - Localize o microsserviço mais crítico segundo os débitos técnicos

Microsserviço:

Tempo:

S1_Q4 - Localize o microsserviço mais crítico segundo a cobertura de testes unitários

Microsserviço:

Tempo:

S1_Q5 - Localize o microsserviço mais crítico segundo o consumo de recursos computacionais

Microsserviço:

Tempo:

S1_Q6 - Localize o microsserviço mais crítico segundo a disponibilidade

Microsserviço:

Tempo:

Seção 2 - Para as respostas abaixo, localize os microsserviços mais críticos após a intervenção.

S2_Q1 - Localize o microsserviço mais crítico segundo o nível de dependência entre os microsserviços

Microsserviço:

Tempo:

S2_Q2- Localize o microsserviço mais crítico segundo o tempo de resposta

Microsserviço:

Tempo:

S2_Q3 - Localize o microsserviço mais crítico segundo os débitos técnicos

Microsserviço:

Tempo:

S2_Q4 - Localize o microsserviço mais crítico segundo a cobertura de testes unitários

Microsserviço:

Tempo:

S2_Q5 - Localize o microsserviço mais crítico segundo o consumo de recursos computacionais

Microsserviço:

Tempo:

S2_Q6 - Localize o microsserviço mais crítico segundo a disponibilidade

Microsserviço:

Tempo:

Seção 3 - Para as respostas abaixo, encontre os microsserviços mais críticos com base em vários critérios, considerando a escolha atual das preferências do usuário para as verificações mais importantes. (Apenas para grupos de tratamento **)**

S3_Q1 - Considerando a atual formação de atributos de qualidade, qual é o microsserviço mais crítico dessa arquitetura?

Microsserviço:

Tempo:

S3_Q2 - Considerando a atual formação de atributos de qualidade, qual é o microsserviço mais crítico dessa arquitetura em termos de tempo de resposta?

Microsserviço:

Tempo:

S3_Q3 - Considerando a atual formação de atributos de qualidade, qual é o microsserviço mais crítico dessa arquitetura em termos de débito técnico?

Microsserviço:

Tempo:

S3_Q4 - Considerando a atual formação de atributos de qualidade, qual é o microsserviço mais crítico dessa arquitetura em termos de cobertura de teste de unidade?

Microsserviço:

Tempo:

S3_Q5 - Considerando a formação atual de atributos de qualidade, qual é o microsserviço mais crítico dessa arquitetura em termos de uso de recursos computacionais?

Microsserviço:

Tempo:

S3_Q6 - Considerando a atual formação de atributos de qualidade, qual é o microsserviço mais crítico dessa arquitetura em termos de disponibilidade?

Microsserviço:

Tempo:

Seção 4 - Para obter respostas abaixo, considere sua experiência com a ferramenta M2CSEA (Apenas para grupos de tratamento **)**

S4_Q1 - Em relação à assertividade, a ferramenta M2CSEA tem a capacidade de orientar uma escolha assertiva de evolução ou manutenção em uma arquitetura estabelecida com base na sugestão do microsserviço mais crítico.

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

S4_Q2- Em relação à eficiência, a ferramenta M2CSEA possui baixo tempo de resposta ao sugerir o microsserviço mais crítico.

*** Considere o tempo baixo como 2 segundos ou menos.**

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

S4_Q3 - A ferramenta M2CSEA poderia auxiliar na tomada de decisões relacionadas à manutenção ou evolução de microsserviços.

( ) Discordo Fortemente

( ) Discordo

( ) Não concordo e nem discordo

( ) Concordo

( ) Concordo Fortemente

Seção 5 - Para as respostas abaixo, considere sua experiência com desenvolvimento de software. (Apenas para grupos de tratamento **)**

S5_Q1- Qual ferramenta você usa ou usou para verificar todas as verificações utilizadas?

*** Para esta pergunta, considere uma ferramenta que cubra todas as verificações e não um kit de ferramentas.**

S5_Q2- Em relação à assertividade, como você compararia a ferramenta M2CSEA à que você usa ou usou?

( ) A ferramenta M2CSEA tem menor assertividade em comparação com as outras ferramentas utilizadas.

( ) A ferramenta M2CSEA tem maior assertividade em comparação com as outras ferramentas utilizadas.

( ) A ferramenta M2CSEA tem a mesma assertividade em comparação com as outras ferramentas usadas.

( ) Eu nunca usei outra ferramenta semelhante.

S5_Q3- Em relação à eficiência, como você compara a ferramenta M2CSEA com a que você usa ou usa?

*** Considere a eficiência como tempo de resposta**

( ) A ferramenta M2CSEA tem menos eficiência em comparação com as outras ferramentas utilizadas.

( ) A ferramenta M2CSEA possui maior eficiência em comparação com as outras ferramentas utilizadas.

( ) A ferramenta M2CSEA tem a mesma eficiência em comparação com as outras ferramentas utilizadas.

( ) Nunca usei outra ferramenta semelhante

Apêndice H – Informações Complementares para Grupo Controle do Estudo de Caso

Acessos aos Sistemas de Apoio

Função	Sistema	Acesso
Verificar Interdependências	Neo4j	http://localhost:7474/browser
Verificar Débito Técnico	SonarQube	http://localhost:9000
Verificar Cobertura de Testes Unitários – Opção 1	SonarQube	http://localhost:9000
Verificar Disponibilidade – Opção 1	Consul	http://localhost:8500
Verificar Consumo de Recursos Computacionais	Docker stats	Conexão Remota via SSH
Verificar Tempo de Resposta	MySQL	MySQL Workbench
Verificar Cobertura de Testes Unitários – Opção 2	MySQL	MySQL Workbench
Verificar Disponibilidade – Opção 2	MySQL	MySQL Workbench

Comando para Console Linux

Comando	Descrição
docker stats	Exibe os containeres em execução e o consumo de recursos computacionais para cada um destes

Queries Base para Verificação com banco MySQL

Tempo de Resposta

```
use `m2csea-metrics`;
SELECT `microservice`, `response_time`, `dt_transaction`
FROM `response_time`;
```

