



Programa de Pós-Graduação em  
**Computação Aplicada**  
Mestrado Acadêmico

Maicon dos Santos

**TechREF: Uma Técnica de Engenharia Reversa  
Orientada a *Features***

São Leopoldo

2018



**Maicon dos Santos**

**TechREF: Uma Técnica de Engenharia Reversa  
Orientada a *Features***

**São Leopoldo  
2018**

S237t	<p>Santos, Maicon dos.</p> <p>TechREF: uma técnica de engenharia reversa orientada a features / por Maicon dos Santos. – São Leopoldo, 2018.</p> <p>106 f. : il. (algumas color.) ; 30 cm.</p> <p>Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2018.</p> <p>Orientação: Prof. Dr. Kleinner Silva Farias de Oliveira, Ciências Exatas e Tecnológicas.</p> <p>1.Engenharia de software. 2.Engenharia reversa. 3.Linhas de produtos de software. I.Oliveira, Kleinner Silva Farias de. II.Título.</p> <p>CDU 004.41</p>
-------	---

Catálogo na publicação:  
Bibliotecária Carla Maria Goulart de Moraes – CRB 10/1252

Maicon dos Santos

## **TechREF: Uma Técnica de Engenharia Reversa Orientada a *Features***

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre, pelo Programa de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos – UNISINOS

Aprovado em 18 de janeiro de 2018

### BANCA EXAMINADORA

---

Prof. Dr. Kleinner Silva Farias de Oliveira – PPGCA/UNISINOS

---

Prof. Dr. Jorge Luis Victória Barbosa – PPGCA/UNISINOS

---

Prof. Dr. Bruno Carreiro da Silva – California Polytechnic State University

Orientador: Prof. Dr. Kleinner Silva Farias de Oliveira

Visto e permitida a impressão  
São Leopoldo,

Prof. Dr. Rodrigo da Rosa Righi  
Coordenador PPG em Computação Aplicada



*Aos meus pais Jucelino Souza dos Santos e Maria Elisabete dos Santos,  
por toda educação concedida, pelo grande amor oferecido todos os  
dias, por todo o grande apoio e incentivo ao desenvolvimento deste  
trabalho.*





## AGRADECIMENTOS

É com muita alegria e satisfação que agradeço por ajudarem na realização deste grande sonho.

Primeiramente, a DEUS por me abençoar sempre, por estar ao meu lado nos momentos difíceis e por não me deixar desistir.

Aos meus pais Jucelino Souza dos Santos e Maria Elisabete dos Santos, pela educação concedida, amor, carinho, compreensão, incentivo e apoio durante todos os momentos em minha vida.

A minha irmã Michele dos Santos, por ser muito importante na minha vida, por sempre acreditar em mim e sempre apoiar em todos momentos quando mais necessitava.

A Vitória Lambertini, por estar ao meu lado sempre que mais necessitava, por me apoiar e me incentivar a nunca desistir.

Ao professor Dr. Kleinner Farias, pela orientação, paciência, incentivo e ensinamentos durante todo o desenvolvimento deste trabalho.

Aos professores do Programa de Pós-Graduação em Computação Aplicada, pela oportunidade, contribuição, ensinamento e pela confiança.



## RESUMO

Engenharia reversa de código desempenha um papel fundamental em várias atividades de Engenharia de Software, tais como geração de modelos a partir de código legado e recuperação de funcionalidades (ou *features*) de sistemas. No contexto de Linha de Produto de Software (LPS), por exemplo, um produto de software é formado por um conjunto de *features* que são constantemente alteradas para acomodar mudanças de regras de negócio. Consequentemente, o modelo (por exemplo, diagrama de classes da UML) que representa toda LPS precisa ser modificado para refletir as atualizações realizadas. Neste contexto, várias ferramentas têm sido propostas nas últimas décadas, por exemplo, Astah e ArgoUML. Porém, as ferramentas (e suas técnicas) não dão suporte à engenharia reversa orientada a *features*, são imprecisas no que se refere à completude dos diagramas gerados, bem como exige um alto esforço para atualização dos modelos pois são manuais ou semiautomáticas. Para mitigar esta problemática, este trabalho propõe a TechREF, uma técnica de engenharia reversa orientada a *features*. De forma automática, a TechREF captura o fluxo de execução do código associado a uma *feature*, identifica as informações estruturais e comportamentais do código, e gera diagramas de classes UML, bem como persiste tais diagramas. A TechREF foi avaliada através de um estudo de caso tendo cenários reais de engenharia reversa. Esta avaliação buscou verificar o esforço e a corretude das atividades que serão realizadas no experimento com o uso dos modelos orientados a *features*.

