UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS GRADUAÇÃO EM INFORMÁTICA

Marcos José Ferreira Neto

VISUALIZAÇÃO DE DADOS DE MONITORAMENTO DE REQUISITOS DE QUALIDADE CONTEXTUALIZADOS COM A ARQUITETURA DE SOFTWARE

Maceió - AL
2017

MARCOS JOSÉ FERREIRA NETO

VISUALIZAÇÃO DE DADOS DE MONITORAMENTO DE REQUISITOS DE QUALIDADE CONTEXTUALIZADOS COM A ARQUITETURA DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas, como requisito para obtenção do grau de Mestre em Informática.

Orientador: Prof. Dr. Patrick H. S. Brito

Maceió - AL
2017

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Bibliotecária Responsável: Helena Cristina Pimentel do Vale

F383v Ferreira Neto, Marcos José.
Visualização de dados de monitoramento de requisitos de qualidade contextualizados provenientes da arquitetura de software / Marcos José Ferreira Neto. – 2017.
121 f. : il.

Orientador: Patrick Henrique da Silva Brito.
Dissertação (mestrado em Informática) – Universidade Federal de Alagoas. Instituto de Computação. Programa de Pós-Graduação em Informática. Maceió, 2017.

Bibliografia: f. 108-114.
Apêndice: f. 115-121.

1. Arquitetura de software. 2. Monitoração de requisitos. 3. Requisitos de qualidade. 4. Requisitos não-funcionais. 5. Software – Visualização.
I. Título.

CDU: 004.41

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Informática – PpgI
Instituto de Computação
Campus A. C. Simões BR 104-Norte Km 14 BL 12 Tabuleiro do Martins
Maceió/AL - Brasil CEP: 57.072-970 | Telefone: (082) 3214-1401

Membros da Comissão Julgadora da Dissertação de Marcos José Ferreira Neto, intitulada: "Visualização de Dados de Monitoramento de Requisitos de Qualidade Contextualizados com a Arquitetura de Software", apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas em 05 de outubro de 2017, às 9h, na Sala "Alan Turing", do Bloco 12 da UFAL.

COMISSÃO JULGADORA

Prof. Dr. Patrick Henrique da Silva Brito
UFAL – Instituto de Computação
Orientador

Prof. Dr. Baldoino Fonseca dos Santos Neto
UFAL – Instituto de Computação
Examinador

Prof. Dra. Regina Lúcia de Oliveira Moraes
Universidade Estadual de Campinas - UNICAMP
Examinadora

Aos meus familiares e amigos pelo apoio e compreensão nesse período. E principalmente a Deus por me dar força, saúde e paciência para conseguir.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus por tudo.

Aos meus pais José Severino Neto e Arlete Ferreira Neto e à minha irmã Aline Priscila de O. Neto, por sempre acreditar, incentivar e apoiar.

À minha querida e amada esposa Mariany Alves do Nascimento pela compreensão, incentivo e apoio durante todo esse período.

Aos meus amigos Carlos Farias, Dyego Ítalo, Henrique Ferreira, Ítalo Carlos e Marcus Piancó pelo apoio e motivação.

Aos participantes do experimento avaliativo desenvolvido nesta dissertação.

A todos os meus professores e ex-professores, que contribuíram direta ou indiretamente para a realização desse sonho.

Ao meu Orientador Prof. Dr. Patrick Henrique, pela paciência e dedicação, sempre indicando o melhor caminho para resolver os problemas.

Ao Prof. Dr. Baldoino Fonseca dos Santos Neto e à Profa. Dra. Regina Lúcia de Oliveira Moraes por aceitarem o convite de fazer parte da banca examinadora.

Por fim, um agradecimento aos meus amigos do Grupo de Tecnologia da Informação (GTINFO) da Prefeitura de Arapiraca e do Núcleo de Tecnologia da Informação (NTI) da Universidade Federal de Alagoas - Campus Arapiraca.

"Quanto mais aumenta nosso conhecimento, mais evidente fica nossa ignorância."
(John F. Kennedy)

"Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes."
(Marthin Luther King)

"Ter acesso ao conhecimento é um presente de Deus, e saber utilizá-lo, é um privilégio divino."
(Carmem Garuzzi)

"Todas as vitórias ocultam uma abdicação."
(Simone de Beauvoir)

RESUMO

O problema do grande volume de dados está presente na indústria de software, em especial nos dados de *log* de monitoramento de software, que são gerados em tempo real, à medida que o software é utilizado. Alguns fatores dificultam consideravelmente a análise manual dos dados de *log*. Um deles é o grande volume de dados gerado; outro fator que dificulta a análise manual dos *logs* é o fato de estarem, normalmente, espalhados em um ou vários arquivos de registro, apresentando-se de forma desordenada. Por essa razão, a análise manual dos *logs* tem se mostrado uma atividade propensa a erros. Sendo assim, se faz necessário utilizar alguma ferramenta computacional para apoiar a análise dos *logs*. Além disso, para se ter uma boa interpretação dos dados analisados é necessário contextualizá-los com modelos abstratos do software, tais como por exemplo a arquitetura de software. Esse trabalho tem como objetivo desenvolver uma abordagem de visualização para facilitar o processo de análise de dados de monitoramento dos requisitos de qualidade (eficiência e confiabilidade), proporcionando uma maior contextualização com abstrações da arquitetura do software e uma ferramenta, chamada VisRQ, com suporte a essa abordagem. Tal ferramenta foi avaliada por um experimento semi-estruturado, em que os usuários analisaram os *logs* de monitoramento e responderam questionários visando aferir o atendimento dos requisitos desejados. Também foram avaliados o tempo de aprendizado da ferramenta e se ela torna o processo de análise dos *logs* mais preciso e rápido. Os participantes do experimento foram discentes do curso técnico de Informática do Instituto Federal de Alagoas - Campus Arapiraca e do curso de Ciência da Computação da Universidade Federal de Alagoas - Campus Arapiraca, além de alguns profissionais formados que trabalham com Tecnologia da Informação. A aceitação da ferramenta foi positiva, conforme as respostas do experimento avaliativo, 100% dos participantes afirmaram que preferem utilizar a ferramenta VisRQ em vez do processo anterior que é menos automatizado e 97,6% consideraram que ela facilita o processo de análise. O experimento mostra que o processo de análise dos *logs* envolvendo a ferramenta VisRQ é mais eficiente, além de permitir a visualização das informações sobre os requisitos não-funcionais de forma contextualizada com diagramas da arquitetura de software.

Palavras-chaves: Arquitetura de Software. Visualização de Dados. Requisitos de Qualidade. Requisitos Não-Funcionais. Ferramenta de Visualização de Software.

ABSTRACT

The problem of the large volume of data is present in the software industry, especially in the software monitoring log data, which is generated in real time, as the software is used. Some factors make it difficult to manually analyze log data. One is the large volume of data generated; another factor that makes it difficult to manually analyze the logs and the fact that they are usually scattered in one or several log files, presenting itself in a disorderly way. For this reason, a manual log analysis has been shown as an error-prone activity. Therefore, it is necessary to use some computational tool to support log analysis. In addition, in order to have a good interpretation of the analyzed data, it is necessary to contextualize them with abstract software models, such as software architecture. This work aims to develop a visualization approach to facilitate the data analysis process of monitoring the requirements of quality, efficiency and reliability, providing a greater contextualization with abstractions of the software architecture and a tool, called VisRQ, supporting this approach. This tool was evaluated by a semi-structured experiment, in which the users analyzed the monitoring logs and answered questionnaires in order to verify the fulfillment of the desired requirements. We also evaluated the tool learning time and whether it makes the log analysis process more accurate and faster. The experiments participants were students of the Informatics technical course of the Federal Institute of Alagoas - Campus Arapiraca and of the course of Computer Science of the Federal University of Alagoas - Campus Arapiraca, besides some professionals that work with Information Technology. The tool acceptance was positive, according to the responses of the evaluative experiment, 100% of the participants stated that they prefer to use the VisRQ tool instead of the previous process that is less automated and 97.6% considered that it facilitates the analysis process. The experiment shows that the log analysis process using VisRQ tool is more efficient, besides allowing the visualization of information about the nonfunctional requirements in a contextualized way with diagrams of the software architecture.

Key-words: Software Architecture. Data Visualization. Quality Requirements. Non-Functional Requirements. Software Visualization Tool.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACM	- *Association for Computing Machinery*
CPU	- *Central Processing Unit*
IEEE	- *Institute of Electrical and Electronics Engineers*
Falibras-TS	- *Falibras baseado em Transferência Sintática*
Falibras-MT	- *Falibras baseado em Memória de Tradução*
GQM	- *Goal-Question-Metric*
HEB	- *Hierarchical Edge Bundles*
ISO	- *Internacional Organization of Standardization*
ISO/IEC	- *Internacional Organization of Standardization/International Electrotechnical Commission*
JDK	- *Java Development Kit*
LIBRAS	- *Língua Brasileira de Sinais*
LOC	- *Lines of Code*
MTBF	- *Mean Time Between Failures*
MTF	- *Mean Time to Failure*
MTTR	- *Mean Time to Repair*
MVC	- *Model-View-Controller*
RF	- *Requisitos Funcionais*
RNF	- *Requisitos Não-Funcionais*
RS	- *Revisão Sistemática*
XMI	- *XML Metadata Interchange*
UFAL	- *Universidade Federal de Alagoas*
UML	- *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo, são apresentados: o contexto, os objetivos, a relevância, a metodologia, o escopo e a organização da dissertação.

1.1 CONTEXTO

O principal fator que afeta negativamente a qualidade de um sistema, em geral, é a sua complexidade, uma vez que está associada ao tamanho das especificações. Em software, o problema de complexidade e tamanho é ainda mais grave, em razão das interações entre os diversos componentes do sistema (KOSCIANSKI; SOARES, 2007). Ou seja, quanto maior for o tamanho das especificações, mais difícil será garantir a qualidade do software. Por isso, o monitoramento constante para garantir o funcionamento adequado torna-se fundamental. Essa necessidade é ainda mais evidente para a aferição dos requisitos não-funcionais (RNFs), também conhecidos como requisitos de qualidade (RQ). São exemplos desses requisitos a confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. Vários requisitos de qualidade só podem ser avaliados a partir de dados de monitoramento do sistema em execução, ocasionando a geração de uma grande quantidade de dados com o passar do tempo.

Esse grande volume de dados (*logs*) são difíceis de serem analisados sem o uso de uma ferramenta computacional, uma vez que os *logs* de um determinado componente podem estar espalhados pelo arquivo de registro, apresentando-se de forma desordenada, e em grande número, já que são gerados cada vez que o método monitorado é executado.

Outro fator que dificulta o processo de leitura e análise dos *logs* de monitoramento é a legibilidade, pois dependendo do formato para representar tais *logs* (seja JSON ou csv) a leitura dos dados torna-se mais complicada, resultando numa análise mais cansativa para o ser humano.

Diante disso, na literatura existem alguns trabalhos relacionados com monitoramento de requisitos de qualidade, como Silva (2015) e Lima (2016). Neles, foram desenvolvidas ferramentas para monitorar requisitos de qualidade dos softwares e gerar dados de *logs* utilizando programação orientada a aspectos. Lima (2016), em seu trabalho, criou um *framework* a partir da ferramenta de Silva (2015). Tal *framework* é baseado no uso de anotações, permitindo anotar todas as classes do sistema. Entretanto, anotar todas as classes não é viável e pode interferir diretamente no comportamento do sistema, uma vez que o tornaria muito lento e dificultaria consideravelmente a análise dos dados de monitoração (MENEZES, 2015). Também porque os dados gerados referem-se a partes espalhadas do software (como métodos) sem oferecer facilidades de contextualização,

agrupamento e muito menos facilidades para visualizar o grande volume de dados gerado.

Tentando minimizar a quantidade de dados gerados, Menezes (2015) propôs um processo centrado na arquitetura de software a fim de identificar pontos estratégicos que precisam ser monitorados, visando obter dados significativos de monitoração, mas sem comprometer o comportamento do sistema. Entretanto, mesmo diminuindo a quantidade de dados gerados, a análise dos *logs* continua difícil.

Dessa forma, torna-se necessário o desenvolvimento de uma abordagem de visualização para facilitar o processo de análise desses *logs*. Essa necessidade ficou evidente por meio de um experimento realizado por Menezes (2015), em que alguns participantes desistiram e outros reclamaram do processo de análise utilizado, que consistia em analisar os *logs* conforme foram gerados, diretamente no arquivo sem qualquer tratamento. Diante da dificuldade de análise, foi sugerido o desenvolvimento de uma ferramenta para auxiliar tal processo.

Assim, com base na monitoração realizada pela ferramenta de Lima (2016) e no processo desenvolvido por Menezes (2015), pretende-se criar uma abordagem e uma ferramenta para visualizar as informações extraídas dos arquivos de *log* do monitoramento de requisitos de qualidade, proporcionando uma maior contextualização semântica dos dados com os elementos da arquitetura do software. Tais informações podem auxiliar, por exemplo, o arquiteto de software a identificar pontos críticos na arquitetura da solução e até mesmo antecipar uma falha iminente.

Sendo assim, a questão de pesquisa que norteia este trabalho é: *"Como visualizar informações sobre requisitos não-funcionais a partir dos logs de monitoramento de um sistema e de modo contextualizado com a arquitetura de software?"*.

1.2 OBJETIVO

1. Objetivo Geral:
   Criar uma abordagem para visualização de informações extraídas de arquivos de *logs* que facilite a análise de dados de monitoramento dos requisitos de qualidade (Desempenho e Confiabilidade) e proporcione uma maior contextualização com abstrações da arquitetura do software.

2. Objetivos Específicos:
   Para realizar o objetivo geral deste trabalho, devem ser atingidos os seguintes objetivos específicos:

   - Realizar uma revisão literária sobre técnicas de visualização de informações.
   - Realizar uma revisão sistemática sobre ferramentas de visualização de informações relacionadas a requisitos não-funcionais.

- Extrair informações dos *logs* gerados pela ferramenta de monitoramento QuAM *Framework* (LIMA, 2016). Os arquivos de *log* podem conter dados de mais de um requisito de qualidade relacionado a um ou vários componentes de software que compõem a arquitetura do sistema.
- Planejar como as informações obtidas a partir dos *logs* serão exibidas e desenvolver uma abordagem de visualização. Dependendo da natureza da informação (numérica ou textual), a forma de visualizá-las pode ser diferente. Diante disso, é preciso analisar os tipos de dados/informações e escolher técnicas de visualização adequadas.
- Desenvolver uma ferramenta, que será chamada de VisRQ, para visualizar informações de monitoramento referentes aos requisitos de qualidade Desempenho e Confiabilidade, de acordo com abstrações da arquitetura do software. Tal ferramenta deve auxiliar desenvolvedores, arquitetos e engenheiros de software a visualizar as informações referentes aos requisitos de qualidade do software de forma contextualizada com a arquitetura.
- Avaliar o impacto da ferramenta de visualização VisRQ, utilizando-se de um experimento com desenvolvedores de software.

1.3 RELEVÂNCIA

A contribuição desse trabalho está relacionada à área de visualização de dados. Como a natureza dos dados refere-se ao monitoramento de requisitos de qualidade de software, a contribuição também está situada no contexto da Engenharia de Software.

A ferramenta de Lima (2016) apenas monitora e gera arquivos de *log* dos requisitos Confiabilidade e Desempenho. Entretanto, Menezes (2015) em seu trabalho, que utilizou o QuAM *Framework* (LIMA, 2016), identificou a necessidade do uso de uma abordagem/-ferramenta para analisar e visualizar os dados gerados por tal ferramenta de monitoração, incluindo a possibilidade de filtragem automática, devido ao grande volume de dados que pode ser gerado nos arquivos de *log* (MENEZES, 2015). Por isso, uma das contribuições deste trabalho consiste numa abordagem para visualizar informações referentes a esses requisitos, contextualizando os dados dos *logs* com os diagramas da arquitetura de software da aplicação monitorada. Dessa forma, será possível visualizar, de uma maneira mais eficiente, como os requisitos de qualidade estão relacionados entre si, com os demais requisitos e com os elementos arquiteturais do software.

A abordagem proposta é suportada por uma ferramenta, denominada VisRQ, que pode facilitar o processo de entendimento do software. Ao permitir o armazenamento e o relacionamento de vários diagramas ou visões de determinado componente de forma

contextualizada com a arquitetura, essa ferramenta contribui também para a organização da documentação e, com isso, entendimento do software.

Utilizando a ferramenta VisRQ, é possível analisar os *logs* de maneira mais eficiente e rápida, pois é permitido relacionar com os diagramas e também agrupar e organizar as informações. Dessa forma, o processo de análise é facilitado, permitindo observar várias informações simultaneamente. Com isso, houve uma economia significativa de tempo em relação a análise dos *logs* brutos diretamente no arquivo ou em relação ao uso de uma ferramenta desenvolvida para outra finalidade, como um editor de planilhas[1].

A ferramenta também oferece facilidades para acompanhamento da evolução histórica dos dados de monitoramento, com o intuito de facilitar a identificação de potenciais gargalos arquiteturais do sistema, favorecendo atividades de manutenção e evolução do software.

Outra contribuição desse trabalho é a realização da revisão sistemática sobre visualização de requisitos de qualidade contextualizada com a arquitetura de software. O resultado dessa revisão servirá de base para outros pesquisadores interessados nessa temática.

Assim, espera-se que a visualização de requisitos de qualidade oferecida pela ferramenta proposta possa servir de *feedback* para ajudar desenvolvedores, engenheiros e arquitetos de software a atenderem de maneira mais adequada os requisitos não-funcionais do sistema.

1.4 METODOLOGIA

Para desenvolver esta dissertação, foi realizada uma revisão sistemática da literatura. Para isso, foram utilizadas as bases ACM (*Association for Computing Machinery*), IEEE (*Institute of Electrical and Electronics Engineers*), *Science Direct*, *Scorpus*, *Springer* e *Engineering Village*; que são fontes de artigos científicos e cujos materiais são acessíveis pela *internet*. O objetivo desse levantamento foi encontrar trabalhos que tratasse da visualização de requisitos não-funcionais do sistema contextualizados com a arquitetura de software.

Depois disso, foi desenvolvida a ferramenta VisRQ, que suporta a abordagem de visualização proposta para visualizar dados sobre requisitos não-funcionais obtidos a partir de *logs* do monitoramento do sistema. Os *logs* utilizados no trabalho foram obtidos da monitoração realizada pela ferramenta QuAM *Framework* (LIMA, 2016) sobre o Sistema Falibras (BRITO; FRANCO; CORADINE, 2012).

A abordagem proposta e a ferramenta VisRQ foram avaliadas por um experimento, em que os usuários analisaram os *logs* de monitoramento e responderam alguns questi-

[1] Caso os *logs* estejam no formato .csv, um editor de planilhas pode exibir os dados, porém informações como a média de tempo de execução de um determinado componente (métodos, classes ou pacotes) não será obtida sem a manipulação dos dados pelo usuário.

onários para verificar se os requisitos levantados foram atendidos, para avaliar o tempo de aprendizado da ferramenta e se ela torna o processo de análise dos *logs* mais preciso e rápido. Nessa avaliação, foi utilizada a abordagem *Goal-Question-Metric* (GQM), conhecida por definir claramente os objetivos, as questões e as medidas. Buscou-se, também, avaliar se a ferramenta desenvolvida atendeu os requisitos levantados e se há melhoria no processo de análise dos *logs* com a utilização da ferramenta VisRQ.

1.5 ESCOPO DO TRABALHO

Este trabalho visa desenvolver uma abordagem para visualização de informações extraídas de arquivos de *logs* relacionados aos requisitos de qualidade Desempenho e Confiabilidade que as contextualize com a arquitetura de software. Para validação, foi implementada uma ferramenta que suporta essa abordagem, para auxiliar a análise e interpretação de dados do monitoramento desses requisitos. Por isso, a descrição da ferramenta VisRQ detalha a descrição da abordagem. Tal ferramenta tem como potenciais usuários: desenvolvedores, engenheiros, arquitetos de softwares e todos aqueles que necessitam analisar como os requisitos citados estão interagindo no sistema.

Assim, o suporte aos demais requisitos de qualidade e das métricas de software estão fora do escopo deste trabalho.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

O restante da dissertação está organizado da seguinte forma: no Capítulo 2 é apresentada a fundamentação do trabalho; no Capítulo 3 é mostrada a revisão sistemática realizada e os trabalhos relacionados; no Capítulo 4 é abordada a solução proposta: a abordagem e a ferramenta VisRQ; no Capítulo 5 é apresentado o planejamento do experimento avaliativo; no Capítulo 6 é realizada uma discussão, com base nas respostas dos participantes do experimento, sobre a avaliação da abordagem utilizando a ferramenta VisRQ; e, finalmente, no Capítulo 7 é apresentada a conclusão do trabalho e os direcionamentos para trabalhos futuros.

2 FUNDAMENTAÇÃO

Neste capítulo, são abordados alguns conceitos relacionados com as atividades desenvolvidas nessa pesquisa.

2.1 ENTENDIMENTO DE UM SISTEMA

Para realizar modificações em um sistema de software é necessário primeiro entendê-lo. Entretanto, o processo de compreensão de programas é difícil e custoso, ocasionado gasto de tempo e recursos financeiros (MCKEE, 1984; PRESSMAN, 2005; JULIANO, 2014). Diante disso, Juliano (2014) destaca algumas características do software que dificultam a sua compreensão, são elas:

- **Software é intangível**. Ele não possui uma forma definida ou tamanho (BALL; EICK, 1996);
- **Software é grande e complexo**. O tamanho e a complexidade dificultam sua compreensão, desde pequenas partes até o sistema como um todo. Além disso, a complexidade do software também está relacionada ao tamanho do mesmo (CASERTA; ZENDRA, 2011);
- **Software evolui**. Para acompanhar a rápida mudança do mercado, os produtos de software devem estar aptos a receber e aceitar mudanças. Com isso, a evolução dos softwares torna-se fator de sucesso para que empresas consigam se destacar no mercado (SOMMERVILLE, 2009). Conforme os produtos de software vão evoluindo, sua arquitetura e documentação tendem a aumentar e, consequentemente, a sua complexidade (LEHMAN, 1980).

Uma vez que a compreensão de programas não é uma tarefa trivial, torna-se necessária a utilização de ferramentas que facilite o processo de entendimento. Nesse cenário, a visualização de software pode contribuir significativamente.

2.2 VISUALIZAÇÃO DE SOFTWARE

A Visualização de Software utiliza a capacidade humana de lidar com informações visuais para proporcionar o entendimento do software (D'ARCE, 2012). Ela é uma subárea da visualização da informação e tem como objetivo representar um software a partir de três pontos de vista distintos: estático, dinâmico e de evolução (DIEHL, 2007; JULIANO, 2014).

A visualização estática busca mostrar as estruturas, relacionamentos e propriedades das entidades do software. Os dados de entrada para o modelo são obtidos, geralmente, do

código-fonte dos aplicativos sem que haja a execução do programa (CASERTA; ZENDRA, 2011; WARE, 2013);

A visualização dinâmica busca auxiliar a compreensão do comportamento do software pela exposição de informações obtidas durante a execução do software. São exemplos dessas informações a chamada de métodos e a troca de mensagens entre objetos (WARE, 2013).

A visualização evolutiva incrementa a visualização estática através da utilização de informações temporais (CASERTA; ZENDRA, 2011). Geralmente os dados que alimentam as visualizações são provenientes de controladores de versão (JULIANO, 2014).

Neste trabalho, os dados utilizados na pesquisa foram obtidos através da execução do software estudado, por isso esta pesquisa está enquadrada na área de visualização dinâmica.

Na área de visualização de informações, alguns conceitos, tais como tipo de dados e representação visual, merecem destaque. Diante disso, esses conceitos serão abordados a seguir.

2.2.1 Tipos de Dados e Representação Visual

Dados descrevem entidades ou processos que são objeto de análise ou estudo, correspondendo a atributos que podem ser caracterizados de acordo com diferentes critérios (FREITAS et al., 2001). Alguns critérios de classificação podem ser: classe de informação, tipo de valores, natureza e dimensão do domínio (conforme exposto na Tabela 1). Tal caracterização dos dados é importante pois, segundo Freitas et al. (2001), é a consideração inicial para escolher uma técnica de visualização.

Tabela 1 – Classificação de informações

Critério	**Classe**	**Exemplo**
Classe de Informação	Característica Escalar Vetor, Tensor Relacionamento	Gênero Temperatura Grandeza física associada a um fluído *Link* num hiperdocumento
Tipo dos valores	Alfanumérico Numérico Símbolo	Gênero Temperatura Link num hiperdocumento
Natureza do domínio	Discreto Contínuo Contínuo-Discretizado	Marcas de automóvel Superfície de um terreno Anos (tempo discretizado)
Dimensão do domínio	1D 2D 3D n-D	Medida de uma grandeza no tempo Superfície de um terreno 3D Volume de dados médicos n-D Dados de uma população

Fonte: Freitas et al. (2001).

Dessa forma, conforme o critério **classe de informação**, um dado pode representar: a) uma característica (categoria, atributo nominal ou ordinal), correspondendo a valores em um grupo restrito de elementos, com ou sem ordem; b) uma propriedade com valores escalares, vetoriais ou tensoriais, que assumem valores inteiros ou reais, dentro de certo intervalo; ou c) a existência de relacionamentos (hierarquia ou ligação) entre entidades.

Em relação ao **tipo**, um dado pode assumir valores alfanuméricos ou numéricos, dentre uma enumeração finita ou infinita, ou pode ser composto de valores simbólicos, representando a identificação de uma entidade ou fenômeno relacionado (VALIATI, 2008).

Por fim, os dados também podem ser caracterizados de acordo com a dimensão e a natureza do domínio no qual estão definidos. Com isso, os dados podem estar associados a um domínio unidimensional, bidimensional, tridimensional, ou n-dimensional [1]. Tal domínio pode ser contínuo, contínuo-discretizado ou discreto (FREITAS et al., 2001). Os exemplos de dados em cada uma dessas classificações podem ser observados na Tabela 1.

Segundo Freitas et al. (2001), representações visuais (ou gráficas) correspondem às "imagens"ou "figuras"empregadas para representar o conjunto de dados.

Em relação as representações visuais, Freitas et al. (2001) as classifica nas seguintes categorias (classes): gráficos, ícones, glifos e objetos geométricos, mapas e diagramas. Na Tabela 2, são apresentadas as categorias, os tipos e uma breve descrição de utilização das representações visuais apresentadas.

2.2.2 Técnicas de Visualização de Informações

Existe uma grande diversidade de técnicas de visualização de informações e nem todas se encaixam corretamente nas classificações existentes, uma vez que as características que separam cada classe são imprecisas e incompletas. Por isso, muitas representações visuais são consideradas híbridas de várias técnicas (VALIATI, 2008).

Diante disso, tais técnicas podem ser classificadas de diferentes formas, de acordo com vários critérios como: as tarefas suportadas, a estrutura do conjunto de dados, a natureza dos dados a serem visualizados, a dimensionalidade dos dados ou do espaço onde serão representados, a abordagem de mapeamento adotada pelas técnicas ou os métodos de manipulação e interação utilizados (FAYYAD; WIERSE; GRINSTEIN, 2002) (VALIATI, 2008).

De acordo com a taxonomia de Keim (2002), as técnicas de visualização podem ser divididas em: gráficos convencionais (para dados 1D a 3D); técnicas iconográficas; geométricas; orientadas a pixel; e técnicas baseadas em dimensões (também conhecidas como “técnicas hierárquicas ou baseadas em grafos” (KEIM; KRIEGEL, 1996)).

[1] Os dados n-dimensional também podem ser chamados de multidimensional ou multivariável (VALIATI, 2008)

Tabela 2 – Classes de representações visuais

Classe	**Tipo**	**Utilização**
Gráficos 2D, 3D	de pontos circulares de linhas de barras de superfícies (para 3D)	Representação da distribuição dos elementos no espaço domínio ou representação da dependência/correlação entre atributos.
Ícones Glifos Objetos geométricos	Elementos geométricos 2D ou 3D diversos	Representação de entidades num contexto ou representação de grupos de atributos de diversos tipos.
Ícones Glifos Objetos geométricos	Elementos geométricos 2D ou 3D diversos	Representação de entidades num contexto ou representação de grupos de atributos de diversos tipos.
Mapas	de pseudo-cores de linhas de superfícies de ícones, símbolos diversos	Representação de campos escalares ou de categorias. Representação de linhas de contorno de regiões, isovalores. Idem, no espaço 3D. Representação de grupos de atributos (categorias, escalares, vetoriais, tensoriais).
Diagramas	Nodos e arestas	Representação de relacionamentos diversos: É-um, É-parte-de, Comunicação, Sequência, Referência, etc.

Fonte: Freitas et al. (2001).

Card e outros (CARD; MACKINLAY; SHNEIDERMAN, 1999) afirmam que estruturas visuais unidimensionais são geralmente utilizadas para a apresentação de documentos de texto ou de linhas do tempo, podendo ser combinadas com um segundo ou terceiro eixo para mostrar comparação entre valores. Já as estruturas bidimensionais (como gráficos de linhas, barras, pizza e mapas) são mais utilizadas para visualizar dados estatísticos, descrever funções matemáticas e visualizar informações geográficas. As estruturas visuais 3D, por sua vez, são úteis para combinar duas ou mais visualizações 2D e para visualizar dados de elementos matemáticos ou físicos (CARD; MACKINLAY; SHNEIDERMAN, 1999 apud NASCIMENTO; FERREIRA, 2011).

Algumas técnicas de visualização podem ser vistas no Capítulo 3, em que serão apresentadas algumas ferramentas de visualização de software.

A abordagem proposta nesse trabalho utiliza cores, tabela, lista, gráfico de linha e diagramas da arquitetura (principalmente diagramas UML - Unified Modeling Language), em que o usuário pode navegar de uma visualização mais genérica para uma mais especifica, conforme será mostrado no Capítulo 4.

Nesse trabalho, o uso de cores foi adotado por permitir uma rápida diferenciação

das informações. Entretanto, colocar "a cor certa no lugar certo" não é trivial, uma vez que a associação inadequada entre os dados e as cores pode causar uma catástrofe visual (NASCIMENTO; FERREIRA, 2011). Vale ressaltar que o uso de cores é muito explorado por várias técnicas na apresentação de informações.

Já o gráfico de linha foi escolhido por exibir a comparação histórica de dados mostrando possíveis variações dos valores e os diagramas UML, por serem utilizados para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos (ALHIR, 1999 apud CASTRO, 2013), permitindo representar sistemas de software sob diversas perspectivas e facilitando a comunicação entre os envolvidos no processo de desenvolvimento (gerentes, coordenadores, analistas, desenvolvedores) por apresentar um vocabulário de fácil entendimento(GUEDES, 2014).

2.3 REQUISITOS NÃO-FUNCIONAIS OU ATRIBUTOS DE QUALIDADE

Existe uma relação direta entre arquitetura e requisitos (funcionais ou não-funcionais). Os requisitos de software são as características, atributos, propriedades e restrições do sistema. Os requisitos são classificados como Requisitos Funcionais (RF), que correspondem as funcionalidades do sistema, e Não-Funcionais (RNF) ou de qualidade (RQ), correspondendo a outras propriedades e restrições do software e que podem afetar o sistema como um todo (SOMMERVILLE, 2009; SILVA, 2015).

Os requisitos de qualidade preocupam-se com a maneira como o software executa as suas funcionalidades. Por isso, a especificação de tais requisitos deve ser realizada desde a etapa do projeto, uma vez que podem estar espalhados por diversos módulos do sistema, concorrendo com as funcionalidades e causando prejuízos de legibilidade, operabilidade e manutenibilidade do software (SILVA, 2015). Devido à sua grande importância, há um conjunto de normas que tratam dos principais atributos de qualidade, entre as quais está a norma ISO/IEC 9126 (STANDARDIZATION; COMMISSION, 2001a), que aborda os principais requisitos de qualidade: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. E a norma brasileira correspondente é a NBR ISO/IEC 9126-1 (ISO; NBR, 2003). Porém a norma ISO/IEC 9126 foi substituída em 2011 pela norma ISO/IEC 25010, que define as características de qualidade que todos os softwares devem ter e também modelos de avaliação da qualidade de software e sistemas (STANDARDIZATION; COMMISSION, 2001b).

De acordo com Pressman (2011), os requisitos de qualidade da norma ISO/IEC 9126 podem ser definidos da seguinte forma:

- *Confiabilidade.* A quantidade de tempo que o software fica disponível para uso.

- *Eficiência*. O grau de otimização do uso, pelo software, dos recursos do sistema.
- *Funcionalidade*. O grau com que o software satisfaz as necessidades declaradas.
- *Manutenibilidade*. A facilidade com a qual uma correção pode ser realizada no software.
- *Portabilidade*. A facilidade com a qual um software pode ser transposto de um ambiente [2] a outro.
- *Usabilidade*. Grau de facilidade de utilização do software.

Contudo, vale ressaltar que os atributos de qualidade da norma ISO/IEC 9126 fornecem uma base razoável para medidas indiretas e uma lista de verificação para avaliar a qualidade de um sistema (PRESSMAN, 2011). Diante disso, em seu trabalho, Silva (2015) mostra árvores de decisões (Figuras 1, 2, 3, 4, 5 e 6) para cada um dos seis atributos descritos acima, em que as cores de preenchimento dos retângulos arredondados correspondem: azul - nome do atributo; verde - local de monitoração; amarelo - ponto a ser monitorado; e salmão - formas de monitoração. O entendimento dessas árvores de decisão facilita a compreensão dos RNF da norma ISO/IEC 9126 (SILVA, 2015).

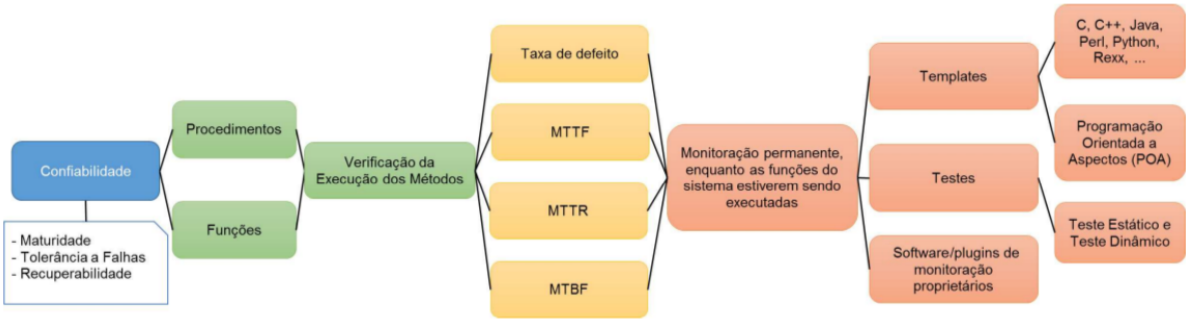
Conforme é possível visualizar na Figura 1, a medição da confiabilidade pode ser feita mediante a verificação das seguintes métricas: taxa de ocorrência de defeito, MTTF, MTTR e MTBF. Mais detalhes sobre essas métricas podem ser vistos na Tabela 3.

Em relação ao requisito usabilidade, é considerada: a facilidade da utilização do sistema, observando o tempo necessário para a aprendizagem e uso das funções; a irritação dos usuários, quando não conseguem usar o programa de forma adequada; a subutilização de recursos; a possibilidade de erros na execução; e o baixo rendimento (MORAES, 2002). Por isso, tal requisito é mensurado através da observação dos seguintes fatores: facilidade de aprendizado, eficiência de uso, facilidade de memorização, baixa taxa de erros e satisfação subjetiva (DIAS, 2007; SILVA, 2015). E, para realizar a medição da usabilidade, várias formas de avaliação podem ser utilizadas, como alguns métodos baseados em usuários: Método Coaching, Aprendizagem por Co-descoberta, Método de Ensino, Método Shadowing, Teste Remoto, Grupos de Foco e Entrevistas (CATECATI et al., 2011); ou métodos baseados em especialistas, como: Passo a Passo Cognitivo, Avaliação Heurística e Inspeção de Características e Funcionalidades (SILVA, 2015).

Em relação ao requisito manutenibilidade, ele pode ser influenciado por alguns fatores ligados ao processo do ciclo de vida do software, tais como: a documentação, a arquitetura, a tecnologia e o código fonte. Para mensurar esse requisito, algumas das principais métricas utilizadas são: complexidade ciclomática, profundidade de herança, acoplamento da classe e número de linhas de código (SILVA, 2015).

[2] O termo ambiente faz referência tanto à plataforma de hardware quanto ao conjunto de produtos de software auxiliares com os quais um determinado software irá interagir (SILVA, 2015).

Figura 1 – Árvore de Decisão - Monitoração do Atributo Confiabilidade.



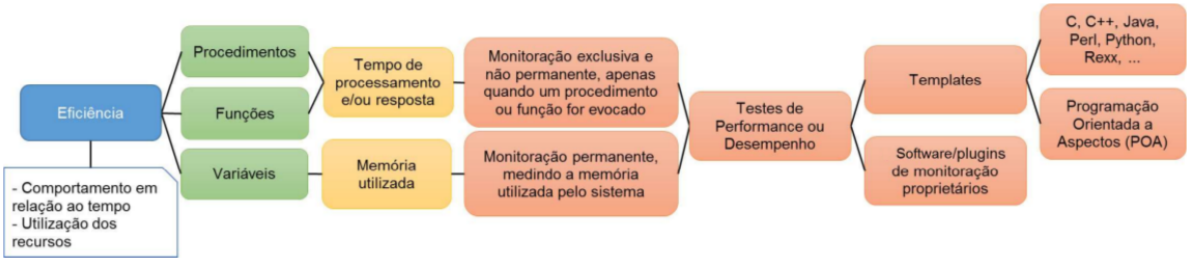
Fonte: Figura obtida de Silva (2015).

Tabela 3 – Definição das medidas de confiabilidade.

