Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Monitoramento em Tempo de Execução: A construção de um módulo em Erlang para uma Arquitetura Orientada a Serviços

Renan Costa Filgueiras

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientadora
Prof. Dr. Edna Dias Canedo

Brasília
2017

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

FF481m

Filgueiras, Renan Costa
Monitoramento em Tempo de Execução: a construção de um módulo em Erlang para uma Arquitetura Orientada a Serviços / Renan Costa Filgueiras; orientador Edna Dias Canedo. -- Brasília, 2017.
87 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2017.

1. Monitoramento de sistema em execução. 2. Métricas de monitoramento. 3. Visualização de software. 4. Dashboard. I. Canedo, Edna Dias, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Monitoramento em Tempo de Execução: A construção de um módulo em Erlang para uma Arquitetura Orientada a Serviços

Renan Costa Filgueiras

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Edna Dias Canedo (Orientadora)
FGA/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
CIC/UnB

Prof. Dr. Uira Kulesza
DIMAp/UFRN

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 28 de Novembro de 2017

Dedicatória

Dedico este trabalho aos meus pais, Elias Costa Filgueiras e Sandra Regina da Costa Filgueiras, pelo apoio, amor, força e auxílio em me guiar sempre pelo melhor caminho. Aos meus familiares, especialmente meus irmãos Rafael Costa Filgueiras, Dyelly Costa Filgueiras e Newton da Silva Miranda Júnior por me apoiarem nos dias difíceis e sempre me motivarem a continuar, sem vocês não seria possível chegar onde cheguei.

Agradecimentos

Ao iniciar um trabalho de grandes dimensões devemos antes nos organizar e verificar a capacidade de execução deste. Ter um ambiente agradável e suscetível é de fundamental importância para sustentar essa tarefa.

Partindo desse princípio agradeço imensamente aos meus familiares por sempre me propiciarem um lar de conforto, harmonia e amor. Por não me faltar incentivos dos mais próximos, como meus amigos de coração. Aos colegas de trabalho e instituição que me permitiram caminhar em conjunto com esse mestrado.

Aos colegas Everton Vargas Agilar e Mateus Manuel Rodrigues Bezerra pelo apoio técnico, incentivo e por estarem prontamente disponíveis sempre que precisei tirar qualquer dúvida para o desenvolvimento desse trabalho.

Agradeço a Universidade de Brasília por ser uma das casas que me acolhe e vem me acompanhando há muitos anos, afinal hoje sou graduado, graduando, mestre e funcionário desta Universidade.

De fundamental importância agradeço aos professores que ajudaram a trilhar esse caminho e em especial a professora Dra. Edna Dias Canedo que foi mais que uma orientadora, um ombro amigo de confiança e referência.

Aos meus amigos de curso em especial a Alysson Ribeiro, Reinaldo Baldoino e Sigfredo Rocha.

Resumo

Monitoramento de software em tempo de execução tem sido usado para detecção de falhas, análise de desempenho, otimização, diagnóstico e recuperação. Evidenciar e utilizar métricas adequadas são o que possibilita a obtenção dessas informações em tempo de execução do software. Quando agir, o que verificar, o que pode melhorar é uma parte crucial do contexto apresentado. Por esse motivo, este estudo apresentou um módulo em Erlang de monitoramento de sistemas em tempo de execução embasado em um mapeamento sistemático de métricas de monitoramento de sistemas computacionais em tempo de execução a fim de auxiliar na manutenibilidade de sistemas por meio de uma camada de visualização de software. O serviço de monitoramento foi utilizado nos sistemas da Universidade de Brasília a fim de analisar e avaliar os resultados obtidos neste trabalho.

Palavras-chave: Monitoramento, Sistema em Execução, Métricas de Monitoramento, Visualização de Software, *Dashboard*.

Abstract

Monitoring runtime software has been used for fault detection, performance analysis, optimization, diagnosis and recovery. Evidence and proper metrics is what makes it possible to obtain this information in the software runtime. When to act, what to check, which can improve is a crucial part of the pop. For this reason, this study presented a module in Erlang for monitoring systems in execution time based on a systematic mapping of computational systems runtime monitoring metrics in order to assist in the maintenance of systems through a visualization layer software. The monitoring service was used with the systems of the University of Brasília in order to analyze and evaluate the results obtained in this work.

Keywords: *Monitoring, System Running, Monitoring Metrics, Viewing Software, Dashboard.*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CPD Centro de Informática. 2, 3, 7, 24, 25, 32, 38, 49

DPO Decanato de Planejamento e Orçamento. 2

ESB Enterprise Service Bus. 6

FUB Fundação Universidade de Brasília. 2

HTTP Hypertext Transfer Protocol. 6, 7

IEC International Electrotechnical Commission. 5

ISO International Organization for Standardization. 5

REST Representational State Transfer. 6, 7, 13

SOA Service-Oriented Architecture. 5, 6

SOAP Simple Object Access Protocol. 6

TI Tecnologia da Informação. 15

TIC Tecnologia da Informação e Comunicação. 1

UnB Universidade de Brasília. 2, 25, 26

Capítulo 1

Introdução

Grandes investimentos são realizados em sistemas de software e para que as organizações obtenham retorno desse investimento, o software deve ser utilizado por vários anos. O tempo de duração de sistemas de software é variável e pode permanecer por mais de 10 anos [38]. Muitos desses sistemas ainda são fundamentais para as organizações e qualquer falha desses serviços acarretará um sério efeito em seu dia a dia [6].

Lientz [40] menciona que a manutenção de software consome a maior parte do custo total do ciclo de vida de um sistema, no qual representa cerca de 75% a 80% dos recursos de programação consumidos e que, apesar desses altos valores, essas modificações não acompanham um planejamento adequado.

Uma parcela considerável do esforço da atividade de manutenção é realizada na compreensão do software, que muitas vezes, é impedida pela não utilização de qualquer tipo de técnica de desenvolvimento durante a sua criação,falta de documentação e informações atualizadas ou disponíveis dele. [11].

Devido à importância que esses sistemas representam para a organização, bem como o seu conjunto de informações, regras de negócio, funcionalidades cruciais para a empresa, eles se tornam um ponto crítico, sendo imprescindível a realização de algum tipo de monitoramento.

Em razão de tecnologias ultrapassadas e da falta de evolução dos sistemas, alguns problemas são comumente evidenciados, tais como: o não suporte do crescimento do sistema e da quantidade de usuários; grande quantidade de requisitos de negócio em banco de dados através de comportamentos implementados em *stored procedures*; e vulnerabilidades de segurança e confiabilidade do sistema [4].

A necessidade de um aumento contínuo da qualidade na indústria brasileira de software e serviços de Tecnologia da Informação e Comunicação (TIC) está contemplada no âmbito da Política de Ciência, Tecnologia e Inovação do Governo, explicitada no documento de Estratégia Nacional de Ciência, Tecnologia e Inovação 2016-2019. O documento destaca

a importância em contribuir para o desenvolvimento nacional por meio de iniciativas que valorizem o avanço do conhecimento e da inovação [16].

A Fundação Universidade de Brasília (FUB), instituída nos termos da Lei n. 3.998, de 15 de dezembro de 1961, tem sede e foro na cidade de Brasília. O Centro de Informática (CPD) é um Órgão Complementar da Universidade de Brasília (UnB), responsável pela Tecnologia da Informação e subordinado ao Decanato de Planejamento e Orçamento (DPO). O CPD tem definido como um dos objetivos "Promover e incentivar a informática na Universidade de Brasília visando obter eficiência institucional em todos os níveis"e "Promover e incentivar a informática na Universidade de Brasília para alcançar maior eficácia no suporte às atividades de ensino, pesquisa, extensão e administração da Instituição", onde um dos softwares principais mantidos pela organização é o SIGRA (Sistema de Informações Acadêmicas de Graduação), com um módulo WEB que atinge a todos os estudantes de graduação e pós-graduação da universidade, o Matrícula Web.

Levantar métricas adequadas ao monitoramento de sistemas em tempo de execução vai ao encontro de contemplar as abordagens de análise dinâmica de código, auxiliando no processo de monitoração dos softwares e servindo de apoio as instituições. Portanto, possuir um *dashboard* dessas informações é uma grande ferramenta para auxiliar o monitoramento pela equipe que mantém o sistema.

Este trabalho irá realizar um estudo de caso na Universidade de Brasília implantando uma ferramenta de monitoramento de sistemas em execução, utilizando-se de métricas encontradas na literatura e que se adeque a realidade do CPD, disponibilizando uma camada de visualização com o *dashboard* para acompanhamento das equipes mantenedoras dos sistemas.

1.1 Justificativa

As abordagens por análise estática são encontradas em um grande número de aplicações [10], porém para a análise dinâmica de código esse número é reduzido. A análise de código fonte não proporciona informações suficientes para entender completamente um sistema, já que existem componentes e relações que só existem durante sua execução [60]. Trabalhar então com uma abordagem complementar poderá contribuir significativamente para o acompanhamento e monitoramento dos sistemas legados, aumentando a qualidade da manutenção e evolução deles.

Períodos de grandes demandas no sistema, como é o caso dos sistemas da Universidade de Brasília que trabalham com demandas sazonais, podem apresentar uma série de gargalos devido a problemas exemplificados por Azevedo [4] e que em virtude da falta de monitoramento, as informações de erros, causas e motivos desses gargalos se perdem. Re-

alizar testes de carga, desempenho, stress após o ocorrido podem ser de muito custo para a Instituição e não refletir o cenário real, o que leva a correções e manutenções possivelmente não tão eficazes. Essa é uma realidade vivida diariamente no CPD da Universidade de Brasília, mudando apenas o sistema da vez.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é propor um serviço de monitoramento de software em tempo de execução com uma camada de visualização a fim de gerar um *dashboard* para auxílio na compreensão do comportamento dos sistemas. Um estudo de caso será realizado nos sistemas da Universidade de Brasília a fim de complementar o trabalho proposto por Agilar [17] e avaliar a solução sugerida.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral, os seguintes objetivos específicos são definidos:

- Realizar um mapeamento sistemático para identificar as métricas de monitoramento em tempo de execução em software;
- Avaliar as ferramentas existentes e verificar se atendem ao contexto do CPD;
- Avaliar a necessidade de customizações da ferramenta selecionada quanto a lista de métricas selecionadas;
- Avaliar a utilidade de propor uma nova métrica para monitoramento de acordo com a necessidade do CPD;
- Implementar o serviço de monitoramento;
- Implementar a camada de visualização e *dashboard*;
- Analisar e avaliar os resultados da experiência da equipe utilizando a solução proposta.

1.3 Estrutura do trabalho

Este trabalho está estruturado em 7 capítulos, além deste. No Capítulo 2 é apresentada uma fundamentação teórica acerca dos assuntos necessários para o entendimento deste trabalho. No Capítulo 3 é apresentada a execução de um mapeamento sistemático com

o objetivo de identificar métricas de monitoramento em tempo de execução em software. No capítulo 4 é apresentado um estudo de interesse a fim de verificar a importância das métricas encontradas na literatura junto ao CPD. No Capítulo 5 é evidenciada a camada de monitoramento que foi realizada para obtenção das métricas. No Capítulo **??** é mostrada a ferramenta desenvolvida para a camada de visualização do monitoramento de sistemas. No Capítulo 6 é apresentada a avaliação da equipe de desenvolvimento do CPD referente a percepção das métricas implantas e do novo serviço de monitoramento. No Capítulo 7 conclui-se a respeito dos resultados obtidos e levanta-se uma frente da possível continuidade desse trabalho.

Capítulo 2

Fundamentação Teórica

2.1 Sistema em execução

Um sistema, segundo definição da ISO/IEC 12207 [30] e ISO/IEC 9126-1 [29], é um conjunto integrado que consiste em um ou mais processos, hardware, software, recursos e pessoas, capaz de satisfazer uma necessidade ou objetivo definido. Um software é definido como um conjunto completo ou apenas uma parte dos programas, procedimentos, regras e documentação associada de um sistema de processamento de informação [29].

A arquitetura de um software corresponde à estrutura ou estruturas de um sistema, abrangendo os componentes, as propriedades externamente visíveis desses componentes e as relações entre eles [39].

Sistema em execução refere-se, portanto, ao estado em que um sistema encontra-se e que permite o uso de suas funcionalidades por seus devidos usuários. É o momento onde é possível observar a interação entre as relações presentes na(s) estrutura(s) da arquitetura de software.

2.2 Arquitetura Orientada a Serviços

A arquitetura orientada a serviços (SOA) retrata um modelo arquitetural que busca melhorar o custo por eficácia e agilidade de resposta de um sistema, tanto no nível da organização quanto na equipe de tecnologia da informação, onde os serviços são o coração dessa solução [20].

A implementação de uma arquitetura orientada a serviços consiste em um conjunto de componentes que dispõe e/ou consome serviços. Em grande parte dessa solução temos os serviços de forma autônoma que repre108268sentam os provedores de serviços e os consumidores de serviços que geralmente são implantados de forma independente, podendo pertencer a sistemas diferentes ou até mesmo a organizações diferentes [14].

Em uma arquitetura SOA tem-se os consumidores de serviços conectados aos provedores de serviços, podendo fazer uso de diversos componentes intermediários como: ESB, registros, *server orchestration*, os prestadores de serviço também podem ser consumidores. [5].

Componentes de uma arquitetura orientada a serviços (SOA):

- **Prestador de serviços**: que fornece um ou mais serviços por meio de interfaces publicadas;
- **Consumidores de serviço**: que invocam serviços diretamente ou por meio de um prestador de serviço intermediário;
- ***Enterprise Service Bus*** *(ESB)*: é um elemento intermediário que pode rotear e transformar mensagens entre prestadores e consumidores de serviços;
- **Registro de serviços**: pode ser usado por prestadores de serviços para registrar seus serviços e consumidores de serviços para descobrir os serviços em tempo de execução;
- ***Server Orchestration***: coordena as interações entre consumidores e provedores de serviços com base em linguagens para processos de negócios e fluxos de trabalho.

Essa arquitetura permite o uso de diferentes tipos de conectores:

- **Simple Object Access Protocol (SOAP)**: usa o protocolo SOAP para comunicação síncrona entre serviços da Web, tipicamente por HTTP.
- **Representational State Transfer (REST)**: depende das operações básicas de solicitação e resposta do protocolo HTTP.
- **Conector de mensagens assíncronas**: usa um sistema de mensagens para oferecer *point-to-point* ou *publish-subscribe* para trocas de mensagens assíncronas.

A interoperabilidade é o principal benefício da arquitetura SOA, já que os provedores e consumidores de serviços podem ser executados em diferentes plataformas, podendo integrar uma variedade de sistemas, incluindo sistemas legados [5].

Diante da arquitetura proposta por Agilar em [17] que representa a arquitetura atual do desenvolvimento de serviços do CPD, o conector REST foi o adotado sob a justificativa da facilidade no desenvolvimento, o aproveitamento da infraestrutura web existente, um esforço de aprendizado menor e o uso de JSON ao invés de XML, que se sobressaem dos benefícios do SOAP que é um padrão da indústria, com protocolos bem definidos e um conjunto de regras bem estabelecidas.

2.3 Transferência de Estado Representacional

O *Representational State Transfer* (REST) é um conector que possibilita um consumidor de serviços enviar solicitações síncronas HTTP. Esses pedidos referenciam os quatro comandos básicos HTTP (POST, GET, PUT, DELETE) para requisitar ao provedor de serviços criar, recuperar, atualizar ou excluir um recurso (um dado). Os recursos têm uma sintaxe bem definida representada em XML, JSON ou uma linguagem de notação [5].

REST distingue três classes de elementos arquiteturais [32]:

- **Elementos de dados**: O aspecto chave do REST é o estado dos elementos de dados, seus componentes se comunicam transferindo representações do estado atual ou desejado dos elementos de dados;
- **Elementos de conexão (conectores)**: Gerenciam a comunicação de rede para um componente;
- **Elementos de processamento (componentes)**: Os componentes REST são identificados pelo seu papel dentro de um aplicativo.

Os serviços REST (*RESTful Services*) são programas leves que são elaborados com ênfase na simplicidade, escalabilidade e usabilidade. Esses serviços podem ser moldados por uma aplicação de princípios de orientação de serviços [20].

2.4 ErlangMS

ErlangMS é uma solução proposta inicialmente no trabalho de Agilar [17] que faz uso de uma arquitetura orientada a serviços (SOA) utilizando como conector o padrão REST e JSON para troca de mensagens entre o prestador e o consumidor. Esta solução tem como finalidade ser um barramento ESB implementado em Erlang para a nova arquitetura de serviços que o Centro de Informática (CPD) vem implementando, todavia pode ser utilizado em qualquer outro domínio que faça uso de um ESB.

A arquitetura da solução barramento ErlangMS é apresentada na Figura 5.1.

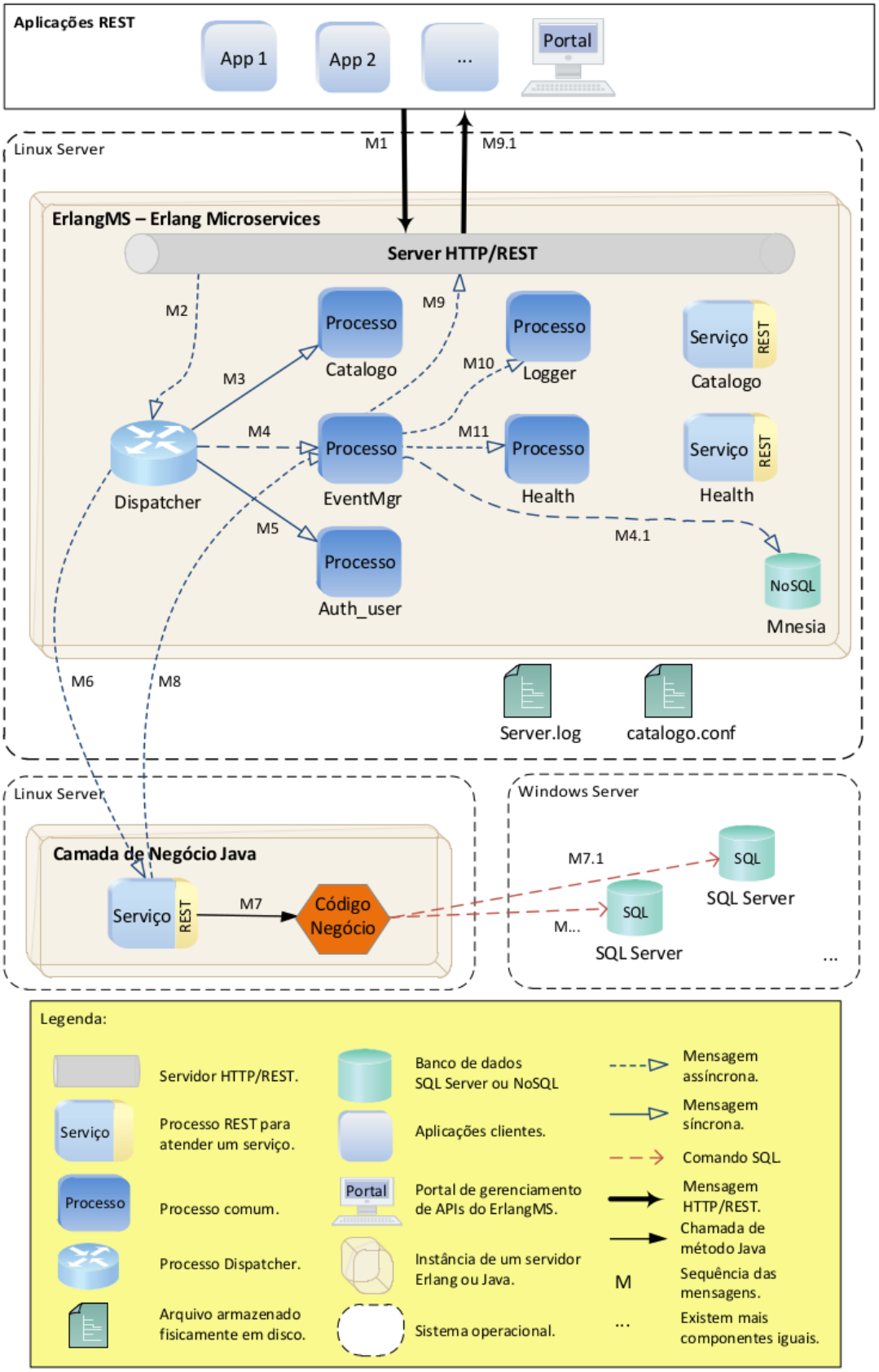

Figura 2.1: Arquitetura ErlangMS proposta por Avilar [17]

2.5 Monitoramento de software

Monitoramento é definido como a capacidade de observar, dado determinado período de tempo, se as condições de um objeto estão definidas de acordo com um padrão estabelecido [46].