Cobertura de Testes Unitários

```
use `m2csea-metrics`;
SELECT `microservice`, `coverage`
FROM `unit_test_coverage`;
```

Disponibilidade

```
use `m2csea-metrics`;
SELECT `microservice`, `status`, `dt_transaction`
FROM `availability`;
```

Exemplo de Query para buscas por data

Availability

```
SELECT * FROM FOO
WHERE MY_DATE_FIELD >= NOW() - INTERVAL 1 DAY
```

Apêndice I – Resultados Individuais Obtidos no Estudo de Caso

Neste apêndice, são apresentados os resultados individuais de cada um dos participantes do estudo de caso. Os resultados expostos são relacionados às perguntas do questionário do estudo de caso, presente no Apêndice G. Todos os resultados contidos nas tabelas abaixo estão descritos em segundos. Os resultados nos quais há uma marcação como "N/A" estão relacionados a uma questão não aplicável ao grupo em questão por este ser um grupo de controle, não tendo utilizado a ferramenta baseada no modelo M2CSEA durante a execução do estudo de caso.

Tabela A - Resultados da Sessão 1 Questões 1 a 6

Participante	Grupo	S1_Q1	S1_Q2	S1_Q3	S1_Q4	S1_Q5	S1_Q6
1	A	13,64s	5,11s	3,85s	3,22s	3,73s	3,39s
2	C	6,9s	4,13s	3,49s	3,5s	5,39s	2,64s
3	A	7,69s	5,42s	4,08s	4,08s	6,28s	3,02s
4	D	22,12s	65,47s	66,41s	21,09s	53,27s	253,26s
5	B	32,75s	66,7s	64,41s	27,9s	54,17s	70,37s
6	C	6,58s	4,17s	1,83s	1,92s	2,7s	2,57s
7	D	8,33s	136,46s	43,82s	38,51s	47s	85,41s
8	B	19,8s	36,39s	77,96s	17,55s	59,09s	188,98s
9	A	34,3s	12,21s	5,24s	4,62s	8,23s	14s
10	B	16,75s	20,61s	34,76s	33,84s	41,55s	176,96s
11	C	7,43s	4,54s	1,83s	1,91s	2,63s	2,23s
12	D	10,54s	156,25s	53,65s	68,73s	67s	95,49s

Tabela B - Resultados da Sessão 2 Questões 1 a 6

Participante	Grupo	S2_Q1	S2_Q2	S2_Q3	S2_Q4	S2_Q5	S2_Q6
1	A	4,15s	3,61s	3,27s	3,42s	3,46s	2,7s
2	C	4,66s	5,88s	2,1s	2,18s	3,96s	2,41s
3	A	5,11s	4,67s	2,91s	3,12s	5,06s	3,42s
4	D	13,51s	68,56s	7,25s	19,05s	21,39s	88,05s
5	B	48,33s	100,09s	21,14s	58,98s	11,81s	255,42s
6	C	3,37s	2,45s	1,87s	1,7s	2,27s	1,95s
7	D	18,46s	15,18s	7,46s	65,58s	7,99s	41,13s
8	B	23,87s	66,61s	9,69s	26,36s	50,7s	71,25s
9	A	15,63s	29,16s	5,93s	4,77s	7,43s	13,37s
10	B	13,63s	32,39s	9,89s	39,45s	21s	80,83s
11	C	3,35s	2,13s	1,92s	1,62s	2,51s	1,91s
12	D	48,13s	20,18s	15,41s	75,51s	27,32s	51,24s

Tabela C - Resultados da Sessão 3 Questões 1 a 6

Participante	Grupo	S3Q1	S3_Q2	S3_Q3	S3_Q4	S3_Q5	S3_Q6
1	A	3,94s	4	15	8,93s	2,89s	2,01s
2	C	3,67s	4,04s	3,59s	3,22s	2,88s	1,28s
3	A	2,11s	2,92s	3,96s	5,59s	3,83s	3,88s
4	D	N/A	N/A	N/A	N/A	N/A	N/A
5	B	N/A	N/A	N/A	N/A	N/A	N/A
6	C	2,87s	13,76s	1,66s	1,73s	2,45s	3,24s
7	D	N/A	N/A	N/A	N/A	N/A	N/A
8	B	N/A	N/A	N/A	N/A	N/A	N/A
9	A	13,19s	1,62s	2,64s	0,65s	2s	6,9s
10	B	N/A	N/A	N/A	N/A	N/A	N/A
11	C	4,42s	7,32s	3,42s	2,25s	4,15s	3,63s
12	D	N/A	N/A	N/A	N/A	N/A	N/A

Tabela D - Resultados da Sessão 4 Questões 1 a 3

Participante	Grupo	S4_Q1	S4_Q2	S4_Q3
1	A	Concordo Fortemente	Concordo Fortemente	Concordo Fortemente
2	C	Concordo Fortemente	Concordo Fortemente	Concordo Fortemente
3	A	Concordo Fortemente	Concordo	Concordo Fortemente
4	D	N/A	N/A	N/A
5	B	N/A	N/A	N/A
6	C	Concordo	Concordo Fortemente	Concordo Fortemente
7	D	N/A	N/A	N/A
8	B	N/A	N/A	N/A
9	A	Concordo Fortemente	Concordo Fortemente	Concordo Fortemente
10	B	N/A	N/A	N/A
11	C	Concordo Fortemente	Concordo Fortemente	Concordo Fortemente
12	D	N/A	N/A	N/A