Métricas	**Definição**	**Questão**
Taxa de defeito - *failure rate*	Elaborar relatório técnico com informações dessas ferramentas.	Com que frequência ocorrem defeitos?
MTTF - *mean time to failure*	Coletar os *logs* do monitoramento de um sistema em funcionamento.	Qual o tempo até o primeiro defeito?
MTTR - *mean time to repair*	Levantar os requisitos da solução desejada.	Qual o tempo gasto para reparar cada defeito?
MTBF - *mean time between failure*	Definir um esboço da solução.	Qual o tempo entre um defeito e outro?

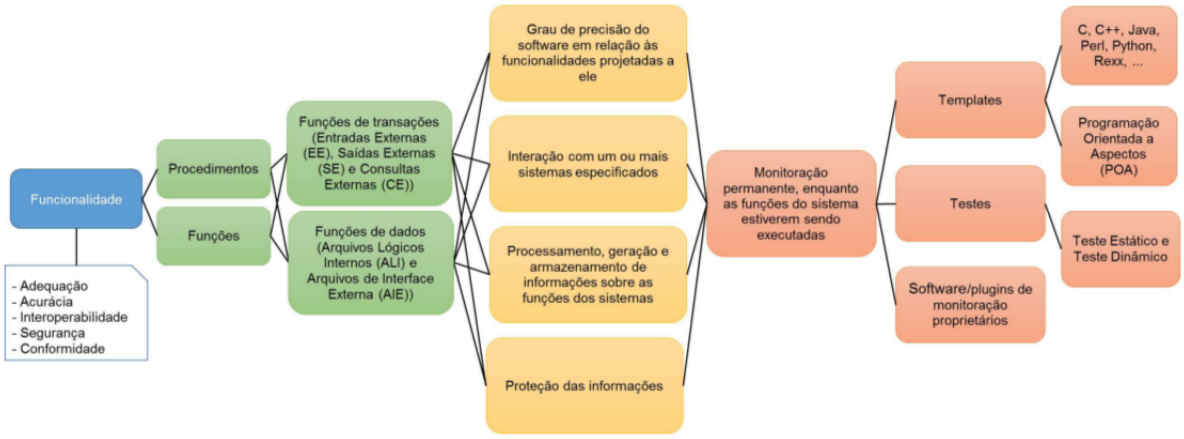
Fonte: Adaptado de Weber (2002) apud Silva (2015).

Figura 2 – Árvore de Decisão - Monitoração do Atributo Eficiência.



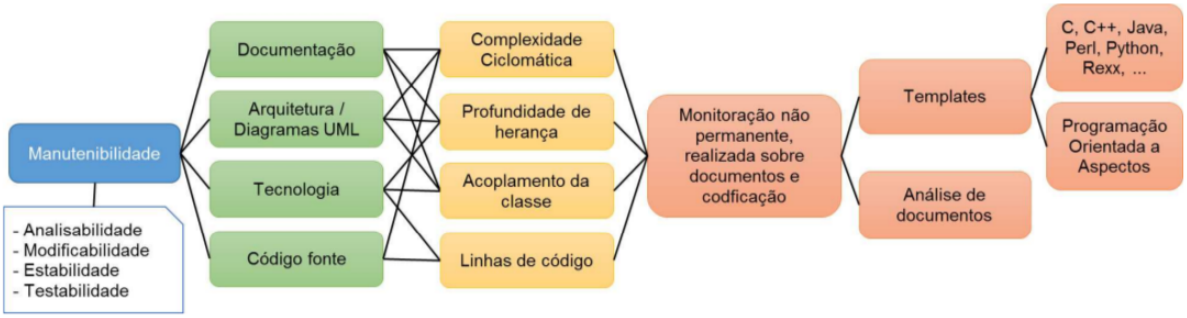
Fonte: Figura obtida de Silva (2015).

Figura 3 – Árvore de Decisão - Monitoração do Atributo Funcionalidade.



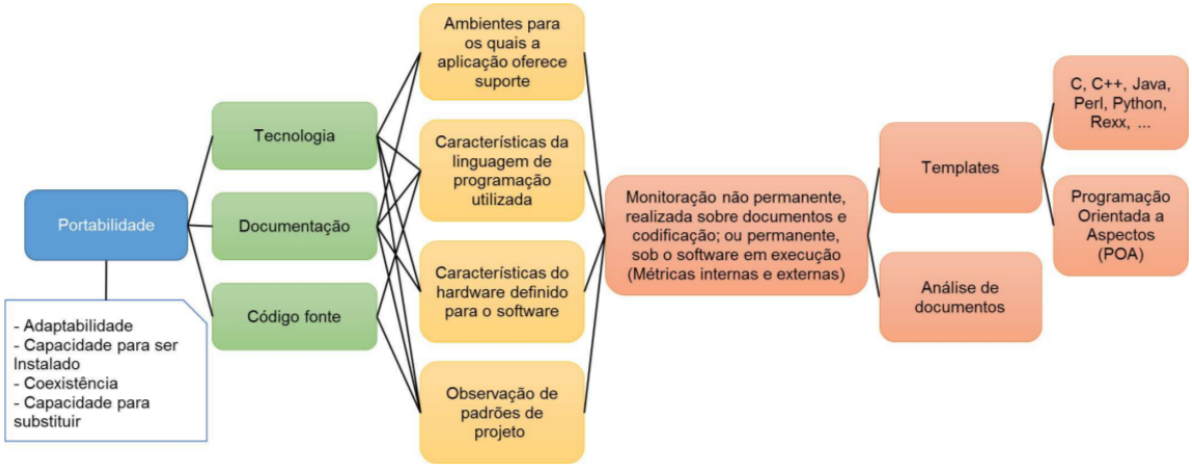
Fonte: Figura obtida de Silva (2015).

Figura 4 – Árvore de Decisão - Monitoração do Atributo Manutenibilidade.



Fonte: Figura obtida de Silva (2015).

Figura 5 – Árvore de Decisão - Monitoração do Atributo Portabilidade.



Fonte: Figura obtida de Silva (2015).

Figura 6 – Árvore de Decisão - Monitoração do Atributo Usabilidade.

Fonte: Figura obtida de Silva (2015).

Em relação ao requisito portabilidade, para medi-lo, Filho (2005) apresenta algumas métricas que devem ser observadas, são elas: adaptabilidade ao ambiente de hardware, adaptabilidade ao ambiente de software, grau de portabilidade, esforço para instalar, flexibilidade de instalação, disponibilidade de coexistência, consistência de funcionalidades, adaptabilidade ao ambiente de hardware, adaptabilidade ao ambiente de software, flexibilidade de instalação, coexistência apresentada e facilidade de migração (FILHO, 2005) (SILVA, 2015).

Assim, neste trabalho, apenas os requisitos Confiabilidade e Desempenho foram abordados pelo fato da ferramenta de monitoração dar suporte ao monitoramento desses requisitos e pela limitação de tempo para a realização e conclusão da pesquisa, uma vez que a implementação do suporte aos demais requisitos ultrapassaria o prazo determinado.

2.4 ARQUITETURA DE SOFTWARE E VISÕES ARQUITETURAIS

Existem vários tipos de sistemas baseados em software. Como exemplo, é possível citar:

- *Inteligência artificial* - simulam ou ampliam a cognição, locomoção ou outros processos orgânicos humanos;
- *Comercial ou sem fins lucrativos* - fundamentais para a operação de um empreendimento comercial;
- *Comunicações* - fornecem a infraestrutura para a transferência e o gerenciamento de dados, para conectar usuários desses dados ou para apresentar dados na periferia de uma infraestrutura;
- *Autoria de conteúdo* - são utilizados para criar ou manipular artefatos de multimídia ou texto;
- *Dispositivos* - interagem com o mundo físico para fornecer algum serviço;

- *Esporte e entretenimento* - gerenciam eventos públicos ou que geram uma experiência de entretenimento a um grande grupo;
- *Financeiros* - fornecem a infraestrutura para transferir e administrar dinheiro e outros valores;
- *Jogos* - geram uma experiência de entretenimento para indivíduos ou grupos;
- *Governo* - dão apoio à conduta e operações de uma entidade política (municipal, estadual, federal);
- *Industriais* - simulam ou controlam processos físicos;
- *Legais* - dão apoio ao setor judiciário;
- *Médicos* - dão suporte ao diagnóstico de doenças ou contribuem para pesquisas médicas;
- *Militar* - para consultas, comunicações, comando, controle e inteligência, armamento;
- *Sistemas operacionais* - situados logo acima do hardware para fornecer serviços básicos de software;
- *Plataformas* - situados acima do sistemas operacionais para fornecer serviços avançados;
- *Científicos* - utilizados para pesquisa e aplicações científicas;
- *Ferramentas* - utilizados para desenvolver outros sistemas;
- *Transportes* - controlam veículos de navegação, terrestres, aéreos ou espaciais;
- *Serviços públicos* - interagem com outros produtos de software para fornecer algum serviço;
- *Entre outros.*

Consequentemente, também existem vários gêneros (domínios) de arquitetura de software, cada um deles é adequado para um determinado tipo de aplicação (PRESSMAN, 2011). Por isso é importante conhecer e entender a arquitetura do software, uma vez que é a base do sistema (SOMMERVILLE, 2007).

Diante disso, para Pressman, a arquitetura de software representa uma estrutura em que algum conjunto de entidades (denominados componentes) é interligado por um conjunto de relacionamento definido (denominados conectores) (PRESSMAN, 2011).

Para Shaw e Garlan, a arquitetura de software envolve a descrição dos elementos que constitui um sistema, as interações entre esses elementos, os padrões e as restrições que direcionam sua composição (SHAW; GARLAN, 1996; OLIVEIRA, 2011).

Já para Sommerville, a arquitetura de software é um *framework* fundamental para estruturar o sistema, influenciando os requisitos de qualidade. Essa influência ocorre porque as decisões de projeto da arquitetura incluem a escolha do tipo de aplicação, a distribuição do sistema, os estilos arquiteturais a serem usados e as formas que a arquitetura deve ser documentada e avaliada (SOMMERVILLE, 2007).

Nesse contexto, dependendo da forma como a arquitetura do software é estruturada, ela pode afetar o desempenho, a robustez, a facilidade de distribuição e a manutenção. Por exemplo, se:

- *O desempenho for importante*: a arquitetura deve possuir componentes com granularidade maior e restringir as operações mais importantes dentro de um pequeno número de subsistemas, para que a comunicação seja a menor possível entre os componentes;
- *A proteção for importante*: deve ser utilizada uma estrutura em camadas com os itens mais importantes protegidos nas camadas mais internas e com um alto nível de validação de proteção nessas camadas;
- *A segurança for importante*: as operações relacionadas com a segurança devem ficar todas localizadas em um único subsistema ou em um pequeno número de subsistemas, reduzindo os custos e os problemas de validação;
- *A disponibilidade for importante*: deve incluir componentes redundantes, para que seja possível substituir e atualizar componentes, sem a interrupção do sistema;
- *A facilidade de manutenção for importante*: deve utilizar componentes encapsulados de menor granularidade.

Entretanto, podem existir conflitos ao implementar os requisitos não-funcionais em algumas arquiteturas. Diante desse fato, Silva (2014) apresentou a potencial relação entre os requisitos de qualidade Eficiência, Manutenibilidade, Usabilidade, Confiabilidade, Testabilidade, Interoperabilidade, Segurança e Portabilidade; apontando os *trade-off* entre eles (Tabela 4). Na Tabela 4 as relações mostradas apresentam as seguintes classificações: cooperativa (+), neutra (0) ou conflituosa (-).

Por exemplo, a facilidade de manutenção é melhorada pelo uso de componentes de menor granularidade, já a eficiência é melhorada pelo uso de componentes de maior granularidade. Caso esses requisitos sejam necessários, será preciso encontrar uma solução intermediária satisfatória (SOMMERVILLE, 2003).

Tabela 4 – Potencial Relacionamento entre Atributos de Qualidade.

	Eficiência	**Manutenibilidade**	**Usabilidade**	**Confiabilidade**	**Testabilidade**	**Interoperabilidade**	**Segurança no Acesso**	**Portabilidade**
Eficiência	+	−	+	−	−	−	−	−
Manutenibilidade	−	+	0	−	0	−	0	+
Usabilidade	+	0	+	0	−	0	−	+
Confiabilidade	−	−	0	+	+	0	+	0
Testabilidade	−	0	−	+	+	+	+	+
Interoperabilidade	−	−	0	0	+	+	0	0
Segurança no Acesso	−	0	−	+	+	0	+	0
Portabilidade	−	+	+	0	+	0	0	+

Fonte: Tabela obtida de Silva (2014).

Conforme exposto, como a estrutura ou o estilo arquitetural escolhido para uma aplicação está relacionado com os requisitos não-funcionais, compreender a arquitetura de software é vital para a construção e manutenção de sistemas de software.

Diante disso, Dwivedi e Rath (2014) relacionaram diferentes estilos arquiteturais com atributos de qualidade. Essas relações podem ser vista na Tabela 5, em que é mostrado o quanto alguns estilos arquiteturais podem suportar determinado RNF. Na Tabela 5, são abordados os seguintes estilos arquiteturais: *Batch-Sequencial* (BS), *Pipe-and-Filter* (PF), *Virtual Machine*(VM), *Client-Server* (CS), *Publish-Subscriber* (PS), *Event-Based* (EB), *Peer-to-Peer* (PP), *Component and connector* (C2) e *Common Object Request Broker Architecture* (CORBA). Nessa tabela, o símbolo "++" indica que o estilo analisado suporta bem determinado atributo de qualidade. Já o símbolo "+" significa que o estilo suporta razoavelmente certo atributo de qualidade. O símbolo "0" mostra que o estilo não afeta o atributo de qualidade. E o símbolo "-" indica que o estilo tem um impacto negativo para o atributo de qualidade.

Porém, pelo fato da arquitetura de software ser uma entidade conceitual intangível, torna-se necessário um mapeamento visual para diminuir o esforço cerebral, facilitando o entendimento (PAIVA, 2015).

Por isso, Bass, Clements e Kazman (2003) sugeriram cinco estruturas de arquiteturas canônicas ou fundamentais, são elas:

- *Estrutura Funcional.* Em que os componentes representam entidades funcionais ou de processamento. Os conectores representam interfaces que possibilitam usar ou passar dados para um componente e as propriedades representam a organização das

Tabela 5 – Relação entre estilos arquiteturais e RNFs.

RQs/Estilos	**BS**	**PF**	**VM**	**CS**	**PS**	**EB**	**PP**	**C2**	**CORBA**
Eficiência	0	0	-	-	+	+	0	0	-
Complexidade	0	0	0	0	+	+	++	+	++
Escalabilidade	0	+	+	+	0	+	+	+	0
Heterogeneidade	-	-	+	-	+	+	0	++	++
Adaptabilidade	0	-	0	0	+	0	0	+	++
Portabilidade	0	0	++	0	++	+	0	++	+
Confiabilidade	0	0	0	-	-	-	++	+	0
Segurança	0	0	0	++	+	0	+	0	-

Fonte: Tabela obtida de Dwivedi e Rath (2014) apud Menezes (2015).

interfaces e a natureza dos componentes.

- *Estrutura de Implementação*. Os componentes podem ser pacotes, classes, objetos, procedimentos, funções, métodos, etc., empacotando funcionalidades em vários níveis de abstração (BASS; CLEMENTS; KAZMAN, 2003). Os conectores controlam e passam os dados enquanto que as propriedades estão concentradas nos requisitos de qualidade.

- *Estrutura de Concorrência*. Os componentes representam unidades de concorrência organizadas como tarefas paralelas (ou *threads*). Os conectores incluem as relações: *sincronizações-com*, *é-de-maior-prioridade-que*, *envia-dados-a*, *não-é-possível-executar-sem* e *não-é-possível-executar-com*. Já as propriedades são: prioridade, preempção e tempo de execução (BASS; CLEMENTS; KAZMAN, 2003).

- *Estrutura Física*. Os componentes são o hardware onde está o software e os conectores são as interfaces entre os componentes de hardware. Já as propriedades são: a capacidade, largura de banda, desempenho e etc.

- *Estrutura de Desenvolvimento*. Os componentes e artefatos são definidos a medida que o processo de engenharia do software prossegue. Os conectores representam os relacionamentos entre os artefatos e as propriedades que caracterizam cada item (BASS; CLEMENTS; KAZMAN, 2003; PRESSMAN, 2011).

Ou seja, para facilitar o entendimento da arquitetura, os aspectos são separados em partes mais simples que o todo, chamadas de visões, que tem o objetivo de gerenciar a complexidade. Essa necessidade de descrever uma arquitetura de software utilizando várias visões originou o conceito de visões arquiteturais, que tem o objetivo de fornecer diferentes formas de representar a arquitetura de um sistema, de acordo com interesses específicos (KRUCHTEN, 1995; OLIVEIRA, 2011).

Contudo, apesar da existência de várias abordagens na literatura para a descrição de arquiteturas, como as citadas em Shaw e Garlan (1996) e Medvidovic e Taylor (2000), muitos trabalhos fazem uso da notação UML (*Unified Modeling Language*) para representar a arquitetura de sistemas orientados a objetos (AVGERIOU; GUELFI; MEDVIDOVIC, 2004; KÜMMEL, 2007). Utilizando a notação UML é possível descrever o projeto arquitetural sobre diferentes perspectivas e níveis de detalhamento e também incluir elementos adicionais por meio da criação de perfis e esteriótipos. Com isso, a notação UML proporciona uma forma para a criação, visualização e documentação de projetos arquiteturais de sistemas (ESTIVALETE, 2007), envolvendo itens concretos (e.g., classes e componentes reutilizáveis) e aspectos conceituais (e.g., processos de negócio e funções de sistema) (BOOCH, 2005) além de permitir representar diferentes visões arquiteturais (OLIVEIRA, 2011).

Um exemplo clássico de visão arquitetural utilizado na engenharia de software é o modelo conhecido como *visões 4+1*, composto por cinco visões arquiteturais que descreve os seguintes níveis de abstração:

- Visão lógica: Descreve os requisitos funcionais do sistema, em que classes e objetos são os principais elementos dessa visão. Ela representa tanto estrutura estática quanto estrutura dinâmica do sistema e é representada, na UML, através de diagramas de classes e de diagramas dinâmicos (sequência, colaboração, atividades,etc.).

- Visão do desenvolvimento (implementação): Descreve a organização do sistema em módulos de código. Na UML, ela é representada pelos diagramas de pacotes ou de componentes, em que podem ser detalhados os módulos, bibliotecas e as dependências entre esses elementos.

- Visão de processos (concorrência): Descreve os processos do sistema e como eles se comunicam, permitindo avaliar requisitos não-funcionais relacionados a desempenho, tolerância a falhas e disponibilidade. Na UML, ela é representada pelos diagramas dinâmicos, como diagramas de implantação e componentes. Essa visão é importante para a construção de sistemas que precisam de temporização e sincronização, tais como sistemas de controle e de tempo real. E também é útil quando há processos ou *threads* concorrentes.

- Visão física (implantação): Descreve o sistema como um conjunto de elementos que se comunicam e que são capazes de realizar processamento (tais como computadores e outros dispositivos). Na UML, essa visão é representada pelos diagramas de implantação. E tal visão também permite avaliar alguns requisitos não-funcionais (desempenho, disponibilidade, confiabilidade, escalabilidade).

- Visão de cenário (caso de uso): Descreve as funcionalidades oferecidas pelo sistema aos seus usuários, detalhando ações e condições em cada caso de uso. Essa visão

é responsável por integrar as outras quatro e é representada, na UML, através de diagramas e especificações de casos de uso (KRUCHTEN, 1995; RUBIRA; BRITO, 2007).

Esse modelo (4+1) se tornou praticamente um padrão na indústria (JACOBSON et al., 1999) e na academia (BASS; CLEMENTS; KAZMAN, 2003; RUBIRA; BRITO, 2007).

2.5 TESTES DE SOFTWARE BASEADOS NA ESPECIFICAÇÃO

Para a geração dos *logs* a partir de testes do sistema monitorado, foi utilizada a estratégia de testes de software baseados na especificação, mais conhecidos como "teste caixa preta" ou "teste comportamental". Esse tipo de teste permite derivar conjuntos de condições de entrada que vão exercitar plenamente todos os requisitos funcionais de um programa (PRESSMAN, 2011).

A estratégia de teste caixa preta recebe esse nome pois nela o software é tratado como uma caixa em que só é conhecida a sua entrada e a sua saída, sem utilizar qualquer conhecimento de como ele é internamente (SANTOS, 2011). Esse tipo de teste possui algumas técnicas, como a partição de equivalência e a análise dos valores limite, que foram utilizadas para a modelagem dos casos testes.

A técnica de partição de equivalência tem como objetivo reduzir o conjunto de entradas de teste (que pode ser grande ou infinito) a um conjunto pequeno, finito e eficiente (GUIMARãES, 2016).

Com isso, busca-se definir um caso de teste com maior probabilidade descobre classes de erros, reduzindo o número total de casos de testes que precisam ser desenvolvidos (PRESSMAN, 2011), já que as entradas são divididas em grupos que tenham um comportamento similar, o que possibilita o mesmo tratamento para qualquer elemento do grupo (CAMPOS, 2009).

Já a técnica de análise de valores limite complementa a abordagem anterior. Por razões que não são claras, uma quantidade maior de erros tende a ocorrer na fronteira do domínio de entrada do que quando comparadas com entradas no centro da partição (PRESSMAN, 2011). Por isso, a análise de valores limite recomenda que sejam testadas as fronteiras das partições de equivalência (SANTOS, 2011). Para ilustrar, tomemos um sistema hipotético de cálculo de média, que recebe dois valores reais relativos a duas notas e retorna a média. Possíveis partições de equivalência para os valores de entrada são: (1) valores de notas aprovadas; (2) valores de notas reprovadas; e (3) valores de notas inválidas. A técnica de valores limite complementa as referidas partições de equivalência, recomendando valores limite em cada partição. Por exemplo, para a partição de notas aprovadas, bons casos de teste poderiam ser: *7*; *7,1*; *6,9*; *9,9*; *10* e *10,1*. Exercitando dessa forma os valores limite e suas fronteiras mais próximas.

3 REVISÃO SISTEMÁTICA

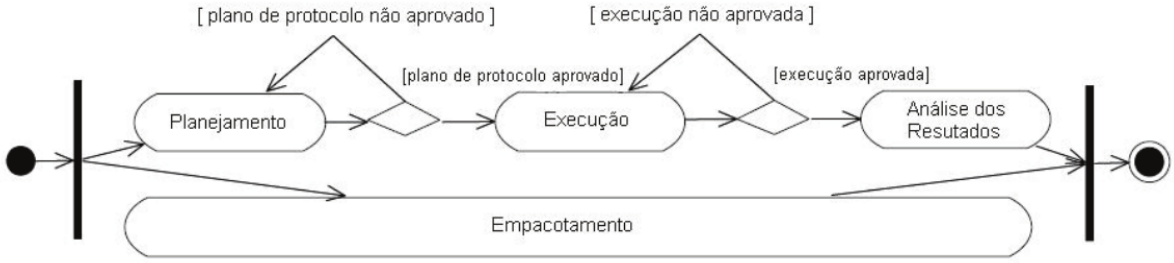
Neste capítulo, são apresentadas as etapas para a realização da revisão sistemática sobre o ponto central deste trabalho: visualização de requisitos não-funcionais do sistema contextualizado com a arquitetura de software.

3.1 VISÃO GERAL

Revisão Sistemática (RS) é uma metodologia de estudo que visa realizar um levantamento formal do estado da arte sobre determinada área, de maneira robusta e consistente, a partir de um criterioso planejamento e execução (BIOLCHINI et al., 2005; COLLABORATION; REVIEWERS'HANDBOOK et al., 2000; KHAN et al., 2001; PAI; MCCULLOCH; COLFORD, 2004; MUNZLINGER; NARCIZO; QUEIROZ, 2012).

Segundo Montebelo et al. (2007), uma RS deve ser conduzida seguindo etapas bem definidas de acordo com um protocolo previamente elaborado. Por isso são várias as atividades e etapas que compõem uma Revisão Sistemática, tornando sua execução demorada e trabalhosa. Essas etapas são mostradas na Figura 7 e detalhadas na Tabela 6, em que é apresentado um resumo de cada uma delas. Tais etapas podem possuir iterações, permitindo que muitas atividades sejam iniciadas na fase de planejamento e refinadas posteriormente.

Figura 7 – Processo de Condução de Revisão Sistemática.



Fonte: Biolchini et al. (2005) apud Montebelo et al. (2007).

3.2 PLANEJAMENTO DA REVISÃO SISTEMÁTICA

A revisão sistemática realizada teve como objetivo: revisar e analisar, por meio de publicações científicas, o estado da arte sobre ferramentas e métodos de visualização dos requisitos não-funcionais (eficiência e confiabilidade principalmente) presentes na arquitetura de software. E, para atingir tal objetivo, foram definidas algumas questões de pesquisa, são elas:

Tabela 6 – Etapas da Revisão Sistemática.

Passo	**Objetivos**	**Etapas**	**Resumo**
Planejamento	Planejar o objetivo central da revisão sistemática	- Identificação da necessidade de uma revisão; - Criação do protocolo de revisão;	- Objetivos da pesquisa são definidos; - Protocolo de revisão é definido e validado;
Execução	Executar as etapas planejadas no passo anterior e coletar material para análise	- Identificação da pesquisa; - Seleção dos estudos primários; - Estudo de avaliação da qualidade;	- Identificação de estudos primários; - Seleção e avaliação dos estudos primários, de acordo com os critérios de inclusão e exclusão;
Análise dos Resultados	Sintetizar os estudos primários que atendem ao propósito da revisão	- Extração de dados; - Síntese dos dados.	- Dados dos artigos são extraídos e sintetizados;

Fonte: Tabela adaptada de Montebelo et al. (2007).

1. Quais são as principais ferramentas utilizadas para visualizar os requisitos não-funcionais do software?

2. Quais são os principais recursos utilizados para visualizar informações sobre requisitos não-funcionais?

A pesquisa dos artigos foi realizada em bibliotecas digitais utilizando os seus respectivos engenhos de busca. As fontes de pesquisa escolhidas são bases de dados eletrônicas indexadas, como ACM (*Association for Computing Machinery*), IEEE (*Institute of Electrical and Electronics Engineers*), *ScienceDirect, Scopus*, etc. Tais fontes foram escolhidas devido à facilidade de acesso a uma grande quantidade de trabalhos.

A pesquisa foi restrita a trabalhos publicados a partir de 2005 e nas línguas portuguesa e inglesa. Essas línguas foram escolhidas pelos seguintes motivos: aquela por ser a língua utilizada na escrita e desenvolvimento deste trabalho e esta por ser a língua mais utilizada internacionalmente em divulgação científica.

A busca dos trabalhos da língua inglesa utilizou inicialmente a seguinte *string* de busca: **Pesquisa1:** *("quality attributes"OR "quality goals"OR "non-behavioral requirements"OR "quality of service requirements"OR "non functional requirements"OR "constraints"OR "ilities") AND ("software visualization"OR "data visualization"OR "information visualization") AND "software architecture"*.

Como o número de artigos encontrado foi pequeno (apenas 37), foi removido o termo *AND "software architecture"*. Dessa forma, a nova *string* de busca é: **Pesquisa2:** *("quality attributes"OR "quality goals"OR "non-behavioral requirements"OR "quality of service*

requirements"OR "non functional requirements"OR "constraints"OR "ilities") AND ("software visualization"OR "data visualization"OR "information visualization").

Tais *strings* foram utilizadas como base para as buscas, pois foi necessário adequá-la a sintaxe dos mecanismos de buscas de cada repositório digital. E os termos e as relações lógicas foram escolhidos com o objetivo de identificar potenciais trabalhos que respondam as perguntas da pesquisa.

Para facilitar o processo de análise e seleção dos artigos, foram elaboradas as seguintes perguntas:

- Critérios de inclusão

  (I1) O trabalho aborda a visualização de dados/informação de requisitos não-funcionais?

- Critérios de exclusão

  (E1) O trabalho foi publicado antes de 2005?

  (E2) O trabalho foi escrito em uma língua diferente da inglesa ou portuguesa?

  (E3) O trabalho está incompleto ou é um tutorial?

  (E4) O trabalho está sem conclusão?

  (E5) O trabalho não apresenta uma ferramenta que visualiza dados/informações de requisitos não-funcionais do sistema contextualizados com a arquitetura?

3.2.1 Processo de Seleção e Sumarização dos Estudos

Durante o processo de seleção preliminar, foi construída e submetida nos engenhos de pesquisa uma *string* de busca formada pelas palavras-chave identificadas. Em seguida, os trabalhos recuperados foram armazenados em um software de organização de referências bibliográficas, o StArt. Depois disso, foi realizada a leitura do título, do resumo e das palavras-chave dos trabalhos recuperados (Análise 1). Caso o trabalho fosse considerado relevante, foram lidas a introdução e a conclusão (Análise 2). Após a Análise 2, os trabalhos que não foram rejeitados foram lidos completamente (Análise 3). Passando pelos critérios de exclusão, os trabalhos foram documentados de acordo com os critérios definidos e as obras repetidas foram registradas uma única vez.

A seleção final consistiu na leitura completa dos trabalhos selecionados na etapa de seleção preliminar. Nela, foi feita uma síntese geral e algumas considerações sobre os resultados observados nos trabalhos lidos, buscando destacar:

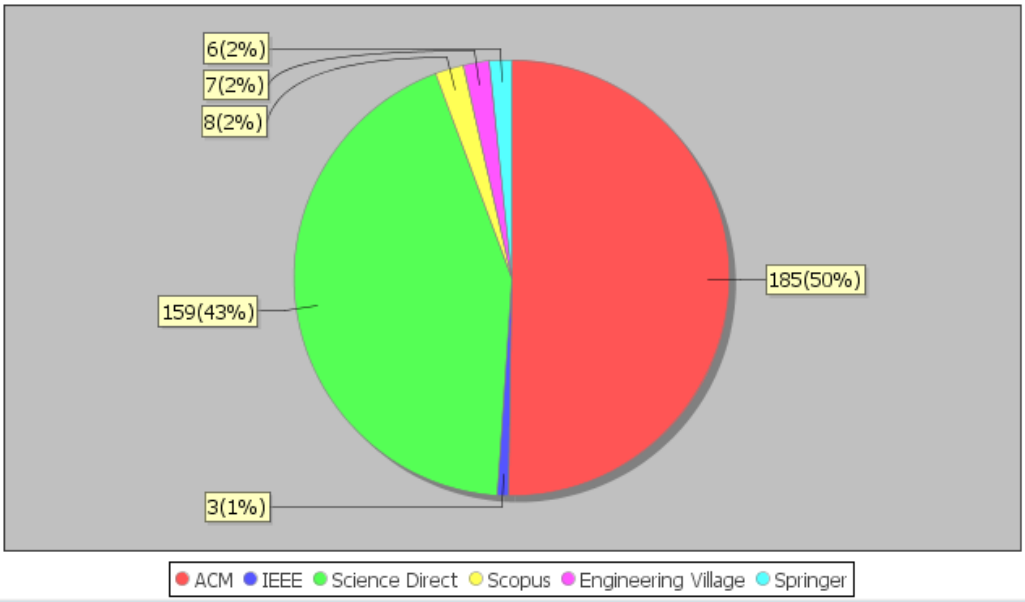
1. O nome da ferramenta;
2. O ano de publicação;

3. O objetivo da ferramenta;
4. As funcionalidades principais da ferramenta;
5. Como os dados são visualizados;
6. Dados de entrada.

3.3 RESULTADOS

A pesquisa nas bases das bibliotecas digitais foi realizada no dia 26 de agosto de 2016. Utilizando as duas *strings* de pesquisas mostradas anteriormente Pesquisa1 e Pesquisa2, foram encontrados 358 artigos. Porém, um dos arquivos com as referências baixadas possuía 3 trabalhos a mais, totalizando 361 artigos. Desses artigos, 178 são do repositório da ACM, 3 do IEEE, 159 do *Science Direct*, 8 do *Scopus*, 6 da *Springer* e 7 da *Engineering Village*, conforme a Figura 8. E dos 361 artigos, 66 (18%) são duplicados, 295 (80%) foram rejeitados e apenas 7 passaram para a segunda fase (Figura 9).

Figura 8 – Repositórios dos artigos.

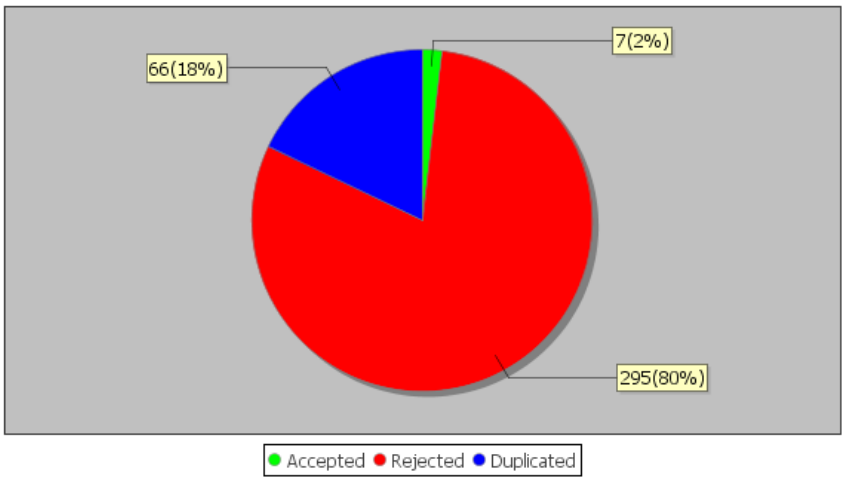


Fonte: Elaborada pelo autor.

Na segunda fase (Análise 2), foram lidas a introdução e a conclusão dos trabalhos selecionados e foi identificado que dois artigos eram diferentes mas falavam das mesmas ferramentas e um dos artigos foi rejeitado por não estar relacionado com o objetivo da pesquisa. Com isso, restaram 6 artigos para serem lidos completamente.

Na terceira fase (Análise 3), dos seis artigos lidos completamente apenas 4 apresentavam ferramentas de visualização de requisitos de qualidade e 2, abordagem para visualizar informações sobre RNFs.

Figura 9 – Classificação dos artigos importados para análise.



Fonte: Elaborada pelo autor.

Um desses dois trabalhos apresentou uma abordagem que foca na ilustração das características de modificabilidade: coesão e acoplamento (HOLVITIE; LEPPÄNEN, 2014). Para isso, utilizou grafos para representar os relacionamentos entre os componentes. E o segundo trabalho, apresentou uma abordagem que integrava cartografia de software (uso de mapas de software) com engenharia de desempenho de software para melhorar a interpretação dos resultados de previsão de desempenho bruto (KROGMANN et al., 2009). Como esses trabalhos não apresentaram uma ferramenta, não foram considerados.

Através da revisão sistemática, também foi encontrado o trabalho de Mattila et al. (2016), em que é apresentado o resultado de uma revisão sistemática da literatura sobre a área de visualização de software no período de 2010 até 2015. Nesse trabalho, foram analisados 83 artigos sobre estrutura, comportamento e evolução de software, abordando os possíveis usuários das ferramentas, a motivação da visualização e as técnicas utilizadas. Nesses artigos, foram identificadas 44 ferramentas de visualização. Porém, não foram apresentadas informações sobre visualização de requisitos de qualidade ou sobre visualização contextualizada com a arquitetura de software. E nem foi considerada a ligação da arquitetura de software com os requisitos não-funcionais.

Por isso, devido à escassez de trabalhos na temática dessa pesquisa, outros artigos foram analisados e revisitados, com o objetivo de categorizar e selecionar aqueles que, de alguma forma, apresentem semelhanças com este trabalho. Ou seja, também foram analisados trabalhos com foco em visualização de dados relacionados com arquitetura de software, evolução, *bug report*, etc. Tais trabalhos foram obtidos por indicações dos mecanismos de busca ou por meio dos artigos lidos.

3.4 FERRAMENTAS OBTIDAS COM A REVISÃO SISTEMÁTICA

Depois da leitura completa dos artigos selecionados, quatro ferramentas foram encontradas, são elas: JIVE, JOVE, GamePro e MultiVizArch. Essas ferramentas são detalhadas a seguir.

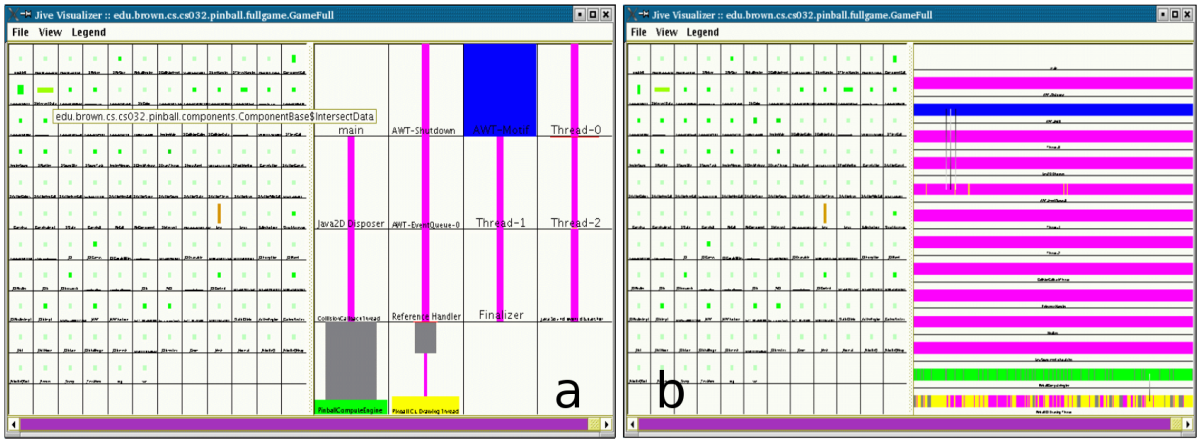
3.4.1 JIVE e JOVE

JIVE (Figura 10) e JOVE (Figura 11) são duas ferramentas que analisam o comportamento de produtos de software desenvolvidos na linguagem Java.