**Palavras-Chave:** Engenharia reversa orientada a *features*. UML. Modelagem de sistemas. Estudos empíricos. Programação orientada a aspectos.



## ABSTRACT

Reverse code engineering plays a key role in various Software Engineering activities, such as model generation from legacy code and retrieval of system features. In the context of Software Product Line (LPS), for example, a software product is composed of a set features that are constantly changed to accommodate changes in business rules. Consequently, the model (for example, UML class diagram) that represents the entire LPS needs to be modified to reflect the updates made. In this context, several tools have been proposed in the last decades, for example, Astah and ArgoUML. However, the tools (and their techniques) do not support feature-oriented reverse engineering, are imprecise in terms of the completeness of the generated diagrams, as well as requiring a high effort to update the models because they are manual or semiautomatic. To mitigate this problem, this paper proposes TechREF, a reverse engineering technique oriented to features. Automatically, TechREF captures the execution flow of code associated with a feature, identifies the structural and behavioral information of the code, and generates diagrams of UML classes, as well as persists such diagrams. TechREF was evaluated through a case study having real reverse engineering scenarios. This evaluation sought to verify the effort and correctness of the activities that will be carried out in the experiment with the use of the models oriented to features.

**Keywords:** Reverse engineering based on features. UML. Systems modeling. Empirical studies. Aspect-oriented programming.



## LISTA DE FIGURAS

Figura 1 – Exemplo de diagrama de classes.....	18
Figura 2 – Exemplo de diagrama de classes orientado a <i>features</i> .....	19
Figura 3 – Representação de engenharia reversa vs código-fonte.....	24
Figura 4 – Exemplo de modelo de <i>features</i> .....	25
Figura 5 – Implementação de elementos do diagrama de classes .....	27
Figura 6 – Representação do diagrama de sequência .....	28
Figura 7 – Desenvolvimento orientado a aspectos .....	29
Figura 8 – Funcionamento da programação orientada a aspectos .....	31
Figura 9 – Visão geral do projeto Eclipse-MDT MoDisco .....	34
Figura 10 – Visão geral do processo proposto .....	36
Figura 11 – Arquitetura da ferramenta PHP2XMI .....	37
Figura 12 – O processo de transformação de FORUML .....	38
Figura 13 – Arquitetura geral do fRex .....	39
Figura 14 – Ferramenta ArgoUML.....	42
Figura 15 – A ferramenta Umbrello UML Modeler .....	43
Figura 16 – A ferramenta Visual Paradigm.....	44
Figura 17 – A Ferramenta StarUML.....	45
Figura 18 – Interface da ferramenta Astah.....	46
Figura 19 – Processo de execução da técnica TechREF .....	51
Figura 20 – Trecho de código de uma <i>feature</i> .....	53
Figura 21 – Trecho de código de <i>features</i> .....	53
Figura 22 – Modelo UML gerado de uma <i>feature</i> .....	55
Figura 23 – Processo de gravação em disco.....	56
Figura 24 – Diagrama de atividades da TechREF .....	58
Figura 25 – Diagrama de componentes da TechREF .....	60
Figura 26 – Trecho do código-fonte dos aspectos implementados para a TechREF.....	61
Figura 27 – Método para adicionar operações da classe .....	62
Figura 28 – Modelo orientado a <i>features</i> .....	63
Figura 29 – Diagrama de classes orientado a <i>features</i> .....	64
Figura 30 – Tempo total de cada tarefa.....	71
Figura 31 – Total de respostas das tarefas.....	74
Figura 32 – Aplicação com o uso da TechREF .....	78
Figura 33 – Tela de confirmação de captura de <i>features</i> .....	80