Este diverge do termo rastreamento que acompanha o ir e vir de um objeto, portanto, monitoramento verifica somente se uma dada condição é aceitável ou não aos padrões pré-estabelecidos.

Logo, monitoramento de software pode ser definido como um processo de extração e levantamento de informações sobre o comportamento de um sistema específico [55].

Atualmente, o CPD não dispõe de nenhum serviço de monitoramento de sistemas, todo o monitoramento de sistemas hoje, é realizado de forma manual o que onera as equipes que mantêm os sistemas. As análises em tempo real, para possíveis correções de *bug* dos sistemas mais críticos durante as grandes demandas que ocorrem sazonalmente no CPD, são realizadas por meio de análise de sentimento de *feedbacks* recolhidos nas redes sociais e *e-mails*, conforme apresentado na Tabela 2.1.

Tabela 2.1: Relatos de alunos do primeiro semestre de 2016 referente ao período de matrícula

Aluno	**Feedback**
1	Prezados, gostaria de pedir, que arrumem o devido site de vocês. Estou desde 13h da tarde tentando fazer uma matricula vinculada e ate agora NADA. Apenas uma crítica construtiva.
2	Olá, estou desde 00h tentando me matricular em disciplinas mas na página do matrícula web só dá mensagens de erro. O que eu faço? Preciso disso para hoje pois sou formanda.
3	Boa tarde! Estou tentando, desde 00:00hrs de hoje, realizar a minha matricula vinculada, porém ao apertar em "OK"a página seguinte sempre dá erro e não conclui a minha solicitação. Já troquei de navegador, porém o erro ainda persiste. O que eu posso fazer para concluir a minha matrícula?
4	Estou acessando o matrícula web e a matrícula está aberta. Quando clico para fazer está dando erro. Gostaria de saber se está aberta mesmo ou se está dando erro por estar sobrecarregado. Abaixo adicionei um anexo do que aparece na oferta. Desde já agradeço.

A lista completa de reclamações obteve 458 relatos referentes ao primeiro semestre de 2016. É um dado importante para verificações de correções no sistema, porém é algo que

demanda tempo para análise, o que impossibilita uma intervenção mais efetiva durante a execução do sistema. Outro fator, é que vários relatos não trazem informação pertinente, sem detalhamento ou apresentando somente problemas em cosequência da falta de conhecimento do usuário junto ao sistema, como são os casos normalmente relatados por alunos do primeiro semestre (calouros) da UnB.

2.6 Métricas de software

Uma métrica é definida como um método e escala de medição [29]. Sommerville [59] define métrica de software como qualquer tipo de medição referente a um software, processo ou documentação. A medição de software auxilia a compreensão do processo de engenharia de software e seu produto, partindo da utilização de métricas [52]. Portanto, métricas são aplicadas para medir produtividade, qualidade do produto, estimar cronograma e custos de desenvolvimento de software [45].

Métricas possuem valores predominantemente numéricos, os quais isolados têm pouco significado para os desenvolvedores e possuem normalmente um processo de extração com grande volume de dados [28]. Planejar, identificar e organizar o que se deseja é uma fase crucial para um desempenho adequado das métricas de software [59].

As métricas de software serão consideradas aqui de acordo com a categorização que define métricas como primitivas e compostas [26]. Métricas primitivas são as obtidas pela observação direta dos atributos e as métricas compostas são obtidas a partir de outras métricas como eficiência, qualidade, confiabilidade.

As métricas que serão consideradas neste trabalho estão classificadas de acordo com a Tabela 2.2

Tabela 2.2: Categorização das métricas

Métricas Primitivas	**Métricas Compostas**
Número de instâncias por transações	Cobertura estruturais de código - MC/DC
Frequência de uso do método	Consumo de recursos físicos
Porcentagem de utilização da CPU	
Número de invariantes	
Quantitativo de erros por serviço	

2.7 Visualização de Software

Com o uso de representações gráficas e interativas, torna-se possível o mapeamento de atributos ou características do domínio observado por meio de propriedades visuais tais

como: posição, forma, cor, tamanho, possibilitando o entendimento, tomadas de decisões, tendências [44]. Essa representação é o meio para extração de conclusões sobre os dados, é a interação direta com a visualização, moldando-a para atingir os objetivos do observador.

Possibilitar que grandes quantidades de informações sejam objetivamente compreendidas é o objetivo da visualização de informação em software, permitindo assim estabelecer análises que apoiam as tarefas de avaliação, melhoria de qualidade e monitoramento [15] [42].

A visualização de software está dividida em três categorias principais:

- **Estrutura**: representa as partes estáticas do sistema, são verificadas sem a necessidade de executar o sistema.
- **Comportamento**: Referente a compreensão do ocorrido com o sistema durante seu tempo de execução, como seus estados mudam dado um conjunto de possíveis entradas e quais instruções são executadas.
- **Evolução**: Retrata as características modificadas ao longo do tempo em um sistema.

A categoria que vai ao encontro do objetivo desse trabalho é a de comportamento. Utilizar-se das técnicas e modelos referentes a esta categoria irá possibilitar a melhor visualização dos dados obtidos e sua compreensão.

2.8 Dashboard

Dashboard é uma forma de agrupar e representar dados em um monitor a fim de possibilitar melhor compreensão e correlações dos dados exibidos que comumente são avaliados de forma independente perdendo informações [51] [19] [43]. Ele oferece uma solução única e poderosa que vai junto a necessidade das organizações em obter informações e tem sido utilizado por diversas aplicações em contextos variados, inserido até mesmo para apoiar aprendizagem e ensino nas escolas [63] [23].

A utilização de soluções baseadas em *dashboard* vem sendo subestimada graças a não utilização do seu potencial e isso advém da falta de compreensão dos dados obtidos e da má seleção de informações a serem adquiridas [21]. Contudo, é possível obter um ganho elevado de informações ao apresentar todas as informações em uma tela, abreviando em forma de resumos ou exceções e, com isso, indicar rapidamente um item que merece uma atenção para ação rápida [21].

De acordo com Few [21], *dashboard* pode ser dividido em 3 grupos de acordo com sua atividade de apoio:

1. **Estratégico**: Concentra métricas de alto nível de desempenho e pode permitir previsões para tomada de decisão;
2. **Suporte**: Não tem foco no avaliadores de desempenho, mas sim em comparações ricas entre os dados e descrição detalhada histórica;
3. **Operacional**: Apresentação de dados em tempo real e com medidas eficientes para atentar o usuário do que está acontecendo.

A importância em categorizar o *dashboard* é tornar o seu uso focado no intuito de utilização pela organização, evitando informações desnecessárias e poluição de dados. Associado a esse trabalho o **grupo operacional** é o adequado para o monitoramento em tempo real e possibilitará um acompanhamento mais eficaz das métricas selecionadas pela equipe de manutenibilidade.

2.9 Trabalhos Relacionados

Na literatura as abordagens por análise estática são encontradas em um grande número de aplicações [10], entretanto para avaliação dinâmica de código esse número é reduzido. Existem componentes e relações que só existem durante sua execução, conforme relatado no trabalho apresentado por [60].

Schmid [55] realizou um trabalho de pesquisa dos termos, problemas e conceitos que envolvem a área, com detalhamento aprofundado e expondo o trabalho inicial do desenvolvimento de um monitor. O trabalho tem sua relevância como fundamento e vale para enfatizar a importância do monitoramento de sistema em execução, já que trabalha com a prospecção positiva desse tema.

O trabalho [18] apresenta a taxonomia das ferramentas de monitoramento em tempo de execução do software, o que possibilita a melhor utilização dessa abordagem e auxilia na compreensão dos resultados deste trabalho. A listagem de ferramentas é uma rica informação para auxiliar na decisão de escolha de alguma delas, e em virtude da sua classificação taxonômica, caso outra ferramenta seja encontrada, é possível classificá-la e compará-la com a lista já existente.

Robinson [54] relata em seu trabalho o monitoramento de requisitos de software usando instrumentação de código. Percebe-se em seu estudo a preocupação com o estado do software em execução, pois este é o resultado final dos requisitos do sistema e que o usuário tem acesso. O seguimento dado para realizar a monitoração (análise do processo, definição do escopo, itens a serem abordados) é interessante como modelo, porém o trabalho utiliza-se da técnica de instrumentação de código a fim de obter os dados desejados. Instrumentar o código em sistemas pode gerar grande esforço de implementação e esbarrar em barreiras

tecnológicas como é a condição da Universidade de Brasília. Por esse motivo as métricas a serem levantadas devem interferir minimamente no código do sistema.

Chodrow [12], Thane [62], Plattner [50] e Jahanian [31] apresentam a aplicação de técnicas de monitoramento em sistemas em execução descrevendo suas abordagens e verificações realizadas. São trabalhos técnicos com estudos de caso, o que possibilita uma melhor visão do problema abordado nesse estudo.

Pode se observar que os trabalhos mencionados não verificam a conformidade das métricas existentes a determinado sistema e não realizam o apanhado destas.

Viabilizar um lista de métricas e sua implantação é um produto que irá permitir um monitoramento mais eficaz de acordo com o escopo desejado. Outro ponto distinto nesse estudo é a disponibilização de um serviço de monitoramento adequando-se as técnicas de visualização de software e a disponibilidade desses resultados em uma visão de *dashboard*.

2.10 Síntese do Capítulo

Este Capítulo apresentou os conceitos necessários relacionados a sistema em execução, arquitetura orientada a serviços, REST, ErlangMS, métricas de software, visualização de software e Dashboard. Estes conceitos são importantes para o entendimento e desenvolvimento deste trabalho. Além disso, este Capítulo apresentou os trabalhos correlatos ao tema desta dissertação. O Capítulo 3 apresentará a revisão sistemática realizada.

Capítulo 3

Mapeamento Sistemático

O mapeamento sistemático é um tipo de estudo que propõe responder a uma pergunta específica e utiliza-se de métodos explícitos e sistemáticos para identificar, selecionar e avaliar criticamente os estudos [9] [48]. Baseado em evidências e foco na síntese dos resultados de pesquisas de acordo com a problemática especificada [48].

A metodologia aplicada para realizar um mapeamento sistemático foi referente as publicações Cochrane Handbook [13] e seguindo as orientações para a realização de estudos de mapeamento sistemático em engenharia de software [49].

O objetivo deste mapeamento sistemático é analisar relatos de experiência e descrição na literatura correspondente a abordagens de utilização de métricas para monitorar sistemas em execução, com propósito de identificar métricas relevantes para realizar o monitoramento dos sistemas em tempo de execução.

3.1 Questões de Pesquisa

Neste trabalho foram definidas 2 questões de pesquisas (QP) para nortear o mapeamento sistemático.Essas questões serão respondidas no desenvolvimento desta dissertação.

As questões de pesquisa definidas para alcançar o objetivo proposto, que é identificar métricas para o monitoramento de sistemas em tempo real são:

QP1. Quais métricas de software são utilizadas para monitoramento em tempo de execução?

QP2. Quais abordagens são utilizadas para monitoramento de sistemas em execução?

3.1.1 Estratégia de Busca

A *string* de busca foi elaborada seguindo propostas apresentadas por Keele [34], que recomenda a criação da *string* por meio de uma lista de sinônimos, abreviaturas e grafias alternativas podendo ser sofisticada, utilizando-se de operadores lógicos AND e OR. A *string* de busca obtida foi:

(("system monitoring"OR "monitoring software") AND ("tracking metrics"OR "monitoring metrics") AND ("code instrumentation") AND ("running system"OR "real-time monitoring"))

Em relação ao escopo da pesquisa, os critérios adotados para selecionar as fontes de busca foram:

1. Bases com renome e difundidas na área de Tecnologia da Informação (TI);
2. Grande quantidade de material publicado e engenhos de busca intuitivos para filtrar os resultados;
3. Possuir relação com o tema a ser pesquisado;,
4. Selecionar bases que pudessem evidenciar o cenário das universidades brasileiras em relação ao tema.

Foram selecionadas as bases de busca:

- ACM – http://dl.acm.org/;
- IEEE xplore – http://ieeexplore.ieee.org/;
- ScienceDirect – www.sciencedirect.com ;
- Springer Link – https://link.springer.com/ .

Estas bases de buscas foram selecionadas por apresentar uma grande base de dados, possuir um bom funcionamento e abrangência dos critérios adotados para seleção dos artigos.

3.1.2 Procedimento e Critérios de Seleção e Exclusão

Os critérios de seleção e exclusão foram adotados com o intuito de refinar o resultado da busca e obter trabalhos mais adequados ao estudo. A análise por meio dos critérios definidos se deu a partir das informações parciais disponibilizadas pela ferramenta de busca (Título, referência, resumo e conclusão) e download do artigo.

Critérios de seleção:

CS1. Estudo publicado entre o período correspondente de 2006 a 2016;

CS2. Estudo estar disponível para download nas plataformas selecionadas;

CS3. Estudo disponibilizado nos idiomas português ou inglês.

Critérios de exclusão:

CE1. Estudo incompleto, apresentando somente resumo, conclusão;

CE2. Estudo não se enquadra como artigo;

CE3. Estudo de outras áreas de conhecimento;

CE4. Estudo não estar correlacionado com monitoramento de sistemas e/ou sistemas legados;

CE5. Estudo não apresenta métricas ou abordagens de monitoramento de sistemas.

3.1.3 Processo de seleção

A seleção dos artigos foi realizada em 3 etapas:

1. Seleção e catalogação preliminar dos artigos coletadas nas fontes a partir da string de busca. Para os artigos selecionados, os seguintes dados foram extraídos: título, autor(es), data da publicação, fonte de publicação, referência e resumo do artigo.
2. Verificação dos critérios de seleção
    - Filtro de seleção dos artigos relevantes - 1, verificar data de publicação aplicando o critério de seleção CS1;
    - Filtro de seleção dos artigos relevantes - 2, verificar disponibilidade do artigo de acordo com o critério de seleção CS2;
    - Filtro de seleção dos artigos relevantes - 4, verificar idioma do estudo de acordo com o critério de seleção CS3;
3. Verificação dos critérios de exclusão
    - Filtro de exclusão dos artigos - 1, verificar completude do artigo conforme critério de exclusão CE1;
    - Filtro de exclusão dos artigos - 2, verificar tipo de estudo conforme critério de exclusão CE2;
    - Filtro de exclusão dos artigos - 3, verificar área de conhecimento do estudo conforme critério de exclusão CE3;
    - Filtro de exclusão dos artigos - 4, por meio de análise do resumo (*abstract*) e aplicando o critério de exclusão CE4;
    - Filtro de exclusão dos artigos - 5, verificar presença de métricas e abordagens conforme critério de exclusão CE5;

E subdividida em 2 fases

1. Fase 1: Realização da etapa 1, 2 e filtros 1, 2 e 3 da etapa 3. Seleção realizada por recursos disponíveis nas plataformas das bases de busca;
2. Fase 2: Realização dos filtros 4 e 5 da etapa 3. Seleção realizada por análise e leitura dos estudos encontrados.

A fim de levantar a listagem de quais métricas de monitoramento foram evidenciadas e/ou abordagem de monitoramento de sistemas e sistemas legados em execução foi utilizado um instrumento para extração de dados conforme apresentado na Figura 3.1, proposto pelo Instituto Joanna Briggs e evidenciado por Soares e Yonekura [58].

Os dados coletados nos artigos selecionados foram analisados quantitativa e qualitativamente. A análise qualitativa se deu através de um mapeamento das abordagens e métricas encontradas e como elas interagiam uma com a outra. A análise quantitativa resultou em uma lista das métricas que podem ser aplicadas para o monitoramento de sistemas em execução.

3.1.4 Análise preliminar da busca

O levantamento inicial da seleção de artigos definido como Fase 1, resultou em um total de 189 estudos, sem considerar os critérios de exclusão CE4 e CE5, pois estes não são realizados por meio da plataforma de busca e sim por uma análise do trabalho. O detalhamento dessa seleção é demonstrado na Tabela 3.1. A distribuição da Fase 1, conforme as bases selecionadas, foi ACM - 15%, IEEE xplorer - 28%, ScienceDirect - 12% e Springer Link - 45%, sendo esta a base com a maior porcentagem de estudos na primeira fase, conforme demonstrado na Figura 3.2. A Fase 2 levou em consideração os critérios de exclusão CE4 e CE5 que dependem da avaliação do estudo com base nas questões de pesquisa levantadas. Com esses critérios, 11 trabalhos foram selecionados, que estão distribuídos conforme Figura 3.3 e demonstrado o refinamento dos resultados na Tabela 3.2. O resultado final dos artigos selecionados foi categorizado por país conforme Figura 3.4.

3.2 Resultados

Um resumo dos resultados encontrados é apresentado nesta Seção e organizados de acordo com as questões de pesquisa definidas na subseção 3.1 e no final apresentado uma listagem das métricas encontradas.

