

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

ANDRÉ ALMEIDA SILVA

**Monitoração de Requisitos de Qualidade  
Baseada na Arquitetura de Software**

**Maceió  
2015**

André Almeida Silva

# Monitoração de Requisitos de Qualidade Baseada na Arquitetura de Software

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal de Alagoas.

Orientador: Prof. Dr. Patrick Henrique da  
Silva Brito

Coorientador: Prof. Dr. Balduino Fonseca  
dos Santos Neto

Maceió  
2015

**Universidade Federal de Alagoas**  
**Biblioteca Central**  
**Divisão de Tratamento Técnico**  
Bibliotecário: Roselito de Oliveira Santos

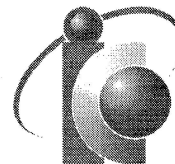
S586m Silva, André Almeida.  
Monitoração de requisitos de qualidade baseada na arquitetura de software  
/ André Almeida Silva. – Maceió, 2015.  
134 f. : il.

Orientador: Patrick Henrique da Silva Brito.  
Coorientador: Balduino Fonseca dos Santos Neto.  
Dissertação (Mestrado em Informática) – Universidade Federal de Alagoas.  
Instituto de Computação. Programa de Pós-Graduação em Informática.  
Maceió, 2015.

Bibliografia: f. 115-121.

1. Monitoração de software. 2. Arquitetura de software.  
3. Qualidade de softwares. I. Título.

CDU:004.42



Membros da Comissão Julgadora da Dissertação de Mestrado de André Almeida Silva, intitulada: “*Monitoração de Requisitos de Qualidade Baseada na Arquitetura de Software*”, apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas em 19 de fevereiro de 2015, às 10h00min, no Miniauditório do Instituto de Computação da UFAL.

### COMISSÃO JULGADORA

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

**Prof. Dr. Balduino Fonseca dos Santos Neto**  
UFAL – Instituto de Computação  
Orientador

**Prof. Dr. Marcelo Costa Oliveira**  
UFAL – Instituto de Computação  
Examinador

**Prof. Dr. Hyggo Oliveira de Almeida**  
UFCG – Universidade Federal de Campina Grande  
Examinador



*A minha família, meu porto seguro,  
pelo carinho e dedicação.*

## AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus por ter me dado, entre uma infinidade de outras dádivas, a oportunidade de vivenciar este processo que muito contribuiu para a minha formação como profissional e pessoa. A Ele que sempre esteve e está presente em minha vida, iluminando meus caminhos e dando-me forças para prosseguir.

A minha mãe Terezinha, grande exemplo de mulher de fibra, que desde sempre batalhou muito para proporcionar a mim e aos meus irmãos uma boa educação e estudo, construindo em nós valores de ética, responsabilidade e respeito. Sou infinito e eternamente grato pelo seu amor, apoio e dedicação incondicionais.

A minha querida irmã Tainá que, com carinho e paciência, esteve sempre ao meu lado durante a realização deste trabalho. Obrigado por tudo, por toda a motivação, conselhos, críticas, sugestões, leituras e revisões.

Aos meus amigos, que sempre compreenderam a importância deste Mestrado para mim. Em especial, agradeço a Nathália, Suzy e Thayse pela amizade de longa data e de valor incalculável. Muito obrigado pela irmandade, companheirismo e momentos vivenciados, esses me tornam, constantemente, uma pessoa melhor e mais forte.

Ao amigo e colega de curso, Italo Carlo. Obrigado pela parceria durante este Mestrado. Foram muitos quilômetros rodados no percurso Arapiraca-Maceió e Maceió-Arapiraca; muitas reuniões, estudos e trabalhos; e, sobretudo, muita paciência.

Aos meus orientadores, os professores doutores Patrick Henrique da Silva Brito e Baldoino Fonseca dos Santos Neto, que durante a construção desta dissertação contribuíram para o direcionamento dos meus estudos, oferecendo disponibilidade, conhecimentos e experiências que me estimularam e encorajaram-me a dar o meu melhor.

Aos professores doutores, Marcelo Costa Oliveira e Hyggo Oliveira de Almeida, por aceitarem o convite de fazer parte da banca examinadora. Muito obrigado pela atenção e pelo precioso tempo dedicado para contribuir com este trabalho. Amplio este agradecimento ao Prof. Dr. Evandro de Barros Costa, pelas importantes e valiosas contribuições durante a qualificação da proposta que originou esta dissertação, as sugestões e ideias oferecidas foram fundamentais para a sua concepção e melhoria.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo fi-

nanciamento ofertado, proporcionando melhores condições para a realização deste curso. Estendo este agradecimento ao Programa de Pós-Graduação em Informática e ao seu coordenador, o Prof. Dr. Leandro Dias da Silva, que não mediram esforços para que os recursos fossem bem direcionadas.

Aos docentes do Instituto de Computação da Universidade Federal de Alagoas, inseridos neste programa, pela dedicação às aulas ministradas e pelos ensinamentos que levarei por toda a vida, contribuindo largamente para a minha formação acadêmica e pessoal.

Finalmente, agradeço também a todos aqueles que contribuíram, direta ou indiretamente, para a realização deste curso e trabalho.

*“A verdadeira medida de um homem não se vê na forma como se comporta em momentos de conforto e conveniência, mas em como se mantém em tempos de controvérsia e desafio.”*

*“É melhor tentar e falhar, que preocupar-se e ver a vida passar. É melhor tentar, ainda que em vão que sentar-se, fazendo nada até o final. Eu prefiro na chuva caminhar, que em dias frios em casa me esconder. Prefiro ser feliz embora louco, que em conformidade viver.”*

*Martin Luther King*

## RESUMO

Os sistemas computacionais ganham dia a dia mais espaço na vida dos indivíduos, fazendo com que a demanda por soluções computadorizadas, cada vez mais sofisticadas e precisas, seja crescente. Assim, há a exigência de efetivas garantias de qualidade aos softwares produzidos, conferidas pela monitoração dos atributos de qualidade. Contudo, as principais técnicas de monitoração atuais voltam-se, sobretudo, aos sistemas baseados em serviços, deixando de lado uma grande parcela de softwares. Neste contexto, o presente trabalho possui como objetivo discutir acerca da monitoração dos atributos de qualidade referenciados pela norma ISO/IEC 9126. Serão definidas árvores de decisão, que relacionarão os elementos arquiteturais às questões de monitoração, e ainda uma ferramenta que utilizará conceitos da Programação Orientada a Aspectos para automatizar o processo de monitoração dos requisitos confiabilidade e eficiência, através da geração de aspectos-monitores destinados ao logging e registro de exceções de determinado sistema-alvo. Ainda será observada a disposição de estudo de caso estruturado pelo paradigma *Goal/Question/Metric* (GQM), realizado com a finalidade de analisar a viabilidade da solução desenvolvida que representa uma maneira simplificada para que arquitetos e desenvolvedores de softwares definam monitores para aferir atributos de qualidade em seus sistemas.

**Palavras-chave:** Atributos de qualidade. Monitoração de softwares. Programação orientada a aspectos.

## ABSTRACT

Computer systems gain more space day by day in the lives of individuals, causing the demand for computerized solutions more and more sophisticated and accurate, become increasing. Thus, there is a requirement of effective quality assurance for software produced, checked by monitoring of quality attributes. However, the main current monitoring techniques are turning mainly to service-based systems, setting aside a large number of software. In this context, this work aims to discuss about the monitoring of quality attributes referenced by ISO/IEC 9126 standard. Decision trees will be set relating to the architectural elements monitoring issues, and also a tool that uses the concepts of Aspect-Oriented Programming to automate the process of monitoring the reliability and efficiency requirements by generating aspects-monitors intended for logging and recording exceptions given target system. Still be observed the case study disposal structured by the *Goal/Question/Metric* (GQM) paradigm, conducted with the purpose of analyze the feasibility of the developed solution which is a simplified way for architects and software developers to define monitors to measure quality attributes in their systems.

**Keywords:** Quality attributes. Software monitoring. Aspect-oriented programming.

## LISTA DE ILUSTRAÇÕES

Figura 1 – Fases genéricas no processo de desenvolvimento de software . . . . .	23
Figura 2 – Modelo de qualidade de software segundo a norma ISO/IEC 9126 . . . . .	27
Figura 3 – Separação de interesses com POA . . . . .	31
Figura 4 – Classe Produto . . . . .	32
Figura 5 – Logging de auditoria e controle de exceções classe Produto . . . . .	32
Figura 6 – Composição de um Sistema Orientado a Aspectos . . . . .	33
Figura 7 – Combinação Aspectual . . . . .	34
Figura 8 – Esquema: relacionamento entre uma classe e um aspecto . . . . .	35
Figura 9 – Implementação de requisito não funcional com POO: código emaranhado	36
Figura 10 – Implementação de requisito não funcional com POA: código agrupado .	36
Figura 11 – Classe Conta: logging no método armazenar . . . . .	37
Figura 12 – Aspecto Voltado ao Logging no Sistema Bancário . . . . .	38
Figura 13 – Interesses transversais em sistemas definidos com POO (a) e POA (b) .	39
Figura 14 – Árvore de Decisão - Monitoração do Atributo Funcionalidade . . . . .	51
Figura 15 – Árvore de Decisão - Monitoração do Atributo Confiabilidade . . . . .	54
Figura 16 – Árvore de Decisão - Monitoração do Atributo Usabilidade . . . . .	57
Figura 17 – Árvore de Decisão - Monitoração do Atributo Eficiência . . . . .	58
Figura 18 – Árvore de Decisão - Monitoração do Atributo Manutenibilidade . . . . .	62
Figura 19 – Árvore de Decisão - Monitoração do Atributo Portabilidade . . . . .	66
Figura 20 – Visão Geral da Ferramenta Proposta . . . . .	68
Figura 21 – Relação dos Módulos de Implementação da Ferramenta . . . . .	70
Figura 22 – Tela Inicial da Ferramenta Proposta . . . . .	71
Figura 23 – Interface para a Criação de Aspectos . . . . .	72
Figura 24 – Aspecto-base Registro de Exceções . . . . .	73
Figura 25 – Aspecto-base Logging de Auditoria . . . . .	75
Figura 26 – Interface Monitor . . . . .	76
Figura 27 – Representação do Método GQM . . . . .	79
Figura 28 – Visão geral do processo GQM . . . . .	81
Figura 29 – Interface da Ferramenta Desenvolvida, sob o DimDimDim . . . . .	93
Figura 30 – Interface da Ferramenta Desenvolvida, sob o Mayam . . . . .	95
Figura 31 – Interface da Ferramenta Desenvolvida, sob o BancoOO . . . . .	96
Figura 32 – Interface da Ferramenta Desenvolvida, sob o BancoOA . . . . .	97
Figura 33 – Inserção de Valores no DimDimDim . . . . .	99
Figura 34 – Tela Monitor de Exceções - Observação das Exceções do DimDimDim .	100
Figura 35 – Arquivo <i>logExc.txt</i> com as Exceções Armazenadas . . . . .	100

Figura 36 – Exceções Retidas e Armazenadas pelo Aspecto-monitor da Confiabilidade no DimDimDim . . . . .	101
Figura 37 – Inserção de Imagens no Mayam . . . . .	102
Figura 38 – Exceções Retidas e Armazenadas pelo Aspecto-monitor da Confiabilidade no Mayam . . . . .	103
Figura 39 – Tela Monitor de Exceções - Observação das Exceções do BancoOO . .	104
Figura 40 – Tela Monitor de Exceções - Observação das Exceções do BancoOA . .	105
Figura 41 – Inserção de Valores para a Monitoração do DimDimDim . . . . .	107
Figura 42 – Tela Monitor de Logging - Monitoração do DimDimDim . . . . .	107
Figura 43 – Tela Monitor de Logging - Monitoração do Mayam . . . . .	108
Figura 44 – Dados Armazenados pelo Aspecto-monitor da Eficiência no Mayam . .	109
Figura 45 – Tela Monitor de Logging - Monitoração do BancoOO . . . . .	110
Figura 46 – Dados Armazenados pelo Aspecto-monitor da Eficiência no BancoOA .	111
Figura 47 – Árvore de Decisão Ampliada - Atributo Funcionalidade . . . . .	123
Figura 48 – Árvore de Decisão Ampliada - Atributo Confiabilidade . . . . .	124
Figura 49 – Árvore de Decisão Ampliada - Atributo Usabilidade . . . . .	125
Figura 50 – Árvore de Decisão Ampliada - Atributo Eficiência . . . . .	126
Figura 51 – Árvore de Decisão Ampliada - Atributo Manutenibilidade . . . . .	127
Figura 52 – Árvore de Decisão Ampliada - Atributo Portabilidade . . . . .	128



## LISTA DE TABELAS

Tabela 1 – Definição dos atributos da norma ISO/IEC 9126 . . . . .	28
Tabela 2 – Análise dos Trabalhos Relacionados . . . . .	46
Tabela 3 – Definição das medidas de confiabilidade . . . . .	53
Tabela 4 – Escala de complexidade ciclomática . . . . .	61
Tabela 5 – Escala para métricas internas . . . . .	65
Tabela 6 – Escala para métricas externas . . . . .	65
Tabela 7 – Plano GQM - Meta i - Questões . . . . .	83
Tabela 8 – Plano GQM - Meta ii - Questões . . . . .	84
Tabela 9 – Plano GQM - Meta iii - Questões . . . . .	84
Tabela 10 – Plano de Medição e Análise . . . . .	89
Tabela 11 – Questionário 1 . . . . .	129
Tabela 12 – Questionário 2 . . . . .	130
Tabela 13 – Questionário 3 . . . . .	132

## LISTA DE ABREVIATURAS E SIGLAS

AIE	Arquivos de Interface Externa
ALI	Arquivos Lógicos Internos
AO	Orientação a Aspectos
CC	Complexidade Ciclomática
CE	Consulta Externa
DICOM	<i>Digital Imaging and Communications in Medicine</i>
DSOA	Desenvolvimento de Software Orientado a Aspectos
EE	Entrada Externa
EOQC	Organização Européia para o Controle de Qualidade
GQM	<i>Goal/Question/Metric</i>
IC	Instituto de Computação
IDE	<i>Integrated Development Environment</i>
IEC	<i>International Electrotechnical Commission</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
ISO	<i>International Organization for Standardization</i>
MTBF	<i>Mean Time Between Failure</i>
MTTF	<i>Mean Time to Failure</i>
MTTR	<i>Mean Time to Repair</i>
OO	Orientação a Objetos
POA	Programação Orientada a Aspectos
POO	Programação Orientada a Objetos
QoS	<i>Quality of Service</i>
RF	Requisitos Funcionais
RNF	Requisitos Não-funcionais
RUP	<i>Rational Unified Process</i>
SBSS	Sistemas Baseados em Serviços
SE	Saída Externa
SO	Sistemas Operacionais
TC	Tomografia Computadorizada
UFAL	Universidade Federal de Alagoas
UML	<i>Unified Modeling Language</i>

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>15</b>
1.1	Contextualização	15
1.2	Motivação	16
1.3	Objetivos	18
1.3.1	Objetivo Geral	18
1.3.2	Objetivos Específicos	18
1.4	Procedimentos Metodológicos	19
1.5	Contribuições	20
1.6	Organização da Dissertação	21
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>22</b>
2.1	Arquitetura de Software	22
2.2	Requisitos Não-funcionais ou Atributos de Qualidade	25
2.3	Programação Orientada a Aspectos (POA)	29
2.3.1	Histórico	29
2.3.2	Definições	30
2.3.3	Conceitos fundamentais	33
2.3.4	Exemplo Prático	37
<b>3</b>	<b>TRABALHOS RELACIONADOS</b>	<b>40</b>
3.1	Monitoração de Atributos de Qualidade em Sistemas Baseados em Serviços	40
3.2	Logging de Aplicação com Aspectos	42
3.3	Registro de Exceções com Aspectos	44
3.4	Análise dos Trabalhos Relacionados	45
<b>4</b>	<b>MONITORAÇÃO DE REQUISITOS DE QUALIDADE</b>	<b>48</b>
4.1	Requisitos de Qualidade	48
4.2	Funcionalidade	48
4.3	Confiabilidade	51
4.4	Usabilidade	54
4.5	Eficiência	57
4.6	Manutenibilidade	59
4.7	Portabilidade	62
<b>5</b>	<b>DESCRIÇÃO DA FERRAMENTA PARA AFERIÇÃO DE ATRIBUTOS DE QUALIDADE</b>	<b>67</b>
5.1	Visão Geral do Processo	67
5.2	Visão Geral da Implementação	69

<b>5.3</b>	<b>Principais Características da Ferramenta para Aferição de Atributos de Qualidade</b> . . . . .	<b>70</b>
<b>5.4</b>	<b>Descrição dos Aspectos-base</b> . . . . .	<b>71</b>
5.4.1	Aspectos-base Registro de Exceções . . . . .	72
5.4.2	Aspectos-base Logging de Auditoria . . . . .	74
<b>5.5</b>	<b>Descrição dos Aspectos-monitores</b> . . . . .	<b>75</b>
<b>6</b>	<b>METODOLOGIA DE AVALIAÇÃO</b> . . . . .	<b>78</b>
<b>6.1</b>	<b>Classificação da Avaliação</b> . . . . .	<b>78</b>
<b>6.2</b>	<b>Desenvolvimento do Método GQM</b> . . . . .	<b>80</b>
6.2.1	Estudo Prévio . . . . .	81
6.2.2	Definição do Plano GQM . . . . .	82
6.2.3	Definição do Plano de Medição e Análise . . . . .	88
<b>7</b>	<b>AVALIAÇÃO DA SOLUÇÃO DESENVOLVIDA</b> . . . . .	<b>92</b>
<b>7.1</b>	<b>Aplicação do Método GQM - Estudo de Caso</b> . . . . .	<b>92</b>
7.1.1	Análise da Execução da Solução . . . . .	92
7.1.1.1	Execução sob a Aplicação <i>DimDimDim</i> . . . . .	92
7.1.1.2	Execução sob a Aplicação <i>Mayam</i> . . . . .	94
7.1.1.3	Execução sob as Aplicações <i>BancoOO</i> e <i>BancoOA</i> . . . . .	95
7.1.2	Análise do Aspecto-monitor da Confiabilidade . . . . .	98
7.1.2.1	Análise sob a Aplicação <i>DimDimDim</i> . . . . .	98
7.1.2.2	Análise sob a Aplicação <i>Mayam</i> . . . . .	100
7.1.2.3	Análise sob as Aplicações <i>BancoOO</i> e <i>BancoOA</i> . . . . .	103
7.1.3	Análise do Aspecto-monitor da Eficiência . . . . .	105
7.1.3.1	Análise sob a Aplicação <i>DimDimDim</i> . . . . .	106
7.1.3.2	Análise sob a Aplicação <i>Mayam</i> . . . . .	108
7.1.3.3	Análise sob as Aplicações <i>BancoOO</i> e <i>BancoOA</i> . . . . .	109
<b>7.2</b>	<b>Síntese dos Resultados</b> . . . . .	<b>110</b>
<b>8</b>	<b>CONSIDERAÇÕES FINAIS</b> . . . . .	<b>113</b>
	<b>REFERÊNCIAS</b> . . . . .	<b>115</b>
	<b>APÊNDICE A ÁRVORES DE DECISÃO AMPLIADAS</b> . . . . .	<b>122</b>
	<b>APÊNDICE B QUESTIONÁRIOS PARA MENSURAÇÃO DAS MEDIDAS</b> . . . . .	<b>129</b>

# 1 INTRODUÇÃO

## 1.1 Contextualização

Nas décadas de 1950 e 1960 os softwares eram demasiadamente simples, com grande parte da sua implementação realizada de forma artesanal, regulada por habilidades individuais e desprovida de uma abordagem sistemática específica (SOMMERVILLE, 2011). Porém, com o aumento da utilização de sistemas e o consequente incremento de complexidade, eclodiu a “crise do software” que, em linhas gerais, expressava as dificuldades do desenvolvimento ante a demanda crescente de aplicações cada vez maiores e mais complexas.

Neste contexto, no ano de 1968 a Conferência da OTAN sobre Engenharia de Software (*NATO Software Engineering Conference*) realizada em Garmisch, Alemanha, discutiu a crise, focando-se na busca de novas técnicas que apoiassem e facilitassem a organização dos processos de software. Na referida conferência surgiu e estruturou-se a disciplina de Engenharia de Software que objetiva o estabelecimento de boas práticas para processo de desenvolvimento de sistemas, aumentando a qualidade dos produtos oferecidos, diminuindo os custos e riscos e criando processos repetíveis e eficazes que podem ser utilizados em diversos ciclos de desenvolvimento de sistemas (JR, 2010).

Com o passar dos anos, diversos métodos e tecnologias sugeriram, como é o caso do paradigma procedimental e mais adiante da programação orientada a objetos (POO), do desenvolvimento baseado em componentes e de uma série de outras técnicas que estão vinculadas à identificação e proposição de soluções de problemas ligados à criação de aplicações. Nota-se que, em termos de implementação, a POO é uma das tecnologias mais utilizadas e oferece um modelo de objeto capaz de realizar a abstração precisa dos problemas de certo domínio real (KICZALES et al., 1997). Embora, quando se estão em questão os requisitos funcionais (RF), a POO ofereça mecanismos para reduzir o acoplamento e aumentar a coesão entre os módulos, não há a disposição de técnicas igualmente eficazes para lidar com interesses transversais ou requisitos não-funcionais (RNF), gerando problemas que são resultado da má modularização das aplicações e que resultam em um programa de difícil compreensão, desenvolvimento e manutenibilidade; afetando ainda, muitas vezes, na sua operabilidade.

Para preencher as brechas deixadas pela POO, diversos estudos abordam como solução a separação de interesses (CHUNG; NIXON, 1995; RASHID et al., 2002; TSANG; CLARKE; BANIASSAD, 2004). Esses trabalhos envolvem apenas a proposição de diretrizes voltadas à separação e manipulação distinta entre RF e RNF, não dispondo de um ferramental próprio que mostre suas aplicabilidades e benefícios junto aos sistemas e programadores. Tecnologias mais abrangentes são sublinhadas em Ossher e Tarr (1999)

e Kiczales et al. (1997) que versam, respectivamente, a nível de implementação, sobre a Programação Orientada a Assunto (*Subject-Oriented Programming*) e a Programação Orientada a Aspectos (*Aspect-Oriented Programming*).

Neste cenário observa-se o Desenvolvimento de Software Orientado a Aspectos (DSOA), visto como otimizador da modularização no desenvolvimento de softwares, reduzindo problemas, como o espalhamento (*scattering*) e o entrelaçamento (*tangling*) de código, tornando a estrutura do software mais legível e de mais fácil manutenção (KICZALES et al., 1997). Ademais, diversos interesses multi-dimensionais (*crosscutting concerns*), como segurança, persistência, distribuição e tratamento de exceções, podem ser implementados a partir do DSOA, através de uma unidade modular denominada de aspecto, responsável por encapsular a implementação de requisitos que, no desenvolvimento orientado a objetos, estariam distribuídos por diversos módulos do sistema, afetando, inclusive, sua eficiência (GARCIA et al., 2004).

## 1.2 Motivação

Os requisitos não-funcionais possuem importância fundamental sobre as decisões de projeto, tanto que a preocupação com eles deve ser expandida para todo o ciclo de vida da aplicação a ser implementada (CHUNG; NIXON, 1995; CHUNG; LEITE, 2009; CYSNEIROS, 2001). Contudo, o tratamento desses não é algo trivial, pois a implementação e monitoração de atributos de qualidade podem causar impactos no funcionamento e operabilidade do sistema, a partir da inserção de funções adicionais; necessidade de um desenvolvimento mais rigoroso e documentação mais detalhada; além de possíveis conflitos entre distintos RNF e entre esses e RF (BOMBONATTI, 2010; BERG; CONEJERO; CHITCHYAN, 2005).

A não observância dos RNF ou atributos de qualidade resulta em grande risco aos desenvolvedores e seus sistemas, pois, caso esses requisitos não sejam tratados de forma adequada, há uma grande probabilidade de insucesso no desenvolvimento, não atendendo às necessidades dos clientes (PENG; WANG; WANG, 2012; ROYCHOWDHURY; PEDRYCZ, 2001). Neste cenário, nota-se que os RF sempre recebem muita atenção, constatada nos vários métodos de análise e projeto da Engenharia de Software, como o Cascata, Interativo, o *Rational Unified Process* (RUP), *Unified Modeling Language* (UML), dentre outros. Entretanto, em contraste, os RNF são, em muitos casos, deliberadamente ignorados (ILLA; FRANCH; PASTOR, 2000). Assim, uma série de sistemas complexos, que tiveram altos investimentos, são acometidos de falhas, ocasionadas pela má gestão e até mesmo o total descaso com os atributos de qualidade.

Dijkstra (1997), na década de 1970, foi o primeiro a falar da importância em representar os diferentes interesses ou funcionalidades de um sistema em módulos distintos, tendo em vista que o aumento da complexidade era e é totalmente proporcional às questões de ilegibilidade de código, prejudicando o desenvolvimento e a funcionalidade dos sistemas.

O referido autor salienta que o foco do desenvolvimento não deve ser concentrado nos RF, devendo haver um cuidado compartilhado entre os diferentes requisitos, pois existirá um auxílio mútuo entre eles.

Assim, a modularização, isto é, a separação de interesses em pacotes específicos possui grande importância no processo de criação de um software, devendo ser idealizado pela construção de seu modelo arquitetural. De fato, para haver uma melhor compreensão e evitar o espalhamento e entrelaçamento, o software não deve ser monolítico, sem fronteiras claras que definam suas funcionalidades. Num cenário ideal, cada requisito, seja ele uma funcionalidade ou atributo de qualidade, tem de ser analisado, estruturado por meio de uma arquitetura adequada e implementado de forma que interfira o mínimo possível em outros módulos e/ou funcionalidades.

Conforme destacado, frequentemente os atributos de qualidade são negligenciados durante o processo de desenvolvimento de um sistema, já que a preocupação é direcionada às funcionalidades. No entanto, tratar desses atributos é fundamental, pois garantirão qualidade ao sistema produzido.

No contexto de sistemas baseados na Web, diversos trabalhos, como Wetzstein et al. (2009), Artaiam e Senivongse (2008), Michlmayr et al. (2009) e Moser, Rosenberg e Dustdar (2008) destacam a monitoração de RNF como um dos grandes indicadores da Qualidade de Serviço (QoS, em inglês, *Quality of Service*). Eles buscam estabelecer técnicas e frameworks para a efetiva monitoração dos mencionados sistemas. Seguindo a mesma linha de pesquisa, trabalhos como Muller et al. (2012), Haiteng, Zhiqing e Hong (2011), Souza et al. (2011) e Godse, Bellur e Sonar (2010), ressaltam a importância que a aferição de atributos de qualidade possui para os processos de negócio em serviços Web, salientando a necessidade da criação de ferramentas que suportem a detecção de violações, fornecendo, de forma amigável ao usuário, explicações sobre os impactos provenientes dessas violações. No entanto, estes trabalhos focam-se apenas em aplicações baseadas na Web.

Partindo para a exploração de requisitos não-funcionais em aplicações independentes da Web, são vistos os trabalhos Rashid et al. (2002), Tsang, Clarke e Baniassad (2004) que definem a separação de interesses como solução à verificação dos atributos de qualidade. Outras abordagens também merecem destaque, é o caso de Fenton e Pfleeger (1998) e Lyu (1996), que se propõem a determinar o grau de conformidade dos sistemas produzidos com RNF previamente estabelecidos, e também de Chung e Nixon (1995), Chung e Leite (2009), Cysneiros (2001) e Cysneiros e Leite (1999) que apresentam frameworks destinados, respectivamente, à elicitación de uma diversidade de atributos de qualidade e ao tratamento de possíveis conflitos existentes entre RF e RNF.

Porém, os trabalhos imediatamente destacados não representam fielmente o retrato atual do estudo da aferição de atributos de qualidade em sistemas independentes da Web, constatação fundamentada através da observação dos anos de publicação. Assim,

abre-se um nicho de pesquisa interessante e importante a ser explorado, por meio do estudo da monitoração da qualidade de sistemas de software, tomando como base suas arquiteturas que representam um nível de abstração adequado para a aferição de requisitos de qualidade, como os definidos na norma ISO/IEC 9126.

### 1.3 Objetivos

#### 1.3.1 Objetivo Geral

Este trabalho possui como principal objetivo o estudo detalhado dos atributos de qualidade definidos na norma ISO/IEC 9126 (funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade), bem como das questões relativas as suas monitorações, tendo em vista que a partir da aferição dos mencionados atributos é possível garantir qualidade ao sistema, observar a precisão dos requisitos funcionais, alocar os recursos necessários à execução das funcionalidades, corrigir eventuais falhas e trazer maior segurança ao sistema de software, impedindo que futuros erros comprometam todo o trabalho realizado.

O estudo especificado considera a criação de árvores de decisão que relacionarão os atributos de qualidade às características de monitoração e ainda a disposição de uma ferramenta voltada a amparar os desenvolvedores na aferição da qualidade dos softwares produzidos. A ferramenta desenvolvida contará com o auxílio da Programação Orientada a Aspectos (POA)<sup>1</sup> e automatizará a criação de aspectos destinados ao processo de monitoração dos atributos confiabilidade e eficiência, através de técnicas efetivadas pelo *logging* das atividades de determinados elementos arquiteturais previamente escolhidos em um sistema-alvo.

#### 1.3.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Estudar detalhadamente os atributos de qualidade referenciados na norma ISO/IEC 9126, focando-se na forma como esses podem ser aferidos em sistemas de softwares. Tomar-se-á como base a arquitetura das aplicações, buscando-se definir, para cada atributo de qualidade, qual ou quais os elementos arquiteturais a serem monitorados, que medição deverá ser realizada e como realizar esta monitoração.
2. Criar árvores de decisão, baseando-se no estudo destacado no item anterior. Essas farão a relação entre os atributos de qualidade e as variabilidades de monitoração,

---

<sup>1</sup> Destaca-se que a POA foi escolhida como técnica para a monitoração por possibilitar a criação de uma solução adaptável, reduzindo impactos na operabilidade do sistema, espalhamento e entrelaçamento, além de prover baixo acoplamento entre o código-fonte original da aplicação e o código referente à citada monitoração.



ou seja, trará de forma expressa as questões de monitoração pertinentes a cada um dos supracitados requisitos; criando, assim, uma espécie de linha de produtos de software, ou melhor, linha de produtos de monitoração de software.

3. Construir uma aplicação responsável por auxiliar desenvolvedores de software em questões relacionadas à monitoração de requisitos de qualidade, inicialmente a confiabilidade e a eficiência, através da geração e análise de logging de auditoria. Por meio dessa ferramenta, será possível criar de forma automatizada aspectos destinados à monitoração. Logo, o programador não necessitará de amplos conhecimentos em POA para usufruir dos benefícios relacionados ao tratamento de interesses transversais que essa tecnologia proporciona.
4. Implementar um módulo que retorne os elementos arquiteturais de um sistema. Este método será inserido na ferramenta sugerida nesta proposta e é responsável pela busca e exibição, ao desenvolvedor, de uma lista com os elementos comportados pelo software-alvo. A partir dessa lista será possível determinar os pontos de ação nos quais a monitoração deverá ser realizada.
5. Desenvolver mecanismos responsáveis pelo logging de aplicação e registro de exceções. Essa tarefa, composta por procedimentos inseridos na ferramenta aqui proposta, consiste na criação de métodos-padrão responsáveis pelos citados logging e registro. Dessa forma, o desenvolvedor apenas indicará quais os elementos arquiteturais que deverão ser monitorados durante a execução do programa e, através dos métodos implementados em POA, os aspectos destinados à monitoração serão automaticamente criados.

Observando a enumeração disposta, sintetiza-se que se prevê a difusão de diagramas de influência que correlacionam a arquitetura de software a questões de monitoração de atributos de qualidade, e ainda a oferta de uma ferramenta multiplataforma, voltado aos desenvolvedores Java, que facilitará a monitoração da confiabilidade e eficiência, conferindo qualidade aos sistemas, registrando e analisando as ações e eventuais falhas ocorridas nesses.

#### 1.4 Procedimentos Metodológicos

A avaliação da solução proposta será realizada com o fim de analisar a sua viabilidade, através da utilização do paradigma *Goal/Question/Metric* (GQM) que representa uma abordagem orientada a metas, destinada à mensuração de produtos e processos de software, apoiando a definição de medidas que estão alinhadas aos objetivos de negócio e toma como pressuposto o entendimento de que a realização adequada de medições de produtos e/ou processos é fruto da elaboração prévia das metas de seus respectivos projetos (ROCHA; SOUZA; BARCELLOS, 2012).

Em síntese, a aplicação do paradigma GQM na avaliação da ferramenta aferidora da confiabilidade e eficiência será idealizada em duas partes: uma voltada à composição do Plano GQM (objetivos, questões e medidas), considerando as características do software implementado, e outra referente à produção do Plano de Medição e Análise que conterá todas as informações sobre como, em que momento e quem realizará as medições durante os processos de análise.

Além disso, ressalta-se que a solução apresentada será avaliada através da sua utilização, em conjunto, com quatro aplicações desenvolvidas na linguagem de programação Java: DimDimDim, caracterizado como um software de pequeno porte; Mayam, visto como sistema de porte médio-grande; e duas aplicações do projeto BancoUnifimes (BancoOO, implementado através do paradigma de orientação a objetos e BancoOA, implementado pelo paradigma da orientação a aspectos).

Logo, haverá a avaliação da execução da ferramenta desenvolvida quanto: à geração dos aspectos-monitores, à observação do funcionamento do aspecto-monitor da confiabilidade e à observação do funcionamento do aspecto-monitor da eficiência, considerando as operações em si e as observações advindas dos questionários criados para facilitar e simplificar a coleta de dados referentes à aplicação do método GQM.

## 1.5 Contribuições

As contribuições deste trabalho visam monitorar a qualidade de sistemas de software, tomando como base a arquitetura e conferindo os atributos de qualidade definidos na norma ISO/IEC 9126. Em síntese, são apresentadas duas efetivas contribuições que estão relacionadas à aferição de atributos de qualidade, situando-se no contexto da Engenharia de Software.

A primeira contribuição corresponde à oferta de uma maneira simplificada para que os desenvolvedores definam monitores para mensurar atributos de qualidade em suas aplicações, encontrando respostas claras para questões referentes ao elemento arquitetural alvo da monitoração, qual medida a ser realizada e ainda qual técnica utilizar para efetivar essa monitoração. As referidas respostas serão concebidas mediante a análise de uma árvore de decisão ou diagrama de influência que detalhará de maneira objetiva a relação entre o atributo de qualidade e as formas de monitoração, isto é, onde, o que e como monitorar determinado atributo, baseando-se na arquitetura do sistema.

A segunda contribuição refere-se à disposição de um ferramenta que automatizará a criação de aspectos destinados a amparar a monitoração da confiabilidade e eficiência de sistemas desenvolvidos na linguagem de programação Java. Este amparo será proporcionado pelo desenvolvimento de um mecanismo de segurança, dirigido ao processamento, geração e armazenamento de informações sobre as funções executadas nos mencionados sistemas; e pelo registro de especificações relativas ao comportamento de sistemas em

relação ao tempo, observando os tempos de resposta e/ou de processamento de seus métodos.

Assim, a partir do conhecimento adquirido neste trabalho, presume-se a concepção de arquiteturas de software mais concisas, com a monitoração ocupando seus módulos específicos; ganho de tempo nas tarefas de projetistas e desenvolvedores, culminando em um sistema mais bem estruturado e com cada funcionalidade em seu devido módulo; impactos na operabilidade do sistema atenuados por haver um controle das funções adicionais inseridas; e ainda a monitoração automatizada dos atributos de qualidade.

## 1.6 Organização da Dissertação

O restante deste documento está organizado como segue. O Capítulo 2 apresenta alguns conhecimentos básicos que estruturam o entendimento do teor deste trabalho. O Capítulo 3 apresenta os principais trabalhos relacionados ao tema proposto, utilizados para a avaliação do estado da arte e ainda para fundamentar e orientar as atividades desta dissertação. O Capítulo 4 apresenta um estudo sistemático dos atributos de qualidade definidos na norma ISO/IEC 9126, bem como a composição das árvores de decisão mencionadas anteriormente. O Capítulo 5 apresenta disposições acerca da ferramenta desenvolvida neste trabalho, destinada à monitoração dos atributos confiabilidade e eficiência. O Capítulo 6 apresenta a metodologia utilizada para a avaliação da aplicação discutida no capítulo anterior. O Capítulo 7 apresenta a avaliação referente à ferramenta implementada, destacando o estudo de caso realizado, a aplicação do método GQM e os resultados alcançados. O Capítulo 8 apresenta as considerações finais deste trabalho.

## 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta alguns conhecimentos básicos que estruturam o entendimento do teor deste trabalho. Será realizada uma breve explanação acerca da arquitetura de software, requisitos não-funcionais e programação orientada a aspectos.

### 2.1 Arquitetura de Software

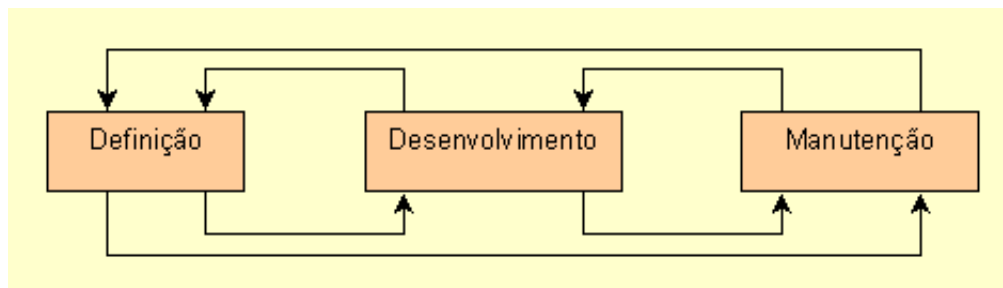
Há algumas décadas, os softwares representavam um pequeno agrupamento de elementos do universo dos sistemas computacionais. Nesse cenário, os custos relativos ao desenvolvimento e à manutenção de aplicações eram baixos quando comparados ao valor do hardware. O tempo passou, as tecnologias evoluíram e estão em constante aperfeiçoamento e os softwares constituem, atualmente, uma representativa parcela da gama dos sistemas computacionais existentes, com altos e crescentes custos. Percebe-se ainda que as inovações tecnológicas impulsionaram e impulsionam a adoção de softwares em diversas áreas, seja para uso doméstico, empresarial ou industrial.

Durante o processo de construção de softwares, sobretudo aqueles maiores e mais complexos, torna-se fundamental o estudo detalhado de todas as funcionalidades a serem implementadas, bem como de todos os artefatos que serão necessários para a sua efetivação. Este estudo irá amparar as decisões relativas ao software e seus componentes em si e ao hardware adequado para o funcionamento preciso do sistema. Todas as decisões a serem tomadas contarão, em um primeiro momento, com análises das funcionalidades pretendidas para a aplicação proposta e ainda dos atributos de qualidade requeridos por ela.

Entretanto, as referidas análises não são tarefas fáceis de serem realizadas, elas envolvem uma série de fatores e, em certas ocasiões, dificultam que as decisões de projeto sejam tomadas, resultado de pressões do mercado, demasiadas exigências dos clientes, desnecessário ou deficiente emprego de algumas tecnologias, entre outras questões. Filho (2009), infere que o processo de desenvolvimento de sistemas de software compreende três fases genéricas inter-relacionais - definição, desenvolvimento e manutenção - denotando uma espécie de arquitetura básica. A Figura 1 ilustra este processo.

Na fase de definição ocorre a identificação de informações ou funcionalidades propostas ao programa que será desenvolvido, são destacadas as funções e desempenho desejados, além da interface a ser utilizada, perfil dos usuários do sistema e uma série de outros fatores. A fase de desenvolvimento atenta-se ao projeto de estruturas de dados e arquitetura de software do sistema, ou seja, à divisão das funções propostas à aplicação em subsistemas ou módulos bem definidos e ainda nos mecanismos de interação entre esses módulos; nessa fase há a conversão do projeto (linguagem natural) para um programa

Figura 1 – Fases genéricas no processo de desenvolvimento de software



Fonte: FILHO, A. M. S. (2009)

(linguagem de programação), compreendendo ainda a realização de testes. Por fim, a fase definida como manutenção será responsável por considerar as modificações e futuras correções necessárias no sistema desenvolvido, objetivando o constante atendimento aos requisitos exigidos pelos usuários. Neste contexto, as técnicas oriundas da Engenharia de Software são vitais para o adequado desenvolvimento de sistemas.

Conforme descrito, a fase destinada ao desenvolvimento do programa considera o projeto de arquitetura de software do sistema, desenvolvido na fase anterior, para, então, implementar as funcionalidades previstas. Garlan e Perry (1994), membros do *Software Engineering Institute* da *Carnegie Mellon University*, apontam que um aspecto crítico do projeto de qualquer sistema de software de grande porte é a sua estrutura bruta, sua arquitetura, isto é, a organização de alto nível de seus elementos computacionais e das suas interações. Os referidos autores estabelecem que a arquitetura de um software relaciona-se à estrutura dos componentes que o sistema possui, seus inter-relacionamentos, princípios e diretrizes, conduzindo o projeto e a evolução do software ao longo do tempo. Projetos arquitetônicos de sistemas grandes e complexos sempre representaram papel significativo na determinação do sucesso de um software, tendo em vista que a concepção de uma arquitetura inadequada pode ter um efeito desastroso, gerando transtornos e imensuráveis gastos.

Neste sentido, o processo de implementação de um sistema de software abarca duas grandes atividades. A primeira delas precede e orienta a etapa de desenvolvimento, correspondendo ao projeto da arquitetura de software; a outra atividade envolve a criação dos módulos do software que implementam as funcionalidades do sistema, sendo resultado da análise arquitetural, projetada anteriormente, amparando as decisões arquiteturais (FILHO, 2009). Em síntese, a arquitetura de software é um fator de grande relevância para a efetivação coerente das funcionalidades de um sistema que, embora não garanta que todos os requisitos sejam atingidos, forma a base das estratégias de futuras decisões do design do projeto e ainda de questões voltadas à manutenção do produto.

Entretanto, a tarefa de projetar uma arquitetura de software eficiente não possui um molde, uma fórmula mágica. Há a necessidade de grande dedicação por parte do

arquiteto, analisando e assimilando as necessidades do sistema proposto que são as peças fundamentais para a concepção do projeto arquitetural. Percebe-se que tal tarefa requer um amplo período de tempo para a sua concretização, devendo ser realizada por um profissional hábil e experiente que necessitará, para que seu trabalho seja feito da maneira mais adequada possível, de uma boa fundamentação das características do sistema, isto é, da qualidade do levantamento dos requisitos, tendo em vista que a má concepção e o entendimento equivocado dos mesmos resultará em uma arquitetura deficiente.

Ademais, as atividades que envolvem o projeto de arquitetura correspondem a tarefas complexas e custosas, considerando as exigências dos usuários e do mercado, além da sofisticação das tecnologias. Uma das mais sensíveis observações notadas quando se compara um projeto de software atual a um projeto trabalhado implementado há alguns anos, diz respeito a sua complexidade: sistemas de software atuais são bem mais complexos e tal complexidade só tende a crescer com o passar do tempo. Neste contexto, o principal desafio colocado aos arquitetos e/ou projetistas de softwares corresponde à concepção de uma arquitetura concisa que atenda tanto às funcionalidades solicitadas pelos clientes quando aos requisitos que determinarão a qualidade do sistema. Nota-se que há uma proporcionalidade entre o acréscimo de complexidade dos softwares e o aumento da dificuldade, tempo e custos dedicados ao projeto de uma arquitetura que atenda a todos os requisitos previstos.

Focando na arquitetura de um sistema de software, percebe-se que sua concepção se dá através da análise de dois elementos distintos - o processo arquitetural e o produto arquitetural. O primeiro deles, o Processo Arquitetural, compreende as atividades integrantes necessárias à construção do sistema, amparando o seu desenvolvimento. Dentre elas destaca-se (BRUZAROSCO, 2011):

1. Elaboração do modelo de negócio para o sistema - volta-se à análise do custo e prazo para a realização do sistema, bem como as restrições de mercado, focando no público-alvo, e interconexões com outros sistemas, visando sempre alcançar os objetivos do negócio.
2. Entendimento dos requisitos - através da utilização de técnicas destinadas ao levantamento de requisitos, obtém-se o modelo do domínio que representa, de forma simplificada, classes conceituais do mundo real relativas ao domínio do software previsto.
3. Criação ou seleção de uma arquitetura - há a identificação dos componentes do sistema e suas interações, dependências de construção e ainda o apontamento das tecnologias que suportam a implementação.