Figura 34 – Modelo de .xml com as informações das classes capturadas .....	80
Figura 35 – Tela de conclusão da captura da <i>feature</i> .....	81
Figura 36 – Tela para salvar o modelo orientado a <i>features</i> .....	82
Figura 37 – Modelo ideal .....	83
Figura 38 – Modelo gerado pela TechREF .....	83
Figura 39 – Processo de execução do estudo de caso .....	85
Figura 40 – Tempo de estudos em universidades .....	87
Figura 41 – Tempo de experiência em desenvolvimento de software.....	88
Figura 42 – Tempo de experiência em modelagem de software .....	88
Figura 43 – Ferramenta SDMetrics.....	90
Figura 44 – Precisão das métricas por cenários.....	93



## LISTA DE TABELAS

Tabela 1 – Tabela de comparação entre trabalhos relacionados .....	48
Tabela 2 – Atividades do experimento .....	68
Tabela 3 – Descrição das atividades do experimento .....	68
Tabela 4 – Tempos de cada participante por tarefas do modelo tradicional .....	70
Tabela 5 – Tempos de cada participante por tarefas do modelo <i>features</i> .....	70
Tabela 6 – Resultado do teste de Wilcoxon .....	72
Tabela 7 – Corretude das respostas do modelo tradicional .....	72
Tabela 8 – Corretude das repostas do modelo orientado a <i>features</i> .....	73
Tabela 9 – Relação das correções com cenários - teste McNemar .....	74
Tabela 10 – Teste corretude McNemar .....	75
Tabela 11 – Faixa etária dos participantes .....	86
Tabela 12 – Formação acadêmica.....	86
Tabela 13 – Cargo dos participantes ou especialidades de ocupação.....	87
Tabela 14 – Tabela com a relação de métricas.....	89
Tabela 15 – Tabela com os resultados das métricas do cenário 1 .....	91
Tabela 16 – Tabela com os resultados das métricas do cenário 2 .....	92
Tabela 17 – Tabela com os resultados das métricas do cenário 3 .....	92
Tabela 18 – Tabela com os resultados das métricas do cenário 4 .....	92
Tabela 19 – Resultado precisão - Z-test.....	94



## LISTA DE SIGLAS

AST	Árvore de Sintaxe Abstrata
EMF	Eclipse Modeling Framework
GIF	Graphics Interchange Format
GQM	Goal Question Metric
IDE	Ambiente de Desenvolvimento Integrado
JVM	Máquina Virtual Java
LPS	Linha de Produto de Software
MDE	Model-Driven Engineering
MDRE	Model-Driven Requirements Engineering
POA	Programação Orientada a Aspectos
POO	Programação Orientada a Objetos
RE	Engenharia Reversa
UML	Unified Modeling Language
VM	Virtual Machine
XMI	XML Metadata Interchange
XML	eXtensible Markup Language