Os estudos selecionados [64] e [1] fogem do escopo de sistemas legados, pois apresentam métricas relacionadas somente com tecnologias de computação em nuvem. Os estudos [27]

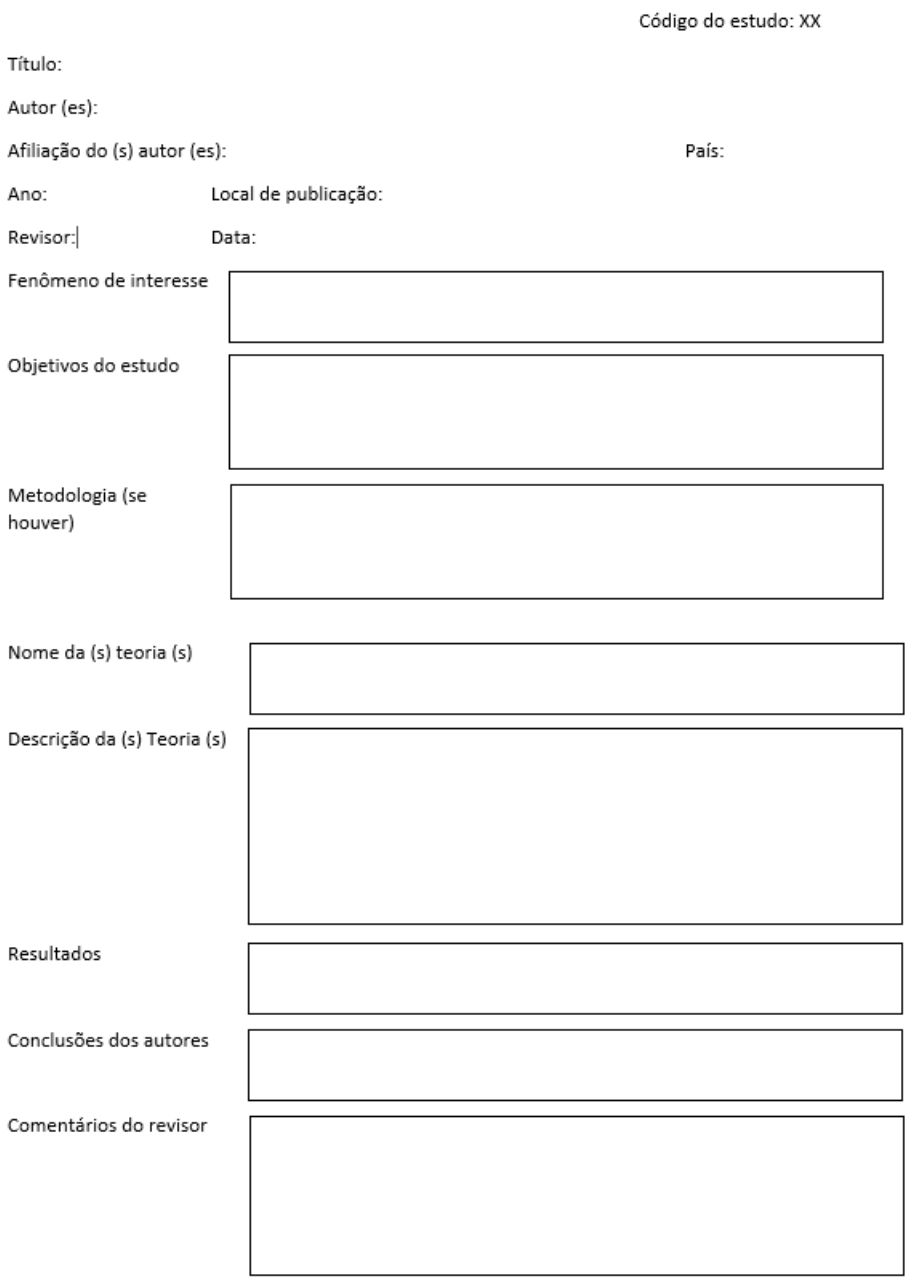

Figura 3.1: Instrumento para extração de dados adaptado de Joanna Briggs Institute - São Paulo - 2010

e [36] não apresentam métricas de sistema durante execução, as métricas apresentadas referenciam somente análise estática de código fonte e não entraram para a listagem.

QP1. Quais métricas de software são utilizadas para monitoramento em tempo de execução?

O estudo [57] demonstra a métrica por número de instâncias por transações. Esta métrica levanta informações úteis para alocação dinâmica de recursos e balanceamento de carga.

Tabela 3.1: Resultados execução da Fase 1

	Etapa 1	**Etapa 2**	**Etapa 3**
Fontes	**Estudos retornados**	**Estudos relevantes**	**Pós - Exclusão CE1 / CE2 / CE3**
ACM	2773	1932	28
IEEE xplorer	124	85	53
ScienceDirect	137	109	85
Springer Link	256	183	23
Total	3290	2309	189

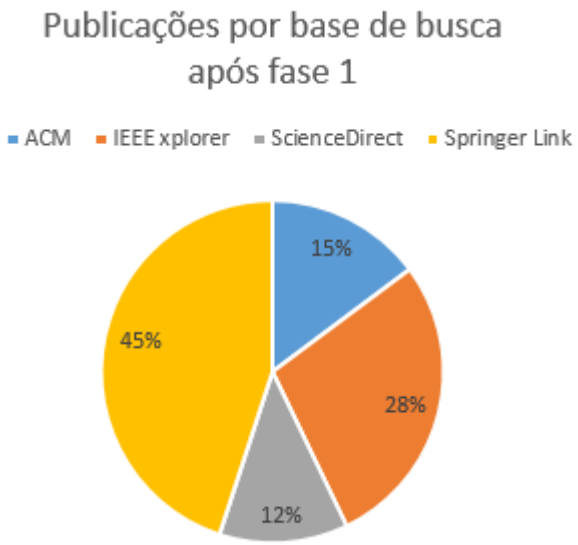

Figura 3.2: Resultado por base em porcentagem - Fase 1

Tabela 3.2: Resultados execução da Fase 2

	Fase 1	**Fase 2**
Fontes	**Estudos selecionados**	**Pós - Exclusão CE4 / CE5**
ACM	28	0
IEEE xplorer	53	6
ScienceDirect	85	2
Springer Link	23	3
Total	189	11

O número de instâncias pode evidenciar também a necessidade de uma mudança na lógica implementada para solucionar o problema, pois é possível a detecção de um possível gargalo monitorando esse número, posto que pode existir uma quantidade desnecessária de instâncias para dada transação.

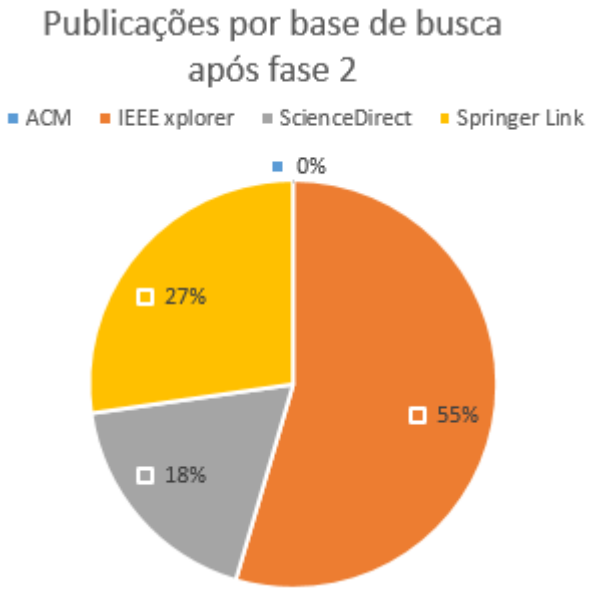

Figura 3.3: Resultado por base em porcentagem - Fase 2

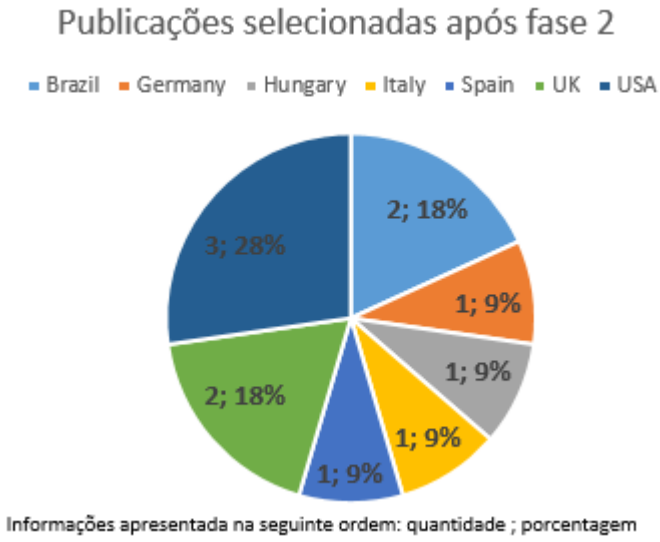

Figura 3.4: Resultado dos estudos selecionados por país - Fase 2

O estudo [37] não apresenta métricas de monitoramento somente disserta sobre uma abordagem.

O estudo [65] evidencia a métrica de cobertura estrutural de código por meio de Decisão modificada / Condição de Cobertura (MC / DC).

Esta métrica busca realizar uma espécie de cobertura de código como é feito na análise estática. Para isso, um conjunto de dados são submetidos de tempos em tempos para verificar se as respostas do sistema estão adequadas.

Com essa métrica é possível detectar erros/falhas em determinadas situações em que o

Tabela 3.3: Estudos Selecionados

E1, Jonathan Stuart Ward, Adam Barker. Cloud cover: monitoring large-scale clouds with Varanus	E5, Michael W. Whalen, Suzette Person, Neha Rungta, Matt Staats . A Flexible and Non-intrusive Approach for Computing Complex Structural Coverage Metrics	E9, Lingyun Yang, J. M. Schopf, C. L. Dumitrescu, I. Foster. Statistical data reduction for efficient application performance monitoring
E2, Bikram Sengupta , Nilanjan Banerjee, Chatschik Bisdikian, Paul Hurley. Tracking transaction footprints for non-intrusive end-to-end monitoring	E6, Tibor Gyimothy. To Use or Not to Use? The Metrics to Measure Software Quality (Developers' View)	E10, Giuseppe Aceto , Alessio Botta Walter de Donato, Antonio Pescapè. Cloud monitoring: A survey
E3, Eric Larson. SUDS: an infrastructure for creating dynamic software defect detection tools	E7, Otto Julio Ahlert Pinno, Sand Luz Correa, Aldri Luiz dos Santos, Kleber Vieira Cardoso. Using Partial Correlation for Effective Metric Selection and Anomaly Detection in Multi-tier System	E11, Heiko Koziolek. Performance evaluation of component-based software systems: A survey
E4, Daniel Cabrero, Javier Garzas, Mario Piattini. Software Artifact Prioritization based on the Frequency of Use	E8, Rafael Pereira, Stephano Goncalves, Lisane Brisolara, Julio C. B. Mattos. Java Code Analyser for Estimating Embedded Software Efficiency	

software está submetido como, por exemplo, a verificação do comportamento do sistema sobre um dado número de requisições ou o uso abusivo de um determinado método.

O estudo [27] apresenta a métrica por frequência de uso do método, que verifica a quantidade de solicitações realizada em cima de cada método do sistema.

Esta métrica permite um mapeamento dos métodos utilizados com frequência, possibilitando uma priorização destes para um possível manutenção. A detecção de gargalos no sistema também é facilitada por essa métrica uma vez que uma má distribuição na frequência de uso dos métodos pode onerar alguns serviços.

O estudo [2] menciona a métrica por número de invariantes (correlações estáveis) detalhada por [33].

O número de invariantes se refere ao número de vezes que um sistema apresenta um comportamento esperado para o usuário, porém ele gerou inconsistências para a realização dessa atividade. Um exemplo seria a realização de uma autenticação, na qual o usuário submete o formulário várias vezes ao clicar no botão de autenticar mais de uma vez sem perceber que já estava carregando.

Para o usuário o comportamento do sistema é transparente e realizará a autenticação de forma correta, mas o sistema pode exibir em seus logs mensagens de falha como: usuário

já autenticado, autenticação não foi realizada por já estar em uso. Essas invariantes possibilitam visualizar o comportamento do uso de determinadas funções do sistema.

O estudo [47] apresenta referência a **métrica de consumo de recursos físicos**, esta realiza a contabilização dos consumos físicos verificando custo de execução de cada instrução.

Monitorar os recursos físicos da máquina permite obter dados do comportamento de consumo do sistema e como cada funcionalidade faz uso dos recursos do servidor. Esses dados podem evidenciar um mau uso de recursos pelo sistema ou também demonstrar recursos ociosos desnecessários para aquela aplicação.

O estudo [66] relata sobre as métricas de porcentagem de utilização da unidade central de processamento (CPU) no nível do usuário, percentagem de utilização da CPU no nível do sistema e porcentagem de tempo que a CPU estava ociosa.

O uso da CPU é um dos indicadores que pode alertar uma sobrecarga no sistema. É complementar aos recursos físicos e uma das métricas mais básicas de monitoramento, pois vários sistemas operacionais disponibilizam a verificação dessa métrica, sem necessidade de nenhuma implementação a mais.

As métricas encontradas podem ser analisadas de forma separadas levando a um conjunto de informações a se avaliar durante a execução de um sistema, porém elas podem ser mescladas e utilizadas de forma complementar, além de analisar os dados levantados de forma histórica.

QP2. Quais abordagens são utilizadas para monitoramento de sistemas em execução?

O estudo [57] apresenta técnicas da abordagem por monitoramento de rastros em logs de transação.

O estudo [37] disserta sobre a abordagem de detecção dinâmica de defeitos de software relacionadas com a entrada por meio de instrumentação de código com software apropriado.

O estudo [65] evidencia MC / DC que requer que cada ponto de entrada e saída no programa tenha sido invocado pelo menos uma vez, cada condição (uma expressão booleana que não contenha operadores booleanos) em uma decisão no programa tenha feito em todos os possíveis resultados, pelo menos uma vez e que cada condição tenha sido mostrada para afetar independentemente o resultado da decisão.

O estudo [27] apresenta a abordagem de priorização baseado na frequência de uso que mantém o foco na eliminação de artefatos do sistema baseado no uso, porém se faz de interesse pela abordagem possibilitar a priorização para manobras de manutenção no sistema legado.

O estudo [2] menciona uma abordagem por monitoramento baseado em correlação que subdividi-se em outras abordagens

1. Por Regressão linear;
2. Por Árvore de geradora mínima;
3. Por Correlação parcial.

O estudo [47] detalha sobre uma abordagem por perfis de execução (profiles), que gera perfis de execução baseados em logs.

No total foram encontrados relatos de 6 métricas e 6 abordagens distintas nos estudos selecionados, conforme apresentado nas Tabelas 3.4 e 3.5.

Tabela 3.4: Métricas encontradas - Q1

M1. Número de instâncias por transações	M2. Cobertura estruturais de código - MC/DC
M3. Frequência de uso do método	M4. Número de invariantes
M5. Consumo de recursos físicos	M6. Porcentagem de utilização da CPU

Tabela 3.5: Abordagens encontradas - Q2

A1. Monitoramento de rastros em logs de transação	A2. Detecção dinâmica de defeitos por instrumentação de código
A3. Decisão modificada / Condição de Cobertura	A4. Priorização na frequência de uso
A5. Monitoramento por correlação	A6. Perfis de execução (profiles)

3.3 Síntese do Capítulo

O capítulo apresenta um conjunto de métricas para monitoramento de sistemas computacionais em tempo de execução que foram obtidas por meio de um mapeamento sistemático. Essas métricas são um leque de opções para monitoramento em tempo de execução, representando várias abordagens para diferentes domínios.

Capítulo 4

Priorização das Métricas

O mapeamento sistemático possibilitou o conhecimento das métricas e abordagens que dão suporte ao monitoramento de sistemas. Essa listagem de resultados auxilia nas decisões futuras de uma camada de acompanhamento dos sistemas, permitindo ajustes que viabilizam maior agilidade nas correções de possíveis anomalias, levando a uma redução de custos e tempo na manutenção dos sistemas utilizados pelas organizações e que não podem ter seus serviços interrompidos, acarretando em enormes prejuízos financeiros caso um de seus serviços fiquem inacessíveis (fora do ar).

Com a importância de manter o CPD no seguimento da melhoria da qualidade de software, as métricas de monitoramento vão ao encontro para respaldar as manutenções realizadas em seus sistemas. Porém, por causa da complexidade de implantação dessas métricas e disponibilidade de recurso para o desenvolvimento dessa atividade, priorizar e ter estabelecido o que será realmente implantado é de extrema importância.

O intuito desse capítulo foi verificar a relevância para o CPD quanto as métricas de monitoramento de software em tempo de execução que foram levantadas no mapeamento sistemático com o propósito de disponibilizar uma lista de priorização dessas métricas e complemento destas com outras necessidades do CPD.

4.1 Técnica de Pesquisa

A entrevista é uma técnica bastante adequada para a obtenção de informações sobre o que as pessoas sabem, acreditam, sentem, esperam, pretendem fazer, fazem ou fizeram [56].

Os dados obtidos na entrevista são suscetíveis de classificação e de quantificação [24], o que possibilita uma análise qualitativa para a classificação das métricas.

A fim de suprir algumas desvantagens conforme apresentadas por Gil [24], foi explicitado cada métrica e o intuito do estudo aos entrevistados, o fator desinteresse não foi

verificado, visto que o domínio faz parte do trabalho dos entrevistados e era de interesse deles uma melhora no monitoramento dos sistemas e evitou-se interferências externas durante as entrevistas.

A entrevista foi definida como uma entrevista estruturada. Ela se dá por uma relação fixa de perguntas, cuja ordem e redação permanece invariável para todos os entrevistados [24]. Por possibilitar o tratamento quantitativo dos dados, este tipo de entrevista foi o mais adequado a obtenção dos resultados.

Foram definidas 6 perguntas fechadas a fim de verificar o quanto a métrica é interessante ao CPD, uma pergunta para cada métrica levantada no mapeamento sistemático. Duas perguntas abertas foram criadas a fim de promover a ordem de prioridade para a implementação das métricas e uma pergunta para sugestão de outras métricas que não foram contempladas no estudo.

Para as perguntas fechadas foi definido a utilização da escala Likert [41] que possui uma elaboração mais simples não medindo o quanto uma atitude é mais ou menos favorável. Essa escala possibilita verificar o quanto uma métrica é interessante para o CPD variando em uma escala de 1 a 5 em que 1 é nada interessante e 5 é muito interessante.

Para as perguntas abertas deixou-se disponível para o entrevistado colocar o que lhe fosse conveniente sem interferência em sua opinião.

4.2 A Entrevista

A aplicação da entrevista foi realizada com 7 participantes que representam as equipes envolvidas tecnicamente com os sistemas, já que estes são os responsáveis por realizar tanto a implementação quanto o acompanhamento dos sistemas da Universidade de Brasília (UnB).

O perfil de cada participante é apresentado na Tabela 4.1, onde foi verificado a escolaridade que teve como opção "Nível Médio", "Técnico", "Graduando", "Graduado", "Pós-Graduado", "Mestrado"e "Doutorado", a experiência com desenvolvimento de software em anos e o tempo de trabalho na UnB, este com intenção de verificar o tempo de experiência com os sistemas da UnB.