4. Representação da arquitetura e divulgação - dispõem-se aos *stakeholders*<sup>2</sup> a arquitetura definida e as atividades a ela atribuídas.
5. Implementação do sistema baseado na arquitetura - neste ponto há a codificação do software, na qual os desenvolvedores devem se restringir às estruturas e protocolos definidos na arquitetura criada.
6. Análise ou avaliação da arquitetura - deve-se verificar a conformação da arquitetura, observado os impactos, riscos e/ou dificuldades encontradas na sua adoção e utilização. Esta tarefa possui grande importância, pois as informações advindas da análise durante a implementação do software contribuem para a evolução da arquitetura que será utilizada em versões posteriores do sistema.

O segundo elemento que integra a ideia de arquitetura de sistema de software é o Produto Arquitetural que corresponde aos objetos produzidos diante do processo arquitetural, descrito nas seis atividades anteriores (BRUZAROSCO, 2011; VILLELA, 2001). Em síntese, os artefatos gerados, vistos como modelos de arquiteturas, podem ser: Modelo do negócio, responsável pela descrição da lógica do software, perfil do cliente e definição de funcionalidades; Modelo do domínio de aplicação, ou seja, a representação das conceituais do mundo real, através, e.g., de diagramas; Modelo dos componentes computacionais e relacionamentos entre eles, voltado ao detalhamento simplificado de todos os elementos de distribuição e de comunicação relacionados à aplicação em desenvolvimento; e a Infraestrutura tecnológica, cujo foco encontra-se na definição das tecnologias físicas que serão necessárias para o funcionamento do sistema.

Assim, pode ser percebido que os requisitos, atributos ou funcionalidades serão os grandes definidores dos modelos arquiteturais de um sistema de software. E, neste contexto, os requisitos não-funcionais ou atributos de qualidade são os elementos que precisam de uma atenção redobrada por parte dos arquitetos e desenvolvedores, já que, no desenvolvimento puramente orientado a objetos, estarão espalhados por diversos módulos do sistema. Com isso, caso não se tenha o cuidado adequado, os modelos arquiteturais poderão ser definidos com vícios que resultarão em uma codificação deficiente, fato que causa perda de tempo e, conseqüentemente, perda de recursos financeiros e possível redução da vida útil do sistema produzido.

## 2.2 Requisitos Não-funcionais ou Atributos de Qualidade

Há uma relação direta, em sistemas, entre arquitetura e requisitos, sejam eles funcionais ou não-funcionais. Os requisitos de um software, isto é, as características, atributos, propriedades e restrições próximas ao sistema, de uma forma geral, são classificados em

---

<sup>2</sup> Termo composto por duas expressões: *Stake* (interesse) e *Holder* (aquele que possui); logo, *Stakeholder* é toda pessoa e/ou organização que tenha interesse em determinado projeto.

RF, responsáveis pela definição dos serviços, funções ou funcionalidades esboçadas para o sistema; e RNF, voltados à definição de outras propriedades e restrições do software, as quais podem afetar o sistema como um todo, caracterizados ainda como atributos de qualidade (SOMMERVILLE, 2011). Logo, nota-se que esses atributos são ortogonais, o que significa dizer que podem interferir em vários serviços ou funções do sistema, estando implementados em diversos módulos desse, fato que causa grande ilegibilidade no código, prejudicando seu desenvolvimento e operabilidade.

Conforme destacado anteriormente, a sociedade moderna depende cada vez mais de softwares e a necessidade de sistemas confiáveis também é crescente. A preocupação com softwares de confiança têm recebido bastante atenção nos últimos anos, tornando-se um dos quatro mais importantes focos de pesquisas em estratégia nacional de software nos EUA, buscando, principalmente, a garantia de segurança (CASAMAYOR; GODOY; CAMPO, 2010).

Assim, um número significativo de pesquisas vem sendo desenvolvido para acompanhar a crescente complexidade dos sistemas de software; o aumento da exigência de qualidade por parte dos clientes; e a necessidade, trazida pelo mercado, de softwares que atendam, não somente as funcionalidades exigidas, mas também questões relativas à forma como o sistema se comporta em relação a alguns atributos observáveis, tais como: segurança, desempenho, disponibilidade, extensibilidade, portabilidade, manutenibilidade, rastreabilidade de informações, entre outros. Por não representarem as funcionalidades específicas do sistema, estes aspectos devem ser tratados desde o início do processo de desenvolvimento como requisitos não-funcionais do software, tratamento este que deve ser expandido para todo o ciclo de vida do software (CHUNG; NIXON, 1995; CHUNG; LEITE, 2009; CYSNEIROS, 2001).

Mesmo com as diversas pesquisas voltadas ao estudo dos requisitos não-funcionais, que vêm sendo salientados em diversos processos de desenvolvimento de software, percebe-se que ainda há certa informalidade relativa a sua elicitación. É perceptível a menção a algumas restrições e condições de contorno, contudo grande parte dos arquitetos e programadores tratam os RNF de forma muito secundária, esquecendo da sua tamanha importância durante a pré e pós implementação do sistema. Por terem sido mal elicitados ou não elicitados, tais requisitos encontram dificuldade para serem considerados durante a etapa de desenvolvimento e sua validação torna-se difícil, resultando em softwares precários que, geralmente, “morrem” previamente, ou seja, são desativados pouco tempo após terem sido implantados (FINKELSTEIN; DOWELL, 1996; LINDSTROM, 1993).

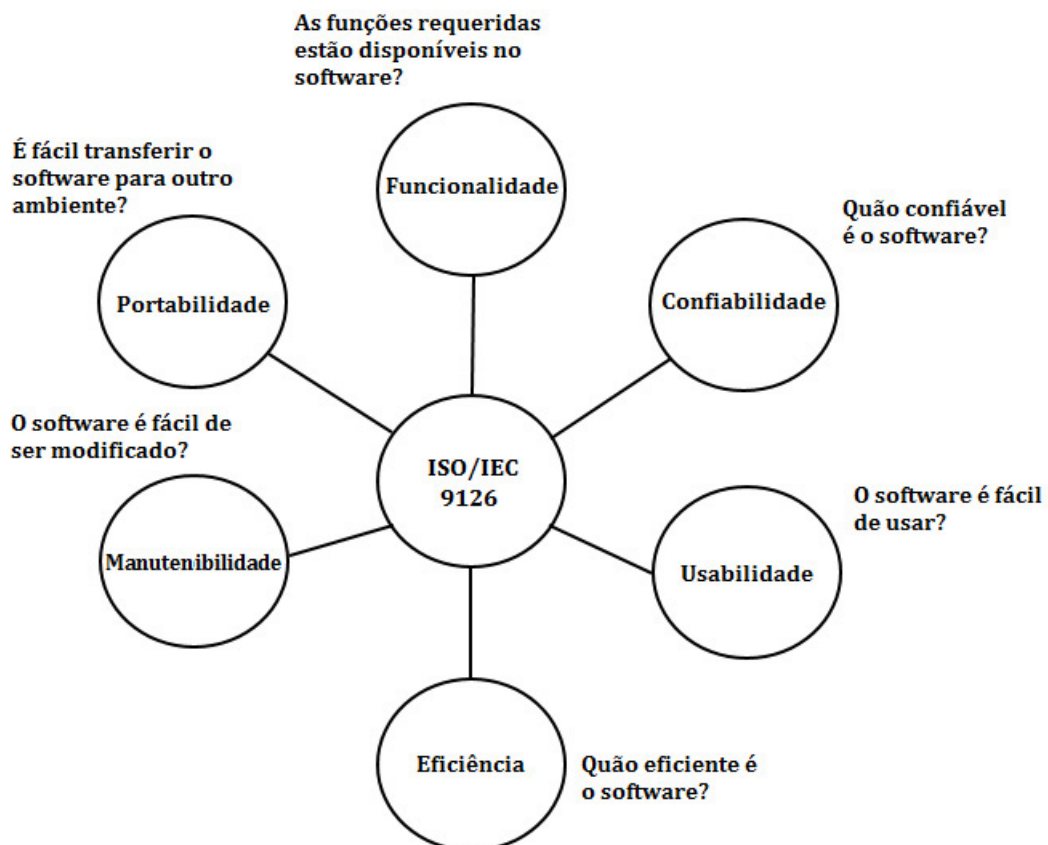
O grande problema envolto ao descuido com os requisitos não-funcionais está no fato de que a preocupação em satisfazer as necessidades do cliente se sobressai junto aos programadores. Os desenvolvedores, muitas vezes, nem sequer percebem que existem atributos de qualidade a serem verificados e que esses serão fundamentais para a construção de um sistema confiável, influenciando totalmente na qualidade do software. As caracterís-



ticas e aspectos internos do sistema que envolvem especificamente a parte técnica, isto é, os atributos de qualidade, não são explicitamente expostos pelo cliente, diferentemente dos RF, porém os desenvolvedores devem ter a sutileza de captá-los e compreendê-los no emaranhado de especificações.

Os requisitos de qualidade de um sistema, os RNF, preocupam-se com a forma como o sistema executa as suas funcionalidades, além do inter-relacionamento entre elas. Por este motivo, a especificação dos atributos de qualidade deve ser realizada desde a etapa de projeto do software, já que esses podem envolver diversos módulos do sistema, reduzindo os problemas inerentes à má modularização. Neste contexto, ressalta-se a série de normas ISO/IEC 9126 (ISO/IEC 9126-1 - 9126-4) (ISO, 2001) que dispõe os principais atributos de qualidade que podem ser conferidos a um sistema, através dos quais se pode aferir sua qualidade. Uma representação dos mencionados atributos pode ser encontrada na Figura 2.

Figura 2 – Modelo de qualidade de software segundo a norma ISO/IEC 9126



Fonte: Adaptado de BERANDER, P. et al. (2006)

De modo sintético, a norma ISO/IEC 9126 apresenta os principais atributos de qualidade inerentes aos softwares; eles são: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. A Tabela 1 apresenta a definição de cada

um dos atributos da referida norma (KIM; KIM, 2014). Destaca-se que esses possuem sub-características que aumentam sua granularidade e nível de detalhamento.

**Tabela 1 – Definição dos atributos da norma ISO/IEC 9126**

<b>Atributo</b>	<b>Definição</b>
<b>Funcionalidade</b>	A capacidade do produto de software, quando utilizado em determinadas condições, prover funções que satisfaçam as necessidades explícitas e implícitas
<b>Confiabilidade</b>	A capacidade do produto de software manter um nível específico de desempenho quando usado sob condições específicas
<b>Usabilidade</b>	A capacidade do produto de software ser compreendido, aprendido e usado intuitivamente pelo usuário, quando utilizado sob condições específicas
<b>Eficiência</b>	A capacidade do produto de software fornecer adequado rendimento, em relação à quantidade dos recursos utilizados, sob condições estabelecidas
<b>Manutenibilidade</b>	A capacidade do produto de software de ser modificado, devido correções, melhorias ou adaptações para mudanças no ambiente ou em requisitos funcionais
<b>Portabilidade</b>	A capacidade do produto de software ser transferida de um ambiente para outro

Fonte: Adaptado de ISO (2001)

Salienta-se que todos os atributos possuem suas respectivas importâncias, não havendo forma de inferir ou predizer que este ou aquele é mais importante ou precisa ser priorizado em detrimento dos demais. Nota-se que a conferência desses garantirá que as funções do sistema estão sendo executadas de forma adequada e eficiente, fato que impede ou prevê a ocorrência de erros que poderiam comprometer todo o trabalho realizado e culminar numa possível invalidade do software desenvolvido.

É importante fazer menção à série de normas ISO/IEC 14598 que dispõe de métodos para mensuração, medição e avaliação de qualidade de softwares. Nela está contida a descrição de técnicas de avaliação tanto para o processo de desenvolvimento do sistema quanto para o produto de software em seu estágio final. A referida série de normas é destinada para uso, em conjunto, com a série de normas ISO/IEC 9126.

Analisando as duas normas, pode-se sintetizá-las conforme segue: a norma ISO/IEC 9126-1 descreve o modelo de qualidade e os demais documentos (ISO/IEC 9126-2, ISO/IEC 9126-3, ISO/IEC 9126-4) correspondem a relatórios técnicos responsáveis por fornecer exemplos de métricas de qualidade; já o documento ISO/IEC 14598-1 define, genericamente, os conceitos envolvidos no processo de avaliação de qualidade de software, os documentos ISO/IEC 14598-2 e ISO/IEC 14598-6 oferecem suporte à avaliação e os demais documentos (ISO/IEC 14598-3, ISO/IEC 14598-4, ISO/IEC 14598-5) são utilizados para apoio no processo de avaliação específico por *stakeholder* (VIEIRA, 2012).

Outro ponto importante a ser destacado é a série de normas ISO/IEC 25000 (ISO/IEC 2500n, ISO/IEC 2501n, ISO/IEC 2502n, ISO/IEC 2503n, ISO/IEC 2504n, ISO/IEC 25050-25099) que formam o modelo de qualidade conhecido como SQuaRE (Requisitos e Avaliação da Qualidade de Produtos de Software). Ela foi construída tomando como base as séries de normas ISO/IEC 9126 e ISO/IEC 14598 e tem o propósito de substituí-las gradativamente. Em resumo, o objetivo geral da série de normas SQuaRE consiste em melhorar e unificar os três principais processos atinentes à qualidade de software, isto é, a especificação de requisitos, a medição de qualidade e a avaliação (KOSCIANSKI; SOFTWARE, 2007).

A série de normas SQuaRE representa uma segunda geração de normas de qualidade de software, construídas para: (i) unificar as normas ISO/IEC 14598 e ISO/IEC 9126 em uma estrutura única de normas; (ii) apresentar uma nova organização de normas; (iii) inserir um novo modelo de referência geral de qualidade; (iv) padronizar guias de qualidade detalhados; (v) introduzir um padrão de primitivas de medição; (vi) normalizar os requisitos de qualidade; e (vii) exibir um guia prático de uso das normas, dotado de exemplos (SURYN; ABRAN; APRIL, 2003).

Em suma, as séries de normas aqui destacadas possuem a finalidade de garantir qualidade a um sistema, seja durante o seu desenvolvimento, seja quando esse já se apresente pronto para uso. É sabido que o modelo de normas para qualidade de produto de software SQuaRE possui a pretensão de substituir os modelos anteriormente concebidos, através da união de normas já em vigor. Contudo, ele ainda possui alguns documentos em fase de revisão e alguns outros em processo de construção. Logo, a efetiva utilização do SQuaRE e consequente cancelamento das séries de normas ISO/IEC 9126 e ISO/IEC 14598 ainda levará algum tempo.

## 2.3 Programação Orientada a Aspectos (POA)

### 2.3.1 Histórico

O paradigma da orientação a aspectos foi desenvolvido durante as décadas de 1980 e 1990 em Palo Alto, especificamente nos laboratórios da Xerox, por Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier e John Irwin (COLYER et al., 2004). Trata-se de uma metodologia de programação inovadora que objetiva complementar o paradigma de orientação a objetos, a fim de amparar as principais fragilidades e problemas encontrados no desenvolvimento de preocupações transversais em softwares orientados a objetos, como os já citados entrelaçamento e espalhamento de código.

Kiczales, no artigo que introduziu a Programação Orientada a Aspectos (KICZALES et al., 1997), infere que a POA não tem a pretensão de substituir os paradigmas existentes, como a programação estruturada ou orientada a objetos, mas atuar em conjunto com esses,

complementando-os na resolução de problemas que tanto a POO quanto a programação procedimental não são capazes de resolver sozinhas de forma eficiente.

Durante o processo de desenvolvimento de um software, as etapas que ocorrem no projeto procuram “quebrar” o sistema em unidades cada vez menores que, por meio dos mecanismos propostos pelas linguagens de programação, permitirão ao programador definir abstrações de subunidades do sistema para, em seguida, juntar tais abstrações e produzir o sistema como um todo. Observa-se que as referidas unidades são representadas, sobretudo, pelos requisitos ou preocupações, partes essenciais para compreender e implementar um software.

De forma geral, os requisitos podem ser considerados componentes, isto é, preocupações facilmente encapsuladas em um objeto, método ou procedimento e que equivalem, geralmente, a uma unidade funcional do sistema. Por outro lado, diferentemente dos componentes, os aspectos ou preocupações transversais não podem ser facilmente encapsulados em um módulo e ainda abarcam atributos que poderão afetar o desempenho ou a semântica de um ou mais desses componentes, como é o caso do tratamento de erros e logging de auditoria (KICZALES et al., 1997).

Nota-se que a POA proporciona uma separação bem definida entre componentes e aspectos, possibilitando ao desenvolvedor um alto nível de abstração. Assim, por estarem alocados em locais distintos e bem definidos, componentes e aspectos aumentam seu grau de reutilização e manutenção e ainda tornam o sistema mais legível e de fácil entendimento. A separação entre componentes e aspectos, principalmente pela sua distribuição em módulos diferentes, simplifica o problema, já que cada preocupação pode ser programada de forma independente das demais, para posteriormente combiná-las entre componentes e aspectos, produzindo, assim, o sistema final (WINCK; JUNIOR, 2006).

Dessa forma, pode-se definir um aspecto como o nível de abstração promovido pela POA para requisitos não-funcionais, ou seja, para preocupações sistêmicas que não estão ligadas diretamente a um componente do sistema e sim a vários deles; enquanto que as preocupações funcionais são implementadas na forma de componentes, utilizando os métodos propostos pelos paradigmas de programação convencionais que podem ser representados pelas linguagens Java, C/C++, Delphi, Lua, Common Lisp, entre outras. Essencialmente, o aspecto é formado pelo código que implementa as preocupações transversais e as regras que definem os locais ou pontos de atuação onde o comportamento implementado deve ser executado, sendo responsável ainda por representar os requisitos difíceis de serem isolados, aqueles que geram o espalhamento dos códigos em vários módulos do sistema (RODRIGUES, 2007).

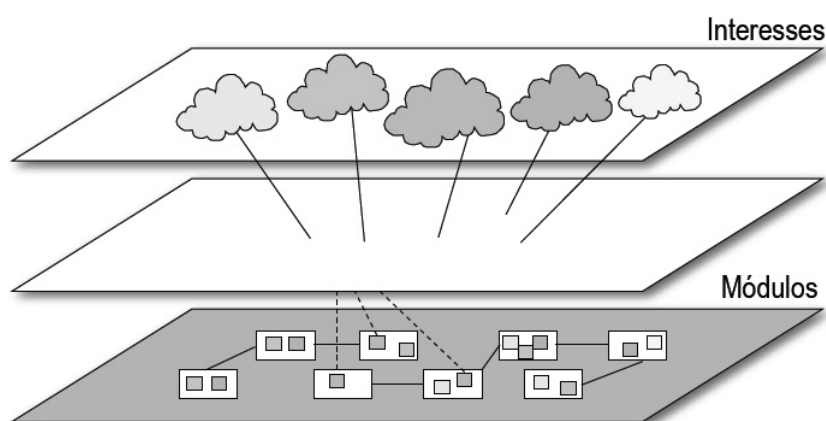
### 2.3.2 Definições

Conforme mencionado, a POA não propõe a substituição dos paradigmas de programação existentes, mas sim um complemento ao tratar seus pontos fracos, voltados,

principalmente, às preocupações sistêmicas. De maneira geral, o alvo principal da POA é separar o código referente ao negócio do sistema (requisitos funcionais) dos interesses transversais, centralizando-os de forma bem definida. Os referidos interesses denotam características relevantes de uma aplicação e podem ser alocados em uma série de aspectos que representam os requisitos do sistema (WINCK; JUNIOR, 2006).

A Figura 3 demonstra a separação e a centralização dos interesses transversais propiciadas pela POA, onde cada nuvem remete a um interesse transversal implementado, como o logging de auditoria, tratamento de exceções, persistência, dentre outros. E, por estarem dispostos em locais distintos e bem definidos, formam componentes que podem ser mais facilmente reutilizados, com alto grau de manutenibilidade e maior legibilidade.

**Figura 3 – Separação de interesses com POA**



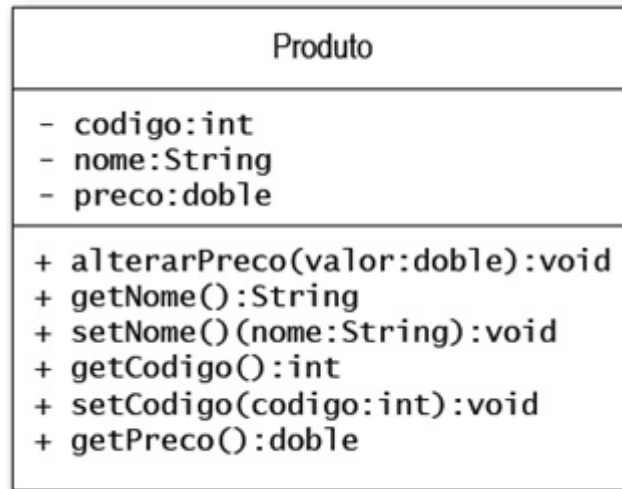
Fonte: WINCK, D. V.; GOETTEN JR., V. (2006)

A programação orientada a aspectos oferece uma simplificação da engenharia de software, tornando os sistemas mais claros, concisos e em bom alinhamento com o modelo de domínio, evitando possíveis problemas em sua funcionalidade. Através dos aspectos há a disponibilização de mecanismos para a abstração de elementos e relações entre requisitos, além da alocação precisa dos interesses sistêmicos em módulos distintos (WINCK; JUNIOR, 2006; KICZALES et al., 1997).

Como exemplificação, tem-se a Figura 4 que apresenta a disposição de uma classe de controle de produtos de um Sistema X, utilizando unicamente a programação orientada a objetos. Em um primeiro momento, observa-se uma implementação adequada, atendendo apenas a um interesse e às responsabilidades que são suas (coesão), além de não infringir nenhum conceito da orientação a objetos.

Entretanto, ao ponto que surge a necessidade de acrescentar mais funções a determinada classe, os conceitos básicos da orientação a objetos, como a coesão, são violados. O desenvolvedor do Sistema X, e.g., deseja adicionar o logging de auditoria e o controle de exceções ao seu projeto, porém não utiliza a POA. Assim, os interesses citados seriam

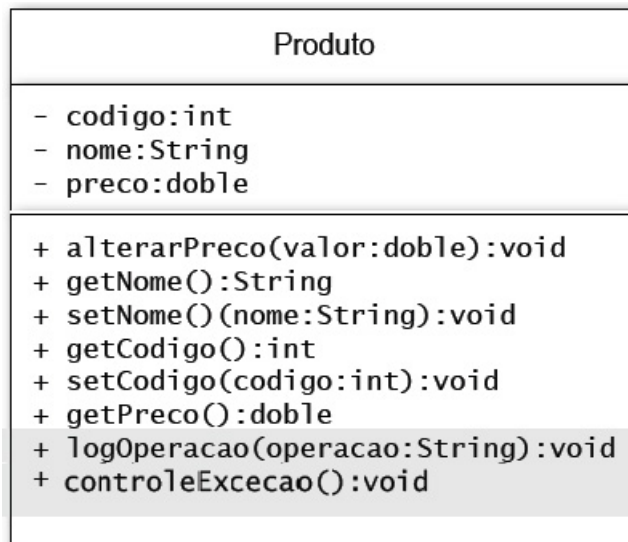
Figura 4 – Classe Produto



Fonte: WINCK, D. V.; GOETTEN JR., V. (2006)

implementados na própria classe Produto (conforme pode ser observado na Figura 5), fato que traz mais complexidade ao código, reduz sua coesão e legibilidade e acarreta diversos outros problemas que tendem a crescer à medida que mais necessidades vão sendo idealizadas e acrescentadas ao sistema.

Figura 5 – Logging de auditoria e controle de exceções classe Produto



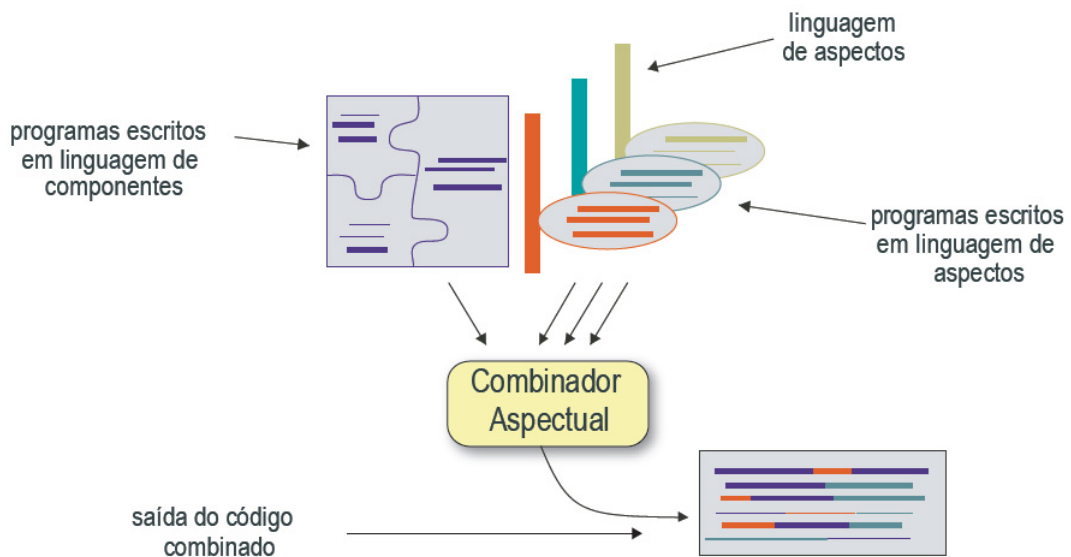
Fonte: Adaptado de WINCK, D. V.; GOETTEN JR., V. (2006)

A implementação de um sistema baseada na POA é composta, basicamente, pela Linguagem de componentes, utilizada para implementar as funcionalidades básicas da aplicação, os requisitos funcionais; Linguagem de aspectos, empregada na implementação das preocupações sistêmicas, ou seja, dos interesses transversais; Programas escritos

em linguagem de componentes, que dispõem a implementação de componentes relativos às preocupações funcionais do sistema; Programas escritos em linguagem de aspectos, que atendem às preocupações sistêmicas; e um Combinador de aspectos, denominado de *aspect weaver*, responsável pela combinação aspectual, isto é, por ajustar os programas escritos em linguagem de componentes com os escritos em linguagem de aspectos (WINCK; JUNIOR, 2006; KICZALES et al., 1997; MONTEIRO; PIVETA, 2003).

A composição de um sistema orientado a aspectos pode ser observada na Figura 6.

**Figura 6 – Composição de um Sistema Orientado a Aspectos**



Fonte: WINCK, D. V.; GOETTEN JR., V. (2006)

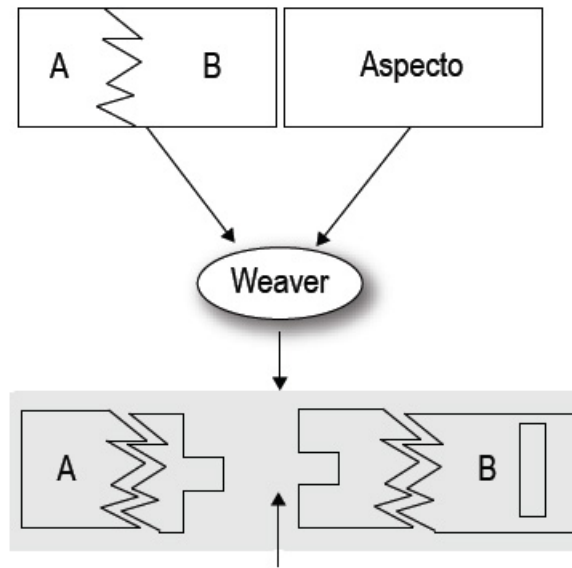
### 2.3.3 Conceitos fundamentais

É imprescindível a disposição de alguns conceitos para entender a orientação a aspectos. O primeiro deles é o já citado Combinador de Aspectos ou *aspect weaver*, responsável por processar os programas em linguagem de componentes e linguagem de aspectos, recompondo-os adequadamente para produzir o funcionamento total desejado do sistema (KICZALES et al., 1997). A Figura 7 ilustra como ocorre a combinação aspectual. Este processo antecede a compilação, gerando um código intermediário na linguagem de componentes, com o qual será possível executar o interesse implementado.

Além do entendimento acerca do *aspect weaver*, quatro outros conceitos são fundamentais em orientação a aspectos. Eles são: Pontos de Junção (*joinpoints*), Pontos de Atuação (*pointcuts*), Adendo (*advice*) e Aspectos.

*JoinPoints* ou Pontos de Junção são termos utilizados em POA para indicar um ponto bem definido no fluxo de execução de um programa (WINCK; JUNIOR, 2006; RESENDE; SILVA, 2005). De forma sintética, os pontos de junção definirão como e onde será feita a junção entre classes (linguagem de componentes) e aspectos (linguagem de aspectos).

Figura 7 – Combinação Aspectual



Fonte: WINCK, D. V.; GOETTEN JR., V. (2006)

Alguns exemplos de *joinpoints* são as chamadas de métodos, execução de construtores, instanciação de objetos e a ocorrência de uma exceção. Ressalta-se que os pontos de junção compõem a base da POA e apenas por meio deles será possível determinar quais métodos terão suas chamadas e/ou execuções interceptadas, interrompendo o fluxo natural de um programa para a execução de uma rotina à parte (RESENDE; SILVA, 2005).

Outro conceito igualmente importante corresponde aos Pontos de Atuação, Pontos de Corte ou *PointCuts*. De forma geral, os pontos de atuação podem ser definidos como um conjunto de pontos de junção ou ainda como um regra criada pelo desenvolvedor para especificar eventos que serão conferidos aos pontos de junção. Os *pointcuts* declararão os pontos nos quais a execução do programa será interrompida e inserir-se-á um outro comportamento (WINCK; JUNIOR, 2006; RESENDE; SILVA, 2005; GARCIA, 2005).

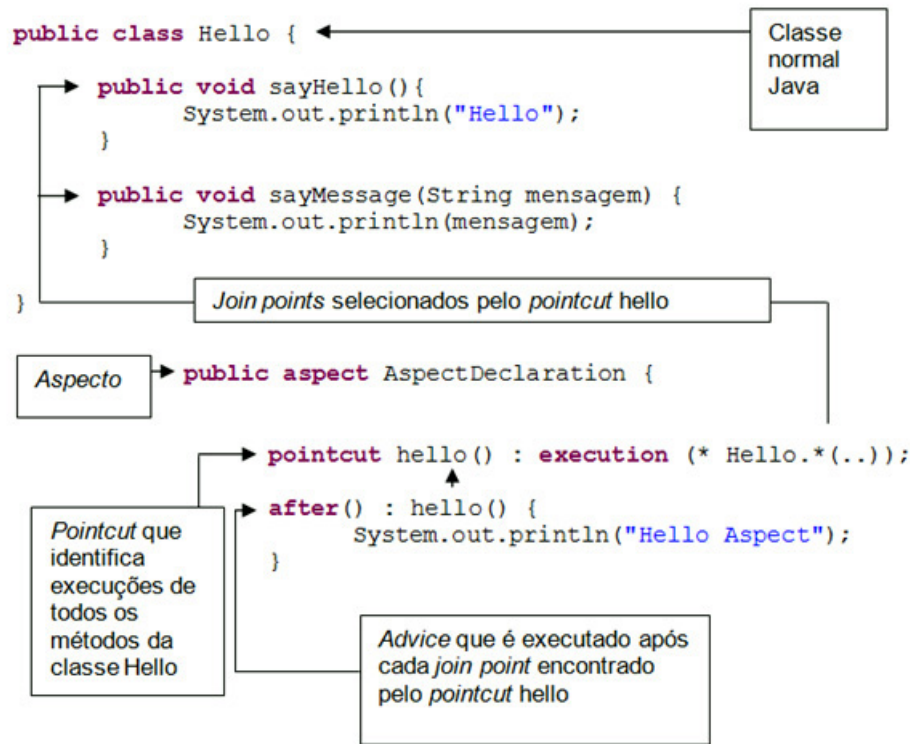
O comportamento referido anteriormente corresponde aos Adendos ou *Advices*. Eles são as partes ou pedaços da implementação presentes em um aspecto que definirão o comportamento a ser executado nos pontos bem definidos do sistema, ou seja, nos pontos de junção. Os adendos são compostos por duas partes: uma determina o ponto de atuação, onde as regras de captura dos pontos de junção serão definidas, e.g., sempre que determinado construtor for invocado; e outra que delimita o código a ser executado quando notar o ponto de junção previamente definido (WINCK; JUNIOR, 2006).

O conceito que une todos os demais é o de Aspectos, definidos como os mecanismos disponibilizados pela programação orientada a aspectos para agrupar fragmentos de código referentes aos componentes não-funcionais, interesses transversais, em um componente do sistema. A Figura 8 mostra um esquema que salienta o relacionamento entre uma classe, em Java, e um aspecto. A partir da imagem é possível exemplificar os quatro conceitos



mencionados anteriormente.

Figura 8 – Esquema: relacionamento entre uma classe e um aspecto



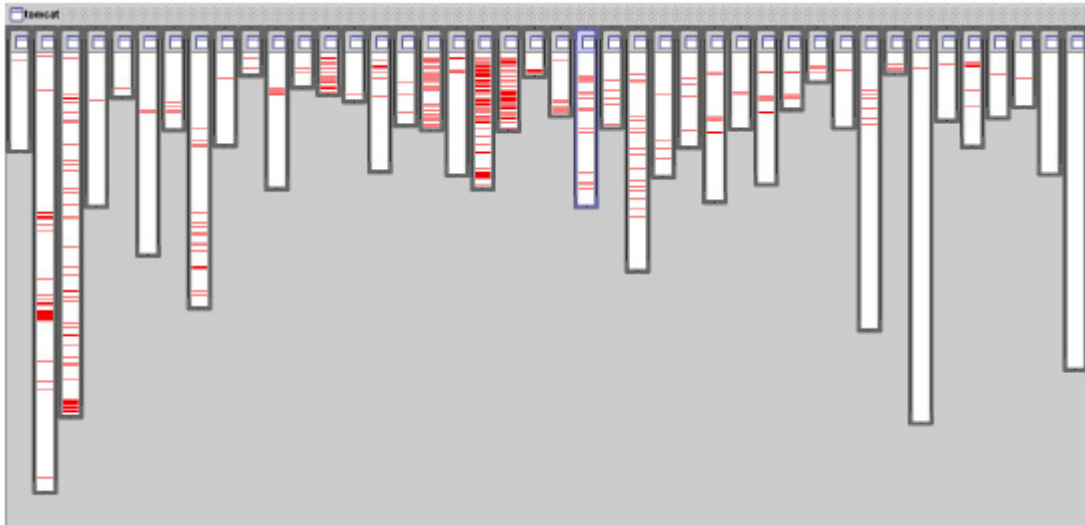
Fonte: GARCIA, R. (2005)

Percebe-se que interesses não relacionados diretamente ao domínio da aplicação, isto é, interesses sistêmicos, poderão ser agrupados em aspectos e assim evitar o espalhamento e emaranhamento de códigos, problemas recorrentes quando utilizadas unicamente as linguagens orientadas a objetos para a implementação de requisitos funcionais e não-funcionais.

A Figura 9 mostra um diagrama que exemplifica a ocorrência de um código emaranhado, no qual a implementação de um interesse transversal está espalhado em diversos componentes do sistema. Realizando um contraponto, a Figura 10 exhibe um outro diagrama que mostra o agrupamento dos códigos relativos ao requisito não funcional implementado, mas com a utilização de aspectos, extinguindo o espalhamento.

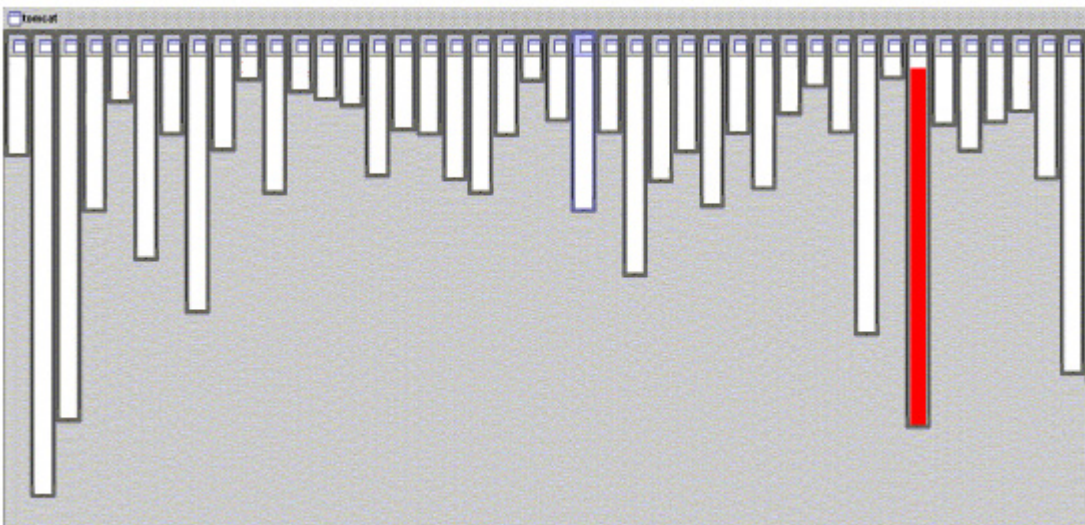
Infere-se que aspectos são módulos distintos e separados fisicamente que possibilitam uma forma adequada para que requisitos que estariam dispersos aleatória e repetidamente pelo código do software sejam implementados. Além desta unidade de modularização, composta pelos elementos aqui abordados, as linguagens pertinentes ao DSOA abrangem métodos para a construção de aspectos e também de classes convencionais da POO. Neste contexto, o paradigma orientado a aspectos comporta duas etapas, a primeira delas corresponde à decomposição do sistema em partes não entrelaçadas, isto é, a separação entre os códigos dos RF e RNF; e a segunda volta-se à junção dessas partes através

Figura 9 – Implementação de requisito não funcional com POO: código emaranhado



Fonte: WINCK, D. V.; GOETTEN JR., V. (2006)

Figura 10 – Implementação de requisito não funcional com POA: código agrupado



Fonte: WINCK, D. V.; GOETTEN JR., V. (2006)

do processo de combinação (*weaving*) a fim de obter o sistema com as funcionalidades pretendidas.

Em síntese, implementar aspectos para a composição de sistemas, sejam eles desenvolvidos em Java ou em qualquer outra linguagem de programação, reduz drasticamente o problema observado na Figura 9, ou seja, o entrelaçamento e espalhamento de código que tornam extremamente difíceis o desenvolvimento, a manutenção e ainda podem interferir na operabilidade dos sistemas. Assim, a POA possibilita ao programador o aumento da modularidade das aplicações que estão sendo implementadas, separando o código respon-

sável pelas funções específicas daquele que envolve preocupações que agem diferente sobre um ou um conjunto de atributos funcionais de determinado software.

### 2.3.4 Exemplo Prático

Supondo que o desenvolvedor de um Sistema Bancário perceba ser fundamental registrar as ações e os usuários que as executaram, não possuindo conhecimento em POA, ele poderia implementar métodos nos diversos módulos do seu sistema a fim de armazenar os dados relativos a essas ações. Um exemplo pode ser visto no trecho de código disposto na Figura 11.

**Figura 11 – Classe Conta: logging no método armazenar**

```

1 public class Conta {
2     private String numero;
3     private double saldo;
4     ...
5     public void debitar (double valor) {
6         if (this.getSaldo() > = valor){
7             this.setSaldo(this.getSaldo() - valor);
8             armazenar("ocorreu um debito!", data, usuario);
9         }...
10    public void creditar (double valor) {
11        if (valor > = 0){
12            this.setSaldo(this.getSaldo() + valor);
13            armazenar("ocorreu um credito!", data, usuario);
14        }...
15    }

```

Fonte: Elaborada pelo autor

Observando o código destacado, infere-se que caso a retenção de dados fosse necessária somente no método *debitar* ou apenas na classe *Conta*, utilizar a chamada do método *armazenar* seria ideal. No entanto, o mesmo *armazenar* também é empregado na função *creditar* e, provavelmente, seria utilizado nos demais procedimentos da aplicação, sendo imprescindível ainda em praticamente todas as classes do sistema.

Assim, o interesse de logging estaria fortemente misturado com código de negócio, repetindo-se em diversos pontos de uma mesma classe e em diversos módulos e outras classes. Para resolver este problema pode-se propor uma solução, simples neste exemplo, para modularizar o logging. O código apresentado na Figura 12 mostra um aspecto, criado através da linguagem AspectJ<sup>3</sup>, que tem por finalidade lidar com o registro das ações que ocorrem nos métodos *debitar* e *creditar* da classe *Conta*.

Os *pointcuts* ou pontos de atuação *logCredito()* e *logDebito()* referem-se, respectivamente, a todas as chamadas dos métodos *creditar* e *debitar*, isto é, aos pontos de junção

<sup>3</sup> AspectJ - linguagem de programação pertencente ao paradigma orientado a aspectos - <<http://eclipse.org/aspectj/>>.

Figura 12 – Aspecto Voltado ao Logging no Sistema Bancário

```

1 public aspect LogContas {
2     ...
3     pointcut logCredito(): call (* Conta.creditar(double));
4     pointcut logDebito(): call (* Conta.debitar(double));
5     after (): logCredito(){
6         armazenar("ocorreu um credito!", data, usuario);
7     }
8     after () returning: logDebito(){
9         armazenar("ocorreu um debito!", data, usuario);
10    }
11 }

```

Fonte: Elaborada pelo autor

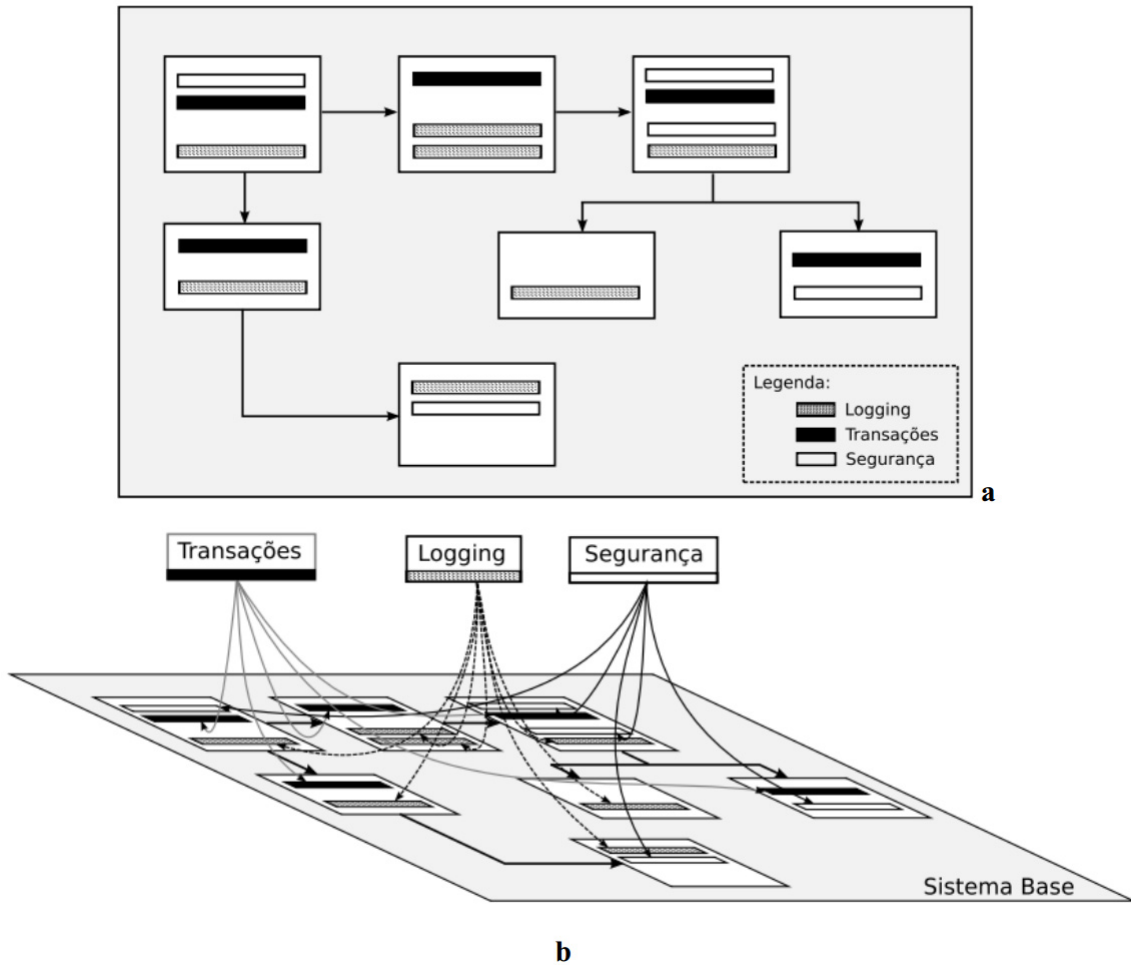
*creditar(double)* e *debitar(double)*. Definidos os pontos de atuação, notam-se os adendos que se figuram iniciados pela expressão *after*. Eles denotam que após cada ponto de *logCredito* e *logDebito* serão executadas as instruções que seguem, neste caso o método *armazenar*. Em suma, este exemplo apresenta a grande semelhança estrutural entre os Aspectos e as classes observáveis na POO, bem como a tamanha utilidade que esses propiciam para a modularização do logging.