JIVE (REISS; RENIERIS, 2005a; REISS; RENIERIS, 2005b) observa o comportamento e a interação entre *threads*, resumindo as informações em intervalos de 10 milisegundos ou mais, de acordo com o interesse do usuário. Tal ferramenta, para cada classe ou conjunto de classes, coleta o número de chamadas dos métodos, o número de alocações de objetos da classe e o número de sincronizações feitas em objetos da classe. Além disso, também monitora o estado das *threads* (executando, executando sincronizada, esperando, bloqueada, dormindo, fazendo operação de I/O ou morta), a quantidade de tempo gasto em cada estado e a sincronização entre as *trheads*. Essas informações são mostradas em um formulário compacto que destaca classes e *threads* com comportamentos comuns.

A ferramenta JIVE também acompanha o histórico da execução para que o usuário possa revisar ou reproduzir buscando visualizar mais detalhadamente o comportamento.

Figura 10 – Visualização da ferramenta JIVE.



Fonte: Figura adaptada de Reiss e Renieris (2005a).

No lado esquerdo das telas da ferramenta JIVE (Figuras 10a e 10b) são mostradas várias classes e pacotes no nível de detalhes escolhido pelo usuário. Cada caixa contém um retângulo cuja altura indica o número de chamadas de métodos da classe e a largura representa o número de alocações dos métodos, em que a cor representa o número de objetos alocados e a saturação o número de sincronizações.

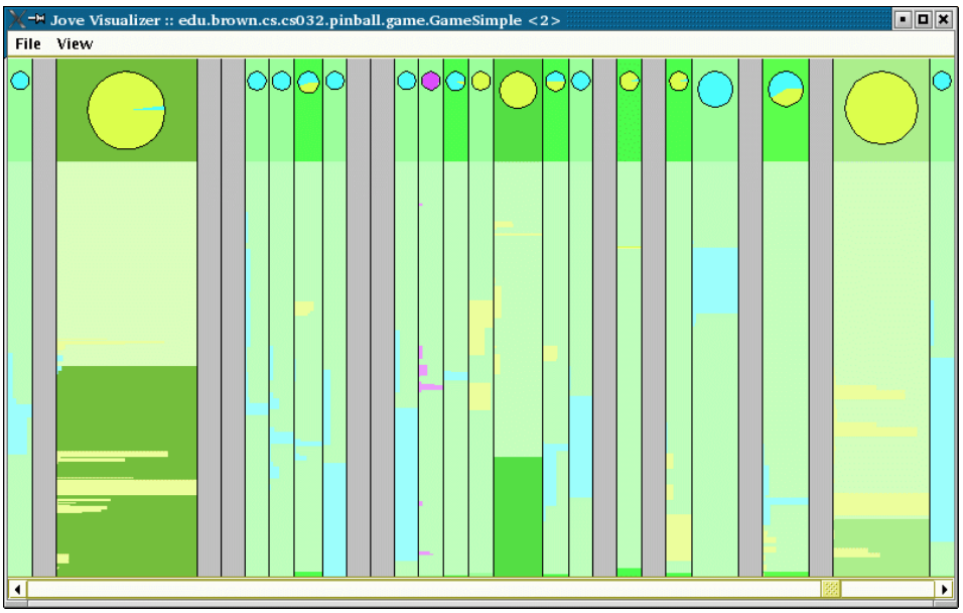
Já no lado direito da Figura 10a, há uma região para cada *thread* na qual uma pilha de retângulos mostra os estados em que cada *thread* se encontra ao longo do tempo. E no lado direito da Figura 10b, é mostrada uma visualização alternativa em que o estado de diferentes *threads* é mostrado em um eixo de tempo com barras verticais indicando sincronizações entre os segmentos (REISS; RENIERIS, 2005a).

Entretanto, apesar da JIVE ser muito útil para análise de alto nível, não fornece informações suficientes sobre problemas específicos, tais como:

- Qual código está sendo executado?
- Qual é o motivo de determinada *thread* estar usando todo o tempo de execução?
- O que cada *thread* está realmente fazendo?

Para responder essas questões, a ferramenta JOVE (REISS; RENIERIS, 2005a; REISS; RENIERIS, 2005b) deve ser utilizada. A ferramenta JOVE obtém informações da execução no nível básico do bloco[1], permitindo visualizar as informações de contexto e globais de cada arquivo de código e todos os blocos de determinado arquivo, além do mapeando dos blocos básicos para as correspondentes linhas de código. Ela possui uma tela configurável que destaca informações incomuns e um mecanismo de histórico que permite navegar pela execução e utiliza a altura, a largura e as cores para mostrar as informações dos arquivos.

Figura 11 – Visualização da ferramenta JOVE.



Fonte: Figura obtida de Reiss e Renieris (2005a).

Na Figura 11, a exibição do contexto é mostrada na parte superior como um gráfico de pizza que mostra a fração do número de instruções do arquivo executadas por cada

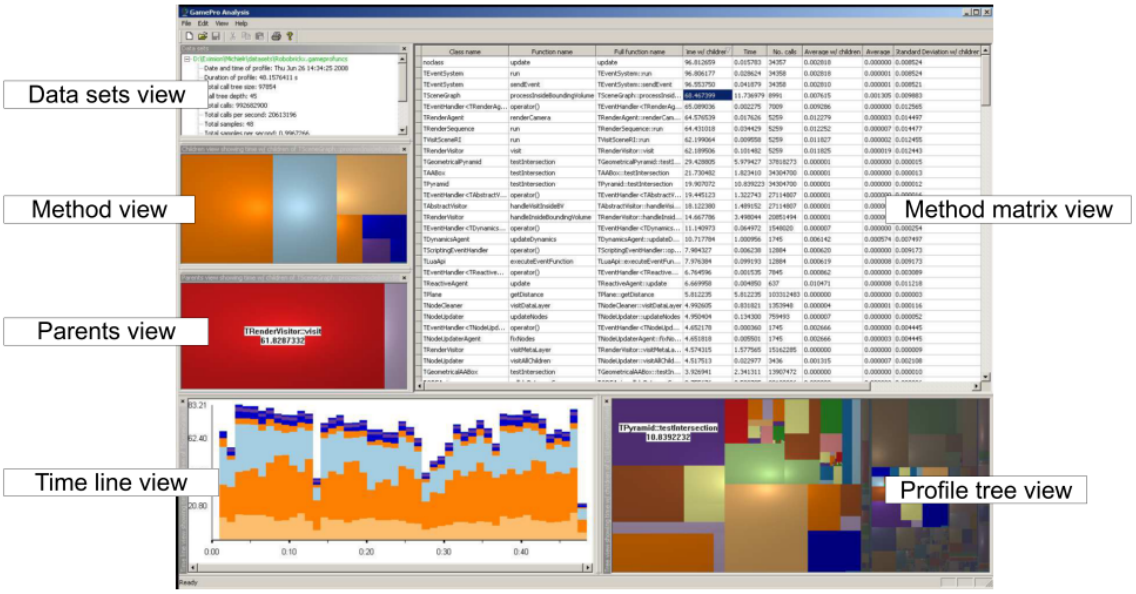
[1] Um bloco básico é um segmento de código linear sem ramificações internas.

thread durante determinado intervalo de tempo. Como é possível visualizar, várias *threads* são mostradas simultaneamente. E a largura de um arquivo é proporcional ao número total de instruções executadas. Por padrão, a cor mostra a proporção que a *thread* executou, a saturação representa o número de segmentos e o brilho corresponde ao número de alocações (REISS; RENIERIS, 2005b).

3.4.2 GamePro

Essa ferramenta é um medidor de desempenho projetado para encontrar a causa de gargalos em games, inspecionando aspectos dinâmicos como: o tempo, o uso e o estouro de memória.

Figura 12 – Tela de análise da ferramenta GamePro.



Fonte: Figura obtida de Roza, Schroders e Wetering (2009).

Conforme é possível visualizar na Figura 12, a tela de análise do GamePro possui 6 opções de visualização, são elas:*Data sets view*, *Method view*, *Parents view*, *Time line view*, *Method matrix view* e *Profile tree view*.

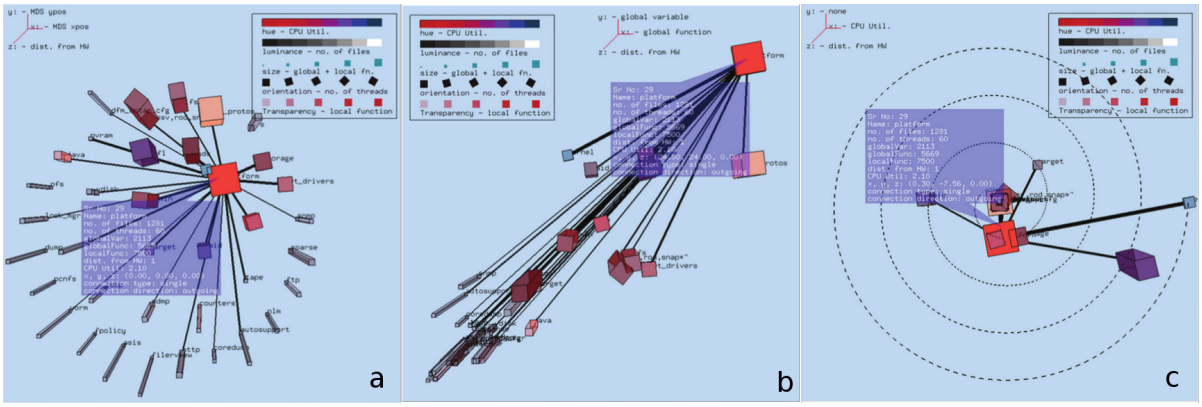
- A visualização *Data sets view* mostra o conjunto de dados que será analisado.
- A visualização *Method view* mostra um conjunto de métodos através de um *treemap* onde o tamanho representa o valor de um atributo selecionado e a cor, um método. Essa visualização pode ser utilizada para encontrar facilmente métodos com o maior valor para determinado atributo.
- A visualização *Parents view* é usada para navegar na árvore de perfil.
- A visualização *Time line view* possibilita analisar a taxa de *frame* durante a execução.

- A visualização *Method matrix view* possibilita explorar atributos numéricos do conjunto de dados.
- A visualização *Profile tree view* mostra a árvore de chamada através de uma *treemap*.

Tal ferramenta funciona da seguinte forma: primeiramente, um conjunto de dados precisa ser selecionado. Depois, um atributo (como duração do método, uso de memória, número de chamada e desvio padrão de duração de chamada) pode ser selecionado para visualização num *treemap*. Em seguida, um intervalo de tempo ou um conjunto de métodos pode ser selecionado para filtrar os dados (ROZA; SCHRODERS; WETERING, 2009).

3.4.3 MultiVizArch

Figura 13 – Visualizações da MultiVizArch: (a) MDS, (b) 2-d *grid* e (c) spiral.



Fonte: Figura adaptada de Sawant e Bali (2008).

Essa ferramenta apresenta uma técnica para visualizar arquitetura de software grande (sistemas operacionais, *middleware* e outras aplicações, por exemplo), utilizando várias representações gráficas, como escalonamento multidimensional, *grid* 2D e layout espiral. Para coletar as informações, algumas ferramentas auxiliares são utilizadas, são elas: BURT, Bugzilla e Understand para C++.

Para alimentar as visualizações, a ferramenta MultiVizArch utiliza a tabela de referência cruzada de símbolos globais que contém o número de referências entre os componentes de software, juntamente com outros atributos, são eles: número de arquivos, número de *threads*, número de funções locais, distância do hardware e utilização da CPU. Essa taxa de utilização foi calculada considerando a simulação das demandas de armazenamento não volátil que um servidor pode receber. Por isso, os usuários dessa ferramenta podem ser engenheiros, arquitetos ou desenvolvedores de software.

Para exibir os dados, essa ferramenta faz uso da posição espacial, cor e textura dos *glyph* (objetos geométricos 3D) para representar as informações do software. Para isso,

são utilizados três gráficos: escalonamento multidimensional (MDS[2]) (Figura 13a), *grid* 2-D (Figura 13b) e uma espiral (Figura 13c) para representar o relacionamento entre os componentes de software.

Na Figura 13a, por exemplo, o tamanho de cada *glyph* é diretamente proporcional a quantidade de código e a distância entre os componentes é inversamente proporcional ao número de referência entre eles. A escala de cores representa a utilização da CPU, em que tons vermelhos correspondem a pouca utilização e tons azuis, muita utilização. A orientação do *glyph* corresponde ao número de *threads* (SAWANT; BALI, 2008).

3.5 TRABALHOS RELACIONADOS

Nesta seção, são apresentadas algumas ferramentas de visualização de software que buscam, cada uma de acordo com o seu objetivo, proporcionar um melhor entendimento do software analisado, visualizando as informações de forma mais simples, fácil de entender e/ou atraente para o usuário.

3.5.1 SHriMP

Desenvolvida em 2001, a ferramenta SHriMP (STOREY; BEST; MICHAND, 2001) mostra as diversas hierarquias presentes num sistema orientado a objetos. Ela permite visualizar as ligações entre Pacotes/Classes e Classes/Classes, analisar o código-fonte ou a documentação produzida em "javadoc".

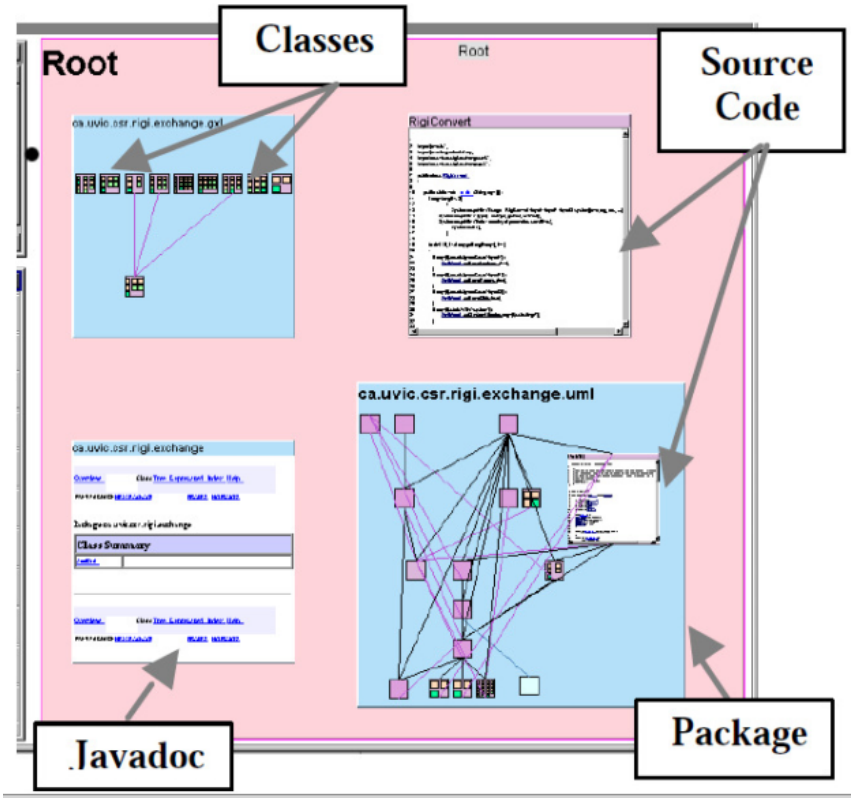
A SHriMP trabalha com 4 visualizações, são elas:

- A visualização por *"Classes"*, em que é exibida a herança entre classes.
- A visualização por *"Source Code"*, em que é mostrado o código-fonte da classe visualizada.
- A visualização por *"Package"*, em que é mostrado a relação das classes e interfaces de um determinado pacote;
- E a visualização por *"Javadoc"*, em que são exibidos comentários do código-fonte.

Essas visualizações são mostradas na Figura 14. A SHriMP também permite a navegação pelo modelo considerando o uso de *hyperlinks*, possibilitando aumentar ou diminuir o nível de abstração da visualização (JULIANO, 2014).

[2] MDS: do inglês *multi-dimensional scaling*

Figura 14 – Visualizações do SHriMP



Fonte: Figura obtida de Storey, Best e Michand (2001) apud Juliano (2014).

3.5.2 CodeCrawler

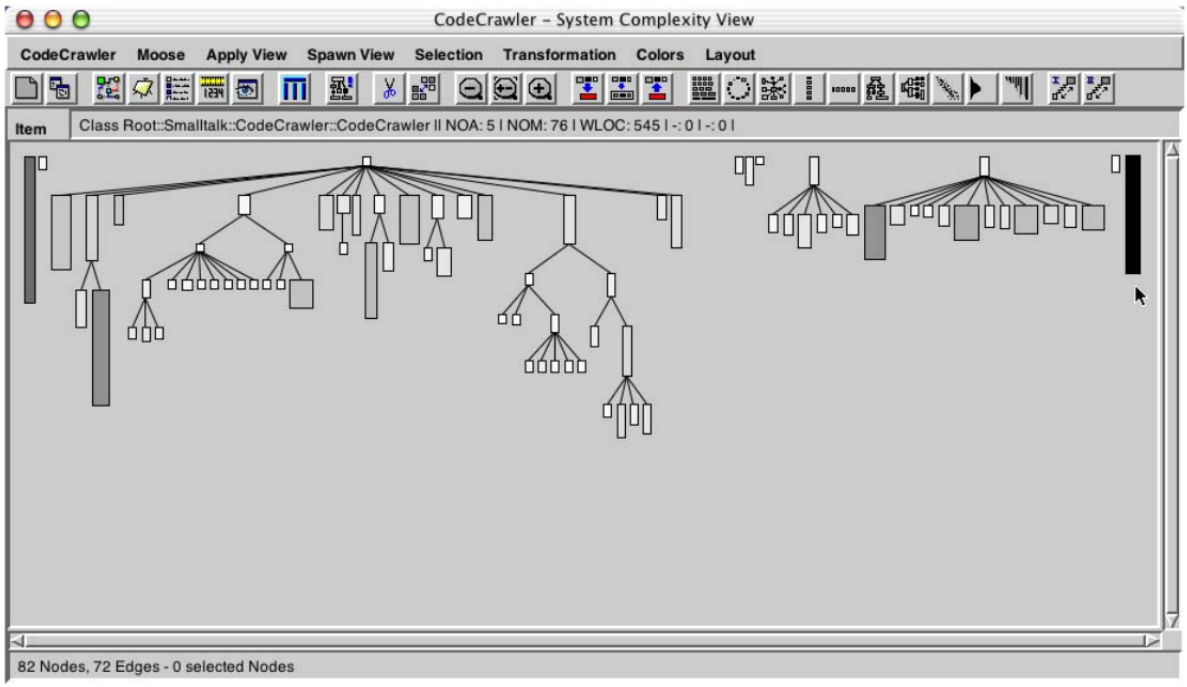
Em 2003, Lanza (2003) utilizou o conceito de visão polimétrica para implementar a ferramenta CodeCrawler, que possibilita a visualização baseada em hierarquia, com informações de métricas de software, conforme ilustrado na Figura 15.

Tal ferramenta é baseada em cinco critérios que direcionam a visualização: posição no eixo X, posição no eixo Y, altura, comprimento e cor dos objetos gráficos. Esses critérios estão associados a métricas e atributos de software, como: o pacote da classe, a quantidade de linhas de código, a quantidade de métodos e a quantidade de atributos das classes (JULIANO, 2014).

Essa ferramenta é independente de linguagem de programação e possui três tipos diferentes de visualizações polimétricas (LANZA, 2004), são elas:

- De granularidade grossa (LANZA; DUCASSE, 2003): é recomendada para visualizar sistemas grandes.
- De granularidade fina (LANZA; DUCASSE, 2001): mostra a estrutura interna e a hierarquia de classe;
- Evolucionária (LANZA, 2001): visualiza a evolução do software.

Figura 15 – CodeCrawler utilizando a visão *Complexidade do Sistema.*

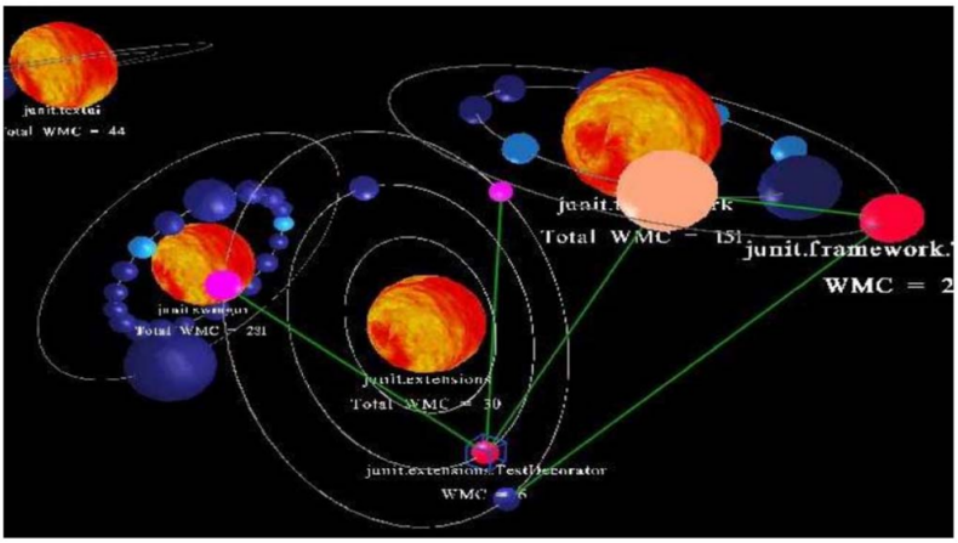


Fonte: Figura obtida de Lanza (2004).

Na Figura 15, é mostrada a visualização da complexidade do sistema, em que a largura corresponde ao número de atributos; a altura representa o número de métodos e a cor faz referência ao número de linhas de código para cada classe.

3.5.3 Solar System

Figura 16 – Visualização utilizando a metáfora do sistema solar.



Fonte: Figura obtida de Caserta e Zendra (2011) apud Juliano (2014).

A ferramenta Solar System utiliza uma metáfora baseada no sistema solar para repre-

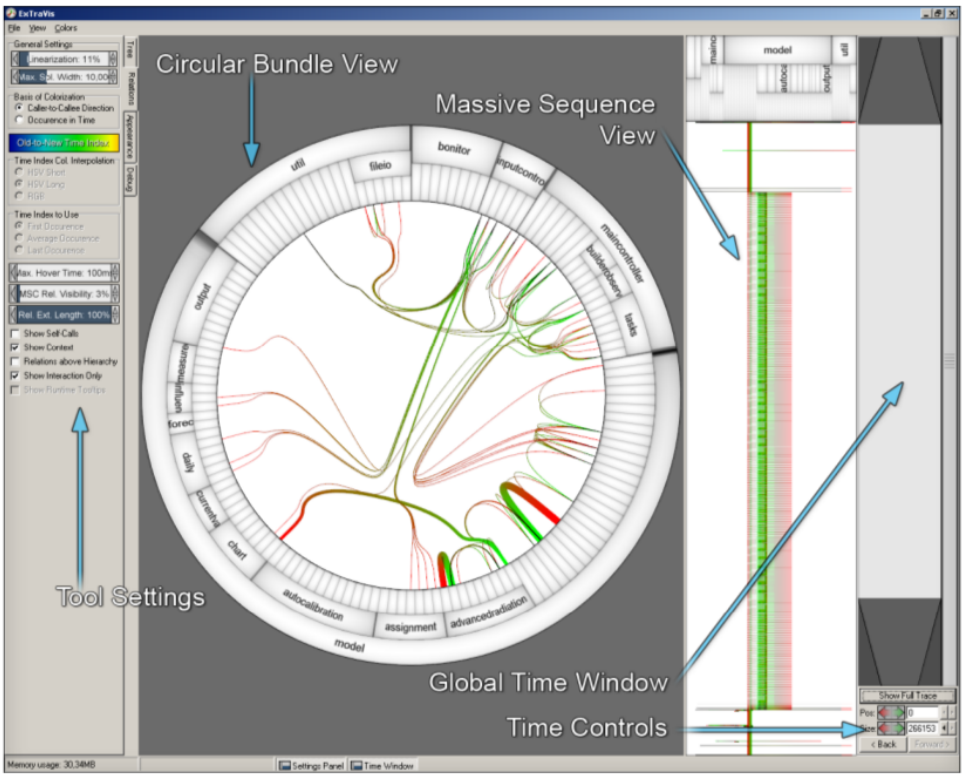
sentar classes e métricas de software desenvolvidos orientados a objetos. Nessa visualização, o software é representado como uma galáxia composta por vários sistemas solares, em que cada sistema solar representa um pacote.

A estrela central é o pacote e os planetas que estão em órbita representam as classes que compõem tal pacote. A profundidade da órbita indica o grau de profundidade da classe, por meio da métrica DIT. A cor representa o tipo do dado (as classes são azuis e as interfaces são vermelhas) e o tamanho do planeta é definido de acordo com a quantidade de linhas de código (LOC) das classes (GRAHAM; YANG; BERRIGAN, 2004; CASERTA; ZENDRA, 2011; JULIANO, 2014). Na Figura 16 é ilustrada a visualização de um software no modelo de sistema solar.

3.5.4 ExtraVis

O ExtraVis (HOLTEN; CORNELISSEN; WIJK, 2007) abrange as áreas de visualização estática e dinâmica. Essa ferramenta mapeia estruturalmente os pacotes, classes e métodos do software e também a troca de mensagens. Seu objetivo é visualizar a execução para dar suporte a compreensão de programas durante as tarefas de manutenção de software.

Figura 17 – Tela do ExtraVis.



Fonte: Figura obtida de Holten, Cornelissen e Wijk (2007).

Para isso, ela apresenta duas *views* sincronizadas, são elas: uma *views* circular, que mostra a decomposição estrutural do sistema e as interações, e uma *views* de sequência

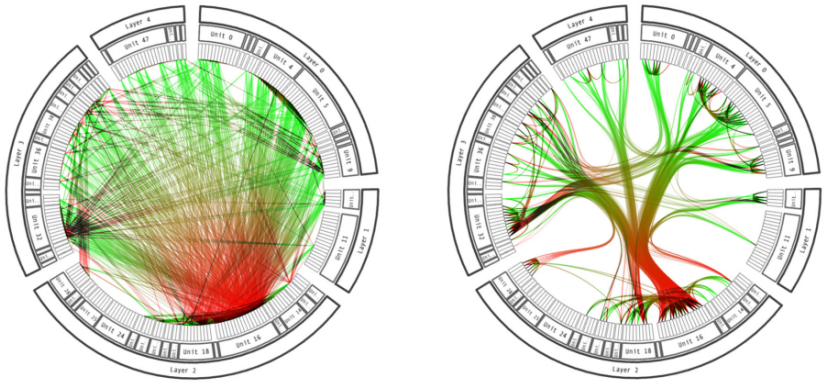
massiva, que permite uma visão geral de chamadas consecutivas entre elementos do sistema em ordem cronológica.

Para melhorar a compreensão da visualização, é utilizado o conceito de *Hierarchical Edge Bundles* (HEB), proposto por Holten (2006). A técnica HEB consiste em unir uma quantidade de arestas que possuem posições próximas, agrupando-as, mas mantendo suas características principais (HOLTEN, 2006).

Na Figura 17 é mostrada a tela principal do ExtraVis e também o rastreamento da execução do JHOTDRAW [3].

Nas Figuras 18a e 18b são mostradas a ausência e o uso da técnica HEB respectivamente. Nessas figuras, a cor das arestas representa o tempo de chamada dos métodos, em que a cor verde representa as chamadas mais antigas e a cor vermelha, as mais recentes. Nessa ferramenta, o tamanho do software analisado interfere na visualização gerada.

Figura 18 – Tipos de visões do ExtraVis. (a) ExtraVis sem HEB. (b) ExtraVis com HEB.



Fonte: Figuras obtidas de Cornelissen et al. (2007).

3.5.5 CodeCity

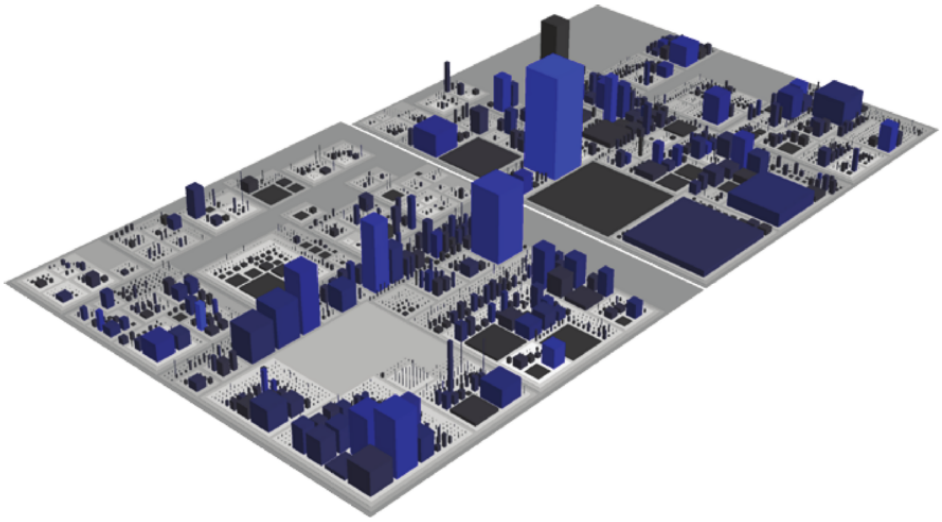
O CodeCity utiliza a metáfora de uma cidade, na qual a estrutura física faz referência a estrutura lógica de um software.

Para exemplificar o funcionamento do CodeCity, na Figura 19, é mostrada a visualização do *Java Development Kit* (JDK), versão 1.5. A visualização gerada é classificada como estática e sua qualidade é inversamente proporcional ao tamanho do software analisado (JULIANO, 2014).

Durante a visualização, o número de atributos e o número de métodos são exibidos. Também existe a possibilidade de destacar as classes como *God* e *Brain*. As classes classificadas como *God* possuem alta complexidade, baixa coesão e uma grande quantidade de acessos em outras classes, tendendo a concentrar grande parte da lógica do sistema. As classes

[3] Mais informações sobre o JHOTDRAW em: http://www.jhotdraw.org/

Figura 19 – Visualização de um sistema pelo CodeCity.



Fonte: Figura obtida de Wettel, Lanza e Robbes (2011) apud Juliano (2014).

classificadas como *Brain* não utilizam muitos dados vindos de outras classes e são mais coesas. As classes também podem ser classificadas como *God+Brain*. Tais classes efetuam várias ações utilizando informações contidas em si e em outras classes e também acumulam uma grande quantidade de regras de negócio do sistema (WETTEL; LANZA; ROBBES, 2011). A classificação das classes facilita a descoberta de classes complexas (OLBRICH; CRUZES; SJØBERG, 2010; WETTEL; LANZA; ROBBES, 2011; JULIANO, 2014).

3.5.6 SArF Map

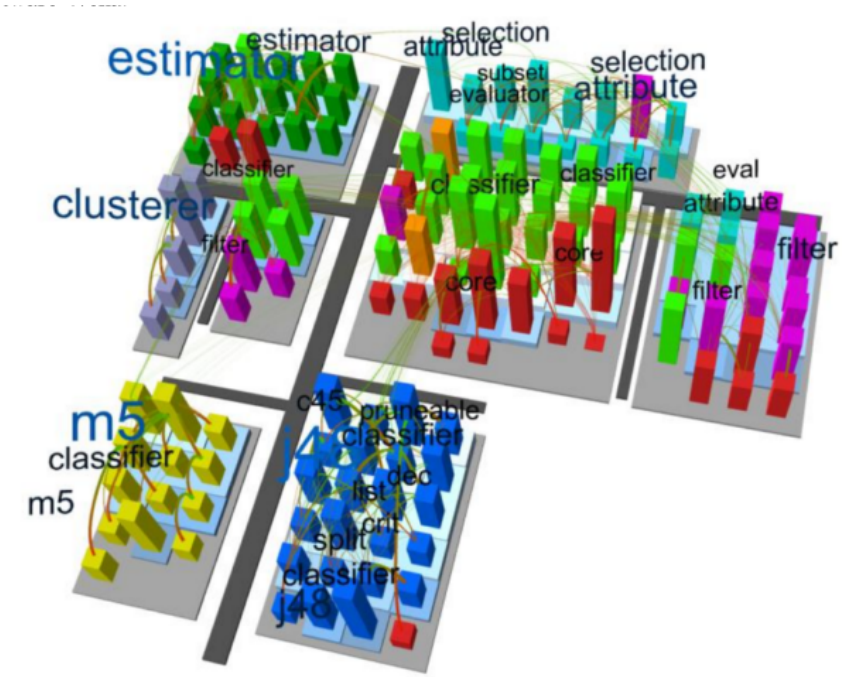
O SArF Map (*Software Architecture Finder*) é uma ferramenta de visualização que busca facilitar o entendimento da arquitetura de software através da visualização de *features*(recursos) e camadas utilizando a metáfora de cidade.

Tal ferramenta utiliza a metáfora de cidades, quadras e prédios. Ela recebe como entrada arquivos .jar ou gráfico de dependência, por isso é independente de linguagem. Nela, cada *feature* é visualizada como um bloco de cidade, as classes na *feature* são mostradas como edifícios, a relevância entre *features* são representadas utilizando ruas e distância e as camadas, como declives.

O SArF visualiza as seguintes propriedades do software: pacote, tamanho (LOC, métodos), tipos de arquivo, utilização, falhas, nível da camada, tipo de ligação entre os componentes, o nível de manutenibilidade e se a classe foi mudada recentemente. Porém, a taxa de utilização, a quantidade de faltas e a manutenibilidade são estimadas.

Vale ressaltar que no SArF as cidades representam características e no CodeCity representam pacotes, essa é a principal diferença entre as ferramentas (KOBAYASHI et al., 2013).

Figura 20 – Visualização do software Weka pela ferramenta SArF.



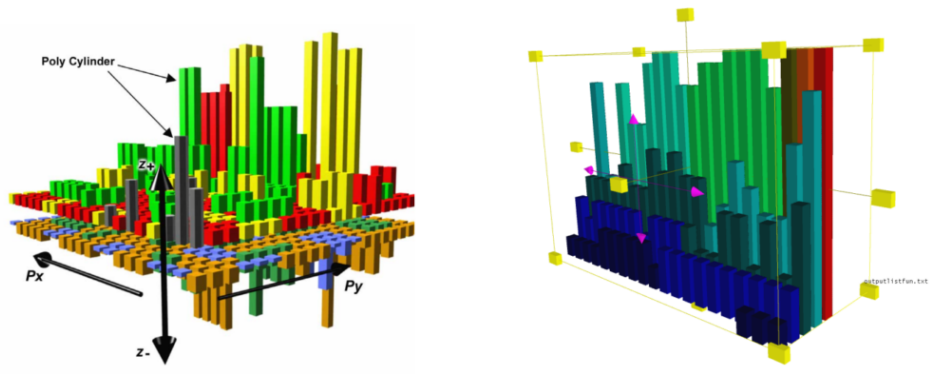
Fonte: Figura obtida de Kobayashi et al. (2013).

Na Figura 20, é exibida a visualização gerada pelo SArF aplicado ao software Weka.

3.5.7 sv3D

O sv3D (*Source Viewer 3D*) é um *framework* para visualização de software que busca estender a metáfora de visualização do SeeSoft (EICK; STEFFEN; SUMNER, 1992). Tal ferramenta tem o objetivo de incrementar a experiência do usuário no uso do sistema, aprimorando a facilidade de interação. Para isso, utiliza uma visão 3D em vez da visão 2D do SeeSoft. Porém busca preservar todas as qualidades da representação de pixel do SeeSoft.

Figura 21 – Visualização de código-fonte pelo sv3D.



(a) (b)

Fonte: Figuras obtidas de Marcus, Feng e Maletic (2003a) apud Juliano (2014).

No sv3D, o usuário pode escolher mapear a função para cilindro ou pixel. E também pode mapear as classes como cilindros. Dessa forma, ele pode gerar múltiplas *views* do software analisado em diferentes níveis de abstração.

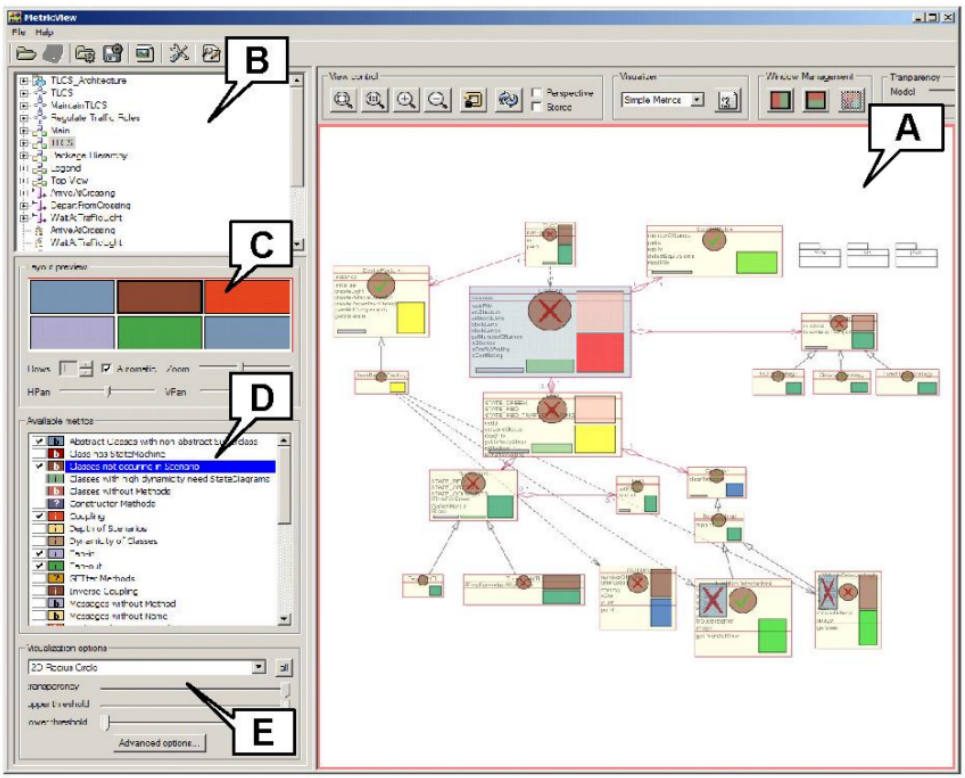
Em relação a interação com usuário, essa ferramenta permite rotacionar, ampliar ou reduzir a visualização (MARCUS; FENG; MALETIC, 2003a).