Tabela E - Resultados da Sessão 5 Questão 1

Participante	Grupo	S5_Q1
1	A	Não. Pois as ferramentas que utilizei (JMeter para tempo de resposta, SonarQube para testes unitários e Fortify para débito técnico) verificavam cada serviço em separado.
2	C	Não. Utilizei o Zabbix apenas para gestão de quedas de servidores e monitoramento de serviços e jobs de processamento em lote
3	A	Não. Na ocasião foram utilizadas ferramentas do próprio provedor de nuvem (ex AWS), mas que não contemplavam todas as verificações
4	D	Não. Separadamente já utilizei o SonarQube (para acompanhamento de débito técnico e cobertura de testes unitários), Prometheus (para métricas de consumo de recursos de hardware e software) e ELK (para métricas de consumo de hardware e software via logs)
5	B	Não. Somente o SonarQube (para débito técnico e cobertura de testes unitários)
6	C	Não. Nunca utilizei outras ferramentas similares.
7	D	Sim, Elasticsearch + Kibana. Através da coleta de logs centralizando-os em um servidor onde pudessem ser extraídas métricas através de gráficos, planilhas e etc. O servidor de integração contínua obtinha informações de cobertura de código e as enviava ao Elasticsearch.
8	B	Não. Utilizei somente o Findbugs e eClemma (para cobertura de testes unitários e qualidade de código) e JMeter (para testes de stress e escalabilidade).
9	A	Sim. AWS X-Ray
10	B	Não. SonarQube para débito técnico, Neo4J para interdependência, ELK para tempo de resposta baseado em log
11	C	Não
12	D	Não

Tabela F - Resultados da Sessão 5 Questões 2 e 3

Participante	Grupo	S5_Q2	S5_Q3
1	A	Nunca utilizei outra ferramenta similar	Nunca utilizei outra ferramenta similar
2	C	Nunca utilizei outra ferramenta similar	Nunca utilizei outra ferramenta similar
3	A	Nunca utilizei outra ferramenta similar	Nunca utilizei outra ferramenta similar
4	D	N/A	N/A
5	B	N/A	N/A
6	C	Nunca utilizei outra ferramenta similar	Nunca utilizei outra ferramenta similar
7	D	N/A	N/A
8	B	N/A	N/A
9	A	Possui menor assertividade	Possui menor assertividade
10	B	N/A	N/A
11	C	Nunca utilizei outra ferramenta similar	Nunca utilizei outra ferramenta similar
12	D	N/A	N/A

Apêndice J – Resultados Individuais Agrupados no Estudo de Caso

Tabela A - Resultados agrupados da Sessão 1 e 2 Questões 1 a 6

Participante	Grupo	S1_S2_Q1	S1_S2_Q2	S1_S2_Q3	S1_S2_Q4	S1_S2_Q5	S1_S2_Q6
1	A	8,895s	4,36s	3,56s	3,32s	3,595s	3,045s
2	A	24,965s	20,685s	5,585s	4,695s	7,83s	13,685s
3	C	5,39s	3,335s	1,875s	1,765s	2,57s	2,07s
4	B	21,835s	51,5s	43,825s	21,955s	54,895s	130,115s
5	C	4,975s	3,31s	1,85s	1,81s	2,485s	2,26s
6	D	17,815s	67,015s	36,83s	20,07s	37,33s	170,655s
7	A	6,4s	5,045s	3,495s	3,6s	5,67s	3,22s
8	B	15,19s	26,5s	22,325s	36,645s	31,275s	128,895s
9	D	13,395s	75,82s	25,64s	52,045s	27,495s	63,27s
10	B	40,54s	83,395s	42,775s	43,44s	32,99s	162,895s
11	C	5,78s	5,005s	2,795s	2,84s	4,675s	2,525s
12	D	29,335s	88,215s	34,53s	72,12s	47,16s	73,365s

Tabela B - Resultados agrupados da Sessão 3 Questões 1 a 3

Participante	Grupo	S3_Q1_Q2_Q3
1	A	6,128s
2	A	4,5s
3	C	4,198s
4	B	Não Aplicável
5	C	2,87s
6	D	Não Aplicável
7	A	3,715s
8	B	Não Aplicável
9	D	Não Aplicável
10	B	Não Aplicável
11	C	3,67s
12	D	Não Aplicável

Tabela C - Resultados da assertividade por participante considerando o agrupamento de todas as questões

Participante	Grupo	Erros	Percentual	Tipo de Grupo
1	A	0	100	Tratamento
2	A	0	100	Tratamento
3	C	0	100	Tratamento
4	B	0	100	Controle
5	C	0	100	Controle
6	D	0	100	Controle
7	A	0	100	Tratamento
8	B	0	100	Tratamento
9	D	1	93,34	Tratamento
10	B	1	93,34	Controle
11	C	0	100	Controle
12	D	0	100	Controle

Apêndice K – Análise Gráfica para Resultados Obtidos no Estudo de Caso

Os dados dispostos neste apêndice são referentes aos resultados obtidos no estudo de caso apresentado no Capítulo 6 – Avaliação da Proposta. Estes resultados foram agrupados conforme apresentado no Apêndice J - Resultados Individuais Agrupados no Estudo de Caso, e a partir destes dados serão apresentados os gráficos de boxplots.

Tabela A - Estatística Descritiva para Questões de S1Q1 a S1Q3

Questão	**S1Q1**				**S1Q2**				**S1Q3**			
Grupo	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Válidos	3	3	3	3	3	3	3	3	3	3	3	3
Inválidos	0	0	0	0	0	0	0	0	0	0	0	0
Média	13.420	25.855	5.382	20.182	10.030	53.798	3.883	77.017	4.213	36.308	2.173	32.333
Desvio Padrão	10.076	13.144	0.403	8.229	9.234	28.517	0.971	10.651	1.188	12.121	0.539	5.910
Mínimo	6.400	15.190	4.975	13.395	4.360	26.500	3.310	67.015	3.495	22.325	1.850	25.640
Máximo	24.965	40.540	5.780	29.335	20.685	83.395	5.005	88.215	5.585	43.825	2.795	36.830
25º percentil	7.647	18.512	5.182	15.605	4.703	39.000	3.322	71.417	3.527	32.550	1.863	30.085
50º percentil	8.895	21.835	5.390	17.815	5.045	51.500	3.335	75.820	3.560	42.775	1.875	34.530
75º percentil	16.930	31.188	5.585	23.575	12.865	67.447	4.170	82.017	4.572	43.300	2.335	35.680