De forma análoga ao Logging de Auditoria, o interesse Registro de Exceções, quando implementado em linguagens de programação pertencente ao POO, também culmina no espalhamento e entrelaçamento de código, prejudicando ainda a funcionalidade e consequente qualidade de determinado sistema. A Figura 13 esquematiza a implementação de interesses transversais utilizando unicamente POO (a), com grande espalhamento de código, e a codificação desses interesses com POA, agrupando-os em um único componente ou módulo, fato que traz maior legibilidade, facilidade de reúso e não interfere diretamente nos RF do sistema.

Percebe-se que a linguagem Java possui meios que permitem que as exceções sejam tratadas de forma adequada, sem que o sistema termine bruscamente seu processo (FILIPPETTO; CALLEGARI, 2006). Além deste tratamento, já amparado em Java, é necessário que todas as exceções sejam agrupadas e armazenadas, tendo em vista uma posterior avaliação e detecção de erros. Este registro é imprescindível para otimizar e corrigir eventuais falhas. Assim, nota-se a Programação Orientada a Aspectos como alternativa para lidar com os problemas oriundos da implementação de RNF em sistemas desenvolvidos em POO, concentrando os interesses transversais em módulos específicos, reduzindo drasticamente os problemas citados.

Dessa forma, a DSOA é uma proposta de solução para os problemas de modularização causados pela baixa eficiência dos atuais paradigmas de programação, otimizando a implementação de softwares e tornando a estrutura do sistema mais legível e de mais fácil manutenção (KICZALES et al., 1997). Além de uma adequada modularidade de interesses transversais - como segurança, desempenho, persistência, distribuição e tra-

Figura 13 – Interesses transversais em sistemas definidos com POO (a) e POA (b)



Fonte: RAINONE, F. (2008)

tamento de exceções - propondo uma clara divisão entre interesses multidimensionais e requisitos funcionais de um sistema, o desenvolvimento orientado a aspectos oferece mecanismos eficientes para lidar com particularidades não tratadas adequadamente pelas tecnologias convencionais de programação, encapsulando códigos de RNF que, e.g., no desenvolvimento orientado a objetos, estariam distribuídos por diversos módulos do sistema (GARCIA et al., 2004).

### 3 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais trabalhos relacionados ao tema proposto, utilizados para a avaliação do estado da arte e ainda para fundamentar e orientar as atividades deste trabalho. Nas pesquisas e trabalhos apresentados foram analisadas questões atinentes à monitoração de atributos de qualidade e à utilização da programação orientada a aspectos para tratar dessa monitoração. Visando uma melhor análise, a bibliografia revisada e aqui disposta foi dividida em três grupos: trabalhos referentes à monitoração de atributos de qualidade em sistemas orientados a serviços; trabalhos concernentes ao logging em aplicações Java por meio da utilização da POA; e trabalhos relativos à captura e registro de exceções em sistemas Java com a utilização de aspectos.

#### 3.1 Monitoração de Atributos de Qualidade em Sistemas Baseados em Serviços

Atribuir qualidade a um sistema é uma tarefa muito importante, porém difícil de ser realizada. Ela é efetivada através do tratamento dos atributos de qualidade; dessa maneira, quando um sistema cumpre determinados requisitos tidos como definidores de qualidade, ele pode ser considerado um sistema capaz de realizar eficiente e eficazmente as funções que lhe foram projetadas.

Grande parte dos trabalhos que discutem e ressaltam a importância da aferição de qualidade voltam-se para sistemas baseados em serviços, como observado em Wetzstein et al. (2009), Artaiam e Senivongse (2008), Michlmayr et al. (2009), Moser, Rosenberg e Dustdar (2008), Muller et al. (2012), Haiteng, Zhiqing e Hong (2011), Souza et al. (2011) e Godse, Bellur e Sonar (2010). Esses trabalhos, de forma geral, destacam a monitoração de atividades, proporcionada pelo tratamento de RNF, como um dos grandes indicadores da Qualidade de Serviço (QoS) da categoria de sistemas mencionados.

Wetzstein et al. (2009) apontam que a monitoração das atividades de negócios e dos relacionamentos entre clientes e servidores permite a observação contínua de indicadores de desempenho. No entanto, atualmente, estes indicadores não trazem dados suficientes que permitam determinar as causas de um possível déficit na operabilidade dos sistemas. Neste contexto, os analistas de negócios ficam carentes de informações acerca dos fatores que influenciam o desempenho dos processos de negócio e, na maioria das vezes, contribuem para a violação do desempenho. Esse trabalho apresenta a construção de um framework que mostra as dependências, indicadores de desempenho e as métricas de QoS, trazendo um conhecimento profundo sobre a estrutura dessas dependências.

Artaiam e Senivongse (2008) expõem que as atuais ferramentas de monitoração e coleta de informações sobre a qualidade do serviço apoiam apenas um número limitado de atributos QoS. Tal constatação ocorre mediante o estudo de seis ferramentas próprias

para a monitoração de Web services. Os autores propõem métricas mais precisas de monitoração, bem como apresentam uma ferramenta voltada a melhorar a medição da QoS de serviços da Web, abrangendo um maior número de atributos de qualidade que as aplicações analisadas.

Michlmayr et al. (2009) mostram os pontos fortes e fracos das abordagens destinadas à monitoração contínua do tempo de resposta e disponibilidade de sistemas cliente-servidor. Esse estudo define uma nova abordagem de monitoração, combinando vantagens observadas em outros trabalhos, que processa eventos e informa aos interessados os valores ou medidas atuais de QoS, além de possíveis violações.

Moser, Rosenberg e Dustdar (2008) destacam que os processos dos sistemas baseados em serviços Web necessitam de uma monitoração robusta e de mecanismos de adaptação dinâmica, ou seja, em tempo de execução, pois diversos serviços necessitam ser frequentemente trocados devido a uma variedade de razões, como a falha de determinado servidor. Os autores exibem o desenvolvimento de um sistema que permite o monitoramento de processos de serviços Web e a substituição, observando a QoS, de serviços parceiros já existentes, com base em diversas estratégias.

Muller et al. (2012) exibem que técnicas de garantia de qualidade têm sido desenvolvidas para monitorar a QoS definida para sistemas baseados em serviços (SBSS), sejam eles clientes ou servidores. As plataformas de monitoração têm sido desenvolvidas para dar suporte à detecção de violações, contudo as explicações acerca de tais violações não são apresentadas em um formato amigável ao usuário. Dessa forma, esse trabalho destaca a implementação do SALMonADA, um sistema que notifica os clientes sobre as violações e suas causas, utilizando termos de especificação de fácil entendimento.

Haiteng, Zhiqing e Hong (2011) corroboram a necessidade da monitoração em tempo de execução de sistemas baseados em serviços. Os autores utilizam a Programação Orientada a Aspectos para acessar as informações de serviços da Web sobre os estados de execução, calculando a QoS. Assim, é vista uma abordagem que permite a clara separação entre a lógica do negócio do sistema e a funcionalidade de monitoração. Além disso, há a oferta da capacidade de aferição tanto do tempo de execução do serviço quanto das propriedades QoS.

Souza et al. (2011) salientam a necessidade crescente de monitorar atributos de qualidade, como o desempenho e disponibilidade em sistemas sob ambientes SOA - *Service-oriented architecture* (Arquitetura Orientada a Serviços). O trabalho mostra que as abordagens existentes para monitorar os referidos atributos, definidos como atributos QoS, não permitem reconfiguração dinâmica dos serviços que estão em execução. Logo, é proposto um mecanismo genérico capaz de monitorar, em tempo de execução, os atributos de serviços de qualidade; além de uma solução de reconfiguração dinâmica, baseada em eventos.

Godse, Bellur e Sonar (2010) defendem que a presença de vários serviços de compar-



tilhamento em uma única interface funcional comum necessita, com no desempenho, de diferenciação entre esses. Porém, a QoS por si só não permite determinar como os serviços foram efetivados, nem predizer o possível funcionamento futuro, informação importante para a seleção de serviços. Assim, esse trabalho descreve um método de monitoramento de desempenho voltado a estabelecer a forma como os serviços foram realizados e amparar a seleção futura de outros serviços.

Em suma, os trabalhos de Wetzstein et al. (2009), Artaiam e Senivongse (2008), Michlmayr et al. (2009) e Moser, Rosenberg e Dustdar (2008) destacam que a monitoração de RNF é um dos grandes indicadores da Qualidade de Serviço (QoS, em inglês, *Quality of Service*) de sistemas baseados na Web. Esses estudos buscam estabelecer técnicas e frameworks para a efetiva monitoração dos referidos sistemas, realizando análises aprofundadas das métricas de QoS a fim de obter padrões que garantam a qualidade dos sistemas. É enfatizado que as atuais ferramentas de monitoração e coleta de informações QoS apoiam somente um número limitado de atributos e, muitas vezes, deixam de lado importantes dimensões de qualidade de serviço, como disponibilidade, acessibilidade, desempenho, confiabilidade e segurança. Esses trabalhos ainda mostram a projeção e desenvolvimento de frameworks voltados à medição de alguns atributos QoS, salientando a necessidade e precisão das técnicas de monitoração oferecidas.

Seguem esta mesma linha de pesquisa, Muller et al. (2012), Haiteng, Zhiqing e Hong (2011), Souza et al. (2011) e Godse, Bellur e Sonar (2010) que ressaltam a importância da aferição de atributos de qualidade para os processo de negócio em serviços Web. É destacada a dinamicidade dos sistemas baseados na Web, bem como a necessidade de criar ferramentas que suportem a detecção de violações, fornecendo, de forma amigável ao usuário, explicações sobre os impactos provenientes dessas violações. Evidencia-se ainda o interesse em criar meios que proporcionem a adequação/substituição, em tempo real, de processos que estejam comprometendo a QoS, além de meios que prevejam aqueles serviços com tendência a apresentar falhas.

Diante dos referenciados trabalhos, pode-se perceber a importância que a monitoração de atributos de qualidade possui em sistemas baseados em serviços. Essa mesma importância pode ser estendida para os demais tipos de sistema. Dessa forma, os estudos apresentados por cada um desses autores contribuirão para o desenvolvimento do trabalho presente neste documento, fundamentando as pesquisas realizadas para a execução das atividades pretendidas.

### 3.2 Logging de Aplicação com Aspectos

Logging ou Auditoria consiste na atividade de registrar, cronologicamente, as operações realizadas em um software. Cada registro de operação, isto é, cada log é gerado sempre que uma tarefa for efetivada, armazenando dados, como o nome de determinada operação executada, o usuário que a efetuou, data e horário, dentre outros atributos.



Nota-se que o registro constante das atividades do software é constantemente deixado de lado durante a concepção e desenvolvimento do sistema, mesmo possuindo grande importância nos processos de auditoria (BRAZ, 2003).

Embora o advento da Programação Orientada a Aspectos seja datado do ano de 1997, a comunidade técnico-científica ainda não a abraçou, mesmo observando a tecnologia valiosa para registro de log que a POA representa (POLOZOFF, 2003). Assim, as principais técnicas relacionadas ao Logging, vistas na atualidade, não utilizam aspectos, fazendo com que trechos de código sejam repetidos em diversos módulos do sistema, gerando espalhamento e entrelaçamento, problemas já mencionados no capítulo anterior.

Entretanto, nota-se a existência de alguns trabalhos desenvolvidos a partir da utilização de aspectos, aliados a linguagens orientadas a aspectos, como o AspectJ. Neste sentido, uma das mais recentes contribuições é o trabalho de Meetei, Goel e Wasan (2011) que utiliza o Logging de Aplicação, definido através de aspectos, como instrumento de análise do resultado de casos de teste, também definidos por meio da POA. Os autores constatarem uma maior facilidade, quando comparada a técnicas puramente orientadas a objetos, na observação de detalhes internos da execução de testes de unidade, integração e sistema; sendo possível, inclusive, gerar relatórios de cobertura através dos dados recolhidos em arquivo de log durante a execução de testes.

Outro trabalho também relevante é o proposto por Filippetto e Callegari (2006). Nele, os autores apresentam um estudo de caso realizado no sistema de uma multinacional, no qual são implementados aspectos voltados ao tratamento de exceções, logging e tracing, realizando-se um paralelo e comparando-se resultados com esses mesmos requisitos implementados outrora sem a utilização de POA. Esse trabalho destaca a diminuição do entrelaçamento do código obtido para as funcionalidades propostas, o possível ganho de tempo em futuras manutenções devido ao encapsulamento de determinadas funções que tornam as modificações mais pontuais e diminuem o percentual de futuros testes de regressão para garantia de consistência da manutenção, reduzindo o custo do projeto como um todo, dentre outros fatores.

Cabe ressaltar ainda a pesquisa de Rodrigues (2007), na qual é desenvolvido um sistema de auditoria para aplicações java, fazendo-se o uso de aspectos. Nesse trabalho é destacada a promoção de um novo e mais alto nível de abstração, em detrimento da exclusiva utilização da linguagem orientada a objetos, proporcionado pela POA. Ademais, os resultados do trabalho demonstram as vantagens da orientação a aspectos e sua eficácia, por meio do estudo de caso de um sistema de auditoria (*logging*) para aplicações Java.

Logo, observando os trabalhos mencionados nesta seção, fica clara a importância que a Programação Orientada a Aspectos possui no tratamento de Requisitos Não-funcionais, trazendo uma série de vantagens à monitoração de atributos de qualidade. Os conhecimentos provenientes dos estudos elencados serão fundamentais para a concretização dos objetivos supracitados neste trabalho.

### 3.3 Registro de Exceções com Aspectos

Sistemas computacionais são formados por elementos físicos (hardware) e lógicos (software) que, eventualmente, podem falhar durante sua execução. As falhas apresentam variadas origens, vão desde fenômenos naturais a ações humanas ostensivas ou acidentais. Quando uma falha ou exceção ocorre em um software, um erro consequente é gerado e há a observação de um defeito no programa que, muitas vezes, finaliza-o bruscamente.

Uma exceção pode ser definida como a indicação de que um problema ocorreu durante a execução de um programa. Nota-se que o termo exceção denota que o problema não ocorre frequentemente (DEITEL; DEITEL, 2010). Logo, o tratamento de exceções abarcará técnicas que objetivam estruturar as possíveis atividades excepcionais de um software, fazendo com que falhas possam ser detectadas, sinalizadas e tratadas, permitindo ao programa continuar executando em vez de encerrar ou, uma vez que descontinuou, ser possível detectar o local onde a falha ocorreu e prosseguir com seu tratamento.

Grande parte das linguagens de programação, como é o caso de Java, permite que as exceções ou falhas lógicas ocorridas durante o curso de um sistema possam ser tratadas de maneira adequada, sem que ocorra a interrupção inesperada do sistema, permitindo gerenciar os erros de forma organizada, ainda durante a execução do programa, ou ter acesso a registros que detalham qual o ponto exato da falha.

O tratamento de exceções pode ser implementado de diversas formas, e.g., através das técnicas convencionas das linguagens de programação orientadas a objetos. Porém, os problemas salientados neste trabalho, quanto a repetição de códigos em diversos módulos da aplicação, far-se-ão presentes. Assim, a utilização de aspectos voltados ao tratamento de exceções visa facilitar a modularização das técnicas que estruturam as atividades excepcionais de um sistema.

Há três pesquisas que se sobressaem quanto ao uso de POA para o tratamento de exceções. A primeira delas é o trabalho de Castor et al. (2009) “*On the modularization and reuse of exception handling with aspects*”, no qual é apresentado um estudo aprofundado sobre a adequação da linguagem AspectJ em vista à modularização e reutilização de código de tratamento de exceção, realizando a refatoração de aplicações existentes que tiveram seus códigos responsáveis por implementar estratégias de tratamento de erros substituídos por aspectos manipuladores de exceção.

O trabalho imediatamente citado possui as seguintes contribuições: (i) uma melhoria substancial, com base na experiência adquirida com a refatoração de cinco aplicações diferentes, para o corpo de conhecimento existente sobre utilização da POA na manipulação de exceção; (ii) um conjunto de cenários que podem ser usados pelos desenvolvedores para entender melhor quando é benéfica ou não usar aspectos para o tratamento de exceções; (iii) uma avaliação inicial dos efeitos do comportamento de aspectos perante a manipulação de exceção; e (iv) uma análise do impacto dos aspectos sobre a reutilização de código de manipulação de exceções.

O segundo trabalho destacado nesta seção é o “Modularização de tratamento de exceções usando programação orientada a aspectos” de Ferreira (2006), nele é exposto um estudo quantitativo da adequação de POA para modularizar códigos de tratamento de exceções. É realizada a refatoração de dois sistemas orientados a objetos e um sistema orientado a aspectos, nos quais foram removidos os códigos responsáveis pelo tratamento de exceções e implementados aspectos equivalentes. Em síntese, o referido trabalho contribui com uma análise de algumas vantagens e desvantagens do uso de POA para modularizar tratamento de exceções em sistemas orientados a objetos e orientados a aspectos; com um conjunto de cenários que buscam documentar padrões orientados a aspectos para modularizar o destacado tratamento; e com outro conjunto de cenários que indicam como aspectos podem promover o reúso de código de tratadores de exceções.

Destaca-se ainda a pesquisa de Lippert e Lopes (2000) “*A Study on Exception Detection and Handling Using Aspect-Oriented Programming*”. Neste trabalho é realizado um estudo sobre a utilização de POA para a implementação do tratamento de exceções, constatando-se, e.g., que a linguagem orientada a aspectos AspectJ suporta implementações que reduzem drasticamente a porção do código relacionada à detecção e tratamento de exceções, mais tolerância para alterações nas especificações de comportamentos excepcionais, melhor suporte para desenvolvimento incremental e melhor reutilização.

Da apreciação dos trabalhos citados nesta seção, extraiu-se importantes conceitos e ideias que estruturam o estudo do tratamento de exceções através da utilização da POA. Conforme pode ser notado, a utilização de aspectos para identificar, armazenar dados e tratar exceções constitui-se em algo totalmente válido, tendo em vista os grandes e importantes ganhos que a utilização dessa tecnologia proporciona em todo o ciclo de vida dos sistemas.

### 3.4 Análise dos Trabalhos Relacionados

Cada estudo, pesquisa ou trabalho disposto e analisado neste capítulo possui sua referida importância e contribuição para a concretização deste trabalho. Contudo, fez-se necessário uma análise mais profunda de alguns desses materiais, observando uma maior similaridade desses com as soluções apresentadas no nosso trabalho.

Dessa maneira, foram analisados os trabalhos de Artaiam e Senivongse (2008), Filippetto e Callegari (2006), Godse, Bellur e Sonar (2010), Haiteng, Zhiqing e Hong (2011), Lippert e Lopes (2000), Michlmayr et al. (2009), Moser, Rosenberg e Dustdar (2008), Muller et al. (2012), Souza et al. (2011), Wetzstein et al. (2009) e Rodrigues (2007). Para tanto observou-se os seguintes atributos: *Tipo de Arquitetura* estudada, ou seja, se o trabalho foca-se em determinado tipo arquitetural ou utiliza-se de elementos comuns a toda arquitetura, e.g., classes e funções; *Solução Acooplável*, conferindo se a solução apresentada é “plugável” ao código referente ao negócio do sistema ou se está entrelaçada a ele; *Solução Extensível*, isto é, se a solução disposta pode ser utilizada em outras aplicações diferentes

das que foram estudo de caso do trabalho; *Aplicação Prática*, referindo-se ao fato do trabalho apresentar um exemplo concreto da sua solução; *Atributos Estudados*, mostrando os atributos de qualidade alvos de cada estudo; *Descrição da Monitoração*, inferindo se a monitoração vista nos trabalhos é descrita em detalhes; e, por fim, *Utilização de POA*, exibindo se os trabalhos utilizam a programação orientada a aspectos em seus estudos.

A síntese dessa análise pode ser vista na Tabela 2, bem como as respostas aos referidos requisitos visando a solução propostas nesta dissertação. No conjunto dos onze trabalhos apontados na referida tabela, em relação ao Tipo de Arquitetura, oito deles focam-se na SOA, enquanto três voltam-se aos elementos arquiteturais comuns a diversos estilos de arquitetura de software. Percebe-se que todos os estudos apresentam soluções acopláveis, isto é, que são inseridas nos sistemas sem interferir na codificação de suas funcionalidades.

**Tabela 2 – Análise dos Trabalhos Relacionados**

<b>Elementos Trabalhos</b>	<b>Tipo de Arquitetura</b>	<b>Solução Acoplável</b>	<b>Solução Extensível</b>	<b>Aplicação Prática</b>	<b>Atributos Estudados</b>	<b>Descrição da Monitoração</b>	<b>Utilização de POA</b>
Artaim e Senivongse (2008)	SOA	SIM	SIM	SIM	QoS	SIM	NÃO
Filippetto e Callegari (2006)	Arquitetura do Software	SIM	NÃO	SIM	ISO/IEC 9126 Parcial	Parcial	SIM
Godse; Bellur e Sonar (2010)	SOA	SIM	SIM	NÃO	QoS	SIM	NÃO
HAITENG; ZHIQING e HONG (2011)	SOA	SIM	SIM	NÃO	QoS	Parcial	SIM
Lippert e Lopes (2000)	Arquitetura do Software	SIM	NÃO	SIM	ISO/IEC 9126 Parcial	Parcial	SIM
Michlmayr et al. (2009)	SOA	SIM	NÃO	SIM	QoS	SIM	NÃO
Moser, Rosenberg e Dustdar (2008)	SOA	SIM	Parcial	Parcial	QoS	Parcial	NÃO
Muller et al. (2012)	SOA	SIM	SIM	NÃO	QoS	Parcial	NÃO
Souza et al. (2011)	SOA	SIM	SIM	SIM	QoS	Parcial	NÃO
Wetzstein et al. (2009)	SOA	SIM	SIM	Parcial	QoS	Parcial	NÃO
Rodrigues (2007)	Arquitetura do Software	SIM	NÃO	SIM	ISO/IEC 9126 Parcial	Parcial	SIM
<b>Sol. Proposta</b>	Arquitetura do Software	SIM	SIM	SIM	ISO/IEC 9126	SIM	SIM

Fonte: Elaborada pelo autor

Sobre a questão de apresentarem ou não Soluções Extensíveis, seis trabalhos mostram aplicações que podem ser utilizadas além dos estudos de caso realizados, enquanto um pode ser parcialmente reutilizado mediante uma série de adaptações e quatro voltam-se apenas aos sistemas alvo dos experimentos de cada trabalho. Em relação ao quesito Aplicação Prática, constata-se que seis trabalhos apresentam aplicações práticas para todas as soluções propostas nesses, dois exibem práticas apenas para algumas das soluções que propõem e três não mostram aplicações para suas soluções.

Ainda analisando a tabela apresentada, destacam-se os Atributos Estudados em cada um dos trabalhos. As oito pesquisas que se focam em SOA toma como base os atributos

QoS (disponibilidade, acessibilidade, desempenho, confiabilidade, segurança e regulamentação). Por outro lado, três baseiam-se nos atributos dispostos na norma ISO/IEC 9126 (funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade), ou melhor, o foco nesses atributos é realizado de forma parcial por esses trabalhos. A Descrição da Monitoração desses atributos é realizada totalmente em três trabalhos e parcialmente em oito deles. Por fim, vê-se que quatro pesquisas utilizam a POA em seus estudos e soluções e sete fazem o uso das técnicas nativas das linguagem de programação.

Na última linha da Tabela 2 é disposto os requisitos, perante os elementos salientados, referentes a solução discutida neste trabalho. Em suma, ela observará os elementos arquiteturais como um todo; apresentará uma solução acoplável e extensível; serão analisados todos os atributos da norma ISO/IEC 9126 que terão suas questões de monitoração integralmente descritas, contando com a disposição de uma aplicação prática idealizada por meio da POA.

## 4 MONITORAÇÃO DE REQUISITOS DE QUALIDADE

Este capítulo apresenta um estudo sistemático dos atributos de qualidade definidos na norma ISO/IEC 9126 (funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade), objetivando a análise e concepção de formas de monitoração dos referidos requisitos.

### 4.1 Requisitos de Qualidade

Conforme destacado no Capítulo 2, há uma relação direta entre a arquitetura de um software e seus requisitos, sejam eles funcionais ou não-funcionais, possuindo, essa, forte influência em todo o ciclo de vida do sistema. Devido a sua tamanha importância, a arquitetura deve ser projetada com cuidado por profissionais capacitados para tanto, tendo como base o documento de requisitos outrora concebido.

É sabido que os RF são os grandes definidores dos modelos arquiteturais de determinado sistema de software, contudo os RNF ou atributos de qualidade carecem de uma atenção redobrada por parte de arquitetos e desenvolvedores. Por, geralmente, estarem espalhados por diversos módulos do sistema, os mencionados atributos podem concorrer com as funcionalidades, causando prejuízos não só na legibilidade, mas na operabilidade e manutenibilidade do software.

Os atributos de qualidade discutidos neste trabalho correspondem àqueles advindos da norma ISO/IEC 9126. A seguir os seis atributos da supracitada norma serão descritos, indicando-se ainda, para cada um deles, *onde* poderá ser monitorado, *qual medida* deve ser observada e *como* realizar esta monitoração. Logo, uma satisfatória base de conhecimento acerca da aferição de atributos qualidade será formulada, dispondo-se de árvores de decisão ou diagramas de influência que relacionam os requisitos não-funcionais às variabilidades de monitoração, compreendendo, portanto, um modelo de decisão que pode ser utilizado para uma gama de sistemas.

### 4.2 Funcionalidade

Em um sistema de software, a característica Funcionalidade refere-se a existência de funções que atendam às necessidades explícitas ou implícitas do software e suas propriedades específicas, quando esse estiver sendo utilizado sob determinadas condições. A subdivisão desse atributo inclui: adequação, acurácia, interoperabilidade, segurança de acesso e conformidade (ISO, 2001). Dessa forma, compreende-se que a funcionalidade de um software diz respeito ao conjunto de características relativas aos requisitos funcionais previamente projetados e dizer que um sistema atende o atributo de qualidade funci-

onalidade é o mesmo que inferir que aquele fornece uma adaptação funcional concisa, satisfazendo às necessidades dos usuários.

Observando a Funcionalidade sobre o enfoque da monitoração, percebe-se que os principais pontos a serem monitorados estão concentrados nos procedimentos e funções de determinado sistema-alvo. Esta constatação advém do fato de que os RF estão fortemente relacionados com os métodos que deverão ser implementados utilizando-se alguma linguagem de programação. Por exemplo, em um sistema bancário, alguns dos possíveis requisitos funcionais poderiam ser:

- o software deve possibilitar o cálculo dos rendimentos da aplicação.
- o software deve emitir relatório das movimentações na conta corrente.
- o software deve oferecer a utilização do crédito especial quando a conta estiver com o saldo zerado.
- os usuários devem poder sacar, depositar e realizar transferências.

Assim, para cada uma das quatro funcionalidades enumeradas, ao menos, uma função ou procedimento deverá ser criado. No primeiro item, deve-se criar um procedimento que realize cálculo dos rendimentos; no segundo item, uma função que busque o histórico de movimentações de uma conta corrente e emita um relatório; no terceiro item, outra função que retorne a oferta de crédito especial quando o saldo for igual a zero; e no quarto item, procedimentos ou funções referentes aos saques, depósitos e transferências. Por meio desse simples exemplo, reforça-se a proposição de que funcionalidade está ligada à concepção de métodos. Destaca-se que o termo método é utilizado no contexto da POO, sendo o nome dado a um procedimento ou a uma função, ao lidar com a programação de classes.

Conforme destacado na Seção 4.1 deste capítulo, serão respondidas três questões - onde monitorar, o que monitorar e como monitorar - para cada um dos seis atributos da norma ISO/IEC 9126. Para o requisito funcionalidade, já se tem a resposta para a primeira questão, isto é, a monitoração deve ocorrer nas funções e/ou procedimentos do sistema a ser verificado, tomando como base, especificamente, os conceitos de Função de Transação e Função de Dados.

As Funções de Transação têm por objetivo receber ou fornecer dados ao usuário, passando-os pela fronteira da aplicação a fim de concretizar as metas propostas para o software. Conceitualmente, essas funções representam a funcionalidade que é fornecida ao usuário para o processamento de dados por uma aplicação, e.g., cadastrar, excluir, editar e listar usuários. Essa classificação de funções subdivide-se em: Entrada Externa (EE), que processa dados ou informações de controle recebidos de fora da fronteira da aplicação, mantendo arquivos lógicos ou modificando o comportamento do sistema; Saída Externa (SE), que envia dados ou informações de controle para fora da fronteira da aplicação, apresentando-os ao usuário após a realização de cálculos internos; e Consulta Externa

(CE), que também envia dados ou informações de controle para fora da fronteira da aplicação, sem a realização de cálculos, ou seja, há apenas uma recuperação simples (CAMPOS, 2011; VAZQUEZ; SIMÕES; ALBERT, 2013).

As Funções de Dados possuem como finalidade o armazenamento dos dados processados pela função de transação, representando, conceitualmente, as funcionalidades fornecidas aos usuários para atender requisitos internos e externos referentes aos dados. Essas funções são classificadas em Arquivos Lógicos Internos (ALI), ou seja, um grupo de dados ou informações de controle que é mantido na fronteira da aplicação; e Arquivos de Interface Externa (AIE), isto é, um grupo de dados ou informações de controle fora da fronteira da aplicação, mas referenciado por essa (VAZQUEZ; SIMÕES; ALBERT, 2013). Destaca-se que um grupo de dados é o conjunto de todos e somente os dados necessários para a caracterização de elementos da entidade que os representa (CAMPOS, 2011). Por exemplo: nome, CPF, endereço e número da conta corresponde a um grupo de dados que caracteriza o usuário de um sistema bancário.

Após essas conceituações, o próximo passo é definir o que deverá ser monitorado nas funções e procedimentos, ou melhor, quais as medições deverão ser realizadas. Deste modo, infere-se que a análise dos métodos de uma aplicação deve estar baseada em quatro pontos:

1. Grau de precisão do software em relação às funcionalidades projetadas a ele.
2. Interação com um ou mais sistemas especificados.
3. Processamento, geração e armazenamento de informações sobre as funções dos sistemas.
4. Proteção das informações.

O ponto número 1 determina que uma das análises a serem feitas corresponde à medição do grau de precisão do software em relação às funcionalidades preestabelecidas, aquelas presentes no documento de requisitos. Essa monitoração é muito subjetiva e, geralmente, realizada a partir da execução do software e posterior comparação entre os testes realizados e o que se esperava da efetivação de cada funcionalidade. O ponto 2 salienta a observação da interação entre sistemas especificados, já que, em muitos casos, os softwares não operam isoladamente, mas sim recebendo ou fornecendo dados a outras aplicações. O ponto 3 destaca o processamento, geração e armazenamento de informações acerca das funções do sistema, isto é, o registro das operações realizadas e das ocorrências de erros e exceções. O último ponto aborda a proteção das informações, ou seja, toda a utilização do software deve ocorrer mediante a salvaguarda das informações inseridas, retornadas e armazenadas nos arquivos lógicos do sistema.

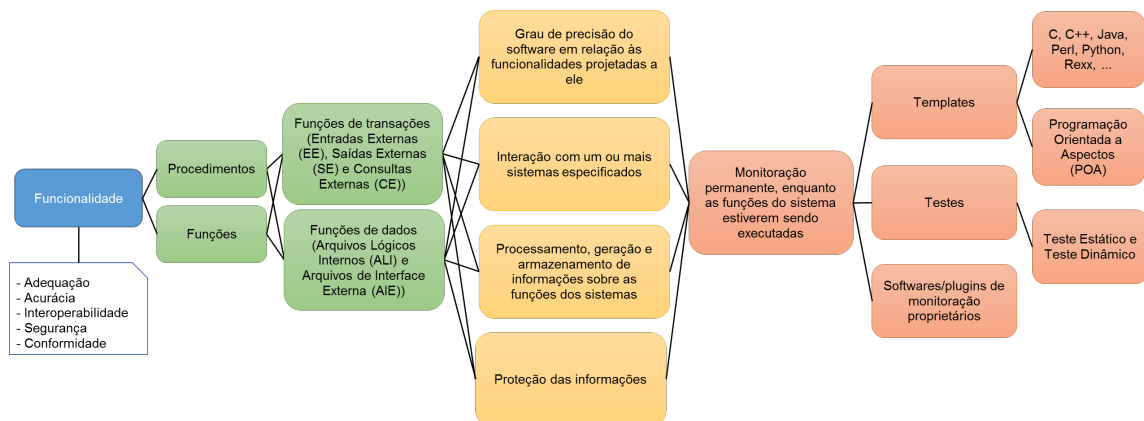
Ainda em relação ao atributo funcionalidade, resta definir como realizar a monitoração dos pontos anteriormente abordados. A princípio, é fundamental inferir que para se



monitorar a funcionalidade de um software é imprescindível que a análise dos métodos ocorra permanentemente: enquanto as funções do sistema estiverem sendo executadas, a monitoração deve estar ativa. Observando esse pressuposto, a referida monitoração pode ser realizada através do uso de templates, desenvolvidos nas mais diversas linguagens de programação (C, C++, Java, Perl, Python, Rexx, etc) ou com a programação orientada a aspectos; testes, estáticos e dinâmicos; e softwares/plugins de monitoração proprietários.

Diante das informações abordadas nesta seção, chega-se à árvore de decisão exposta na Figura 14. Salienta-se que todas as árvores apresentadas neste capítulo possuirão a seguinte definição em relação às cores de preenchimento dos retângulos arredondados, que exibem as questões de monitoração: Azul - nome do atributo; Verde - local de monitoração; Amarelo - ponto a ser monitorado; e Salmão - formas de monitoração. Além do mais, as árvores dispostas nesta e nas próximas seções estão disponíveis em tamanho ampliado no Apêndice A deste documento.

**Figura 14 – Árvore de Decisão - Monitoração do Atributo Funcionalidade**



Fonte: Elaborada pelo autor

### 4.3 Confiabilidade

A Confiabilidade relaciona-se à capacidade de um software manter um nível de desempenho adequado, mediante determinadas condições e tempo. Ela subdivide-se em: maturidade, tolerância a falhas, recuperabilidade e conformidade (ISO, 2001).

Embora a definição de confiabilidade seja clara e apresente convergência com outras conceituações encontradas na literatura - como a da EOQC (Organização Européia para o Controle de Qualidade) que a define como a medida da habilidade de um produto operar com sucesso, ao ser solicitado, por um período de tempo pré-determinado, e sob condições de utilização específicas (BREWER, 1972); e a de Hnatek (2003) que a estabelece como o desempenho em relação a requisitos por um período de tempo - o termo, muitas vezes, mostra-se ambíguo e mal entendido pelo senso comum. Assim, entende-se que a medida de confiabilidade associa-se aos seguintes fatores (FILHO, 2011):

- o desempenho específico esperado, ou seja, o resultado das operações executadas no sistema, relacionadas integralmente às expectativas do cliente e, comumente, abordadas nas especificações do produto de software.
- as condições de aplicação, resumindo-se às condições do ambiente no qual o produto é utilizado, abrangendo o ambiente em si e os usuários.
- a variável tempo de utilização do software, envolvendo todos os aspectos temporais ligados à aplicação.

Neste ponto é importante mencionar a chamada Teoria da Confiabilidade que reúne vários campos do conhecimento, lidando com a interdisciplinaridade ao empregar objetos oriundos da probabilidade, estatística e modelagem estocástica, que são associados à engenharia na fase do processo de projeto e no entendimento científico dos mecanismos de falha, a fim de analisar os vários aspectos da confiabilidade, sobretudo aqueles relacionados ao aparecimento de falhas nos componentes do sistema (MURTHY; RAUSAND; ØSTERÅS, 2008).

A tolerância a falhas e recuperabilidade são dois aspectos de extrema importância em todo e qualquer sistema, seja ele de pequeno, médio ou grande porte. A falha diz respeito a má execução ou não execução de funções pretendidas para uma aplicação, sob condições operacionais previstas, reduzindo parcial ou totalmente a operação do sistema como um todo ou da tarefa que estava sendo realizada - a depender das consequências advindas de uma falha, essa pode ser classificada como mais ou menos grave/relevante. Neste contexto, o sistema deve possuir maturidade, sendo capaz de evitar falhas decorrentes de defeitos no software; tolerar falhas, mantendo seu funcionamento adequado ainda que ocorram defeitos nele ou em suas interfaces externas; e recuperar-se, restabelecendo os níveis de desempenho e recuperando os possíveis dados perdidos (ISO, 2001).

É importante definir que a causa da falha está ligada a circunstâncias provenientes da fase de *projeto*, por meio da identificação inadequada das necessidades ou ainda no uso inapropriado ou deficiente da engenharia para planejar a aplicação; *desenvolvimento*, através de processos deficientes ou implementação incorreta de componentes; *utilização do produto de software*, pelo uso incorreto, manutenção inadequada e uma série de outros maus usos; e ainda devido a *falhas físicas* que estão dissociadas ao software em si, referindo-se a integridade física do hardware que pode sofrer danos e acarretar falhas na execução da aplicação que este ampara.

Percebe-se que as possíveis falhas de um sistema serão notadas no momento em que o software estiver em execução, relacionando-se aos seus procedimentos e funções. Por diversos fatores poderão ocorrer erros pontuais (erros simples), dificultando ou não realizando determinada tarefa ou mesmo afetando apenas um componente; e também generalizados (erros múltiplos), ocasionando a interrupção brusca do sistema que é provocada por falhas que culminam em erros simultâneos em mais de um componente da aplicação.

Dessa forma, a monitoração deve analisar a verificação dos métodos responsáveis pelas funcionalidades.

A observação dos métodos dos softwares para aferição da confiabilidade se dará pela verificação das medidas de confiabilidade que são: taxa de ocorrência de defeito, MTTF, MTTR e MTBF. A Tabela 3 define, de maneira sintética, cada uma dessas medidas.

**Tabela 3 – Definição das medidas de confiabilidade**

<b>Medida de Confiabilidade</b>	<b>Definição</b>	<b>Questão</b>
Taxa de defeito - <i>failure rate</i>	Número esperado de defeitos em um dado período de tempo	Com que frequência ocorrem defeitos?
MTTF - <i>mean time to failure</i>	Tempo esperado até a primeira ocorrência de defeito	Qual o tempo até o primeiro defeito?
MTTR - <i>mean time to repair</i>	Tempo médio para reparo do sistema	Qual o tempo gasto para reparar cada defeito?
MTBF - <i>mean time between failure</i>	Tempo médio entre as defeitos do sistema	Qual o tempo entre um defeito e outro?

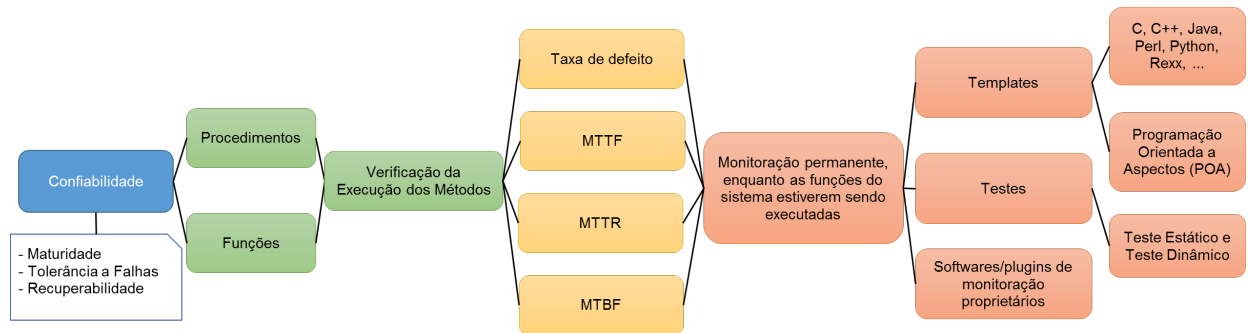
Fonte: Adaptado de WEBER, T. S. (2002)

Ressalta-se que quanto maior for o MTTR, pior será o sistema em termos de disponibilidade e quanto maior o MTBF, mais confiável será o produto de software. Outro ponto a destacar corresponde a definição dos conceitos de falha, erro e defeito. O desvio da especificação do sistema é definido como um defeito (*failure*), ou seja, uma não conformidade que faz parte do produto, estando implementada em seu código. Um erro (*error*) é a manifestação concreta de um defeito ou de uma falha em determinada aplicação, culminando em resultados não esperados ou incorretos. Finalmente, a falha ou falta (*fault*) é a resultante da execução de um defeito no código do software, ocasionando um comportamento funcional diferente daquele que é esperado pelo usuário (WEBER, 2002).

Enfim, igualmente ao atributo funcionalidade, a monitoração da confiabilidade de um software é impreterivelmente realizada pela análise permanente dos métodos da aplicação, isto é, enquanto as funções do sistema estão sendo executadas, a monitoração deve se fazer ativa. Dessa maneira, tal monitoração pode ser realizada com a utilização de templates, implementados nas linguagens de programação convencionais e ainda com a POA; testes, que podem ser estáticos e/ou dinâmicos; e ainda através softwares/plugins de monitoração proprietários.

A Figura 15 exhibe a árvore de decisão com as questões referentes à monitoração do atributo confiabilidade, tratadas nesta seção. Atenta-se ao significado das cores de preenchimento dos retângulos arredondados, remetido na Seção 4.3.

Figura 15 – Árvore de Decisão - Monitoração do Atributo Confiabilidade



Fonte: Elaborada pelo autor

#### 4.4 Usabilidade

A Usabilidade refere-se ao esforço na utilização e no aprendizado de certo software, bem como o julgamento individual da sua utilização por um conjunto de usuários. Assim, abarca a forma como o sistema é compreendido, aprendido, operado e atraente ao usuário, quando submetido a condições especificadas. Esta característica envolve a inteligibilidade, apreensibilidade, operacionalidade, atratividade e conformidade (ISO, 2001).

Tratar do atributo usabilidade é referir-se à qualidade de uso que determinado produto de software possui, definida ou mensurada considerando todo o contexto no qual esse está implantado e é operado. Logo, a depender do mencionado contexto, o sistema pode proporcionar boa usabilidade para um usuário experiente, porém péssima para um iniciante; ou ainda, poderá ser facilmente operado quando usado esporadicamente, mas difícil de ser conduzido se for utilizado com frequência (CYBIS, 2003).

Nielsen (1993) apresenta cinco fatores, advindos da parte 11 da norma ISO 9241 (ISO, 1998), determinantes para que um sistema possua boa usabilidade, compondo a natureza multidimensional desse atributo. Os fatores ou requisitos são explanados por Dias (2003), conforme segue:

1. Facilidade de aprendizado - o sistema deve ser fácil de aprender, de tal forma que o usuário consiga rapidamente e sem dificuldade explorar as funcionalidades e realizar suas tarefas.
2. Eficiência de uso - o sistema deve ser eficiente a tal ponto de permita ao usuário, já inteirado da aplicação, interagir com ela, atingindo altos níveis de produtividade na realização de suas tarefas.
3. Facilidade de memorização - após um certo período sem utilizá-lo, o usuário esporádico deve ser capaz de retornar ao sistema e realizar suas tarefas sem que seja necessário reaprender a interagir com ele.