Tabela 4.1: Perfil dos Participantes da Pesquisa

Participante	**Escolaridade**	**Experiência**	**Tempo de trabalho**
P1	Graduando	5	2
P2	Graduando	3	2
P3	Pós-Graduado	15	5
P4	Pós-Graduado	10	5
P5	Graduando	7	6
P6	Pós-Graduado	5	1
P7	Mestrado	15	5

Foi possível observar que entre as equipes de desenvolvimento do CPD existe uma diferença de tempo de casa e tempo de experiência com desenvolvimento de software, porém, os sistemas da UnB iniciaram uma frente de modernização e refatoração no ano de 2015, o que faz com que todos os participantes tenham conhecimento prévio dos sistemas e de seus devidos problemas e deficiências.

O nível de escolaridade não atingiu nenhum participante com nível médio, técnico ou doutorado. Todos os participantes que responderam como Graduando, são estudantes de cursos da UnB e já trabalhavam de forma indireta para a Instituição antes de entrar como funcionário efetivo do Centro de Informática.

Os dados levantados por meio da entrevista foram consolidados em 6 gráficos, uma lista de priorização para a implementação das métricas e uma lista com métricas sugeridas pelos entrevistados.

A avaliação foi realizada referente as métricas encontradas no mapeamento sistemático, conforme apresentado na Tabela 4.2.

Tabela 4.2: Lista de Métricas Encontradas

M1. Número de instâncias por transações	M2. Cobertura estrutural de código-MC/DC
M3. Frequência de uso do método	M4. Número de invariantes
M5. Consumo de recursos físicos	M6. Porcentagem de utilização da CPU

As métricas M1, M4 e M5 apresentaram uma média na escala de 4,29, sendo as métricas com maior número de votos na pesquisa 5, com 4 de 7.

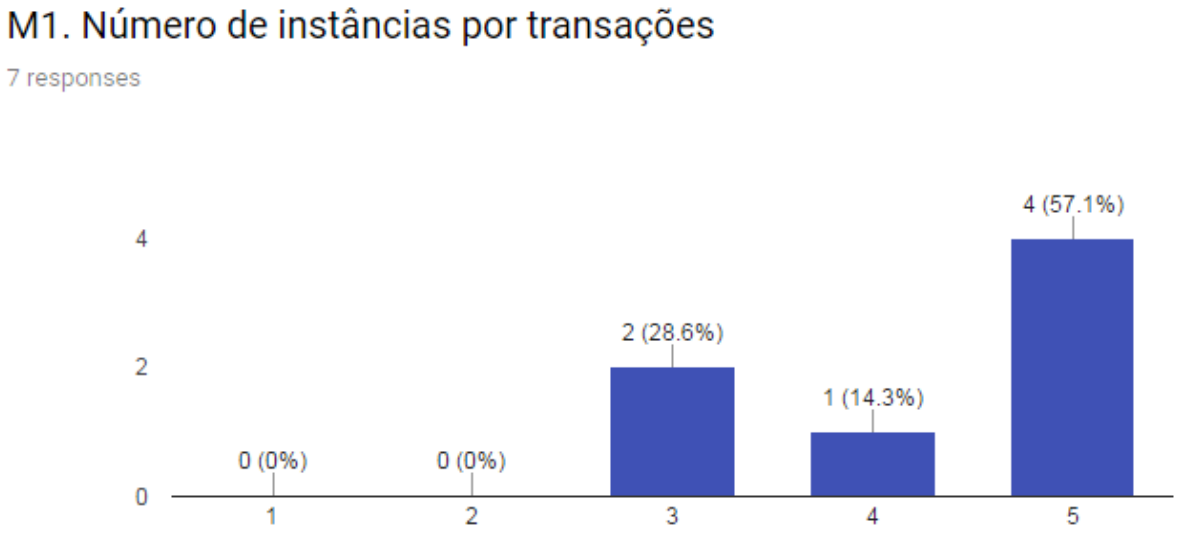

Figura 4.1: Respostas Métrica 1

A métrica M2 obteve a menor média entre as métricas, ficando com 3,42. Esta apresenta uma distribuição dos votos em todas as categorias o que evidencia uma métrica de difícil aceitação, isso se deve ao fato dos participantes enxergarem como uma boa métrica, porém com alta complexidade de implementação o podendo onerar o desempenho dos sistemas.

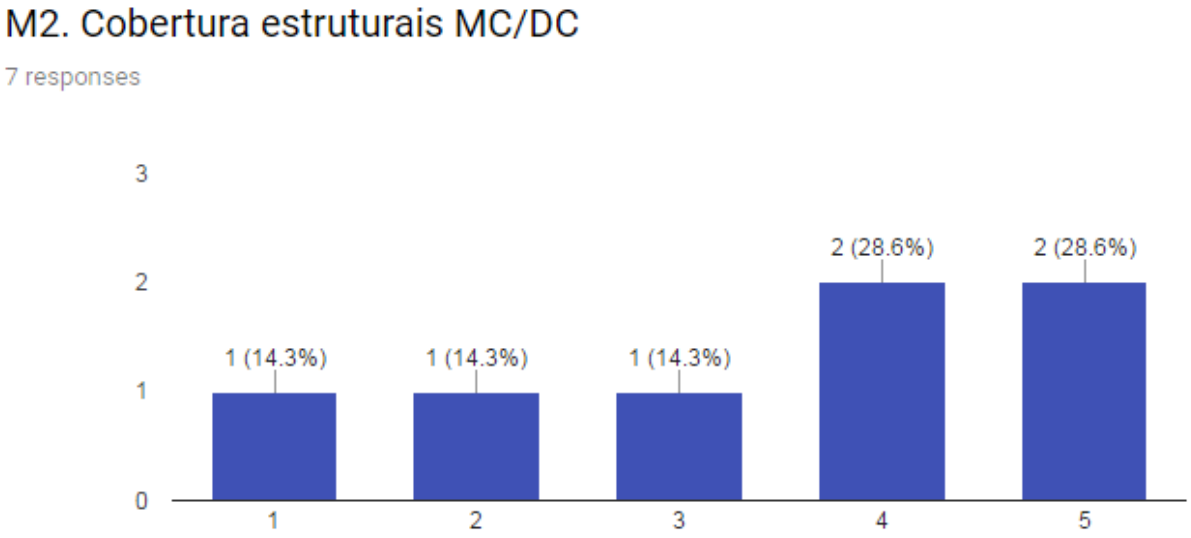

Figura 4.2: Respostas Métrica 2

A Métrica M3 teve uma média 4,29 e possui a distribuição de votos mais uniforme ficando com 6 votos entre 4 e 5 pontos.

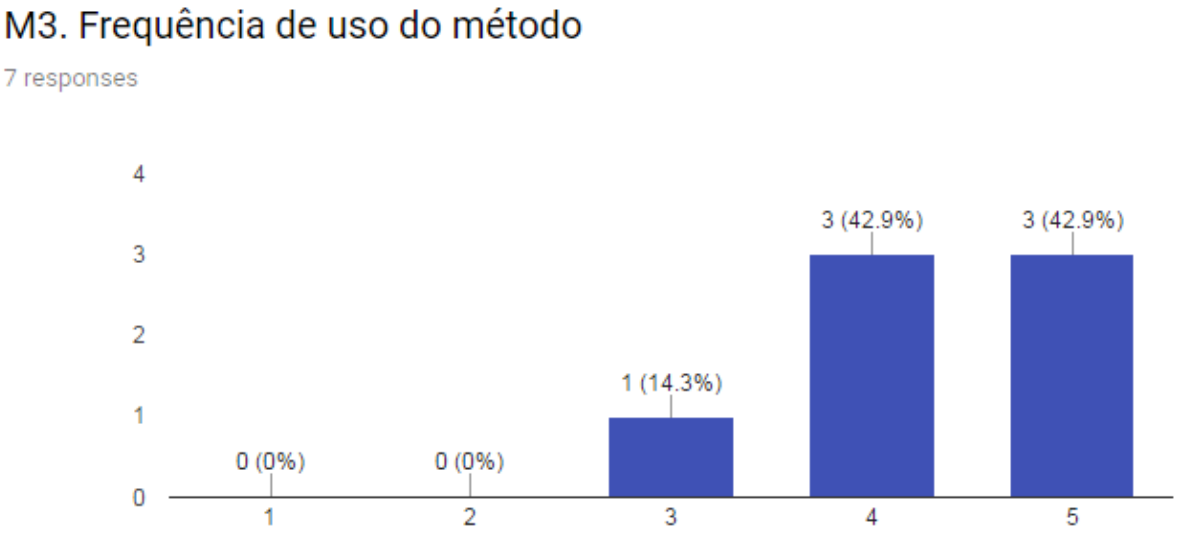

Figura 4.3: Respostas Métrica 3

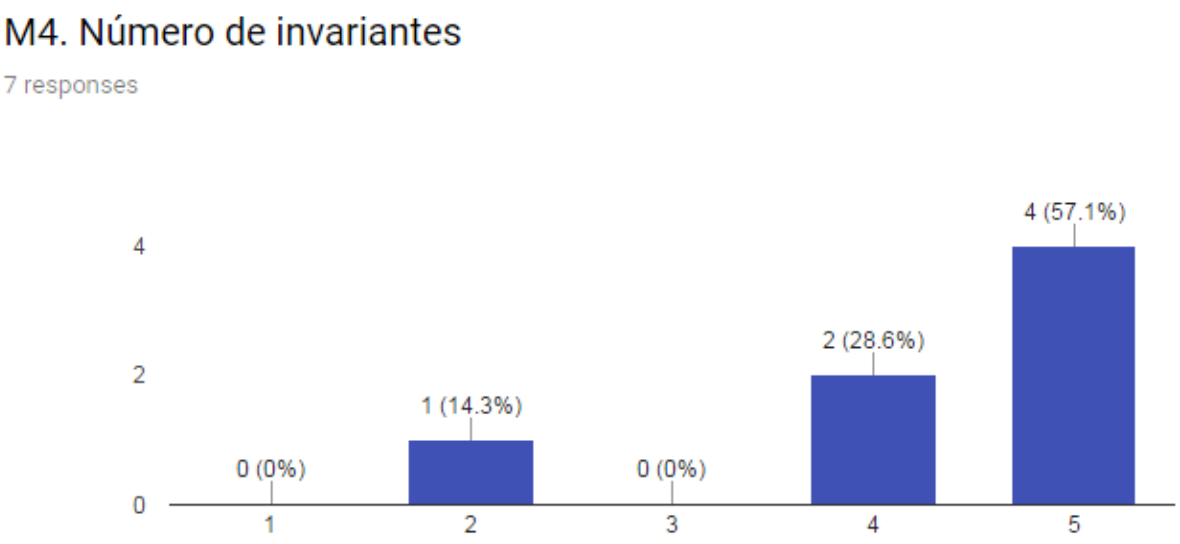

Figura 4.4: Respostas Métrica 4

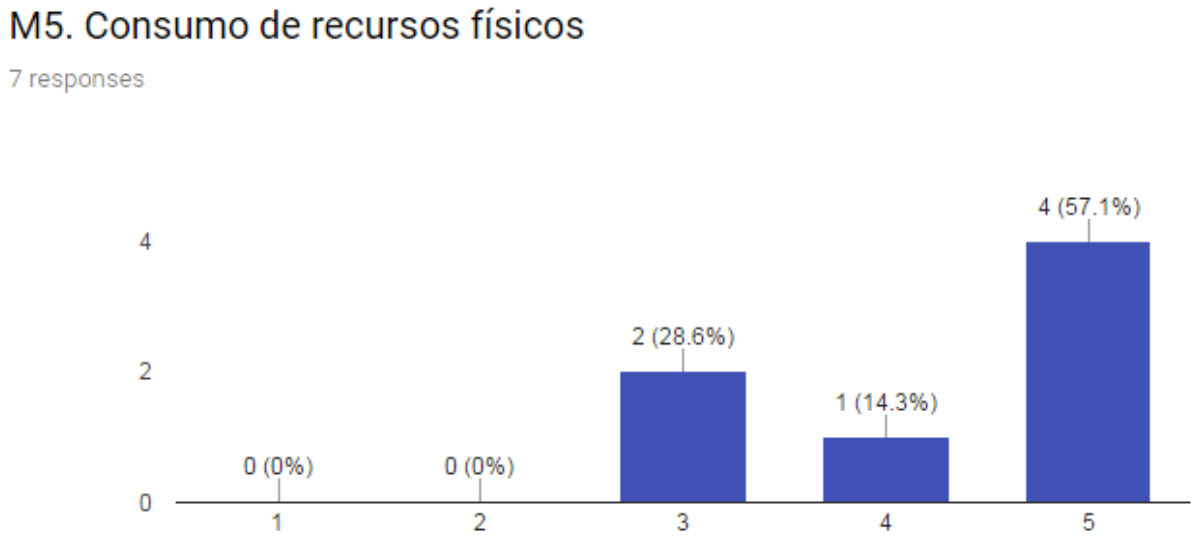

Figura 4.5: Respostas Métrica 5

A métrica M6 possui a segunda pior média com 3,86 apesar de ter seus votos concentrados em 3,4 e 5, isso ocorre em razão de ser uma métrica básica, apesar de importante, e que foi considerada de baixa complexidade para implementação ou que já deveria estar em uso.

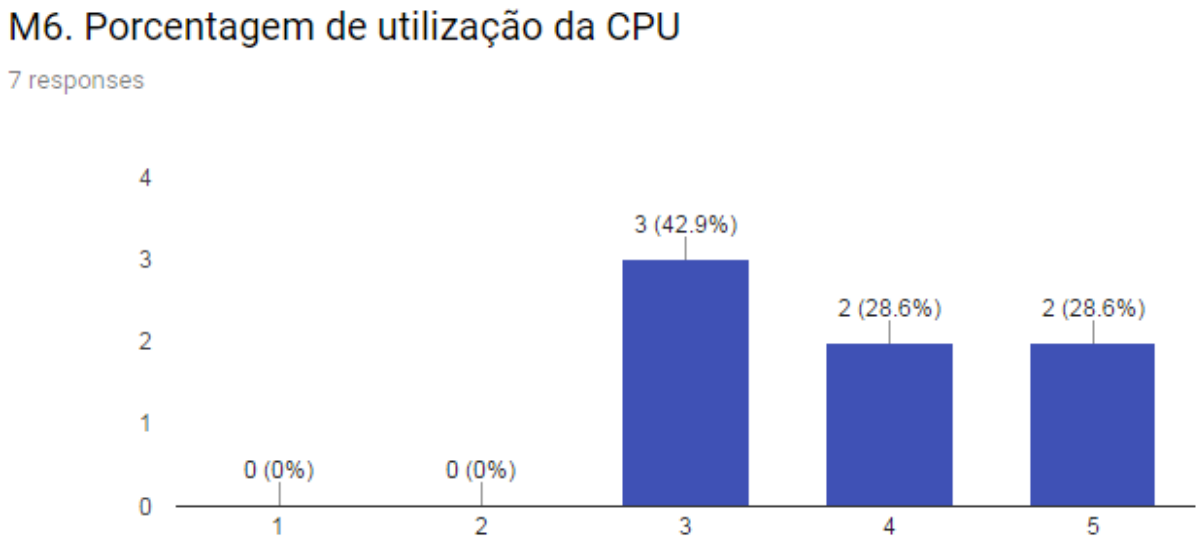

Figura 4.6: Respostas Métrica 6

Para a criação da lista de prioridade foi solicitado que o participante elencasse as métricas da mais importante para a menos importante gerando uma lista individual de acordo com a perspectiva pessoal.

A lista geral de priorização das métricas foi montada utilizando a quantidade que aquela métrica foi votada em dada posição multiplicado por um fator de pontuação dado pela sequência Fibonacci, similar ao uso para estimar esforço no planning poker [53], já que este não correlaciona o crescimento de um valor ao outro, não linear, tornando assim os pesos distribuídos de forma não determinada [35]. Portanto, cada prioridade de 1 a 6 recebeu um valor correspondente na escala Fibonacci, iniciando em ordem decrescente de prioridade conforme apresentado na Tabela 4.3.

Tabela 4.3: Pontuação por Prioridade

Prioridade	**Pontos por voto (Escala Fibonacci)**
6	1
5	1
4	2
3	3
2	5
1	8

O número de votos que cada métrica recebeu em determinada prioridade é exemplificado na Tabela 4.4

Tabela 4.4: Votação de Prioridade por Métrica

Métricas	**Prioridade**					
	1	**2**	**3**	**4**	**5**	**6**
M1	1	2	1	1	1	1
M2	1	0	1	0	2	3
M3	1	3	2	1	0	0
M4	2	1	3	0	0	1
M5	2	1	0	1	3	0
M6	0	0	0	4	1	2

A distribuição das prioridades com o fator de pontuação ficou de acordo com a Tabela 4.5. O Valor em cada prioridade é dado pelo número de votos que a métrica recebeu, conforme apresentado na Tabela 4.4, multiplicado pela pontuação que cada prioridade tem na escala Fibonacci, conforme Tabela 4.3.

Tabela 4.5: Distribuição da Priorização das Métricas

Métricas	**Prioridade**					
	1	**2**	**3**	**4**	**5**	**6**
M1	1	2	1	1	1	1
M2	1	0	1	0	2	3
M3	1	3	2	1	0	0
M4	2	1	3	0	0	1
M5	2	1	0	1	3	0
M6	0	0	0	4	1	2

A priorização de implantação das métricas ficou da seguinte forma:

1. M3 - Frequência de uso do método;
2. M5 - Consumo de recursos físicos;
3. M1 - Número de instâncias por transação;
4. M4 - Número de invariantes;
5. M6 - Porcentagem de utilização da CPU;
6. M2 - Cobertura estruturais de código - MC/DC.

Podemos observar que a lista resultante está em consonância com a média do nível de interesse de cada métrica, já que M2 obteve a menor média de 3,42 juntamente com M6 com 3,86. As métricas M1, M3, M4 e M5 que possuem médias iguais também se comportaram de forma esperada na priorização, um exemplo é que M3 que está em primeiro possui o maior quantitativo de notas 5 e 4 e uma distribuição das notas uniforme, sem variar entre não é interessante a muito interessante.

A questão aberta para sugestão de outras métricas de monitoramento em tempo de execução foi uma abordagem interessante para evidenciar novas oportunidades e um interesse em comum dos entrevistados. Essa questão foi conduzida sem nenhuma intervenção e sem explicitar os resultados já levantados de outros participantes e mesmo assim 4 pessoas sugeriram a mesma métrica para verificação. Dois participantes não responderam a essa pergunta. A lista ficou da seguinte forma:

Tabela 4.6: Lista de Métricas Sugeridas

Métricas Sugeridas	**Quantidade de Sugestões**
Quantidade de conexões recusadas com o banco de dados	4
Tempo médio de uma transação	1
Tempo em cache de um serviço	1

A entrevista serviu para evidenciar uma necessidade da equipe referente ao monitoramento das transações junto ao banco de dados e não somente do estado do sistema de forma individual.

Outro fator importante foi levantar uma nova lista de métricas, evidenciadas na Tabela 4.6, proposta pelos avaliados o que resultou em uma reflexão sobre a área crítica a ser monitorada. Essa lista possibilitou verificar que a conexão com o banco de dados dos sistema é um ponto importante a ser monitorado pois foi ressaltado por 3 entrevistados e questionado por 1 entrevistado.