Tabela B - Estatística Descritiva para Questões de S1Q4 a S1Q6

Questão	**S1Q4**				**S1Q5**				**S1Q6**			
Grupo	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Válidos	3	3	3	3	3	3	3	3	3	3	3	3
Inválidos	0	0	0	0	0	0	0	0	0	0	0	0
Média	3.872	34.013	2.138	48.078	5.698	39.720	3.243	37.328	6.650	140.635	2.285	102.430
Desvio Padrão	0.727	10.982	0.608	26.251	2.118	13.170	1.241	9.833	6.093	19.287	0.229	59.300
Mínimo	3.320	21.955	1.765	20.070	3.595	31.275	2.485	27.495	3.045	128.895	2.070	63.270
Máximo	4.695	43.440	2.840	72.120	7.830	54.895	4.675	47.160	13.685	162.895	2.525	170.655
25º percentil	3.460	29.300	1.788	36.057	4.633	32.133	2.527	32.413	3.132	129.505	2.165	68.317
50º percentil	3.600	36.645	1.810	52.045	5.670	32.990	2.570	37.330	3.220	130.115	2.260	73.365
75º percentil	4.147	40.042	2.325	62.083	6.750	43.943	3.623	42.245	8.453	146.505	2.393	122.010

Tabela C - Estatística Descritiva para Questão de S3Q1

Questão	**S3Q1**	
Grupo	**A**	**C**
Válidos	3	3
Inválidos	0	0
Média	4.783	3.580
Desvio Padrão	1.230	0.670
Mínimo	3.720	2.870
Máximo	6.130	4.200
25º percentil	4.110	3.270
50º percentil	4.500	3.670
75º percentil	5.315	3.935