4. Baixa taxa de erros - em um sistema com baixa taxa de erros, o usuário é capaz de utilizar as funcionalidades desse e realizar tarefas sem maiores transtornos, recuperando erros, caso ocorram.
5. Satisfação subjetiva - o usuário considera agradável e amigável a interação com o sistema, sentindo-se subjetivamente satisfeito com ele.

Dessa forma, a usabilidade, diferentemente do que é imaginando por muitos desenvolvedores, ultrapassa a fronteira da simples facilidade na utilização do sistema, devendo atenuar o tempo necessário para a aprendizagem e uso das funções da aplicação; a irritação dos usuários, quando se veem incapazes de navegar nos programas; a subutilização de recursos; as possibilidades de erros na operação; e o baixo rendimento do trabalho (MORAES, 2002). Destaca-se que os problemas decorrentes da má usabilidade, diagnosticada pelas experiências negativas no uso de interfaces deficientes, pode gerar sérios aborrecimentos e até mesmo frustrações que levarão os usuários a se sentirem diminuídos por não conseguirem realizar tarefas que, teoricamente, outros conseguem (GONÇALVES, 2008).

De acordo com Cybis (2003), desenvolver sistemas com boa usabilidade pode criar um impacto positivo na eficiência, eficácia e produtividade do produto de software como um todo. A partir deste benefício, é possível proporcionar ao usuário meios para atingir seus objetivos com mais facilidade, menos esforço e ainda maior satisfação. O autor aponta que interfaces difíceis de serem utilizadas, comumente, aumentam a carga de trabalho do utilizador do software e resultam em consequências negativas que vão desde a simples resistência ao uso, passando pela subutilização, até, muitas vezes, ao abandono do sistema, causando a mortalidade precoce de aplicações que representam um alto investimento, culminando em prejuízos podem ser bastante elevados.

Como pode ser percebido, medir ou mesmo monitorar<sup>4</sup> a usabilidade em um sistema de software é algo que requer processos, geralmente, subjetivos, já que considerarão a opinião dos usuários perante a utilização das funcionalidades de um programa. Partindo dos pressupostos da norma ISO 9241 (ISO, 1998), nota-se que a medição do atributo usabilidade será feita através de análises que terão como ponto de atuação a utilização/execução do sistema pelos seus usuários finais. Serão observados critérios, como: a interação observada entre usuário e aplicação, eficiência e eficácia do software e a satisfação do usuário diante da utilização das funcionalidades do sistema.

Neste cenário, há diversas formas de realizar a avaliação da usabilidade no design de produtos de softwares. Dentre as quais destacam-se alguns Métodos Baseados em Usuários (CATECATI et al., 2010):

---

<sup>4</sup> A monitoração da Usabilidade será possível quando uma funcionalidade implementada e introduzida em uma interface interativa for imediatamente testada. Quando os testes forem realizados após a total implementação do sistema, o termo a ser utilizado é medição ou avaliação de usabilidade.

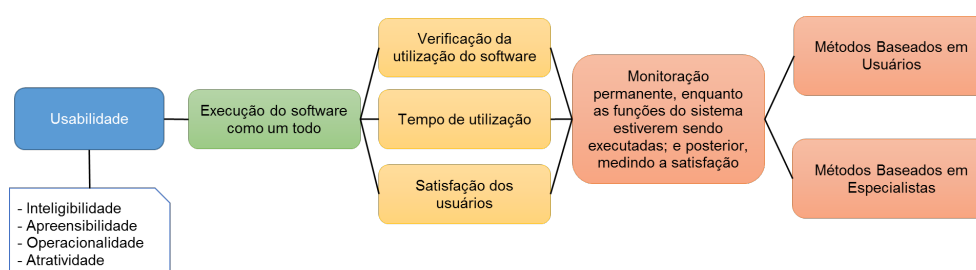
- Método Coaching - consiste na interação explícita entre o observador e o usuário durante a realização de determinada tarefa, podendo esse perguntar àquele - indivíduo que aplica o teste - acerca das suas dúvidas relativas ao uso do sistema.
- Aprendizagem por Co-descoberta - consiste na exploração da aplicação por dois usuários, ao mesmo tempo, que tentam descobrir como realizar determinada tarefa, enquanto são observados.
- Método de Ensino - consiste na seleção de usuários que irão interagir com o sistema, familiarizando-se com o mesmo de forma que consigam realizar tarefas pré-determinadas. Após este período de familiarização, cada usuário “experiente” será responsável por apresentar o sistema ao usuário novato, que o desconhece por completo, ensinando-o a realizar o conjunto de tarefas outrora aprendidos.
- Método Shadowing - consiste na explanação, para um avaliador, de um usuário experiente com o sistema sobre o comportamento dos demais usuários na realização de tarefas.
- Teste Remoto - consiste na aplicação de qualquer teste de usabilidade no qual o observador e o participante estão em diferentes locais ou tempo, geralmente realizado por meio da internet.
- Grupos de Foco - consiste em uma reunião voltada à discussão sobre alguma questão em foco, sendo abordadas, e.g., as experiências de uso e problemas encontrados na utilização do produto de software.
- Entrevistas - consiste na aplicação de questões formuladas com base nas principais áreas de interesse, nas quais determinadas informações necessitam ser levantadas.
- Registro do Uso Real - consiste na coleta automática e, geralmente, em tempo real de dados referentes à interação do usuário com o sistema, observando dados como o tempo de realização de certa tarefa.
- Estudo de Campo - consiste na análise do usuário, em seu ambiente próprio, e das suas interações com o sistema, observando detalhadamente uma série de aspectos que somente podem ser identificados no contexto do usuário.
- Questionários - consiste na coleta de dados referentes a um grupo representativo da população de usuários, sendo formulados através de questionamentos que podem ser fechados (quantitativos), abertos (qualitativos) ou mistos (quali-quantitativos).

(CATECATI et al., 2010) apresentam ainda os Métodos Baseados em Especialistas, representados por: *Passo a Passo Cognitivo*: no qual um avaliador ou um grupo de avaliadores, composto por especialistas em usabilidade, ergonomia, design, engenharia,

marketing, dentre outros, analisam a interface do sistema juntamente com o usuário, realizando uma série de testes destinados à avaliação da facilidade de entendimento e de aprendizagem; *Avaliação Heurística*: na qual um conjunto mínimo de cinco avaliadores distintos, como os especialistas citados anteriormente, sem envolvimento com o projeto, avaliarão o sistema; e *Inspeção de Características e Funcionalidades*: realizada por meio de uma técnica que analisa apenas as características e aspectos utilizados para a efetivação de uma determinada tarefa, observando critérios como a compreensibilidade, viabilidade e a eficácia.

Enfim, é claro que cada método possui uma característica ímpar que trará formas adequadas de analisar a usabilidade de um produto de software. Não há como determinar que um tipo de teste específico é o melhor para testar esse ou aquele sistema, cabe uma análise do projeto antes da seleção de um método de avaliação de usabilidade, que deve trabalhar permanentemente sobre a execução do programa. Na Figura 16 é ilustrado o processo de avaliação/monitoração do atributo de qualidade discutido nesta seção.

**Figura 16 – Árvore de Decisão - Monitoração do Atributo Usabilidade**



Fonte: Elaborada pelo autor

#### 4.5 Eficiência

Falar em Eficiência é observar a relação que existe entre o desempenho - que deve ser apropriado - de um produto de software e a quantidade de recursos utilizados para a execução de suas funcionalidades, sob condições específicas. Esse atributo possui como subcaracterísticas o comportamento em relação ao tempo, o comportamento em relação aos recursos e a conformidade (ISO, 2001).

Desse modo, monitorar a Eficiência de um produto de software é analisar o seu comportamento ou ainda o seu desempenho, tomando como ponto de estudo o tempo e os recursos que são consumidos durante a realização das diversas tarefas que um sistema se propõe a realizar, objetivando descobrir eventuais pontos de gargalo (do inglês *bottle-necks*) de processamento - tudo o que impede que o sistema apresente maior vazão (Do inglês *throughput*), de processamento - , predizendo, assim, o desempenho da aplicação em seu ambiente real.

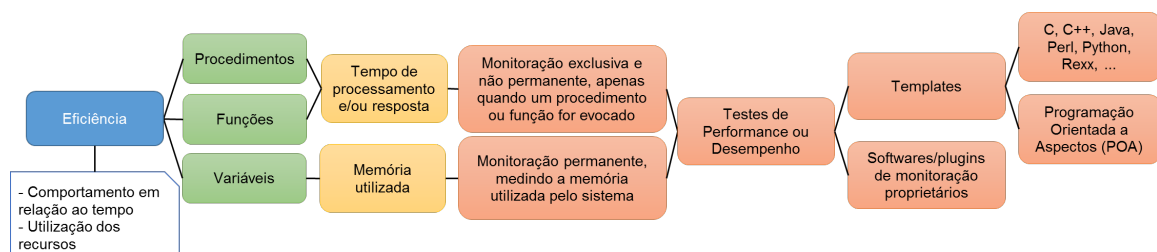
Neste sentido destacam-se os Testes de Performance ou Desempenho, fundamentais para verificar a eficiência de um sistema, proporcionando o planejamento de correções e melhorias com o fim de atender a demandas atuais e futuras. De forma geral, esses testes estarão voltados, sobretudo, à avaliação da capacidade de resposta de uma aplicação, identificando uma série de questões provenientes do processamento das tarefas. Nota-se a existência de três tipos comuns de testes de desempenho, eles são: testes de carga, que testam o desempenho do sistema analisando uma carga de usuários simultâneos reais; testes de estresse, que testam o desempenho considerando o número máximo de usuários simultâneos que esse pode suportar; e testes de volume, que testam a quantidade de dados que o sistema pode gerir (PRESSMAN, 2011). Há ainda os testes de estabilidade que testam se, com o tempo, o sistema degrada seu desempenho.

Assim, a monitoração da eficiência estará ligada à análise do tempo de resposta das requisições realizadas pelos procedimentos e funções, além do processamento e memória utilizada durante a realização de uma tarefa. A medição desse atributo de qualidade deverá ser realizada parte exclusiva, parte permanente. A observação dos tempos de resposta, necessariamente, deverá ser efetivada de forma isolada, pois, a monitoração de um outro atributo em paralelo poderia comprometer os resultados dessa análise. Por outro lado, a verificação da memória consumida poderá ser realizada de modo permanente, conferindo os recurso alocados pelo sistema.

Em suma, a monitoração da Eficiência de um produto de software é feita através da utilização de Testes de Performance ou Desempenho, idealizados por templates, implementados nas linguagens de programação convencionais ou ainda contando com programação orientada a aspectos. Há ainda a possibilidade da utilização de softwares/plugins de monitoração proprietários e também da execução de testes analíticos, assim como na avaliação da Usabilidade, com questionários, entrevistas e grupos de foco.

A Figura 17 sintetiza a monitoração do atributo de qualidade eficiência, levando em consideração o mesmo padrão de cores definido na Seção 4.2.

**Figura 17 – Árvore de Decisão - Monitoração do Atributo Eficiência**



Fonte: Elaborada pelo autor



## 4.6 Manutenibilidade

A Manutenibilidade corresponde ao esforço necessário para serem realizadas modificações específicas no sistema, bem como para localizar e reparar eventuais erros; em linhas gerais determina a capacidade do software de ser modificado. A subdivisão das características desse atributo corresponde a: analisabilidade, modificabilidade, estabilidade, testabilidade e conformidade (ISO, 2001).

Nesta seção é importante destacar o conteúdo da norma ISO/IEC 12207. Em síntese, a referida norma apresenta os *Processos do Ciclo de Vida do Software*, tendo como objetivo principal fornecer uma estrutura comum para que os *stakeholders* utilizem uma mesma linguagem ao lidar com o desenvolvimento de sistemas. Assim, é estabelecido o ciclo de vida comum a todos os produtos de software que é composto pelos seguintes processos: (i) Aquisição; (ii) Fornecimento; (iii) Desenvolvimento; (iv) Operação; e (v) Manutenção (ISO, 1995).

Neste contexto, a manutenibilidade estaria inserida no último processo do ciclo de vida do software, sendo responsável por modificações tanto no código das aplicações quanto na documentação associada, seja por conta de um problema, seja pela necessidade de melhorias ou adaptações. Conforme salienta o *Institute of Electrical and Electronics Engineers* (IEEE) (IEEE, 1990), a manutenção ocorre pela análise do processo de modificação de um sistema ou de componentes, visando corrigir falhas, melhorar a performance ou outros atributos, ou realizar adaptações devido às mudanças de ambiente após sua entrega.

Contudo, Pigoski (1996) infere que é equivocada a ideia de o processo de manutenção iniciar apenas após o produto já estar pronto e em utilização. Logo, o autor sugere que o processo mencionado inicie juntamente com o desenvolvimento do software, ou seja, no terceiro ponto do seu ciclo de vida, com especialistas em manutenção atuando ativamente de forma a se obter um produto com arquitetura adequada para que os futuros reparos sejam rápidos, precisos e menos custosos.

Alguns fatores influenciam na manutenibilidade das aplicações, esses estão ligados a qualquer processo do ciclo de vida definido pela norma ISO/IEC 12207. Os principais influenciadores do grau de manutenção dos produtos de software são (BRUSAMOLIN, 2004):

- Documentação - pois à medida que a documentação ou as especificações de design do software estão indisponíveis ou são precárias, o mantenedor deverá investir muito tempo para a compreensão do produto antes mesmo de modificá-lo.
- Arquitetura - já que a criação de uma arquitetura, dividindo a aplicação em componentes concisos, bem como o investimento em arquitetura de software aprimora, entre outros atributos de qualidade, a manutenibilidade.

- Tecnologia - pois a tecnologia que será empregada na implementação do software será determinante para um bom grau de manutenibilidade, podendo oferecer meios para reduzir a degradação do código quando esse necessitar ser alterado.
- Código Fonte - pois a escolha da tecnologia e a aplicação dessa para a implementação do código fonte impactará na compreensibilidade do programa, que deve ser aumentada ao passo que se deseja diminuir os custos de manutenção.

Dessa forma, para avaliar a manutenibilidade é imprescindível observar os fatores que a influenciam e aplicá-los sob métricas de software ou, mais especificamente, métricas de manutenibilidade que se definem como medidas associadas ao processo ou ao produto de software em si, incluindo código fonte e documentação. Percebe-se que há dois tipos de métricas: as dinâmicas, coletadas através de medições realizadas no momento da execução do programa; e as estáticas, coletadas em análises feitas na documentação do projeto ou ainda no código fonte do software. Para mensurar a manutenibilidade, as métricas mais adequadas são as estáticas, já que a efetivação das medições será melhor realizada em cima dos artefatos já construídos.

Algumas das principais métricas de manutenibilidade são: Complexidade Ciclomática, Profundidade de Herança, Acoplamento da Classe e Linhas de Código.

A Complexidade Ciclomática (CC) é uma técnica desenvolvida por McCabe para mensurar a testabilidade e a manutenibilidade de um software, medindo o número de caminhos linearmente independentes de determinado método ou do programa como um todo. A fim de determinar todos os caminhos possíveis, o método a ter a complexidade medida será transformado em um grafo fortemente conectado que possui apenas uma entrada e saída única, sendo os nodos blocos sequenciais de código e as arestas decisões que causam uma ramificação (CUNHA, 2006). O valor da CC do grafo gerado pode ser calculado conforme a equação a seguir.

$$CC = E - N + 2$$

onde:

**CC** é a complexidade ciclomática;

**E** é o número de arestas do grafo;

**N** é número de nós do grafo; e

**2** refere-se ao número de entradas e saídas.

O resultado desta métrica é um único número que pode ser comparado com o resultado de outros métodos ou programas. (VANDOREN, 1997) apresenta escala para a medição da CC, avaliando o risco através do valor encontrado com a equação anteriormente explicitada. O parecer do autor pode ser visto na Tabela 4 que deixa claro que quanto mais

**Tabela 4 – Escala de complexidade ciclomática**

<b>Complexidade Ciclométrica</b>	<b>Avaliação do Risco</b>
1 - 10	programa simples, sem muito risco
11 - 20	programa mais complexo, risco moderado
21 - 50	programa de risco elevado
acima de 50	programa incompreensível, risco muito alto

Fonte: VANDOREN, E. (1997)

baixa for a complexidade ciclométrica, de mais fácil compreensão e, conseqüentemente, alteração será o software.

A Profundidade de Herança está relacionada a softwares implementados a partir da tecnologia da Orientação a Objetos (OO). De maneira geral, ela mede a quantidade de classes que é utilizada para a geração de uma nova classe. Observa-se a árvore de heranças gerada através da análise dos métodos e atributos herdados. Neste ponto infere-se que quanto maior a profundidade dessas árvores, ou seja, da herança, mais complexo e menos entendível será projeto; fato que reduz o reúso de código, aumenta a probabilidade de falhas e dificulta a manutenção do produto.

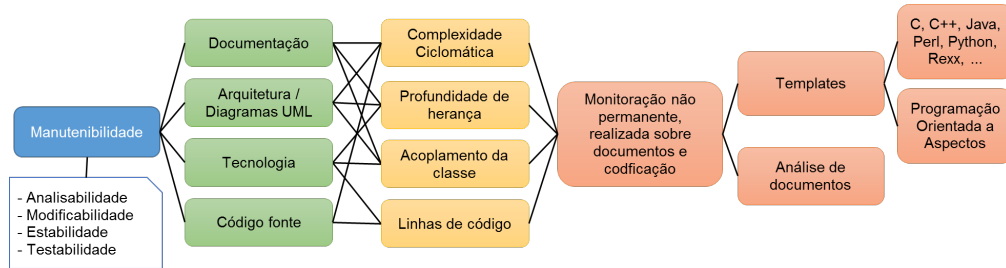
Similarmente à Profundidade de Herança, o Acoplamento da Classe também está ligado ao desenvolvimento orientado a objetos, trazendo à tona a ideia de dependência. Nos softwares que possuem alto grau de acoplamento entre suas classes, essas são altamente dependentes umas das outras. Essa situação prejudica a manutenção dos sistemas, pois quando se modifica uma classe, todas as outras que estão ligadas a ela, geralmente, devem ser alteradas para que não apresentem erros, criando uma “bola de neve” que pode inviabilizar toda e qualquer modificação. Assim, o acoplamento só é admissível em casos nos quais as classes não mudam, como aquelas nativas das linguagens e ambientes de programação.

Outra técnica que, embora primitiva, é bastante utilizada é a medição de Linhas de Código, observada diretamente em uma única medida. A análise das linhas de código pode ser realizada para prever a complexidade de um programa, a sintaxe dos métodos e variáveis, repetições desnecessárias de código, trechos de códigos e variáveis não utilizados e uma série de outras questões que, comumente, baseiam-se na compreensibilidade do código fonte, trazendo maior legibilidade que amparará futuras modificações.

Enfim, a monitoração ou a verificação do grau de manutenibilidade realizar-se-á por medições não permanentes que serão feitas na documentação do projeto e códigos dos produtos de software. Para a análise de aspectos referentes aos códigos fonte, pode-se fazer o uso de templates desenvolvidos nas linguagens de programação convencionais ou pela POA. Já a observação dos documentos, incluindo a arquitetura do sistema, será realizada por desenvolvedores ou mantenedores que poderão contribuir para a redução da complexidade e ilegibilidade da documentação e código, beneficiando, conseqüentemente,

a manutenibilidade. A Figura 18 ilustra o processo de verificação da manutenibilidade discutido nesta seção.

**Figura 18 – Árvore de Decisão - Monitoração do Atributo Manutenibilidade**



Fonte: Elaborada pelo autor

#### 4.7 Portabilidade

Esse atributo determina a característica responsável por mensurar a capacidade de um software ser transportado de um ambiente de hardware ou software para outro. Possui como subcaracterísticas a adaptabilidade, capacidade para ser instalado, coexistência, capacidade para substituir e conformidade (ISO, 2001).

O termo ambiente é utilizado para fazer referência tanto à plataforma de hardware quanto ao conjunto de softwares auxiliares com os quais um produto de software irá interagir, apresentando variadas configurações que geram um série de combinações possíveis. A nível de hardware, as principais variações observadas abarcam a ordem em que os bits e bytes são armazenados na memória, a quantidade de bits utilizada para representar os valores inteiros e os endereços de memória, limite superior e inferior dos tipos de variáveis, a forma de representar os números de ponto flutuante, dentre outras. A nível de software, as principais variações vistas relacionam-se aos Sistemas Operacionais (SO) e ainda à troca de informações ou compartilhamento de recursos entre diferentes aplicações (FILHO, 2005).

Braude (2005) salienta que os projetos de um produto de software devem ser idealizados de forma a pensar nas mudanças que o mesmo, possivelmente, enfrentará, já que raramente os requisitos (hardware e software) definidos no início do ciclo de vida da aplicação serão mantidos até o final do projeto. Desse modo, o autor sugere que a construção de sistemas deve buscar possibilidades de flexibilidade que possibilitarão, de forma satisfatória, a capacidade de execução em ambientes de tipos diferentes.

A flexibilidade aqui abordada diz respeito à portabilidade, uma série de fatores que determinam a capacidade do software ser transferido de um ambiente hardware e/ou software para outro. A norma ISO/IEC 9126 define a portabilidade a partir das subcaracterísticas citadas no início desta seção, em resumo, é destacada a capacidade do software de ser adaptado a diferentes ambientes especificados, sem o auxílio de aplicações

ou ações à parte para realizar a adaptação; capacidade do produto de software para ser instalado em um determinado ambiente, a partir dos meios que o próprio produto oferece; capacidade de o software coexistir com outros produtos de software independentes, compartilhando dados e recursos em um mesmo ambiente; e capacidade do produto de software de ser utilizado em substituição a outro software especificado (ISO, 2001).

Mooney (2011) determina que uma unidade de software, quando inserida em uma classe de ambientes especificados, possui alto grau de portabilidade quando o esforço necessário para transportá-la e adaptá-la para um ambiente, diverso do que ela estava inserida, é inferior ao esforço de desenvolvê-la novamente. Dessa forma, nota-se que o esforço sobre a adaptabilidade, capacidade para ser instalado, coexistência e capacidade para substituir determinará o grau de portabilidade que um software possui, sendo este atributo apresentado de forma gradual e não binária.

Assim como no atributo manutenibilidade, para medir a Portabilidade há de se valer de uma conjunto de métricas que visarão medições fáceis de serem realizadas e resultados igualmente fáceis de serem utilizados. Filho (2005), de forma clara, apresenta uma série de métricas para a análise da portabilidade em softwares. O autor faz a divisão entre métricas internas, aplicáveis a um produto de software não executável (e.g. documentação e código fonte), e métricas externas, usadas quando a unidade de software está em uso. As referidas métricas serão apresentadas a seguir.

### Métricas Internas

- Adaptabilidade ao ambiente de hardware: mede o quão adaptável é a unidade de software às mudanças no ambiente de hardware, definindo-se por  $\mathbf{AH} = \mathbf{A}/\mathbf{B}$  ( $\mathbf{A}$  - número de componentes que oferecem capacidade de executar em diferentes ambientes de hardware e  $\mathbf{B}$  - número de componentes que não oferecem essa capacidade).
- Adaptabilidade ao ambiente de software: mede o quão adaptável é a unidade de software às mudanças nos softwares que compõem o ambiente no qual está inserida, definindo-se por  $\mathbf{AS} = \mathbf{A}/\mathbf{B}$  ( $\mathbf{A}$  - número de componentes que oferecem capacidade de executar em diferentes ambientes de software e  $\mathbf{B}$  - número de componentes que não oferecem essa capacidade).
- Grau de portabilidade: mede a relação entre o esforço de modificar e o esforço de refazer cada componente da unidade de software a ser adaptável a um ambiente de hardware ou software, definindo-se por  $\mathbf{GP} = 1 - (\mathbf{A}/\mathbf{B})$  ( $\mathbf{A}$  - esforço necessário para adaptar o componente e  $\mathbf{B}$  - esforço requerido para refazer o componente).
- Esforço para instalar: mede o nível de esforço preciso para instalar o produto de software, definindo-se por  $\mathbf{EI} = \mathbf{A}/\mathbf{B}$  ( $\mathbf{A}$  - número de passos de instalação automáticos e  $\mathbf{B}$  - número total de passos de instalação necessários).

- Flexibilidade de instalação: mede a flexibilidade e customização no processo de instalação da unidade de software, definindo-se por  $\mathbf{FI} = \mathbf{A}/\mathbf{B}$  ( $\mathbf{A}$  - número de operações de instalação que admitem customização e  $\mathbf{B}$  - número total de operações de instalação necessárias).
- Disponibilidade de coexistência: mede a flexibilidade que o produto de software possui em compartilhar recursos com outros softwares sem causar prejuízos, sendo definida por  $\mathbf{DC} = \mathbf{A}/\mathbf{B}$  ( $\mathbf{A}$  - número de produtos previamente definidos com os quais a unidade de software pode coexistir sem problemas e  $\mathbf{B}$  - número de produtos no ambiente de produção com os quais o software deve coexistir sem problemas).
- Consistência de funcionalidades: mede a consistência da unidade de software em relação ao produto que está sendo substituído, definindo-se por  $\mathbf{CF} = \mathbf{A}/\mathbf{B}$  ( $\mathbf{A}$  - número de funcionalidades da unidade de software que produzem resultados semelhantes aos do software antigo e  $\mathbf{B}$  - número de funcionalidades do produto substituído).

### Métricas Externas

- Adaptabilidade ao ambiente de hardware: avalia a capacidade de auto-adaptação da unidade de software ao ambiente de hardware, oferecendo facilidades ao usuário do produto e definindo-se por  $\mathbf{AHe} = 1 - (\mathbf{A}/\mathbf{B})$  ( $\mathbf{A}$  - número de operações que não puderam ser executadas ou apresentaram resultados insatisfatórios devido às mudanças e  $\mathbf{B}$  - número de operações que foram executadas).
- Adaptabilidade ao ambiente de software: avalia a capacidade de auto-adaptação da unidade de software ao ambiente de hardware, oferecendo facilidades ao usuário do produto e definindo-se por  $\mathbf{ASe} = 1 - (\mathbf{A}/\mathbf{B})$  ( $\mathbf{A}$  - número de operações que não puderam ser executadas ou apresentaram resultados insatisfatórios devido às mudanças e  $\mathbf{B}$  - número de operações que foram executadas).
- Flexibilidade de instalação: avalia a simplicidade do processo de instalação da unidade de software, definindo-se por  $\mathbf{Fie} = \mathbf{A}/\mathbf{B}$  ( $\mathbf{A}$  - número de interações do usuário no processo de instalação a fim de adequar o produto ao ambiente e  $\mathbf{B}$  - número total de etapas necessárias à correta instalação da unidade de software).
- Coexistência apresentada: mede a frequência de restrições de uso ou erros encontrados pelo usuário durante a utilização do software e compartilhamento de recursos com outros produtos, sendo definida por  $\mathbf{CAe} = \mathbf{A}/\mathbf{T}$  ( $\mathbf{A}$  - número de restrições ou erros encontrados e  $\mathbf{T}$  - tempo de uso da unidade de software enquanto executa e compartilha recurso).

- Facilidade de migração: mede a facilidade para o usuário migrar para a unidade de software em questão, definindo-se por  $FMe = A/B$  (**A** - número de funcionalidades da unidade de software que produzem resultados satisfatórios em relação ao produto antigo e **B** - número de funcionalidades testadas).

Para verificar se os resultados encontrados pelas métricas são ou não satisfatórios, Filho (2005) apresenta uma escala para a comparação entre os valores advindos das métricas e os intervalos ditos insatisfatórios e satisfatórios, formulados através de análises empíricas, devendo ser ajustados quando submetidos a novos processos de avaliação de portabilidade. As Tabelas 5 e 6 exibem as escalas de valores, respectivamente, para as métricas internas e para as métricas externas.

**Tabela 5 – Escala para métricas internas**

<b>Métrica</b>	<b>Valores insatisfatórios</b>	<b>Valores satisfatórios</b>
Adaptabilidade ao ambiente de hardware ( <b>AH</b> )	Entre 0 e 0,7	Acima de 0,7
Adaptabilidade ao ambiente de software ( <b>AS</b> )	Entre 0 e 0,7	Acima de 0,7
Grau de portabilidade ( <b>GP</b> )	Entre 0 e 0,6	Acima de 0,6
Esforço para instalar ( <b>EI</b> )	Entre 0 e 0,4	Acima de 0,4
Flexibilidade de instalação ( <b>FI</b> )	Acima de 0,4 ou abaixo de 0,2	Entre 0,2 e 0,4
Disponibilidade de coexistência ( <b>DC</b> )	Entre 0 e 0,7	Acima de 0,7
Consistência de funcionalidades ( <b>DF</b> )	Entre 0 e 0,9	Acima de 0,9

Fonte: FILHO, M. J. A. G. (2005)

**Tabela 6 – Escala para métricas externas**

<b>Métrica</b>	<b>Valores insatisfatórios</b>	<b>Valores satisfatórios</b>
Adaptabilidade ao ambiente de hardware ( <b>AHe</b> )	Entre 0 e 0,8	Acima de 0,8
Adaptabilidade ao ambiente de software ( <b>ASe</b> )	Entre 0 e 0,8	Acima de 0,8
Flexibilidade de instalação ( <b>FIe</b> )	Acima de 0,4	Entre 0 e 0,4
Coexistência apresentada ( <b>CAe</b> )	Acima de 0,3	Entre 0 e 0,3
Facilidade de migração ( <b>FMe</b> )	Entre 0 e 0,9	Acima de 0,9

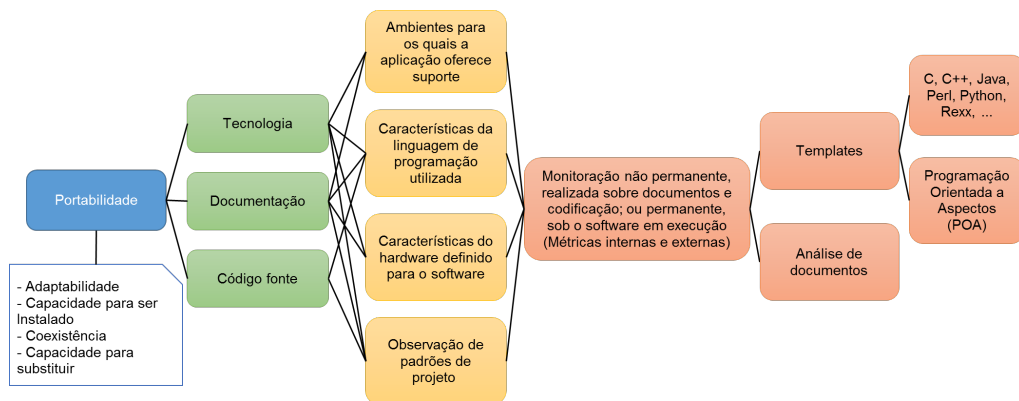
Fonte: FILHO, M. J. A. G. (2005)

Em suma, a avaliação do atributo portabilidade será possível por meio da verificação da documentação, código fonte e da tecnologia utilizada, através do uso das métricas anteriormente destacadas ou de outros meios que objetivam conferir os níveis de adaptabilidade, capacidade para ser instalado, coexistência e capacidade para substituir. Serão

observadas medidas como referentes aos ambientes para os quais a aplicação oferece suporte, as características da linguagem de programação utilizada e do hardware definido para o funcionamento do produto de software, além da análise da arquitetura e dos padrões adotados no projeto.

Salienta-se que a monitoração/avaliação poderá ser não permanente, realizada sobre documentos e codificação, ou permanente, sob o software em execução. A idealização de templates, implementados nas linguagens de programação convencionais ou através da programação orientada a aspectos, é adequada para a avaliação da portabilidade, bem como a utilização de softwares/plugins. Infere-se ainda que o uso das métricas deverá ser feita por desenvolvedores e profissionais aptos para a aplicação e análise dos resultados encontrados. A Figura 19 sintetiza a avaliação do atributo portabilidade discorrido nesta seção.

**Figura 19 – Árvore de Decisão - Monitoração do Atributo Portabilidade**



Fonte: Elaborada pelo autor



## 5 DESCRIÇÃO DA FERRAMENTA PARA AFERIÇÃO DE ATRIBUTOS DE QUALIDADE

Este capítulo tem o objetivo de apresentar uma visão geral acerca da solução para a aferição de atributos de qualidade, salientando as principais características da aplicação e a descrição do seu processo de implementação.

### 5.1 Visão Geral do Processo

Neste trabalho é destacada a construção de uma ferramenta que visa auxiliar arquitetos e desenvolvedores de software na integração entre interesses transversais (atributos de qualidade ou requisitos não-funcionais) e não-transversais (funcionalidades ou atributos funcionais) de softwares desenvolvidos na linguagem de programação Java, buscando a eliminação de problemas que afetam a legibilidade do código e a eficiência do sistema. Para tanto será disposto um conjunto de atividades que se voltam à identificação do interesse a ser monitorado e do ponto no qual a monitoração deverá ocorrer, apoiando o processo de aferição de atributos de qualidade que deve ser expandido para todo o ciclo de vida do software.

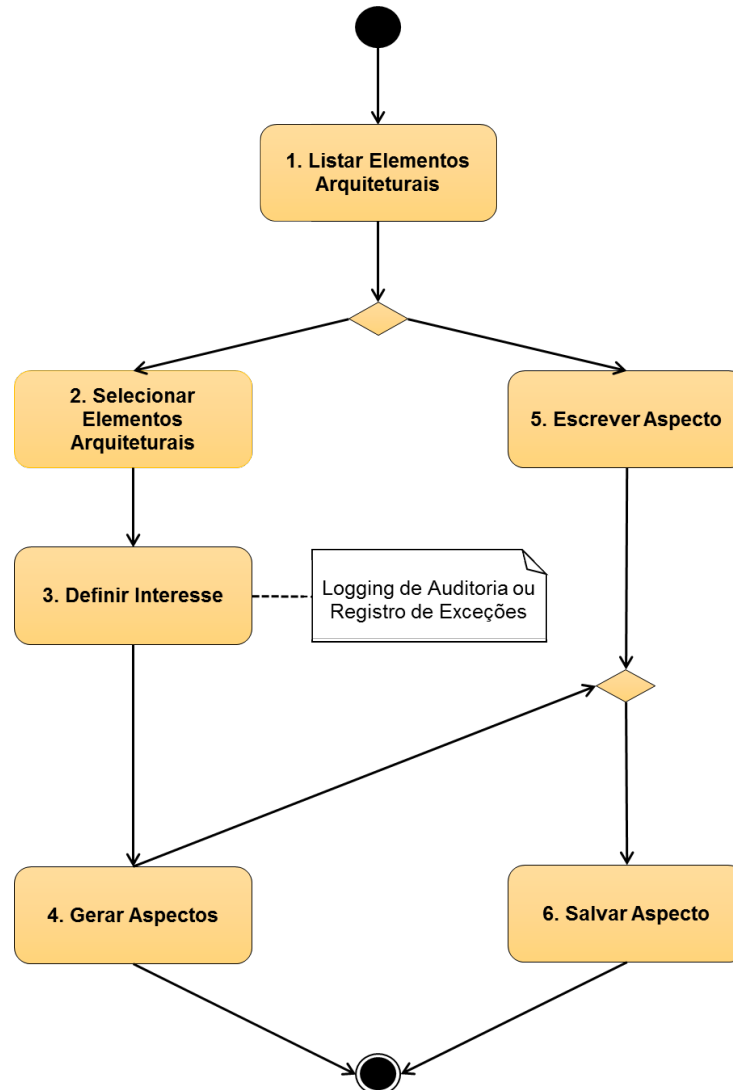
A aplicação aqui apresentada tratará, em um primeiro momento, da monitoração dos atributos confiabilidade e eficiência através da automatização da criação do Registro de Exceções e Logging de Auditoria, amparadas pela utilização conjunta da Programação Orientada a Objetos, base de grande parte dos programas implementados na atualidade, e da Programação Orientada a Aspectos, importante provedora de recursos que lidam com os interesses transversais.

A referida monitoração, proporcionada pela ferramenta implementada, tomará como base os elementos arquiteturais de certo sistema-alvo e realizar-se-á por meio de aspectos-monitores que serão automaticamente gerados, ficando a cargo do usuário apenas a escolha dos elementos a serem monitorados e qual o atributo a ser aferido. Assim, não é necessário que o desenvolvedor ou arquiteto de software possua conhecimentos em POA para configurar aspectos referentes aos dois atributos citados; contudo, caso seja conhecedor, poderá criar seus próprios aspectos de monitoração, através de uma interface específica provida pela ferramenta.

Neste cenário, a aplicação possui seis atividades primordiais: (i) Listar Elementos Arquiteturais; (ii) Selecionar Elementos Arquiteturais; (iii) Definir Interesse; (iv) Gerar Aspectos; (v) Escrever Aspectos; e (vi) Salvar Aspecto. A Figura 20 mostra a visão geral do processo proposto para a ferramenta definida neste trabalho, destacando a disposição das já citadas atividades.

- Listar Elementos Arquiteturais: permite ao usuário da ferramenta listar os elementos arquiteturais de software que compõem seu sistema, como classes, procedimentos

Figura 20 – Visão Geral da Ferramenta Proposta



Fonte: Elaborada pelo autor

e funções, sendo possível observar a disposição de possíveis pontos de junção que ampararão a futura seleção, na etapa seguinte, dos elementos a serem monitorados.

- **Selecionar Elementos Arquiteturais:** consiste na seleção daquele ou daqueles elementos arquiteturais nos quais os aspectos gerados incidirão, isto é, a escolha dos pontos bem definidos do sistema que derivarão os pontos de atuação do aspecto-monitor.
- **Definir Interesse:** possibilita a escolha do interesse a ser monitorado, ou seja, do registro de exceções e ou logging de auditoria que se correlacionam, respectivamente, aos atributos confiabilidade (garantida pela proposição de um mecanismo de segurança, dirigido ao processamento, geração e armazenamento de informações sobre os comportamentos inesperados das funções dos sistemas) e eficiência (observada por meio de especificações referentes ao comportamento de sistemas em relação ao tempo, avaliando os tempos de resposta e/ou de processamento de métodos).

- Gerar Aspectos: destina-se a ação de gerar o aspecto-monitor após a escolha dos pontos de junção e do interesse a ser monitorado. Esta é uma ação interna ao software, resultando na criação de um arquivo com a extensão *.aj*, isto é, um aspecto proveniente da POA.
- Escrever Aspectos: volta-se à possibilidade de escrita de aspectos autorais de monitoração, por parte do desenvolvedor. Através de uma interface específica, aspectos para atributos previamente idealizados poderão ser codificados, necessitando-se de conhecimento prévio em POA.
- Salvar Aspecto: associa-se à atividade anterior, correspondendo à ação de salvar o aspecto que foi implementado na aplicação pelo usuário. Assim como *Gerar Aspectos*, *Salvar Aspecto* também é uma ação interna da ferramenta, resultando em um arquivo de aspecto (*.aj*).

## 5.2 Visão Geral da Implementação

A ferramenta desenvolvida neste trabalho oferece uma forma simplificada de geração de aspectos destinados à monitoração de atributos de qualidade, especificamente a confiabilidade e a eficiência, baseando-se na arquitetura do software e realizando o registro das ações executadas no sistema e também possíveis exceções, sejam elas de negócio ou decorrentes de falhas. Para tanto, o planejamento da aplicação foi estruturado sob três pilares ou módulos: (i) Varredura do código do sistema-alvo; (ii) Aspectos-base; e (iii) Aspectos-monitores.

O primeiro módulo corresponde à implementação da análise do código do sistema-alvo, trazendo a localização do projeto e a disposição de alguns elementos arquiteturais, como as classes que compõem o software e os métodos presentes nessas classes. Este módulo é fundamental à ferramenta, já que os referidos elementos arquiteturais que ele retornará são fundamentais, sobretudo, para a atuação dos aspectos-monitores. Destaca-se que a varredura do código do sistema-alvo foi implementada de forma independente dos demais módulos, tendo em vista que sua efetivação não possui como pré-requisito qualquer elemento desses.

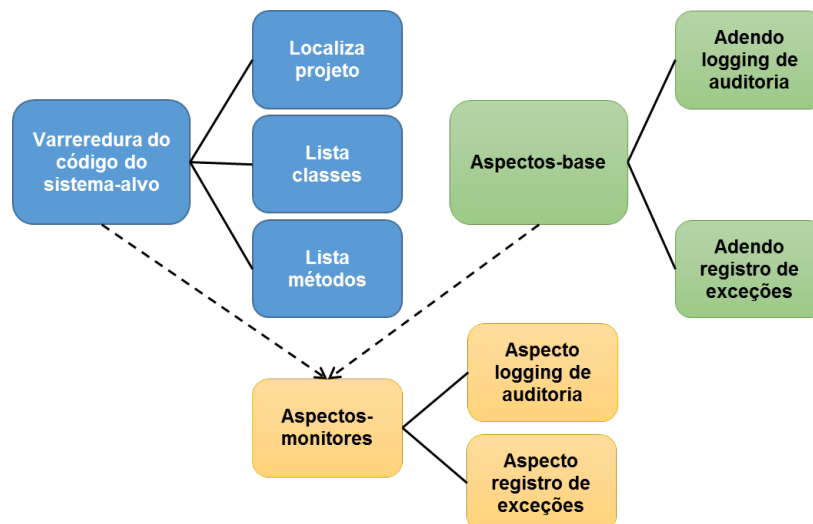
O segundo módulo engloba a implementação dos aspectos-base, que possuem extrema importância para a aplicação desenvolvida. Neste contexto, os aspectos voltados à monitoração são pré-implementados, servindo como base para os aspectos-monitores futuramente gerados. Observada a organização estrutural dos aspectos, introduzida na Subseção 2.3.3, infere-se que o usuário da aplicação aqui discutida será responsável apenas pela definição dos pontos bem definidos do sistema (pontos de atuação ou *pointcuts*) - proporcionada através da execução do módulo anterior - uma vez que os trechos de código responsáveis pelo comportamento dos interesses transversais (adendos ou *advices*) já virão codifica-

dos. A implementação dos aspectos-base também ocorre independentemente dos demais módulos, pois não se observam dependências.

O terceiro módulo, diferentemente dos anteriores, possui como pré-requisito a implementação dos dois módulos já mencionados, pois esses compõem a sua base. De maneira geral, o desenvolvimento deste módulo é destinado a unir os pontos de atuação, que poderão ser definidos através do resultado da implementação do módulo *i*, e os adendos, advindos do resultado da codificação do módulo *ii*. Por meio dessa união, ocorre a criação dos aspectos-monitores registro de exceções e logging de auditoria.

A Figura 21 ilustra a relação existente entre os três módulos definidos para a implementação da ferramenta para aferição de atributos de qualidade. Ressalta-se que a seta tracejada indica a dependência existente entre o módulo Aspectos-monitores e os demais.