Na Figura 21a é mostrada a representação policilíndrica e na Figura 21b é mostrada a representação de funções em que a cor indica a quantidade de chamadas para aquela função e a altura representa o tempo de execução da função. Nessas figuras, cada arquivo é representado por um contêiner, que possui vários cilindros. Cada cilindro representa uma linha de código. E as coordenadas *px* e *py* de cada cilindro são determinadas pela posição da linha do código fonte no arquivo.

O sv3D pode ser usado para tarefas como: localização de faltas, visualização de traces de execução, navegação pelo código fonte, análise de impacto, evolução e complexidade, além de outras tarefas (MARCUS; FENG; MALETIC, 2003b).

3.5.8 MetricView

Figura 22 – Tela da ferramenta MetricView.



Fonte: Figura obtida de Termeer et al. (2005).

MetricView é uma ferramenta de visualização que combina o tradicional diagrama UML com visualização de métricas de software (TERMEER et al., 2005).

Essa ferramenta pode visualizar diagramas de classes, sequências, estados, casos de uso e colaboração importados de arquivos XMI (*XML Metadata Interchange*) e utiliza mais de 40 métricas fornecidas por ferramenta de análise de arquitetura de software (LANGE, 2003).

Na Figura 22, é mostrada a tela da ferramenta MetricView, em que é possível visualizar o diagrama de classe combinado com seis métricas de classe (A), o navegador de diagramas (B), a lista de métricas disponíveis (C), a opção para seleção das métricas desejadas (D) e o ajuste da visualização das métricas (E).

A ferramenta visualiza informações sobre métricas utilizando: ícones, retângulos 2D, barras 2D, círculos 2D, gráficos de pizza 2D, barras 3D e cilindros 3D. Ela também permite que vários parâmetros das métricas, tais como: ícones, símbolos da caixa de seleção, cor, entre outras; possam ser ajustados através dos componentes da interface gráfica (E) (TERMEER et al., 2005).

3.6 COMPARAÇÃO ENTRE AS FERRAMENTAS

Existe uma variedade de ferramentas de visualização de software que possuem os mais diversos objetivos, tais como: visualizar a evolução do software (OGAWA; MA, 2009; CAUDWELL, 2010), quando determinado código foi inserido no projeto (EICK; STEFFEN; SUMNER, 1992), a taxa de sucesso da execução dos casos de teste (JONES; HARROLD; STASKO, 2002) ou ainda características da arquitetura, tais como: organização, relacionamentos e métricas (STOREY; BEST; MICHAND, 2001; LANZA, 2003; MARCUS; FENG; MALETIC, 2003a; GRAHAM; YANG; BERRIGAN, 2004; CASERTA; ZENDRA, 2011; LANZA, 1999; WETTEL; LANZA; ROBBES, 2011; HOLTEN, 2006; LANZA et al., 2013).

Comparando a ferramenta VisRQ com algumas das ferramentas apresentadas anteriormente, é possível notar algumas semelhanças. Como exemplo, é possível citar:

- A visualização das relações entre Pacotes/Classes de maneira parecida com a ferramenta SHriMP, CodeCity e Solar System.
- A possibilidade de mais de uma forma de visualização, como ocorre com o SHriMP, JIVE, GamePro e MultiVizArch.
- A possibilidade de visualizar informações do software sobre diagramas UML, como ocorre no software MetricView.
- E o uso de cores para destacar/diferenciar os elementos ou visualizar algumas informações.

Como possuem diversas finalidades e podem implementar várias técnicas de visualização de dados/informações, compará-las não é uma tarefa trivial, já que são inúmeras as

possibilidades de uso uma vez que há vários tipos de sistemas e arquiteturas. Por exemplo, os produtos de software SArF Map e CodeCity, embora utilizem a mesma metáfora (de cidade), visualizam informações diferentes, uma vez que no SArF as cidades representam características e no CodeCity representam pacotes.

Vale ressaltar também que alguns trabalhos mostram informações referentes aos atributos de certos RNFs, apesar de não deixar explícita a visualização de requisitos de qualidade, uma vez que o foco é a visualização de métricas de software. Como exemplo, é possível citar as ferramentas SHriMP, CodeCrawler, Solar System, CodeCity e SArF Map, que apresentam informações relacionadas à manutenibilidade do software.

Baseando-se nos textos e imagens dos trabalhos, as principais informações visualizadas e sobre os recursos visuais utilizados foram reunidas nas Tabelas 7 e 8 respectivamente.

Tabela 7 – Comparação entre as ferramentas.

Nome	**RNF**	**Objetivo**	**Visualização**	**Dados de Entrada**
SHriMP (STOREY; BEST; MICHAND, 2001)	X	Visualizar hierarquias em sistema OO	*Javadoc*, código fonte, classes e pacotes	JAVA
CodeCrawler (LANZA, 2003)	X	Visualizar métricas, a evolução do software, classes e hierarquia	Retângulos de diferentes cores e tamanhos conectados por linhas	Modelo FAMIX
Solar System (GRAHAM; YANG; BERRIGAN, 2004)	X	Visualizar, pacotes, classes e métricas de softwares OO	Esferas com diferentes cores e tamanhos	JAVA
ExtraVis (HOLTEN; CORNELISSEN; WIJK, 2007)		Visualizar os pacotes, as classes e os métodos e a troca de mensagens	*Views* circular e *views* de sequência	RSF (RigiStandard-Format)
CodeCity (WETTEL; LANZA, 2007)	X	Visualizar a estrutura lógica de um *software*	Metáfora de cidade	Modelo FAMIX
SArF Map (KOBAYASHI et al., 2013)	X	Busca facilitar o entendimento da arquitetura de *software*	Metáfora de cidade	Banco de dados do Weka
sv3D (MARCUS; FENG; MALETIC, 2003a)	X	Facilitar o entendimento do *software*	Container de cilindros e conjunto de *pixel*	XML
MetricView (TERMEER et al., 2005)		Visualizar diagrama UML com métricas de *software*	Diagramas UML, ícones e cores	XMI, arquivo com as métricas
JIVE (REISS; RENIERIS, 2005a; REISS; RENIERIS, 2005b)	X	Visualizar o comportamento e a interação entre *thread*	Tabela e barras	JAVA
JOVE (REISS; RENIERIS, 2005a; REISS; RENIERIS, 2005b)	X	Visualizar informações de contexto e globais de cada arquivo de código	Gráfico de pizza e barras variando o tamanho, a largura e a cor	JAVA
MultiVizArch (SAWANT; BALI, 2008)	X	Visualizar informações da arquitetura de *software*	Objetos 3D simples variando a posição, cor e textura	Saída de algumas ferramentas de análise
GamePro (ROZA; SCHRODERS; WETERING, 2009)	X	Visualizar o desempenho de *games*	*Treemap*, gráfico de barra, lista e tabela	*Trace*[4]
VizRQ	X	*Visualizar informações sobre os RNFs eficiência e confiabilidade*	*Gráficos de pontos, lista, objetos 2D simples, cores e ícones*	Imagens, *logs* no formato JSON

Fonte: Elaborada pelo Autor.

Tabela 8 – Comparação entre os recursos visuais utilizados.

Recursos	**SHriMP**	**CodeCrawler**	**Solar System**	**ExtraVis**	**CodeCity**	**SArF Map**	**sv3D**	**MetricView**	**JIVE**	**JOVE**	**MultiVizArch**	**GamePro**	**VisRQ**
Cor	X	X	X	X	X	X	X	X	X	X	X	X	*X*
Altura		X	X		X	X	X		X				
Largura		X	X		X	X			X	X			
Posição		X	X		X	X					X		*X*
Textura											X		
Objetos 2D		X					X						*X*
Objetos 3D			X		X	X	X				X		
Metáfora de cidade					X	X							
Metáfora do universo			X										
Diagramas UML								X					*X*
Gráfico de barras												X	
Gráfico de pizza										X			
Gráfico de pontos													*X*
Tabela									X			X	*X*
Barras									X				
Lista												X	*X*
Treemap												X	
Ícones								X					*X*
Código fonte	X												
Javadoc	X												
View circular				X									
View de sequência				X									

Fonte: Elaborada pelo Autor.

Diante disso, na Tabela 7, é apresentado um resumo sobre as ferramentas abordadas nessa seção. Nessa tabela, são mostradas algumas informações, tais como: o nome da ferramenta, os autores, o objetivo, a forma de visualização das afirmações e os dados de entrada. Observando essa tabela, é possível comparar os objetivos, as visualizações e os dados de entrada das ferramentas.

Na Tabela 8, é mostrada uma comparação entre os recursos utilizados pelas ferramentas de visualização para exibir informações. Dessa forma, é possível visualizar as ferramentas que fazem uso de um determinado recurso. Nota-se, por exemplo, que o uso das cores nas visualizações das ferramentas abordadas serve para transmitir alguma informação.

Ao analisar a Tabela 9, é possível notar que a maioria das informações visualizadas pelas ferramentas são diferentes, com exceção do *número de alocações* entre as ferramentas JIVE e JOVE e o *tempo de execução* entre as ferramentas sv3D, GamePro e VisRQ. Além disso, as métricas visualizadas pelas ferramentas não foram diferenciadas, pois tais informações

Tabela 9 – Comparação entre as principais informações visualizadas.

Informações	**SHriMP**	**CodeCrawler**	**SolarSystem**	**ExtraVis**	**CodeCity**	**SArF Map**	**sv3D**	**MetricView**	**JIVE**	**JOVE**	**MultiVizArch**	**GamePro**	**VisRQ**
Estado das *threads*									X				
Tempo em cada estado									X				
Sincronização entre *trheads*									X				
Nº de sincronizações									X				
Nº de alocações									X	X			
Nº de segmentos										X			
Nº de instruções executadas										X			
Proporção executada										X			
Nº de referências entre componentes											X		
Nº de arquivos											X		
Nº de *threads*											X		
Nº de funções locais											X		
Distância do *hardware*											X		
Porcentagem de uso da CPU											X		
Taxa de *frame*												X	
Árvore de chamada												X	
Uso e estouro de memória												X	
Tempo de execução							X					X	*X*
Tempo de uso da CPU													*X*
Relacionamentos entre componentes	X		X	X		X		X					*X*
Nº de exceções													*X*
Classificação por tempo de execução													*X*
Classificação por número de exceção													*X*
Informações de execução real							NE						*X*
Informações de predição ou testes						X	NE		X	X	X	X	*X*
Métricas de *software*	X	X	X	X	X	X		X			X		

Não especificado no trabalho (NE).
Fonte: Elaborada pelo Autor.

não estão relacionadas com o escopo do trabalho.

Já na Tabela 10, foi realizada uma comparação entre as ferramentas para mostrar os principais pontos fortes da ferramenta VisRQ. Para facilitar tal comparação, os recursos listados na Tabela 10 foram mapeados para as seguintes questões:

1. Visualização de requisitos de qualidade - A ferramenta exibe explicitamente alguns dos requisitos de qualidade definidos na norma ISO/IEC 9126 ou 25010?

2. Visualização integrada com a arquitetura - A visualização mostrada está contextualizada com a arquitetura do software?

3. Detalhamento recursivo dos elementos arquiteturais - A ferramenta permite, por exemplo, especificar/detalhar os subpacotes de um componente, as classes de um

pacote e os métodos de uma classe?

4. Visualização escalável para software de grande porte - A ferramenta consegue representar um software grande mantendo o entendimento da visualização?

5. Visualização de diferentes granularidade - A ferramenta permite visualizar da mesma forma desde métodos até componentes?

6. Permite múltiplas visões arquiteturais - A ferramenta permite visualizar várias visões arquiteturais?

7. Integração entre os diagramas - Permite a navegação entre os diagramas relacionados com o mesmo componente?

8. Visualiza dados sobre uso real - Os dados visualizados são do ambiente de produção?

9. Permite filtrar dados por período - Possibilita filtrar, visualizar e comparar dados de execuções distintas?

Tabela 10 – Pontos fortes da ferramenta VisRQ e comparação com as demais ferramentas.

Informações/Recursos	**SHriMP**	**CodeCrawler**	**SolarSystem**	**ExtraVis**	**CodeCity**	**SArF Map**	**sv3D**	**MetricView**	**JIVE**	**JOVE**	**MultiVizArch**	**GamePro**	**VisRQ**
1. Visualização de requisitos de qualidade	X	X	X		X	X	X		X	X	X	X	*X*
2. Visualização centrada na arquitetura								X					*X*
3. Detalhamento recursivo dos elementos arquiteturais													*X*
4. Visualização escalável para softwares de grande porte		X		X	X	X	X						*X*
5. Visualização de diferentes granularidades													*X*
6. Permite múltiplas visões arquiteturais								X					*X*
7. Integração entre os diagramas													*X*
8. Visualiza dados sobre uso real													*X*
9. Permite filtrar dados por período													*X*

Fonte: Elaborada pelo Autor.

Conforme é possível notar na Tabela 10, as demais ferramentas não apresentam certas características presentes na ferramenta VisRQ. Como exemplo dessas características, pode-se citar o “Detalhamento recursivos dos elementos arquiteturais”, a “Visualização de diferentes granularidades”, a “Integração entre os diagramas”, a “Visualização de dados sobre uso real” e a “Comparação de dados de períodos diferentes”.

Em relação as características “Visualização centrada na arquitetura” e “Permite múltiplas visões arquiteturais” apenas as ferramentas VisRQ e MetricViews as possuem. Porém, na ferramenta MetricViews essas características estão relacionadas com a utilização de diagramas UML. Já na VisRQ, o usuário pode importar, além de diagramas UML, qualquer

diagrama que ache conveniente. Por isso, a VisRQ possibilita a visualização de mais visões arquiteturais.

Assim, o diferencial da ferramenta VisRQ em relação as demais ferramentas apresentadas é fato de exibir informações sobre monitoramento de requisitos não-funcionais (eficiência e confiabilidade) do uso real do software, permitindo analisar e comparar as execuções entre diferentes períodos além da contextualização das informações com a arquitetura do software, do detalhamento recursivos dos elementos arquiteturais, da visualização de diferentes granularidades e da integração entre os diagramas.

3.7 ANÁLISE GERAL

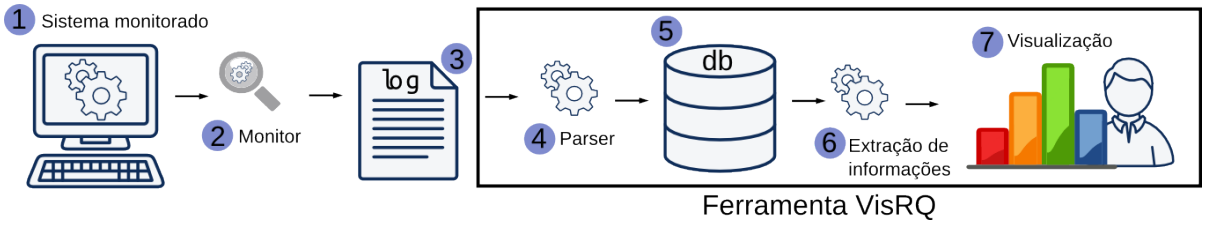
Apesar de ter trabalhos com foco na visualização de dados relacionados à arquitetura de software, nenhum dos trabalhos identificados possui preocupação em visualizar dados de monitoramento relacionados a requisitos de qualidade de forma contextualizada com a arquitetura de software. Outro diferencial do trabalho proposto na presente dissertação é a preocupação com os dados históricos da monitoração, de forma a possibilitar a identificação de potenciais cenários degenerativos, o que ajudaria os desenvolvedores, arquitetos e engenheiros de software a prever potenciais falhas na satisfação dos requisitos de qualidade.

4 SOLUÇÃO PROPOSTA

Neste capítulo, são apresentadas a abordagem proposta e a ferramenta que a suporta (ferramenta VisRQ). Tal ferramenta é voltada à visualização dos requisitos de qualidade do sistema de modo contextualizado com a arquitetura de software. A seguir, são mostradas algumas telas da aplicação juntamente com suas principais características.

4.1 ABORDAGEM PROPOSTA - VISÃO GERAL

Figura 23 – Processo desta pesquisa.



Fonte: Elaborada pelo Autor.

A abordagem proposta nesta dissertação, tem o objetivo de auxiliar desenvolvedores, arquitetos e engenheiros de software a visualizar os dados do monitoramento dos requisitos Confiabilidade e Desempenho [1], contextualizando-os com a arquitetura de software. Tal abordagem foi embutida na ferramenta VisRQ e permite relacionar os diagramas arquiteturais com os *logs* de monitoramento. Para isso, ela segue o processo representado na Figura 23. Com base nessa figura, apenas os pontos 4, 5, 6 e 7 fazem parte do escopo desta pesquisa. Os pontos 2 e 3 correspondem ao uso da ferramenta QuAM *Framework* (LIMA, 2016) e o ponto 1 refere-se ao software alvo da monitoração em execução.

O processo de monitoramento e visualização, representado na Figura 23, pode ser explicado da seguinte forma:

- Inicialmente, o sistema a ser monitorado (1) precisa ser anotado de acordo com as especificações do *framework* de monitoramento (LIMA, 2016) (2). O *framework* observará o funcionamento do sistema, gerando e armazenando seus eventos em arquivos de *logs* (3).
- Em seguida, depois da execução do *parser* para tratamento dos dados (4), os **logs** são inseridos num banco de dados (5) para facilitar o trabalho da fase de extração de informações (6).

[1] Os requisitos não-funcionais Confiabilidade e Desempenho foram escolhidos pelo fato da ferramenta monitora dar suporte ao monitoramento desses requisitos e pela limitação de tempo para a realização e conclusão do trabalho, uma vez que a implementação de mais requisitos ultrapassaria o prazo determinado.

- Na etapa de extração de informações (6), os **logs** armazenados são analisados e organizados buscando gerar um conjunto de representações gráficas (7). Tais representações devem ser contextualizadas com a arquitetura de software na forma de visões arquiteturais.

4.2 REQUISITOS PARA IMPLEMENTAR ABORDAGEM PROPOSTA

Para implementar a abordagem desenvolvida neste trabalho, a ferramenta VisRQ deve possuir os seguintes requisitos funcionais:

1. Possibilitar a importação dos *logs* gerados pela ferramenta de monitoramento para o banco de dados da ferramenta de visualização;
2. Possibilitar a importação e utilização de diagramas da arquitetura do software desenvolvidos em outras ferramentas (como Posseidon, DIA, ArgoURL, Astha, etc.);
3. Possibilitar a associação dos elementos arquiteturais com os objetos relacionados aos dados de monitoramento importados;
4. Possibilitar a visualização, de forma mais eficiente, das informações contidas nos *logs*;
5. Possibilitar interagir com a visualização gerada, exibindo visões arquiteturais com diferentes níveis de abstração (omitindo ou adicionando detalhes dos elementos arquiteturais);
6. Possibilitar a avaliação de dados históricos, de acordo com intervalos de tempo determinados pelo usuário.

Esses itens são fundamentais para atingir os objetivos definidos.

4.3 FERRAMENTA VISRQ

Visando implantar e validar a abordagem proposta, foi desenvolvida a ferramenta VisRQ. Tal ferramenta foi planejada com foco nos seguintes requisitos de qualidade:

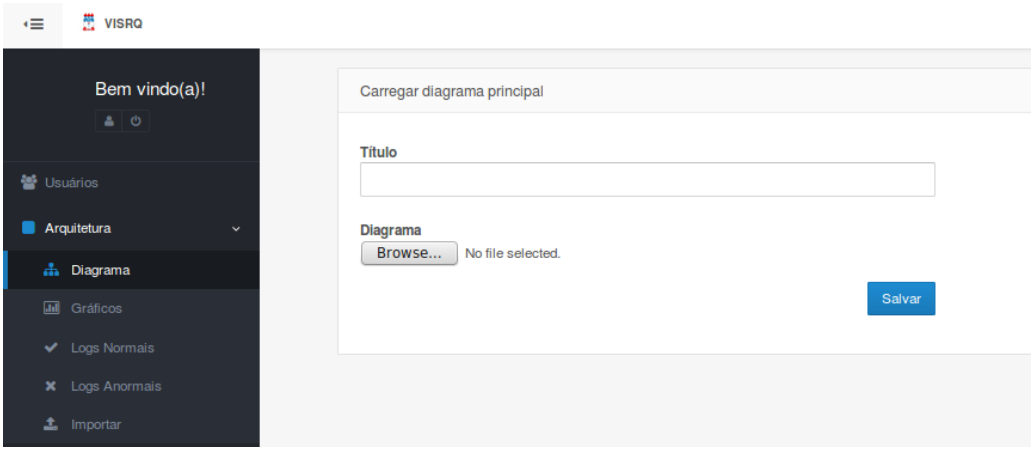
1. Ser de fácil uso, possibilitando a aprendizagem em torno de 10 minutos;
2. Mostrar os resultados de forma mais eficiente, automatizando cálculos, utilizando gráficos e cores para representar as informações.

A seguir, são apresentadas as principais telas da aplicação, partindo da configuração (importação de diagramas e *logs*, criação dos grupos) até a visualização das informações.

4.3.1 Carregar Diagrama Principal da Arquitetura

Ao clicar no link “Diagrama”, caso o usuário não tenha carregado nenhum diagrama ainda, será exibida a tela para fazer *upload* do respectivo arquivo (Figura 24). Nessa tela, o usuário preencherá os campos: *Título*, que corresponde ao texto que ficará acima da imagem da arquitetura, e *Diagrama*, que corresponde ao arquivo do diagrama da arquitetura do sistema.

Figura 24 – Tela Carregar Diagrama Principal



Fonte: Elaborada pelo autor.

Ao relacionar um diagrama com outros mais detalhados, o usuário cria, de forma transparente, uma hierarquia entre tais diagramas. Por exemplo, o primeiro diagrama (diagrama principal) será mais abstrato, uma vez que mostra os subpacotes de um software. Já outro diagrama pode detalhar melhor um desses subpacotes, mostrando as classes ou outros subpacotes, reduzindo o nível de abstração em relação ao primeiro diagrama. Dessa forma, o nível de abstração será determinado pelo usuário através da distribuição hierárquica dos diagramas, podendo partir de diagramas que representem uma abstração maior (pacotes ou módulos, por exemplo) até diagramas com um nível menor de abstração, que podem representar classes ou métodos.

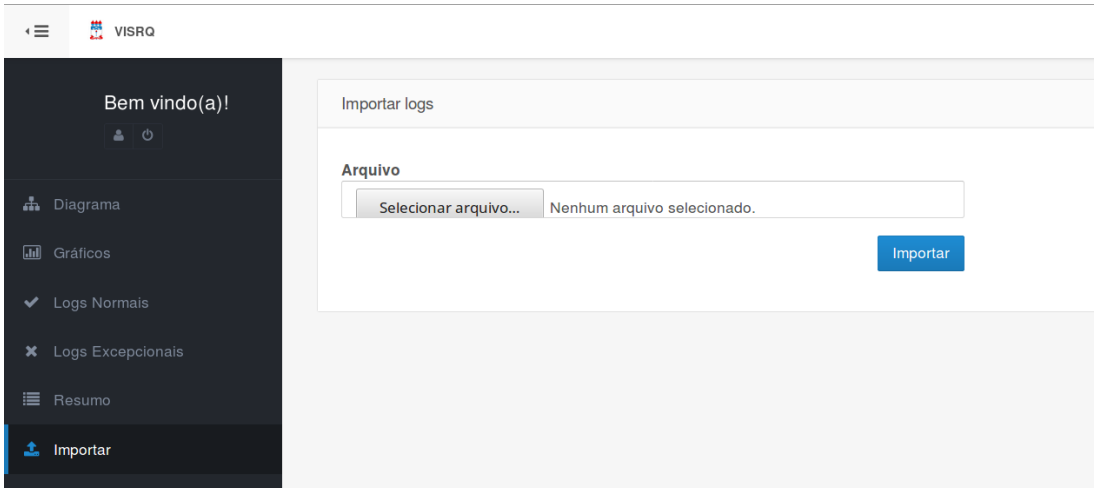
Vale salientar que o usuário poderá importar vários diagramas da arquitetura, fornecendo, por exemplo, detalhes internos de elementos arquiteturais de forma recursiva, ou até mesmo diferentes visões arquiteturais em um mesmo nível de abstração.

Na barra lateral esquerda das telas, conforme é possível visualizar na Figura 24, o usuário tem acesso aos *links* das demais telas da ferramenta. Entre esses *links* estão: “Diagrama”; “Gráficos”; “*Logs* Normais”, “*Logs* Excepcionais”, “Editar”, “Importar” (para importar os *logs*) e “Resumo”. Os *links* “Editar” e “Resumo”, dependendo do contexto, podem ou não ser exibidos. Para o usuário que tiver privilégio, também será exibido o *link* “Usuários”. Ao clicar nesse *link*, haverá o redirecionamento para a “Tela de Usuários”, na qual é possível visualizar todos os usuários cadastrados na ferramenta e também adicionar novos usuários.

4.3.2 Importar *Logs*

Na "Tela Importar *Logs*", apresentada na Figura 25, o usuário deve pesquisar e escolher o arquivo de *log* que deseja importar e, em seguida, acionar o botão "Importar". A aplicação vai identificar o tipo de *logs* presente no arquivo e irá armazenar no banco de dados para posterior consulta e relacionamento com as visões arquiteturais.

Figura 25 – Tela Importar *Logs*



Fonte: Elaborada pelo autor.

4.3.3 Edição

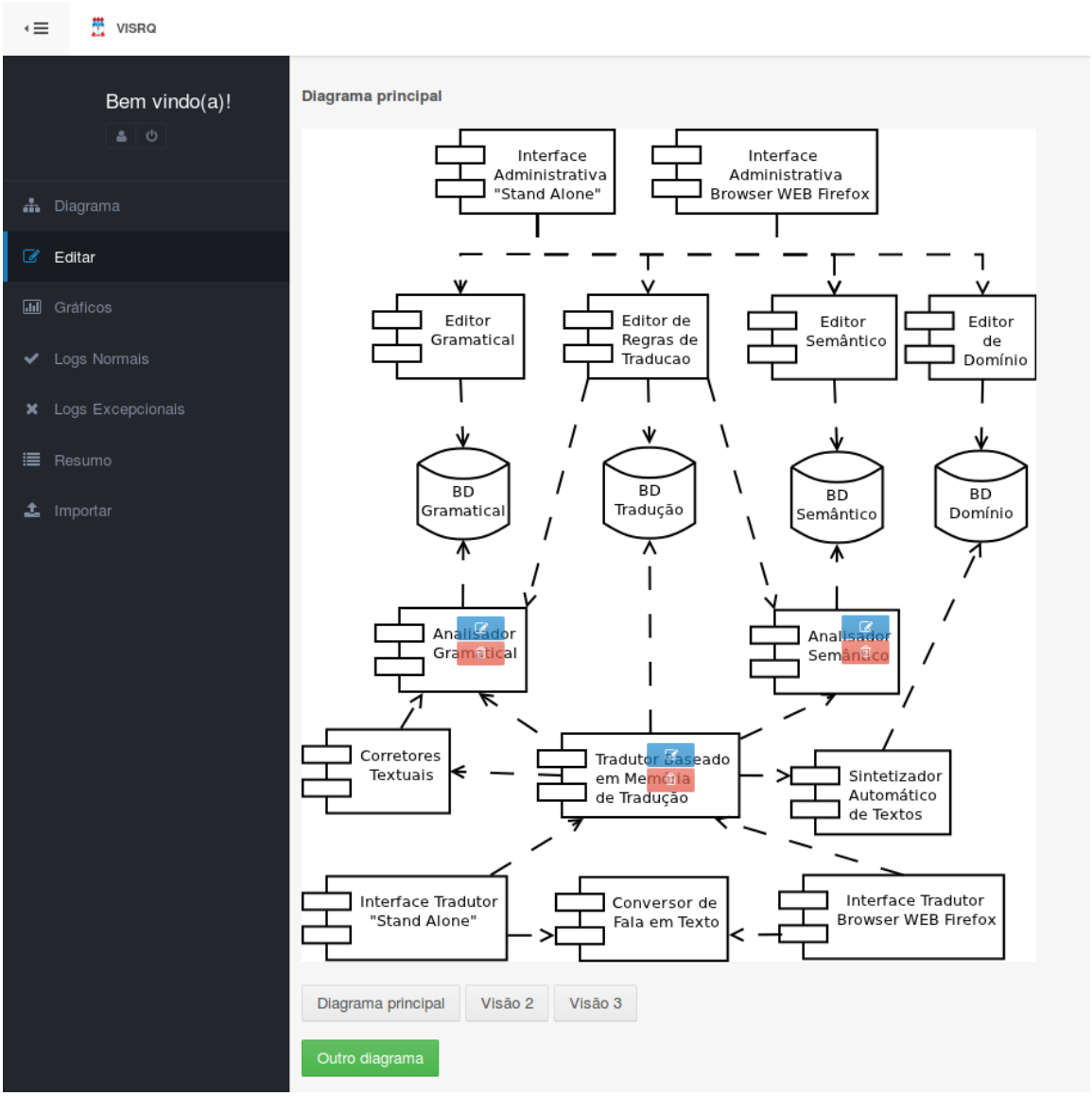
Na "Tela de Edição", o usuário pode criar ou editar um grupo [2] e visualizar onde ficarão posicionadas as informações dos *log* importados (Figura 26). A função do grupo é relacionar alguns *logs* com um determinado componente de software. Um grupo pode representar um método, uma classe, um pacote ou um conjunto de pacotes e tal grupo exibirá as informações contidas nos *logs* através do agrupamento dos dados relacionados.

Conforme mostrado na Figura 27, um grupo deve possuir alguns atributos. São eles:

- Título do grupo - Serve para diferenciá-lo dos demais;
- Pacote do componente - Serve para associar os *logs* com um componente;
- Diagrama detalhado dos componentes - Serve para carregar uma imagem mais detalhada sobre um pacote, sendo opcional. Vale salientar que é possível adicionar diferentes visões arquiteturais (por exemplo, mais de um diagrama com representações visuais diferentes para o mesmo pacote).
- X e Y - São as coordenadas para posicionar a representação do grupo próximo do seu respectivo componente. Para cada grupo, deve ser indicado um par de coordenadas.

[2] Representação gráfica que corresponde a um elemento ou a um conjunto de elementos da arquitetura.

Figura 26 – Tela de Edição - Diagrama com grupos



Fonte: Elaborada pelo autor.

As coordenadas podem ser obtidas clicando na imagem da arquitetura exibida pela ferramenta.

- Intervalo de alerta para cada RNF - “Seção Requisitos”, no lado direito da figura. Essa parte da tela é composta pelos campos:
  - Nome do requisito;
  - Escala - Campo *select* com os valores de escala que será multiplicado pelo valor de P1 e P2 para definir o intervalo de alerta;
  - Barra de seleção - Utilizada para definir os valores dos pontos *P1* e *P2*, indo de 0 a 100.

A seção “Requisitos” é utilizada para definir o intervalo de alerta, que será marcado na visualização do grupo com a cor amarela. Caso a métrica relacionado ao atributo seja menor que o primeiro valor do intervalo (*P1*), o ícone do atributo na visualização será destacado com a cor verde. Caso a métrica do atributo seja maior que o último ponto (*P2*) do intervalo de alerta, o ícone do atributo na visualização será destacado com a cor vermelha. O resultado da definição desses valores pode ser visto na Figura 29, em que o primeiro grupo está destacado com as cores verde e vermelha, o segundo com as cores amarela e verde e o terceiro, vermelha e vermelha.

Os elementos visuais que representam um grupo na tela de edição são exibidos no formato de um quadrado com dois botões: um para edição, que seta os campos da Figura 26 para possibilitar alterar os dados, e o outro para excluir o grupo.

Já os elementos visuais que representam um grupo na “Tela Exibir Componente” (Figura 29) são diferentes e são apresentados na Seção 4.3.5.

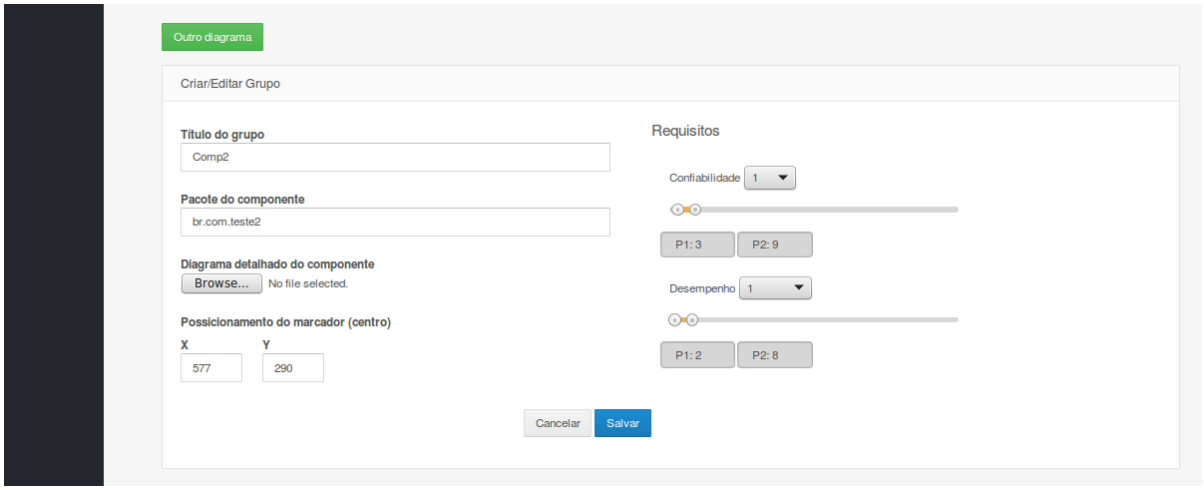
4.3.4 Importar Outro Diagrama

Nessa tela é possível importar outro diagrama para um componente (Figura 28, permitindo, com isso, visualizar várias visões de um determinado elemento da arquitetura. Com isso, é possível carregar um diagrama de classes e um diagrama de entidade e relacionamento (DER) de determinado pacote, por exemplo. Desse forma, é possível o relacionamento e a exibição de vários diagramas diferentes e a navegação entre eles.

4.3.5 Exibir Componentes

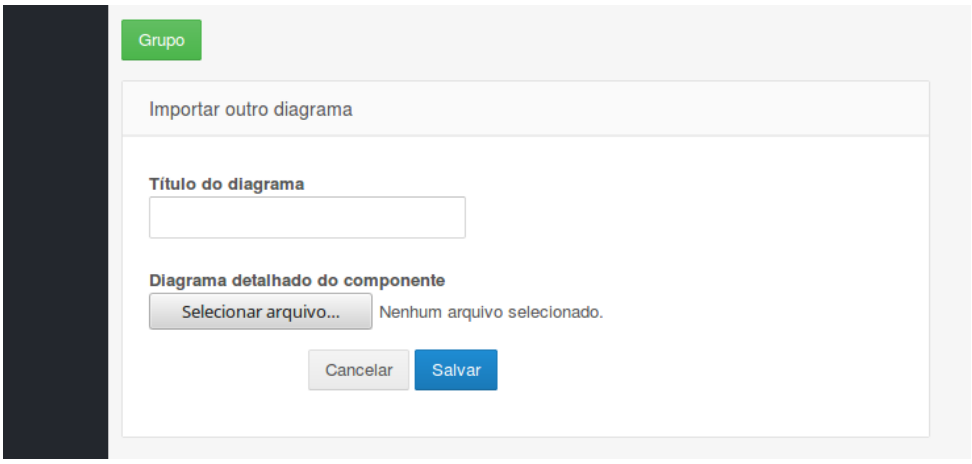
A “Tela Exibir Componentes” (ou “Tela Diagrama”) mostra um diagrama da arquitetura criado em outra ferramenta e carregado (importado) pelo usuário. No contexto da ferramenta,

Figura 27 – Tela de Edição - Campos para criar/editar um grupo



Fonte: Elaborada pelo autor.

Figura 28 – Tela Importar Outro Diagrama.



Fonte: Elaborada pelo autor.

os componentes do software pode ser um pacote, subpacote, classe ou método e são representados nos diagramas. Já o grupo (componente visual utilizado pela ferramenta VisRQ para exibir algumas informações dos *logs*) será exibido sobre o componente de software correspondente. Dessa forma, através da utilização dos grupos, as informações dos *logs* são contextualizadas com os diagramas da arquitetura de software importados.