4.3 Síntese do Capítulo

Este capítulo apresentou uma entrevista que auxiliou na definição de uma lista de prioridade para implantação das métricas no CPD, além de averiguar o quão estas métricas estão correspondendo as expectativas das equipes para monitoramento dos sistemas.

Assim sendo, obter esses dados apoiou na implantação das métricas e priorização das atividades no CPD, evitando gastos de recursos com apropriação de tecnologias que não serão necessárias de acordo com a problemática atual dos sistemas legados.

Capítulo 5

Serviço de Monitoramento

O capítulo 4 possibilitou a geração de uma lista de priorização das métricas encontradas na literatura. Visando seguir essa lista, o foco de implementação se deu nas 3 primeiras métricas e como a métrica Porcentagem de utilização da CPU é complementar a métrica Consumo de recursos físicos também foi implementada, obtendo assim um serviço de monitoramento que contempla 4 métricas evidenciadas no mapeamento sistemático.

As métricas Número de invariantes e Cobertura estruturais de código - MC/DC não foram selecionadas, pois estas obtiveram um número muito inferior na lista de priorização e os entrevistados declararam que não seria interessante nesse primeiro momento, porque poderia onerar os serviços já em produção devido a característica de abordagem de implementação dessas métricas.

Com a importância da métrica **Quantidade de conexões recusadas com o banco de dados** sugerida pelos desenvolvedores, ela foi repensada e entrou para a lista de monitoramento. Como o resultado da conexão é demonstrado no status do serviço quando este apresenta erro, foi estabelecido uma métrica de verificação de erros pelo status do serviço. Essa nova métrica M5 possibilita visualizar o quantitativo dos erros gerados em qualquer nível da arquitetura disponibilizada para a execução do serviço.

A fim de facilitar o acompanhamento das métricas selecionadas, elas foram apresentadas em ordem alfabética e com uma nova numeração distinta dos capítulos 3 e 4. Assim sendo, as métricas que foram implementadas e avaliadas são:

- **M1**. Consumo de recursos físicos;
- **M2**. Frequência de uso do método;
- **M3**. Número de instâncias por transação;
- **M4**. Porcentagem de utilização da CPU;
- **M5**. Quantitativo de erros por serviço.

Para implementação do serviço de monitoramento, as métricas foram agrupadas por similaridade de abordagem em 2 grandes grupos a fim de diminuir o impacto na performance do barramento.

As métricas M1 e M4 foram associadas a uma abordagem de verificação direta ao barramento, pois a linguagem Erlang mantém um conjunto de funções para monitoramento do sistema operacional, que são dados essenciais para o bom funcionamento de seus processos, necessitando apenas de tratamento das informações obtidas e criação de funções para resgate dos dados em memória.

As métricas M2, M3 e M5 foram associadas a uma abordagem de análise de log, uma vez que a verificação dessas métricas no momento em que o serviço é requisitado ao sistema iria ocasionar em um alto custo de desempenho, e devido ao domínio sazonal de uso dos sistemas no CPD, esse tratamento não poderia ser aceito.

5.1 Verificação Direta ao Barramento

A linguagem Erlang mantém o monitoramento do sistema operacional e disponibiliza esses dados por meio de duas aplicações sendo Observer e os_mon.

O Observer é uma ferramenta gráfica para observar as características dos sistemas Erlang. Pode ser iniciado pelo comando observer:start() e exibe informações do sistema, processos, uso da memória, além de conter uma frente para o rastreamento Erlang, a tela inicial da ferramenta pode ser observada na Figura 5.1.

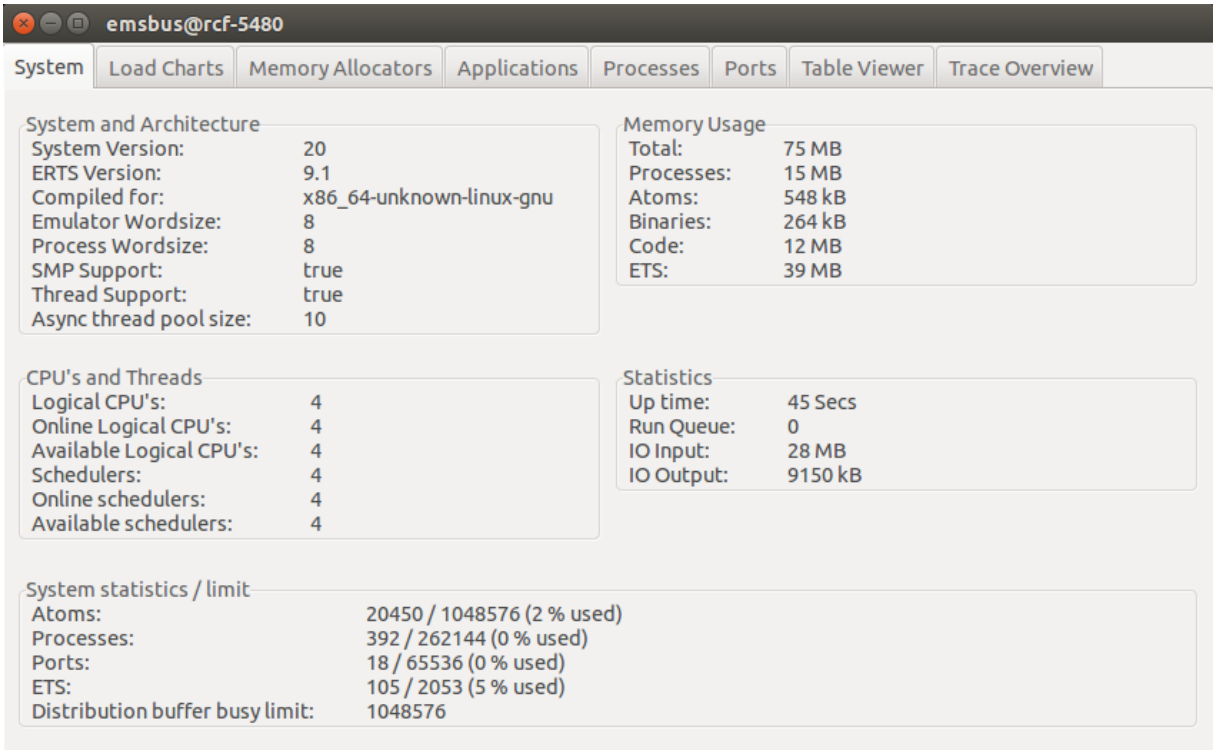

Figura 5.1: Ferramenta Observer - Monitoramento do Sistema

A ferramenta exibe dados interessantes para as métricas M1 e M4 conforme apresentado na Figura 5.2, porém os fornece de forma gráfica e devido a gama de informações referente aos processos do Erlang, esta é muito onerosa e a própria documentação recomenda uma execução parcial em um nó autônomo.

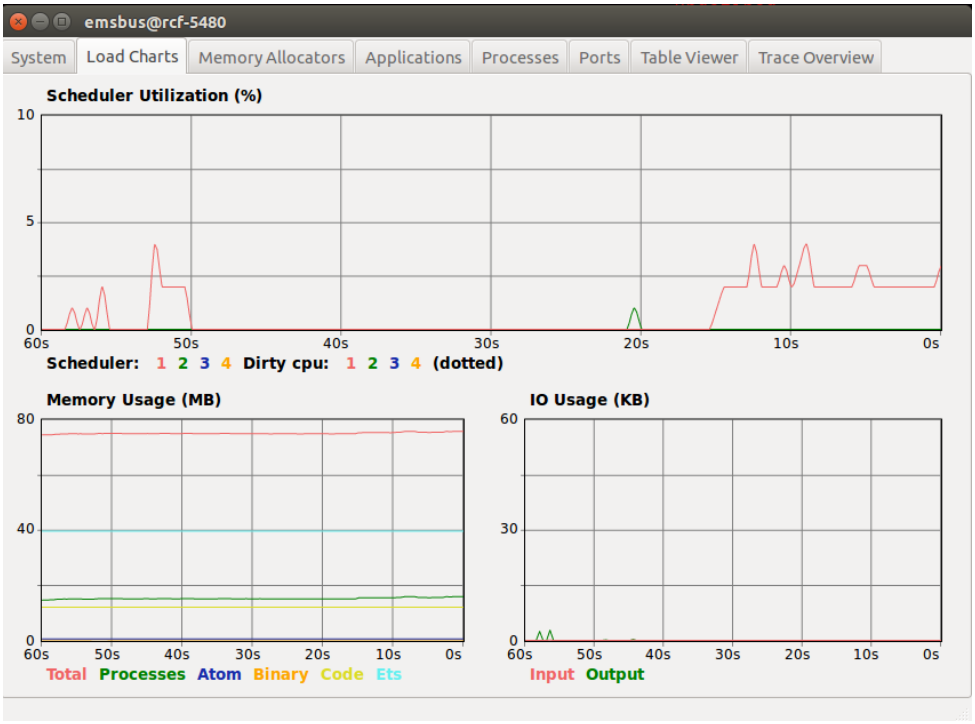

Figura 5.2: Ferramenta Observer - Gráficos de carga

A outra aplicação de monitoramento Erlang é a os_mon que fornece os seguintes serviços:

- Cpu_sup Monitoramento de carga e utilização da CPU (Unix);
- disksup Monitoramento de disco (Unix, Windows);
- memsup Monitoramento da memória (Unix, Windows, VxWorks);
- OS_sup Interface para mensagens do sistema operacional (Solaris, Windows).

Os serviços só podem ser consumidos em sistemas operacionais específicos, conforme indicado entre parênteses a cada serviço.

Para atender as métricas M1 e M4 foi utilizado somente o serviço cpu_sup que é um processo que contém as funções avg10, avg50 e avg150 que podem ser usadas para recuperar valores de carga do sistema e as funções util0 e util1 que podem ser usadas para recuperar os valores de utilização da CPU.

A primeira métrica a ser atendida foi a M4 utilizando a função util1, na qual a utilização da CPU é definida como a soma das porcentagens de compartilhamento dos ciclos de CPU usados em todos os estados ocupados do processador e afim de especificar separadamente cada CPU foi utilizado o parâmetro per_cpu.

Os dados obtidos por meio da função util1 foram tratados utilizando o padrão de correspondência a fim de obter somente os dados adequados para a disponibilização do serviço conforme Figura 5.3.

```
cpus(Request) ->
        ContentData = [{lists:concat(['N',K]),U} || { K, U, _, _} <- cpu_sup:util([per_cpu])],
        io:format("is ~p\n", [ContentData]),
        {ok, Request#request{code = 200,
                                                response_data = ems_schema:to_json(ContentData)}
        }.
```

Figura 5.3: Função de obtenção do uso de todas as CPUs

A quantidade de CPUs são definidas pelo número de *threads* em que o Erlang foi instanciado e é em torno desse número que as funções estão definidas. Para complementar a métrica também foi especificado a média de uso das CPUs e o uso de acordo com a distribuição por Kernel, Usuário, Em espera e Ocioso implementado, conforme apresentado na Figura 5.4.

```
cpudetailed(Request) ->
        ContentData = lists:nth(1,[[X,Y,Z,W] || {_,[_,_,X,_,Y],[_,Z,W],_} <- [cpu_sup:util([detailed])]]),
        io:format("is ~p\n", [ContentData]),
        {ok, Request#request{code = 200,
                                              response_data = ems_schema:to_json(ContentData)}
        }.
```

Figura 5.4: Função de obtenção do uso da CPU de forma distribuída

A métrica M1 - Consumo de recursos físicos, foi atendida por meio de serviços que especificam o uso da memória, processos, tarefas e o tempo de execução do barramento, complementando assim, os dados obtidos do uso das CPUs.

Esse conjunto de dados que as métricas M1 e M4 levantam, tem como objetivos para o monitoramento verificar a situação de alguns recursos físicos da máquina em que o barramento está iniciado, servir como dados complementares as outras métricas, já que ao monitorar determinado serviço ou período sazonal, podemos verificar o comportamento deste diante dos recursos físicos disponíveis, além de tomada de decisão preventiva caso haja um consumo elevado durante uma fase crítica do processo, em que esses dados podem indicar uma necessidade de aumento dos recursos físicos e como em geral alguns sistemas se encontram em máquinas virtuais há uma facilidade na escalabilidade de recursos.

5.2 Análise de Log

As métricas M3. Número de instâncias por transação, M2. Frequência de uso do método e M5. Quantitativo de erros por serviço, podem ter diversas abordagens para obtenção dos seus dados, como por exemplo:

- Análise de log;
- Armazenamento das requisições;
- Instrumentação de código.

Essas métricas fornecem dados essenciais para o monitoramento, porém se verificada de forma incorreta podem onerar o tempo de resposta do servidor, causando lentidão no uso. A fim de obter melhor custo de monitoramento x desempenho, foi feita uma análise diante os recursos disponibilizados pelo CPD e a arquitetura proposta do barramento Erlangms.

A instrumentação de código pode ser feita de forma manual ou por meio do uso de softwares específicos para esse fim. A inclusão de instruções no código-fonte manualmente necessita recompilação repetida antes da execução, o que acrescenta esforço e tempo ao processo de depuração [3]. Outro fator é o tempo de aprendizado dos programadores memorizaem as novas instruções e fazerem o uso adequado no código-fonte, causando também uma nova nomenclatura distinta da padrão utilizada o que pode levar a uma maior dificuldade na manutenção desse código.

O uso de softwares para instrumentação de código, já define uma sintaxe e documentação específica, o que auxilia na fase de implantação e manutenção do código. Não obstante, o custo dessa implantação é alto já que o CPD depende de uma série de políticas para uso de uma nova ferramenta e o tempo de treinamento da equipe para iniciar a obtenção dos dados.

O armazenamento das requisições retira o tempo de implantação de uma solução por instrumentação de código e obtém os dados para as métricas de forma imediata. Essa abordagem tem uma baixa complexidade de implementação, sendo necessário apenas um método de inserção nos serviços já existentes ou no método implementado no *back-end*.

O que ocorre com a solução dessa abordagem é o custo de desempenho ao barramento, posto que uma inserção de dados é realizada a cada requisição, diminuindo o tempo de resposta do serviço e diante o domínio do CPD que possui uma demanda em larga escala em períodos sazonais, poderia levar a uma lentidão em todo o barramento.

Diante dessas abordagens e o domínio do CPD foi então proposta a obtenção das métricas por análise de log. O custo de escrita no log é um custo já existente e necessário para o barramento, logo este não foi alterado. A fim de evitar onerar o barramento com a

análise do *log*, foi realizado uma abordagem de captura do arquivo de *log*, onde o sistema atualiza o *log* para análise de acordo com um tempo definido.

A realização dessa solução foi elaborada utilizando o conjunto de ferramentas Elastic Stack que é um projeto *open source* para armazenamento de dados que garante sua confiabilidade vindo de qualquer fonte, em qualquer formato e possibilita procurar, analisar e visualizar os dados em tempo real.

O conjunto de produtos utilizados do Elastic Stack são representados na Figura 5.5

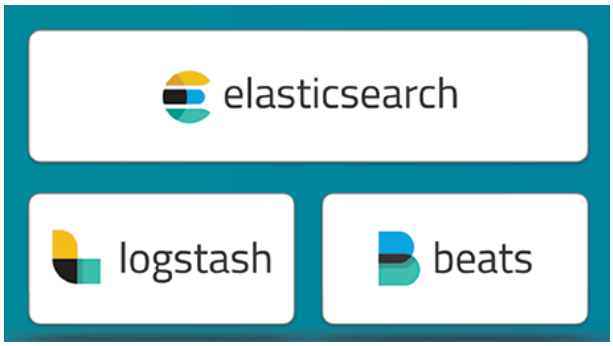

Figura 5.5: Arquitetura de funcionamento do Elastic [61]

Beats é uma plataforma de carregadores de dados que enviam dados das máquinas de borda para o Logstash e Elasticsearch.

O **Logstash** é um pipeline de processamento de dados do lado do servidor que ingere dados de uma infinidade de fontes ao mesmo tempo e os transformam para, em seguida, enviar para o Elasticsearch.

O **Elasticsearch** é o mecanismo de pesquisa e análise distribuído, baseado em JSON, projetado para escalabilidade horizontal, máxima confiabilidade e fácil gerenciamento dos dados.

Utilizando-se dessa tecnologia foi possível realizar a captura do *log* por meio de um Filebeat que executa essa tarefa de 5 em 5 segundos, verificando se há mudanças no *log* e atualizando somente o novo conteúdo. O Filebeat é um dos tipos de *beat* que o produtos Beats oferece, sendo esse o mais adequado para a captura de log do barramento. O arquivo de configuração do Filebeat foi definido em 3 partes, conforme apresentado na Figura 5.6.

```
1 filebeat.config.prospectors:
    path: ./*.yml
    reload.enabled: true
    reload.period: 10s

2 filebeat.prospectors:
  - input_type: log.
    paths:
      - /home/user/ems-bus/priv/log/*/*/*/*.log
    scan_frequency: 5s
    exclude_lines: ["DEBUG"]
    include_lines: ['[A-Z]+ [0-9]{2}\/[0-9]{2}\/[0-9]{4} [0-9]{2}\:[0-9]{2}\:[0-9]{2}  [A-Z]+ http']
    multiline.pattern: '[A-Z]+ [0-9]{2}\/[0-9]{2}\/[0-9]{2}'

3 output.logstash:
   hosts: ["localhost:5044"]
```

Figura 5.6: Definição do arquivo Filebeat

O conjunto de instruções 1 define a verificação de tempos em tempos do(s) arquivo(s) de configuração do Filebeat, pois permite alteração em tempo de execução que passará a valer a partir da próxima verificação.

O conjunto 2 é parte lógica do Filebeat que representa os locais a serem verificados na instrução *paths*, a frequência de verificação da mudança do arquivo é definida por *scan_frequency*. No conjunto 2 também é definido por *regex* as linhas que se devem capturar, evitando assim um conjunto de dados que não serão utilizados. Como essa solução busca atender as métricas M2, M3 e M5, foram retiradas qualquer linha referente a DEBUG e capturadas somente as instruções que se referem a requisição e resposta de um serviço.

Por último, o conjunto 3 define o caminho onde o Logstash está instanciado, estabelecendo assim a via de comunicação dos dois produtos por meio da biblioteca *libbeat*, escrita inteiramente em Golang que oferece a API para o envio dos dados.