Imagem A - Gráficos de Distribuição por grupo para Questão S1Q1

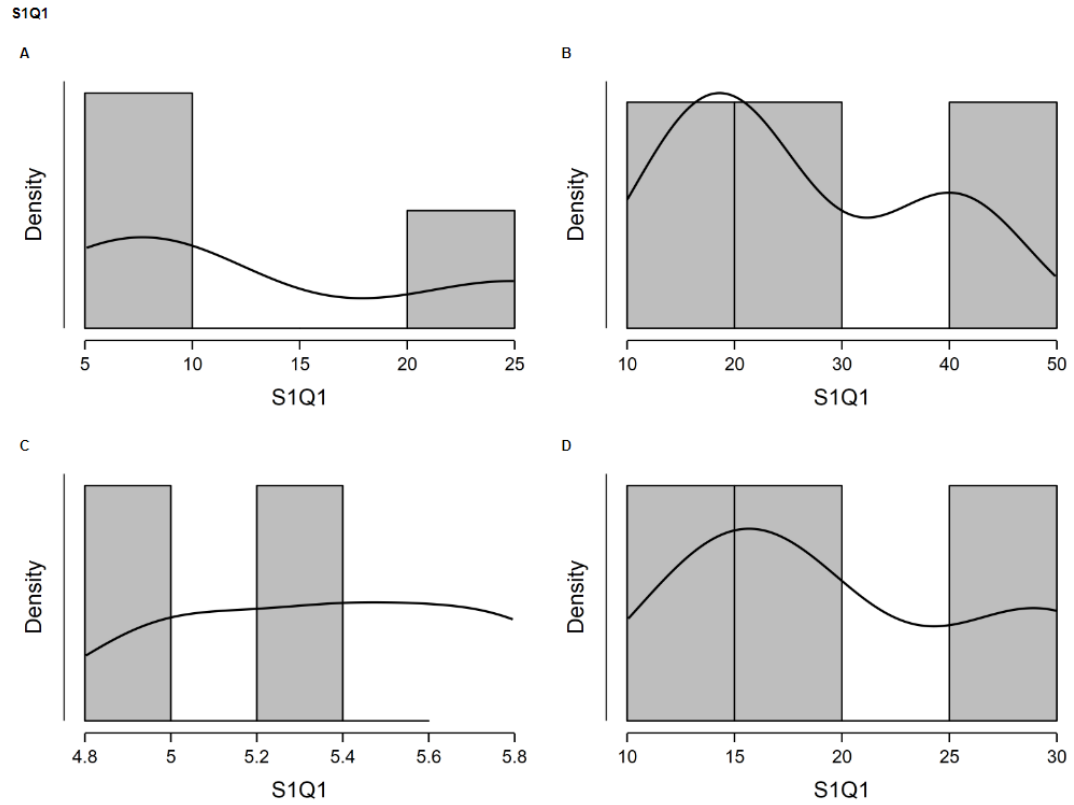


Imagem B - Gráficos de Distribuição por grupo para Questão S1Q2

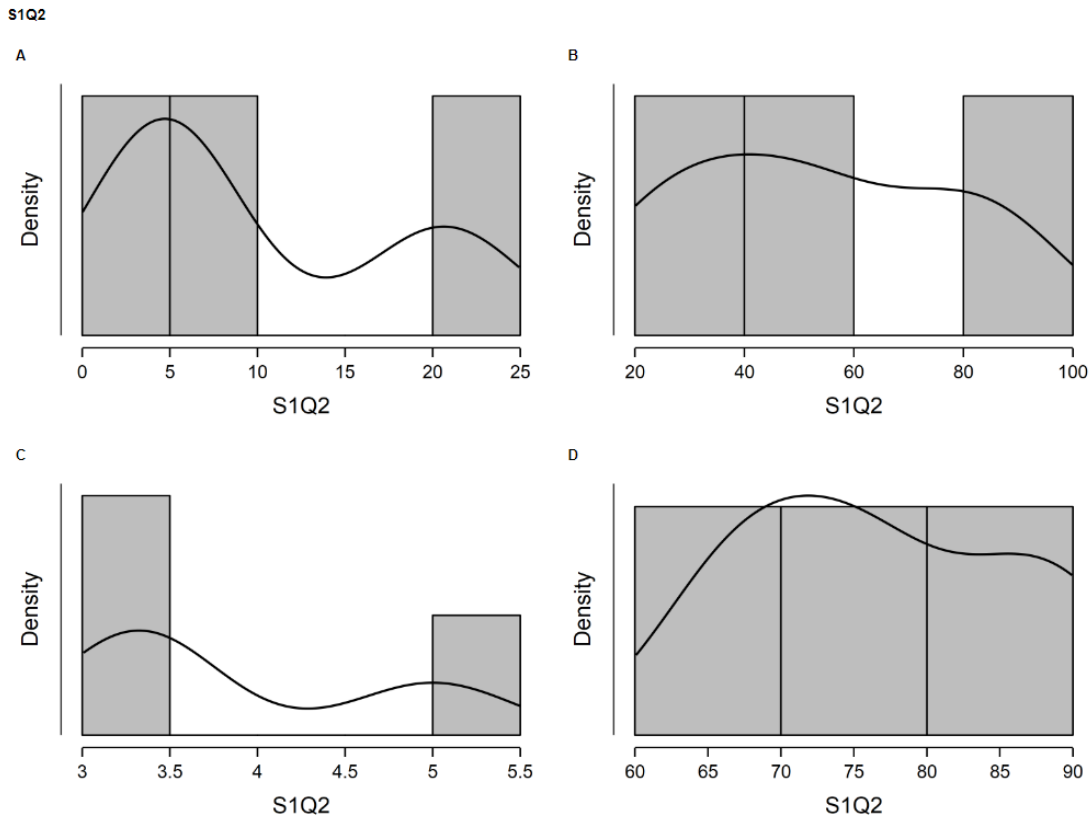


Imagem C - Gráficos de Distribuição por grupo para Questão S1Q3

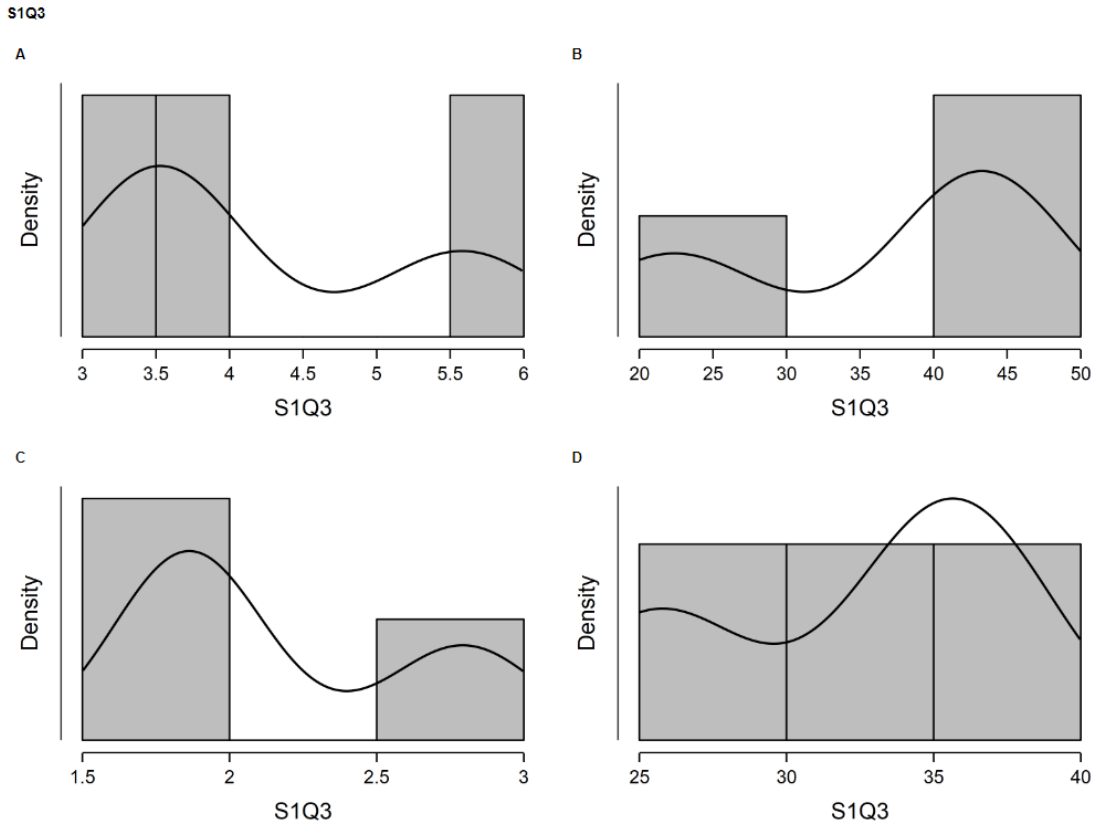


Imagem D - Gráficos de Distribuição por grupo para Questão S1Q4

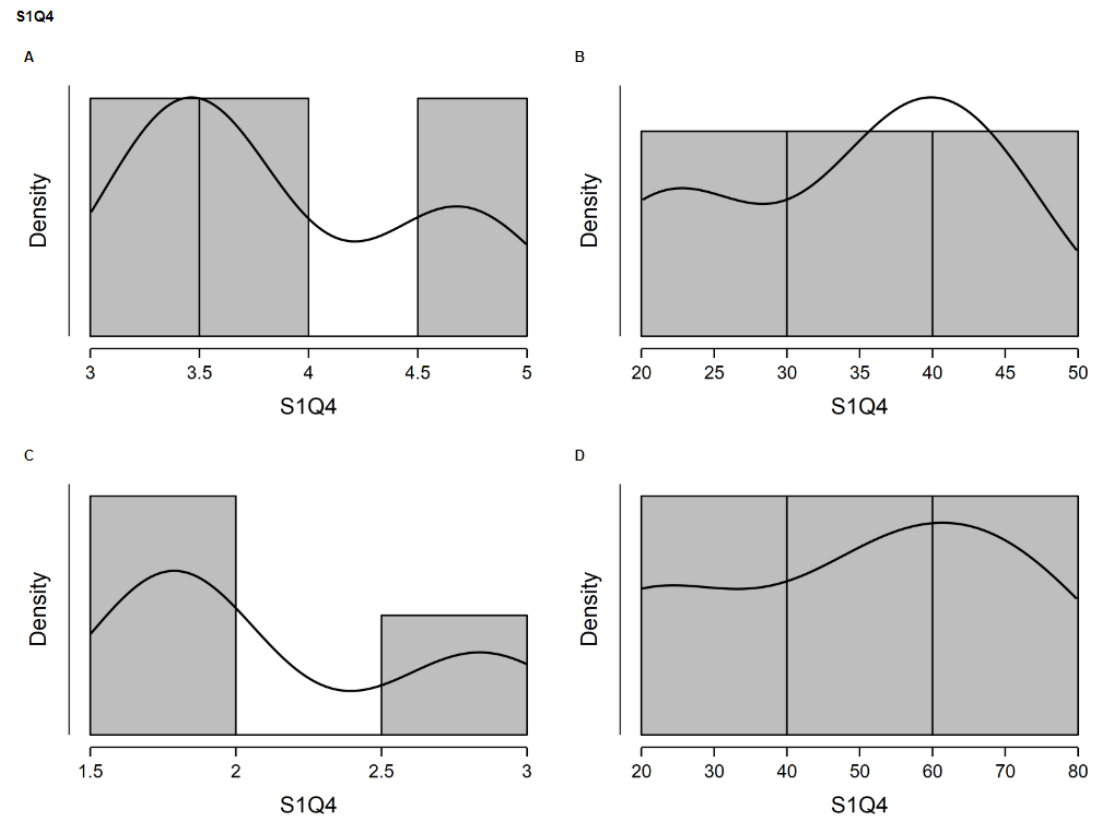


Imagem E - Gráficos de Distribuição por grupo para Questão S1Q5

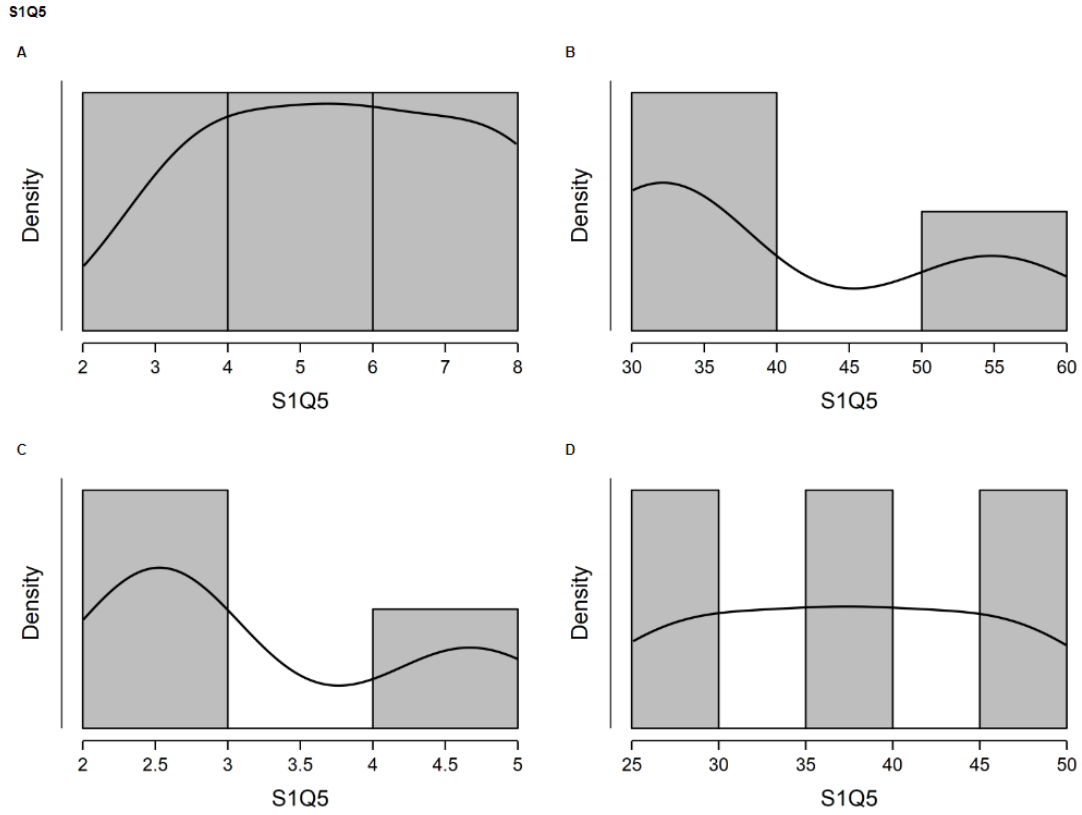


Imagem F - Gráficos de Distribuição por grupo para Questão S1Q6

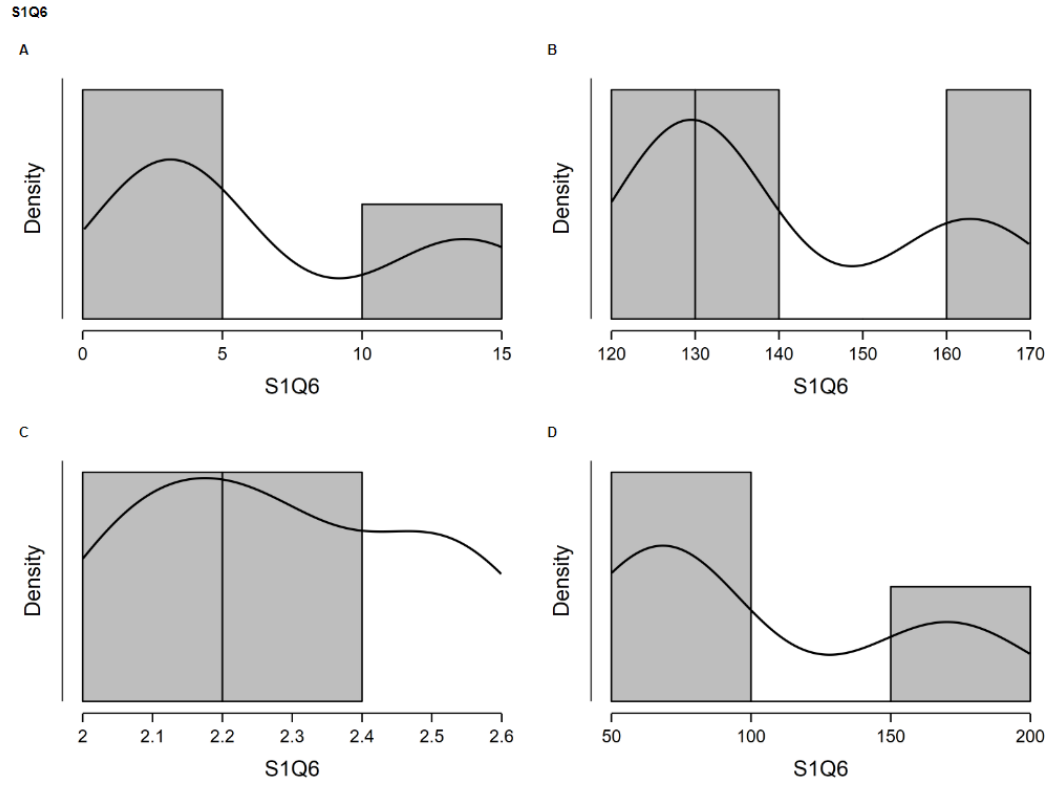


Imagem G - Gráficos de Distribuição por grupo para Questão S3Q1

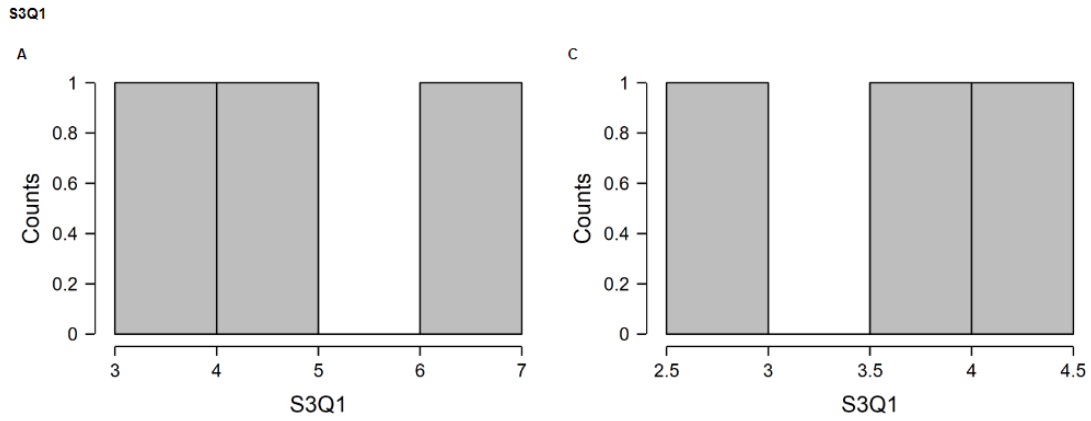


Imagem G – Boxplots por grupo para Questões de S1Q1 a S1Q4

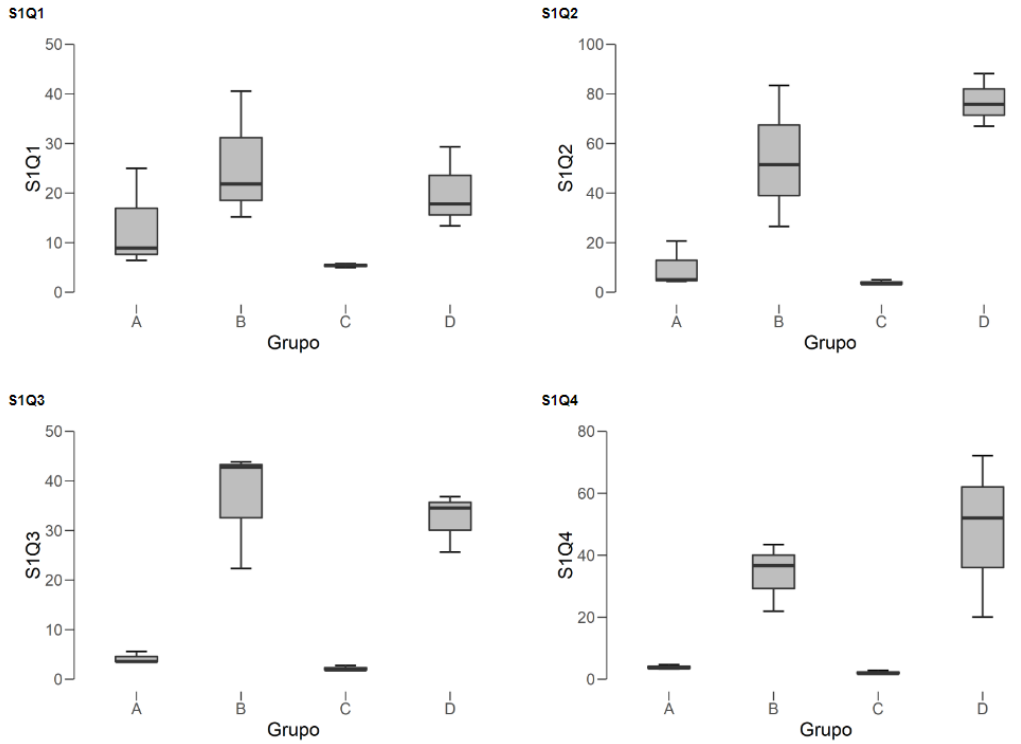


Imagem H - Boxplots por grupo para Questões de S1Q5 a S1Q6

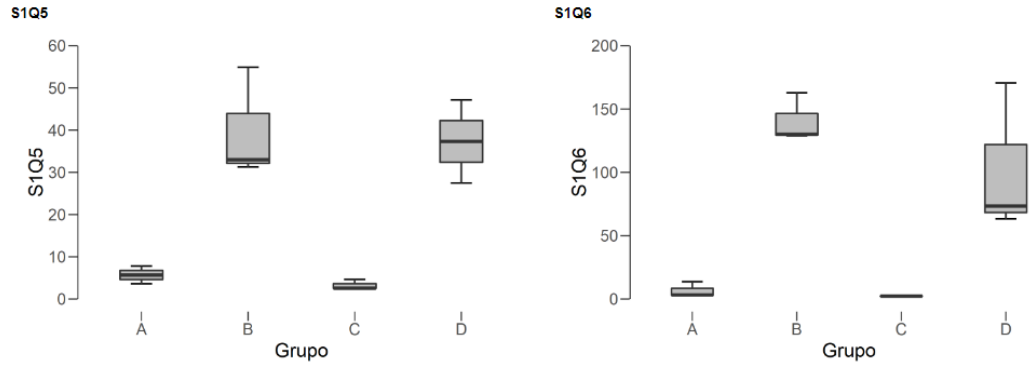
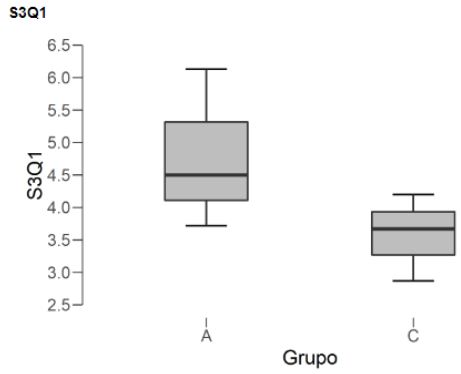


Imagem H - Boxplots por grupo para Questão S3Q1



Apêndice L – Utilização do Ferramental de suporte