**Figura 21 – Relação dos Módulos de Implementação da Ferramenta**



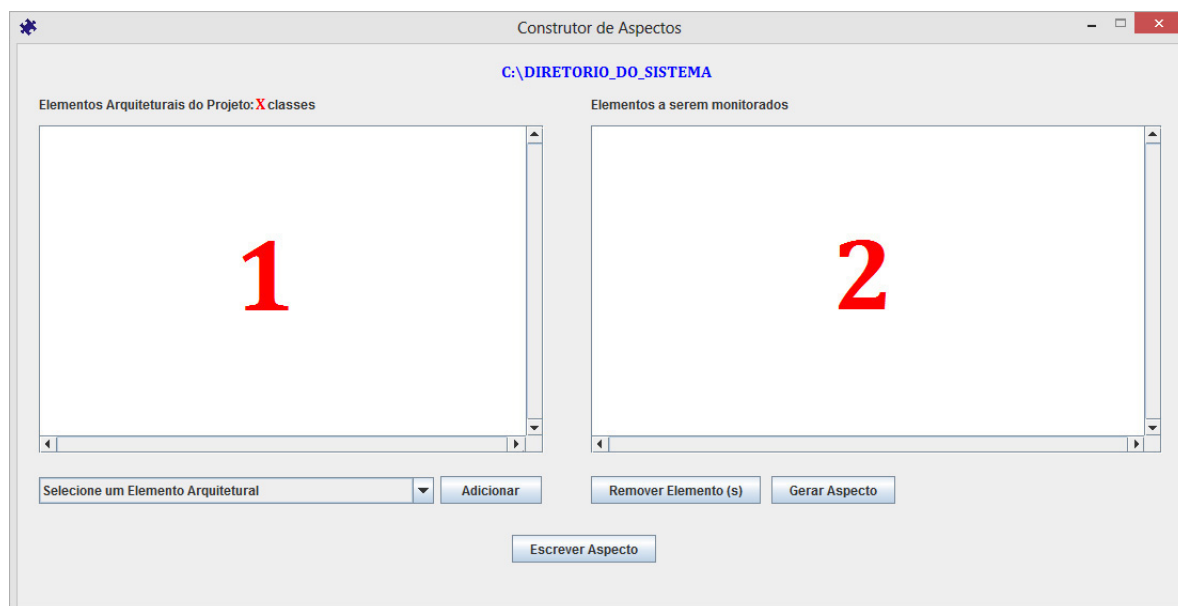
Fonte: Elaborada pelo autor

### 5.3 Principais Características da Ferramenta para Aferição de Atributos de Qualidade

Após o estabelecimento dos módulos referentes à implementação da ferramenta alvo deste trabalho, é importante exibir a interface dessa aplicação responsável por gerar e construir os aspectos-monitores destinados à monitoração da confiabilidade e eficiência, fazendo o uso do registro de exceções e do logging de auditoria. A Figura 22 ilustra a tela inicial da ferramenta supracitada.

No topo da figura apresentada é exibido o endereço do diretório (*C:\DIRETORIO\_DO\_SISTEMA*) no qual a implementação do sistema-alvo da monitoração está codificado. Imediatamente abaixo, no lado esquerdo, pode ser vista a indicação do número de classes que compõem o sistema (*Elementos Arquiteturais do Projeto: X classes*, onde o X representa o número de classes). Na área de texto marcada com o número 1 serão exibidos os elementos arquiteturais (classes e métodos) relativos à aplicação alvo da monitoração,

Figura 22 – Tela Inicial da Ferramenta Proposta



Fonte: Elaborada pelo autor

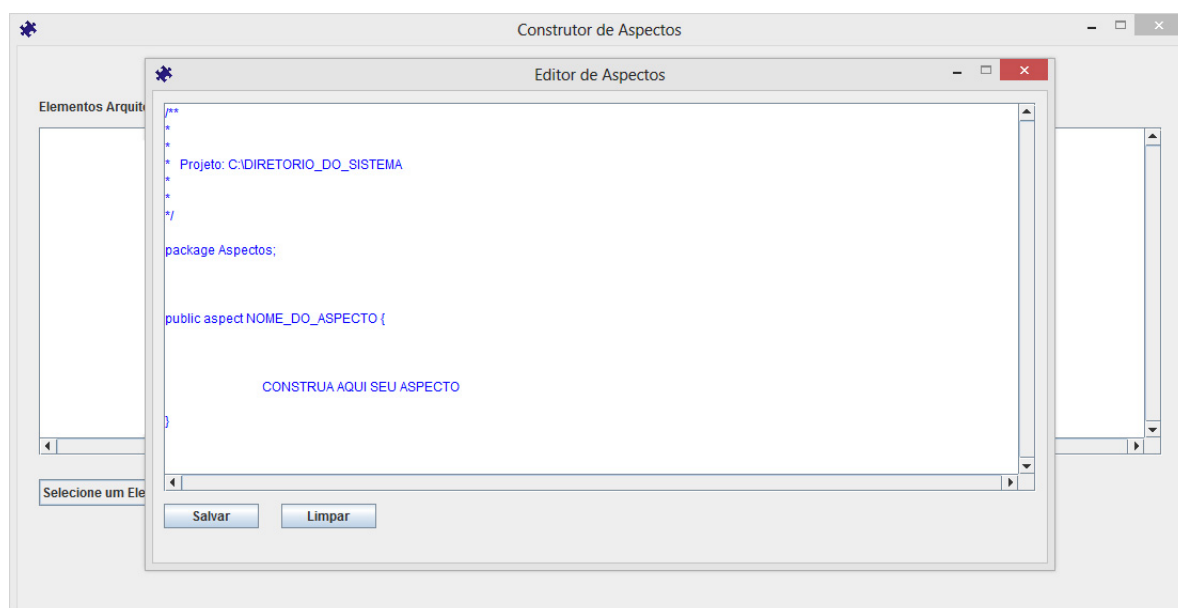
nesse campo será possível vislumbrar quais os pontos de junção que derivarão os pontos de atuação do aspecto-monitor. Logo abaixo da área de texto 1, observa-se um caixa de combinação (do inglês *combo box*), na qual será possível selecionar um ou mais elementos arquiteturais para serem monitorados, adicionando-os (por meio do botão *Adicionar*) a uma lista exibida na área de texto marcada com número 2.

Nota-se também a existência de outros três botões: o botão *Remover Elemento(s)*, responsável por retirar alguns pontos de junção da lista de elementos a serem monitorados; *Gerar Aspecto* que possibilita a escolha de qual aspecto-monitor será gerado, ou seja, o registro de exceções ou o logging de auditoria; e o botão *Escrever Aspecto* que, quando acionado, abrirá uma janela (ver Figura 23) para a edição de aspectos-monitores que está ligada à experiência do desenvolvedor com a programação orientada a aspectos.

#### 5.4 Descrição dos Aspectos-base

Conforme já definido neste trabalho, os dois atributos de qualidade que serão alvo de monitoração voltam-se, em um primeiro momento, à Confiabilidade e Eficiência. Para tanto, será adotada a POA que, associada com a POO, trará formas adequadas de observar os mencionados atributos, evitando problemas como o espalhamento e entrelaçamento de código que trazem diversos pontos negativos ao software. Dessa forma, os interesses transversais que embasarão os dois atributos de qualidade inicialmente monitorados são implementados sob a forma de aspectos, respectivamente, os aspectos Registro de Exceções e Logging de Auditoria, discutidos a seguir.

Figura 23 – Interface para a Criação de Aspectos



Fonte: Elaborada pelo autor

#### 5.4.1 Aspectos-base Registro de Exceções

O interesse Registro de Exceções está inteiramente ligado à confiabilidade, ao ponto que proporciona um mecanismo de segurança que analisa em tempo real a execução dos métodos de um certo sistema-alvo. Esse interesse está voltado ao processamento, geração e armazenamento de informações sobre as funções do software, mais especificamente acerca dos possíveis comportamentos excepcionais decorrentes da utilização do mesmo. Percebe-se que a linguagem Java possui técnicas que permitem que as exceções oriundas da execução de um sistema possam ser tratadas de forma adequada, sem que o sistema termine bruscamente seu processo (FILIPPETTO; CALLEGARI, 2006).

Contudo, além deste tratamento, já amparado pela referida linguagem de programação, é necessário que todas as exceções sejam agrupadas e armazenadas, tendo em vista uma posterior avaliação e detecção de erros que serão imprescindíveis para otimizar o sistema que os gerou.

Neste cenário, o objetivo do Aspecto-base Registro de Exceções corresponde à análise do processamento das funcionalidades, retendo e armazenando todas as exceções (de negócio ou decorrentes de falhas) ocorridas durante a execução do sistema. Logo, neste momento, o foco não está voltado ao provimento de técnicas que visam ao tratamento dos erros, mas sim ao registro de todos os comportamentos excepcionais em um banco de dados, evitando que dados referentes às exceções percam-se durante a utilização do software.

A Figura 24 exhibe o código do aspecto-base aqui discutido, desenvolvido a partir das definições da linguagem de programação orientada a aspectos *AspectJ*, voltado à

interceptação e guarda das exceções decorrentes da utilização de determinada aplicação.

**Figura 24 – Aspecto-base Registro de Exceções**

```

1 public abstract aspect AspectExceptionLogging {
2   ...
3   abstract pointcut customException();
4   abstract pointcut exceptionHandlerPointcut(Exception exception);
5
6   before (Exception e): exceptionHandlerPointcut(e) {
7     log(dateFormat.format(new Date()) + "Erro detectado:" +
8       e.toString());
9     log("ARQUIVO:" + thisJoinPoint.getSourceLocation().getFileName
10      ()
11      + "LOCAL:" + thisJoinPoint.getSourceLocation().
12      getWithinType()
13      + "LINHA:" + thisJoinPoint.getSourceLocation().getLine()
14      );
15     e.printStackTrace();
16   }
17
18   after() throwing (Throwable ex) : customException() {
19     try {
20       log(dateFormat.format(new Date())+"Erro detectado:"+ex.toString
21       ());
22       log("ARQUIVO:" + thisJoinPoint.getSourceLocation().getFileName()
23         + "LOCAL:" + thisJoinPoint.getSourceLocation().getWithinType
24         ()
25         + "LINHA:" + thisJoinPoint.getSourceLocation().getLine());
26     } catch (Exception e) {
27       e.printStackTrace();
28     }
29   }
30   ...
31 }

```

Fonte: Elaborada pelo autor

O código apresentado mostra, nas linhas 3 e 4, dois pontos de atuação abstratos (*customException()* e *exceptionHandlerPointcut(Exception exception)*), cujos pontos de junção serão posteriormente definidos pelo usuário. Há também a disposição, a partir das linhas 6 e 15, de dois adendos (iniciados por *before (Exception e)* e *after() throwing (Throwable ex)*), o primeiro é responsável pela captura e armazenamento das exceções oriundas das falhas do sistema, enquanto o outro se volta à identificação e retenção das exceções de negócio.

Logo, utilizando o Aspecto-base Registro de Exceções, será possível monitorar um determinado sistema-alvo, aferindo a confiabilidade. Nota-se que o aspecto apresentado nesta subseção corresponde à base do Aspecto-monitor Registro de Exceções que realizará, efetivamente, a monitoração do sistema ao inserir os pontos de atuação definidos pelo desenvolvedor neste aspecto-base. Destaca-se que o banco de dados com os registros das

exceções serão compostos de arquivos de texto, tendo em vista a simplicidade e facilidade de utilização.

#### 5.4.2 Aspectos-base Logging de Auditoria

Logging de Auditoria corresponde a um requisito transversal que, de forma análoga ao Registro de Exceções, possui implementação, geralmente, espalhada por diversos módulos do sistema. Este interesse está relacionado ao atributo de qualidade eficiência, ao oferecer uma forma de observar especificações referentes ao comportamento do software em relação ao tempo, avaliando-se os tempos de resposta e/ou de processamento de métodos. Entretanto, mesmo sendo importante para aferir eficiência, o logging de auditoria é constantemente desvalorizado durante o projeto e desenvolvimento de aplicações (BRAZ, 2003).

De maneira geral, esta preocupação transversal denota a atividade de registrar as operações mais relevantes ocorridas em um software ou em parte dele, a depender dos objetivos pretendidos pelos *stakeholders*. Cada log, ou seja, cada registro de operação é efetivado ao ponto que uma tarefa é realizada, retendo informações, como a data e horário que determinada função foi executada, seu nome e o usuário que a efetuou. Salienta-se que, além desses dados, outros poderão ser armazenadas, ficando a escolha desses a critério do desenvolvedor.

Assim, infere-se que o objetivo principal do Aspectos-base Logging de Auditoria implementado nesta ferramenta é desacoplar o interesse de monitoração da implementação das funcionalidades da aplicação em si, facilitando a ativação e desativação desse, quando desejado.

A Figura 25 apresenta o referido aspecto-base, também desenvolvido através da linguagem *AspectJ*. Ele é responsável por registrar a execução das operações efetivadas, capturando e armazenando a assinatura dos métodos (linhas 6 e 15), o tempo inicial no momento da execução (linha 7), o tempo final (linha 16) e ainda tempo de resposta (linha 17). Ressalta-se que o registro, assim como no aspecto-base da subseção anterior, é realizado em arquivos de texto, devido à facilidade de manuseio, não apresentando restrições quanto ao uso de tecnologias especiais ou conhecimentos específicos.

Em suma, por meio do Aspecto-base Logging de Auditoria, será possível monitorar sistemas e aferir o atributo de qualidade eficiência, à medida que se verificam, dentre outras questões, o tempo de resposta de cada método indicado pelo usuário para a monitoração. Salienta-se que esse aspecto-base, como o próprio nome sugere, estrutura o Aspecto-monitor Logging de Auditoria que une os pontos de atuação e os adendos aqui definidos.



Figura 25 – Aspecto-base Logging de Auditoria

```

1 public abstract aspect AspectLogging {
2
3     abstract pointcut method();
4     ...
5     before() : method() {
6         log("IN ", thisJoinPointStaticPart.getSignature() + " " +
7             dateFormat.format(new Date()));
8         ident++;
9         _StartTime.add(System.nanoTime());
10    }
11
12    after(): method() {
13        ...
14        ident--;
15        assinatura = thisJoinPointStaticPart.getSignature() + " " +
16            dateFormat.format(new Date()) + "Tempo de Resposta: "+
17            methodExecutionTime + " nanossegundos";
18        ...
19        log("OUT ", assinatura);
20    }
21
22    private void log(String prefix, Object message) {
23        ...
24        StringBuilder builder = new StringBuilder();
25        for(int i = 0, spaces = ident * 2; i < spaces; i++){
26            builder.append(" ");
27        }
28        builder.append(prefix + ": " + message);
29        gravarLog(builder.toString());
30        ...
31    }
32    ...
33 }

```

Fonte: Elaborada pelo autor

## 5.5 Descrição dos Aspectos-monitores

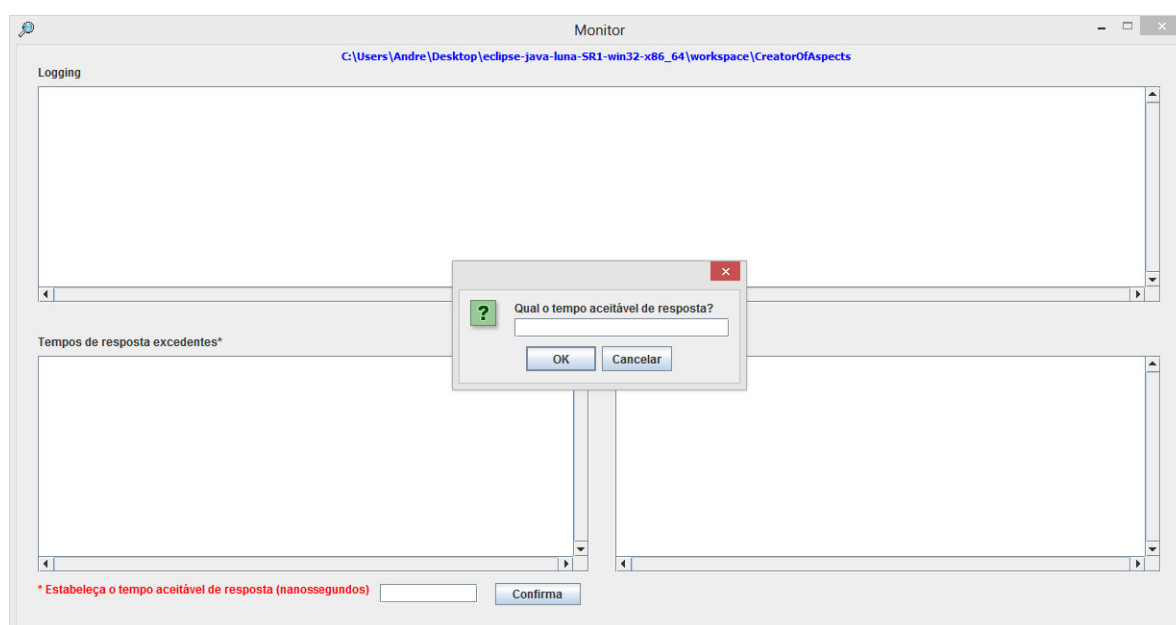
De acordo com o que foi definido na Seção 5.2, os aspectos-monitores intercalam os pontos de atuação e os aspectos-base, que correspondem aos adendos, a fim de garantir a efetividade da monitoração dos atributos Confiabilidade e Eficiência. Dessa forma, a implementação do código responsável por gerar estes monitores só foi possível após o desenvolvimento dos módulos vistos como pré-requisitos.

Em síntese, a ferramenta armazenará a lista de elementos arquiteturais a serem monitorados, transformando-a em pontos de atuação que serão inseridos nos aspectos-monitores. Ressalta-se que os aspectos-base são abstratos, ou seja, correspondem a modelos de aspectos que não podem ser diretamente utilizados, mas sim herdados por aspectos concretos que, neste caso, são os Aspectos-monitores Registro de Exceções e Logging de Auditoria.

Assim, percebe-se que há certa simplicidade na geração desses aspectos, tendo em vista que os adendos já foram implementados por meio dos aspectos-base.

Neste ponto é importante destacar a presença de uma outra interface. Assim que os aspectos-monitores são gerados e esses são submetidos em um sistema real, visando à monitoração, é acionada uma tela denominada de *Monitor*. Ela é responsável por exibir os logs referentes ao registro das exceções, bem como às ações executadas pelo usuário do sistema. A Figura 26 exibe a mencionada interface. Ao centro dessa figura pode ser vista uma caixa de entrada de dados, na qual o usuário da ferramenta deverá definir o tempo de resposta aceitável (em nanossegundos) do processamento dos métodos, tempo que pode ser redefinido na caixa de texto esquerda inferior.

**Figura 26 – Interface Monitor**



Fonte: Elaborada pelo autor

Após a caixa de entrada de dados ser preenchida e confirmada, ocorre a ativação de três áreas de texto: a primeira área (superior) mostra a assinatura, data, hora, minutos, segundos e milésimos no momento do acesso e após finalizar o processamento dos métodos monitorados e ainda os tempos de resposta; a segunda área de texto (inferior esquerda) exibe as mesmas informações destacadas na área anterior, porém para os métodos cujos tempos de resposta ultrapassaram o limiar definido; e a terceira área (inferior direita) exibe as exceções ocorridas, mostrando o erro detectado, o nome do arquivo no qual ele se encontra, a localização deste arquivo e a linha onde o erro foi observado.

Deste modo, a monitoração não será visível somente numa verificação futura dos arquivos de logs gerados. Durante toda a execução do sistema-alvo, o usuário da ferramenta desenvolvida poderá verificar em tempo real o processamento dos métodos marcados para serem monitorados; a ocorrência daqueles que extrapolaram o tempo de resposta conside-

rado aceitável e previamente definido; e ainda as exceções de negócio e aquelas ocasionadas por erros durante a utilização das funcionalidade do sistema.

## 6 METODOLOGIA DE AVALIAÇÃO

Este capítulo apresenta a metodologia utilizada para a avaliação da ferramenta vista neste trabalho, que será feita através da realização de um estudo de caso que visa à aferição dos requisitos de qualidade confiabilidade e eficiência de quatro aplicações desenvolvidas na linguagem de programação Java.

### 6.1 Classificação da Avaliação

A avaliação da solução defendida nesta dissertação classifica-se, quanto aos fins, como sendo qualitativa, descritiva e aplicada, e, quanto aos meios, como um estudo de caso. Neste contexto, objetivando atingir resultados avaliativos satisfatórios, foi utilizado o paradigma *Goal/Question/Metric* (GQM) que suporta a definição *top-down* de um programa de avaliação e a análise e interpretação *bottom-up* dos dados mensurados (WANGENHEIM, 2000).

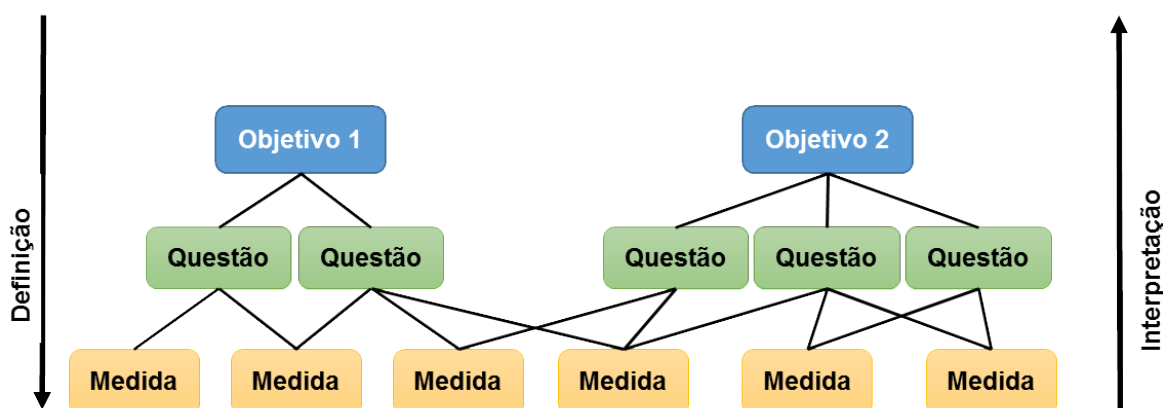
De forma geral, o método GQM tem como principal objetivo apoiar a definição de medidas que estão alinhadas aos objetivos de negócio de determinada organização, tomando como prerrogativa o entendimento de que a realização adequada de medições de produtos e/ou processos é fruto da elaboração prévia das metas de seus respectivos projetos (ROCHA; SOUZA; BARCELLOS, 2012). Assim, infere-se que o resultado da composição do GQM corresponderá a uma estrutura hierárquica formada por três níveis, também definida como *Plano GQM*: (i) nível conceitual (Objetivos ou Metas), (ii) nível operacional (Questões) e (iii) nível quantitativo (Medidas).

A Figura 27 apresenta os três níveis citados, hierarquicamente dispostos. Nota-se que através da abordagem *top-down*, os objetivos ou metas serão criados e refinados sob a forma de questões. Em seguida, medidas são projetadas a fim de que sejam definidos meios para a resolução das questões anteriormente determinadas. Então, com objetivos, questões e medidas delineadas, parte-se para a análise de certo produto, interpretando e verificando de forma *bottom-up* o grau de alcance das metas.

No nível conceitual, os objetivos ou metas estão relacionados à identificação daquilo que se deseja conseguir com a análise do produto ou processo. No nível operacional, as questões auxiliam na compreensão de como as metas serão atingidas, criando um contexto de qualidade a partir da perspectiva dos objetivos. No nível quantitativo, as métricas correlacionam as medidas a serem projetadas às questões previamente definidas, a fim de inferir como essas serão respondidas e alcançar, conseqüentemente, as metas estabelecidas.

Destaca-se que, logo após a composição dos objetivos, questões e medidas, é indispensável a produção do *Plano de Medição e Análise* que deverá abarcar informações, como:

Figura 27 – Representação do Método GQM



Fonte: Adaptado de BASILI, V. R. (1992)

definição das medidas; possíveis valores para as medidas; procedimentos para coleta das medidas, que podem ser manuais ou automatizados; responsabilidade pela coleta das medidas, dispendo sobre os responsáveis que realizarão as medições; escolha do momento em que a coleta deve ser realizada; procedimentos para análise das medidas; forma de apresentação dos dados para os envolvidos; dentre outras (SOLINGEN; BERGHOUT, 1999).

Com relação às vantagens da utilização do paradigma GQM, embora a principal seja o estabelecimento de métricas úteis e relevantes direcionadas à qualidade e à análise e interpretação dos dados coletados, há a percepção de uma série de outros benefícios que estão ligados à compreensão das práticas de software de certa organização, acompanhamento dos processos dos produtos e avaliação de novas tecnologias de desenvolvimento.

Entretanto, para proporcionar essas vantagens, os programas de mensuração baseados em GQM têm de ser planejados e executados mediante a observação de alguns aspectos (WANGENHEIM, 2000):

- A tarefa de análise a ser executada precisa ser especificada precisamente e explicitamente através de uma meta de mensuração, ou seja, um objetivo claramente definido.
- Medidas precisam ser derivadas de uma forma *top-down*, baseando-se nos objetivos e questões desenvolvidos. Assim, uma estrutura de metas e questões não pode ser utilizada como modelo fixo para um conjunto de medidas existentes.
- Cada medida precisa ter fundamentação lógica que a apoie, explicitamente documentada, para justificar a coleta dos dados e para guiar a análise e interpretação.
- Os dados que são coletados a partir das medidas necessitam ser interpretados de

forma *bottom-up* no contexto dos objetivos GQM e das questões, destacando as limitações e suposições do produto em análise por meio da verificação de cada medida.

- Os utilizadores dos resultados do programa de mensuração precisam estar profundamente envolvidos na definição e interpretação desse, já que são os reais peritos com respeito ao objeto e ao enfoque da qualidade investigada no programa e, portanto, proporcionam interpretações válidas no ambiente específico.

Enfim, a abordagem do paradigma destacado auxilia a gerência dos riscos, suportando a adequação das métricas aos objetivos de negócio da organização, possibilitando, dessa forma, melhorias nos processos de software que culminam na mensuração apropriada da qualidade dos produtos. Neste ponto, é importante salientar que o entendimento preciso dos três níveis do modelo GQM é fundamental para o sucesso da verificação do produto em análise; logo, gerentes de projeto, desenvolvedores, clientes e todas as partes interessadas necessitam estar alinhados para a concepção deste modelo.

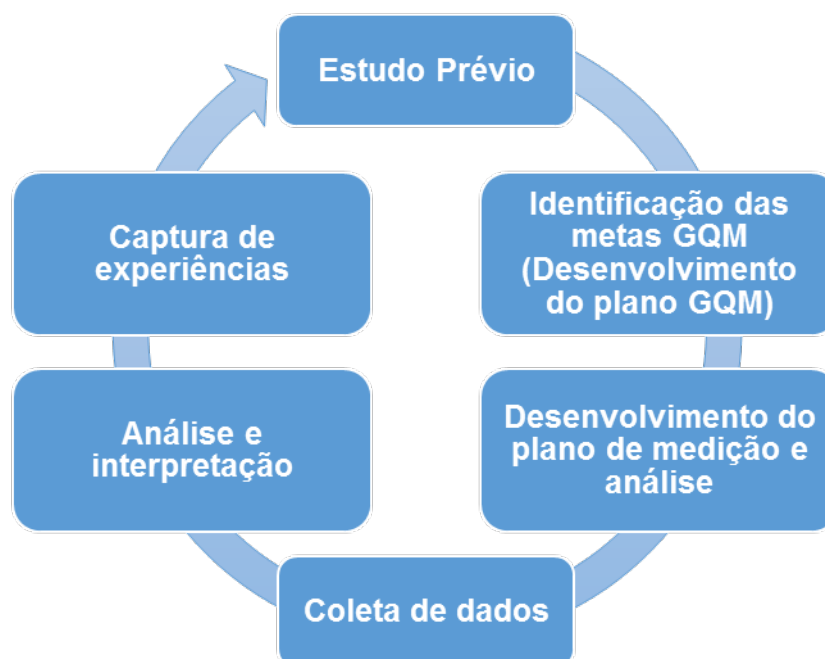
## 6.2 Desenvolvimento do Método GQM

De maneira sintética, a aplicação do paradigma GQM na avaliação da ferramenta aferidora da confiabilidade e eficiência será projetada em duas etapas: a primeira delas é relativa à composição do nível hierárquico, ou seja, do Plano GQM (objetivos, questões e medidas), considerando as características da aplicação desenvolvida; e a segunda refere-se a produção do Plano de Medição e Análise que conterá todas as informações sobre como, em que momento e quem realizará as medições no software.

A partir da definição desses planos, será possível capturar as experiências concebidas durante a utilização do programa de mensuração, culminando em dados fundamentais à verificação da solução proposta neste trabalho. A Figura 28 expõe uma visão geral dos passos do processo do método GQM aqui utilizado. A princípio, é realizado um estudo prévio visando encontrar modelos de experiência com características similares as do produto em análise, resultando na concepção do objeto, objetivo, enfoque de qualidade, ponto de vista e contexto da análise, relativos ao programa de mensuração.

Em seguida, cria-se o plano GQM que será composto pelos objetivos, questões e medidas a serem verificadas. Logo após, parte-se para a concepção do plano de medição e análise, contendo aspectos relacionados à forma como o programa de mensuração será aplicado. Então, é realizada a aplicação do método, procedendo a coleta de dados que serão analisados e interpretados, tomando como base os objetivos destacados no plano GQM. Assim, as experiências serão capturadas e, possivelmente, converter-se-ão em modelos reutilizáveis.

Figura 28 – Visão geral do processo GQM



Fonte: Adaptado de WANGENHEIM, C. G. (2000)

### 6.2.1 Estudo Prévio

Conforme já discutido, o objetivo da ferramenta desenvolvida neste trabalho é aferir qualidade a determinados sistemas, por meio da observação de alguns atributos de qualidade que são especificados na norma ISO/IEC 9126. Neste cenário, a aplicação aqui disposta será avaliada através da sua utilização, em conjunto, com quatro aplicações desenvolvidas na linguagem de programação Java, fazendo-se o uso do ambiente de desenvolvimento integrado - do inglês *Integrated Development Environment* (IDE) - Eclipse.

Desse modo, os dois sistemas reais utilizados como alvo da monitoração advinda do software, cuja implementação é apresentada neste documento, são: DimDimDim (MERCER, 2007), um sistema simples para controle financeiro pessoal, desenvolvido e com código aberto pela empresa UseInet; e Mayam (DCM4CHE, 2014), uma estação de trabalho com código aberto voltada à visualização de imagens DICOM - *Digital Imaging and Communications in Medicine* - e implementada usando o conjunto de *widgets* DCM4CHE Java toolkit.

Além dos softwares DimDimDim e Mayam, outras duas aplicação serão utilizadas nesta avaliação. Trata-se dos softwares do projeto BancoUnifimes (JÚNIOR; AFONSO, 2014) que oferece duas versões para um mesmo sistema bancário hipotético - *BancoOO*, implementado unicamente com o paradigma da orientação a objetos, e *BancoOA*, desenvolvido através da união entre a programação orientada a objetos e a programação orientada a aspectos - que realiza registros de operações importantes, como saques, depósitos, transferência, programa de relacionamento com o cliente e oferta de crédito especial.

Salienta-se que antes da aplicação do método e criação do Plano GQM e do Plano de Medição e Análise, é perceptível a necessidade de um estudo preliminar destinado a concentrar mais conhecimento acerca do paradigma que ampara a definição dos referidos planos; além de pesquisar modelos de experiência já concebidos, detentores de características similares as do software em análise. Neste estudo não foram encontrados modelos ou mesmo aplicações do método GQM semelhantes ao processo de verificação explicitado neste capítulo, isto é, a literatura não apresenta a mensuração de qualidade por meio do paradigma GQM para softwares que lidam com requisitos não-funcionais, como é o caso da solução aqui disposta.

Assim, a execução do estudo prévio contribuiu para a formulação de um entendimento mais sólido a respeito do paradigma *Goal/Question/Metric*, bem como para a compreensão do Plano GQM, do Plano de Medição e Análise e da forma como utilizar esses planos para a aplicação do método e obtenção de resultados desejados. Entende-se que, como a avaliação da solução proposta não está inserida em um ambiente organizacional, não se faz necessária a elaboração de projetos potenciais nem de um projeto piloto, já que não há uma organização a ser caracterizada e, conseqüentemente, não existem metas de melhoria organizacionais a serem identificadas.

### 6.2.2 Definição do Plano GQM

O Plano GQM congrega informações precisas para o planejamento da mensuração do produto, sua análise e interpretação dos dados a serem coletados. O procedimento para a sua composição compreende a concepção das metas que deverão ser alcançadas, desenvolvendo cada uma delas a partir de uma série de questões e métricas. Especificamente, o plano define as **metas**, objetivos determinantes para conferir a qualidade ao software; **questões** ou perguntas, voltadas a identificar as informações necessárias para se chegar às metas propostas; e as **métricas**, destinadas a determinar quais e como os dados serão coletados para que as questões sejam respondidas.

Cinco quesitos relativos ao produto que será analisado e ao contexto no qual ele está inserido são determinantes para a definição das metas, eles são: objeto, que se refere ao alvo do processo ou do estudo, ou seja, ao produto que será verificado; finalidade, que exhibe a motivação por trás de cada meta, o propósito da análise do objeto; foco, que se volta à observação das propriedades do objeto que serão analisadas; ponto de vista, que determina a ótica de quem utilizará os dados coletados; e contexto, que envolve o ambiente no qual a mensuração de qualidade será realizada.

Determina-se, portanto, que o **Objeto** corresponde à ferramenta aferidora de confiabilidade e eficiência, a **Finalidade** é a avaliação da sua execução e funcionamento dos aspectos-monitores, o **Foco** é a exatidão relativa à geração de aspectos e a efetivação das funcionalidades dos aspectos gerados, o **Ponto de Vista** será do desenvolvedor ou do usuário e o **Contexto** é Instituto de Computação (IC - UFAL). Neste sentido, as



metas (*Goal*) são resumidas em: (i) avaliar a execução do software quanto à geração dos aspectos-monitores; (ii) avaliar o software quanto à observação do funcionamento do aspecto-monitor da confiabilidade; e (iii) avaliar o software quanto à observação do funcionamento do aspecto-monitor da eficiência.

O próximo passo consiste em elaborar as questões (*Question*) que auxiliarão para o atendimento dos objetivos. Para tanto, cada uma das metas é analisada, idealizando-se perguntas que deverão ser adequadas para que mensuração seja efetivada e os objetivos alcançados. Dessa forma, a referida análise das metas resultou nas questões encontradas nas Tabelas 7, 8 e 9 que são relativas, respectivamente, às metas i, ii e iii.

**Tabela 7 – Plano GQM - Meta i - Questões**

---

**Meta i**

Avaliar a execução do software quanto à geração dos aspectos-monitores

---

**Questões**

**Q1.** A solução pode ser executada em aplicações com diferentes estilos arquiteturais?

**Q2.** Os elementos arquiteturais exibidos pelo software correspondem aos elementos da aplicação sob a qual ele está sendo executado?

**Q3.** A seleção de um elemento faz com que esse seja inserido na lista de elementos a serem monitorados?

**Q4.** A ação Remover Elemento culmina na exclusão de determinado elemento selecionado da lista de elementos a serem monitorados?

**Q5.** A ação Gerar Aspecto, executada quando a lista de elementos a serem monitorados for vazia, resulta em um alerta de impossibilidade de geração de aspectos-monitores?

**Q6.** A ação Gerar Aspecto, executada quando a lista de elementos a serem monitorados não for vazia, resulta na oferta das opções de interesses para monitoração?

**Q7.** A seleção de um dos interesses a serem monitorados resulta na geração de um arquivo com extensão *.aj* presente no diretório \Aspectos do projeto da aplicação?

**Q8.** A ação Escrever Aspecto culmina na abertura da interface Editor de Aspectos voltada para a implementação de aspectos-monitores?

**Q9.** A ação Salvar, realizada na interface Editor de Aspectos, resulta na geração de um arquivo extensão *.aj* com nome e local de armazenamento a serem definidos pelo usuário?

---

Fonte: Elaborada pelo autor

Realizada a elaboração das questões relativas às metas propostas, resta definir as medidas ou métricas (*Metric*) que serão utilizadas para a resolução das referidas questões. Assim, tomando como base a observação das metas e questões apresentadas nas tabelas anteriores, as seguintes métricas ( $M_i$ ) são determinadas:

- **META 1: Avaliar a execução do software quanto à geração dos aspectos-monitores**

**Q1.** A solução pode ser executada em aplicações com diferentes estilos arquiteturais?

Tabela 8 – Plano GQM - Meta ii - Questões

**Meta ii**

Avaliar o software quanto à observação do funcionamento do aspecto-monitor da confiabilidade

**Questões**

- Q1.** O aspecto-monitor da confiabilidade permite avaliar aplicações com diferentes estilos arquiteturais?
- Q2.** É possível selecionar as funcionalidades críticas, em termos de confiabilidade?
- Q3.** Como avaliar a confiabilidade de cada funcionalidade?
- Q4.** A aferição da confiabilidade pode ser influenciada por requisitos ortogonais à funcionalidade?
- Q5.** Como o monitor de confiabilidade se comporta quando a aplicação monitorada usa POA?
- Q6.** Há como distinguir se as exceções são consideradas corretas/desejáveis ou geradas por erros?
- Q7.** Qual a proporção de exceções de negócio (esperadas) e exceções decorrentes de bug (inesperadas)?
- Q8.** Há a geração de um arquivo cujo conteúdo seja as exceções advindas da aplicação?
- Q9.** A solução possibilita a observação das exceções em uma interface específica?

Fonte: Elaborada pelo autor

Tabela 9 – Plano GQM - Meta iii - Questões

**Meta iii**

Avaliar o software quanto à observação do funcionamento do aspecto-monitor da eficiência

**Questões**

- Q1.** O aspecto-monitor da eficiência permite avaliar aplicações com diferentes estilos arquiteturais?
- Q2.** É possível selecionar as funcionalidades críticas, em termos de eficiência?
- Q3.** Como avaliar a eficiência de cada funcionalidade?
- Q4.** A solução retém os tempos de resposta quando as funcionalidades monitoradas são executadas?
- Q5.** A aferição da eficiência pode ser influenciada por requisitos ortogonais à funcionalidade?
- Q6.** Como o monitor de eficiência se comporta quando a aplicação monitorada usa POA?
- Q7.** Qual o critério para classificar uma funcionalidade, a partir do seu tempo de resposta?
- Q8.** Há a geração de um arquivo cujo conteúdo seja as informações advindas das funcionalidades da aplicação?
- Q9.** A solução possibilita a observação do logging das funcionalidades em uma interface específica?

Fonte: Elaborada pelo autor

- M1.** *Executar em aplicações com estilos arquiteturais distintos.*
- Q2.** Os elementos arquiteturais exibidos pelo software correspondem aos elementos da aplicação sob a qual ele está sendo executado?
- M2.** *Verificar a lista de elementos mostrados pela solução e compará-los aos elementos arquiteturais dispostos com o código e/ou documentação da aplicação em uso.*
- Q3.** A seleção de um elemento faz com que esse seja inserido na lista de elementos a serem monitorados?
- M3.** *Observar se o elemento inserido é adicionado à lista de elementos a serem monitorados.*
- Q4.** A ação Remover Elemento culmina na exclusão de determinado elemento selecionado da lista de elementos a serem monitorados?
- M4.** *Observar se a ação de remoção de um elemento da lista de elementos a serem monitorados resultou na sua exclusão da referida lista.*
- Q5.** A ação Gerar Aspecto, executada quando a lista de elementos a serem monitorados for vazia, resulta em um alerta de impossibilidade de geração de aspectos-monitores?
- M5.** *Executar a ação Gerar Aspecto, com a lista de elementos a serem monitorados vazia, notando a ocorrência de um aviso/alerta.*
- Q6.** A ação Gerar Aspecto, executada quando a lista de elementos a serem monitorados não for vazia, resulta na oferta das opções de interesses para monitoração?
- M6.** *Executar a ação Gerar Aspecto, contendo ao menos um elementos na lista de elementos a serem monitorados, observando a disposição de opções de interesses para monitoração.*
- Q7.** A seleção de um dos interesses a serem monitorados resulta na geração de um arquivo com extensão .aj presente no diretório \Aspectos do projeto da aplicação?
- M7.** *Escolher um dos interesses de monitoração e conferir a geração do respectivo aspecto-monitor com extensão .aj (presente no diretório \Aspectos do projeto da aplicação).*
- Q8.** A ação Escrever Aspecto culmina na abertura da interface Editor de Aspectos voltada para a implementação de aspectos-monitores?
- M8.** *Executar a ação Escrever Aspecto e observar a abertura da interface Editor de Aspectos.*

**Q9.** A ação Salvar, realizada na interface Editor de Aspectos, resulta na geração de um arquivo extensão *.aj* com nome e local de armazenamento a serem definidos pelo usuário?

**M9.** *Executar a ação Salvar, nomear o arquivo a ser salvo e escolher o diretório no qual esse será armazenado, observando posteriormente se o armazenamento foi realizado.*

• **META 2: Avaliar o software quanto à observação do funcionamento do aspecto-monitor da confiabilidade**

**Q1.** O aspecto-monitor da confiabilidade permite avaliar aplicações com diferentes estilos arquiteturais?

**M10.** *Executar em aplicações com estilos arquiteturais distintos.*

**Q2.** É possível selecionar as funcionalidades críticas, em termos de confiabilidade?

**M11.** *Executar a aplicação em funcionalidades específicas, selecionadas pelo usuário.*

**Q3.** Como avaliar a confiabilidade de cada funcionalidade?

**M12.** *Observar a ocorrência de exceções de negócio e de exceções decorrentes de erros.*

**Q4.** A aferição da confiabilidade pode ser influenciada por requisitos ortogonais à funcionalidade?

**M13.** *Observar se o aspecto-monitor da confiabilidade deve ser executado de forma exclusiva ou combinada, notando o possível impacto às funcionalidades da aplicação.*

**Q5.** Como o monitor de confiabilidade se comporta quando a aplicação monitorada usa POA?

**M14.** *Executar o aspecto-monitor da confiabilidade em dois cenários: duas versões de um mesmo sistema, uma OO e outra POA. Notar se apresentam o mesmo resultado.*

**Q6.** Há como distinguir se as exceções são consideradas corretas/desejáveis ou geradas por erros?

**M15.** *Executar a solução, distinguindo as exceções advindas do negócio daquelas geradas por bugs.*

**Q7.** Qual a proporção de exceções de negócio (esperadas) e exceções decorrentes de bug (inesperadas)?

**M16.** *Determinar a proporção de exceções esperadas e inesperadas, através da distinção das exceções advindas do negócio das geradas por bugs.*

**Q8.** Há a geração de um arquivo cujo conteúdo seja as exceções advindas da aplicação?

**M17.** *Após a execução da solução, verificar a existência de um arquivo contendo as exceções decorrentes da utilização da aplicação.*

**Q9.** A solução possibilita a observação das exceções em uma interface específica?

**M18.** *Durante a execução da solução, junto à aplicação alvo da monitoração, observar a disposição de uma interface específica que mostre as exceções em tempo real de utilização da referida aplicação.*

• **META 3: Avaliar o software quanto à observação do funcionamento do aspecto-monitor da eficiência**

**Q1.** O aspecto-monitor da eficiência permite avaliar aplicações com diferentes estilos arquiteturais?

**M19.** *Executar em aplicações com estilos arquiteturais distintos.*

**Q2.** É possível selecionar as funcionalidades críticas, em termos de eficiência?

**M20.** *Executar a aplicação em funcionalidades específicas, selecionadas pelo usuário.*

**Q3.** Como avaliar a eficiência de cada funcionalidade?

**M21.** *Observar os tempos de resposta em nanossegundos.*

**Q4.** A solução retém os tempos de resposta quando as funcionalidades monitoradas são executadas?

**M22.** *Executar as funcionalidades e observar a retenção do tempo de acesso, do tempo de conclusão e do tempo de resposta.*

**Q5.** A aferição da eficiência pode ser influenciada por requisitos ortogonais à funcionalidade?

**M23.** *Observar se o aspecto-monitor da eficiência deve ser executado de forma exclusiva ou combinada, notando o possível impacto às funcionalidades da aplicação*

**Q6.** Como o monitor de eficiência se comporta quando a aplicação monitorada usa POA?

**M24.** *Executar o aspecto-monitor da eficiência em dois cenários: duas versões de um mesmo sistema, uma OO e outra POA. Notar se apresentam o mesmo resultado.*

**Q7.** Qual o critério para classificar uma funcionalidade, a partir do seu tempo de resposta?

**M25.** *Estabelecer um critério - e.g. permitir ao usuário definir um limiar quantitativo - voltado à mensuração do tempo de resposta das funcionalidades.*

**Q8.** Há a geração de um arquivo cujo conteúdo seja as informações advindas das funcionalidades da aplicação?

**M26.** *Após a execução da solução, verificar a existência de um arquivo contendo o logging das funcionalidades monitoradas.*

**Q9.** A solução possibilita a observação do logging das funcionalidades em uma interface específica?

**M27.** *Durante a execução da solução, junto à aplicação alvo da monitoração, observar a disposição de uma interface específica que mostre, em tempo real, o logging das funcionalidades monitoradas.*

Enfim, diante das metas, questões e métricas apresentadas, o Plano GQM para a avaliação da solução proposta apresenta-se formulado. Antes de ser utilizado, contudo, é de suma importância a elaboração do Plano de Medição e Análise que guiará a aplicação método GQM no estudo de caso a ser realizado.