O arquivo de configuração do Logstash também é definido em 3 partes conforme apresentado na Figura 5.7. O conjunto de instruções 1 especifica a porta que o Logstash ficará escutando e esperando os arquivos do Beats.

```
1 input {
    beats {
      port => 5044
    }
  }
2 filter {
      grok {
          patterns_dir => ["/etc/logstash/pattern.d"]
      match => ["message", "%{LOGLEVEL:log_level}%{SPACE}%{DATESTAMP:time:date}%{SPACE}%{WORD:method}%
      {SPACE}%{URIPROTO:protocol}://%{URIHOST:service_host}%{URIPATH:service_path}%{URIPARAM:params_send}%
      {SPACE}%{URIPROTO:protocol_send}%{URIPATH:protocol_send_version}%
      {SPACE}{\n\tRID: %{WORD:request_id}%{SPACE}%{OPENP}ReqHash: %{WORD:request_hash}%
      {CLOSEP}\n\tAccept: <<%{GREEDYDATA:accept_type}>>\n\tContent-Type in: <<%{GREEDYDATA:content_type_in}>>
      \n\tContent-Type out: <<%{GREEDYDATA:content_type_out}>>\n\tPeer: <<%{GREEDYDATA:peer}>>
      Referer: <<%{GREEDYDATA:referer}>>\n\tUser-Agent: <<%{GREEDYDATA:user_agent}>>
      \n\tService: <<%{GREEDYDATA:service}>>\n\tParams: \#{%{EVERDATA:params}}\n\tQuery: \#{%{EVERDATA:query}}
      \n\tPayload: \#{%{EVERDATA:payload}}\n\t%{UNTILSTATUS:req_message}: %{INT:status_code:int}
      <<%{WORD:status_msg}>> %{OPENP}%{INT:status_time:int}ms%{CLOSEP}\n}"]
      remove_field => ["message"]
      }
      geoip {
          source => "host"
      target => "geoip"
      }
  }
3 output {
    elasticsearch {
      hosts => "localhost:9200"
      user => elastic
      password => changeme
      manage_template => false
      index => "log-ems-bus-%{+YYYY.MM.dd}"
      document_type => "%{[@metadata][type]}"
    }
  }
```

Figura 5.7: Definição do arquivo Logstash

O conjunto 2 é a parte principal do Logstash, pois é este que define como os dados serão tratados. O conjunto de dados enviado pelo Filebeat é formatado como uma mensagem, tendo de cabeçalho o status do serviço e formatando os outros campos do log de forma que cada informação possa ser indexada futuramente. A Figura 5.8 exemplifica uma linha do *log* capturado pelo Filebeat com os campos que são armazenados pelo barramento.

```
INFO 27/10/2017 08:49:33  POST http://164.41.121.112:2301/authorize?grant_type=password&username=geral&password=123456 HTTP/1.1 {
    RID: 1545323493343232  (ReqHash: 105815305)
    Accept: <<"*/*">>
    Content-Type in: <<>>
    Content-Type out: <<"application/json;charset=UTF-8">>
    Peer: <<"164.41.121.112">>  Referer: <<>>
    User-Agent: <<"insomnia/5.9.6">>
    Service: <<"ems_oauth2_authorize:execute">>
    Params: #{}
    Query: #{<<"client_id">> => <<>>,
             <<"client_secret">> => <<>>,
             <<"code">> => <<>>,
             <<"grant_type">> => <<"password">>,
             <<"password">> => <<"123456">>,
             <<"redirect_uri">> => <<>>,
             <<"refresh_token">> => <<>>,
             <<"scope">> => <<>>,
             <<"secret">> => <<>>,
             <<"state">> => <<>>,
             <<"username">> => <<"geral">>}
    Payload: #{}
    Cache-Control: <<>>  ETag: <<>>
    If-Modified-Since: <<>>  If-None-Match: <<>>
    Authorization mode: public
    Authorization header: <<>>
    OAuth2 grant type: <<"password">>
    OAuth2 access token: <<"4hsetAbRV66g3n0COmCx0xd5ujGyRtf7">>
    OAuth2 scope: <<>>
    Client: <<"public">>
    User: <<"public">>
    Node: <<>>
    FileName: <<>>
    Status: 200 <<ok>> (218ms)
}
```

Figura 5.8: Exemplo de *log* capturado pelo Filebeat

O conjunto 3 apresenta a descrição de como será realizada a comunicação do Logstash com o Elasticsearch e como o arquivo deve ser indexado.

O Elasticsearch é o coração dessa solução, dado que ele permite armazenar, pesquisar e analisar grandes volumes de dados rapidamente e em tempo próximo ao real, para isso foi implementado índices invertidos com transdutores de estados finitos para consultas de texto completo, *BKD-tree* para armazenar dados numéricos e geográficos e um armazenamento de colunas para análise. Os campos formatados no Logstash são indexados o que permite alta performance para a busca dos dados. Portanto as consultas realizadas no Elasticsearch deram-se de acordo com as Figuras 5.9, 5.10 e 5.11.

```
{
    "size": 0,
    "aggs": {
        "status_code": {
            "terms" : { "field" : "service.keyword" },
            "aggs": {
                "servi": {
                    "terms" : { "field" : "status_code" }
                }
            }
        }
    }
}
```

Figura 5.9: Consulta de serviço e seu status_code

Figura 5.10: Consulta de erros de requisição dos serviços

Figura 5.11: Consulta de instâncias de um serviço

As consultas são realizadas utilizando o formato JSON por meio de uma requisição HTTP, na qual deve ser indicado de forma hierárquica os campos a serem resgatados e ou verificados de acordo com alguma condição por meio de agregação ou filtros, seguindo sempre o formato estabelecido no Logstash.

Todo o processamento Elastic é realizado em um servidor diferente do barramento ErlangMS, sendo este afetado somente para atualização do *log* por meio do Filebeat, levando a um menor custo de desempenho.

A métrica M3- Número de instâncias por transação foi atendida por meio da consulta apresentada na Figura 5.11, em que esta verifica o número de vezes que um mesmo serviço foi instanciado via uma agregação que é realizada pela URL do mesmo serviço. O resultado apresentado por essa consulta é apresentado na Figura 5.12.

```
{
    "took": 3,
    "timed_out": false,
    "_shards": {
        "total": 10,
        "successful": 10,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": 15,
        "max_score": 0.0,
        "hits": [
        ]
    },
    "aggregations": {
        "service": {
            "doc_count_error_upper_bound": 0,
            "sum_other_doc_count": 0,
            "buckets": [
                {
                    "key": "\"user_service:loader_sync\"",
                    "doc_count": 11
                },
                {
                    "key": "\"ems_oauth2_authorize:execute\"",
                    "doc_count": 3
                }
            ]
        }
    }
}
```


Figura 5.12: Resposta - instâncias de um serviço

A métrica M2- Frequência de uso do método foi atendida por meio da consulta apresentada na Figura 5.9 que retorna todas as solicitações daquele serviço juntamente com seu status_code o que permite a verificação da frequência de uso.

Já a métrica M5- Quantitativo de erros por serviço foi atendida por meio da consulta apresentada na Figura 5.10 obtendo assim o quantidade de cada tipo de erro que pode ocorrer ao longo da arquitetura do barramento para a execução de um serviço especificado pelo serviço requisitante.

5.2.1 Camada de Visualização

O serviço de monitoramento de sistemas em tempo de execução tem como objetivo externalizar as métricas levantadas utilizando de visualização de software para facilitar a

observação dos dados por parte da equipe que mantém os sistemas que estão sendo monitorados.

A implementação do sistema, retirando a parte de back-end referente ao barramento descrita no Capítulo 5, foi realizada utilizando-se das seguintes tecnologias:

- HTML [7];
- JavaScript [22];
- JSON [8];
- Google Material Design [25];
- *Template* AdminBSB [67].

AdminBSB é um *template* para construção de portais de administração totalmente responsivo e gratuito. Ele foi desenvolvido com o Bootstrap 3.x Framework e o Google Material Design o que possibilita o uso de diversos componentes gráficos, fazendo uso de vários conceitos de usabilidade para melhor disponibilização das informações, como arranjo de cores, agrupamento de componentes, classes pré-definidas para formas e tamanhos do *layout*.

O *template* disponibiliza 5 classes distintas para construção de gráficos o que enriquece a disponibilização dos dados para o *dashboard* que são: Morris, Flot, ChartJS, Sparkline e Jquery Knob.

O sistema de monitoramento disponibiliza uma tela de acesso para autenticação do serviço por meio de perfil de uso. Como esse serviço trabalha com dados restritos, é necessária a disponibilização somente para equipes autorizadas e por esse motivo a autenticação do acesso é realizada via LDAP [68] através de um serviço de autenticação pré-existente no barramento. A Figura 5.13 apresenta a tela de acesso do sistema.

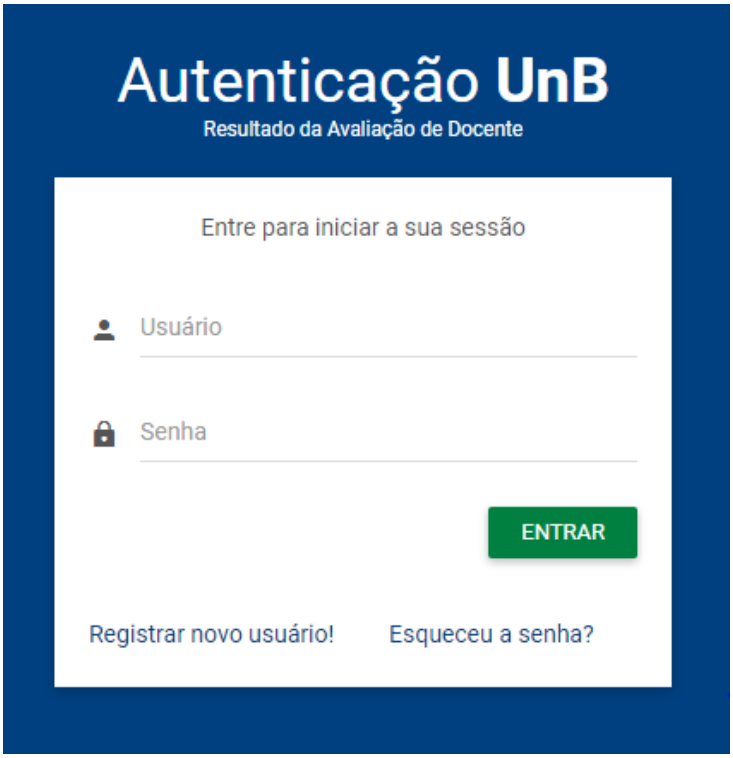

Figura 5.13: Tela de Acesso ao Serviço de Monitoramento

A construção do *front-end* para o serviço de monitoramento seguiu um padrão de cores estabelecidos para o novo *layout* de sistemas do CPD, mantendo assim uma identidade visual entre os sistemas.

O monitoramento de recursos físicos é composto por 3 gráficos e 11 cartões conforme apresentado na Figura 5.14. O primeiro gráfico representa a utilização da CPU de forma percentual, mostrando por meio de um gráfico de linha atualizado em tempo real, possibilitando a visualização de oscilação do uso da CPU.

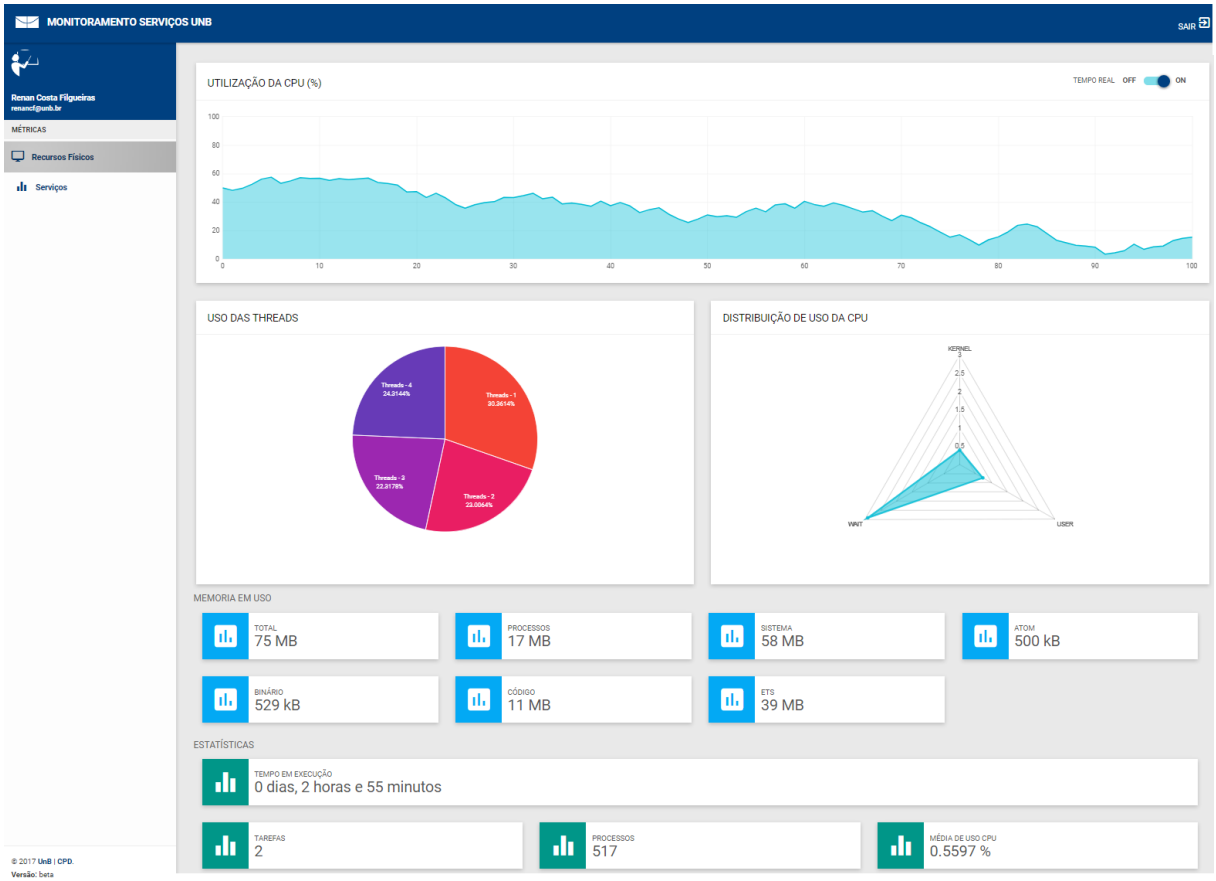

Figura 5.14: Tela do Serviço de Monitoramento - Recursos Físicos

O segundo gráfico apresenta a distribuição de uso em porcentagem de cada *threads* que a máquina possui, totalizando o 100% da CPU da máquina por meio de um gráfico de pizza. Este possibilita observar como está sendo a distribuição dos recursos por *thread*, podendo identificar um possível desbalanceamento ou um algoritmo inadequado para distribuição da execução de cada serviço por consumo de recurso.

Um gráfico de teia (gráfico de radar) foi utilizado para apresentar a distribuição de uso da CPU por consumidor que é um dado composto de três variáveis quantitativas que foram representadas em eixos que partem do mesmo ponto. Essas variáveis são o uso da CPU alocado ao Kernel, Usuário e Em espera. O resultado da subtração de 100 pela soma desses 3 valores representa a quantidade de CPU que não está em uso, esse valor foi retirado como o 4 eixo do gráfico, pois causaria uma disparidade sempre que a máquina estivesse ociosa em uso.

Os cartões de informações foram distribuídos em dois conjunto de dados, Memória em uso e dados estatísticos da máquina. Os 11 cartões estão distribuídos da seguinte forma:

- **Memória em Uso**

  - **Total**: A quantidade total de memória atualmente alocada. Isso é o mesmo que a soma do tamanho da memória para processos e sistema.
    * **Processos**: Total de memória atualmente alocada para os processos Erlang;
    * **Sistema**: Total de memória atualmente alocada para o emulador que não está diretamente relacionada a nenhum processo Erlang.
      · **ATOM**: Total de memória atualmente alocada para átomos;
      · **Binário**: Total de memória atualmente alocada para binários;
      · **Código** : Total de memória atualmente alocada para código Erlang;
      · **ETS** : Total de memória atualmente alocada para tabelas ETS.

- **Estatísticas**
  - **Tempo de execução** : tempo em que a máquina está executando o barramento ininterruptamente;
  - **Tarefas** : Total de tarefas no barramento;
  - **Processos** : Total de processos no barramento;
  - **Média de uso CPU** : Média aritmética do uso de CPU de todas as threads.

Os dados nos cartões fornecem informações que ajudam a nortear o estado da máquina para melhor compreender o que está ocorrendo no exato momento, sendo complementar as outras métricas.

O monitoramento de serviços com abordagem em análise de log é composto por 1 gráfico e 2 cartões conforme apresentado na figura 5.15. Os cartão apresentados desse serviço são compostos de mais informações comparado com o monitoramento de recursos físicos.

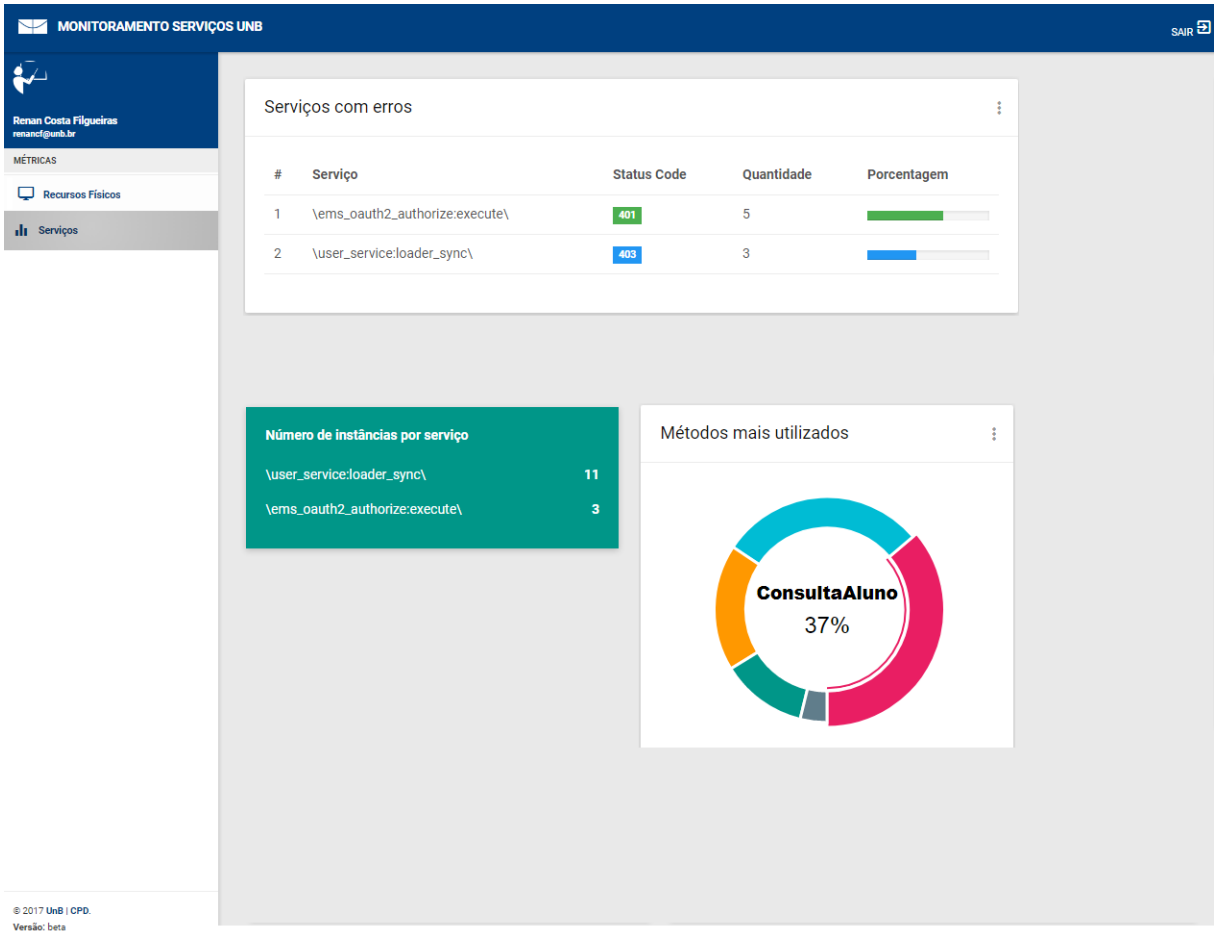

Figura 5.15: Tela do Serviço de Monitoramento - Serviços

O primeiro cartão apresenta os serviços com erros e faz a categorização por cor para facilitar o acompanhamento e é composto pelo nome do serviço, o status code do erro, a quantidade de erros no *log* e a porcentagem referente ao quantitativo geral de erros encontrados. O segundo cartão traz o número de instâncias por serviço, especificando qual é o serviço e a quantidade de instâncias.