Neste apêndice são apresentados os requisitos para utilização do ferramental de apoio para análise de uma arquitetura estabelecida orientada a microsserviços. Para acesso aos repositórios é necessário permissão prévia do autor. Sendo requisitado, o autor pode enviar uma imagem de uma máquina virtual criada com Oracle VirtualBox 6.0 para substituir as etapas de instalação abaixo. Para os requisitos de instrumentação de código, este pode ser implementado em qualquer linguagem que possua dependências disponíveis para o Hashicorp Consul, MySQL e SonarQube.

Requisitos de Hardware

- 4 GB de memória RAM
- 20 GB livres no disco rígido
- 1 core de CPU

Requisitos de Software

- Sistema Operacional: CentOS 7 ou superior
- Docker 18.09.2 ou superior
- Docker-Compose versão 1.23.2
- Cliente do git
- SE Linux desabilitado ou com permissão para execução de contêineres
- Permissão no firewall para comunicação com os serviços a serem analisados

Requisitos para Instrumentação do código (Exemplo para Java)

- Java 1.8 ou superior
- Possuir um gerenciador de dependência (ex: Apache Maven, Gradle)
- Para projetos com spring boot, adicionar a dependências a seguir aos projetos a serem analisados:
    - spring-cloud-dependencies – versão Dalston.RELEASE
    - spring-cloud-starter-consul-config – versão 1.5.3.RELEASE
    - spring-cloud-starter-consul-discovery – versão 1.5.3.RELEASE
    - mysql-connector-java – versão 5.1.45

- Adicionar aos arquivos de configuração (yml) do spring boot a referências ao consul