### 6.2.3 Definição do Plano de Medição e Análise

Uma vez que as métricas foram identificadas, é necessário determinar como o Plano GQM será aplicado, levando em consideração, por exemplo, alguns critérios relacionados às medidas, como: definição; possíveis valores, os quais poderão ser encontrados durante a análise; procedimentos para coleta, os instrumentos de mensuração utilizados para o recolhimento das medidas; decisão dos responsáveis pela coleta; estabelecimento do tempo, momento ou frequência na qual a coleta deve ser realizada; procedimentos para análise, forma como os dados coletados serão analisados; e a forma de apresentação dos dados.

Neste contexto, antes de explicitar no Plano de Medição e Análise cada medida e seus respectivos aspectos de análise, deve-se recordar qual o perfil da solução avaliada e ainda quais serão as aplicações que se fazem necessárias para que essa avaliação ocorra. Dessa maneira, ressalta-se, que o software verificado trata-se de ferramenta aferidora dos atributos de qualidade confiabilidade e eficiência para sistemas desenvolvidos na linguagem de programação Java. Este software foi implementado através da Programação Orientada a Aspectos, usando conceitos dessa que foram associados aos da Programação Orientada a Objetos.

Por ser uma ferramenta aferidora da qualidade de outras aplicações, é indispensável que a avaliação ocorra mediante à utilização conjunta do software desenvolvido com outros sistemas. Assim, quatro aplicações auxiliares foram escolhidas para compor a fase de verificação, essas, conforme já referenciadas, são DimDimDim, Mayam e aquelas advindas

do projeto BancoUnifimes (BancoOO e BancoOA), todas com código aberto, codificados em Java e com projeto carregado no Eclipse IDE.

A escolha das mencionadas aplicações ocorreu por três razões principais: o interesse em observar a mensuração de qualidade em sistemas de diferentes portes, um mais simples e outro maior, como é o caso, respectivamente, do DimDimDim e do Mayam; devido ao critério de facilidade de utilização, tendo em vista que este fator facilitaria a realização do estudo de caso, que tanto poderia ser feito pelo próprio desenvolvedor do software em processo de avaliação quanto por um usuário minimamente treinado; e ainda pelo interesse em testar a solução em aplicações OO e aplicações OA, conforme oferta-se no projeto BancoUnifimes.

Uma das formas mais usuais para que um Plano de Medição e Análise, utilizado para complementar e amparar o processo de avaliação traçado no Plano GQM, seja representado é no formato de tabelas que abordam a definição das medidas, os valores que podem ser encontrados, responsáveis pela coleta (desenvolvedor ou usuário), momento da medição (durante a execução das aplicações), forma de coleta das medidas (questionário) e forma de apresentação dos dados (relatório). Percebe-se, na avaliação discutida neste trabalho, que alguns aspectos de análise serão constantes, é o caso dos dados apresentados anteriormente entre parênteses.

Logo, a Tabela 10 referencia as definições das métricas vistas na Subseção 6.2.2 e estabelece os valores ou critérios que serão observados. Tais valores, após a execução da solução junto às aplicações, deverão ser explicitados - através de um questionário - pelo responsável por coletar os dados, ou seja, aquele que aplicou o método GQM.

**Tabela 10 – Plano de Medição e Análise**

<b>Métricas</b>	<b>Valores</b>
<b>M1.</b>	<i>Sucesso</i> ou <i>Falha</i> na execução da aplicação
<b>M2.</b>	<i>Igualdade</i> ou <i>Diferença</i> ao comparar elementos exibidos com a documentação da aplicação
<b>M3.</b>	<i>Sucesso</i> ou <i>Falha</i> na adição de elementos à lista de elementos a serem monitorados
<b>M4.</b>	<i>Sucesso</i> ou <i>Falha</i> na remoção de um elemento da lista de elementos a serem monitorados
<b>M5.</b>	<i>Ocorrência</i> ou <i>Não Ocorrência</i> de aviso/alerta na ação Gerar Aspecto (observar condição)
<b>M6.</b>	<i>Ocorrência</i> ou <i>Não Ocorrência</i> de disposição de interesses de monitoração na ação Gerar Aspecto (observar condição)
<b>M7.</b>	<i>Sucesso</i> ou <i>Falha</i> na geração de aspectos-monitores

continua na próxima página

Tabela 10 – Plano de Medição e Análise (continuação)

Métricas	Valores
M8.	<i>Exibição</i> ou <i>Não Exibição</i> da interface Editor de Aspectos
M9.	<i>Sucesso</i> ou <i>Falha</i> no armazenamento de aspectos implementados
M10.	<i>Sucesso</i> ou <i>Falha</i> na execução do aspecto-monitor da confiabilidade
M11.	<i>Sucesso</i> ou <i>Falha</i> na monitoração de funcionalidades especificadas
M12.	<i>Ocorrência</i> ou <i>Não Ocorrência</i> de exceções de negócio e de exceções decorrentes de erros
M13.	<i>Lentidão</i> ou <i>Funcionamento Padrão</i> quanto à aferição da confiabilidade
M14.	<i>Comportamento da Monitoração</i> em sistemas OO e OA
M15.	<i>Distinção</i> ou <i>Não Distinção</i> entre exceções de negócio e de exceções decorrentes de erros
M16.	<i>Determinação</i> ou <i>Não Determinação</i> da proporção dos dois tipos de exceções
M17.	<i>Sucesso</i> ou <i>Falha</i> na criação de um arquivo com o registro das exceções
M18.	<i>Ocorrência</i> ou <i>Não Ocorrência</i> de uma interface que mostra as exceções
M19.	<i>Sucesso</i> ou <i>Falha</i> na execução do aspecto-monitor da eficiência
M20.	<i>Sucesso</i> ou <i>Falha</i> na monitoração de funcionalidades especificadas
M21.	<i>Igualdade</i> ou <i>Diferença</i> ao comparar o limiar de resposta estabelecido com os métodos que ultrapassaram este limite de tempo
M22.	<i>Sucesso</i> ou <i>Falha</i> na retenção do tempo de acesso, do tempo de conclusão e do tempo de resposta das funcionalidades
M23.	<i>Lentidão</i> ou <i>Funcionamento Padrão</i> quanto à aferição da eficiência
M24.	<i>Comportamento da Monitoração</i> em sistemas OO e OA
M25.	<i>Ocorrência</i> ou <i>Não Ocorrência</i> do estabelecimento do tempo aceitável de resposta
M26.	<i>Sucesso</i> ou <i>Falha</i> na criação de um arquivo com o logging da aplicação
M27.	<i>Ocorrência</i> ou <i>Não Ocorrência</i> de uma interface que mostra o logging

Fonte: Elaborada pelo autor

Finalizando a discussão acerca do Plano de Medição e Análise, salienta-se que a verificação da solução proposta ocorrerá no Eclipse IDE, através da inserção dessa no projeto de cada uma das aplicações auxiliares. Dessa forma, ocorre uma múltipla execução, isto é, o carregamento do software desenvolvido e do sistema que será avaliado por ele.

Com a finalidade de prover uma forma adequada para a coleta das medidas advindas da execução do método GQM, ficou estabelecido que o meio utilizado para tanto corresponde aos questionários que são projetados para suportar a integração da coleta dos dados no processo definido nos planos GQM e de Medição e Análise.



Portanto, para cada meta, foi desenvolvido um questionário que será respondido pelo responsável pela verificação da solução apresentada, recomendando-se que esta resolução ocorra o mais cedo possível para garantir maior precisão dos dados coletados. Para tanto, algumas diretrizes foram seguidas: clareza em relação às medidas que deverão ser observadas; linguagem minimamente técnica; organização estruturada das perguntas em torno de cada meta alvo da análise; e preferência na utilização de questões objetivas. Estes questionários podem ser encontrados no Apêndice B.

## 7 AVALIAÇÃO DA SOLUÇÃO DESENVOLVIDA

Este capítulo apresenta o estudo de caso realizado, voltado à análise da viabilidade da solução desenvolvida, bem como a disposição e discussão dos resultados alcançados através da aplicação do método GQM, discutido no capítulo anterior.

### 7.1 Aplicação do Método GQM - Estudo de Caso

O estudo de caso, que visa a avaliação da solução proposta, ocorreu por meio da utilização do método GQM detalhado através do Plano GQM e do Plano de Medição e Análise. Por meio deles o software desenvolvido neste trabalho foi executado juntamente com as aplicações auxiliares (DimDimDim, Mayam, BancoOO e BancoOA), sendo as ações observadas transformadas em dados que foram coletados pelos questionários disponíveis no Apêndice B deste documento.

A seguir será apresentada a aplicação do método GQM no contexto de cada um dos softwares auxiliares, detalhando-se como ocorreu o estudo de caso voltado à análise da ferramenta aferidora da confiabilidade e eficiência. Esse estudo foi realizado pelo desenvolvedor da solução definida neste trabalho, com constatações explicitadas em questionários. As três metas presentes no Plano GQM foram examinadas da seguinte forma: a princípio observou-se a execução do software desenvolvido junto a cada uma das aplicações em foco, considerando, sobretudo, a geração dos aspectos-monitores; em seguida, como os monitores já gerados, partiu-se para a verificação da atuação desses, também associados a cada aplicação auxiliar utilizada.

#### 7.1.1 Análise da Execução da Solução

##### 7.1.1.1 Execução sob a Aplicação *DimDimDim*

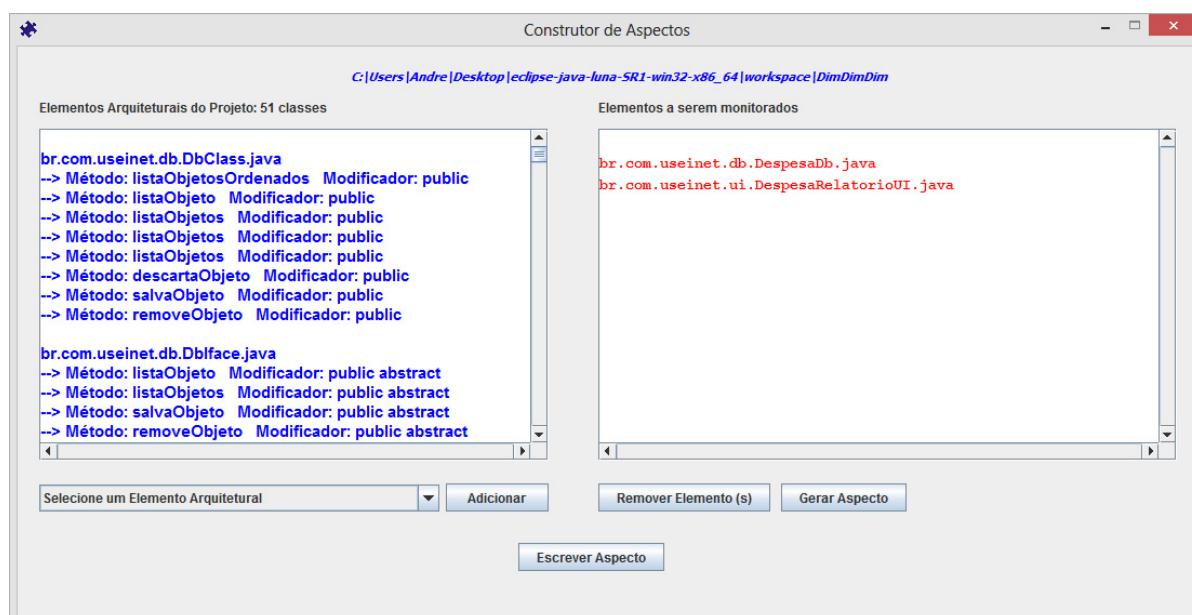
De acordo com o que foi exposto, a aplicação DimDimDim (MERCER, 2007), desenvolvida pela empresa UseInet, representa um sistema simples destinado à realização de controle financeiro pessoal. Para a execução da solução desenvolvida, foi necessário carregá-la, juntamente com o projeto do software DimDimDim, no ambiente de desenvolvimento *Eclipse IDE for Java Developers* (Versão: Luna Service Release 1 - 4.4.1 - Build id: 20140925-1800), combinado com o *AspectJ* (Versão: AJDT 2.2.4).

Após o carregamento dos projetos supracitados, a ferramenta avaliada foi executada pelo desenvolvedor, iniciado-se a observação do seu funcionamento e coleta de dados através do Questionário 1 que é encontrado no Apêndice B.

A avaliação da solução, executada sob o software DimDimDim, focou-se previamente em determinar se essa apresentava erros aparentes de funcionamento, como o não carregamento, falhas inesperadas, interrupções bruscas e os demais quesitos encontrados no

questionário imediatamente referido. A Figura 29 mostra a tela inicial da ferramenta afe-ridora de requisitos de qualidade, exibindo a localização do projeto da aplicação auxiliar, o número de elementos arquiteturais que o compõe, a disposição desses elementos (em azul) e ainda alguns elementos selecionados para monitoração (em vermelho) a nível de teste das funcionalidades desta interface.

**Figura 29 – Interface da Ferramenta Desenvolvida, sob o DimDimDim**



Fonte: Elaborada pelo autor

Ainda em relação ao funcionamento da ferramenta, observou-se a efetivação da geração dos aspectos monitores. Para tanto, diversas tentativas de geração foram realizadas: contendo apenas um elemento, mais de um elemento, todos os elementos e também sem conter elemento algum. Todas essas ações culminaram em esperados, isto é, quando a lista de elementos a serem monitorados apresenta um ou mais elementos e ocorria a ação Gerar Aspecto, o aspecto-monitor selecionado era criado; porém, quando essa lista encontrava-se vazia e a mesma ação era efetuada, o botão referente a geração do aspecto voltado à monitoração do sistema era desativado até a inserção de, ao menos, um elemento da já mencionada lista.

Em síntese, mediante as análises advindas da execução da ferramenta e das respostas dadas aos questionamentos, determina-se que a solução pôde ser iniciada, sem a ocorrência de erros evidentes; os elementos arquiteturais exibidos no software correspondem aos mesmos presentes na aplicação auxiliar, constatação realizada através da comparação desta exibição como o código do DimDimDim; após escolher um elemento para monitoração, o mesmo foi inserido na lista de elementos a serem monitorados; e a ação de remoção de um ou mais elementos da lista anteriormente citada, fez com que o mesmo fosse excluído dessa.

Cabe ainda destacar que a ação Gerar Aspecto, quando executada com êxito, ocasionou a geração de dois arquivos com extensão *.aj*, presentes no diretório \Aspectos do projeto da aplicação auxiliar. Trata-se dos arquivos *AspectoLogTempo.aj* e *AspectoLogExeccao.aj* que são, respectivamente, responsáveis pelo logging dos elementos escolhidos para monitoração e pelo registro das possíveis exceções decorrentes da utilização do sistema a ser monitorado, neste caso, o DimDimDim.

Enfim, além das observações colocadas, nota-se que a ação do botão Escrever Aspecto culminou na abertura da interface Editor de Aspectos que, após ter um aspecto-monitor escrito e salvo, esse é armazenado no diretório especificado pelo usuário. Dessa forma, infere-se que houve êxito no desempenho da solução ao gerar aspectos-monitores para a aplicação auxiliar alvo desta primeira análise.

#### 7.1.1.2 Execução sob a Aplicação *Mayam*

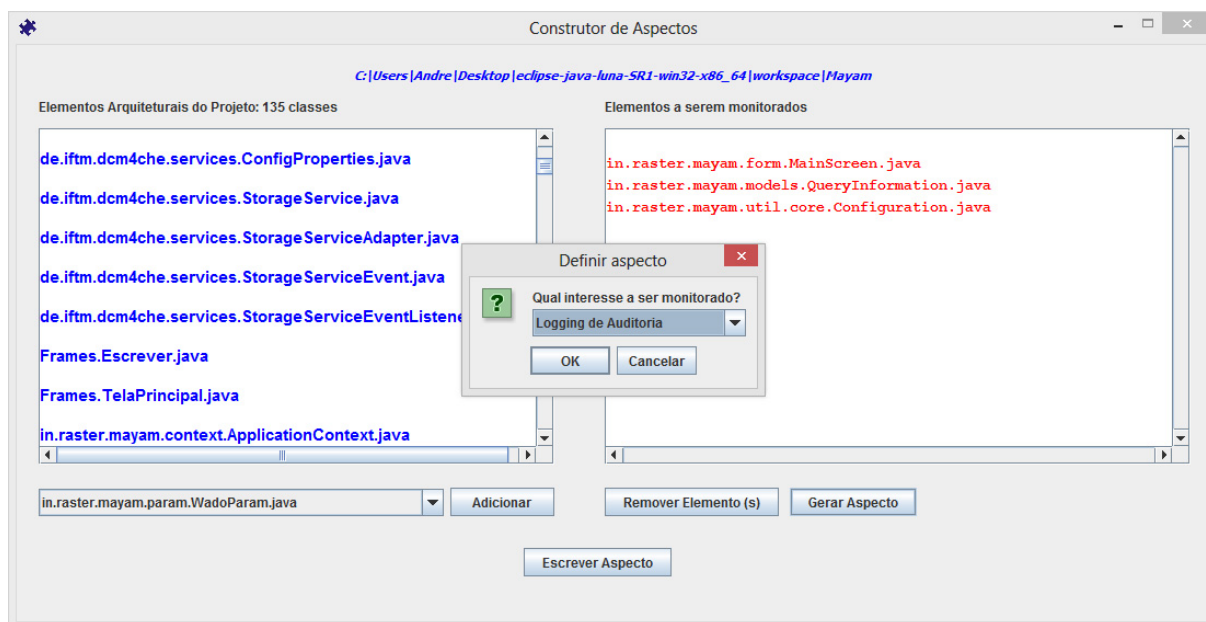
O software *Mayam* (DCM4CHE, 2014), conforme visto, corresponde a uma estação de trabalho *open source* responsável pela exibição de imagens no padrão DICOM, utilizado na área da medicina para armazenar e prover a comunicação padronizada de imagens geradas durante exames feitos pelos aparelhos de diagnósticos médicos. Ele é implementado usando o conjunto de *widgets* DCM4CHE Java toolkit, através do ambiente de desenvolvimento integrado NetBeans IDE, sendo desenvolvido na linguagem de programação orientada a objetos Java.

Da mesma forma que a execução vista na subseção anterior, o projeto da ferramenta desenvolvida e do *Mayam* foram carregados no ambiente de desenvolvimento Eclipse IDE, associado ao AspectJ; antes desse carregamento, o referido projeto precisou ser convertido, já que ele é originário de um outro ambiente de desenvolvimento. Logo, procedeu-se com a observação do funcionamento dessa ferramenta, verificando os mesmos quesitos da análise feita sob o DimDimDim, ou seja, apenas o contexto da aplicação auxiliar foi mudado, passando de um software simples (com 51 classes) para um de maior porte (com 135 classes). É importante deixar claro que a coleta de dados acerca da mencionada execução, bem como das próximas análises, serão sempre realizadas através da observação da operação dos software e da resolução dos questionários desenvolvidos para este fim.

A Figura 30 apresenta a ferramenta aferidora de requisitos de qualidade em funcionamento sob o projeto do *Mayam*. Através dela, é possível notar a exibição dos mesmos dados vistos na avaliação anterior, porém no contexto da aplicação auxiliar em questão.

Por meio das respostas dadas ao Questionário 1, conclui-se que a solução desenvolvida pôde ser iniciada, sem a ocorrência de erros evidentes que impedissem sua execução. Após uma análise do projeto da aplicação em questão, percebeu-se que os elementos arquiteturais listados no software correspondem aos mesmos presentes naquele; contudo, diferentemente do observado com a aplicação DimDimDim, a execução sob o *Mayam* não apresentou a listagem dos métodos que compõem cada classe. Ademais, a escolha de

Figura 30 – Interface da Ferramenta Desenvolvida, sob o Mayam



Fonte: Elaborada pelo autor

um elemento para monitoração resultou na sua inserção na lista de elementos a serem monitorados, bem como a ação de remoção de algum elemento dessa lista, fez o mesmo ser retirado e reposicionado na caixa de combinação que comporta os elementos a serem selecionados.

Nota-se ainda a ocorrência da ação Gerar Aspecto, com a lista de elementos a serem monitorados vazia, contendo um ou mais elementos e todos os elementos do projeto. Nestas ações, constatou-se que o software comporta-se como previsto, ou seja, quando a citada lista não contém elemento algum, o botão Gerar Aspecto permanece desabilitado; ao passo que um, mais de um ou todos os elementos vão sendo inseridos, esse botão é habilitado e sua ativação resulta na disposição das opções de aspectos-monitores que podem ser gerados.

Enfim, quando efetivada com sucesso, a ação Gerar Aspecto cria dois arquivos - *AspectoLogTempo.aj*, voltado aos logging dos elementos escolhidos para monitoração e *AspectoLogExcecao.aj*, responsável pelo registro das exceções - no diretório \Aspectos do projeto Mayam. Ademais, a ativação do botão Escrever Aspecto fez com que interface Editor de Aspectos fosse exibida, sendo armazenado no diretório especificado pelo usuário, o aspecto-monitor implementado nessa. Logo, a execução da ferramenta desenvolvida sob a aplicação Mayam obteve êxito.

#### 7.1.1.3 Execução sob as Aplicações *BancoOO* e *BancoOA*

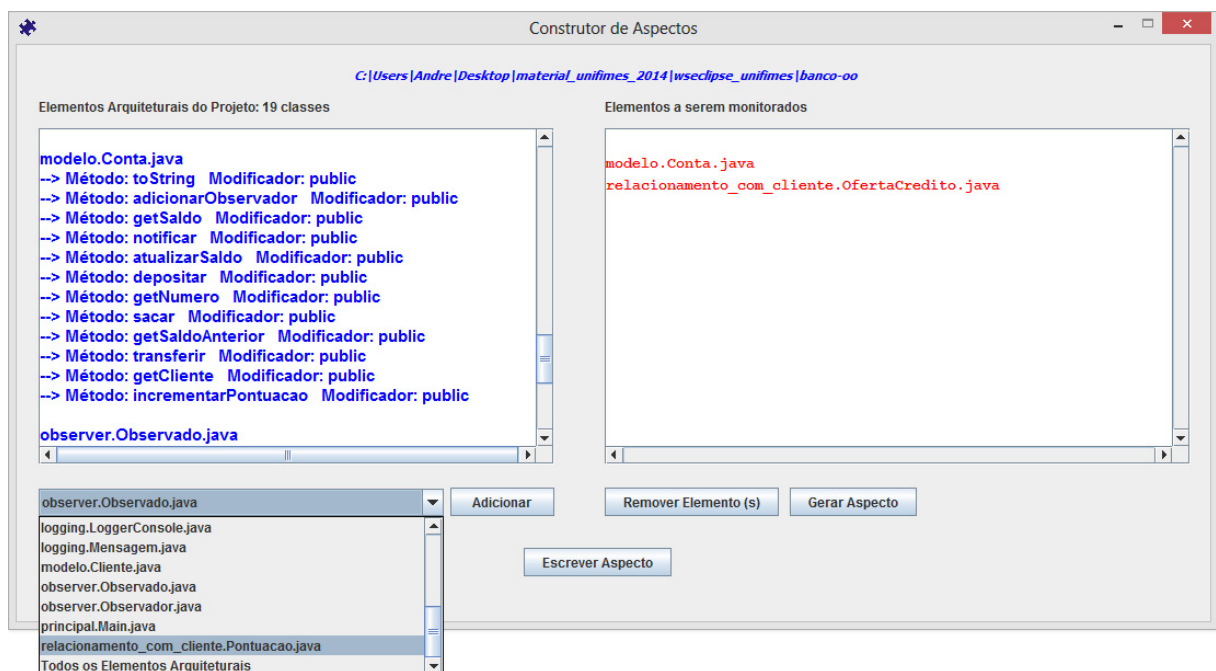
As aplicações BancoOO e BancoOA fazem parte do projeto BancoUnifimes (Júnior; AFONSO, 2014) que dispõe de duas versões para um mesmo sistema bancário hipotético

simples. De forma geral, esses softwares oferecem as funcionalidade de saque, depósito, transferência, programa de relacionamento com o cliente e ainda a possibilidade de crédito especial. Tais aplicações foram desenvolvidas na linguagem de programação Java, a primeira utilizando unicamente a programação orientada a objetos e a segunda valendo-se também da programação orientada a aspectos.

O projeto BancoUnifimes foi, junta à ferramenta desenvolvida, carregado no Eclipse IDE dotado do plugin AspectJ. O primeiro software verificado foi o BancoOO e, em seguida, o BancoOA. É importante destacar que o objetivo principal da análise dessa execução consistiu em observar se a solução apresentada possuía comportamentos diversos, quando submetida a aplicações implementadas, convencionalmente, pela POO e a sistemas com implementação auxiliada pela POA. Para que essa comparação ocorresse conforme idealizado, buscou-se por softwares iguais, isto é, com as mesmas funcionalidades e, preferencialmente, que fossem duas versões de um mesmo sistema, fato que é observado no projeto BancoUnifimes.

Dessa forma, a ferramenta implementada neste trabalho foi executada sob o BancoOO, cujo funcionamento foi observado e mensurado através de questionário. Neste cenário, a Figura 31 expõe a destacada execução, podendo ser percebida a localização do projeto da aplicação auxiliar, o número de elementos arquiteturais que o compõe, a listagem desses elementos e a disposição de dois elementos passíveis de monitoração. Nota-se ainda, logo abaixo dos elementos arquiteturais que formam o BancoOO, a visualização de uma caixa de combinação, na qual é possível selecionar um ou mais elementos para serem monitorados.

Figura 31 – Interface da Ferramenta Desenvolvida, sob o BancoOO

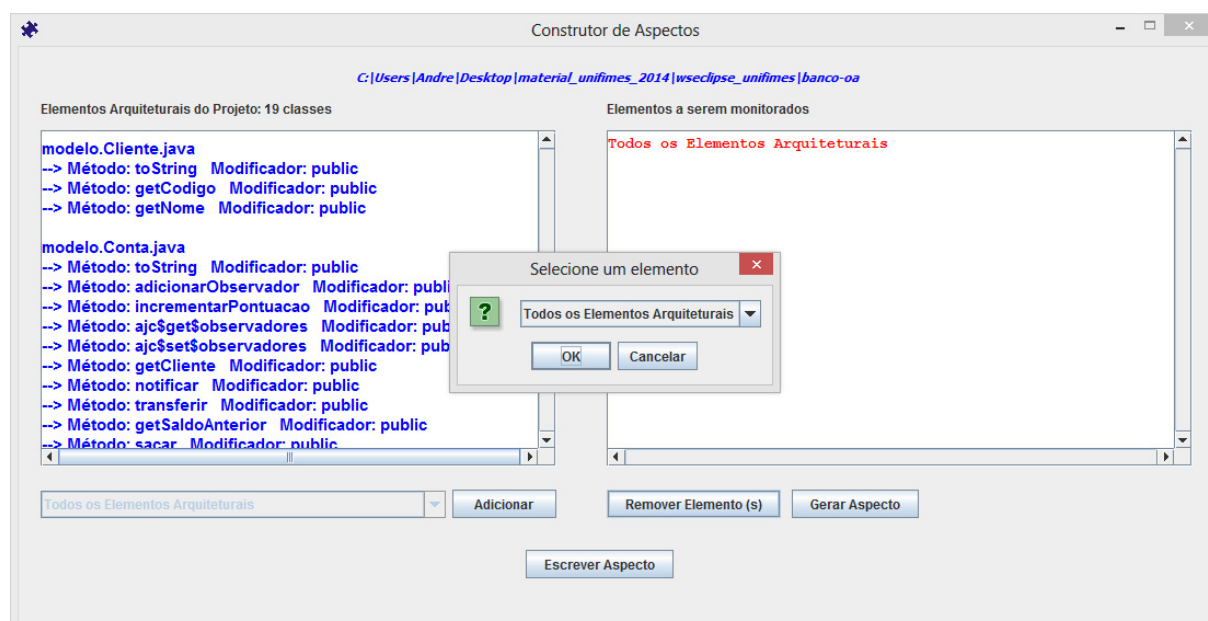


Analisando as respostas dadas ao Questionário 1, preenchido no decorrer da execução anteriormente discutida, chegou-se às seguintes conclusões: a solução desenvolvida pôde ser iniciada, sem a ocorrência de erros que comprometessem seu funcionamento; os elementos arquiteturais listados no software correspondem aos mesmos presentes no projeto da aplicação auxiliar; a seleção de um elemento para monitoração culminou na sua inserção na lista de elementos a serem monitorados; e a exclusão de um elemento da lista resultou na sua eliminação dessa e no seu reposicionamento na caixa de combinação.

Além do mais, a ação Gerar Aspecto ocasionou a criação dos aspectos *AspectoLogTempo.aj* e *AspectoLogExcecao.aj* no diretório \Aspectos do projeto BancoOO, ao ser realizada quando continha um ou mais elementos na lista de elementos a serem monitorados. Salienta-se ainda a efetivação do botão Escrever Aspecto e da interface Editor de Aspectos, fazendo com que aspectos-monitores fossem implementados e armazenados no diretório especificado pelo usuário.

Concluída a análise do funcionamento da ferramenta desenvolvida no contexto do BancoOO, partiu-se para a verificação da sua execução sob a aplicação BancoOA. Para tanto, os projetos foram inseridos no Eclipse IDE e a mensuração do comportamento observado, igualmente às análises anteriores, foi feita através do Questionário 1. Na Figura 32 é possível visualizar a tela inicial da ferramenta aferidora de requisitos de qualidade, destacando os mesmos elementos apresentados na Figura 31, mas no âmbito do BancoOA. Nessa figura é salientada também a seleção para monitoração (em vermelho) de todos os elementos arquiteturais; bem como a caixa de diálogo (ao centro), exibida por uma ação no botão Remover Elemento(s), que possibilita a remoção e conseqüente esvaziamento da lista de elementos arquiteturais a serem monitorados.

**Figura 32 – Interface da Ferramenta Desenvolvida, sob o BancoOA**



Em suma, a execução da ferramenta sob as duas aplicações provenientes do projeto BancoUnifimes foi realizada com êxito, ou seja, o desempenho da solução ao exibir elementos arquiteturais sujeitos à monitoração, ao gerar aspectos-monitores e ao possibilitar o desenvolvimento desses aspectos foi efetivado. Ressalta-se que este estudo, mesmo sendo realizado na amplitude de aplicações com diferentes implementações - POO e POA -, não obteve resultados totalmente distintos nem incoerentes ou inesperados.

### 7.1.2 Análise do Aspecto-monitor da Confiabilidade

Feita a verificação da geração dos aspectos-monitores, resta observar o desempenho desses quando utilizados para monitorar os requisitos de qualidade Confiabilidade e Eficiência de cada uma das aplicações auxiliares, que conterão em seus projetos esses aspectos previamente gerados. O primeiro passo para realizar esta verificação consiste em determinar qual ou quais os elementos que serão alvo da monitoração, para tanto algumas funcionalidades foram definidas como primordiais nos softwares, aquelas que correspondem ao objetivo do negócio do sistema.

#### 7.1.2.1 Análise sob a Aplicação *DimDimDim*

Na aplicação *DimDimDim*, as funcionalidades escolhidas para monitoração foram: cadastro de renda, cadastro de tipo, cadastro de despesa e geração de relatório renda x despesa. Elas estão correlacionadas, respectivamente, às classes *RendaAction*, *TipoAction*, *DespesaAction* e *RendaDespesaAction* que serão definidas como foco do aspecto-monitor da confiabilidade.

Conforme esperado - por se tratar de uma versão já distribuída e, possivelmente, amplamente testada - a execução do *DimDimDim* e efetivação das quatro funcionalidades abordadas foi realizada sem a ocorrência de erros, ao utilizar os valores determinados para cada campo, isto é, concatenação de caracteres nos campos *String* e concatenação de números nos campos *Double*. Assim, para verificar se os erros passavam despercebidos ou se eram capturados pelo aspecto-monitor da confiabilidade, validando-o conseqüentemente, foram implantados erros de *divisão por zero* em métodos aleatórios de cada classe monitorada.

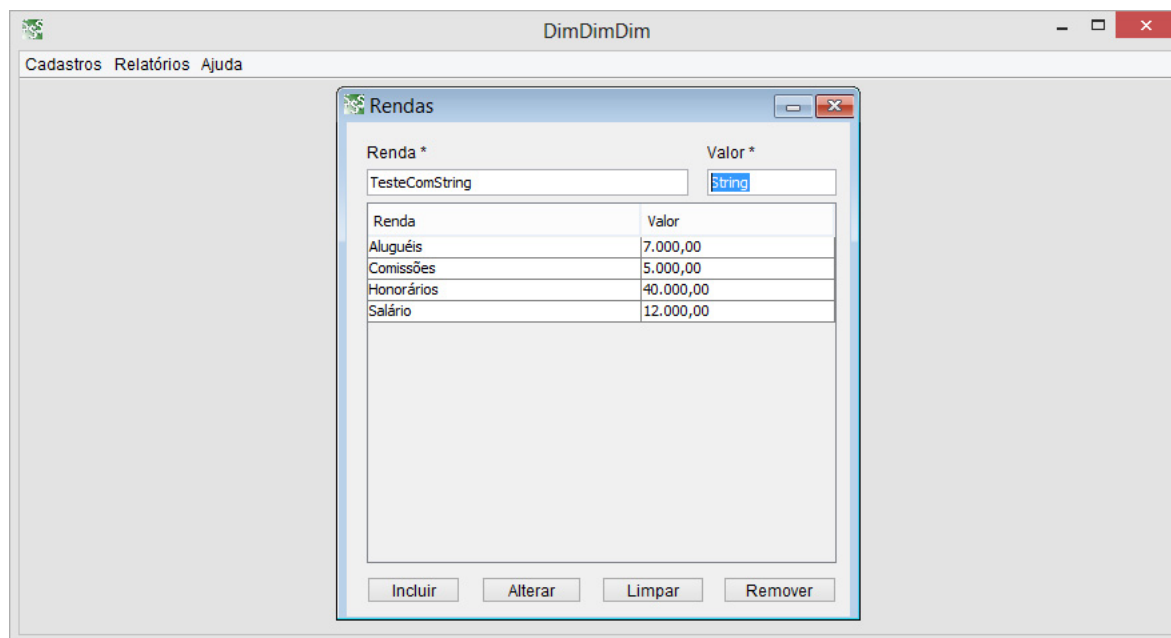
Um outro tipo de exceção prevista, além das geradas pelas divisões por zero, eram as exceções de negócio que se originariam em duas situações: a primeira quando uma *String* fosse inserida em algum campo, em Renda ou Despesa, setado com o tipo de valor *Double* e a segunda ao passo que fosse solicitado o relatório renda x despesa, estando os valores das rendas e/ou despesas zerados.

Partiu-se, portanto, para uma nova execução da aplicação, considerando os critérios de constatação de exceções pré-estabelecidos. Como pode ser visto na Figura 33, após quatro inserções de valores adequados (Aluguéis, Comissões, Honorários e Salário), foi realizada



uma tentativa de registro passível de gerar uma exceção: a introdução de caracteres em um campo definido para aceitar apenas valores decimais.

**Figura 33 – Inserção de Valores no DimDimDim**



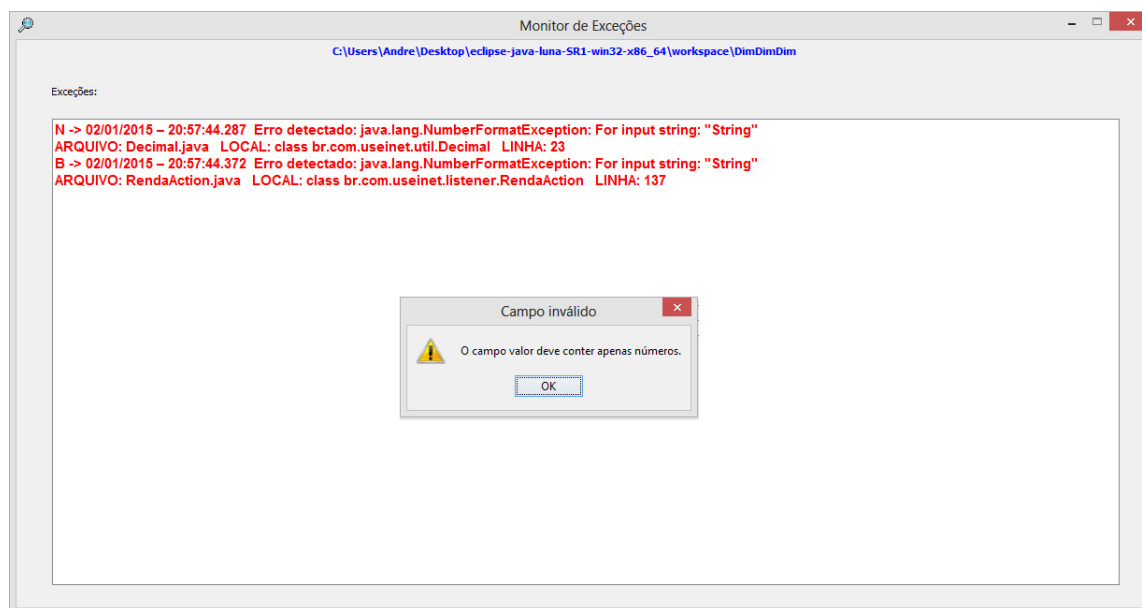
Fonte: Elaborada pelo autor

O resultado do teste realizado foi a exibição de uma caixa de diálogo, oferecida pela aplicação auxiliar, para enfatizar o erro observado e também a disposição da tela Monitor de Exceções (ver Figura 34), destacando a data e horário que o erro foi detectado, o tipo do erro, o arquivo correspondente e a linha a partir da qual ele foi lançado. Nota-se que as mesmas exceções vistas na interface oferecida pelo aspecto-monitor da confiabilidade foram retidas e armazenadas no arquivo *logExc.txt* (ver Figura 35) presente no diretório \Logs do projeto do DimDimDim no Eclipse IDE.

Em síntese, as demais verificações também resultaram em exceções e a consequente abertura da janela Monitor de Exceções, mostrando, em tempo de execução, todas as exceções decorrentes da utilização do DimDimDim. Da mesma forma, ocorreu o registro, de tudo o que era exibido nesta tela, no já referenciado arquivo *logExc.txt*. A nível de ilustração, a Figura 36 expõe as demais exceções mencionadas que ocorreram no software alvo da monitoração, abrangendo as exceções de negócio (marcadas com N e B) e aquelas ocasionadas por erros/bugs (marcadas apenas com B).

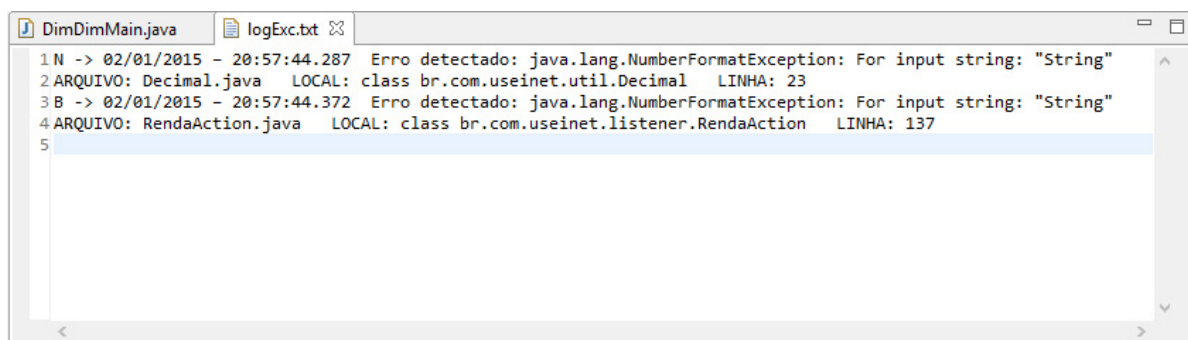
Enfim, observando tanto a execução do DimDimDim quanto as respostas ao Questionário 2, determina-se que: o aspecto-monitor foi iniciado sem a ocorrência de erros, ao passo que a primeira exceção foi observada; houve a possibilidade de seleção dos elementos arquiteturais a serem monitorados; houve a ocorrência de exceções de negócio e de exceções decorrentes de bugs, sendo diferenciadas; as exceções puderam ser visuali-

Figura 34 – Tela Monitor de Exceções - Observação das Exceções do DimDimDim



Fonte: Elaborada pelo autor

Figura 35 – Arquivo *logExc.txt* com as Exceções Armazenadas



Fonte: Elaborada pelo autor

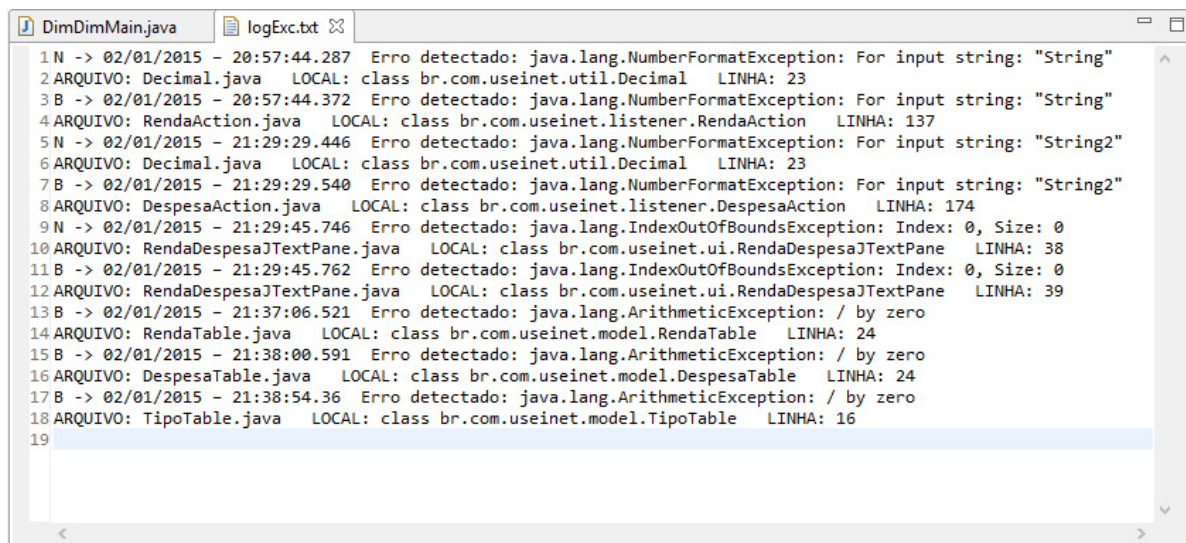
zadas em uma interface gráfica específica; e os dados dessa visualização foram retidos e armazenados em arquivo e diretório específico no projeto da aplicação em monitoração.

#### 7.1.2.2 Análise sob a Aplicação *Mayam*

Na aplicação *Mayam*, as funcionalidades escolhidas para monitoração foram importar, excluir e visualizar exames gerados a partir de um aparelho de Tomografia Computadorizada (TC), que estão associadas às classes *ImportDcmDirDelegate* e *ApplicationContext*. Para que essas operações fossem realizadas, contou-se com a utilização de dois conjuntos de imagens DICOM (PHENIX e VIX) presentes na base de dados da aplicação *Osirix Viewer*<sup>5</sup>.

<sup>5</sup> Disponível em: <<http://www.osirix-viewer.com/datasets/>>. Acesso: dez. 2014.