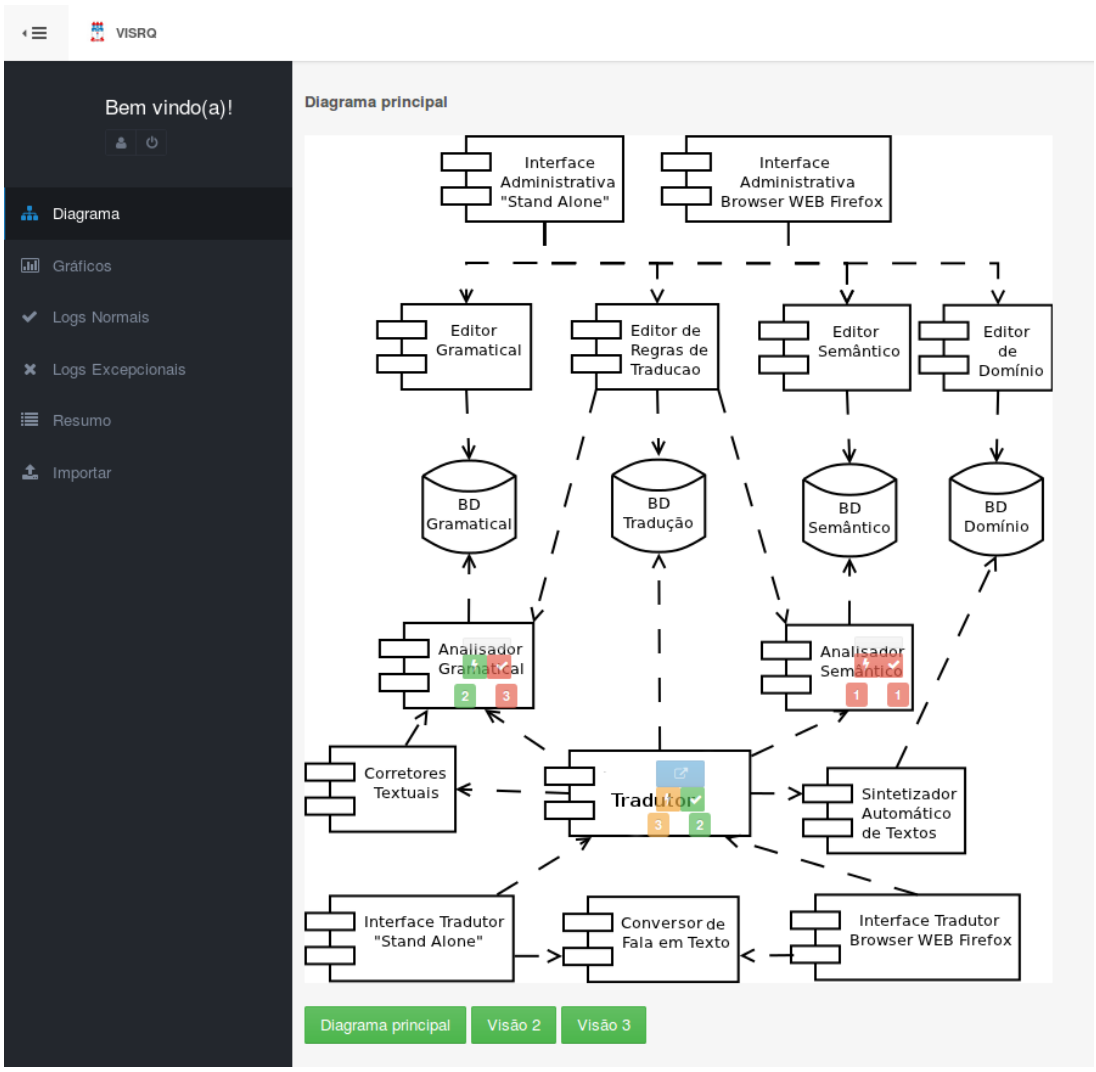
Conforme apresentado na Figura 29, os grupos que possuírem ao menos um diagrama mais detalhado (geralmente com o nível de abstração menor) terá um *link* na parte superior (com a cor azul) que encaminhará o usuário para visualizar as informações mais detalhadas do referido elemento arquitetural.

Em cada um dos grupos apresentados, são exibidas informações relativas aos requisitos de qualidade e possíveis refinamentos (visões mais detalhadas) através de três botões (ícones):

- o primeiro, que fica no topo do elemento, serve para o usuário navegar para diagramas mais detalhados do elemento visualizado, com visões menos abstratas e de menor granularidade;
- o segundo, que fica no canto inferior esquerdo de cada grupo, possui o ícone de um raio e representa o requisito desempenho;
- o terceiro, que fica no canto inferior direito de cada elemento, possui o ícone de um *check* e representa o requisito confiabilidade.

As informações sobre os requisitos de qualidade são exibidas através da utilização de uma escala de cores: verde (melhor), amarela (intermediária) e vermelha (pior). Essas cores são utilizadas como a cor de fundo do ícone do respectivo atributo de qualidade. E elas são determinadas através da definição dos limiares das métricas (intervalo de alerta) passados

Figura 29 – Tela Exibir Componentes - Diagrama Principal.



Fonte: Elaborada pelo autor.

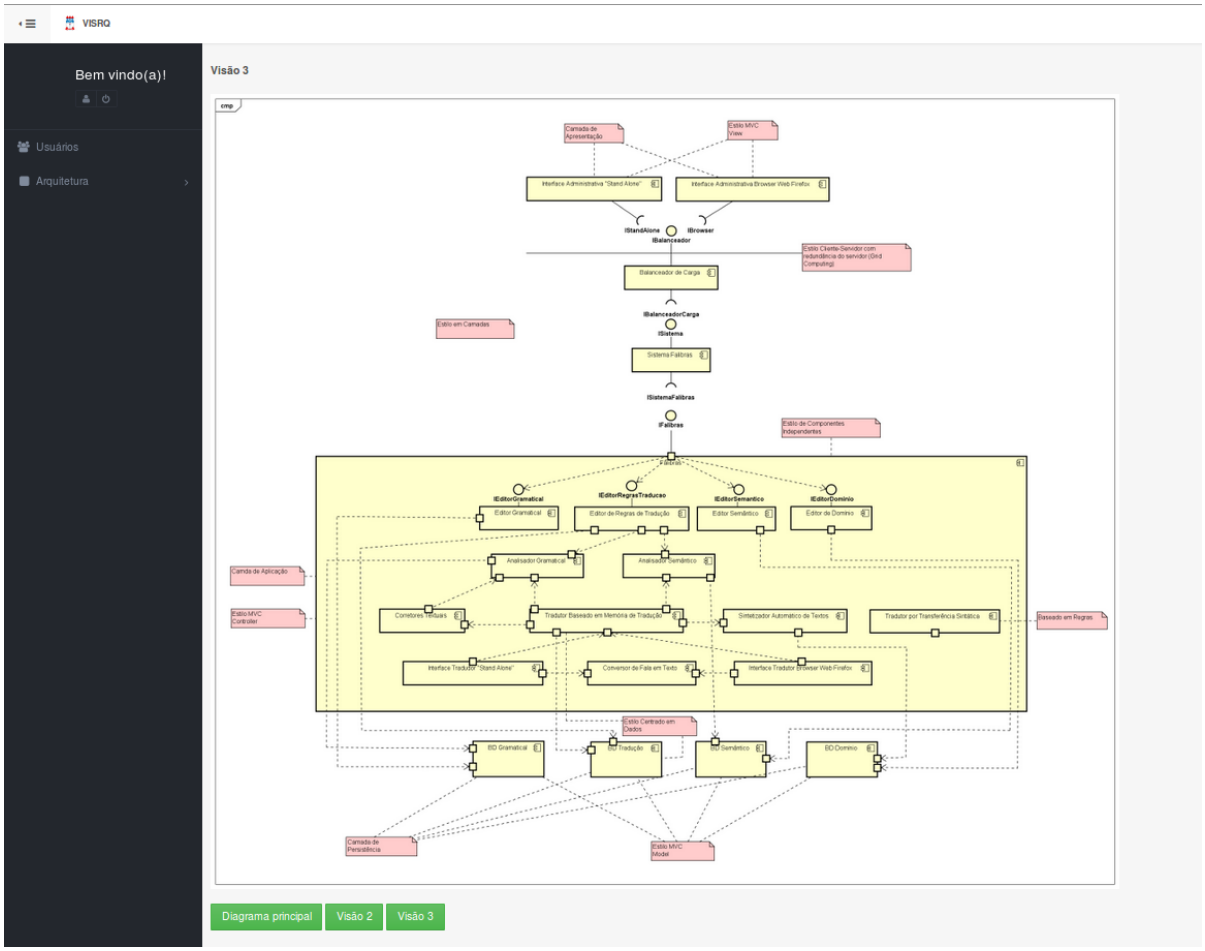
pelo usuário. Essa configuração pode ser feita utilizando as barras que estão localizadas na "Tela de Edição" na seção *Requisitos* (Figura 27). O intervalo de alerta corresponde ao trecho de valores em que os elementos visuais correspondentes a cada atributo são destacados com a cor amarela.

Na Figura 30, é exibido um diagrama auxiliar que está inserido no mesmo nível do diagrama principal: no topo da árvore de visualização. Esse diagrama não possui componentes sendo visualizados porque sua finalidade é apenas realçar a organização da arquitetura.

4.3.6 Gráficos

Nessa tela é possível pesquisar, por exemplo, informações sobre o tempo médio de execução dos métodos (Figura 31, gráfico azul) e o número de exceções (Figura 31, gráfico vermelho) em um determinado intervalo de tempo. Também são mostrados gráficos com a comparação da média do tempo de execução de cada componente, conforme é mostrado na

Figura 30 – Tela Exibir Componente - Diagrama Auxiliar.



Fonte: Elaborada pelo autor.

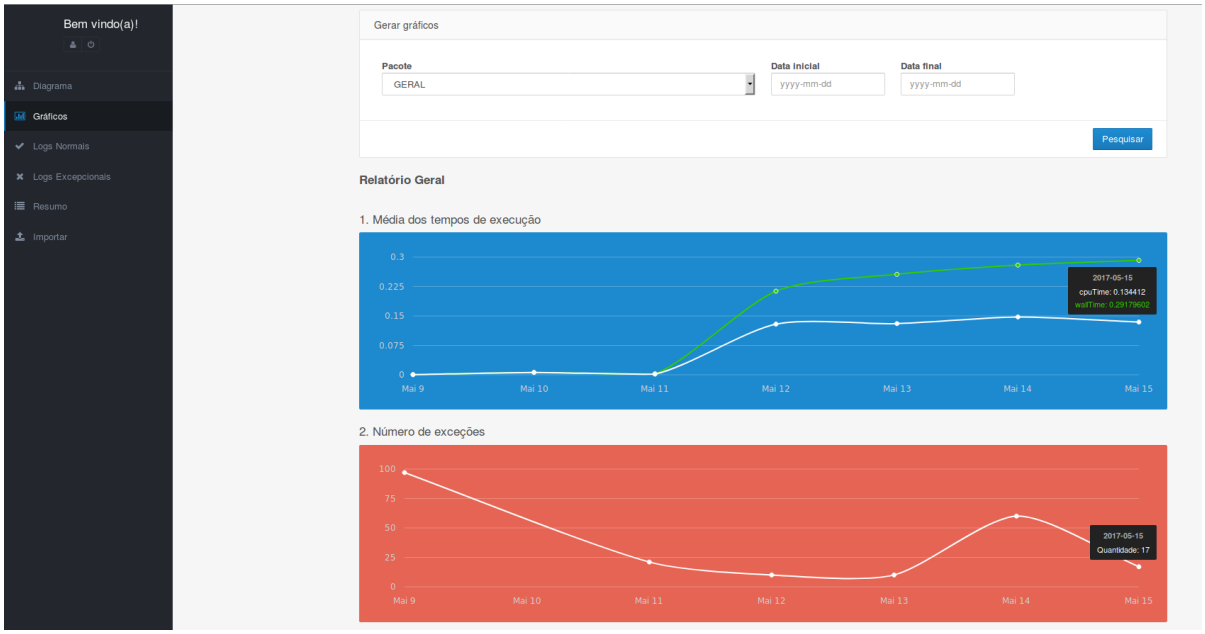
Figura 32. Nessa figura, é apresentada a comparação entre as médias do tempo de execução (primeiro gráfico) e entre as médias do tempo de execução normalizadas (segundo gráfico) dos componentes. Caso não deseje definir um intervalo de tempo, basta deixar os campos das datas vazios. Configurando o campo “Pacote” o usuário pode selecionar a opção GERAL para visualizar as informações de todos os componentes ou escolher um componente específico para visualizar apenas as informações sobre ele. Para um componente aparecer na listagem do campo “Pacote”, deverá existir um grupo cadastrado que o represente.

4.3.7 *Logs* Normais

Nessa tela, o usuário pode pesquisar e visualizar os *logs* gerados pelo funcionamento normal da aplicação. Os *logs* podem ser filtrados informando o identificador do componente e o período desejado (*Data inicial* e *Data final*). E podem ser ordenados pelo atributos *id*, *cpuTime*, *wallTime* e *datetime*.

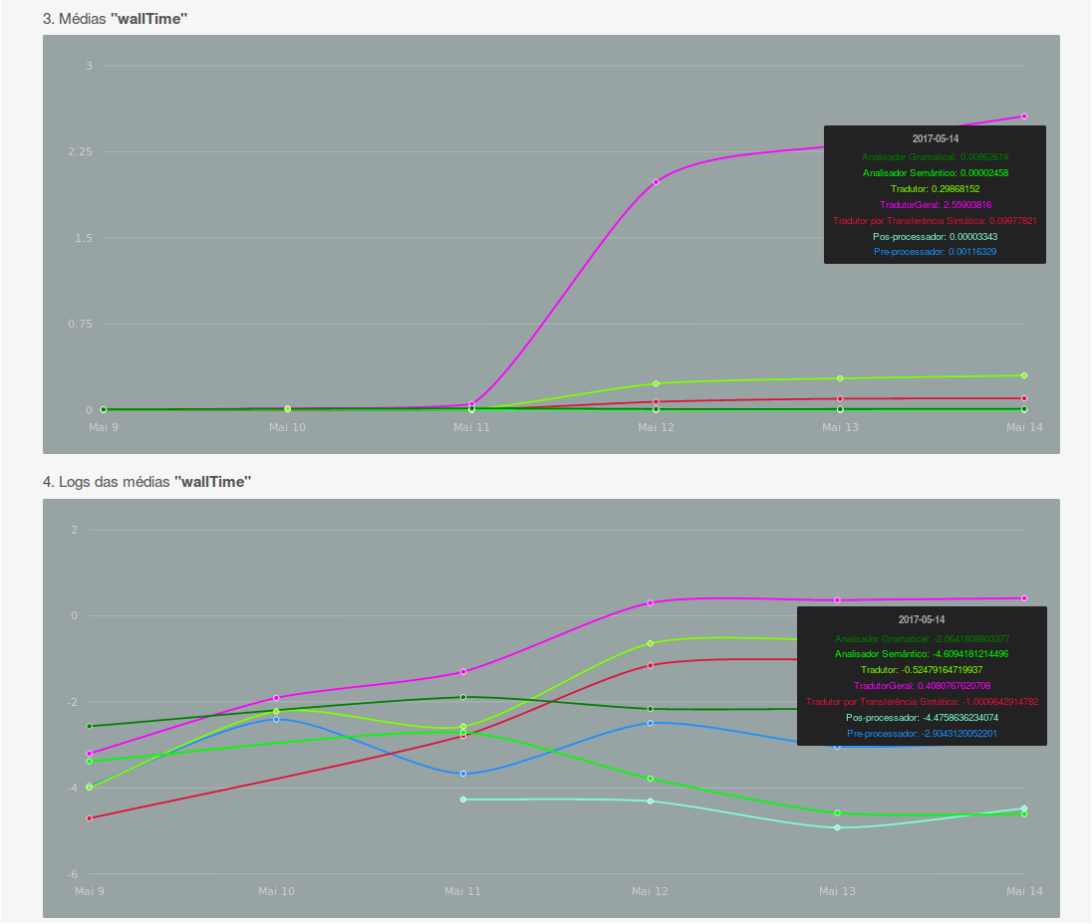
Na Figura 33, é mostrada a “Tela *Logs* Normais” com os campos de pesquisa na parte superior da tela e os *logs* encontrados sendo exibidos numa tabela abaixo dos campos de

Figura 31 – Tela Gráficos.



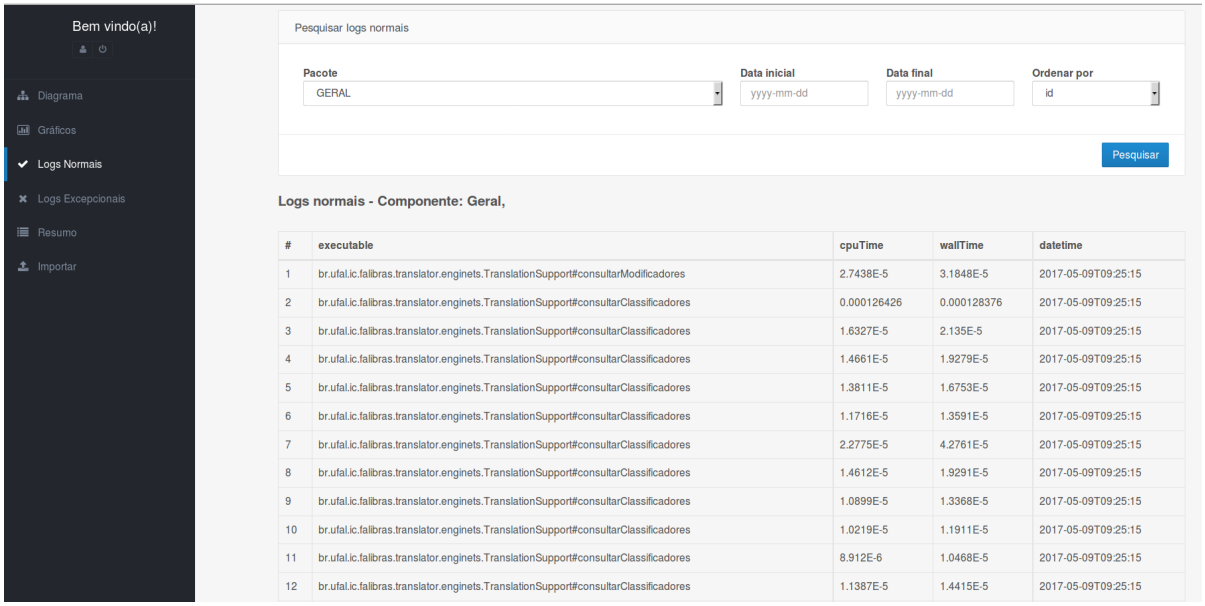
Fonte: Elaborada pelo autor.

Figura 32 – Tela Gráficos - Comparação entre os componentes.



Fonte: Elaborada pelo autor.

Figura 33 – Tela *Logs* Normais.



Fonte: Elaborada pelo autor.

pesquisa. Conforme é possível notar na figura, esses *logs* são compostos pelos seguintes dados:

- *executable* - Nome do pacote, da classe e do método executados;
- *cpuTime* - Total de tempo em segundos que a CPU foi usada para processar as instruções do elemento executável;
- *wallTime* - Total de tempo real em segundos entre o início até o término da execução do elemento;
- *datetime* - Data e hora, acompanhados do fuso horário, indicando quando os dados foram registrados.

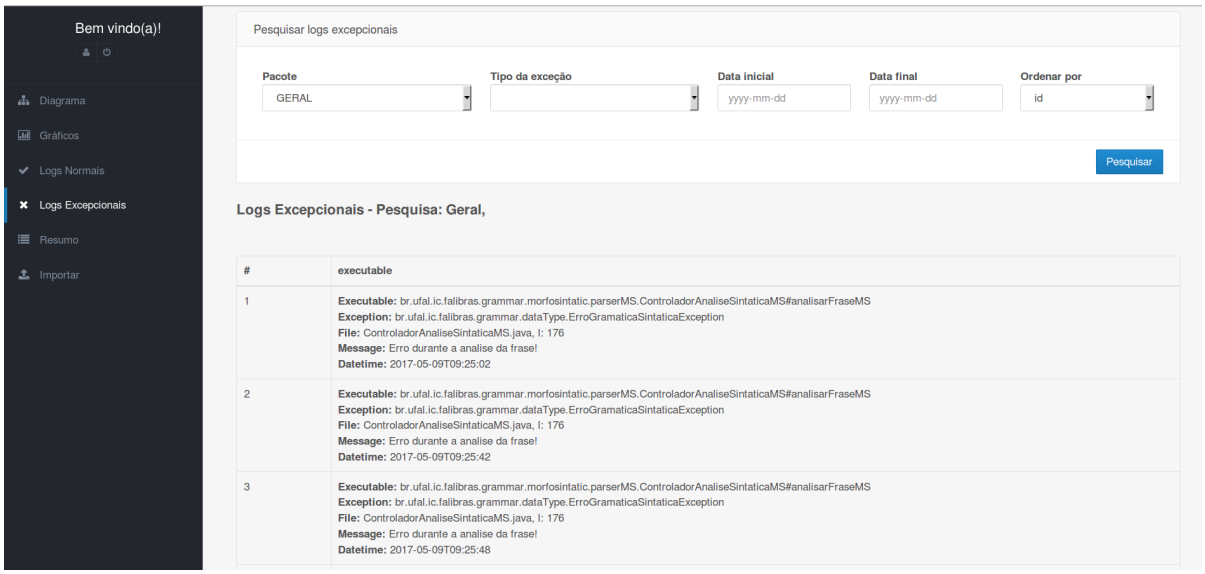
4.3.8 *Logs* Excepcionais

De maneira semelhante à "Tela *Logs* Normais", nessa tela o usuário pode pesquisar e visualizar os *logs* relacionados ao lançamento de exceções. Os *logs* podem ser filtrados informando o nome do componente, o período desejado (Data inicial e Data final) e o nome da exceção lançada, podendo ser ordenados pelos atributos *id*, *datetime*, *exception* e *file*.

Na Figura 34, é mostrada a "Tela *Logs* Excepcionais" com os campos de pesquisa na parte superior da tela e os *logs* encontrados sendo exibidos numa tabela abaixo dos campos de pesquisa. Conforme é possível notar na figura, esses *logs* são representados pelos campos *executable*, *datetime* e pelas informações relacionadas à exceção lançada, tais como:

- *exception* - Exceção lançada;

Figura 34 – Tela *Logs* Excepcionais.



Fonte: Elaborada pelo autor.

- *file* - Nome do arquivo e o número da linha onde a exceção foi lançada;
- *message* - Caso exista, mensagem da exceção.

4.3.9 Resumo dos *Logs*

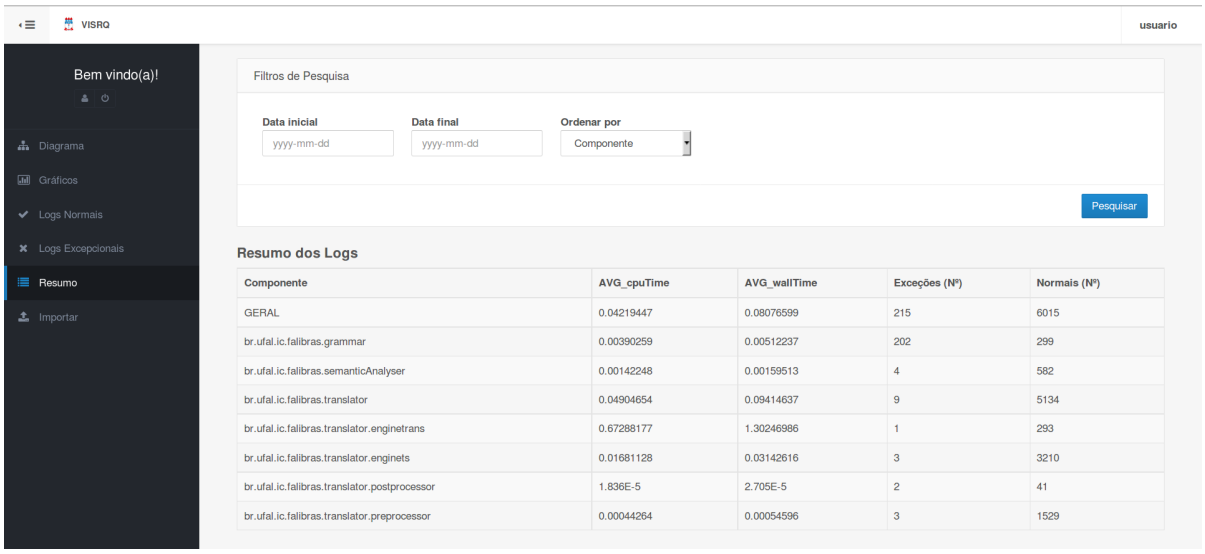
Na Figura 35, é mostrada a "Tela Resumo dos *Logs*" com os campos de pesquisa na parte superior e com o resumo das informações sendo exibidos numa tabela abaixo dos campos de pesquisa.

Nessa tela, é possível visualizar, em forma de tabelas, algumas informações sobre cada um dos componentes, tais como: a quantidade de *logs* normais e excepcionais ou a média do atributo *cpuTime* e *wallTime*. O usuário pode organizar as informações pelo nome do componente, pela quantidade de exceções (nº de *logs* excepcionais) ou pela média do tempo de CPU (*cpuTime*) ou do tempo de trabalho (*wallTime*).

4.4 LIMITAÇÕES E EXEMPLO DE USO DA FERRAMENTA VISRQ

A ferramenta VisRQ não gera diagramas. Ela utiliza os diagramas gerados/desenvolvidos em outras ferramentas como Posseidon, DIA, ArgoURL, Astha, etc. Por isso, as visualizações mostradas para os usuários dependem dos diagramas importados. Outra limitação da ferramenta é a análise de métodos que invocam outros métodos, pois aqueles vão apresentar um tempo de execução maior. Com isso, para uma análise mais detalhada da ferramenta monitorada, é aconselhável conhecer a sua arquitetura, uma vez que é responsabilidade do usuário:

Figura 35 – Tela Resumo dos *Logs*.



Fonte: Elaborada pelo autor.

- Anotar os componentes que serão monitorados durante sua execução pela ferramenta monitora QuAM *Framework*;
- Escolher e importar os diagramas que serão utilizados pela ferramenta VisRQ;
- Definir os elementos visuais (grupos) responsáveis por exibir as informações dos *logs* sobre os correspondentes componentes arquiteturais dos diagramas.

Dessa forma, considerando que os *logs* e os diagramas que serão utilizados pela ferramenta VisRQ já foram gerados, o usuário deve seguir os passos seguintes.

Inicialmente, o usuário deve importar o diagrama principal, que será utilizado, para a ferramenta VisRQ. Para isso, ele deve acionar o *link* "Diagrama", para aparecer a "Tela Carregar Diagrama da Arquitetura" e fazer o *upload* da imagem que desejar e informar um título para o diagrama, conforme apresentado na Figura 24.

Em seguida, com o diagrama principal importado, é necessário definir quais componentes devem ter suas informações exibidas pela ferramenta. Para isso, o usuário deve acionar o *link* "Editar", para que a "Tela de Edição" seja apresentada e, com isso, poder criar e editar os grupos. Na "Tela de Edição", são exibidos os grupos que foram criados para o diagrama em uso, conforme são mostrados na Figura 26. Para criar um grupo, o usuário deve escolher o ponto sobre o diagrama da arquitetura onde o grupo deve ser posicionado, dando um duplo clique na tela, informar o título do grupo, o pacote do(s) componente(s), fazer o *upload* de um diagrama mais detalhado do(s) componente(s) mapeado pelo grupo (caso seja do interesse importar tal diagrama) e, por fim, acionando o botão "Salvar".

Para editar um grupo, o usuário deve acionar o botão "Editar" que fica na parte superior da representação do grupo, conforme pode ser visto na Figura 26. Acionando esse

botão, as informações do grupo escolhido são exibidas nos respectivos campos, possibilitando realizar alterações.

Ao criar ou editar um grupo, o usuário pode definir os intervalos em que o atributo será marcado com as cores verde, amarela ou vermelha. Para isso, é necessário escolher os pontos delimitadores do intervalo de alerta para cada atributo analisado, utilizando as escalas localizadas no lado direito da Figura 27 (seção “Requisitos”).

A definição dos grupos é importante pois eles são responsáveis pela exibição das informações sobre os componentes e pelo relacionamento entre os diagramas, possibilitando a navegação entre esses diagramas e a exibição generalizada ou detalhada de informações.

Ainda na "Tela de Edição", o usuário pode carregar outro diagrama para o componente cujo grupo está sendo editado. Para isso, basta acionar o botão "Outro diagrama", que fica na parte superior dos campos "Criar/Editar grupo", para ser direcionado à "Tela Importar Outro Diagrama"(Figura 28), em que é possível fazer o *upload* da imagem desejada.

Caso o usuário deseje, um diagrama que foi importado para a aplicação pela associação com um grupo pode ter seus componentes/grupos associados com outros diagramas. Assim, é possível gerar hierarquias entre os diagramas, ampliando a árvore de visualização através do detalhamento dos pacotes, subpacotes, classes, até o componente mínimo que pode ser representado como um componente: o método. Para isso, é necessário navegar até o diagrama escolhido e acionar o *link* “Editar” para ser redirecionado à “Tela de Edição”, na qual deve ser criado o grupo do componente que devem ter o diagrama detalhado carregado para ser exibido.

Com os diagramas importados e com os grupos necessários definidos, é possível visualizar as informações dos *logs*. Para importar os *logs*, é preciso acionar o *link* “Importar” para ser redirecionado até a “Tela Importar *Logs*”. Nessa tela, o usuário deve escolher o arquivo de *logs* e acionar o botão “Importar” para fazer *upload* dos *logs* para a base de dados da aplicação VisRQ. A aplicação verifica se tais *logs* são registros de execução normal ou excepcional e os inserem na tabela correta.

Após as etapas citadas anteriormente, considerando os grupos definidos, o usuário pode visualizar as informações dos *logs*, que vão sinalizar a situação dos atributos de cada componente por meio do uso das cores vermelha, amarela e verde, conforme estão sendo mostradas na Figura 29. Acionando o botão que fica na parte superior do grupo, é possível visualizar um diagrama mais detalhado com informações mais específicas sobre determinado componente. Ou seja, é possível visualizar informações e diagramas mais gerais ou mais detalhados de acordo com a necessidade do usuário.

Caso o usuário deseje visualizar o resumo dos dados, ele pode utilizar a “Tela Resumo” para visualizar as principais informações sobre cada componente numa tabela.

Caso ele queira visualizar as informações dos *logs* através de gráficos e/ou com mais

detalhes, ele pode utilizar a "Tela Relatório", na qual é possível filtrar os *logs* selecionando o pacote ou componente desejado e definindo um intervalo de tempo, conforme apresentado na Figura 31.

Se desejar visualizar os *logs* completos, basta acionar os *link* "*Logs* Normais" e "*Logs* Excepcionais" para ser redirecionado para as telas "*Logs* Normais" (Figura 33) e "*Logs* Excepcionais" (Figura 34) respectivamente. Nessas telas, o usuário pode pesquisar e filtrar os *logs* da maneira que desejar, facilitando e aumentando a eficiência do processo de análise dos *logs*.

5 AVALIAÇÃO

Para o validação deste trabalho, foi realizado um experimento para verificar o impacto da ferramenta VisRQ no auxílio ao processo de análise dos *logs*.

Os dados de monitoração utilizado no experimento foram obtidos utilizando o QuAM *Framework* (LIMA, 2016) para monitorar o sistema Falibras (BRITO; FRANCO; CORADINE, 2012). A seguir, são apresentadas mais informações sobre o Sistema Falibras, o QuAM *Framework* e o experimento realizado.

5.1 O SISTEMA ALVO DO MONITORAMENTO: SISTEMA FALIBRAS

O Sistema Falibras é um tradutor do português (escrito ou falado), para Língua Brasileira de Sinais (LIBRAS), no formato gestual e animado (BRITO; FRANCO; CORADINE, 2012). Ele tem o objetivo de auxiliar na comunicação entre ouvintes e surdos, facilitando o convívio entre os mesmos e possibilitando aos surdos sua integração em locais públicos.

Tal sistema tem sido capaz de traduzir frases da língua portuguesa para LIBRAS, adicionando adequadamente classificadores para flexões e realizando análise sintática. Para isso, ele combina duas interfaces tradutoras: uma baseada em Transferência Sintática (Falibras-TS) e outra baseada em Memória de Tradução (Falibras-MT) (FALIBRAS, 2016).

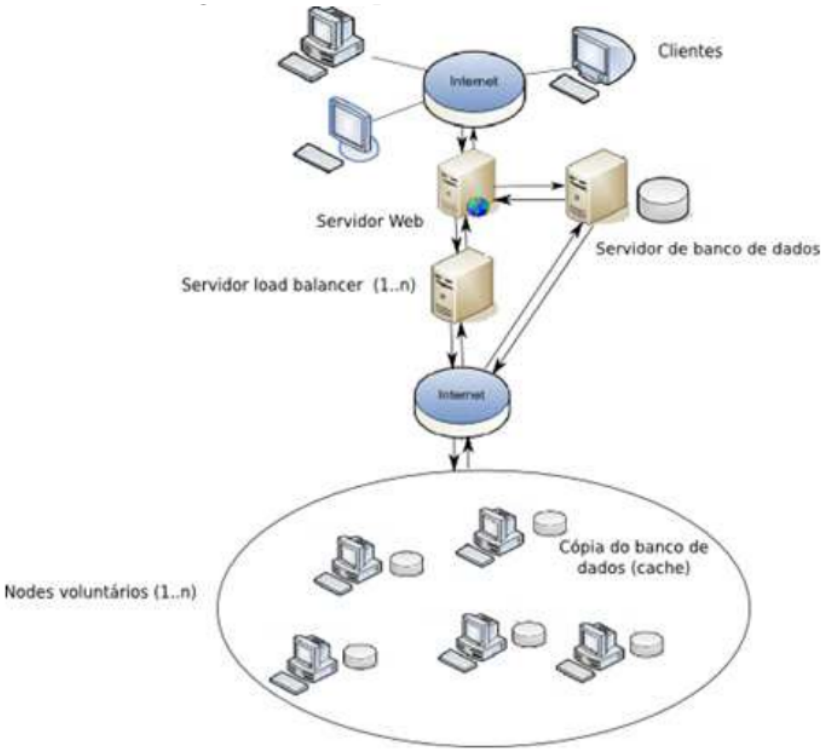
No Falibras-TS, as sentenças em LIBRAS são construídas utilizando processamento de linguagem natural, estruturando uma sequência de análise léxica, morfológica, sintática e de contexto, adequando a tradução para a estrutura frasal da LIBRAS. Já no Falibras-MT, os textos em português são interpretados a partir das estruturas morfossintáticas cadastradas na memória de tradução. Essa técnica permite o registro de tabelas de equivalência de formas Português-LIBRAS (FALIBRAS, 2016).

Atualmente, o servidor do Falibras possui uma arquitetura distribuída que utiliza Computação em Grade (*Grid Computing*) seguindo os estilo “Componentes Independentes” e “Cliente-Servidor” (BRITO; FRANCO; CORADINE, 2012; MENEZES, 2015). Conforme são mostrados na Figura 36, os componentes da arquitetura são:

- *Nodes* - Voluntários que contribuem com o *grid* processando as traduções;
- Clientes - Usuários que requisitam as traduções ao sistema via serviços Web;
- Servidor web - Entidade responsável por receber as solicitações dos clientes e encaminhá-las ao servidor de tradução;

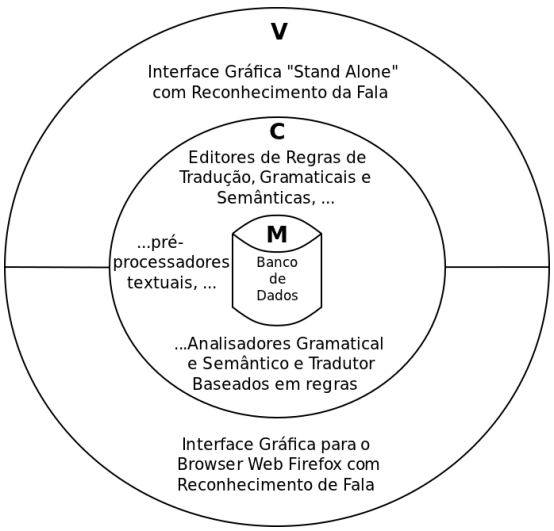
- Servidor de banco de dados - Responsável por armazenar o banco de dados de tradução e também disponibilizar os dados de *cache* para os *nodes*;
- Servidor *load balancer* - Possui a função de escalonar tarefas de tradução aos *nodes* utilizando o algoritmo Round-Robin, possibilitando ter mais de um servidor (MENEZES, 2015).

Figura 36 – Arquitetura do Sistema Falibras.



Fonte: Figura obtida de Brito, Franco e Coradine (2012).

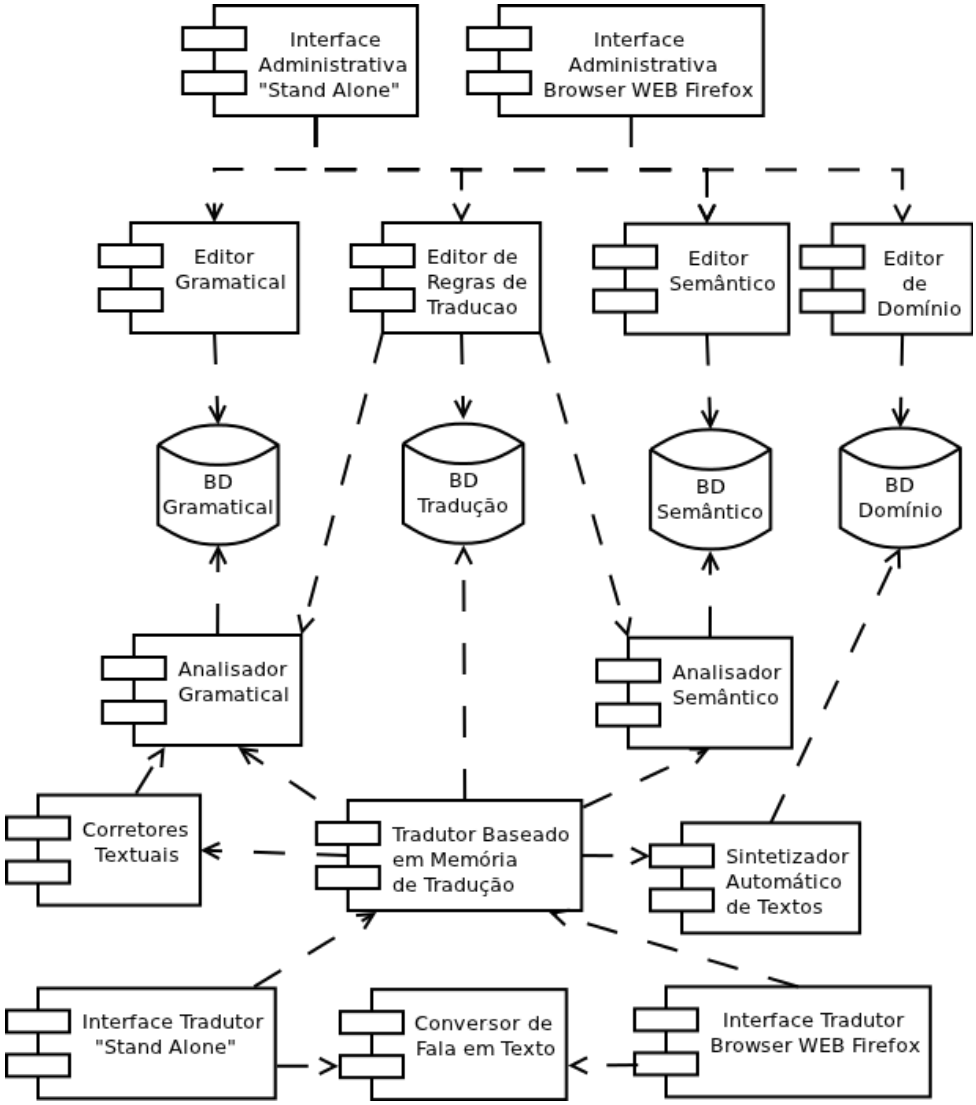
Figura 37 – Arquitetura de referência do sistema Falibras.