## SUMÁRIO

<b>1 INTRODUÇÃO</b> .....	<b>17</b>
<b>1.1 Contextualização do problema</b> .....	<b>18</b>
<b>1.2 Questões de pesquisa</b> .....	<b>19</b>
<b>1.3 Objetivos</b> .....	<b>20</b>
<b>1.4 Metodologia</b> .....	<b>20</b>
<b>1.5 Organização do trabalho</b> .....	<b>21</b>
<b>2 FUNDAMENTAÇÃO TEÓRICA</b> .....	<b>23</b>
<b>2.1 Engenharia reversa</b> .....	<b>23</b>
<b>2.2 Modelos de <i>features</i></b> .....	<b>25</b>
<b>2.3 UML</b> .....	<b>26</b>
2.3.1 Diagrama de classes .....	27
2.3.2 Diagrama de sequência .....	28
<b>2.4 Programação orientada a aspectos</b> .....	<b>29</b>
2.4.1 Pontos de junção ( <i>Join points</i> ) .....	29
2.4.2 Pontos de atuação ( <i>Pointcuts</i> ) .....	30
2.4.3 Adendo ( <i>Advice</i> ) .....	30
2.4.4 AspectJ - Orientação a Aspectos com Java .....	31
<b>3 TRABALHOS RELACIONADOS</b> .....	<b>33</b>
<b>3.1 Metodologia para seleção dos trabalhos</b> .....	<b>33</b>
<b>3.2 Análise do estado da arte</b> .....	<b>34</b>
<b>3.3 Análise do estado da prática</b> .....	<b>41</b>
<b>3.4 Comparação dos trabalhos relacionados</b> .....	<b>46</b>
<b>3.5 Oportunidades de pesquisa</b> .....	<b>49</b>
<b>4 TÉCNICA DE ENGENHARIA REVERSA</b> .....	<b>51</b>
<b>4.1 Visão geral da técnica</b> .....	<b>51</b>
4.1.1 Execução e captura de <i>features</i> .....	52
4.1.2 Geração dos modelos orientados a <i>features</i> .....	54
4.1.3 Persistência dos modelos orientados a <i>features</i> .....	56
<b>4.2 Processo de engenharia reversa orientado a <i>features</i></b> .....	<b>57</b>
<b>4.3 Aspectos de implementação</b> .....	<b>59</b>
4.3.1 Arquitetura .....	59
4.3.2 Implementação dos componentes .....	60
<b>5 EXPERIMENTO CONTROLADO</b> .....	<b>65</b>
<b>5.1 Experimento controlado</b> .....	<b>65</b>
5.1.1 Objetivo e questões de pesquisa.....	65
5.1.2 Formulação de hipóteses .....	66
5.1.3 Variáveis de estudo .....	67
5.1.4 Contexto e seleção de participantes .....	67
5.1.5 Detalhes do experimento .....	67
5.1.6 Análise do estudo .....	69
5.1.7 Resultados .....	69
<b>6 AVALIAÇÃO DO ESTUDO DE CASO</b> .....	<b>77</b>
<b>6.1 Utilizando a TechREF</b> .....	<b>77</b>
<b>6.2 Estrutura do estudo de caso</b> .....	<b>84</b>
6.2.2 Cenários do estudo de caso .....	85

<b>6.3 Resultados do estudo de caso .....</b>	<b>91</b>
<b>6.4 Análise dos resultados .....</b>	<b>93</b>
<b>6.5 Avaliação final.....</b>	<b>94</b>
<b>6.6 Ameaças à validade do estudo .....</b>	<b>95</b>
<b>7 CONCLUSÃO.....</b>	<b>97</b>
<b>7.1 Contribuições .....</b>	<b>97</b>
<b>7.2 Limitações e trabalhos futuros .....</b>	<b>98</b>
<b>REFERÊNCIAS.....</b>	<b>99</b>
<b>ANEXO A – QUESTIONÁRIO DE SELEÇÃO DOS PARTICIPANTES.....</b>	<b>105</b>

## 1 INTRODUÇÃO

A engenharia reversa de código desempenha um papel fundamental em várias atividades da Engenharia de Software como, por exemplo, na geração de modelos a partir de código legado, e na recuperação de funcionalidades (ou *features*) de sistemas. Desenvolvedores de software realizam engenharia reversa para manter os modelos da aplicação atualizados. Tais modelos são tipicamente representados através de modelos da UML (*Unified Modeling Language*) (OMG-UML, 2015), tais como os diagramas de classes, sequência e de componentes. De acordo com Chaudron (2011) e Nugroho (2009), a UML tem sido amplamente adotada como linguagem padrão de modelagem de software nas empresas, gerando uma necessidade cada vez maior de técnica de engenharia reversa de modelos UML. Com modelos bem definidos, eles podem ajudar na compreensão do software e na comunicação entre os times de desenvolvimento e os clientes.

Com o objetivo de ampliar este pensamento, métodos mais eficazes precisam ser criados para que os modelos de projeto da UML possam ser mais amplamente utilizados nas empresas (CHAUDRON et al., 2011). A grande discussão na literatura da área de modelagem de software é sobre os benefícios e os custos do uso da UML dentro do processo de desenvolvimento de software. Segundo Osman e Chaudron (2014) e também Dzidek (2008) reportam sobre os benefícios do uso de modelos UML em atividades de desenvolvimento e manutenção de software. Esses benefícios estão relacionados à redução no esforço de atividades de manutenção, aumento na qualidade do código alterado mediante consulta prévia de modelos UML e redução de defeitos no código.