Um gráfico de pizza foi usado para demonstrar os métodos mais utilizados e quanto cada um corresponde em porcentagem do total de uso dos métodos.

Essas informações ficam guardadas e indexadas no ElasticSearch, conforme apresentado no capítulo 5, o que permite a recuperação e a análise de qualquer período de tempo desejado. Atualmente o serviço de monitoramento apresenta suas análises somente do estado em tempo real do barramento e faz a verificação nos *logs* do mesmo dia. As futuras evoluções do serviço irá recuperar e demonstrar os dados já contemplados em uma escala de tempo para melhor acompanhamento das equipes do CPD.

5.3 Síntese do Capítulo

Este capítulo apresentou as abordagens e como se realizou a implementação das métricas selecionadas no capítulo 4. As soluções apresentadas podem ser realizadas em outros domínios, possibilitando assim a verificação das métricas em outras plataformas. Também foi apresentado a camada de visualização do serviço de monitoramento de sistemas em tempo de execução, exibindo os componentes e os dados que representam as métricas selecionadas que foram implementadas conforme este capítulo.

Capítulo 6

Avaliação do Serviço de Monitoramento

A fim de avaliar o serviço implementado para o monitoramento de sistemas em tempo de execução, uma entrevista estruturada conforme Seção 4.1 foi feita com a mesma equipe que realizou a entrevista para priorização das métricas de acordo com o Capítulo 4, com um total de 7 participantes. A entrevista se deu após o uso da ferramenta, diante de observação de um caso real utilizando o sistema de Resultado da Avaliação do Docente já em produção, construído seguindo a nova arquitetura de desenvolvimento de sistemas do CPD.

Foram definidas 4 questões fechadas com a finalidade de verificar o quanto a métrica implementada estava correspondendo a expectativa, uma questão para cada métrica levantada no mapeamento sistemático. Duas questões abertas, uma a fim de levantar a opinião se o uso da ferramenta foi importante durante o acompanhamento de uso do sistema e outra questão para sugestões de possíveis melhorias na ferramenta.

6.1 Resultados

Todas as métricas atingiram um grau de satisfação da expectativa de 3 a 5, o que demonstra que atenderam as expectativas geradas pelo uso das métricas de acordo com o Capítulo 4.

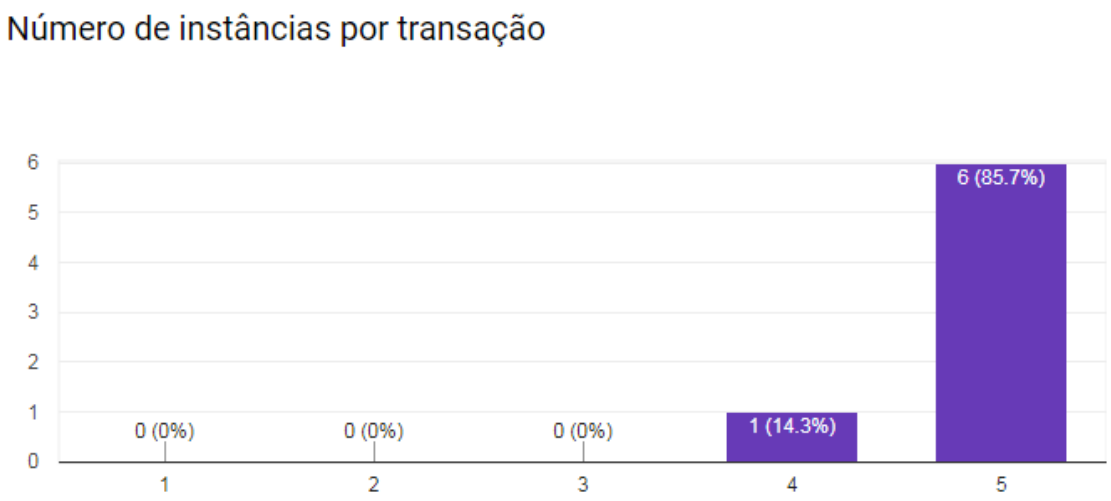

Figura 6.1: Respostas Métrica 3

A métrica M3 obteve 6 votos de grau 5 e 1 voto de grau 4, sendo assim a métrica que mais atendeu as expectativas da equipe conforme Figura 6.1. O número de instâncias por transação foi, portanto, um dado muito interessante para observação em tempo de execução.

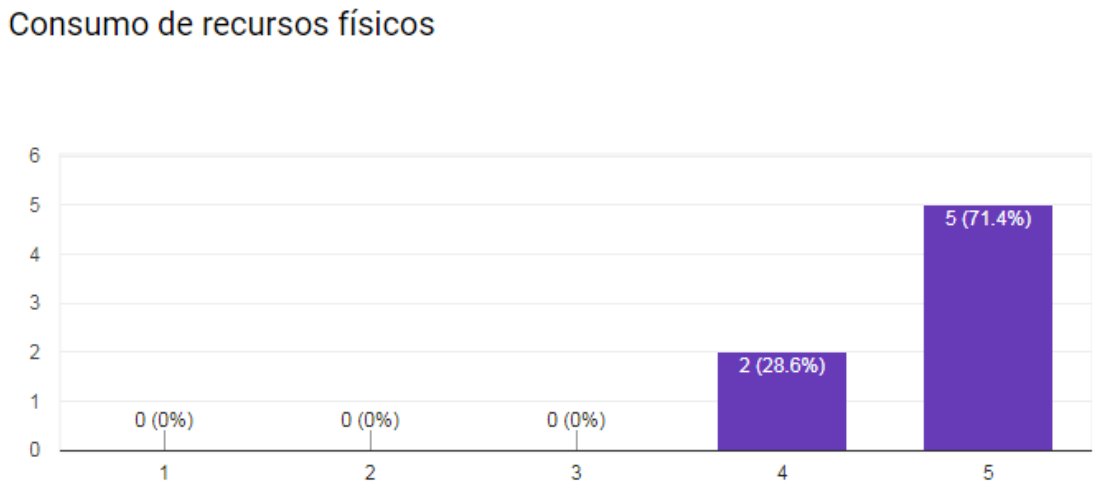

Figura 6.2: Respostas Métrica 1

As métricas M1 e M2 apresentaram 5 votos grau 5 e 2 votos grau 4, conforme figuras 6.3 e 6.2, representando também uma grande expressividade no acompanhamento do sistema. Essas métricas possuem a maior quantidade de elementos em tela para representá-las, que

são dados de apoio como é o caso de consumo de recursos físicos, que traz informações de memória e da CPU.

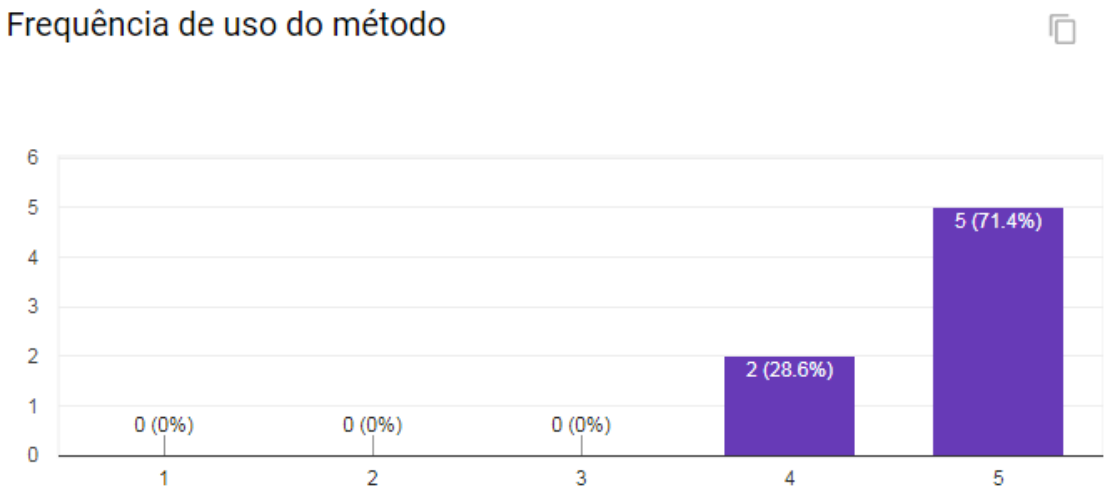

Figura 6.3: Respostas Métrica 2

A métrica M4 - Porcentagem de utilização da CPU foi a que apresentou o menor resultado entre as métricas conforme Figura 6.4, porém isso se deve ao fato dos desenvolvedores terem aferido a importância da métrica M1 - Consumo de recursos físicos e concluíram que a M4 é uma métrica apoio para esta, que ,assim sendo, ela sozinha não apresentaria um dado substancial.

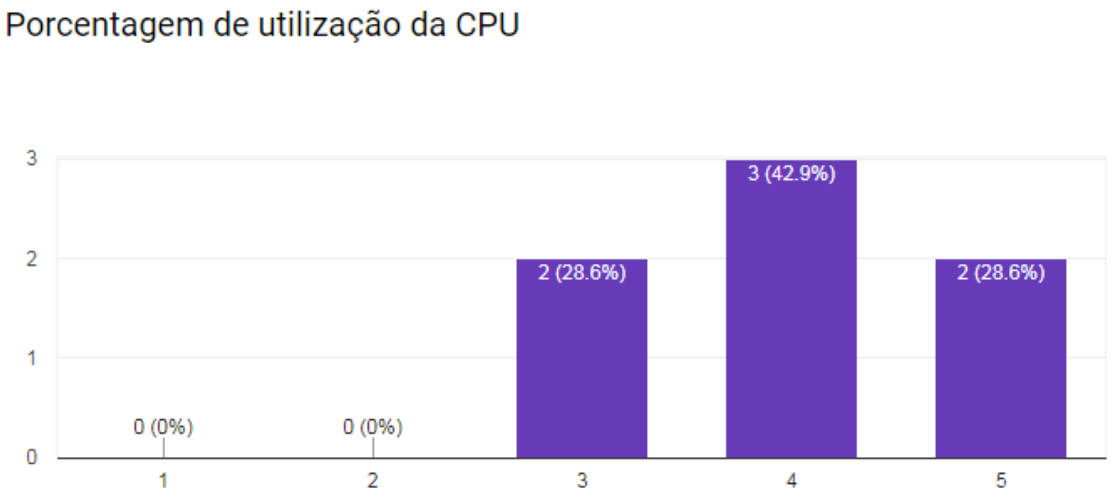

Figura 6.4: Respostas Métrica 4

A métrica M5 - Quantitativo de erros por serviço, obteve 2 votos de grau 5 e 5 votos de grau 4, conforme figura 6.5, o que consolidou a importância dessa métrica que foi proposta pela própria equipe do CPD. Foi por meio desta métrica que foi detectada uma brecha de segurança por SQLInjection na requisição do serviço o que foi fundamental para a equipe compreender melhor a importância de um serviço de monitoramento em tempo real.

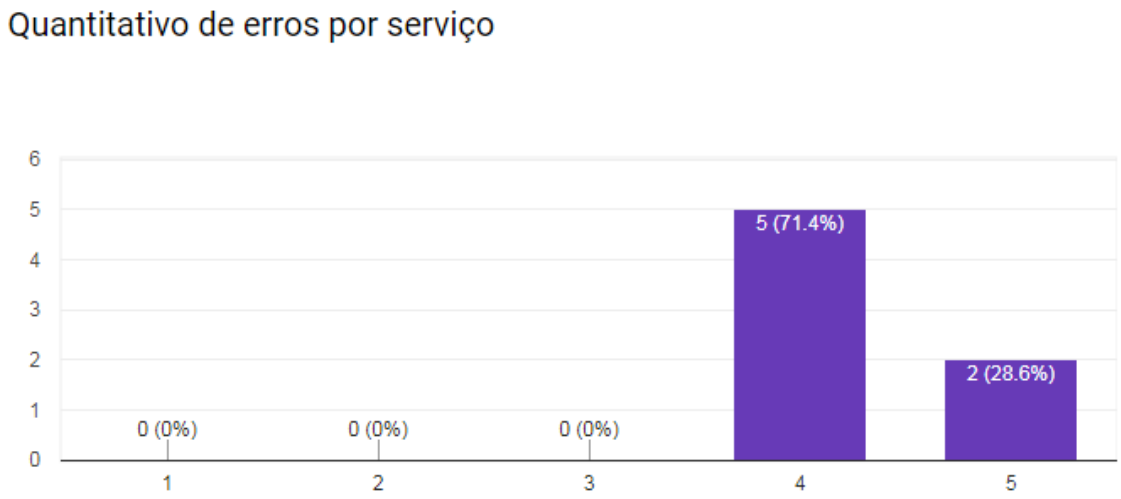

Figura 6.5: Respostas Métrica 5

A questão aberta "O uso da ferramenta foi importante durante o acompanhamento de uso do sistema?"apresentou a resposta "Sim"de todos os entrevistados variando apenas algumas frases complementares conforme apresentado na Tabela 6.1.

Tabela 6.1: Resposta Questão Aberta 1

Respostas QA 1
Sim (4)
Sim, auxiliou no acompanhamento do que estava se passando no sistema
Sim, pois não existia nenhuma ferramenta similar no Centro
Sim, pude acompanhar melhor o que estava ocorrendo

A questão aberta "O que poderia melhorar na ferramenta?", gerou respostas distintas, contudo com uma convergência para a implementação de novas métricas. Com a demonstração e o uso de serviço de monitoramento de sistemas em tempo real, a equipe ficou motivada com o potencial da ferramenta, o que levou a uma necessidade de novas métricas com o objetivo de se obter um ambiente mais rico. As novas métricas que podem vir

a ser implementadas são contempladas normalmente por instrumentação de código que possivelmente será a nova frente da continuação desse trabalho.

Tabela 6.2: Resposta Questão Aberta 2

Respostas QA 2
Alguns dados poderiam ser apresentados como gráficos
Para mim está ok
O uso de instrumentação de código para mais métricas
Mais gráficos
Informações das métricas
Nada
Uso de métricas por instrumentação

6.2 Síntese do Capítulo

Este capítulo apresentou uma avaliação por parte da equipe do CPD das métricas selecionadas e do serviço de monitoramento implementado. As respostas demonstraram uma boa aceitação da equipe quanto à nova tecnologia e exaltou a importância de tal ferramenta no contexto da instituição.

Capítulo 7

Conclusão

Permitir que as organizações acompanhem um planejamento adequado da fase de manutenção de seus sistemas, já que esta corresponde a cerca de 75% a 80% dos recursos de programação consumidos, é fundamental para tomadas de decisões futuras e planejamento do seguimento desses sistemas como legado.

Verificar outras abordagens para monitoramento de sistemas demonstrou-se interessante ao que se refere aos complementos de informações técnicas e de negócio a instituição. O conhecimento das possibilidades de avaliação do sistema durante sua execução permite tomadas de decisões mais imediatas tanto da equipe de acompanhamento quanto da organização.

Este trabalho possibilitou o conhecimento de métricas e abordagens que dão suporte ao monitoramento de sistemas e sistemas legados. A listagem de resultados auxiliou nas decisões de uma camada de acompanhamento dos sistemas, permitindo ajustes que viabilizam maior agilidade nas correções de possíveis anomalias, levando a uma redução de custos e tempo na manutenção dos novos sistemas e sistemas legados utilizados pelas organizações e que não podem ter seus serviços interrompidos, acarretando em enormes prejuízos financeiros, caso um de seus serviços fiquem fora do ar.

O serviço de monitoramento de sistemas em tempo de execução é uma grande oportunidade do CPD passar a ter no seu dia a dia essa solução pontual para seus períodos críticos sazonais e a ferramenta demonstrou estar atendendo as expectativas da equipe de desenvolvimento que fará uso dela. Durante o período teste da ferramenta no sistema Resultado da Avaliação Docente, foi possível detectar uma brecha de segurança de SQLInjection no momento da requisição por meio da análise de *logs*. Esse tipo de falha não corresponde diretamente ao que as métricas implantadas buscam, mas demonstra que uma análise em tempo de execução irá permitir um novo olhar da equipe do Centro e possibilitará a realização de manutenção mais adequada a necessidade do sistema de acordo com os dados fornecidos pela ferramenta.

7.1 Contribuições

A contribuição fundamental deste trabalho foi a implementação de um sistema de monitoramento de sistemas em tempo de execução para o CPD, favorecendo a necessidade que existia no Centro. Como a ferramenta teve seu alicerce no barramento ErlangMS, ela foi desenvolvida a se tornar um módulo de monitoramento para Erlang, isso possibilita que qualquer organização utilizando Erlang, poderá utilizar desta solução para obtenção das métricas levantadas no mapeamento sistemático. As soluções arquiteturas utilizando SOA, REST e Erlang também poderão fazer uso da ferramenta criada para o monitoramento de sistemas em tempo real.

Outra contribuição foi o resultado do mapeamento sistemático que gerou uma lista de métricas que foram validadas para a obtenção de dados em tempo de execução de um sistema. Essa lista possibilita que outros estudos possam utilizar esse levantamento e que outras ferramentas venham a implementar essas métricas.

Por fim, a entrevista realizada para priorização das métricas, além do uso da ferramenta desenvolvida, também levantou uma nova frente no CPD, pois despertou a relevância do monitoramento de sistemas, o que gerou *features* no *backlog* para implementação de novas funcionalidades e métricas para a ferramenta.

7.2 Trabalhos Futuros

Foi apresentada uma abordagem para as métricas encontradas na literatura que pudesse resultar em benefícios de forma mais rápida e menos onerosa ao serviço. Uma abordagem que não foi realizada nesse trabalho é a de instrumentação de código.

Visando expandir o potencial do módulo de monitoramento de métricas Erlang, propõe-se o uso de instrumentação de código para gerar um ambiente mais completo. A exemplo o pacote Exometer, que é um pacote de instrumentação de código para Erlang que traz diversas formas de instrumentação como contadores, histogramas e funções.