Fonte: Figura obtida de Brito, Franco e Coradine (2012).

A arquitetura do Falibras sofreu influência de três estilos e padrões arquiteturais: MVC[1], Centrado em Dados e Cliente-Servidor. Por isso, o detalhamento dos componentes obedece a separação proposta pelo padrão MVC, conforme é mostrado na Figura 37. Já na Figura 38, são mostradas as interações entre os módulos do sistema Falibras e a modularização funcional baseada em componentes.

Figura 38 – Componentes do sistema Falibras.

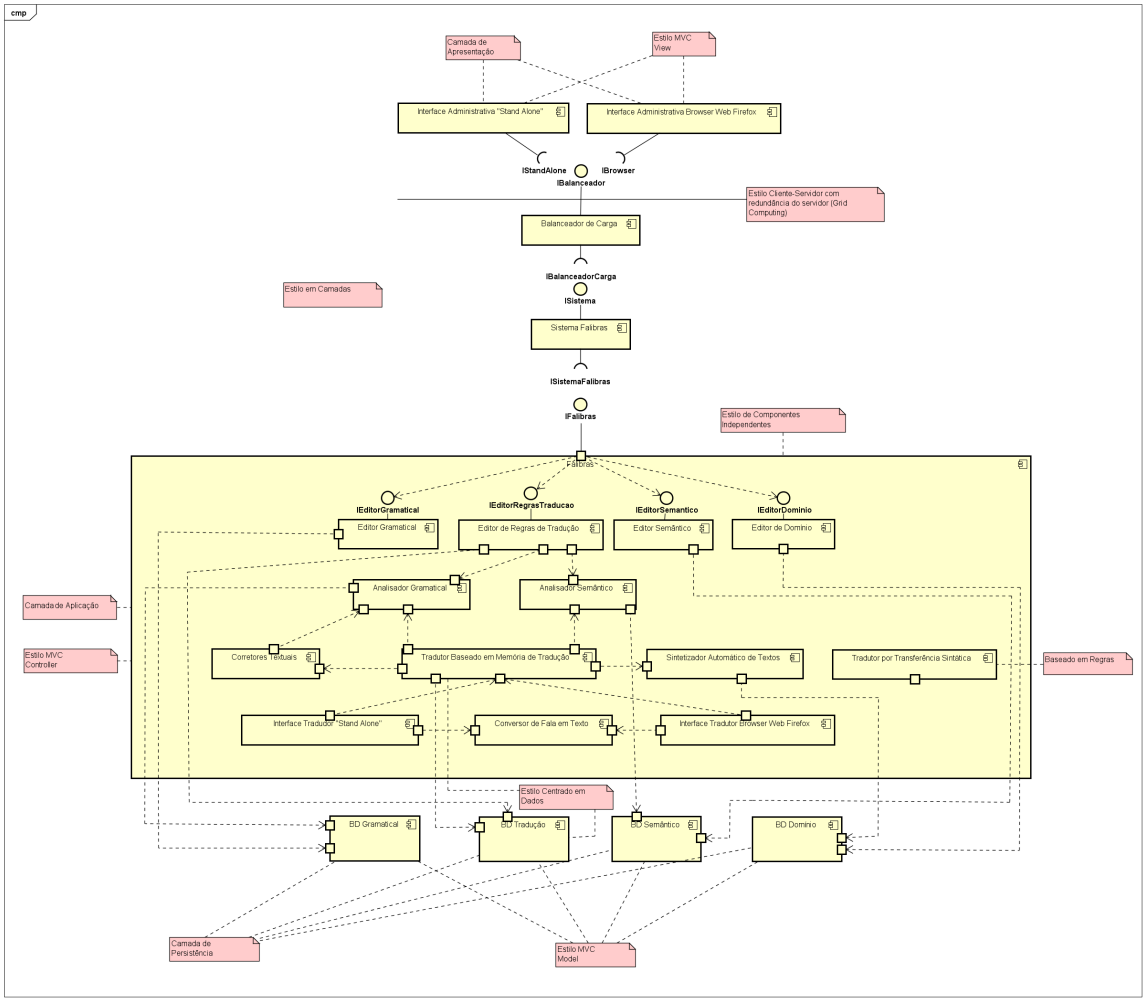


Fonte: Figura obtida de Brito, Franco e Coradine (2012).

Observando as Figuras 37 e 38, nota-se que o componente de persistência de dados está localizado na camada de modelo (*Model*). Na camada de controle (*Controller*) encontram-se funcionalidades básicas, como editores de regras de tradução, editor gramatical, analisador gramatical e semântico, corretor ortográfico e gramatical, sintetizador automático de texto e tradutor. E na camada de visão (*View*) estão as interfaces do sistema, que podem ser *desktop* ou *web* e permitem a interação do usuário por meio de entradas em formato de textos, escritos ou falados. Com isso, a camada de visão requisita as funcionalidades da camada de controle,

[1] Do inglês: *Model-View-Controller*

Figura 39 – Diagrama de componentes com estilos.



Fonte: Figura obtida de Menezes (2015).

localizando os editores, que irão realizar a busca no banco de dados na camada de modelo. As informações obtidas passarão pelos analisadores e poderão ir para o tradutor, no qual serão direcionadas para a saída na forma de animação para o usuário (BRITO; FRANCO; CORADINE, 2012).

Na Figura 39, é mostrado um diagrama da arquitetura do sistema Falibras com os estilos arquiteturais utilizados. Tal diagrama facilita a compreensão de como os estilos estão organizados dentro do sistema, sendo importante para auxiliar no planejamento da monitoração dos componentes do sistema (MENEZES, 2015).

Vale destacar que apenas um dos *nodes* que executa o sistema Falibras foi monitorado para gerar os *logs* utilizado nessa pesquisa.

5.2 A FERRAMENTA DE MONITORAMENTO - QUAM *FRAMEWORK*

O QuAM *Framework* (LIMA, 2016) utiliza programação orientada a aspectos e pode ser utilizado em aplicações orientadas a aspectos ou orientadas a objetos. Essa ferramenta é capaz de monitorar os elementos do software em tempo real através do uso de *Java Annotations* e também permite a criação de monitores personalizados, além do registro de arquivos de *log* em formato JSON[2]. Tal ferramenta pode monitorar os métodos e construtores da aplicação-alvo. Para isso, durante o processo de monitoramento, o diretório de *logs* é criado automaticamente na raiz da aplicação-alvo em que o *framework* é usado. Esse diretório contém os dados de *log* gerados durante a monitoração separados em dois arquivos: um com os *logs* da execução normal e outro com os *logs* de exceção.

5.3 EXPERIMENTO PILOTO

Foi realizado um experimento piloto que contou com a participação de 10 alunos do curso de Ciência da Computação da Universidade Federal de Alagoas - Campus Arapiraca e que teve como objetivo coletar informações para planejar o experimento avaliativo.

Através da análise desse experimento e dos relatos dos alunos foi possível notar que:

- As informações sobre os componentes mostradas pela ferramenta VisRQ não foi perceptível para um dos alunos devido ao seu problema de visão;
- As funções utilizadas no editor de planilhas, embora simples, não eram do domínio dos alunos, isso interferiu no resultados;
- A fato da existência de outras atividades acadêmicas depois do experimento pode ter interferido, pois alguns alunos responderam um dos questionários baseados na análise do processo anterior, errando a resposta uma vez que os *logs* analisados eram diferentes;
- A quantidade de dúvidas e erros identificados pode ser decorrência da falta de atenção durante a explicação ou durante a leitura do texto com as instruções ou ainda da falta de uma demonstração da execução dos processos (conforme foi relatado por um dos alunos);
- Alguns alunos, mesmo cometendo erros, relataram que gostaram da ferramenta, pois ela facilitou o processo de análise e também sugeriram algumas melhorias.

Uma das melhorias sugeridas foi a exibição de um gráfico que comparasse o tempo médio de todos os componentes simultaneamente, pois no caso de muitos componentes torna-se trabalhoso analisá-los individualmente devido a quantidade de informações e a diferença entre as escalas utilizadas.

[2] JSON: sigla do inglês de *JavaScript Object Notation*

Essa sugestão, originou também a ideia de uma nova maneira de visualizar as informações dos *logs*: utilizando uma lista que exibisse as informações sobre os *logs* dos componentes, mostrando o identificador (na forma *pacotes.classe#metodo*), o tempo médio de execução, a quantidade de exceções e o número de *logs* registrados.

Visando solucionar esses problemas, foi criado um vídeo explicativo para mostrar a execução dos processos e o uso das ferramentas de análise utilizando uma aplicação de exemplo e também foram realizadas algumas alterações na interface gráfica da ferramenta VisRQ para melhorar a usabilidade.

5.4 AVALIAÇÃO DOS PROCESSOS E DA FERRAMENTA VISRQ

Visando validar a ferramenta VisRQ, foi realizado um experimento com usuários para verificar o impacto do uso de tal ferramenta ao analisar os *logs* dos requisitos desempenho e confiabilidade do sistema no olhar da arquitetura. Após o uso da ferramenta, foi aplicado um questionário para verificar os aspectos qualitativos do uso da ferramenta. Dessa forma, para a criação do questionário avaliativo, foi utilizada a abordagem GQM (Goal-Question-Metric).

5.4.1 Planejamento do experimento

O experimento buscou avaliar o processo de análise dos *logs* com e sem o uso da ferramenta VisRQ, na perspectiva dos desenvolvedores, engenheiros e/ou arquitetos de software. Nele a taxa de acertos (respostas corretas), a eficiência (tempo de análise) e outras opiniões dos participantes sobre os processos foram observadas. Para isso, o experimento foi dividido em duas partes: uma consistiu em analisar um conjunto de *logs* do sistema Falibras utilizando um editor de planilhas e a segunda, outro conjunto de *logs* com a ferramenta VisRQ. Entre as informações obtidas estão o perfil dos participantes, informações relacionadas à execução do processo (eficiência, índice de acertos, opinião sobre os processos, etc.) e a ferramenta VisRQ.

Vale ressaltar que, a partir do experimento piloto, o experimento avaliativo foi planejado para:

- Representar as principais dificuldades enfrentadas para analisar os *logs*;
- Poder ser executado em, no máximo, uma hora e meia, pois mais do que isso poderia interferir no resultado, já que os participantes poderiam se preocupar com outras atividades acadêmicas e responder os formulários sem analisar com atenção as perguntas;
- Poder ser executado por um usuário com conhecimento mediano em relação a editores de planilhas. Dessa forma, a análise com o editor de planilhas foi baseada na aplicação de fórmulas básicas, como a MÉDIA() e CONT.VALORES();

- Avaliar apenas o processo de análise, por isso tarefas preparativas (importação, formatação dos *logs*, etc.) foram desconsideradas.

5.4.1.1 Fatores, variáveis de resposta e processos analisados

A variável independente (fator) do experimento é **ferramenta**, que corresponde a ferramenta utilizada para analisar os *logs* de monitoramento. Esse fator possui dois níveis: Editor de planilhas ou VisRQ.

Já as variáveis de resposta são:

- **Número de acertos**: quantidade de acertos decorrente do uso de cada ferramenta;
- **Tempo de análise**: tempo gasto para analisar os *logs* com cada ferramenta;

Neste trabalho, os processos analisados estão relacionados com o fator **ferramenta**. O primeiro processo consiste na análise dos *logs* com o editor de planilhas (Processo 1) e o segundo, na análise utilizando a ferramenta VisRQ (Processo 2).

Com o editor de planilhas, os participantes organizaram/agruparam os *logs* de acordo com o nome do componente para extrair a média do tempo de execução e a quantidade de exceções de um determinado componente utilizando as devidas funções do editor de planilhas. Os *logs* foram disponibilizados num arquivo no formato *.ods* ou *.xlsx*, para facilitar a importação para o editor de planilhas.

Com a ferramenta VisRQ, os participantes analisaram os *logs* buscando extrair a média do tempo de execução e a quantidade de exceções dos componentes analisados. Porém, como tal ferramenta automatiza a ordenação dos *logs* e o cálculo do número de exceções e a média do tempo de execução, as chances de erro será muito menor, uma vez que não é necessário contar linhas, copiar ou digitar fórmulas como ocorre no uso do editor de planilhas.

Vale ressaltar que a análise dos *logs* diretamente no arquivo não foi realizada por ser bastante trabalhosa, inviável na prática. Esse fato foi comprovado por um experimento realizado anteriormente por Menezes (2015), em que alguns participantes desistiram.

5.4.1.2 Identificação dos participantes

O público participante da avaliação dos processos de análise dos *logs* e da ferramenta VisRQ foi composto por voluntários, alunos do curso técnico de informática do Instituto Federal de Alagoas - Campus Arapiraca (IFAL), alunos e ex-alunos do curso de Ciência da Computação da Universidade Federal de Alagoas - Campus Arapiraca (UFAL), profissionais de tecnologia da informação da UFAL - Campus Arapiraca.

Para a realização da avaliação, foi aplicado o delineamento em quadrado latino (KEEDWELL; DÉNES, 2015) para planejar a ordem que os recursos deveriam ser disponibilizados para os participantes.

Tabela 11 – Quadrado latino.

Grupos	**Nº Participantes**	**E. Planilhas**	**VisRQ**	**Descrição**
A1	4	1º	2º	Discentes curso técnico
A2	8	1º	2º	Graduandos em Ciência da Computação
A3	10	1º	2º	Profissionais graduados
B1	5	2º	1º	Discentes curso técnico
B2	9	2º	1º	Graduandos em Ciência da Computação
B3	5	2º	1º	Profissionais graduados

Fonte: Elaborada pelo autor.

Conforme o planejamento mostrado no quadrado latino (Tabela 11), os participantes foram divididos em 6 grupos, em que:

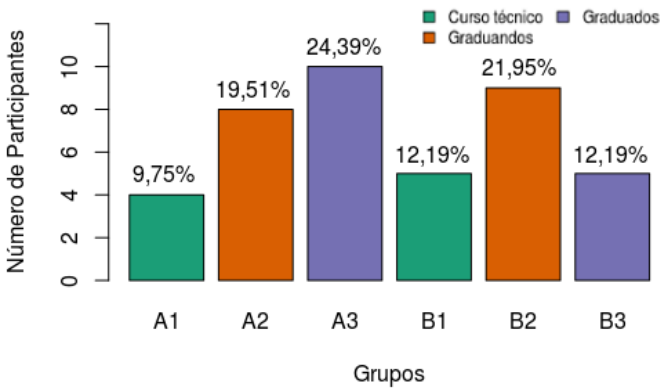
- Os grupos A1, A2 e A3 utilizaram primeiro o editor de planilhas e, em seguida, a ferramenta VisRQ para realizar a análise dos *logs*;
- Os grupos B1, B2 e B3 analisaram os *logs* utilizando primeiro a ferramenta VisRQ e, em seguida, o editor de planilhas;
- Os grupos A1 e B1 são composto por discentes do curso técnico de Informática;
- Já os grupos A2 e B2 são composto por graduandos em Ciência da Computação;
- E os grupos A3 e B3 são compostos por profissionais graduados em Análise de Sistema ou Ciência da Computação.

A inclusão dos participantes nesses grupos ocorreu com base na escolaridade seguindo a ordem: “com a ferramenta VisRQ” e “com o editor de planilhas”. Porém, alguns voluntários se propuseram e não participaram do experimento, por isso os grupos do mesmo nível de escolaridade possuem tamanhos diferentes.

Na Figura 40, é apresentado o número de participantes e a porcentagem de cada grupo em relação ao número total de participantes.

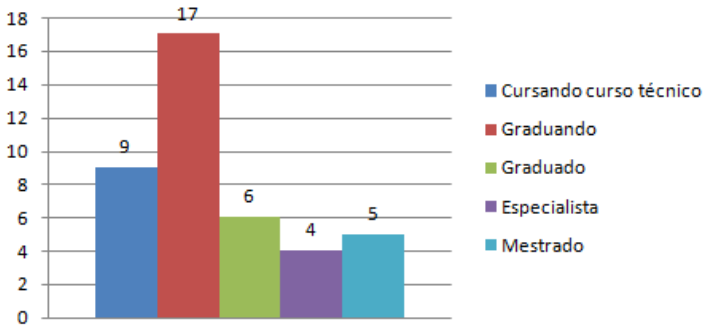
Na Figura 41, é mostrado o nível de escolaridade dos participantes. Nela é possível observar que 9 participantes estão no curso técnico em Informática (nível médio), 17 estão na graduação, 6 são apenas graduados, 4 são especialistas e 5 são mestres. Ou seja, 15 participantes são graduados.

Figura 40 – Número de participantes por grupo.



Fonte: Elaborada pelo autor.

Figura 41 – Nível de escolaridade.



Fonte: Elaborada pelo autor.

5.4.1.3 Ameaças à validade

A seguir, algumas ameaças à validade deste trabalho serão discutidas. A análise de tais ameaças foi baseada nas divisões proposta por Wohlin et al. (2012): ameaças à validade interna, de conclusão, de construção e externa.

As principais ameaças à validade interna identificadas foram:

- **Maturação:** Os participantes podem ficar mais motivados ou desmotivados com o passar do tempo, uma vez que os processos possuem a mesma finalidade e um é manual enquanto o outro foi automatizado. Provavelmente essa comparação foi realizada durante a execução da análise dos *logs*.
- **História**: Há a possibilidade de que o resultado da análise com o editor de planilhas tenha sofrido influência do conhecimento e experiência dos usuários com esse tipo de software.
- **Mortalidade seletiva:** Cinco voluntários graduados do grupo B não realizaram o experimento, com isso o número de graduados no grupo A foi o dobro do grupo B.

Porém, essa ameaça foi minimizada pois cada participante realizou a análise com as duas ferramentas.

- **Efeito da expectativa do sujeito:** Os participantes foram observados pelo avaliador, isso pode ter interferido de forma positiva ou negativa no resultado.

Já entre as ameaças à validade de conclusão, não foi identificada nenhuma ameaça.

E entre as ameaças à validade de construção, é possível citar:

- **Projeto do experimento:** É possível que o caso de teste usado não tenha diferenciado os processos de forma significativa. Uma vez que o caso utilizado pode ser considerado simples, com uma pequena quantidade de *logs*, pois o experimento não podia ser muito demorado. Na prática, a quantidade de *logs* e recursos da ferramenta VisRQ utilizados poderia ser maior.
- **Fatores humanos (ou sociais):** O nível de concentração ou o comprometimento dos participantes pode afetar o número de acertos e o tempo necessário para realizar a análise dos *logs* e, com isso, o experimento.

Já em relação as ameaças à validade externa, foi identificada uma:

- **Participantes:** A maioria dos participantes são alunos, o ideal seria desenvolvedores, engenheiros ou arquitetos de software.

5.4.2 Execução do experimento

A execução do experimento avaliativo envolveu os seguintes passos:

1. Leitura das instruções iniciais;
2. Treinamento - vídeo demonstrativo com a análise de um pequeno conjunto de *logs* para entender e aprender os processos e o preenchimento dos questionários sobre os resultados das análises;
3. Esclarecimentos das dúvidas sobre os processos de análise dos *logs* (caso existissem);
4. Análise dos *logs* do sistema Falibras com o editor de planilhas e com a ferramenta VisRQ e também preenchimento de questionários sobre o resultado dessas análises;
5. Análise estatística dos resultados.

Para realizar o experimento e responder as perguntas determinadas, os participantes foram divididos em seis grupos de forma a compor um quadrado latino (KEEDWELL; DÉNES, 2015).

Para ambos os grupos foi disponibilizado um arquivo de texto com todas as instruções necessárias para a realização do experimento e um vídeo que demonstrava como aplicar cada um dos processos, utilizando a ferramenta correspondente (VisRQ e editor de planilhas) para analisar os *logs*.

O vídeo de demonstração tinha o objetivo de familiarizar os participantes com os processos de análise dos *logs* com e sem o uso da ferramenta VisRQ, servindo de treinamento antes da análise dos *logs* do sistema Falibras. Para o vídeo, o conjunto de *logs* utilizado foi gerado a partir de uma aplicação de exemplo e foi analisado com as duas ferramentas. Tal aplicação trata-se de uma calculadora com 4 métodos: *div*, *mult*, *som* e *sub*. Os códigos fonte utilizado para gerar esse primeiro conjunto de *logs* pode ser visto no Apêndice A.

Para a análise com o editor de planilhas, foram utilizados 20 *logs* de execução normal e 4 de exceções. Já para a análise com a ferramenta VisRQ foram utilizados 126 *logs* de execução normal e 9 de exceções. Essa pequena quantidade de *logs* é justificável pois tal vídeo teve finalidade didática.

Com objetivo de gerar os *logs* necessários para a etapa de avaliação, o sistema Falibras foi monitorado. Para isso, foram escolhidos textos da página [3] web que conta a história do projeto Falibras. Com isso, foram obtidas 30 frases, de diferentes tamanhos. Buscando gerar *logs* com exceções, foram adicionadas ao plano de teste palavras e frases em inglês, com caracteres especiais, com erros ortográficos, entre outras.

As "strings" que foram utilizadas como entradas do sistema Falibras para a geração dos *logs* estão descritas na Tabela 21 e as partições de equivalência adotadas estão nas Tabelas 22 e 23 do Apêndice B.

Para simular a degradação do sistema, foi inserida uma falha no componente "Tradutor Geral". O objetivo foi verificar se os participantes notavam tal problema e identificavam qual componente foi o causador.

Com isso, no experimento, o conjunto de *logs* utilizado na análise com o editor de planilhas possuía 1380 *logs* de execução normal e 50 de exceções. Já o conjunto utilizado com a ferramenta VisRQ possuía 6015 *logs* de execução normal e 215 de exceções, sendo que os *logs* utilizados com cada ferramenta eram diferentes.

Buscando obter um número maior de participantes e também para que a execução do experimento se adequasse a disponibilidade de horário dos voluntários, a avaliação foi realizada de forma presencial e à distância. Mas, em ambos os casos, teve o acompanhamento do avaliador para tirar dúvidas, seja presencialmente ou via *Hangout*.

Por isso, o tempo gasto para realizar as atividades foi calculado por meio da diferença entre a hora em que cada formulário foi submetido (o inicial, o da análise com a ferramenta VisRQ e o da análise com o editor de planilhas).

[3] Página: http://www.falibras.org/nossa-hist-ria. Acesso em: 12 de dezembro de 2016.

Vale ressaltar que o experimento teve como objetivo comparar a análise dos *logs*, desconsiderando as etapas de preparação para análise, tais como: conversão de formato, importação dos *logs* e definição dos elementos visuais. E os participantes foram informados do que seria observado no experimento.

5.4.3 Planejamento GQM para avaliação da ferramenta VisRQ

A abordagem *Goal-Question-Metric* (GQM) é bastante utilizada para estruturar a avaliação de produtos ou processos de software (BASILI; CALDIERA; ROMBACH, 1994). De acordo com essa abordagem, é necessário definir inicialmente os objetivos ou metas (*goal*) para realizar a mensuração. Em seguida, é necessário especificar as perguntas (*questions*), que serão respondidas através de métricas (*metrics*). Para o sucesso da avaliação, as metas devem ser claras. Na Figura 42, é mostrada graficamente a representação hierárquica do modelo GQM, destacando os elementos que devem ser definidos para utilizar esse modelo.

Figura 42 – Definição das métricas GQM.

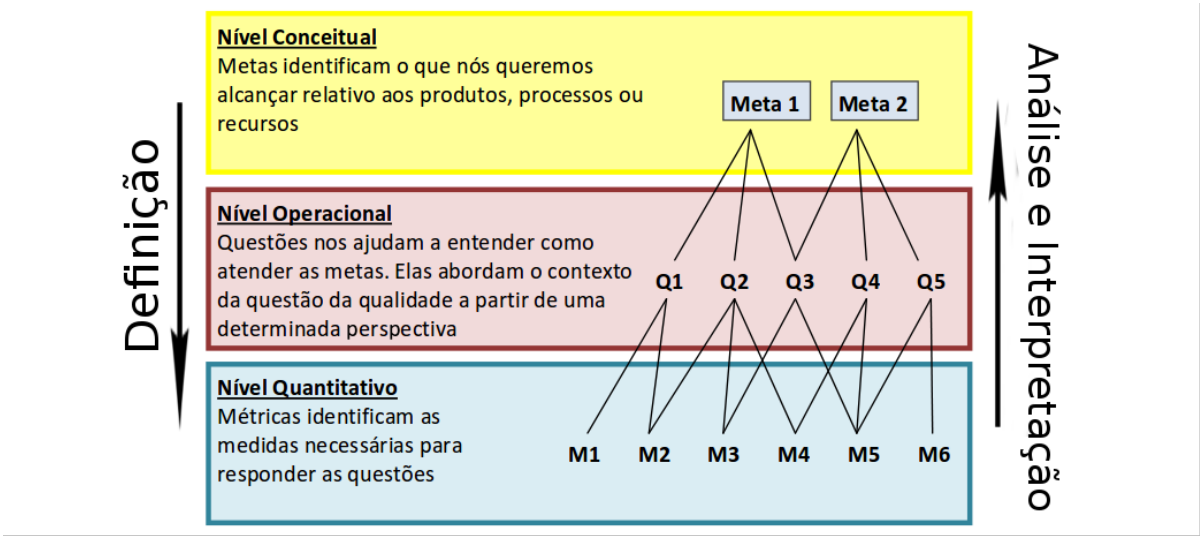


Figura adaptada de Silva et al. (2009).

Conforme ilustrado na Figura 42, uma única medição da meta pode ser aplicada a múltiplos objetivos. Para cada meta, pode haver várias questões e uma mesma questão pode estar relacionada a vários objetivos. Para cada questão, podem existir várias métricas e algumas delas podem ser aplicáveis a mais de uma questão. Assim, o mapeamento entre os objetivos, questões e métricas não é um para um. Dessa forma, a utilização desta estrutura hierárquica ajuda a assegurar que o modo de medição foca nas métricas corretas e evita trabalho extra (SILVA et al., 2009). Por isso, para utilizar o método GQM, foi necessário definir os objetivos (G), as perguntas (Q) e as métricas (M) no contexto do experimento da ferramenta VisRQ. Sendo assim, tem-se:

G1: Avaliar se a ferramenta torna o processo de análise dos *logs* mais preciso e rápido (eficiente).

- **Q1.** A ferramenta torna o processo de análise dos *logs* mais preciso?

  M1: Média dos acertos utilizando o editor de planilhas;

  M2: Média dos acertos utilizando a ferramenta VisRQ;

- **Q2.** A ferramenta torna o processo de análise dos *logs* mais rápido (eficiente)?

  M3: Média dos tempos utilizando o editor de planilhas;

  M4: Média dos tempos utilizando a ferramenta VisRQ;

G2: Avaliar o tempo de aprendizado da ferramenta VisRQ.

- **Q3.** O uso das ferramentas pode ser aprendido em menos de 6 minutos?

  M5: Porcentagem de participantes que classificaram o grau de facilidade para entender o processo de análise com a ferramenta VisRQ como *Muito fácil: pode ser aprendido/entendido em menos de 3 minutos* ou *Fácil: entre 3 e 6 minutos.*

G3: Avaliar a aceitação da ferramenta VisRQ.

- **Q4.** Qual processo você prefere

  M6: Porcentagem de participantes que responderam "Com a ferramenta VisRQ".

- **Q5.** Você acha que a utilização da ferramenta VisRQ facilita o processo de análise?

  M7: Porcentagem de participantes que responderam "Muito: ajuda a reduzir o esforço consideravelmente".

- **Q6.** Como você avalia cada processo de análise dos *logs*?

  M8: Porcentagem de participantes que avaliaram o processo com a ferramenta VisRQ como *Muito bom: pode ser utilizado da forma que está* ou *Bom: necessita de ajustes pontuais.*

G4: Avaliar a importância das principais funcionalidades da ferramenta VisRQ.

- **Q7.** Importar e utilizar diagramas desenvolvidos em outras ferramentas (como DIA, Astha, etc.) é uma funcionalidade útil

  M9: Porcentagem de participantes que responderam "Sim".

- **Q8.** Associar os elementos arquiteturais registrados nos *logs* com os diagramas importados facilita o entendimento?

  M9: Porcentagem de participantes que responderam "Sim".

- **Q9.** A visualização das informações dos *logs* de forma gráfica é útil?

  M9: Porcentagem de participantes que responderam "Sim".

- **Q10.** A visualização das informações dos *logs* de forma resumida é útil?

  M9: Porcentagem de participantes que responderam "Sim".

G5: Avaliar usabilidade da ferramenta VisRQ para analisar os *logs*.

- **Q11.** A ferramenta VisRQ diminui a quantidade de erros de uso durante a análise dos *logs*?

M10: Média de erros na utilização.

As respostas em que a métrica utilizada for **M9** será considerada positiva quando o percentual for maior que 70%.

5.4.4 Preparação dos dados

Antes da análise estatística, as respostas dos formulários foram analisadas visando melhorar a qualidade dos dados. Dessa forma, foram identificados certos problemas com algumas respostas, tais como:

- 3 participantes não submeteram o formulário inicial, que indicava o início da contagem do tempo, porém foi possível determinar o início da análise a partir do tempo de submissão dos outros participantes que iniciaram a avaliação simultaneamente;
- 6 esqueceram de responder o formulário sobre perfil, por isso foi enviando um *email* informando o problema e, com isso, eles preencheram tal formulário;
- 2 marcaram o grupo errado no formulário de considerações, então suas respostas foram inseridas no grupo correto;
- 1 teve problema com a submissão de pelo menos um dos formulários devido à queda na conexão com a *internet*, por isso as análises que observam a hora de submissão não consideraram as respostas desses participantes;
- 1 analisou os *logs* apenas com uma ferramenta, por isso sua resposta foi desconsiderada para qualquer análise.

Após essa etapa, foi realizada a análise estatística e o resultado pode ser visto no Capítulo 6.

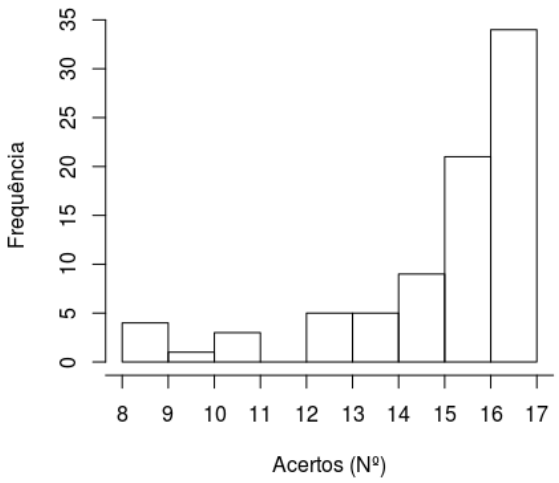
6 RESULTADOS DA AVALIAÇÃO E DISCUSSÕES

Neste capítulo, os principais resultados da avaliação são apresentados. Para facilitar a análise dos dados, foram elaborados gráficos com base nos dados obtidos durante o experimento avaliativo.

6.1 ANÁLISE DESCRITIVA DOS DADOS

Realizando uma breve análise dos dados é possível entender o comportamento das variáveis e perceber padrões e/ou inconsistências nos dados. Diante disso, nas Figuras 43 e 44 são mostrados respectivamente o histograma e o *boxplot* da variável resposta acertos. Observando essas figuras, é possível notar que as maiores frequências foram 17 e 16 acertos, não houve respostas inferiores a 8 acertos e o número de acertos igual ou inferior a 11 foi considerado *outliners*. Nas Figuras 45 e 46 são mostrados respectivamente o histograma e o *boxplot* da variável resposta **tempo**. Observando tais figuras, é possível notar que a maior frequência foi entre 5 e 10 minutos (23) e a segunda maior foi entre 10 e 15 minutos (15).

Figura 43 – Frequência da variável acertos.

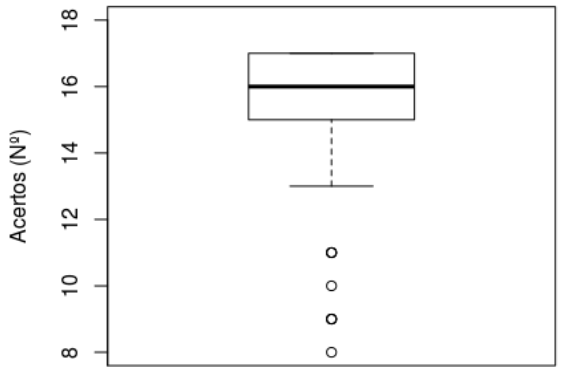


Fonte: Elaborada pelo autor.

Nas Figuras 47 e 48, é possível comparar a quantidade de acertos e o tempo gasto por cada participante na análise com cada ferramenta respectivamente. Analisando a Figura 48, é possível observar que:

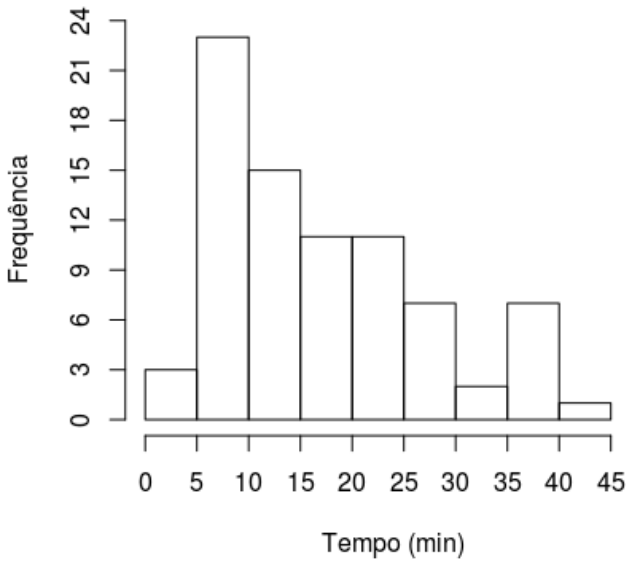
- Os participantes **P16**, **P18** e **P21** obtiveram um tempo menor utilizando o editor de planilhas do que com a ferramenta, ao contrário dos demais participantes. Esse fato deve ser consequência da experiência deles com editores de planilhas. Entretanto, tem

Figura 44 – *Boxplot* da variável acertos.



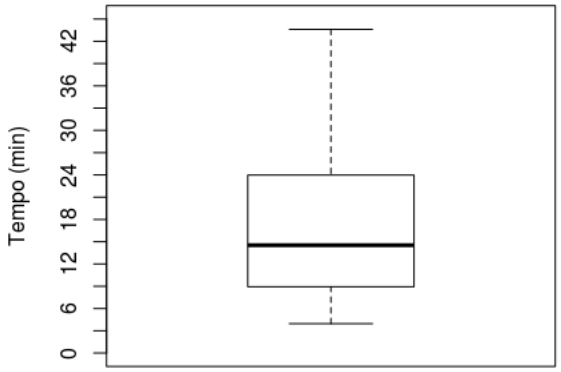
Fonte: Elaborada pelo autor.

Figura 45 – Frequência da variável tempo.



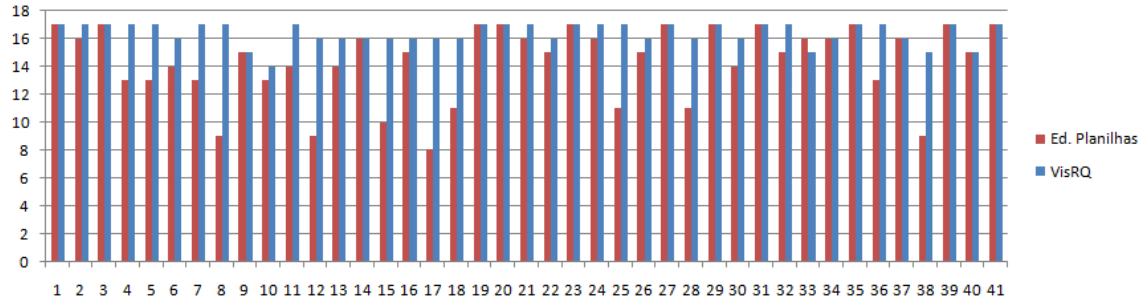
Fonte: Elaborada pelo autor.

Figura 46 – *Boxplot* da variável tempo.



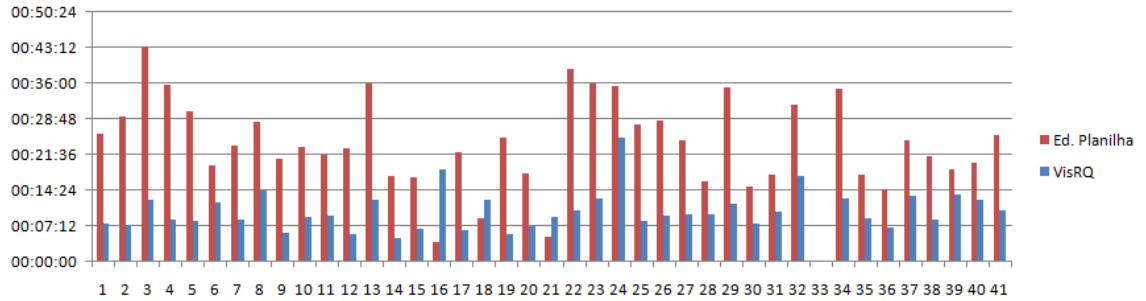
Fonte: Elaborada pelo autor.