Além disso, a literatura atual tem proposto várias ferramentas de modelagem que suportam engenharia reversa de modelos UML, tais como *Astah* (ASTAH, 2017), *ArgoUML* (ARGO UML, 2017), *Umbrello Uml Modeler* (UMBRELLO, 2016) e *Visual Paradigm* (VISUAL PARADIGM, 2016). Porém, tais ferramentas não dão suporte à engenharia reversa de *features* da aplicação, pelo contrário, geram modelos UML de todas as *features* da aplicação. Isso representa um problema, visto que os desenvolvedores decompõem aplicações em *features* para permitir que times de desenvolvimento possam focar em partes específicas da aplicação, geralmente aquelas mais relevantes para eles. Ao realizar a implementação ou manutenção de *features*, os desenvolvedores não possuem diagramas específicos das funcionalidades do sistema. Logo, o modelo gerado pelas técnicas tradicionais usualmente não satisfaz às necessidades dos desenvolvedores, sendo diagramas frequentemente grandes, confusos e por vezes sem compreensão adequada.

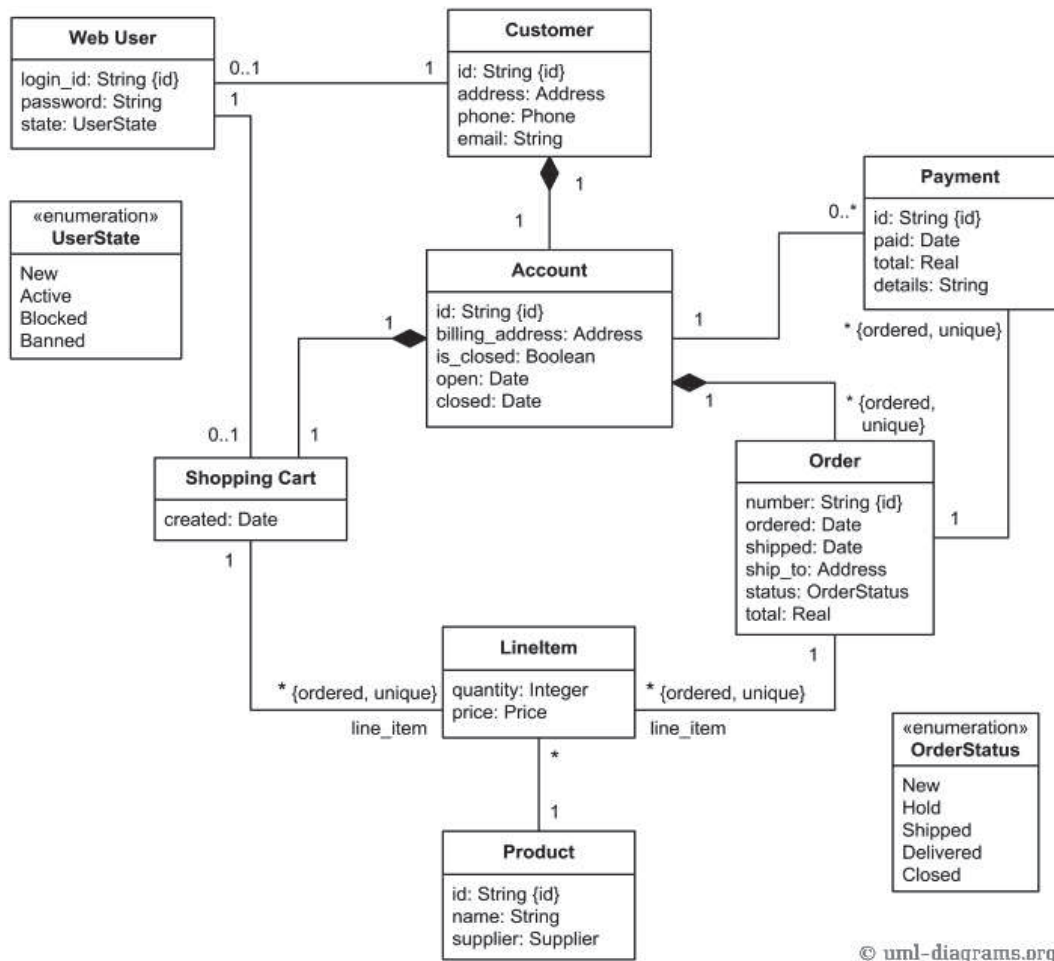
Embora estudos recentes, tais como Fernández-Sáez (2013) reportam questões específicas sobre o uso de modelos UML, tais como o esforço de modelagem, pouco tem sido feito para explorar a temática de engenharia reversa na prática. Desenvolvedores se beneficiariam de uma técnica que gerasse de forma automática novos modelos, sempre que uma nova *feature* no sistema fosse implementada ou alterada. De fato, há uma lacuna na literatura atual de técnicas que mantenham modelos UML sincronizados com o código da aplicação. Em particular, esta lacuna torna-se ainda mais evidente quando busca-se fazer engenharia reversa de *features* específicas da aplicação. Esta problemática é melhor explorada na Seção 1.1 a seguir.