Evoluir o módulo de monitoramento criado nesse trabalho com instrumentação de código é uma das formas de enriquecê-lo e dar continuidade ao monitoramento dos serviços do CPD.

Referências

[1] Giuseppe Aceto, Alessio Botta, Walter De Donato, e Antonio Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, 2013. 17

[2] Otto Julio Ahlert Pinno, Sand Luz Correa, Aldri Luiz dos Santos, e Kleber Vieira Cardoso. Using partial correlation for effective metric selection and anomaly detection in multi-tier system. In *Computer Networks and Distributed Systems (SBRC), 2015 XXXIII Brazilian Symposium on*, pages 269–277. IEEE, 2015. 21, 23

[3] David J Angel, James R Kumorek, Farokh Morshed, e David A Seidel. Byte code instrumentation, November 6 2001. US Patent 6,314,558. 38

[4] Paulo Roberto Azevedo. Arquitetura de convivência: escalabilidade e evolução planejada de sistemas legados, jun 2015. 1, 2

[5] Len Bass. *Software architecture in practice*. Pearson Education India, 2007. 6, 7

[6] Keith H Bennett e Václav T Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 73–87. ACM, 2000. 1

[7] Tim Berners-Lee e Dan Connolly. Hypertext markup language-2.0. Technical report, 1995. 45

[8] Tim Bray. The javascript object notation (json) data interchange format. 2014. 45

[9] David Budgen, Mark Turner, Pearl Brereton, e Barbara Kitchenham. Using mapping studies in software engineering. In *Proceedings of PPIG*, volume 8, pages 195–204. Lancaster University, 2008. 14

[10] D Cabrero, J Garzas, e M Piattini. Software artifact prioritization based on the frequency of use. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 7(3):369–376, 2009. 2, 12

[11] Maria Istela Cagnin. *PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks*. Tese (Doutorado), Universidade de São Paulo (USP). Instituto de Ciências Matemáticas e de Computação de São Carlos, 2005. 1

[12] Sarah E Chodrow, Farnam Jahanian, e Marc Donner. Run-time monitoring of real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 74–83. IEEE, 1991. 13

[13] Mike Clarke e AD Oxman. Cochrane reviewers handbook 4.1. *Review Manager*, 4(1):208–215, 2000. 14

[14] Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, e Reed Little. *Documenting software architectures: views and beyond*. Pearson Education, 2002. 5

[15] Don Coleman, Dan Ash, Bruce Lowther, e Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994. 11

[16] Ministério da Ciência e Tecnologia. Estrategia nacional de ciência, tecnologia e inovação, 2016. Secretaria de Politica de Informatica. 2

[17] Everton de Vargas Agilar e Rodrigo Bonifacio de Almeida. Uma abordagem orientada a serviços para a modernização dos sistemas legados da unb. 2016. x, 3, 6, 7, 8

[18] Nelly Delgado, Ann Q Gates, e Steve Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on software Engineering*, 30(12):859–872, 2004. 12

[19] Wayne W Eckerson. *Performance dashboards: measuring, monitoring, and managing your business*. John Wiley & Sons, 2010. 11

[20] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005. 5, 7

[21] Stephen Few. Information dashboard design. 2006. 11

[22] David Flanagan. *JavaScript: the definitive guide*. "O'Reilly Media, Inc.", 2006. 45

[23] Laure France, Jean-Mathias Heraud, Jean-Charles Marty, Thibault Carron, e Joseph Heili. Monitoring virtual classroom: Visualization techniques to observe student activities in an e-learning system. In *ICALT*, pages 716–720, 2006. 11

[24] Antonio Carlos Gil. *Métodos e técnicas de pesquisa social*. 6. ed. Ediitora Atlas SA, 2008. 24, 25

[25] Google. Material design, oct 2017. 45

[26] Robert B Grady e Deborah L Caswell. Software metrics: establishing a company-wide program. 1987. 10

[27] Tibor Gyimothy. To use or not to use? the metrics to measure software quality (developers' view). In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pages 3–4. IEEE, 2009. 17, 21, 22

[28] John P. Hudepohl, Stephen J Aud, Taghi M. Khoshgoftaar, Edward B Allen, e Jean Mayrand. Emerald: Software metrics and models on the desktop. *IEEE Software*, 13(5):56–60, 1996. 10

[29] ISO Iso. Iec 9126-1: Software engineering-product quality-part 1: Quality model. *Geneva, Switzerland: International Organization for Standardization*, page 27, 2001. 5, 10

[30] NBR ISO. Iec 12207–tecnologia de informação-processos de ciclo de vida de software. *Rio de Janeiro: ABNT*, 1998. 5

[31] Farnam Jahanian. Run-time monitoring of real-time systems. *Advances in real-time systems*, pages 435–460, 1995. 13

[32] Michael Jakl. Rest representational state transfer. 2008. 7

[33] Miao Jiang, Mohammad A Munawar, Thomas Reidemeister, e Paul AS Ward. System monitoring with metric-correlation models: problems and solutions. In *Proceedings of the 6th international conference on Autonomic computing*, pages 13–22. ACM, 2009. 21

[34] Staffs Keele. Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. 2007. 15

[35] Thomas Koshy. *Fibonacci and Lucas numbers with applications*, volume 51. John Wiley & Sons, 2011. 30

[36] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010. 18

[37] Eric Larson. Suds: an infrastructure for creating dynamic software defect detection tools. *Automated Software Engineering*, 17(3):301–346, 2010. 20, 22

[38] Meir M Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980. 1

[39] Rick Kazman Len Bass, Paul C. Clements. *Software Architecture in Practice*, volume 2 of *0321154959*. Addison-Wesley Professional, 2003. 5

[40] Bennet P Lientz, E. Burton Swanson, e Gail E Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, 1978. 1

[41] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932. 25

[42] Bev Littlewood. How to measure software reliability, and how not to. In *Proceedings of the 3rd international conference on Software engineering*, pages 37–45. IEEE Press, 1978. 11

[43] Shadan Malik. *Enterprise dashboards: design and best practices for IT*. John Wiley & Sons, 2005. 11

[44] Rafael Messias Martins. *Técnicas de Visualização para Avaliação e Melhoria de Qualidade de Software Livre e Aberto*. Tese (Doutorado), Instituto de Ciências Matemáticas e de Computação - ICMC-USP, 2012. 11

[45] Paulo Roberto Miranda Meirelles. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado), Universidade de São Paulo, 2013. 10

[46] Dicionário Michaelis. Disponível em:<http://michaelis.uol.com.br>. *Acesso em*, 28, 2016. 9

[47] Rafael Pereira, Stephano Gonçalves, Lisane Brisolara, Julio CB Mattos, e Ulisses B Correa. Java code analyser for estimating embedded software efficiency. In *Computing System Engineering (SBESC), 2011 Brazilian Symposium on*, pages 8–14. IEEE, 2011. 22, 23

[48] Kai Petersen, Robert Feldt, Shahid Mujtaba, e Michael Mattsson. Systematic mapping studies in software engineering. In *12th international conference on evaluation and assessment in software engineering*, volume 17. sn, 2008. 14

[49] Kai Petersen, Sairam Vakkalanka, e Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015. 14

[50] Bernhard Plattner. Real-time execution monitoring. *IEEE transactions on Software Engineering*, (6):756–764, 1984. 13

[51] V Podgorelec e S Kuhar. Taking advantage of education data: Advanced data analysis and reporting in virtual learning environments. *Elektronika ir Elektrotechnika*, 114(8):111–116, 2011. 11

[52] Roger S Pressman. *Engenharia de software*. AMGH Editora, 2009. 10

[53] Alan Ramirez-Noriega, Reyes Juarez-Ramirez, Raul Navarro, e Janeth Lopez-Martinez. Using bayesian networks to obtain the task's parameters for schedule planning in scrum. In *Software Engineering Research and Innovation (CONISOFT), 2016 4th International Conference in*, pages 167–174. IEEE, 2016. 30

[54] William N Robinson. Monitoring software requirements using instrumented code. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3967–3976. IEEE, 2002. 12

[55] Ulrich Schmid. Monitoring distributed real-time systems. *Real-Time Systems*, 7(1):33–56, 1994. 9, 12

[56] Claire Selltiz. *Métodos de pesquisa nas relações sociais*. EPU, 1974. 24

[57] Bikram Sengupta, Nilanjan Banerjee, Chatschik Bisdikian, e Paul Hurley. Tracking transaction footprints for non-intrusive end-to-end monitoring. *Cluster Computing*, 12(1):59–72, 2009. 18, 22

[58] Cassia Baldini Soares e Tatiana Yonekura. Revisão sistemática de teorias: uma ferramenta para avaliação e análise de trabalhos selecionados. *Revista da Escola de Enfermagem da USP*, 45(6):1507–1514, 2011. 17

[59] Ian Sommerville, Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, e Edilson de Andrade Barbosa. *Engenharia de software*, volume 6. Addison Wesley São Paulo, 2003. 10

[60] Tim Souder, Spiros Mancoridis, e Maher Salah. Form: A framework for creating views of program executions. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, page 612. IEEE Computer Society, 2001. 2, 12

[61] Elastic Stack. Elastic architecture, oct 2017. x, 39

[62] Henrik Thane. *Monitoring, testing and debugging of distributed real-time systems*. Tese (Doutorado), Ph. D. Thesis, MRTC Report 00/15, 2000. 13

[63] Katrien Verbert, Erik Duval, Joris Klerkx, Sten Govaerts, e José Luis Santos. Learning analytics dashboard applications. *American Behavioral Scientist*, page 0002764213479363, 2013. 11

[64] Jonathan Stuart Ward e Adam Barker. Cloud cover: monitoring large-scale clouds with varanus. *Journal of Cloud Computing*, 4(1):1–28, 2015. 17

[65] Michael W Whalen, Suzette Person, Neha Rungta, Matt Staats, e Daniela Grijincu. A flexible and non-intrusive approach for computing complex structural coverage metrics. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 506–516. IEEE Press, 2015. 20, 22

[66] Lingyun Yang, Jennifer M Schopf, Catalin L Dumitrescu, e Ian Foster. Statistical data reduction for efficient application performance monitoring. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 8–pp. IEEE, 2006. 22

[67] Guray Yarar. Adminbsb, oct 2017. 45

[68] Wengyik Yeong, Tim Howes, e Steve Kille. Lightweight directory access protocol. Technical report, 1995. 45

Apêndice A

Catálogo de Serviço do Monitoramento

Código A.1: Catálogo de Serviço para o monitoramento.

```
[
        {
                "name": "/netadm/restart",
                "comment": "Restart ems-bus.",
                "owner": "netadm",
                "version": "1.0.0",
                "service" : "ems_netadm_service:restart",
                "url": "/netadm/restart",
                "async": "false",
                "result_cache" : 0,
                "type": "GET",
                "authorization": "basic",
                "lang" : "erlang"
        },

        {
                "name": "/netadm/pid",
                "comment": "Get main pid of ems-bus.",
                "owner": "netadm",
                "version": "1.0.0",
                "service" : "ems_netadm_service:pid",
                "url": "/netadm/pid",
```

```
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/avgcpu",
        "comment": "Get average cpu usage.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:avgcpu",
        "url": "/netadm/avgcpu",
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/cpus",
        "comment": "Get cpus usage.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:cpus",
        "url": "/netadm/cpus",
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/cpudetailed",
```

```
        "comment": "Get cpus usage.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:cpudetailed",
        "url": "/netadm/cpudetailed",
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/tasks",
        "comment": "Get total tasks running.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:tasks",
        "url": "/netadm/tasks",
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/uptime",
        "comment": "Get ems-bus uptime.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:uptime",
        "url": "/netadm/uptime",
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
```

```
        "lang" : "erlang"
},

{
        "name": "/netadm/names",
        "comment": "List cluster services and port.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:names",
        "url": "/netadm/names",
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/world",
        "comment": "List cluster services.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:world",
        "url": "/netadm/world",
        "async": "false",
        "result_cache" : 0,
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/hostfile",
        "comment": "List content of Erlang hostfile.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:hostfile",
```

```
        "url": "/netadm/hostfile",
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/hostname",
        "comment": "Get ems-bus hostname.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:hostname",
        "url": "/netadm/hostname",
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/localhost",
        "comment": "Get ems-bus localhost.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:localhost",
        "url": "/netadm/localhost",
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/memory",
        "comment": "Get ems-bus memory.",
        "owner": "netadm",
```

```
        "version": "1.0.0",
        "service" : "ems_netadm_service:memory",
        "url": "/netadm/memory",
        "result_cache" : 0,
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/timestamp",
        "comment": "Get ems-bus timestamp.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:timestamp",
        "url": "/netadm/timestamp",
        "result_cache" : 0,
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/threads",
        "comment": "Get ems-bus threads count.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:threads",
        "url": "/netadm/threads",
        "result_cache" : 0,
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},
```

```
{
        "name": "/netadm/info",
        "comment": "Get ems-bus info.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:info",
        "url": "/netadm/info",
        "result_cache" : 0,
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/config",
        "comment": "Get config.",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_netadm_service:config",
        "url": "/netadm/config",
        "result_cache" : 0,
        "async": "false",
        "type": "GET",
        "authorization": "basic",
        "lang" : "erlang"
},

{
        "name": "/netadm/datasource/:id",
        "comment": "Get datasource by id",
        "owner": "netadm",
        "version": "1.0.0",
        "service" : "ems_api_query_service:find_by_id",
        "url": "/netadm/datasource/:id",
        "type": "GET",
```

```
                "authorization": "basic",
                "datasource" : {
                        "type" : "mnesia",
                        "table_name" : "service_datasource"
                },
                "lang" : "erlang"
        },

]
```

Apêndice B

Código do Serviço do Monitoramento

```
-module(ems_netadm_service).

-include("../include/ems_config.hrl").
-include("../include/ems_schema.hrl").

-export([names/1, world/1, hostfile/1, hostname/1, localhost/1, memory/1, timestamp/1,
     threads/1, info/1, config/1, restart/1, pid/1, uptime/1, tasks/1, avgcpu/1, cpus↩
         /1, cpudetailed/1]).

names(Request) ->
  ContentData = case net_adm:names() of
    {ok, Names} -> ems_schema:to_json(Names);
    Error -> ems_schema:to_json(Error)
  end,
  {ok, Request#request{code = 200,
              response_data = ContentData}
  }.

world(Request) ->
  try
    ContentData = [ atom_to_list(R) || R <- net_adm:world() ],
    {ok, Request#request{code = 200,
                response_data = ems_schema:to_json(ContentData)}
    }
  catch
    _Exception:_Reason ->
      {ok, Request#request{code = 200,
                  response_data = <<"[]">>}
      }
  end.

hostfile(Request) ->
  ContentData = net_adm:host_file(),
  {ok, Request#request{code = 200,
```

```
                response_data = ems_schema:to_json(ContentData)}
  }.

localhost(Request) ->
  ContentData = {ok, net_adm:localhost()},
  {ok, Request#request{code = 200,
                response_data = ems_schema:to_json(ContentData)}
  }.

hostname(Request) ->
  Conf = ems_config:getConfig(),
  ContentData = {ok, Conf#config.ems_hostname},
  {ok, Request#request{code = 200,
                response_data = ems_schema:to_json(ContentData)}
  }.

memory(Request) ->
  ContentData = [{erlang:atom_to_binary(K, utf8), case V=<1048576 of true -> ↩
      integer_to_list(erlang:trunc(V / 1024)) ++ " kB"; false -> integer_to_list(erlang↩
      :trunc(V / 1024 / 1024)) ++ " MB" end} || {K,V} <- erlang:memory()],
  {ok, Request#request{code = 200,
                response_data = ems_schema:to_json(ContentData)}
  }.

timestamp(Request) ->
  ContentData = {ok, ems_util:timestamp_str()},
  {ok, Request#request{code = 200,
                response_data = ems_schema:to_json(ContentData)}
  }.

threads(Request) ->
  ContentData = {ok, length(erlang:processes())},
  {ok, Request#request{code = 200,
                response_data = ems_schema:to_json(ContentData)}
  }.

info(Request) ->
  ContentData = ranch_info(),
  {ok, Request#request{code = 200,
                response_data = ems_schema:to_json(ContentData)}
  }.

config(Request) ->
  ContentData = lists:flatten(io_lib:format("~p", [ems_config:getConfig()])),
  {ok, Request#request{code = 200,
                content_type = <<"text/plain">>,
                response_data = ContentData}
  }.

restart(Request) ->
  init:restart(),
  {ok, Request#request{code = 200,
                response_data = ?OK_JSON}
  }.

pid(Request) ->
  ContentData = {ok, list_to_integer(os:getpid())},
```

```
  {ok, Request#request{code = 200,
              response_data = ems_schema:to_json(ContentData)}
  }.

uptime(Request) ->
  ContentData = {ok, ems_util:uptime_str()},
  {ok, Request#request{code = 200,
              response_data = ems_schema:to_json(ContentData)}
  }.

tasks(Request) ->
  ContentData = {ok, statistics(total_active_tasks)},
  {ok, Request#request{code = 200,
              response_data = ems_schema:to_json(ContentData)}
  }.

avgcpu(Request) ->
  ContentData = {ok,  average([U || {_, U, _, _} <- cpu_sup:util([per_cpu])])},
  {ok, Request#request{code = 200,
              response_data = ems_schema:to_json(ContentData)}
  }.

cpus(Request) ->
  ContentData = [{lists:concat(["N",K]),U} || { K, U, _, _} <- cpu_sup:util([per_cpu])]↩
      ,
  io:format("is ~p\n", [ContentData]),
  {ok, Request#request{code = 200,
              response_data = ems_schema:to_json(ContentData)}
  }.

cpudetailed(Request) ->
  ContentData = lists:nth(1,[[X,Y,Z,W] || {_,[_,_,X,_,Y],[_,Z,W],_} <- [cpu_sup:util([↩
      detailed])]]),
  io:format("is ~p\n", [ContentData]),
  {ok, Request#request{code = 200,
              response_data = ems_schema:to_json(ContentData)}
  }.

average(ListValue) ->
  lists:sum(ListValue)/length(ListValue).

ranch_info() -> ranch_info(ranch:info(), []).

ranch_info([], R) -> R;
ranch_info([{Listener, Info}|T], R) ->
  Info2 = [{erlang:atom_to_binary(K, utf8), ranch_value_to_binary(V)} || {K,V} <- Info,↩
       K =/= pid andalso
                                              K =/= protocol_options andalso
                                              K =/= pid andalso
                                              K =/= transport_options],
  ListenerBin = erlang:atom_to_binary(Listener, utf8),
  ranch_info(T, [{ListenerBin, maps:from_list(Info2)} | R]).

ranch_value_to_binary(V) when is_integer(V) -> integer_to_binary(V);
ranch_value_to_binary(V) when is_atom(V) -> erlang:atom_to_binary(V, utf8);
ranch_value_to_binary(V) when is_list(V) -> list_to_binary(V);
```

```
ranch_value_to_binary({A,B,C,D}) -> list_to_binary(io_lib:format("{~p,~p,~p,~p}", [A,B,↩
    C,D]));
ranch_value_to_binary(V) -> V.
```

Código B.1: Arquivo *ems_netadm_service.erl*.