```
spring:
  profiles: dev
   cloud:
    consul:
     host: <ip do servidor do M2CSEA Engine>
     port: 8500
```

- Adicionar uma classe de filtro conforme o exemplo a seguir:
    - https://github.com/eduardomioto/m2csea-java-sample-projects/blob/master/microservice-product-rest/src/main/java/br/com/mioto/cloud/filter/StatsFilter.java

- Possuir uma classe de conexão ao banco de dados para inserção no mysql. O ip do servidor para conexão será o mesmo no qual será instalado o M2CSEA Engine, utilizando a porta 3306.

- Adicionar uma classe dedicada aos acessos do Consul conforme o exemplo a seguir:
    - https://github.com/eduardomioto/m2csea-java-sample-projects/blob/master/microservice-product-rest/src/main/java/br/com/mioto/cloud/controllers/ConsulController.java

Etapas para análise da arquitetura e projetos associados

1. Obter os repositórios privados abaixo:
    - https://github.com/eduardomioto/m2csea-engine
    - https://github.com/eduardomioto/m2csea-architecture-probe
    - https://github.com/eduardomioto/m2csea-visualizer

2. Acessar a pasta containers do projeto m2csea-engine

3. Executar o scritpt docker-compose.sh para download das imagens docker, criação dos containeres e inicialização da engine.