Figura 36 – Exceções Retidas e Armazenadas pelo Aspecto-monitor da Confiabilidade no DimDimDim



```

DimDimMain.java  logExc.txt
1 N -> 02/01/2015 - 20:57:44.287 Erro detectado: java.lang.NumberFormatException: For input string: "String"
2 ARQUIVO: Decimal.java LOCAL: class br.com.useinet.util.Decimal LINHA: 23
3 B -> 02/01/2015 - 20:57:44.372 Erro detectado: java.lang.NumberFormatException: For input string: "String"
4 ARQUIVO: RendaAction.java LOCAL: class br.com.useinet.listener.RendaAction LINHA: 137
5 N -> 02/01/2015 - 21:29:29.446 Erro detectado: java.lang.NumberFormatException: For input string: "String2"
6 ARQUIVO: Decimal.java LOCAL: class br.com.useinet.util.Decimal LINHA: 23
7 B -> 02/01/2015 - 21:29:29.540 Erro detectado: java.lang.NumberFormatException: For input string: "String2"
8 ARQUIVO: DespesaAction.java LOCAL: class br.com.useinet.listener.DespesaAction LINHA: 174
9 N -> 02/01/2015 - 21:29:45.746 Erro detectado: java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
10 ARQUIVO: RendaDespesaJTextPane.java LOCAL: class br.com.useinet.ui.RendaDespesaJTextPane LINHA: 38
11 B -> 02/01/2015 - 21:29:45.762 Erro detectado: java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
12 ARQUIVO: RendaDespesaJTextPane.java LOCAL: class br.com.useinet.ui.RendaDespesaJTextPane LINHA: 39
13 B -> 02/01/2015 - 21:37:06.521 Erro detectado: java.lang.ArithmeticException: / by zero
14 ARQUIVO: RendaTable.java LOCAL: class br.com.useinet.model.RendaTable LINHA: 24
15 B -> 02/01/2015 - 21:38:00.591 Erro detectado: java.lang.ArithmeticException: / by zero
16 ARQUIVO: DespesaTable.java LOCAL: class br.com.useinet.model.DespesaTable LINHA: 24
17 B -> 02/01/2015 - 21:38:54.36 Erro detectado: java.lang.ArithmeticException: / by zero
18 ARQUIVO: TipoTable.java LOCAL: class br.com.useinet.model.TipoTable LINHA: 16
19

```

Fonte: Elaborada pelo autor

A princípio, é fundamental lembrar que o Mayam é um software desenvolvido no NetBeans IDE e não no Eclipse IDE como as outras aplicações que estão sendo utilizadas nesta avaliação. Sendo assim, além de verificar o comportamento da solução desenvolvida em um sistema de maior porte, como é o caso no Mayam, será possível determinar o referido comportamento sob a ótica de aplicações implementadas em diferentes ambientes de desenvolvimento. Outro ponto a ser destacado corresponde a análise de apenas duas classes para três funcionalidades conferidas, este fato ocorre porque o software alvo apresenta uma alta granularidade em relação as suas classes e métodos, concentrando, posteriormente, um conjunto de ações em algumas classes específicas.

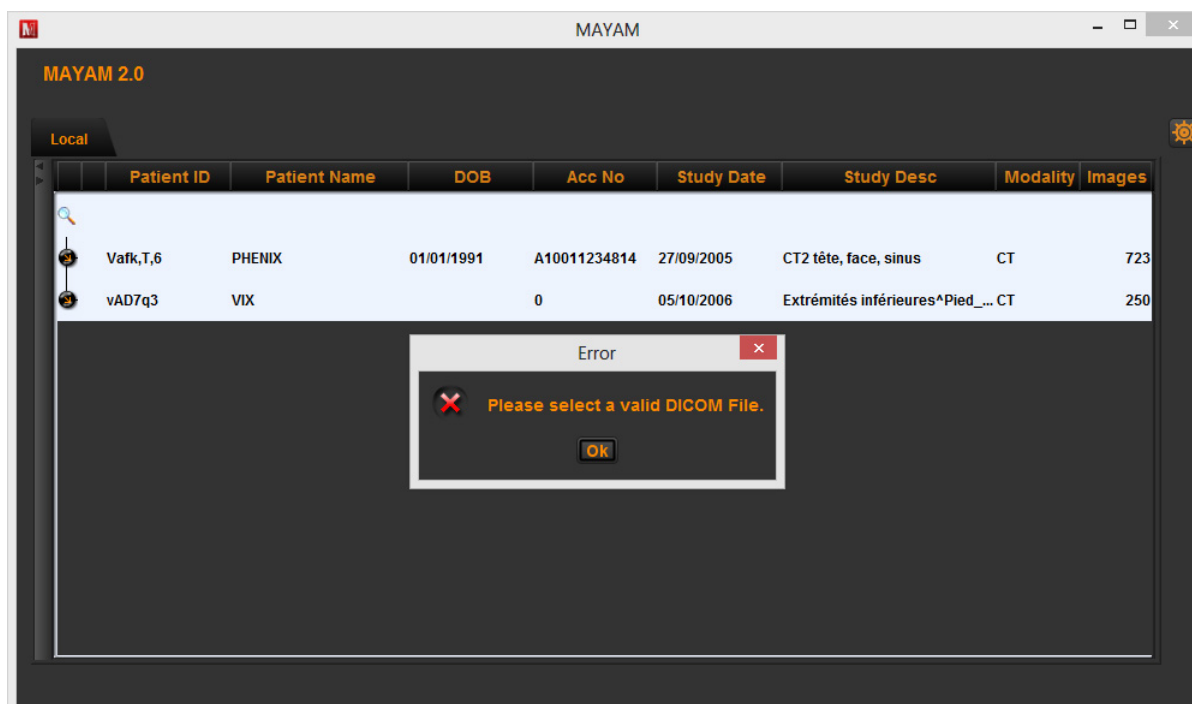
Logo, a execução do Mayam foi realizada, carregando os conjuntos de imagens PHENIX e VIX para o contexto dessa aplicação. Em seguida, esses foram visualizados e mais adiante excluídos. Não foi observada a ocorrência de qualquer exceção durante o processo realizado, já que as imagens utilizadas estavam no padrão DICOM e, por ser um software amplamente difundido entre os profissionais que lidam com o Diagnóstico por Imagem, presume-se que foi largamente testado.

Contudo, afim de observar a efetivação do aspecto-monitor da confiabilidade foi feita a tentativa de carregamento de imagens nos formatos JPEG, GIF, PNG, Bitmap, TIFF, SVG e RAW, além de arquivos nos formatos PDF, PPT e TXT. Além do mais, para reforçar a atividade do destacado aspecto, foram implantadas divisões por zero em métodos aleatórios das classes monitoradas. Tomando como base esses critérios, um nova execução do Mayam foi realizada.

A Figura 37 mostra os dois conjuntos de imagens DICOM - PHENIX e VIX - carregados na aplicação e o resultado da tentativa de carregamento de um arquivo JPEG. Essa

resultou no lançamento de uma exceção de negócio e na consequente disposição da caixa de diálogo vista no centro da figura em questão, solicitando ao usuário a seleção de um arquivo DICOM válido. Constatou-se a mesma situação ao selecionar os demais formatos de imagens e arquivos (GIF, PNG, Bitmap, TIFF, SVG, RAW, PDF, PPT e TXT).

**Figura 37 – Inserção de Imagens no Mayam**



Fonte: Elaborada pelo autor

Da mesma forma, ao ser executada a divisão por zero, o aspecto-monitor da confiabilidade dispôs o erro encontrado, informando a data e horário que ele foi detectado, seu tipo, o arquivo correspondente e a linha a partir da qual ele foi gerado. Ressalta-se que tanto essa exceção quanto aquelas lançadas pelo carregamento de arquivos diferentes do DICOM, resultaram na exibição dos dados das suas ocorrências na tela Monitor de Exceções, além da retenção e armazenamento destes dados no arquivo *logExc.txt*, conforme visto na Figura 38, situado no diretório \Logs do projeto do Mayam no Eclipse IDE.

Portanto, analisando o funcionamento do Mayam e as respostas dadas ao Questionário 2, conclui-se que: o aspecto-monitor foi iniciado sem a ocorrência de erros, ao passo que a primeira exceção foi observada; houve a possibilidade de seleção dos elementos arquitetais a serem monitorados; houve a ocorrência de exceções de negócio (N) e de exceções decorrentes de bugs (B), sendo diferenciadas; as exceções puderam ser visualizadas em uma interface gráfica específica; e os dados dessa visualização foram retidos e armazenados em arquivo e diretório específico no projeto da aplicação em monitoração.

Figura 38 – Exceções Retidas e Armazenadas pelo Aspecto-monitor da Confiabilidade no Mayam

```

MainScreen.java ApplicationFacade.java logExc.txt
1 N ->05/01/2015 - 11:29:25.901 Erro detectado: java.io.FileNotFoundException: C:\Users\Andre\Desktop\figs_ava
2 ARQUIVO: ImageView.java LOCAL: class in.raster.mayam.form.ImageView LINHA: 275
3 N ->05/01/2015 - 12:04:21.597 Erro detectado: org.dcm4che2.io.DicomCodingException: Not a DICOM Stream
4 ARQUIVO: ImportDcmDirDelegate.java LOCAL: class in.raster.mayam.delegates.ImportDcmDirDelegate LINHA: 202
5 N ->05/01/2015 - 12:04:52.438 Erro detectado: org.dcm4che2.io.DicomCodingException: Not a DICOM Stream
6 ARQUIVO: ImportDcmDirDelegate.java LOCAL: class in.raster.mayam.delegates.ImportDcmDirDelegate LINHA: 202
7 ARQUIVO: ImportDcmDirDelegate.java LOCAL: class in.raster.mayam.delegates.ImportDcmDirDelegate LINHA: 202
8 N ->05/01/2015 - 12:08:10.174 Erro detectado: org.dcm4che2.io.DicomCodingException: Not a DICOM Stream
9 ARQUIVO: ImportDcmDirDelegate.java LOCAL: class in.raster.mayam.delegates.ImportDcmDirDelegate LINHA: 202
10 N ->05/01/2015 - 12:08:31.979 Erro detectado: org.dcm4che2.io.DicomCodingException: Not a DICOM Stream
11 ARQUIVO: ImportDcmDirDelegate.java LOCAL: class in.raster.mayam.delegates.ImportDcmDirDelegate LINHA: 202
12 B ->05/01/2015 - 12:15:49.772 Erro detectado: java.lang.ArithmeticException: / by zero
13 ARQUIVO: ImportDcmDirDelegate.java LOCAL: class in.raster.mayam.delegates.ImportDcmDirDelegate LINHA: 142
14 B ->05/01/2015 - 12:15:49.851 Erro detectado: java.lang.ArithmeticException: / by zero
15 ARQUIVO: ImportDcmDirDelegate.java LOCAL: class in.raster.mayam.delegates.ImportDcmDirDelegate LINHA: 91
16 B ->05/01/2015 - 12:17:12.368 Erro detectado: java.lang.ArithmeticException: / by zero
17 ARQUIVO: DatabaseHandler.java LOCAL: class in.raster.mayam.util.database.DatabaseHandler LINHA: 1426
18 B ->05/01/2015 - 12:17:12.414 Erro detectado: java.lang.ArithmeticException: / by zero
19 ARQUIVO: MainScreen.java LOCAL: class in.raster.mayam.form.MainScreen LINHA: 571
20 B ->05/01/2015 - 12:17:12.430 Erro detectado: java.lang.ArithmeticException: / by zero
21 ARQUIVO: MainScreen.java LOCAL: class in.raster.mayam.form.MainScreen LINHA: 541
22 B ->05/01/2015 - 12:17:38.555 Erro detectado: java.lang.ArithmeticException: / by zero
23 ARQUIVO: MainScreen.java LOCAL: class in.raster.mayam.form.MainScreen LINHA: 192
24 B ->05/01/2015 - 12:17:38.571 Erro detectado: java.lang.ArithmeticException: / by zero
25 ARQUIVO: MainScreen.java LOCAL: class in.raster.mayam.form.MainScreen$1 LINHA: 111
26 |
  
```

Fonte: Elaborada pelo autor

### 7.1.2.3 Análise sob as Aplicações *BancoOO* e *BancoOA*

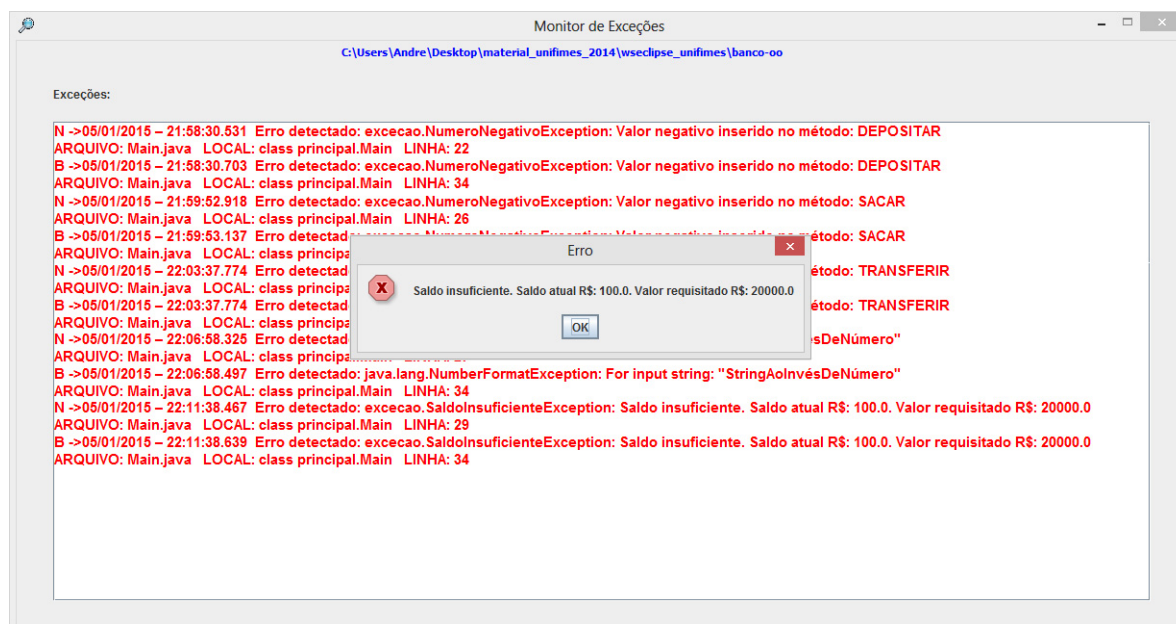
Nas aplicações do projeto BancoUnifimes (BancoOO e BancoOA), as funcionalidades escolhidas para monitoração foram depositar, sacar e transferir que são definidas pela classe *Conta*. Por serem softwares simples, voltados apenas para a realização de testes no contexto OO e OA, essas aplicações contam com um número pequeno de classes, agrupando algumas funcionalidades em apenas uma delas.

A execução do BancoOO, justamente com o respectivo aspecto-monitor da confiabilidade, foi a primeira a ser realizada. A princípio, após serem criados três clientes que foram associados a três contas correntes, apenas valores considerados aceitáveis - strings numéricas - foram inseridos no método *depositar*, formando-se um saldo. Em seguida, números positivos e menores que o saldo foram dispostos no método sacar. Finalmente, outros valores igualmente positivos e menores que o saldo resultante foram inseridos no método transferir, simulando a transferência de dinheiro entre as contas.

Durante as operações anteriormente descritas, não se observou qualquer lançamento de exceção, sendo exibido no console todo o registro das ações efetivadas no sistema. Desse modo, buscou-se a verificar o efetivo funcionamento do aspecto-monitor aqui utilizado, através da inserção de valores que fogem do convencional, ou seja, números negativos e caracteres nos métodos depositar, sacar e transferir, bem como saques e transferências maiores que o saldo. A Figura 39 mostra as exceções lançadas pelo BancoOO, a partir da introdução dos valores anteriormente citados, e retidas pelo aspecto alvo desta verificação.



Figura 39 – Tela Monitor de Exceções - Observação das Exceções do BancoOO



Fonte: Elaborada pelo autor

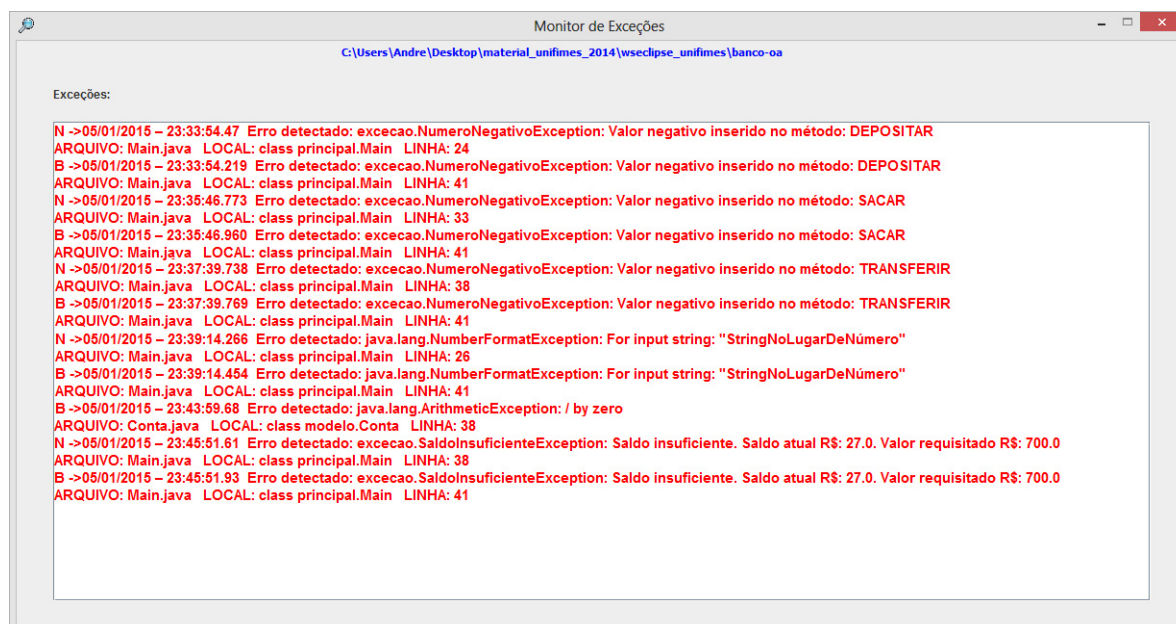
Nota-se que, ao centro dessa figura, é vista uma caixa de diálogo aberta pela aplicação monitorada, informando acerca da última exceção lançada. Percebe-se que todas as exceções vistas nessa figura remota-se à classe *Main*. Embora o monitor da confiabilidade esteja agindo sobre a classe *Conta*, os argumentos para os parâmetros dos métodos depositar, sacar e transferir são determinados na classe principal; logo o aspecto-monitor, de forma indireta, também irá observá-la.

Além das exceções de negócio destacadas na figura imediatamente apresentada, foram implantadas divisões por zero em métodos aleatórios da classe *Conta* a fim de simular um bug no software e a sua conseqüente constatação. Mais uma vez o aspecto-monitor da confiabilidade se comportou conforme previsto: exibindo a exceção na interface Monitor de Exceções e armazenando-a no arquivo *logExc.txt* dentro diretório \Logs do projeto BancoOO.

Da mesma forma que as verificações realizadas em cima da aplicação BancoOO, procedeu-se a análise sob o BancoOA que conta com as mesmas funcionalidades dessa, porém com parte delas implementada com a programação orientada a aspectos. Em um primeiro momento, apenas strings numéricas e positivas foram utilizadas nos métodos depositar, sacar e transferir. A execução dos métodos, neste caso, não apresentou exceção alguma. Assim, seguiu-se para a inserção dos mesmos valores usados na análise do software orientado a objetos. O resultado da monitoração das exceções advindas do BancoOA pode ser visto na Figura 40.

Em síntese, a figura expõe a tela Monitor de Exceções que dispõe de todas as exceções decorrentes da utilização da aplicação auxiliar. Ressalta-se que todos os dados vistos nesta

Figura 40 – Tela Monitor de Exceções - Observação das Exceções do BancoOA



Fonte: Elaborada pelo autor

interface foram armazenados no diretório \Logs do projeto BancoOO, especificamente no arquivo *logExc.txt*.

Enfim, ao analisar o funcionamento do aspecto-monitor da confiabilidade sob as aplicações BancoOO e BancoOA, bem como as respostas dadas ao Questionário 2, determina-se que: assim que houve a percepção da primeira exceção, o aspecto foi iniciado sem a ocorrência de erros; ocorreu a possibilidade de seleção dos elementos arquiteturais a serem monitorados; notou-se a existência de exceções de negócio (N-B) e de exceções decorrentes de bugs (B), sendo diferenciadas; as exceções puderam ser vistos em uma interface gráfica específica, o Monitor de Exceções; os dados visualizados foram retidos e armazenados em arquivo e diretório específico, dentro do projeto de cada aplicação em monitoração; e não houve comportamentos distintos, tendo em vista as diferentes formas de implementação dos softwares monitorados.

### 7.1.3 Análise do Aspecto-monitor da Eficiência

Os mesmos pressupostos definidos e utilizados para a verificação do aspecto-monitor discutido na Subseção 7.1.2, serão tomados para analisar a monitoração da eficiência nas mesmas aplicações auxiliares anteriormente usadas. O ponto principal a ser observado consiste na visualização e armazenamento das funcionalidades ou ações executadas pelos elementos arquiteturais monitorados, retendo dados como o tempo de resposta de cada método.

Enfatiza-se que serão consideradas aceitáveis as execuções que não ultrapassem 0,5 segundos ou 500000000 nanossegundos (ns). É importante sublinhar que o tempo de

resposta aceitável ou não, para determinada aplicação ou parte dela, varia de acordo com a regra de negócio; logo, a determinação do limiar aqui disposto ocorre apenas para fins de teste.

### 7.1.3.1 Análise sob a Aplicação *DimDimDim*

Na aplicação auxiliar *DimDimDim*, conforme já estabelecido, os elementos arquiteturais que serão monitorados pelo aspecto-monitor da eficiência serão aqueles que foram tomados para a análise da monitoração da confiabilidade, ou seja, as classes *RendaAction*, *TipoAction*, *DespesaAction* e *RendaDespesaAction* - que estão relacionadas às ações cadastro de renda, cadastro de tipo, cadastro de despesa e geração de relatório renda x despesa.

Com o aspecto-monitor gerado e inserido no projeto do *DimDimDim*, partiu-se para a execução deste software. Foram inseridas quatro rendas - Salário (15.000,00), Aluguéis (20.000,00), Pensão (4.000,00) e Lucro (10.000,00) -, cinco tipos de despesas - Alimentação, Lazer, Moradia, Saúde e Serviços - e seis despesas - Supermercado (1.000,00; Alimentação), Cinemas e Shows (700,00; Lazer), Viagem (6.000,00; Lazer), Odontologista (500,00; Saúde); Telefone (200,00; Serviços) e Condomínio (650,00; Moradia). Houve ainda a geração do relatório renda x despesa.

Por meio da execução das supracitadas ações, ocorreu a captura e o armazenamento de dados relativos às classes monitoradas, como a assinatura dos métodos, o tempo inicial no momento da execução, o tempo final e ainda tempo de resposta. Neste cenário, a visualização do logging se fez a partir da tela Monitor de Logging que foi exibida ao passo que a primeira monitoração foi realizada.

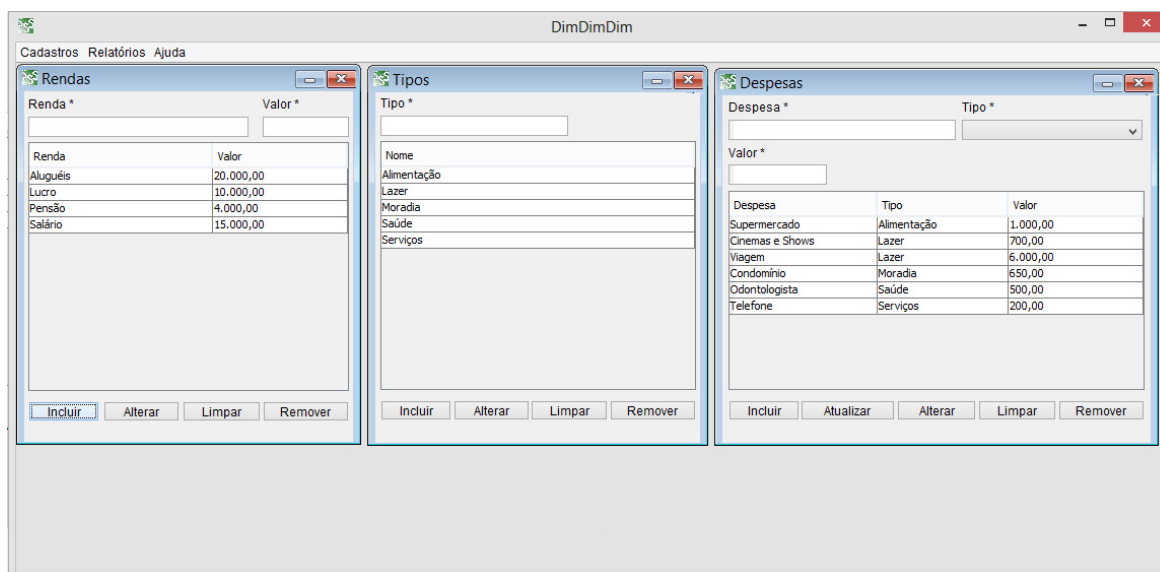
A seguir são apresentadas duas figuras referentes à aplicação auxiliar em análise. Na Figura 41 é exibida a interface do *DimDimDim* após a execução das funcionalidades de cadastro de renda, tipo de despesa e cadastro de despesa. Em seguida, na Figura 42, é exposto o Monitor de Logging que dispõe, na caixa de texto superior, do registro de todas as ações efetivadas por meio das classes *RendaAction*, *TipoAction*, *DespesaAction*.

Na Figura 42 ainda são vistos: na parte inferior esquerda, um campo com os métodos que extrapolaram o tempo de resposta aceitável; logo abaixo, uma caixa para o estabelecimento deste tempo; e, à direita, a caixa de texto que exhibe as exceções já armazenadas, presentes no arquivo *logExc.txt*. Destaca-se que tempo o aceitável de resposta foi, posteriormente, setado como 0 ns, para evidenciar a captura de métodos que ultrapassem o limiar estabelecido. Este fato foi constatado, isto é, o aspecto-monitor reteve os dados de todos os métodos, já que eles possuíam tempos de resposta superiores a 0 ns.

Diante da execução do *DimDimDim*, acrescido do aspecto-monitor da eficiência, além da verificação das respostas dadas ao Questionário 3, constatou-se que o aspecto foi iniciado sem a ocorrência de erros; houve a possibilidade de seleção dos elementos arquiteturais a serem monitorados; o usuário foi capaz de estabelecer o tempo aceitável de resposta das

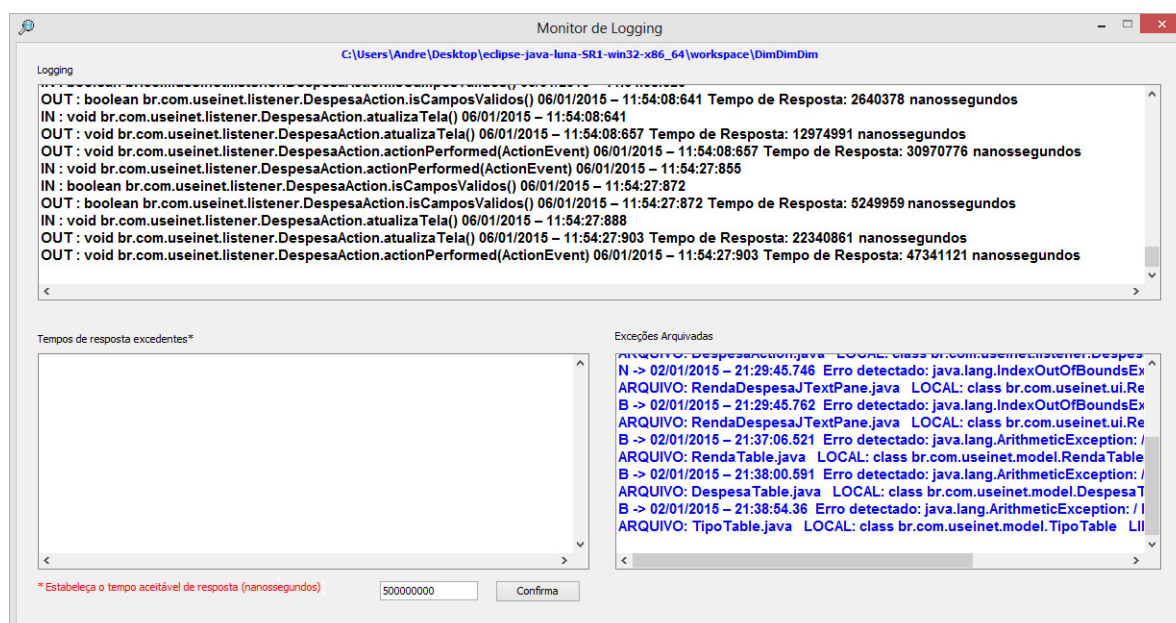


Figura 41 – Inserção de Valores para a Monitoração do DimDimDim



Fonte: Elaborada pelo autor

Figura 42 – Tela Monitor de Logging - Monitoração do DimDimDim



Fonte: Elaborada pelo autor

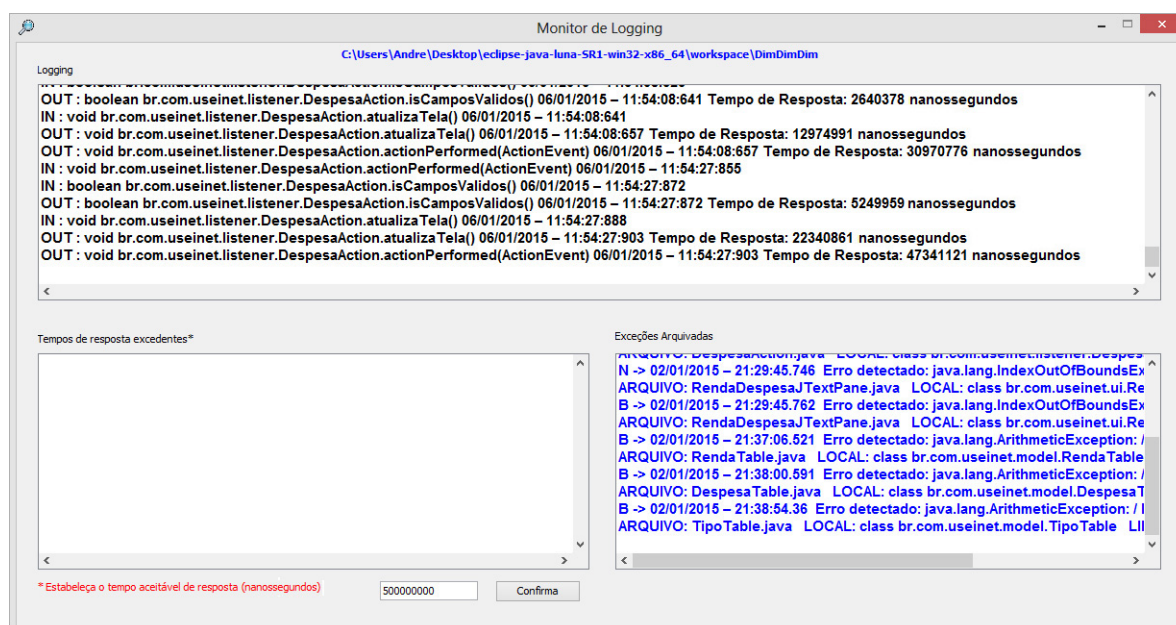
funcionalidades; e efetivou-se a retenção de dados, como a assinatura dos métodos, tempo de acesso, do tempo de conclusão e do tempo de resposta das ações monitoradas, que puderam ser visualizados no Monitor de Logging, sendo também armazenados no arquivo *logAud.txt* no diretório \Logs do projeto DimDimDim.

### 7.1.3.2 Análise sob a Aplicação Mayam

Na aplicação auxiliar Mayam, as classes escolhidas para monitoração foram a *ImportDcmDirDelegate* e a *ApplicationContext* que correlacionam-se às funcionalidades importar, excluir e visualizar exames. Reforça-se que os mencionados exames correspondem aos dois exames PHENIX e VIX, que estão no padrão DICOM, gerados a partir de um aparelho de TC.

Assim, com o aspecto-monitor da eficiência gerado e inserido no projeto Mayam, foram executadas as seguintes ações: carregamento dos conjuntos de imagens PHENIX e VIX para o contexto desta aplicação, visualização dessas imagens e exclusão dos exames PHENIX e VIX. O resultado dessa execução é mostrado na Figura 43.

Figura 43 – Tela Monitor de Logging - Monitoração do Mayam



Fonte: Elaborada pelo autor

Essa figura destaca a interface Monitor de Logging no contexto da aplicação Mayam. Percebe-se, na caixa de texto superior, a disposição do registro das ações destacadas; na caixa de texto inferior-esquerda, o campo para exibição dos métodos que superaram o tempo de resposta de 500000000 ns; e, na caixa de texto inferior-direita, as exceções armazenadas no arquivo *logExc.txt*, presente no projeto Mayam.

Do mesmo modo como ocorreu na análise discutida na subseção que observou o Dim-DimDim, a verificação da aspecto da eficiência também gerou um arquivo de texto - *logAud.txt*, disponível no diretório \Logs do projeto Mayam - contendo toda a monitoração realizada. A Figura 44 expõe parte deste arquivo que, apenas nesta execução, armazenou 5550 linhas, devido ao intenso processamento advindo da importação, visualização e exclusão dos exames PHENIX - composto por 723 imagens - e VIX - composto por 250 imagens.

Figura 44 – Dados Armazenados pelo Aspecto-monitor da Eficiência no Mayam

```

MainScreen.java ApplicationFacade.java logAud.txt
52 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:126 Tempo d ^
53 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:241
54 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:281 Tempo d
55 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:384
56 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:400 Tempo d
57 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:495
58 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:503 Tempo d
59 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:590
60 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:605 Tempo d
61 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:683
62 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:683 Tempo d
63 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:840
64 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:840 Tempo d
65 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:918
66 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:53:933 Tempo d
67 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:58
68 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:58 Tempo de
69 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:168
70 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:168 Tempo d
71 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:308
72 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:324 Tempo d
73 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:418
74 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:418 Tempo d
75 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:486
76 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:502 Tempo d
77 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:564
78 OUT : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:580 Tempo d
79 IN  : String in.raster.mayam.context.ApplicationContext.getAppDirectory() 06/01/2015 - 16:21:54:674

```

Fonte: Elaborada pelo autor

Logo, através da observação do funcionamento do Mayam e do aspecto-monitor da eficiência, e das respostas dadas ao Questionário 3, determina-se que: o monitor foi iniciado sem a ocorrência de erros; foi possível selecionar os elementos arquiteturais alvos da monitoração; o usuário pôde estabelecer o tempo aceitável de resposta das funcionalidades; e a retenção de dados, como a assinatura dos métodos, tempo de acesso, do tempo de conclusão e do tempo de resposta das ações monitoradas foi efetivada, visualizando tais informações no Monitor de Logging que foram armazenadas no arquivo *logAud.txt* no diretório \Logs do projeto DimDimDim.

### 7.1.3.3 Análise sob as Aplicações *BancoOO* e *BancoOA*

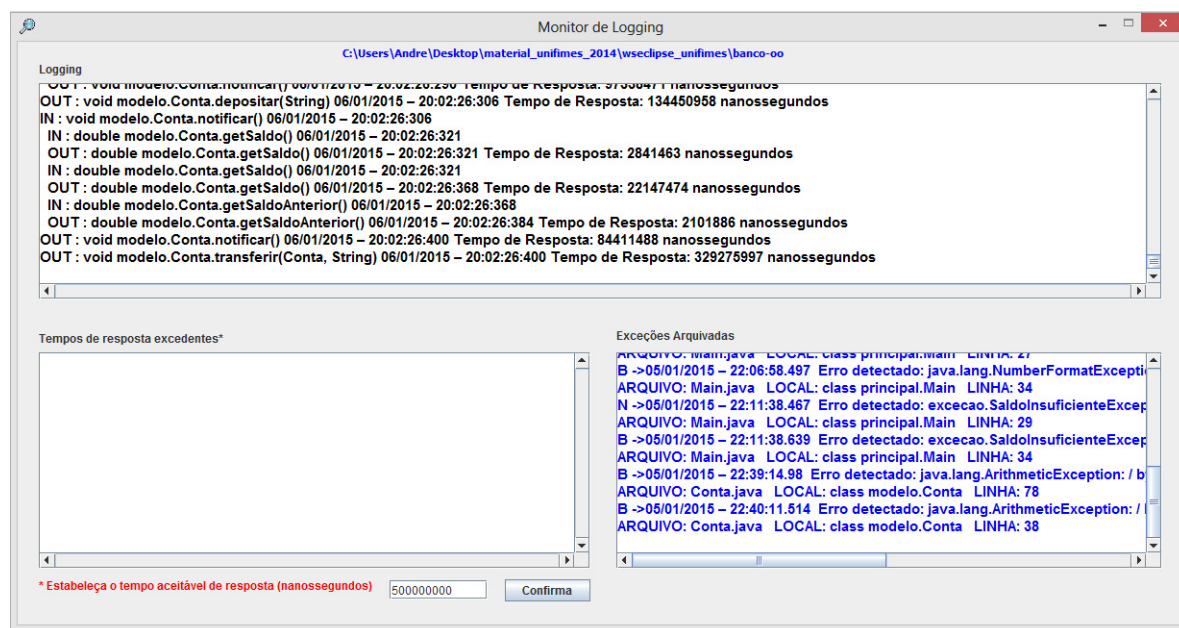
Nas aplicações BancoOO e BancoOA, o elemento arquitetural monitorado corresponde à classe *Conta* que compreende as funcionalidades depositar, sacar e transferir. A princípio, o aspecto utilizado nesta verificação terá seu funcionamento analisado no contexto da aplicação BancoOO e, em seguida, junto da aplicação BancoOA.

A fim de padronizar as duas análises, procedeu-se com a utilização das mesmas ações para ambos os softwares. Desse modo, foi definido um conjunto de atividades a serem executadas durante a monitoração da eficiência das aplicações do projeto BancoUnifimes. As ações compreendem: criação de três clientes (Cliente 1, Cliente 2 e Cliente 3); abertura de uma conta para cada cliente (Conta 1, Conta 2 e Conta 3); três depósitos, um cada conta (800,00; 1.000,00 e 1.300,00); três saques, um em cada conta (50,00; 800,00; 100,00); e duas transferências (100,00 da Conta 1 para a Conta 2 e 37,00 da Conta 2 para a Conta 3).



A realização das atividades citadas ocorreu de maneira satisfatória no BancoOO, ocorrendo a visualização - através da tela Monitor de Logging (ver Figura 45) - e registro da operação dos métodos monitorados - através do armazenamento dos dados no arquivo *logAud.txt*.

Figura 45 – Tela Monitor de Logging - Monitoração do BancoOO



Fonte: Elaborada pelo autor

A mesma sequência de ações foi executada sob a aplicação BancoOA, dotada do seu respectivo aspecto-monitor da eficiência. Notou-se que o comportamento do referido aspecto ocorreu conforme previsto, ou seja, houve a disposição do Monitor de Logging, com os elementos advindos da monitoração, além do armazenamento de todos os dados exibidos no arquivo de texto destinado para este fim. A Figura 46 apresenta um recorte do arquivo *logAud.txt* gerado no diretório \Logs do projeto BancoOA.

Em suma, tomando como base as respostas do Questionário 3 para cada uma das aplicações auxiliares e as constatações oriundas das execuções anteriormente descritas, percebeu-se que o aspecto-monitor da eficiência foi iniciado sem a ocorrência de erros em ambos os softwares; além do mais foi possível selecionar os elementos arquiteturais alvos da monitoração, o usuário pôde estabelecer o tempo aceitável de resposta das funcionalidades e a visualização e armazenamento dos dados da monitoração foi efetivada. Ressalta-se que o aspecto em análise trabalhou corretamente, mesmo nos distintos contextos OO e OA.

## 7.2 Síntese dos Resultados

Diante das execuções realizadas e dos resultados encontrados, nota-se que a solução implementada neste trabalho apresentou um comportamento satisfatório, sendo viável sua

Figura 46 – Dados Armazenados pelo Aspecto-monitor da Eficiência no BancoOA

```

Main.java  logAud.txt
266   OUT : void modelo.Conta.incrementarPontuacao(int) 06/01/2015 - 20:11:46:381 Tempo de Resposta: 1024358 ^
267   OUT : void modelo.Conta.notificar() 06/01/2015 - 20:11:46:381 Tempo de Resposta: 43544955 nanossegundos
268   OUT : void modelo.Conta.atualizarSaldo(double) 06/01/2015 - 20:11:46:396 Tempo de Resposta: 52969248 nanos
269   IN  : String modelo.Conta.toString() 06/01/2015 - 20:11:46:396
270   OUT : String modelo.Conta.toString() 06/01/2015 - 20:11:46:396 Tempo de Resposta: 1333323 nanossegundos
271   IN  : void modelo.Conta.notificar() 06/01/2015 - 20:11:46:396
272   IN  : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:412
273   OUT : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:412 Tempo de Resposta: 1106875 nanossegundos
274   IN  : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:412
275   OUT : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:427 Tempo de Resposta: 2949706 nanossegundos
276   IN  : double modelo.Conta.getSaldoAnterior() 06/01/2015 - 20:11:46:427
277   OUT : double modelo.Conta.getSaldoAnterior() 06/01/2015 - 20:11:46:443 Tempo de Resposta: 1618647 nanosseg
278   IN  : void modelo.Conta.incrementarPontuacao(int) 06/01/2015 - 20:11:46:443
279     IN  : int modelo.Conta.getNumero() 06/01/2015 - 20:11:46:459
280     OUT : int modelo.Conta.getNumero() 06/01/2015 - 20:11:46:459 Tempo de Resposta: 1450170 nanossegundos
281   OUT : void modelo.Conta.incrementarPontuacao(int) 06/01/2015 - 20:11:46:459 Tempo de Resposta: 14982674
282   OUT : void modelo.Conta.notificar() 06/01/2015 - 20:11:46:474 Tempo de Resposta: 65864982 nanossegundos
283   OUT : void modelo.Conta.depositar(String) 06/01/2015 - 20:11:46:474 Tempo de Resposta: 144862562 nanossegund
284   IN  : void modelo.Conta.notificar() 06/01/2015 - 20:11:46:490
285   IN  : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:490
286   OUT : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:506 Tempo de Resposta: 1349627 nanossegundos
287   IN  : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:521
288   OUT : double modelo.Conta.getSaldo() 06/01/2015 - 20:11:46:521 Tempo de Resposta: 1432960 nanossegundos
289   IN  : double modelo.Conta.getSaldoAnterior() 06/01/2015 - 20:11:46:537
290   OUT : double modelo.Conta.getSaldoAnterior() 06/01/2015 - 20:11:46:537 Tempo de Resposta: 1487307 nanosseg
291   OUT : void modelo.Conta.notificar() 06/01/2015 - 20:11:46:552 Tempo de Resposta: 56954271 nanossegundos
292   OUT : void modelo.Conta.transferir(Conta, String) 06/01/2015 - 20:11:46:552 Tempo de Resposta: 394554481 nan
293

```

Fonte: Elaborada pelo autor

utilização no contexto dos softwares analisados. Tanto a geração de aspectos-monitores quando a efetiva monitoração realizada por eles foi conferida a partir da visualização, em tempo de execução, em interfaces específicas - denominadas de monitores - e pela retenção em arquivos de textos que foram salvos em diretórios presentes no projeto de cada aplicação auxiliar.

Por meio da observação do funcionamento da solução, softwares e aspectos-monitores, amparada pela aplicação do método GQM, é possível concluir que as três metas definidas (avaliar a execução da ferramenta quanto à geração dos aspectos-monitores, diante da observação do funcionamento do aspecto-monitor da confiabilidade e do aspecto-monitor da eficiência) foram atingidas. A oferta de questionários auxiliou na elicitación e organização dos pontos relevantes para uma análise adequada. Salienta-se que as medidas MTTF, MTTR e MTBF, presentes na árvore de decisão vista na Seção 4.3, podem ser calculadas mediante a observação dos registros retidos e armazenados pelo *AspectoLogExcecao.aj* no arquivo *logExc.txt*.

É importante destacar que ocorreram algum erros pontuais nas execuções realizadas sob a aplicação Mayam. A princípio, os métodos presentes nas classes não foram exibidos pela ferramenta desenvolvida. Além do mais, durante as atividades de monitoração, ao passo que as telas Monitor de Logging e Monitor de Exceções eram minimizadas, esse software auxiliar tinha seu funcionamento comprometido, ficando extremamente lento. Entretanto, quando as referidas telas eram maximizadas e postas em segundo plano, a aplicação voltava ao seu funcionamento normal. Não se encontrou razões para este

comportamento, contudo o fato de o Mayam ser originário do NetBeans IDE pode ser uma dos motivos para tanto.