Figura 47 – Quantidade de acertos por participante.



Fonte: Elaborada pelo autor.

Figura 48 – Tempos por participante.



Fonte: Elaborada pelo autor.

que observar que o número de *logs* utilizados na análise com editor de planilhas foi menor que o número utilizado com a ferramenta VisRQ.

- O participante **P33** não teve o tempo das análises considerado, pois teve problemas com a submissão de um dos formulários, impossibilitado que se determinasse o tempo de execução de cada processo.

Tabela 12 – Análise do número de acertos e dos tempos por ferramenta.

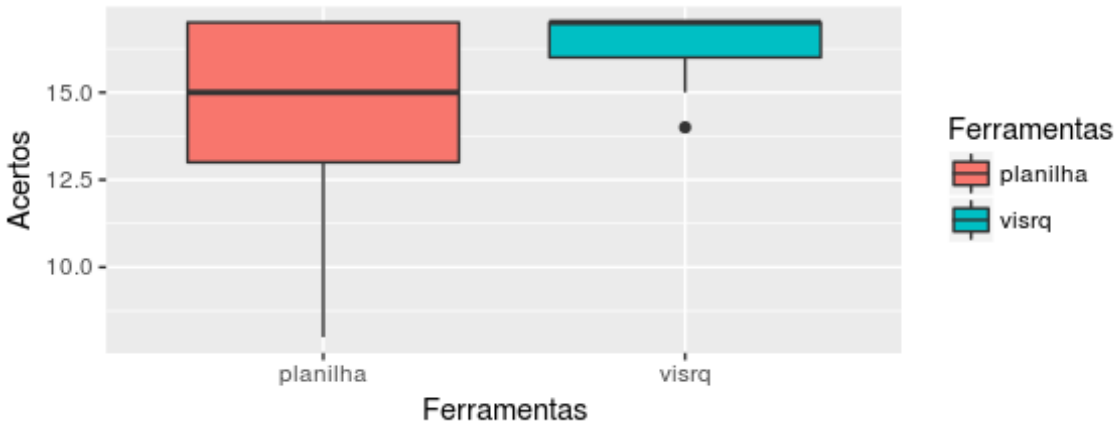
Acertos (Nº)						
Ferramenta	**Min**	**1º Q**	**Mediana**	**Média**	**3º Q**	**Máx**
Ed. Planilha	8,00	13,00	15,00	14,39	17,00	17,00
VisRQ	14,00	16,00	17,00	16,39	17,00	17,00
Tempos (minutos)						
Ferramenta	**Min**	**1º Q**	**Mediana**	**Média**	**3º Q**	**Máx**
Ed. Planilha	3,96	17,74	23,27	24,07	29,60	43,60
VisRQ	4,81	7,70	9,40	10,28	12,46	25,05

Fonte: Elaborada pelo autor.

Na Tabela 12 são mostradas as informações sobre o número de acertos e o tempo de uso com cada ferramenta, tais como: o valor mínimo e máximo, a média, a mediana, os valores do 1º e 3º quartis. E na Figura 49, que representa graficamente as informações sobre o número

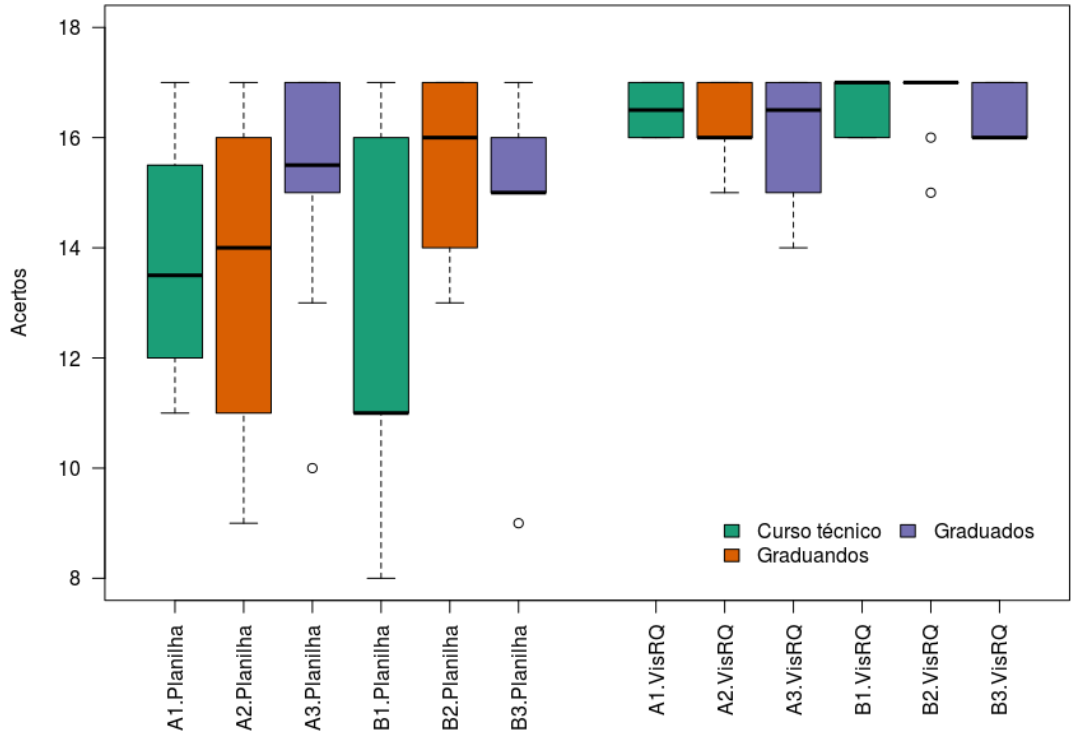
de acertos da Tabela 12, é mostrada a comparação entre o número de acertos por ferramenta. Observando a figura juntamente com a Tabela 12, é possível notar que, utilizando o editor de planilhas, a mediana foi 15 acertos e houve uma variação maior entre a quantidade de acertos, sendo que vários participantes obtiveram menos de 15 acertos. Enquanto com a ferramenta VisRQ, a variação foi menor, a mediana foi 17 acertos e apenas um dos participantes acertou menos de 15 respostas.

Figura 49 – Comparação entre os acertos por ferramenta.



Fonte: Elaborada pelo autor.

Figura 50 – Comparação entre os acertos dos grupos.

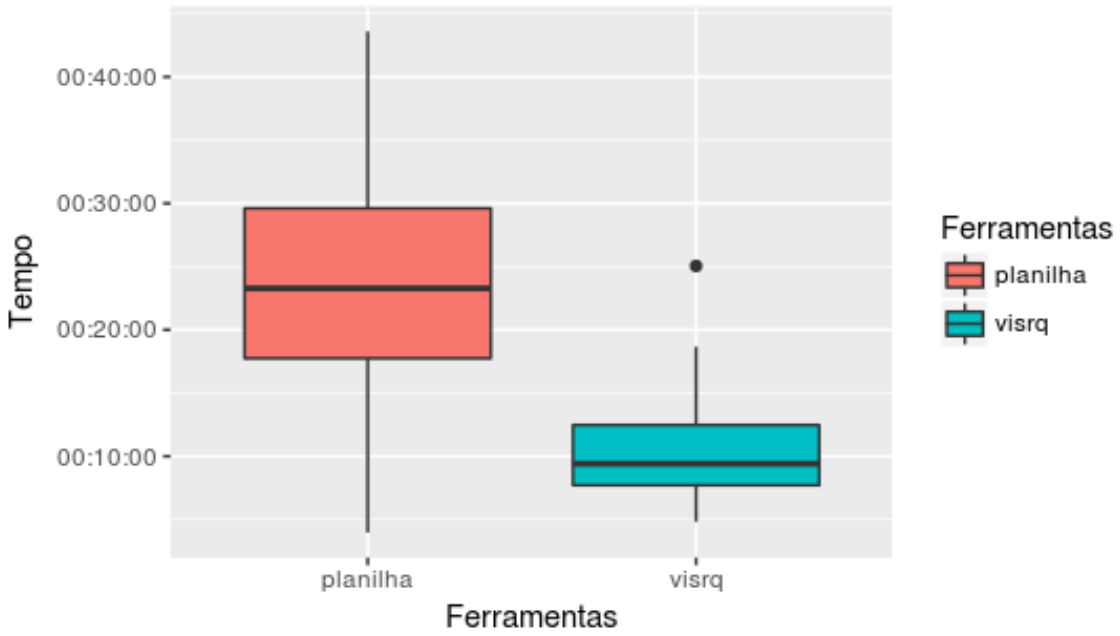


Fonte: Elaborada pelo autor.

Observando a Figura 50, é possível notar que todos os grupos obtiveram um resultado

melhor em relação a quantidade de acertos com a ferramenta VisRQ do que com o editor de planilhas. Com a ferramenta, a variabilidade foi menor, o número de acertos e, consequentemente, a média e a mediana foram maiores. Na maioria dos casos, a quantidade de acertos ficou entre 16 ou 17.

Figura 51 – Tempo por ferramenta.



Fonte: Elaborada pelo autor.

Ao observar a Figura 51, que representa graficamente as informações sobre os tempos de análise dos *logs* da Tabela 12, nota-se que o tempo de análise com a ferramenta VisRQ foi inferior. A mediana com o uso da ferramenta foi 9,4 minutos, enquanto com o editor de planilhas foi 23,27 minutos. Com a ferramenta, 75% dos participantes analisaram em menos de 12,46 minutos. Já com o editor de planilhas 75% necessitaram de mais de 17,74 minutos para realizar a análise.

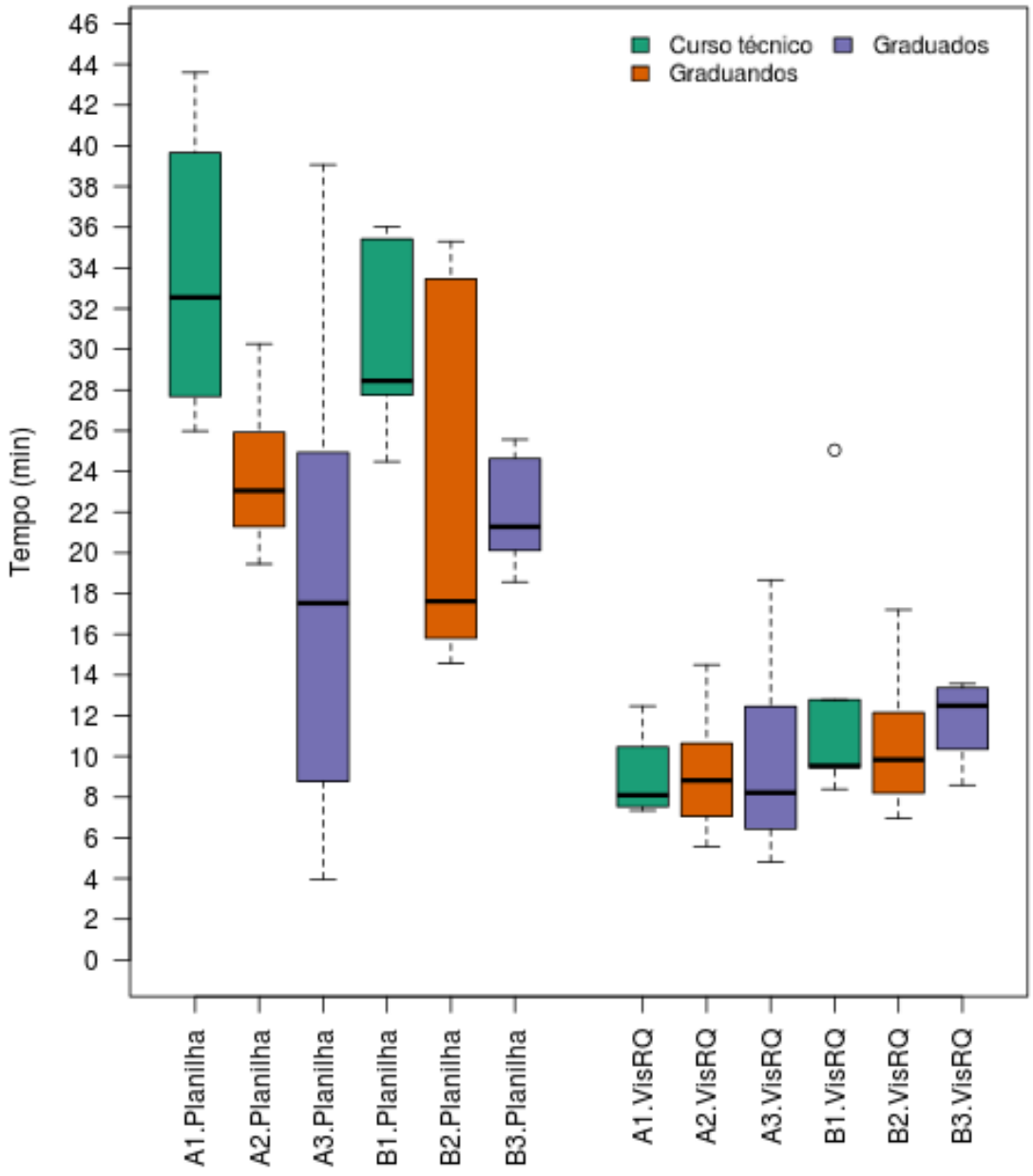
Observando a Figura 52, também é possível notar que todos os grupos obtiveram um resultado melhor com a ferramenta do que com o editor de planilhas em relação ao tempo necessário para analisar os *logs*. Com a ferramenta VisRQ, a variabilidade, o tempo gasto e, consequentemente, a média e a mediana dos grupos foram menores do que com o editor de planilhas. Na maioria dos casos, o tempo com a ferramenta ficou abaixo dos 12 minutos, enquanto com o editor de planilhas ficou bem acima desse tempo.

Com base nas respostas dos participantes, é possível fazer algumas observações, sobre a usabilidade, produtividade e corretude.

Em relação a usabilidade, foi considerado o número de erros e o número de vezes que os participantes se sentiram confusos, com base na intuição dos participantes. Esses erros podem ser: a realização incorreta de cálculos, uma contagem incorreta, a escrita incorreta de

uma fórmula, etc. Desse modo, para realizar as análises dos *logs*, a média de erros informados ao utilizar a planilha foi 4,07 e o número médio de vezes que se sentiram confusos foi 3,76 aproximadamente. Já com a ferramenta VisRQ, a média de erros foi de 1,17 e o número médio de vezes que se sentiram confusos foi 1,32 aproximadamente.

Figura 52 – Comparação entre os tempos dos grupos.



Fonte: Elaborada pelo autor.

Utilizando o editor de planilhas, o número de pessoas que cometeram apenas um erro na utilização foi 6 (14,63%) e 9 (21,95%) não cometeram erros. Entretanto, com a ferramenta VisRQ, 8 (19,51%) pessoas cometeram apenas um erro na utilização e 24 (58,54%) não cometeram erros. Com isso, o número de participantes que cometeram no máximo um erro com o editor de planilhas foi 15 (36,59%), enquanto com a ferramenta VisRQ foi 32

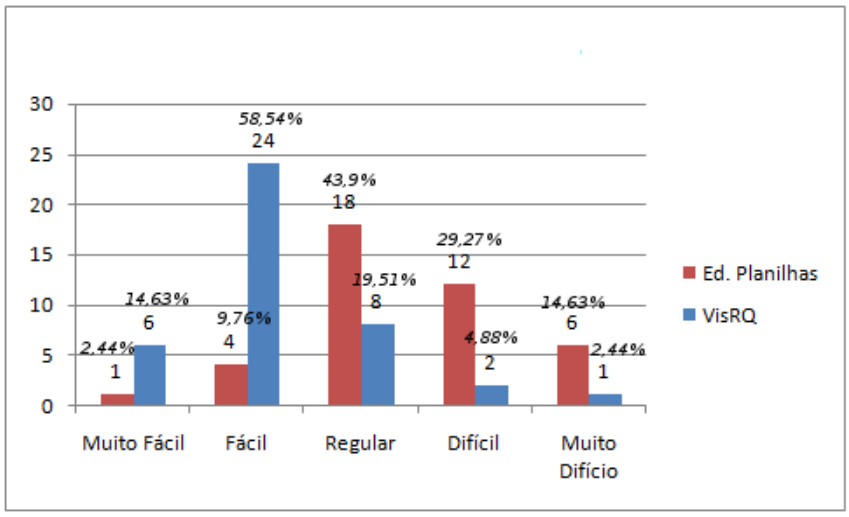
(78,05%).

Em relação a produtividade, o tempo médio de resposta com o editor de planilhas foi 24,07 minutos enquanto com a ferramenta VisRQ foi 10,28 minutos. Com isso, a porcentagem de ganho de tempo em média foi de 13,79 minutos, que corresponde a 57,3% da média do tempo com o uso do editor de planilhas. A mediana com o editor de planilhas foi 23,27 e com a ferramenta VisRQ foi 9,4 minutos. E, dos 40 participantes que tiveram o tempo de análise considerado, 37 (92,5%) melhoraram a produtividade através do uso da ferramenta enquanto apenas 3 (7,5%) pioraram.

Em relação aos acertos, com o editor de planilhas a proporção de acertos foi 0,846, enquanto com a ferramenta foi superior, sendo 0,964 aproximadamente. E com o editor de planilhas, 7 (17,07%) pessoas cometeram apenas um erro na resposta e 12 (29,27%) não cometeram erros. Já com a ferramenta VisRQ, 14 (34,15%) pessoas cometeram apenas um erro na resposta e 22 (53,66%) não cometeram erros. Dessa forma, o número de participantes que cometeram no máximo um erro com o editor de planilhas foi 19 (46,34%) e com a ferramenta foi maior, sendo 36 (87,81%).

Outra medida importante é a média de tempo para acertar uma resposta, que com o editor de planilha foi 01:38 (minutos) e com a ferramenta VisRQ foi 00:37 (minutos).

Figura 53 – Números e porcentagens sobre a facilidade de entendimento dos processos.



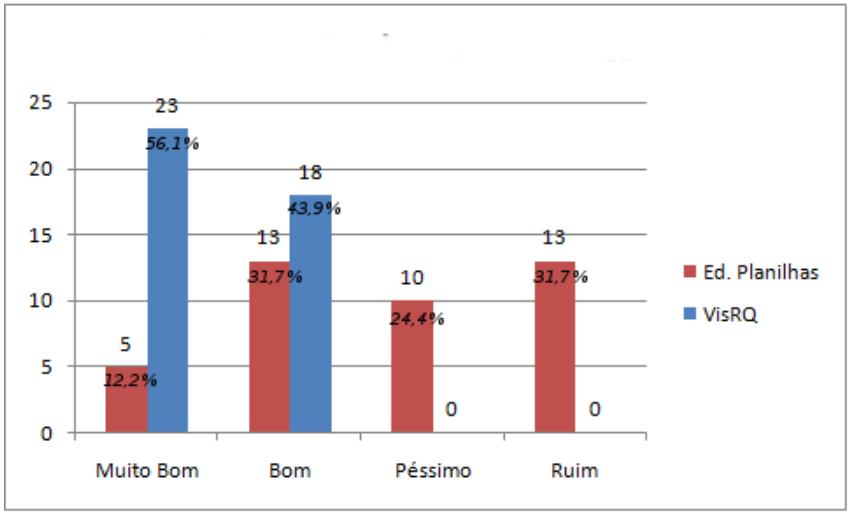
Fonte: Elaborada pelo autor.

Na Figura 53, são mostrados o número de resposta e a porcentagem sobre a facilidade de entendimento dos processos de análise dos *logs* do ponto de vista de cada participante. Nessa figura, a classificação utilizada possui o seguinte significado:

- **Muito fácil**: pode ser aprendido/entendido em menos de 3 minutos;
- **Fácil**: entre 3 e 6 minutos;

- **Regular**: entre 6 e 9 minutos;
- **Difícil**: entre 9 e 12 minutos;
- **Muito difícil**: em mais de 12 minutos.

Figura 54 – Números e porcentagens sobre a avaliação dos processos.



Fonte: Elaborada pelo autor.

Na Figura 54, são mostrados o número e a porcentagem da avaliação dos processos de análise dos *logs* por cada participante. Nessa figura, a classificação utilizada possui o seguinte significado:

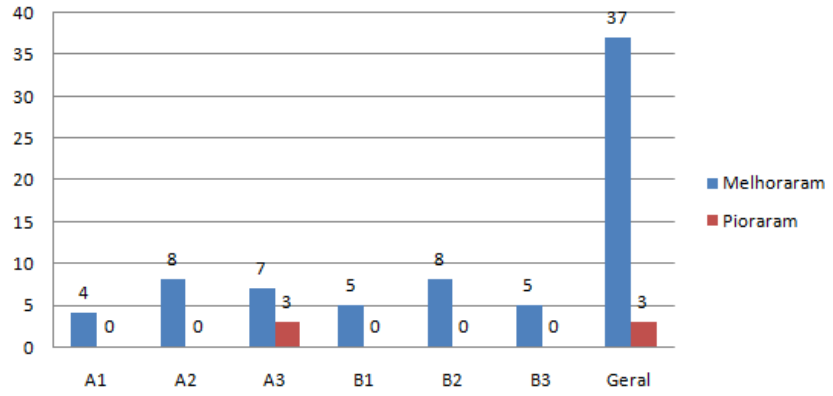
- **Muito bom**: pode ser utilizado da forma que está;
- **Bom**: necessita de ajustes pontuais;
- **Ruim**: necessita de ajustes consideráveis;
- **Péssimo**: inviável para utilização real.

Diante disso, como é possível observar nessas figuras, dos 41 participantes, apenas 5 (12,2%) avaliaram o processo de análise com o editor de planilhas como “Muito Bom” e 13 (31,71%) como “Bom”. Já com a ferramenta VisRQ, 23 (56,1%) avaliaram como “Muito Bom” e 18 (43,9%) como “Bom”. Ou seja, todos os participantes avaliaram como ou “Muito Bom” ou “Bom”.

Na Figura 55, é mostrada a quantidade de participantes que melhoraram ou não a eficiência com o uso da ferramenta VisRQ. Nela é possível notar que 3 participantes do grupo **A3** (graduados) analisaram os *logs* com o editor de planilhas com um tempo menor que a média e também sendo mais rápido do que com a ferramenta VisRQ. Provavelmente esse resultado foi causado pela experiência na utilização do editor de planilhas. Enquanto que os

outros 37 participantes gastaram menos tempo para analisar os *logs* utilizando a ferramenta VisRQ. Vale lembrar que a quantidade de *logs* utilizada com o editor de planilhas foi quase 5 vezes menor que a utilizada com a ferramenta VisRQ. Como o tempo de análise com o editor de planilhas é diretamente proporcional a quantidade de *logs*, é possível notar que realizar a análise com tal ferramenta pode ser muito trabalhosa.

Figura 55 – Nº de participantes que melhoraram a eficiência com a ferramenta VisRQ.



Fonte: Elaborada pelo autor.

No entanto, as análises realizadas e as figuras mostradas não apresentam evidências estatísticas da eficiência do uso da ferramenta VisRQ na análise dos *logs*. Por isso, somente a aplicação de um teste estatístico poderá confirmar se essa diferença de valores possui significância estatística.

6.2 ANÁLISE ESTATÍSTICA

Para escolher o teste estatístico apropriado, o usuário precisa saber: (1) classificar o tipo de dado que está estudando (contínuo, categórico: ordinal ou nominal); (2) como esses dados estão distribuídos após a coleta (distribuição normal ou não normal), e (3) os tipos de amostras examinadas (independentes ou dependentes) (NORMANDO; TJÄDERHANE; QUINTÃO, 2010).

Para verificar a normalidade dos dados, foi aplicado o teste de normalidade Shapiro-Wilk. Na Tabela 13, é mostrado o resultado do P-valor da análise de normalidade das variáveis *Acertos* e *Tempos*. Como é possível observar, os testes nos indicam que os dados das variáveis não possuem distribuição normal (p-valor < 0,05).

Como essas variáveis não apresentam uma distribuição normal deve-se aplicar o teste estatístico não-paramétrico (não-contínuo). Medidas paramétricas são contínuas, enquanto que uma medida, representada por escore, não é parâmetro para outra, portanto deve ser considerada como uma variável não-paramétrica (NORMANDO; TJÄDERHANE; QUINTÃO, 2010).

Tabela 13 – Resultados dos testes de normalidade.

Variável	**P-valor**
ACERTOS	6,301e-11
TEMPOS	0,0001018

Fonte: Elaborada pelo autor.

No caso dessa pesquisa, as amostras são dependentes pois cada indivíduo tem duas respostas: uma com e outra sem a ferramenta VisRQ.

Diante disso, visando auxiliar na escolha do teste estatístico apropriado, foi utilizado o trabalho de Normando, Tjäderhane e Quintão (2010). Este trabalho apresenta um tutorial em forma de apresentação que, através de perguntas e respostas relacionadas ao conjunto de dados, orienta a escolha do teste estatístico.

Dessa forma, foi sugerido o teste de Wilcoxon (ou *signed rank test*) para amostras pareadas devido às características do conjunto de dados em questão: com dados numéricos, distribuição não normal, dependentes (relacionadas ou pareadas) e divididos em dois grupos.

O teste de Wilcoxon é aplicado em comparações de dois grupos relacionados e que a variável deve ser de mensuração ordinal.

Tabela 14 – Resultados do teste de Wilcoxon.

Comparação	**P-valor**
ACERTOS_PLANILHA e ACERTOS_VISRQ	2,053e-05
TEMPOS_PLANILHA e TEMPOS_VISRQ	2,199e-07

Fonte: Elaborada pelo autor.

Conforme a Tabela 14, a comparação entre o número de acertos com e sem o uso da ferramenta VisRQ apresentou o P-valor = 2,053e-05 < 5%, então, ao nível de significância de 5%, existem evidências de diferença entre as duas amostras. Com isso, é possível rejeitar a hipótese nula: o número de acertos são iguais.

A comparação entre os tempos de análise apresentou o P-valor = 2,199e-07 < 5%, então, ao nível de significância de 5%, existem evidências de diferença entre as duas amostras. O resultado do teste mostrou que há uma diferença significativa entre os tempos de análise com e sem a ferramenta VisRQ. Dessa forma, é possível rejeitar a hipótese nula: os tempos de análise são iguais.

Para visualizar as diferenças entre a média dos tempos e a média do número de acertos com e sem o uso da ferramenta VisRQ, foi aplicado o método Bootstrap, que é uma técnica de reamostragem, bastante utilizada em diferentes situações estatísticas. A base da técnica consiste na obtenção de um “novo” conjunto de dados, por reamostragem do conjunto

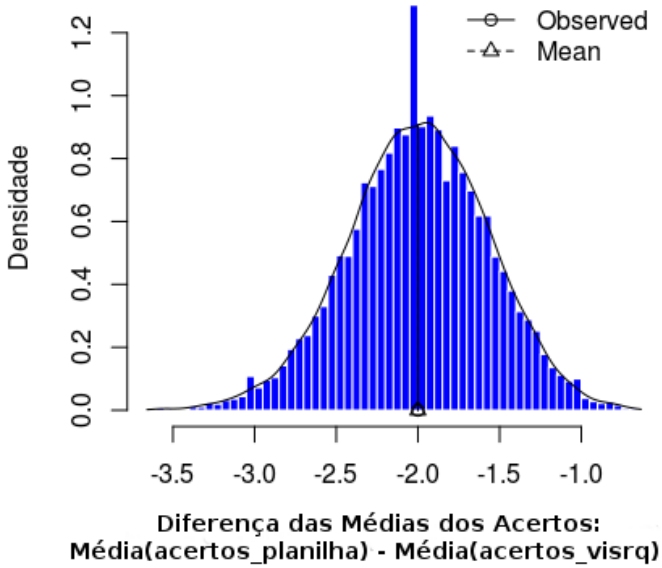
de dados original (EFRON; TIBSHIRANI, 1994) (MARTINEZ-ESPINOSA; SANDANIELO; LOUZADA-NETO, 2006).

Tabela 15 – Resultados método Bootstrap.

Variável	**2,5%**	**97,5%**
Diferença_Média(ACERTOS): Média(acertos_planilha) - Média(acertos_visrq)	-2,902439	-1,121951
Diferença_Média(TEMPOS): Média(tempo_planilha) - Média(tempo_visrq)	10,6664	16,91358

Fonte: Elaborada pelo autor.

Figura 56 – Método Bootstrap - Diferença estimada entre as médias dos acertos.

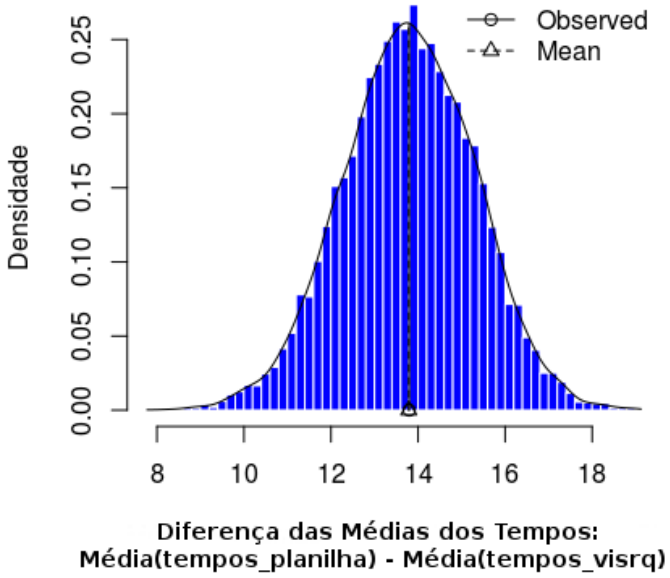


Fonte: Elaborada pelo autor.

Na Tabela 15, é mostrada a diferença entre a média de acertos e a média de tempo da análise dos *logs* com e sem a ferramenta VisRQ usando o método Bootstrap após 10000 replicações. Nas Figuras 56 e 57, são mostradas a distribuição da diferença da média dos acertos e da média dos tempos respectivamente. Como é possível visualizar, a média dos acertos com a ferramenta VisRQ é maior do que com o editor de planilhas, com a diferença da média dos acertos variando de aproximadamente -2,9 a -1,12 com intervalo de confiança de 95%. Já a média dos tempos com a ferramenta VisRQ é menor do que com o editor de planilhas e a diferença da média dos tempos variou de 10,66 minutos a 16,91 minutos com intervalo de confiança de 95%.

Observando as Figuras 56 e 57 é possível notar que alguns valores apresentaram a densidade acima da curva. Isso pode ser decorrência da presença de *outliners* nos conjuntos de dados analisados.

Figura 57 – Método Bootstrap - Diferença estimada entre as médias dos tempos.



Fonte: Elaborada pelo autor.

Assim, com 95% de confiança, é possível afirmar que houve um aumento do número de acertos e uma diminuição do tempo de análise dos *logs* com a utilização da VisRQ.

6.3 ANÁLISE GQM

6.3.1 G1: Avaliar se a ferramenta torna o processo de análise dos *logs* mais preciso e rápido (eficiente).

Q1. A ferramenta torna o processo de análise dos *logs* mais preciso?

Com a ferramenta VisRQ, a proporção, a mediana e a média dos acertos foi melhor do que utilizando o editor de planilhas, conforme mostradas na Tabela 16.

O número de participantes que cometeram no máximo um erro com o editor de planilhas foi 19 (46,34%), enquanto com a ferramenta VisRQ foi 36 (87,81%). Já os que não cometeram erros com o editor de planilhas foram 12 (29,27%), enquanto com a ferramenta VisRQ foram 22 (53,66%).

Além disso, aplicado o método de reamostragem Bootstrap com 95% de confiança, a diferença da média dos acertos com e sem o uso da ferramenta VisRQ variou de aproximadamente -2,9 a -1,12. Ou seja, a média de acertos com a ferramenta VisRQ foi, aproximadamente, de 1,12 a 2,9 maior do que com o editor de planilhas.

Q2. A ferramenta torna o processo de análise dos *logs* mais rápido (eficiente)?

Conforme é possível observar na comparação realizada na Tabela 17, a análise dos

Tabela 16 – Comparação das medidas de acertos.

Medidas	**Ed. Planilhas**	**VisRQ**
Proporção de acertos	0,846	0,964
Mediana dos acertos	15	17
Média dos acertos	14,39	16,39
Número dos que não cometeram erros	12 (29,27%)	22 (53,66%)
Número dos que cometeram apenas um erro na resposta	7 (17,07%)	14 (34,15%)
Número dos que cometeram no máximo um erro	19 (46,34%)	36 (87,80%)

Fonte: Elaborada pelo autor.

Tabela 17 – Comparação das medidas de produtividade.

Medidas	**Ed. Planilhas**	**VisRQ**
Melhoraram a produtividade	3 (7,5%)	37 (92,5%)
Mediana do tempo de resposta (minutos)	23,27	9,40
Média do tempo de resposta (minutos)	24,07	10,28
Média de tempo para cada resposta correta (minutos)	01:38	00:37

Fonte: Elaborada pelo autor.

logs com ferramenta VisRQ foi mais rápida. Com a ferramenta VisRQ, 92,5% dos participantes diminuíram o tempo de análise em relação a utilização do editor de planilhas, mesmo com a quantidade de *logs* analisados sendo mais de quatro vezes maior.

Além disso, aplicado o método de reamostragem Bootstrap com 95% de confiança, a diferença da média dos tempos com e sem o uso da ferramenta VisRQ variou de aproximadamente 10,67 a 16,91 minutos (Tabela 15). Ou seja, a média de tempo com a ferramenta VisRQ ficou entre 10,67 a 16,91 minutos a menos do que sem tal ferramenta.

6.3.2 G2: Avaliar o tempo de aprendizado da ferramenta VisRQ.

Q3. O uso das ferramentas pode ser aprendido em menos de 6 minutos?

Em relação ao tempo de aprendizagem do processo com o editor de planilhas, conforme é possível visualizar na Tabela 18, apenas 5 (12,2%) participantes responderam que poderia ser aprendido em menos de 6 minutos. Já com a ferramenta VisRQ, esse número corresponde a 30 (73,17%) participantes.

6.3.3 G3: Avaliar a aceitação da ferramenta VisRQ.

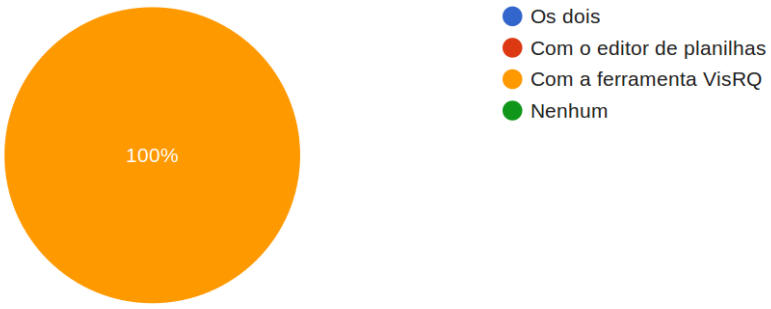
Q4. Qual processo você prefere?

Tabela 18 – Comparação das medidas de aprendizagem.

Medidas	**Ed. Planilhas**	**VisRQ**
Processo pode ser aprendido em menos de 3 minutos	1 (2,44%)	6 (14,63%)
Processo pode ser aprendido entre 3 e 6 minutos	4 (9,76%)	24 (58,54%)
Processo pode ser aprendido em menos de 6 minutos	5 (12,2%)	30 (73,17%)

Fonte: Elaborada pelo autor.

Figura 58 – Respostas sobre processo preferido.

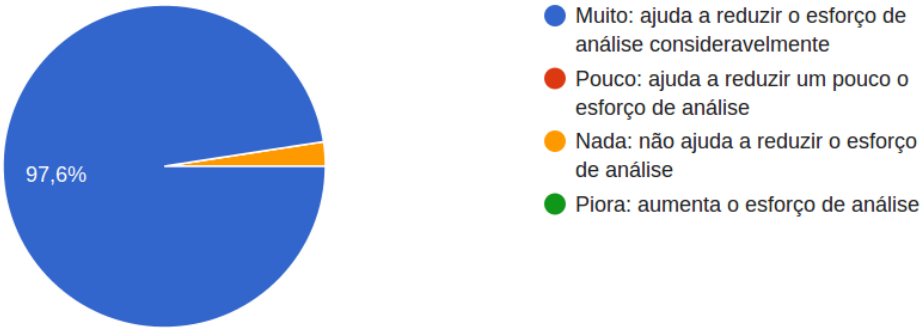


Fonte: Elaborada pelo autor.

Foi consenso entre os 41 participantes da avaliação a preferência pelo uso da ferramenta VisRQ para analisar os *logs* (Figura 58). As justificativas para tal escolha foram diversas, como: a facilidade de uso e visualização dos dados, praticidade, eficiência, por automatizar os cálculos e diminuir as chances de erros.

Q5. Você acha que a utilização da ferramenta VisRQ facilita o processo de análise?

Figura 59 – Respostas sobre facilitação da análise dos *logs* pela ferramenta VisRQ.



Fonte: Elaborada pelo autor.

Para essa questão, 97,6% dos voluntários (40 participantes) responderam que a ferramenta facilita muito, ajudando a reduzir o esforço de análise consideravelmente. Enquanto 1 participante, que corresponde a 2,4%, respondeu que não ajuda a reduzir o esforço de análise.

Q6. Como você avalia cada processo de análise dos *logs*?

Tabela 19 – Comparação das avaliações das ferramentas.