4. Ao executar a ferramenta pela primeira vez, acesse o NEO4J para criação de um usuário inicial chamado “admin” com senha “admin”. Este serviço está localizado na URL abaixo.
    - http://<ip do servidor da engine>:7474

5. Ao executar a ferramenta pela primeira vez, acesse o SonarQube para obtenção do identificador de instalação. Este serviço está localizado na URL abaixo.
    - http://<ip do servidor da engine>:9000

6. Ao possuir uma modificação no código-fonte estável (ex: produção), é necessário executar a análise estática do código, conforme a seguir.
    - mvn clean install sonar:sonar -Dsonar.host.url=http://localhost:9000 -Dsonar.login=<identificador de instalação obtido na etapa anterior>

7. Acessar a pasta target no projeto m2csea-architecture-probe e executar o comando abaixo:
    - -jar architecture-probe-1.0.0.jar

8. Acessar a pasta do projeto m2csea-visualizer e executar o comando abaixo:
    - python -m SimpleHTTPServer 7000

Acesso aos Serviços

Serviço	Acesso	Usuário	Senha
M2CSEA Finder	http://host:9100/	-	-
M2CSEA Architecture Probe	http://host:9099/	-	-
M2CSEA Specialist Opinion	http://host:9092/	-	-
M2CSEA Visualizer	http://host:7000/	-	-
Hashicorp Consul	http://host:8500/	-	-
Apache Neo4J	http://host:7474/	admin	admin
SonarQube	http://host:9000/	admin	admin
MySQL	host:3306	root	admin