Enfim, obteve-se êxito na avaliação da solução desenvolvida, com análises realizadas em aplicações de pequeno porte (DimDimDim), médio porte (Mayam), orientada a objetos (BancoOO) e orientada a aspectos (BancoOA), conforme pôde ser visto neste capítulo. Enfatiza-se que a execução conjunta dos dois aspectos-monitores, diferente do que se havia pensado, aconteceu sem a ocorrência de erros ou mesmo danos no funcionamento um do outro e ainda não houve comportamentos discrepantes entre a monitoração nos diferentes tipos de aplicações utilizadas.

## 8 CONSIDERAÇÕES FINAIS

Visando realizar um estudo dos atributos de qualidade definidos na norma ISO/IEC 9126, bem como das questões relativas as suas monitorações, esta dissertação apresentou um exame detalhando dos mencionados atributos, a criação de árvores de decisão ou diagramas de influência que relacionam os requisitos não-funcionais às variabilidades de monitoração e ainda o desenvolvimento de uma ferramenta para a aferição dos atributos confiabilidade e eficiência.

O estudo inicial realizado focou-se na observação da arquitetura das aplicações. Ele buscou definir, para cada atributo de qualidade (funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade), qual ou quais elementos arquiteturais deveriam ser monitorados, qual medição deveria ser realizada e como a monitoração seria efetivada. O resultado desta análise foi a disposição de seis árvores de decisão contendo indicações, de uma forma simplificada, para a mensuração de qualidade aos produtos de software.

A ferramenta desenvolvida tomou como base os diagramas de influência previamente desenvolvidos e voltou-se para a implementação de uma forma de aferição automatizada de alguns atributos da norma ISO/IEC 9126, antecipadamente definidos, de sistemas codificados no ambiente de desenvolvimento Eclipse IDE na linguagem de programação Java. Através da disposição dos aspectos-base Registro de Exceções e Logging de Auditoria, a solução oferece a capacidade de selecionar elementos arquiteturais para serem monitorados por meio dos aspectos-monitores da confiabilidade (AspectoLogTempo) e da eficiência (AspectoLogExcecao).

Todo o processo de avaliação considerou o paradigma *Goal/Question/Metric* (GQM) que possibilitou o estabelecimento de métricas úteis e relevantes direcionadas à qualidade e interpretação das verificações realizadas e dos dados coletados. Com a utilização do destacado paradigma, foram criados questionários que proveram uma forma simplificada para a coleta das medidas advindas da execução do método GQM sob as aplicações em análise. Salienta-se que a aplicação desse método - mesmo sendo realizada pelo desenvolvedor da solução, resultando em um risco à validade do experimento - foi realizada de maneira satisfatória, ao avaliar o funcionamento da solução geradora de aspectos-monitores e o funcionamento desses quando inseridos no contexto dos softwares tidas como auxiliares neste processo, isto é, o DimDimDim, o Mayam, o BancoOO e o BancoOA.

Inferese que este trabalho não é um estudo finalizado, muito pelo contrário, ele abre precedentes para a realização de uma série outros estudos orientados à mensuração da qualidade de produtos de softwares, advinda da aferição de atributos de qualidade que, em muitos casos, são largamente negligenciados durante o ciclo de desenvolvimento de determinado sistema. Ademais, há a possibilidade de refinamento e complemento da pes-

quisa e dos artefatos aqui produzidos, realizando-se: a aplicação do processo de avaliação em um número maior de softwares, para tornar a solução mais confiável; planos experimentais em Quadrados Latinos, medindo a viabilidade da solução em diversos cenários e com diferentes usuários; melhoria da monitoração dos aspectos-monitores existentes e introdução de outros para os demais atributos da norma ISO/IEC 9126; extensão para aplicações de outros ambientes integrados de desenvolvimento; e extensão para outras linguagens de programação.

Assim, diante do que foi apresentado e discutido nesta dissertação, nota-se um importante ganho para o campo da Engenharia de Software, especificamente à subárea associada às técnicas que objetivam a monitoração da qualidade de produtos de software, sendo apresentadas duas efetivas contribuições relacionadas à aferição de atributos de qualidade. Logo, haverá a concepção de arquiteturas de software mais concisas, com a monitoração ocupando seus módulos específicos; ganho de tempo em relação às tarefas de projetistas e desenvolvedores, resultando em um sistema mais bem estruturado e com cada funcionalidade em seu devido módulo; atenuação dos impactos na operabilidade do sistema, por haver um controle das funções adicionais inseridas; e ainda a monitoração automatizada, a princípio, dos atributos de qualidade confiabilidade e eficiência.



## REFERÊNCIAS

- ARTAIAM, N.; SENIVONGSE, T. Enhancing service-side qos monitoring for web services. In: *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPDP '08. Ninth ACIS International Conference on*. [S.l.: s.n.], 2008. p. 765–770.
- BERG, K.; CONEJERO, J.; CHITCHYAN, R. *AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation*. Enschede: AOSD-Europe, 2005. 90 p. AOSD-Europe-UT-01.
- BOMBONATTI, D. L. G. *PARNAFOA: um processo de análise de requisitos não-funcionais orientado a aspectos*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2010.
- BRAUDE, E. J. (Ed.). *Projeto De Software: da programação à arquitetura: uma abordagem baseada em java*. Porto Alegre, RS: Bookman, 2005.
- BRAZ, F. *Logging de Aplicação*. 2003. Disponível em: <[http://smashingreader.com/Logging\\_de\\_Aplicação\\_i1449628](http://smashingreader.com/Logging_de_Aplicação_i1449628)>. Acesso em: mar. 2014.
- BREWSTER, R. Reliability terms and definitions based on the conceptual relationship between reliability and quality. *Microelectronics Reliability*, Elsevier, v. 11, n. 5, p. 435–461, 1972.
- BRUSAMOLIN, V. Manutenibilidade de software. *Revista Digital Online - Instituto Científico de Ensino Superior e Pesquisa*, janeiro 2004. Disponível em: <[http://www.revdigonline.com/artigos\\_download/art\\_10.pdf](http://www.revdigonline.com/artigos_download/art_10.pdf)>.
- BRUZAROSCO, D. C. *Arquitetura de Software*. 2011. Disponível em: <[http://infouem.sots.com.br/wp-content/uploads/2011/03/Arquitetura\\_de\\_Software-apresenta%C3%A7%C3%A3o.pdf](http://infouem.sots.com.br/wp-content/uploads/2011/03/Arquitetura_de_Software-apresenta%C3%A7%C3%A3o.pdf)>. Acesso em: mar. 2014.
- CAMPOS, C. *Tipos de Função*. 2011. Disponível em: <<http://carloscamposinfo.com/mundoapf/?p=197>>. Acesso em: nov. 2014.
- CASAMAYOR, A.; GODOY, D.; CAMPO, M. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology*, v. 52, n. 4, p. 436 – 445, 2010. ISSN 0950-5849. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0950584909001918>>.
- CASTOR, F. et al. On the modularization and reuse of exception handling with aspects. *Software: Practice and Experience*, John Wiley & Sons, Ltd., v. 39, n. 17, p. 1377–1417, 2009. ISSN 1097-024X. Disponível em: <<http://dx.doi.org/10.1002/spe.939>>.
- CATECATI, T. et al. Métodos para a avaliação da usabilidade no design de produtos. *Revista DAPesquisa - Revista do Centro de Artes da UDESC*, agosto 2010. Disponível em: <[http://www.ceart.udesc.br/dapesquisa/edicoes\\_antiores/8/files/04DESIGN\\_Fernanda\\_Gom\es\\_Faust.pdf](http://www.ceart.udesc.br/dapesquisa/edicoes_antiores/8/files/04DESIGN_Fernanda_Gom\es_Faust.pdf)>.

- CHUNG, L.; LEITE, J. C. P. On non-functional requirements in software engineering. In: BORGIDA, A. T. et al. (Ed.). *Conceptual Modeling: Foundations and Applications*. Berlin, Heidelberg: Springer-Verlag, 2009. p. 363–379. ISBN 978-3-642-02462-7. Disponível em: <[http://dx.doi.org/10.1007/978-3-642-02463-4\\_19](http://dx.doi.org/10.1007/978-3-642-02463-4_19)>.
- CHUNG, L.; NIXON, B. A. Dealing with non-functional requirements: Three experimental studies of a process-oriented approach. In: *Proceedings of the 17th International Conference on Software Engineering*. New York, NY, USA: ACM, 1995. (ICSE '95), p. 25–37. ISBN 0-89791-708-1. Disponível em: <<http://doi.acm.org/10.1145/225014.225017>>.
- COLYER, A. et al. *Eclipse Aspectj: Aspect-oriented Programming with Aspectj and the Eclipse Aspectj Development Tools*. First. [S.l.]: Addison-Wesley Professional, 2004. ISBN 0321245873.
- CUNHA, K. T. *Software para cálculo da complexidade ciclomática em código-fonte PL/SQL*. Monografia (Graduação) — Universidade Regional de Blumenau, 2006.
- CYBIS, W. de A. *Engenharia de usabilidade: uma abordagem ergonômica*. 2003. Disponível em: <[www.inf.ufsc.br/~cybis/Univag/Apostila\\_v5.1.pdf](http://www.inf.ufsc.br/~cybis/Univag/Apostila_v5.1.pdf)>. Acesso em: nov. 2014.
- CYSNEIROS, L.; LEITE, J. Integrating non-functional requirements into data modeling. In: *Requirements Engineering, 1999. Proceedings. IEEE International Symposium on*. [S.l.: s.n.], 1999. p. 162–171.
- CYSNEIROS, L. M. *Requisitos Não-Funcionais: da Elicitação ao Modelo Conceitual*. Tese (Doutorado) — Pontifícia Universidade Católica RJ, 2001.
- DCM4CHE. *Mayam*. 2014. Disponível em: <<http://sourceforge.net/projects/dcm4che/files/Mayam/>>. Acesso em: mar. 2014.
- DEITEL, H. M.; DEITEL, P. D. *Java: como programar*. [S.l.]: PRENTICE HALL BRASIL, 2010.
- DIAS, C. *Usabilidade na web: criando portais mais acessíveis*. [S.l.]: Alta Books, 2003.
- DIJKSTRA, E. W. *A Discipline of Programming*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997. ISBN 013215871X.
- FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2nd. ed. Boston, MA, USA: PWS Publishing Co., 1998. ISBN 0534954251.
- FERREIRA, R. de A. M. *Modularização de tratamento de exceções usando programação orientada a aspectos*. Dissertação (Mestrado) — Universidade Estadual de Campinas - Instituto de Computação, 2006.
- FILHO, A. M. da S. Análise da arquitetura de software. *Engenharia de Software Magazine*, 2009. Disponível em: <<http://www.devmedia.com.br/artigo-engenharia-de-software-14-analise-da-arquitetura-de-software/13254>>.
- FILHO, M. J. A. G. *Um Processo de Avaliação da Portabilidade de Unidades de Software*. Monografia (Graduação) — Universidade Federal de Pernambuco, 2005.

- FILHO, T. A. R. *Método para análise dos riscos operacionais associados a falhas epidêmicas de novos produtos eletrônicos: Uma proposta utilizando redes bayesianas*. Dissertação (Mestrado) — Universidade do Vale do Rio dos Sinos, 2011.
- FILIPPETTO, A. S.; CALLEGARI, D. A. Programação orientada a aspectos: Um estudo de caso em uma multinacional. *Anais Conferencia Latinoamericana de Informatica. 32nd CLEI Latin America Conference on Informatics*, Santiago, Chile, 2006.
- FINKELSTEIN, A.; DOWELL, J. A comedy of errors: The london ambulance service case study. In: *Proceedings of the 8th International Workshop on Software Specification and Design*. Washington, DC, USA: IEEE Computer Society, 1996. (IWSSD '96), p. 2-. ISBN 0-8186-7361-3. Disponível em: <<http://dl.acm.org/citation.cfm?id=857204.858287>>.
- GARCIA, R. *O que é Programação Orientada a Aspectos?* 2005. Disponível em: <<http://www.javaframework.org/portal/2010/04/14/o-que-programao-orientada-a-aspectos/>>. Acesso em: mar. 2014.
- GARCIA, V. C. et al. Em direção a uma abordagem para separação de interesses por meio de mineração de aspectos e refactoring. *30th Conferência Latino-Americana de Informática (CLEI 2004)*, 2004.
- GARLAN, D.; PERRY, D. E. “software architecture: Practice, pitfalls, and potential” panel introduction. *16th International Conference on Software Engineering*, maio 1994.
- GODSE, M.; BELLUR, U.; SONAR, R. Automating qos based service selection. In: *Web Services (ICWS), 2010 IEEE International Conference on*. [S.l.: s.n.], 2010. p. 534–541.
- GONÇALVES, M. K. *Usabilidade de software: estudo de recomendações básicas para verificação do nível de conhecimento dos alunos dos cursos de Design Gráfico e sistema de informação da UNESP/Bauru*. Dissertação (Mestrado) — Universidade Estadual Paulista, 2008.
- HAITENG, Z.; ZHIQING, S.; HONG, Z. Runtime monitoring web services implemented in bpel. In: *Uncertainty Reasoning and Knowledge Engineering (URKE), 2011 International Conference on*. [S.l.: s.n.], 2011. v. 1, p. 228–231.
- HNATEK, E. *Practical Reliability Of Electronic Equipment And Products*. [S.l.]: Marcel Dekker, 2003.
- IEEE. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, p. 1–84, Dec 1990.
- ILLA, X. B.; FRANCH, X.; PASTOR, J. A. Formalising erp selection criteria. In: *Proceedings of the 10th International Workshop on Software Specification and Design*. Washington, DC, USA: IEEE Computer Society, 2000. (IWSSD '00), p. 115–123. ISBN 0-7695-0884-7. Disponível em: <<http://dl.acm.org/citation.cfm?id=857171.857217>>.
- ISO. International organization for standarization. *Information Technology - Software life cycle processes*, 1995.
- ISO. International organization for standarization. *Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability*, 1998.

ISO. International organization for standarization. *ISO Standard 9126: Software Engineering - Product Quality, parts 1, 2 and 3, Geneve*, 2001. 2001 (part 1), 2003 (parts 2 and 3).

JÚNIOR, P.; AFONSO, P. *Programação Orientada a Aspectos: uma Visão Prática*. 2014. IV Semana de Sistemas de Informação. Disponível em: <<http://paulojunior.jatai.ufg.br/p/5098-apresentacoes>>. Acesso em: nov. 2014.

JR, H. E. *Engenharia de Software na Prática*. [S.l.]: Novatec, 2010. ISBN 9788575222171.

KICZALES, G. et al. Aspect-oriented programming. In: SIT, M. A.; MATSUOKA, S. (Ed.). *ECOOP'97 - Object-Oriented Programming*. Springer Berlin Heidelberg, 1997, (Lecture Notes in Computer Science, v. 1241). p. 220–242. ISBN 978-3-540-63089-0. Disponível em: <<http://dx.doi.org/10.1007/BFb0053381>>.

KIM, K.; KIM, R. Y. A case study of quality improvement for water resource management system based on iso/iec 9126. *International Journal of Software Engineering & Its Applications*, v. 8, n. 3, 2014.

KOSCIANSKI, A.; SOFTWARE, M. dos Santos Soares Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de. *Qualidade de Software*. [S.l.]: Novatec, 2007. ISBN 9788575221129.

LINDSTROM, D. R. Five ways to destroy a development project. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 10, n. 5, p. 55–58, set. 1993. ISSN 0740-7459. Disponível em: <<http://dx.doi.org/10.1109/52.232400>>.

LIPPERT, M.; LOPES, C. V. A study on exception detection and handling using aspect-oriented programming. In: *Proceedings of the 22Nd International Conference on Software Engineering*. New York, NY, USA: ACM, 2000. (ICSE '00), p. 418–427. ISBN 1-58113-206-9. Disponível em: <<http://doi.acm.org/10.1145/337180.337229>>.

LYU, M. R. *Handbook of Software Reliability Engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996. ISBN 0-07-039400-8.

MEETEI, M.; GOEL, A.; WASAN, S. Observability using aspect-oriented programming for oo software testing. *International Journal of System Assurance Engineering and Management*, Springer-Verlag, v. 2, n. 2, p. 85–96, 2011. ISSN 0975-6809. Disponível em: <<http://dx.doi.org/10.1007/s13198-011-0066-5>>.

MERCER, C. D. M. *DimDimDim*. 2007. Disponível em: <[www.useinet.com.br/dimdimdim/](http://www.useinet.com.br/dimdimdim/)>. Acesso em: mar. 2014.

MICHLMAYR, A. et al. Comprehensive qos monitoring of web services and event-based sla violation detection. In: *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*. New York, NY, USA: ACM, 2009. (MWSOC '09), p. 1–6. ISBN 978-1-60558-848-3. Disponível em: <<http://doi.acm.org/10.1145/1657755.1657756>>.

MONTEIRO, E. da S.; PIVETA, E. K. Programação orientada a aspectos em aspectj. *Anais Encoinfo - Encontro de Estudantes de Informática*, outubro 2003. Disponível em: <<http://arquivo.ulbra-to.br/ensino/43020/artigos/anais2003/anais/aspectj-encoinfo2003.pdf>>.

- MOONEY, J. D. *Software Portability Home Page*. 2011. Disponível em: <<http://www.cs.wvu.edu/~jdm/research/portability/home.html>>. Acesso em: nov. 2014.
- MORAES, A. de. *Design e avaliação de Interface: Ergodesign e Interação Humano-Computador*. Rio de Janeiro: iUsEr, 2002.
- MOSER, O.; ROSENBERG, F.; DUSTDAR, S. Non-intrusive monitoring and service adaptation for ws-bpel. In: *Proceedings of the 17th International Conference on World Wide Web*. New York, NY, USA: ACM, 2008. (WWW '08), p. 815–824. ISBN 978-1-60558-085-2. Disponível em: <<http://doi.acm.org/10.1145/1367497.1367607>>.
- MULLER, C. et al. Salmonada: A platform for monitoring and explaining violations of ws-agreement-compliant documents. In: *Principles of Engineering Service Oriented Systems (PESOS), 2012 ICSE Workshop on*. [S.l.: s.n.], 2012. p. 43–49. ISSN 2156-7921.
- MURTHY, D.; RAUSAND, M.; ØSTERÅS, T. *Product Reliability: Specification and Performance*. [S.l.]: Springer, 2008. (Springer Series in Reliability Engineering). ISBN 9781848002715.
- NIELSEN, J. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 0125184050.
- OSSHHER, H.; TARR, P. Using subject-oriented programming to overcome common problems in object-oriented software development/evolution. In: *Software Engineering, 1999. Proceedings of the 1999 International Conference on*. [S.l.: s.n.], 1999. p. 687–688. ISSN 0270-5257.
- PENG, Y.; WANG, G.; WANG, H. User preferences based software defect detection algorithms selection using {MCDM}. *Information Sciences*, v. 191, n. 0, p. 3 – 13, 2012. ISSN 0020-0255. Data Mining for Software Trustworthiness. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0020025510001751>>.
- PIGOSKI, T. M. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. [S.l.]: Wiley Computer Publishing, 1996.
- POLOZOFF, A. Proactive application monitoring. *IBM Software Group, Software Services for WebSphere*, Chicago, Illinois, USA, 2003.
- PRESSMAN, R. S. *Engenharia de Software*. 7nd. ed. [S.l.]: McGraw Hill Brasil, 2011. ISBN 9788580550443.
- RASHID, A. et al. Early aspects: a model for aspect-oriented requirements engineering. In: *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*. [S.l.: s.n.], 2002. p. 199–202. ISSN 1090-705X.
- RESENDE, A. M. P. de; SILVA, C. C. da. *Programação Orientada a Aspectos em Java*. [S.l.]: Brasport, 2005. ISBN 9788574522128.
- ROCHA, A. R. C. da; SOUZA, G. dos S.; BARCELLOS, M. P. *Medição de Software e Controle Estatístico de Processos*. [S.l.]: Ministério da Ciência, Tecnologia e Inovação - Secretaria de Política de Informática, 2012.

RODRIGUES, V. A. *Paradigma orientado a aspectos, estudo de caso: sistema de auditoria para aplicações java*. Monografia (Graduação) — Universidade Federal de Santa Maria, 2007.

ROYCHOWDHURY, S.; PEDRYCZ, W. A survey of defuzzification strategies. *Int. J. Intell. Syst.*, v. 16, n. 6, p. 679–695, 2001. Disponível em: <<http://dblp.uni-trier.de/db/journals/ijis/ijis16.html\#RoychowdhuryP01>>.

SOLINGEN, R. V.; BERGHOUT, E. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*. [S.l.]: McGraw-Hill London, 1999.

SOMMERVILLE, I. *Software Engineering*. [S.l.]: Pearson/Addison-Wesley, 2011. (International Computer Science Series). ISBN 9780137053469.

SOUZA, F. et al. Dynamic event-based monitoring in a soa environment. In: *Proceedings of the 2011th Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part II*. Berlin, Heidelberg: Springer-Verlag, 2011. (OTM'11), p. 498–506. ISBN 978-3-642-25105-4. Disponível em: <<http://dl.acm.org/citation.cfm?id=2075764.2075773>>.

SURYN, W.; ABRAN, A.; APRIL, A. Iso/iec square: The second generation of standards for software product quality. In: *7th IASTED International Conference on Software Engineering and Applications*. [S.l.: s.n.], 2003.

TSANG, S. L.; CLARKE, S.; BANIASSAD, E. An evaluation of aspect-oriented programming for java-based real-time systems development. In: *Object-Oriented Real-Time Distributed Computing, 2004. Proceedings. Seventh IEEE International Symposium on*. [S.l.: s.n.], 2004. p. 291–300.

VANDOREN, E. *Complexidade ciclomática: software technology roadmap*. 1997. Disponível em: <[http://www.sei.cmu.edu/str/descriptions/cyclomatic\\_body.html](http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html)>. Acesso em: mar. 2014.

VAZQUEZ, C. E.; SIMÕES, G. S.; ALBERT, R. M. *Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software*. 13. ed. [S.l.]: Érica, 2013.

VIEIRA, E. *Ferramenta de apoio ao processo de avaliação de produto de software*. Monografia (Graduação) — Universidade do Vale do Itajaí - Centro de Ciências Tecnológicas da Terra e do Mar, 2012.

VILLELA, C. V. *Ambiente baseado em componentes para o desenvolvimento de sistemas computacionais microcontrolados distribuídos*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2001.

WANGENHEIM, C. G. von. *Utilização do GQM no Desenvolvimento de Software*. 2000. Disponível em: <<http://www.casi.xpg.com.br/IDSI/Gqm.PDF>>. Acesso em: nov. 2014.

WEBER, T. S. *Um roteiro para exploração dos conceitos básicos de tolerância a falhas*. 2002. Disponível em: <<http://www.inf.ufrgs.br/~taisy/disciplinas/textos/Dependabilidade.pdf>>. Acesso em: nov. 2014.

WETZSTEIN, B. et al. Monitoring and analyzing influential factors of business process performance. In: *Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International*. [S.l.: s.n.], 2009. p. 141–150. ISSN 1541-7719.

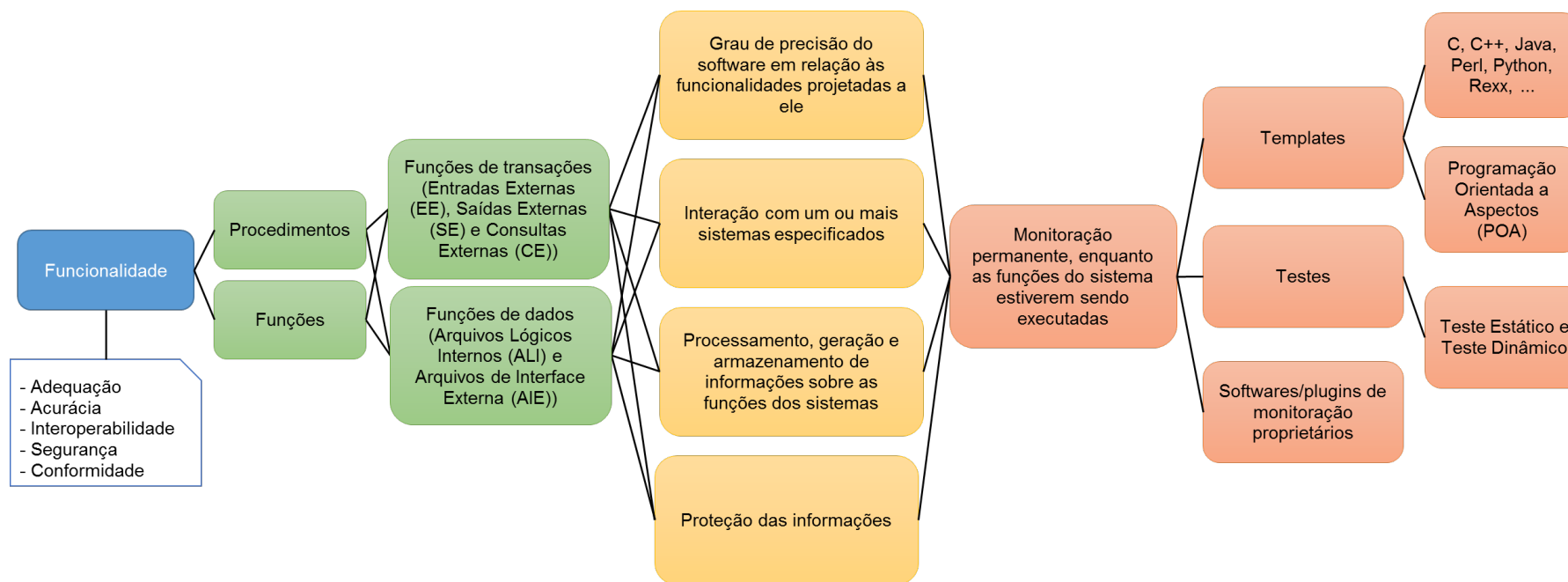
WINCK, D. V.; JUNIOR, V. G. *AspectJ: Programação Orientada a Aspectos com Java*. [S.l.]: Novatec, 2006. ISBN 9788575220870.

## APÊNDICE A – ÁRVORES DE DECISÃO AMPLIADAS

Este apêndice apresenta as árvores de decisão referentes à monitoração/avaliação dos atributos de qualidade especificados na norma ISO/IEC 9126. As árvores dispostas nas Figuras 47, 48, 49, 50, 51 e 52 são apresentadas no Capítulo 4, porém em escala menor, fato que pode dificultar sua análise. Aqui as mesmas imagens podem ser vistas em tamanho ampliado, conforme segue.

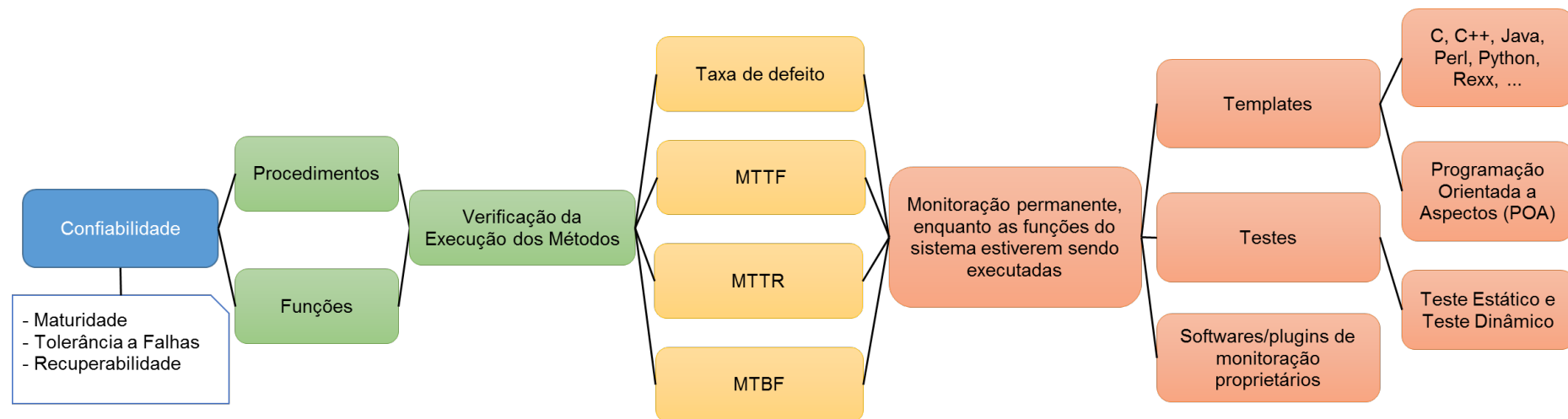


Figura 47 – Árvore de Decisão Ampliada - Atributo Funcionalidade



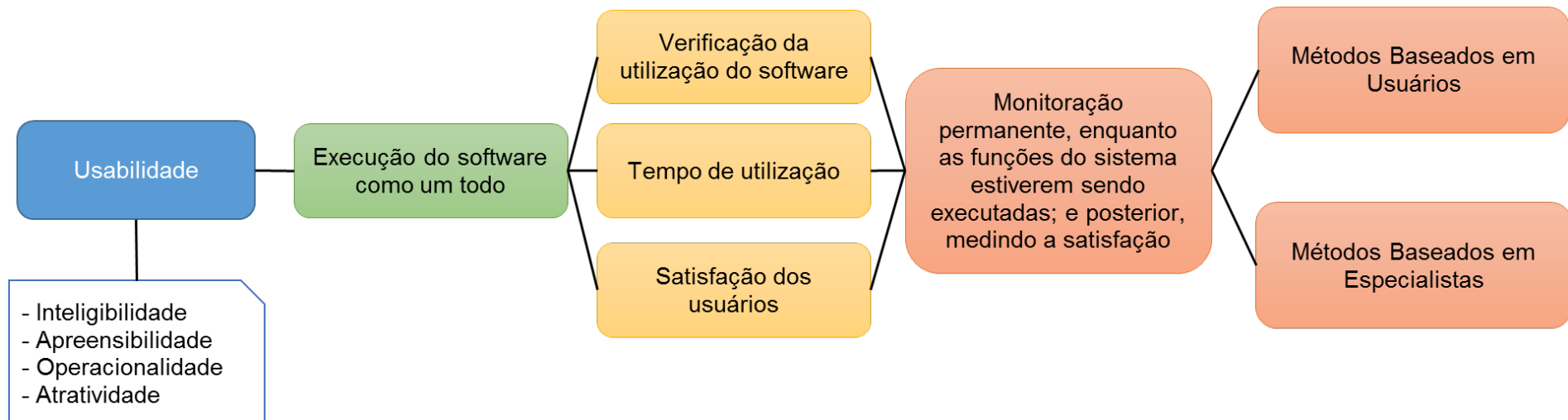
Fonte: Elaborada pelo autor

Figura 48 – Árvore de Decisão Ampliada - Atributo Confiabilidade



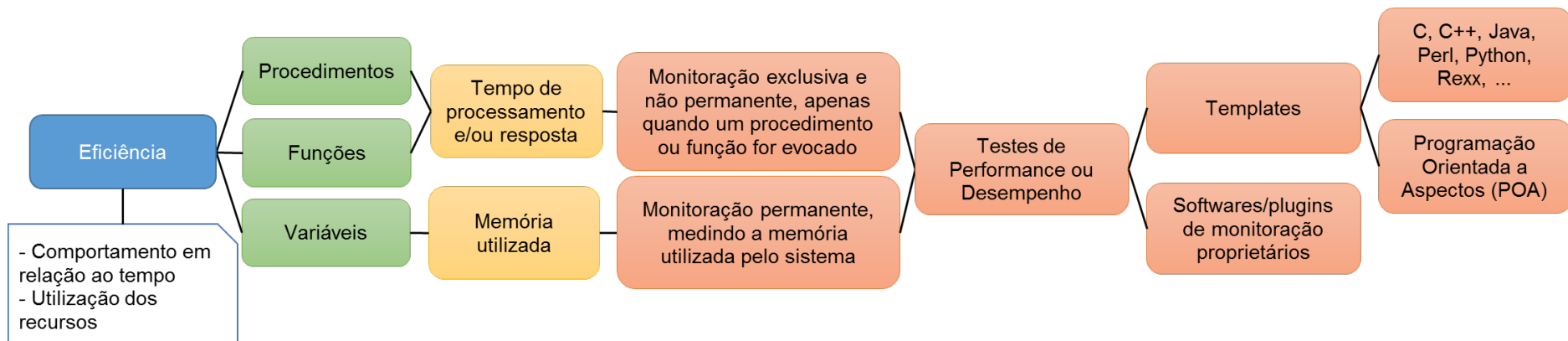
Fonte: Elaborada pelo autor

Figura 49 – Árvore de Decisão Ampliada - Atributo Usabilidade



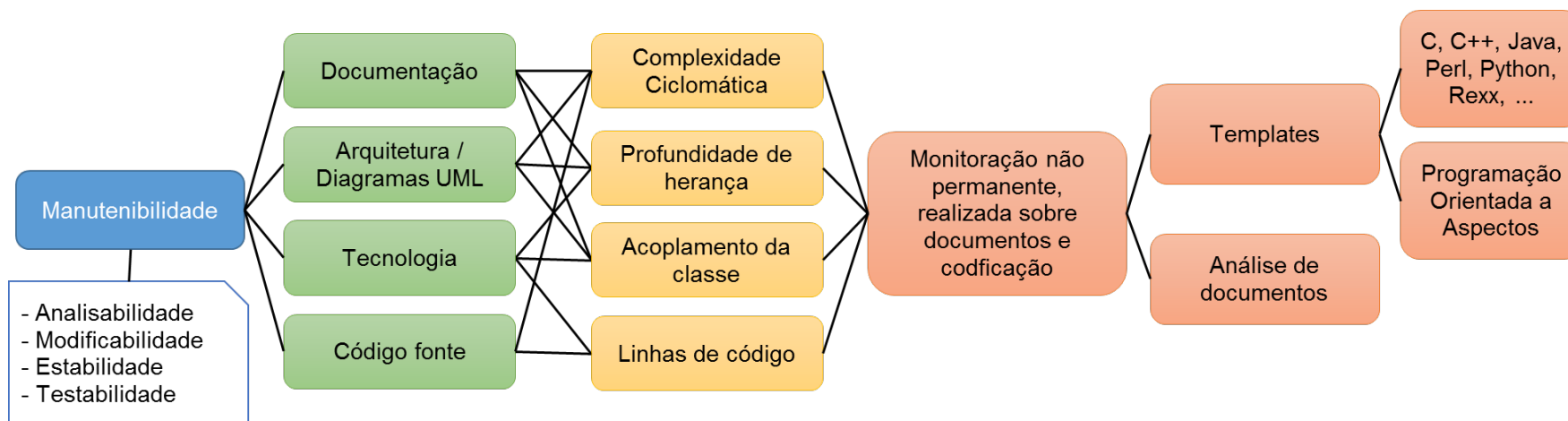
Fonte: Elaborada pelo autor

Figura 50 – Árvore de Decisão Ampliada - Atributo Eficiência



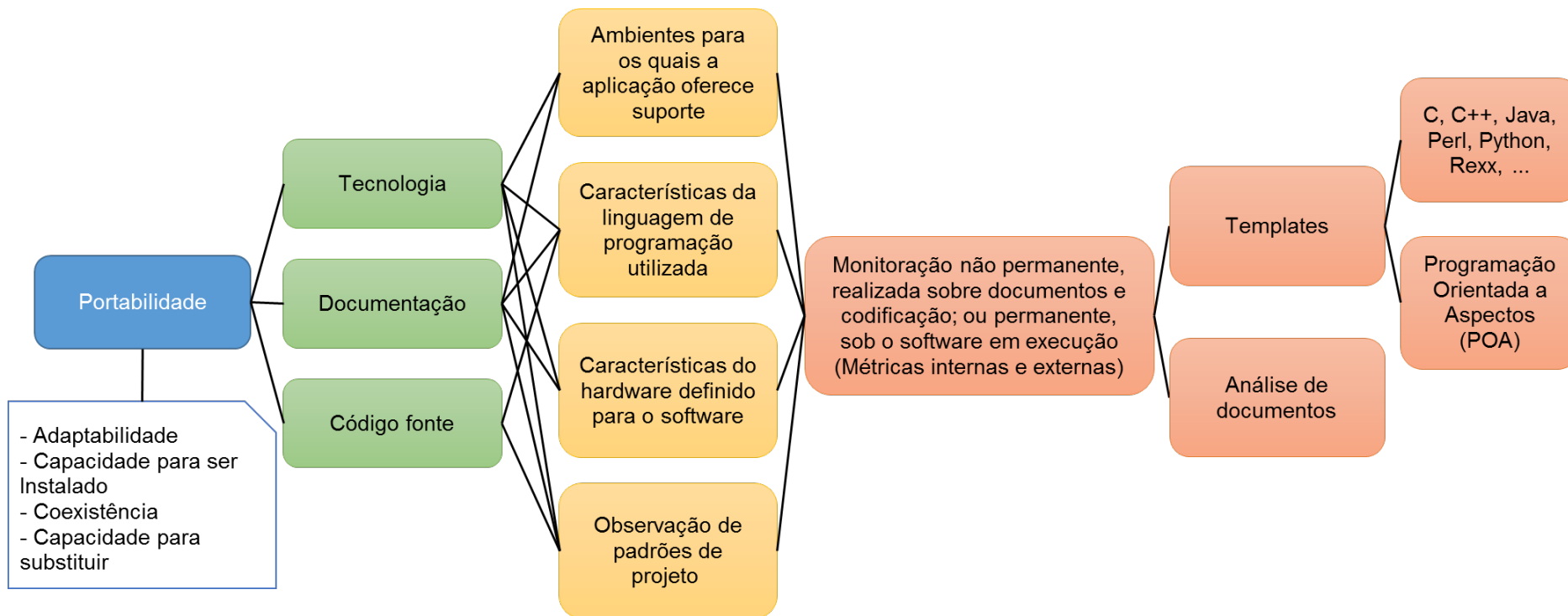
Fonte: Elaborada pelo autor

Figura 51 – Árvore de Decisão Ampliada - Atributo Manutenibilidade



Fonte: Elaborada pelo autor

Figura 52 – Árvore de Decisão Ampliada - Atributo Portabilidade



Fonte: Elaborada pelo autor

## APÊNDICE B – QUESTIONÁRIOS PARA MENSURAÇÃO DAS MEDIDAS

Neste apêndice são apresentados os questionários atinentes à coleta dos dados advindos da aplicação do método GQM. Eles estão categorizados de acordo com a meta que se referem. Assim, o questionário 1 (11) volta-se à meta *avaliar a execução do software quanto à geração dos aspectos-monitores*, o questionário 2 (12) à meta *avaliar o software quanto à observação do funcionamento do aspecto-monitor da confiabilidade* e o questionário 3 (13) à meta *avaliar o software quanto à observação do funcionamento do aspecto-monitor da eficiência*.

**Tabela 11 – Questionário 1**

---

**Questionário 1 - Avaliação da Execução do Software quanto à Geração dos Aspectos-monitores**

**Solução avaliada:**

**Aplicação auxiliar:**

**Programação utilizada na aplicação auxiliar:** ( )OO ( )OA

**Papel:** ( )Desenvolvedor ( )Usuário

**1- A solução pôde ser iniciada, sem a ocorrência de erros evidentes, sob a aplicação auxiliar?**

( )SIM

( )NÃO

**2- Os elementos arquiteturais exibidos no software correspondem aos mesmos presentes na aplicação auxiliar?**

( )SIM

( )NÃO, há um ou mais elementos inexistentes:

( )NÃO, há um ou mais elementos extras:

**3- Ao escolher um elemento para monitoração, esse é inserido na lista de elementos a serem monitorados?**

( )SIM

( )NÃO

**4- Ao remover um elemento da lista de elementos a serem monitorados, esse é retirado da referida lista?**

( )SIM

( )NÃO

Tabela 11 – Questionário 1 (continuação)

---

5- Quando a lista de elementos a serem monitorados estiver vazia, a ação Gerar Aspecto poderá ser executada, possibilitando a geração de aspectos-monitores?

SIM

NÃO

6- Quando a lista de elementos a serem monitorados NÃO estiver vazia e a ação Gerar Aspecto é executada, há a disposição das opções de interesses para monitoração?

SIM

NÃO

7- Há a geração de um arquivo com extensão .aj, presente no diretório \Aspectos do projeto da aplicação auxiliar, quando um dos interesses é selecionado?

SIM

NÃO

8- Ao clicar no botão Escrever Aspecto, há a abertura da interface Editor de Aspectos?

SIM

NÃO

9- Ao salvar um aspecto criado na interface Editor de Aspectos, o mesmo é salvo no local especificado pelo usuário?

SIM

NÃO

Considerações:

---



---



---



---



---

Fonte: Elaborada pelo autor

Tabela 12 – Questionário 2

---

Questionário 2 - Avaliação do software quanto à observação do funcionamento do aspecto-monitor da confiabilidade

Solução avaliada:

Aplicação auxiliar:

Programação utilizada na aplicação auxiliar:  OO  OA

Papel:  Desenvolvedor  Usuário

---

continua na próxima página



Tabela 12 – Questionário 2 (continuação)

---

1- O aspecto-monitor da confiabilidade pôde ser iniciado, sem a ocorrência de erros evidentes, sob a aplicação auxiliar?

SIM

NÃO

2- O usuário é capaz de selecionar funcionalidades específicas para que o aspecto-monitor da confiabilidade atue?

SIM

NÃO

3- Durante a execução da aplicação auxiliar, notou-se a ocorrência exceções de negócio e de exceções decorrentes de erros?

SIM

NÃO

4- A utilização conjunta do aspecto-monitor da confiabilidade e do aspecto-monitor da eficiência dificultou a aferição da confiabilidade?

SIM

NÃO

5- Durante a execução da aplicação auxiliar, as exceções decorrentes de falhas são diferenciadas, pelo aspecto-monitor da confiabilidade, daquelas que são próprias do negócio?

SIM

NÃO

6- Em caso afirmativo à questão anterior, qual a proporção de exceções de negócio (esperadas) e exceções decorrentes de bug (inesperadas)?

---

7- Após a execução da aplicação auxiliar, dotada do aspecto-monitor da confiabilidade, ocorreu a geração de um arquivo de registro com as exceções observadas?

SIM, com todas as exceções observadas

SIM, com algumas das exceções observadas:

NÃO

8- Durante a execução da aplicação auxiliar, a observação das exceções pôde ser realizada em uma interface específica?

SIM

NÃO

Considerações:

---

---

---

Tabela 12 – Questionário 2 (continuação)

---

---

---

Fonte: Elaborada pelo autor

Tabela 13 – Questionário 3

---

**Questionário 3 - Avaliação do software quanto à observação do funcionamento do aspecto-monitor da eficiência**

**Solução avaliada:**

**Aplicação auxiliar:**

**Programação utilizada na aplicação auxiliar:** ( )OO ( )OA

**Papel:** ( )Desenvolvedor ( )Usuário

**1- O aspecto-monitor da eficiência pôde ser iniciado, sem a ocorrência de erros evidentes, sob a aplicação auxiliar?**

( )SIM

( )NÃO

**2- O usuário é capaz de selecionar funcionalidades específicas para que o aspecto-monitor da eficiência atue?**

( )SIM

( )NÃO

**3- Durante a execução da aplicação auxiliar, o usuário é capaz de estabelecer o tempo aceitável de resposta das funcionalidades?**

( )SIM

( )NÃO

**4- O aspecto-monitor da confiabilidade retém o tempo de acesso, o do tempo de conclusão e do tempo de resposta das funcionalidades monitoradas?**

( )SIM

( )NÃO

**5- A utilização conjunta do aspecto-monitor da confiabilidade e do aspecto-monitor da eficiência dificultou a aferição da confiabilidade?**

( )SIM

( )NÃO

**6- Qual o critério utilizado para estabelecer que um tempo de resposta é adequado ou excede o aceitável?**

---

continua na próxima página

Tabela 13 – Questionário 3 (continuação)

---

**7- Após a execução da aplicação auxiliar, dotada do aspecto-monitor da eficiência, ocorreu a geração de um arquivo contendo o logging das funcionalidades monitoradas?**

- SIM, totalmente  
 SIM, parcialmente:  
 NÃO

**8- Durante a execução da aplicação auxiliar, a observação do logging pôde ser realizada em uma interface específica?**

- SIM  
 NÃO

**Considerações:**

---

---

---

---

---

Fonte: Elaborada pelo autor