Medidas	**Ed. Planilhas**	**VisRQ**
Avaliação como “Bom”	13 (31,71%)	18 (43,9%)
Avaliação como “Muito Bom”	5 (12,2%)	23 (56,1%)
Avaliação como “Bom” ou “Muito Bom”	18 (43,91%)	41 (100%)

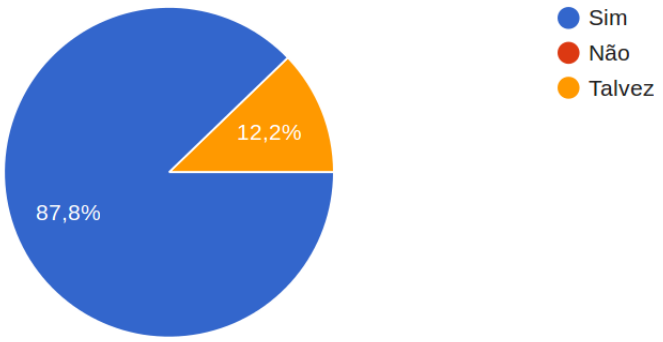
Fonte: Elaborada pelo autor.

Dos 41 participantes, 41 (100%) avaliaram o processo com a ferramenta VisRQ como “Bom” ou “Muito Bom”. Desses 23 (56,1%) avaliaram como “Muito Bom”. Já o processo com o editor de planilhas foi avaliado como “Bom” ou “Muito Bom” por 18 (43,91%) participantes. Desses apenas 5 (12,2%) avaliaram como “Muito Bom”.

6.3.4 G4: Avaliar a importância das principais funcionalidades da ferramenta VisRQ.

Q7. Importar e utilizar diagramas desenvolvidos em outras ferramentas (como DIA, Astha, etc) é uma funcionalidade útil?

Figura 60 – Respostas sobre a utilidade da importação de diagramas.



Fonte: Elaborada pelo autor.

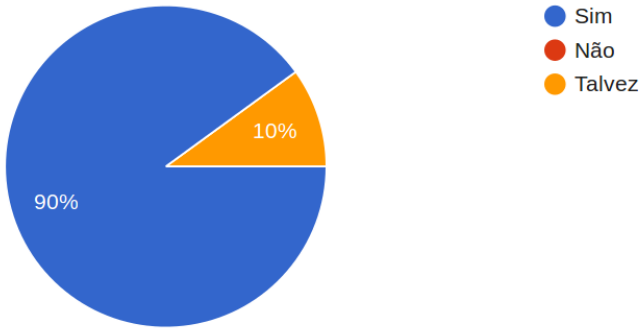
Em relação a essa pergunta, 36 participantes (87,8%) responderam que consideram essa funcionalidade útil enquanto 5 (12,2%) responderam “Talvez”.

Q8. Associar os elementos arquiteturais registrados nos *logs* com os diagramas importados facilita o entendimento?

Em relação a essa pergunta, 36 participantes (90%) concordaram que associar os *logs* com os diagramas facilita o entendimento, enquanto 4 (10%) responderam “Talvez”. Essa pergunta teve 40 respostas, pois um dos participantes esqueceu-se de respondê-la.

Q9. A visualização das informações dos *logs* de forma gráfica é útil?

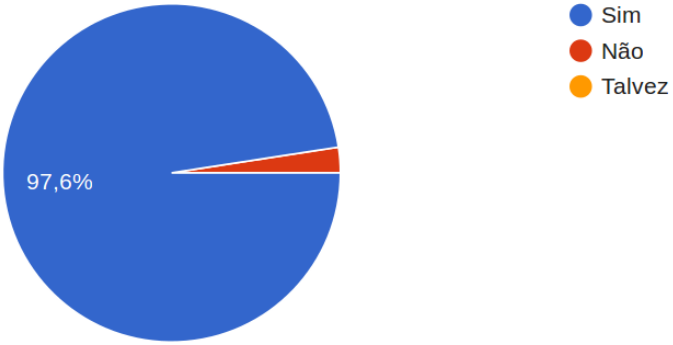
Figura 61 – Respostas sobre a associação dos *logs* com os diagramas.



Fonte: Elaborada pelo autor.

Em relação a essa pergunta, 40 participantes (97,6%) consideram a visualização das informações dos *logs* de forma gráfica útil, enquanto 1 participante (2,4%) não considera.

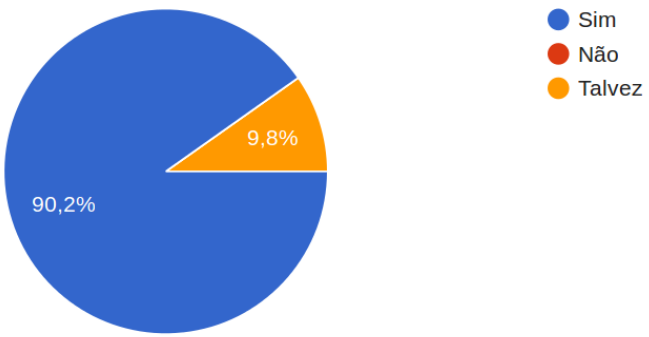
Figura 62 – Respostas sobre a visualização de forma gráfica.



Fonte: Elaborada pelo autor.

Q10. A visualização das informações dos *logs* de forma resumida é útil?

Figura 63 – Respostas sobre a utilidade da visualização resumida.



Fonte: Elaborada pelo autor.

Em relação a essa pergunta, 37 participantes (90,2%) consideram essa funcionalidade útil, enquanto 4 participantes (9,8%) responderam “Talvez”.

6.3.5 G5: Avaliar usabilidade da ferramenta VisRQ para analisar os *logs*.

Q11. A ferramenta VisRQ diminui a quantidade de erros de uso durante a análise dos *logs*?

Tabela 20 – Comparação das medidas de usabilidade.

Medidas	**Ed. Planilhas**	**VisRQ**
Média de erros na utilização	4,07	1,17
Número dos que não cometeram erros na utilização	9 (21,95%)	24 (58,54%)
Número dos que cometeram apenas um erro na utilização	6 (14,63%)	8(19,51%)
Número dos que cometeram no máximo um erro na utilização	15 (36,59%)	32 (78,05%)

Fonte: Elaborada pelo autor.

Como é possível observar na Tabela 20, com o uso da ferramenta VisRQ o número médio de erros na análise dos *logs* diminuiu de 4,07 para 1,17. Com isso, a média de erros com o uso da ferramenta VisRQ é aproximadamente 28,74% da média de erros com o uso do editor de planilhas. E o número de participantes que não cometeram erros na utilização subiu de 9 (21,95%) para 24 (58,54%). Já o número dos que cometeram no máximo um erro na utilização subiu de 15 (36,59%) para 32 (78,05%).

6.4 CONSIDERAÇÕES SOBRE OS RESULTADOS

As Tabelas 16, 17, 18 e 20 apresentam de forma resumida o resultado das análises realizadas anteriormente sobre as respostas dos usuários. Elas foram utilizadas como base para responder as perguntas elaboradas no planejamento do GQM.

Assim, no geral, é possível notar que a análise com a ferramenta VisRQ obteve um resultado melhor na avaliação do que a análise com o editor de planilhas.

7 CONCLUSÃO

De acordo com o que foi exposto nos capítulos anteriores, é possível afirmar que a questão de pesquisa que norteou este trabalho (*"Como visualizar informações sobre requisitos não-funcionais a partir dos logs de monitoramento de um sistema e de modo contextualizado com a arquitetura de software?"*) foi respondida.

Conforme foi citado, os *logs* são importantes para o processo de desenvolvimento de software uma vez que possibilita entender o funcionamento do software monitorado. Por isso, a análise dos *logs* é bastante utilizada para identificar problemas e orientar melhorias ou correções, buscando garantir as exigências cada vez maior dos governos, das empresas e da sociedade em relação à qualidade dos produtos software.

Porém, em sistemas reais, o processo de monitoração pode gerar grandes volumes de dados, podendo dificultar e até inviabilizar a avaliação da real situação do software monitorado, principalmente quando se pretende avaliar características não-funcionais do software, tais como desempenho e confiabilidade. Diante disso, as técnicas de visualização de informações permitem a conversão da informação que não pode ser percebida facilmente pelo olho humano numa forma mais adequada para as pessoas. Por isso, torna-se fundamental o uso de uma ferramenta que facilite a visualização das informações desses *logs*.

Esta dissertação apresentou uma solução para o problema apontado. Para isso, foram utilizadas técnicas de visualização de informações para ajudar desenvolvedores, arquitetos e engenheiros de software a contextualizar os dados de monitoramento de requisitos de qualidade com visões arquiteturais do software. Dessa forma, foi desenvolvida uma abordagem capaz de visualizar informações sobre os requisitos não-funcionais Desempenho e Confiabilidade a partir dos *logs* de monitoramento de um sistema e de modo contextualizado com a arquitetura de software.

Para validar essa abordagem, foi implementada uma ferramenta de visualização, chamada VisRQ. Tal ferramenta permite sintetizar as informações dos *logs*, possibilitando a listagem e a exibição de gráficos relacionados ao número de exceções e ao tempo médio de cada componente e também a filtragem dos dados por período, pacote, classe ou método.

Conforme mostrado na Capítulo 6, ao aplicar a reamostragem Bootstrap, a média dos acertos com a ferramenta VisRQ foi maior do que com o editor de planilhas, variando de 1,12 a 2,9 aproximadamente. Já a diferença da média dos tempos foi menor com a ferramenta VisRQ, variando de 10,66 a 16,91 minutos a menos.

O número dos que não cometeram erros com o editor de planilhas foi 12 (29,27%) e com a ferramenta VisRQ foi 22 (53,66%). Já o número dos que cometeram no máximo um erro com o editor de planilhas foi 19 (46,34%) e com a ferramenta VisRQ foi 36 (87,80%).

Em relação ao tempo de resposta, respectivamente, a mediana e a média foram 23,27 e 24,07 com o editor de planilhas e com a ferramenta VisRQ foram 9,4 e 10,28. Vale ressaltar também que a média de tempo para cada resposta correta foi, aproximadamente, 00:37 minutos com a ferramenta VisRQ e 01:38 minutos com o editor de planilhas.

Foi consenso entre os 41 participantes da avaliação a preferência pelo uso da ferramenta VisRQ para analisar os *logs* e 40 (97,6%) deles responderam que tal ferramenta ajuda a reduzir o esforço de análise consideravelmente.

A ferramenta VisRQ também pode facilitar o processo de entendimento do software, ao permitir o relacionamento dos *logs* com vários diagramas de determinado componente.

Além disso, foi realizada uma revisão sistemática com o objetivo de encontrar ferramentas que visualizassem requisitos não-funcionais do sistema. As ferramentas encontradas foram mostradas e comparadas com a VisRQ.

Apesar do término desta pesquisa e de um artefato funcional da ferramenta VisRQ, ainda há muito para ser estudado e aperfeiçoado. É possível afirmar que este trabalho pode servir de ponto de partida para muitos outros. Como sugestões, têm-se:

- A visualização de novas métricas relacionadas ao requisito confiabilidade;
- A expansão da ferramenta para dar suporte a métricas estáticas de software;
- A visualização de informações sobre os demais requisitos não-funcionais;
- O relacionamento dos atributos de qualidade monitorados através da utilização de métricas personalizadas, definidas pelos próprios usuários;
- A diferenciação entre os tipos de *logs*: os obtidos no ambiente de testes e os do ambiente de produção. Tal diferenciação pode ser útil para analisar o nível de interferência do ambiente na execução do sistema.

A contribuição desse trabalho está relacionada às áreas de Visualização de Dados e Engenharia de Software, já que a natureza dos dados refere-se ao monitoramento de requisitos de qualidade de software.

Assim, espera-se que a visualização de requisitos de qualidade oferecida pela ferramenta VisRQ possa servir de *feedback* para ajudar desenvolvedores, engenheiros, arquitetos de software a atenderem de maneira mais adequada como os requisitos não-funcionais do sistema estão interagindo.

REFERÊNCIAS

ALHIR, S. S. Understanding the unified modeling language (uml). **Methods and Tools**, 1999.

AVGERIOU, P.; GUELFI, N.; MEDVIDOVIC, N. Software architecture description and uml. In: SPRINGER. **International Conference on the Unified Modeling Language**. [S.l.], 2004. p. 23–32.

BALL, T.; EICK, S. G. Software visualization in the large. **Computer**, IEEE, v. 29, n. 4, p. 33–43, 1996.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. In: **Encyclopedia of Software Engineering**. [S.l.]: Wiley, 1994.

BASS, L.; CLEMENTS, P.; KAZMAN, R. Software architecture in practice. Addison-Wesley Longman Publishing Co., Inc., 2003.

BIOLCHINI, J. et al. Systematic review in software engineering. **System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES**, v. 679, n. 05, p. 45, 2005.

BOOCH, G. **The unified modeling language user guide**. [S.l.]: Pearson Education India, 2005.

BRITO, P. H. S.; FRANCO, N. M.; CORADINE, L. C. Falibras: Uma ferramenta flexível para promover acessibilidade de pessoas surdas. **TISE–Nuevas Ideas em Informatica Educativa**, v. 8, 2012.

CAMPOS, F. F. d. **Técnicas de Modelagem de Teste - Parte 1**. 2009. Https://qualidadebr.wordpress.com/tag/particao-de-equivalencia-analise-do-valor-limite-tabela-de-decisao-teste-de-transicao-de-estados-teste-de-caso-de-uso/>. Acesso em: 2016-09-17.

CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. **Readings in information visualization: using vision to think**. [S.l.]: Morgan Kaufmann, 1999.

CASERTA, P.; ZENDRA, O. Visualization of the static aspects of software: A survey. **IEEE transactions on visualization and computer graphics**, IEEE, v. 17, n. 7, p. 913–933, 2011.

CASTRO, M. J. V. d. Automação de testes baseados em cenários com uml e programação orientada a aspetos. 2013.

CATECATI, F. et al. Métodos para avaliação da usabilidade no design de produtos. **DA Pesquisa**, v. 4, p. 564–581, 2011.

CAUDWELL, A. H. Gource: visualizing software version control history. In: ACM. **Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion**. [S.l.], 2010. p. 73–74.

COLLABORATION, C.; REVIEWERS'HANDBOOK, C. et al. Version 4.2. 1. december 2003. **Australian National Health and Medical Research Council."How to review the evidence: systematic identification and review of the scientific literature**, 2000.

CORNELISSEN, B. et al. Understanding execution traces using massive sequence and circular bundle views. In: IEEE. **15th IEEE International Conference on Program Comprehension (ICPC'07)**. [S.l.], 2007. p. 49–58.

D'ARCE, F. **Avaliação da ferramenta de visualização de software SoftVisOAH como apoio à depuração de programas: um experimento controlado**. [S.l.]: Dissertação (Mestrado) – Universidade Estadual Paulista, Instituto de Biociências, Letras e Ciências Exatas, 2012.

DIAS, C. **Usabilidade na web: criando portais mais acessíveis**. [S.l.]: Alta Books, 2007.

DIEHL, S. **Software visualization: visualizing the structure, behaviour, and evolution of software**. [S.l.]: Springer Science & Business Media, 2007.

DWIVEDI, A. K.; RATH, S. K. Selecting and formalizing an architectural style: A comparative study. In: IEEE. **Contemporary Computing (IC3), 2014 Seventh International Conference on**. [S.l.], 2014. p. 364–369.

EFRON, B.; TIBSHIRANI, R. J. **An introduction to the bootstrap**. [S.l.]: CRC press, 1994.

EICK, S.; STEFFEN, J. L.; SUMNER, E. E. Seesoft-a tool for visualizing line oriented software statistics. **IEEE Transactions on Software Engineering**, IEEE, v. 18, n. 11, p. 957–968, 1992.

ESTIVALETE, P. B. **DOCUMENTAÇÃO DA ARQUITETURA DE SISTEMAS E FRAMEWORKS PARA PROCESSAMENTO E ANÁLISE DE IMAGENS: UMA ABORDAGEM BASEADA EM VISÕES DA UML E PADRÕES**. [S.l.: s.n.], 2007.

FALIBRAS. **Sobre nós**. 2016. Disponível em: <http://www.falibras.org/nossa-hist-ria>. Acesso em: 2016-07-17.

FAYYAD, U. M.; WIERSE, A.; GRINSTEIN, G. G. **Information visualization in data mining and knowledge discovery**. [S.l.]: Morgan Kaufmann, 2002.

FILHO, M. J. A. G. **Um Processo de Avaliação da Portabilidade de Unidades de Software**. 2005. Monografia (Graduação) — Universidade Federal de Pernambuco.

FREITAS, C. M. D. S. et al. Introdução à visualização de informações. **Revista de informática teórica e aplicada. Porto Alegre. Vol. 8, n. 2 (out. 2001), p. 143-158**, 2001.

GRAHAM, H.; YANG, H. Y.; BERRIGAN, R. A solar system metaphor for 3d visualisation of object oriented software metrics. In: AUSTRALIAN COMPUTER SOCIETY, INC. **Proceedings of the 2004 Australasian symposium on Information Visualisation-Volume 35**. [S.l.], 2004. p. 53–59.

GUEDES, G. T. **UML 2–Guia Prático-2ª Edição**. [S.l.]: Novatec Editora, 2014.

GUIMARãES, D. **Técnicas de teste de software**. 2016. Disponível em: <https://guimaraesdani.wordpress.com/testes-e-qualidade-de-software/tecnicas-de-teste-de-software/>. Acesso em: 2016-09-17.

HOLTEN, D. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. **IEEE Transactions on visualization and computer graphics**, IEEE, v. 12, n. 5, p. 741–748, 2006.

HOLTEN, D.; CORNELISSEN, B.; WIJK, J. J. V. Trace visualization using hierarchical edge bundles and massive sequence views. In: IEEE. **Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on**. [S.l.], 2007. p. 47–54.

HOLVITIE, J.; LEPPÄNEN, V. Illustrating software modifiability–capturing cohesion and coupling in a force-optimized graph. In: IEEE. **Computer and Information Technology (CIT), 2014 IEEE International Conference on**. [S.l.], 2014. p. 226–233.

ISO; NBR. 9126-1. associação brasileira de normas técnicas. **NBR ISO/IEC**, v. 9126, 2003.

JACOBSON, I. et al. **The unified software development process**. [S.l.]: Addison-wesley Reading, 1999.

JONES, J. A.; HARROLD, M. J.; STASKO, J. Visualization of test information to assist fault localization. In: ACM. **Proceedings of the 24th international conference on Software engineering**. [S.l.], 2002. p. 467–477.

JULIANO, R. C. Visualização de software baseada em uma metáfora do universo utilizando o conjunto de métricas ck. Dissertação (Mestrado) – Universidade Federal de Uberlândia, Uberlândia, 2014.

KEEDWELL, A.; DÉNES, J. **Latin Squares and Their Applications**. Elsevier Science, 2015. ISBN 9780444635587. Disponível em: <https://books.google.com.br/books?id=hsxLCgAAQBAJ>.

KEIM, D. A. Information visualization and visual data mining. **IEEE transactions on Visualization and Computer Graphics**, IEEE, v. 8, n. 1, p. 1–8, 2002.

KEIM, D. A.; KRIEGEL, H.-P. Visualization techniques for mining large databases: A comparison. **IEEE Transactions on knowledge and data engineering**, IEEE, v. 8, n. 6, p. 923–938, 1996.

KHAN, K. S. et al. **Undertaking systematic reviews of research on effectiveness: CRD's guidance for carrying out or commissioning reviews**. [S.l.]: NHS Centre for Reviews and Dissemination, 2001.

KOBAYASHI, K. et al. Sarf map: Visualizing software architecture from feature and layer viewpoints. In: IEEE. **2013 21st International Conference on Program Comprehension (ICPC)**. [S.l.], 2013. p. 43–52.

KOSCIANSKI, A.; SOARES, M. d. S. **Qualidade de Software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. [S.l.]: Editora Novatec, 2007.

KROGMANN, K. et al. Improved feedback for architectural performance prediction using software cartography visualizations. **Architectures for adaptive software systems**, Springer, p. 52–69, 2009.

KRUCHTEN, P. B. The 4+1 view model of architecture. **IEEE software**, IEEE, v. 12, n. 6, p. 42–50, 1995.

KÜMMEL, G. Z. **Uma Abordagem para a Criação de Arquiteturas de Referência de Domínio a partir da Comparação de Modelos Arquiteturais de Aplicações**. Tese (Doutorado) — UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2007.

LANGE, C. F. Empirical investigations in software architecture completeness. **TU Eindhoven**, 2003.

LANZA, M. **Combining metrics and graphs for object oriented reverse engineering**. Tese (Doutorado) — Citeseer, 1999.

LANZA, M. The evolution matrix: Recovering software evolution using software visualization techniques. In: ACM. **Proceedings of the 4th international workshop on principles of software evolution**. [S.l.], 2001. p. 37–42.

LANZA, M. Codecrawler-lessons learned in building a software visualization tool. In: IEEE. **Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on**. [S.l.], 2003. p. 409–418.

LANZA, M. Codecrawler-polymetric views in action. In: IEEE COMPUTER SOCIETY. **Proceedings of the 19th IEEE international conference on Automated software engineering**. [S.l.], 2004. p. 394–395.

LANZA, M. et al. Manhattan: Supporting real-time visual team activity awareness. In: IEEE. **2013 21st International Conference on Program Comprehension (ICPC)**. [S.l.], 2013. p. 207–210.

LANZA, M.; DUCASSE, S. A categorization of classes based on the visualization of their internal structure: the class blueprint. **ACM SIGPLAN Notices**, ACM, v. 36, n. 11, p. 300–311, 2001.

LANZA, M.; DUCASSE, S. Polymetric views-a lightweight visual approach to reverse engineering. **IEEE Transactions on Software Engineering**, IEEE, v. 29, n. 9, p. 782–795, 2003.

LEHMAN, M. M. Programs, life cycles, and laws of software evolution. **Proceedings of the IEEE**, IEEE, v. 68, n. 9, p. 1060–1076, 1980.

LIMA, I. C. A. d. **QuAM Framework: solução extensível para monitoramento de software orientado a aspectos e objetos**. [S.l.]: Monografia (graduação) – Universidade Federal de Alagoas, 2016.

MARCUS, A.; FENG, L.; MALETIC, J. I. Comprehension of software analysis data using 3d visualization. In: IEEE. **Program Comprehension, 2003. 11th IEEE International Workshop on**. [S.l.], 2003. p. 105–114.

MARCUS, A.; FENG, L.; MALETIC, J. I. Source viewer 3d (sv3d) a system for visualizing multi dimensional software analysis data. In: CITESEER. **Second IEEE International Workshop on Visualizing Software for Understanding and Analysis**. [S.l.], 2003. p. 62.

MARTINEZ-ESPINOSA, M.; SANDANIELO, V. L. M.; LOUZADA-NETO, F. O método de bootstrap para o estudo de dados de fadiga dos materiais. **Revista de Matemática e Estatística**, v. 2, p. 41–54, 2006.

MATTILA, A.-L. et al. Software visualization today: systematic literature review. In: ACM. **Proceedings of the 20th International Academic Mindtrek Conference**. [S.l.], 2016. p. 262–271.

MCKEE, J. R. Maintenance as a function of design. In: ACM. **Proceedings of the July 9-12, 1984, national computer conference and exposition**. [S.l.], 1984. p. 187–193.

MEDVIDOVIC, N.; TAYLOR, R. N. A classification and comparison framework for software architecture description languages. **IEEE Transactions on software engineering**, IEEE, v. 26, n. 1, p. 70–93, 2000.

MENEZES, S. K. d. O. **ArMoni: Um processo para monitoração de requisitos de qualidade de software utilizando informações arquiteturais do software**. [S.l.]: Dissertação (Mestrado) – Universidade Federal de Alagoas, 2015.

MONTEBELO, R. et al. Srat (systematic review automatic tool) uma ferramenta computacional de apoio à revisão sistemática. In: **V Experimental Software Engineering Latin American Workshop, ICMC-São Carlos**. [S.l.: s.n.], 2007.

MORAES, A. d. Design e avaliação de interface: ergodesign e interação humano-computador. **Rio de Janeiro: iUsEr**, 2002.

MUNZLINGER, E.; NARCIZO, F. B.; QUEIROZ, J. E. R. de. Sistematização de revisões bibliográficas em pesquisas da área de ihc. In: BRAZILIAN COMPUTER SOCIETY. **Companion Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems**. [S.l.], 2012. p. 51–54.

NASCIMENTO, H. A. D.; FERREIRA, C. B. R. Uma introdução à visualização de informações. **Visualidades**, v. 9, n. 2, 2011.

NORMANDO, D.; TJÄDERHANE, L.; QUINTÃO, C. C. A. A escolha do teste estatístico–um tutorial em forma de apresentação em powerpoint. **Dental Press J. Orthod**, v. 15, n. 1, p. 101–106, 2010.

OGAWA, M.; MA, K.-L. code_swarm: A design study in organic software visualization. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, v. 15, n. 6, p. 1097–1104, 2009.

OLBRICH, S. M.; CRUZES, D. S.; SJØBERG, D. I. Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems. In: IEEE. **Software Maintenance (ICSM), 2010 IEEE International Conference on**. [S.l.], 2010. p. 1–10.

OLIVEIRA, M. S. **PREVIA: Uma Abordagem para a Visualização da Evolução de Modelos de Software**. [S.l.: s.n.], 2011.

PAI, M.; MCCULLOCH, M.; COLFORD, J. **Systematic Review: A Road Map Version 2.2. Systematic Reviews Group, UC Berkeley, 2002**. 2004.

PAIVA, R. E. R. **Explorando a Combinação de Visualização de Software com Clusterização de Dados em um Processo de Reconstrução de Arquitetura**. Tese (Doutorado) — Universidade de Brasília, 2015.

PRESSMAN, R. S. Engenharia de software: uma abordagem profissional. 7ª edição. **Ed: McGraw Hill**, 2011.

PRESSMAN, S. R. **Engenharia de Software**. [S.l.]: McGraw Hill, 6th edition, 2005.

REISS, S. P.; RENIERIS, E. M. Tool demonstration: Jive and jove: Java as it happens. In: IEEE. **Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on**. [S.l.], 2005. p. 169–172.

REISS, S. P.; RENIERIS, M. Jove: Java as it happens. In: ACM. **Proceedings of the 2005 ACM symposium on Software visualization**. [S.l.], 2005. p. 115–124.

ROZA, M.; SCHRODERS, M.; WETERING, H. van de. A high performance visual profiler for games. In: ACM. **Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games**. [S.l.], 2009. p. 103–110.

RUBIRA, C. M. F.; BRITO, P. H. da S. Introduçao a análise orientada a objetos e projeto arquitetural. 2007.

SANTOS, L. R. d. **Técnicas de Teste de Software**. 2011. Disponível em: <http://www.inf.ufpr.br/lmperes/ci221/aula_tecnicas_teste_Luis_Renato.pdf>. Acesso em: 2016-09-17.

SAWANT, A. P.; BALI, N. Multivizarch: multiple graphical layouts for visualizing software architecture. **Crossroads**, ACM, v. 14, n. 3, p. 17–21, 2008.

SHAW, M.; GARLAN, D. **Software architecture: perspectives on an emerging discipline**. [S.l.]: Prentice Hall Englewood Cliffs, 1996.

SILVA, A. A. **Monitoração de Requisitos de Qualidade Baseada na Arquitetura de Software**. [S.l.]: Dissertação (Mestrado) – Universidade Federal de Alagoas, 2015.

SILVA, C. V. P. d. et al. **GQM: Goal – Question – Metric**. [S.l.: s.n.], 2009. Disponível em: <http://www.cin.ufpe.br/ scbs/metricas/seminario/GQM_texto.pdf>. Acesso em: 2016-08-02.

SILVA, C. L. **Uma Solução para Apoiar Processos de Desenvolvimento de Software Centrado na Arquitetura**. [S.l.]: Dissertação (Mestrado) – Universidade Federal de Alagoas, 2014.

SOMMERVILLE, I. **Engenharia de Software, Tradução de André Maurício de Andrade Ribeiro; Revisão técnica de Kechi Hirama**. [S.l.]: São Paulo, Addison Wesley, 2003.

SOMMERVILLE, I. Engenharia de software, 8ª edição, tradução: Selma shin shimizu melnikoff, reginaldo arakaki, edilson de andrade barbosa. **São Paulo: Pearson Addison-Wesley**, v. 22, 2007.

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Addison Wesley, São Paulo, 9th edition, 2009.

STANDARDIZATION, I. O. for; COMMISSION, I. E. **Software Engineering–Product Quality: Quality model**. [S.l.]: ISO/IEC, 2001.

STANDARDIZATION, I. O. for; COMMISSION, I. E. **Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models**. [S.l.]: ISO/IEC, 2001.

STOREY, M.-A.; BEST, C.; MICHAND, J. Shrimp views: An interactive environment for exploring java programs. In: IEEE. **Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on**. [S.l.], 2001. p. 111–112.

TERMEER, M. et al. Visual exploration of combined architectural and metric information. In: IEEE. **Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on**. [S.l.], 2005. p. 1–6.

VALIATI, E. R. d. A. Avaliação de usabilidade de técnicas de visualização de informações multidimensionais. 2008.

WARE, C. **Information Visualization: Perception for Design**. [S.l.]: Elsevier, 2013.

WEBER, T. S. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. **Relatório técnico, Instituto de Informática UFRGS**, 2002.

WETTEL, R.; LANZA, M. Visualizing software systems as cities. In: IEEE. **2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis**. [S.l.], 2007. p. 92–99.

WETTEL, R.; LANZA, M.; ROBBES, R. Software systems as cities: A controlled experiment. In: ACM. **Proceedings of the 33rd International Conference on Software Engineering**. [S.l.], 2011. p. 551–560.

WOHLIN, C. et al. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

APÊNDICE A – CÓDIGOS PARA GERAR OS *LOGS* DE DEMONSTRAÇÃO

Os códigos fonte para gerar o conjunto de *logs* que foi utilizado no vídeo demostrativo pode ser visto a seguir:

```
##############################
Autor: MARCOS JOSE
Arquivo: Calculadora.java
##############################

/**
 * Classe Calculadora
 */
package br.com.calculadora;
import com.quam.fw.annotations.Loggable;

@Loggable
public class Calculadora {
    public int som(int num1, int num2) throws Exception {
    /*Alteração para lançar exceção*/
     if (num1==-1 && num2==-1){
     throw new Exception();
     }
        return num1 + num2;
    }
    public int sub(int num1, int num2) {
        return num1 - num2;
    }
    public int div(int num1, int num2) {
        return num1 / num2;
    }
    public int mult(int num1, int num2) {
        return num1 * num2;
    }
}
```

```
################################
Autor: MARCOS JOSE
Arquivo: Main.java
################################

/**
 * Classe Main
 */
package br.com.main;
import br.com.calculadora.Calculadora;

public class Main {
public static void main(String[] args) {
//criando um objeto Calculadora
        Calculadora c = new Calculadora();

        int x = 1;
        for(int i = 0; i<5; i++ ){
            try {
                c.som(x, x);
                c.sub(x, x);
                c.div(x, x);
                c.mult(x, x);
                x++;
            } catch (Exception e) { }
        }

        try{
            c.mult(x, x);
        } catch (Exception e){};

        try{
            c.som(-1, -1);
        } catch (Exception e){};

        try{
            c.div(x, 0);
        } catch (Exception e){};
```

```
        try{
            c.div((x+1), 0);
        } catch (Exception e){};

        x++;
        System.out.println("Fim");
    }
}
```

APÊNDICE B – PLANO DE TESTE

Tabela 21 – Plano de testes - Classes Válidas.

Classes Válidas	**Entradas**
CV1. Números	-1000, -1, 0, 1, 1000
CV2. Letras e números	Quarto1
CV3. Com caracteres especiais	Guarda-chuva
CV4. Com sinais de pontuação	Quê!, Você
CV5. Com concordância verbal correta	Os alunos estão bem. A gente está bem.
CV6. Com concordância nominal correta	Maria está aborrecida.
CV7. Com pontuação correta	Ele precisa de ajuda.
CV8. Com frase na voz reflexiva	O atleta feriu-se.
CV9. Com frase na voz passiva	O atleta foi ferido pelo adversário.
CV10. Com frase na voz ativa	O atleta feriu o adversário.
CV11. Sintaticamente correto e ortografia correta	Ele precisa de ajuda.

Tabela 22 – Plano de testes - Classes Inválidas.

Classes Inválidas	**E**ntradas
CI1. Entrada vazia	
CI2. Espaço	
CI3. Em inglês	How are you?
CI4. Caracteres aleatórios	Ajsdelç
CI5. Valor null	null
CI6. Com erro de concordância verbal	Os alunos está bem. A gente estamos bem.
CI7. Com erro de concordância nominal	Maria está aborrecido.
CI8. Com erro de pontuação	Ele, precisa de ajuda. Ele precisa, de ajuda.
CI9. Com a mesma palavra mas com significados diferentes	Lave a manga. O suco de manga caiu na manga da camisa.
CI10. Sintaticamente correto e ortografia incorreta	Ele pressisa de ajuda. Ela precisa de vc.
CI11. Sintaticamente incorreto e ortografia correta	Ela precisa de
CI12. Sintaticamente incorreto e ortografia incorreta	Ela preciza de

Tabela 23 – Plano de teste - Frases retiradas da página da história do projeto Falibras.

Id	**P**alavas	**C**aracteres	**E**ntrada
1	2	9	Sobre nós
2	59	390	O projeto Falibras, desenvolvido na UFAL, com apoio do CNPq, FAPEAL e CAPES, desde agosto de 2001, sob a coordenação do Prof. Luis Cláudius Coradine, tem o intuito de auxiliar na comunicação entre ouvintes e surdos, facilitando o convívio entre os mesmos, possibilitando aos surdos sua integração em locais públicos, principalmente em escolas, garantindo seu aprendizado e sua participação.
3	22	112	O trabalho tem tido o apoio da Associação de Surdos de Alagoas (ASAL) e do Centro de Apoio à Pessoa Surda (CAS).
4	31	222	Também, vale ressaltar a participação da equipe da Escola Tavares Bastos (escola estadual de ensino fundamental, especializada em educação de pessoas surdas), sendo as duas últimas instituições ligadas ao governo estadual.
5	38	217	O projeto Falibras foi concebido, inicialmente, como um sistema que, ao captar a fala no microfone, exibe, no monitor de um computador, a tradução do que foi dito, em Libras, na forma gestual e animada, em tempo real.
6	13	81	É um sistema interativo que busca auxiliar a comunicação entre ouvintes e surdos.
7	39	277	Atualmente, tem-se buscado avanços no desenvolvimento das interfaces tradutoras do Falibras, definindo seu potencial não só como um possível intérprete virtual, mas também como uma ferramenta pedagógica, que pode contribuir para o processo de aprendizagem de surdos e ouvintes.
8	23	173	O projeto Falibras tem sido capaz de traduzir frases em língua portuguesa para Libras, adicionando adequadamente classificadores para flexões e realizando análise sintática.
9	24	176	Para isso, o Falibras combina duas concepções de interface tradutora: uma baseada em Transferência Sintática (Falibras-TS) e outra baseada em Memória de Tradução (Falibras-MT).
10	34	244	No Falibras-TS, a forma sentencial em Libras é construída utilizando Processamento de Linguagem Natural, estruturando uma seqüência de análise léxica, morfológica, sintática e de contexto, adequando a tradução para a estrutura
11	18	133	No Falibras-MT, os textos em português são interpretados a partir das estruturas morfossintáticas cadastradas na memória de tradução.
12	12	86	Essa técnica permite o registro de tabelas de equivalência de formas Português-Libras.
13	25	151	Assim, o sistema pode ser visto como objeto de aprendizagem para ajudar à aprendizagem de Libras, pelos ouvintes, e de Português, por parte dos surdos.

id	palavas	caracteres	frase
14	56	367	Como parte da inserção do projeto Falibras junto ao público alvo, desenvolveu-se uma experiência junto à Escola Estadual Tavares Bastos onde, a partir do projeto pedagógico da escola, definiu-se pela apresentação, junto a uma turma de estudantes com deficiência auditiva, de um texto (música) em Libras, na forma gestual animada, dentro do padrão do sistema Falibras.
15	26	190	Essa experiência de apresentação do sistema Falibras aos surdos, a primeira num contexto educacional, mostrou resultados positivos para a dinâmica de busca do desenvolvimento da ferramenta.
16	18	105	A comparação entre o que se está desenvolvendo e a realidade do contato na escola se mostrou fundamental.
17	101	688	Um dos principais resultados dessa interação na Escola Estadual Tavares Bastos foi constatar que o Falibras tem potencial de contribuir para: promover a aprendizagem de Libras pelos agentes envolvidos no processo de ensino-aprendizagem; promover a aprendizagem do Português pelos agentes envolvidos no processo de ensino-aprendizagem; oportunizar o acesso por parte dos surdos à informação curricular e cultural da sociedade; promover a aquisição e o desenvolvimento da linguagem, do pensamento e do indivíduo; preparar o surdo para o exercício da cidadania, estudando e trabalhando nas mesmas condições de uma pessoa sem essa diferença e facilitar a inclusão do surdo no ambiente social.
18	25	154	Atualmente, o projeto conta com parceiros importantes como o professor Orivaldo de Lira Tavares, da UFES e o professor José Mario Dr Martino, da Unicamp.
19	26	187	Nessa fase, está se trabalhando o aperfeiçoamento do tradutor a partir de técnicas de inteligência artificial e métodos estatísticos, no tratamento de ambiguidade e construção sintática.
20	21	144	Vale ressaltar que o Sistema Falibras foca principalmente no processo de tradução, abstraindo todo o processo de geração *on-line* das animações.
21	17	108	O aperfeiçoamento do processo de tradução tem como fim facilitar a evolução natural da gramática da Libras.
22	14	86	Para isso, se faz necessária uma maior integração com o público-alvo e com linguistas.
23	19	141	Alguns artigos, fruto do Projeto Falibras, foram produzidos e apresentados em congressos nacionais e internacionais, ao longo desse período.
24	2	21	Socialize-se conosco
25	8	56	Siga-nos nas redes sociais para receber nossas notícias.
26	5	32	Como contribuir com o Falibras?

id	**palavas**	**caracteres**	**frase**
27	32	75	Você pode contribuir de diversas formas com o Projeto Falibras, tais como:
28	9	67	utilizando os nossos aplicativos e reportando defeitos e sugestões;
29	10	71	fornecendo correções de traduções em nosso serviço de tradução online.
30	15	98	Em breve, teremos novidades inovadoras e sua colaboração terá um papel ainda mais fundamental! :-)