## 1.1 Contextualização do problema

Conforme apresentado anteriormente, as ferramentas que geram modelos UML de forma automática não possibilitam a geração de novos modelos orientados a *features*. Por exemplo, as ferramentas atualmente geram diagramas de classes da UML contendo todas as classes do sistema, ao contrário de ter apenas as classes relativas a um *feature* em particular. Além disso, as ferramentas de engenharia reversa exigem um alto esforço por parte dos desenvolvedores para gerar modelos específicos para *features*, visto que elas são essencialmente manuais ou semiautomáticas. As técnicas atuais (e suas ferramentas de suporte) não dão suporte à engenharia reversa orientada a *features*, essas ferramentas são imprecisas no que se refere a completude dos modelos gerados. Com base nisso, é identificado os problemas que são abordados neste trabalho.

Os estudos não apresentam ganho em reduzir esforços dos desenvolvedores e analistas de software para atualização de modelos orientados a *features*, uma vez que não há ferramentas que sejam capazes de gerar novos modelos orientados a *features*. As ferramentas de engenharia reversa disponíveis não possibilitam a criação de modelos orientados a *features*, assim, não é possível criar novos modelos de *features*. As ferramentas já citadas apenas geram o modelo tradicional UML, por exemplo o diagrama de classes conforme é apresentado na Figura 1.

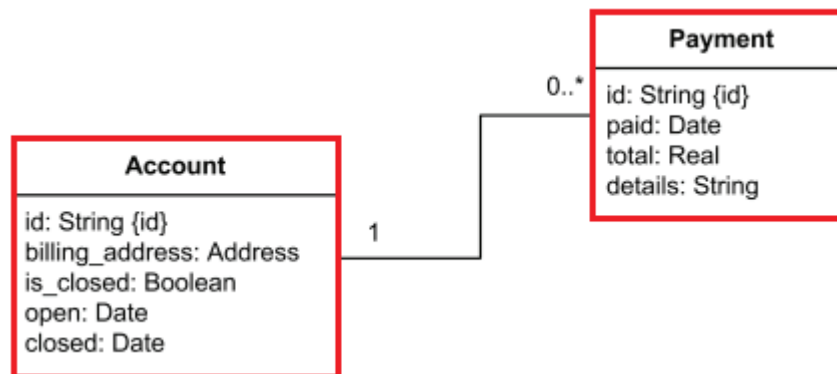
Figura 1 – Exemplo de diagrama de classes.





Isto é, não é encontrado nas ferramentas a funcionalidade de criação de modelos orientados as *features*. Os modelos UML de *features* são modelos que representam funcionalidades específicas do sistema, ao contrário de todas as funcionalidades, como anteriormente mencionado. Na Figura 2 é apresentado um exemplo de diagrama de classes orientado a *features*, onde apenas as classes que envolvem a funcionalidade são apresentadas no diagrama. A seguir a Figura 2 apresenta um diagrama de classes orientado a *features*.

**Figura 2 – Exemplo de diagrama de classes orientado a *features*.**



Fonte: adaptado de uml-diagrams.

O diagrama acima apresentado na Figura 2 ilustra uma *feature* do sistema de pagamento de um *e-commerce*. O diagrama de classes orientado a *feature* representa as informações da conta do pedido e o pagamento. Desta forma, o diagrama representa a relação pedido-pagamento. Essa ausência prejudica a atualização dos modelos UML, pois não contempla as novas funcionalidades que estão sendo adicionadas ao projeto. Desta forma, é possível identificar uma lacuna de pesquisa nesta questão. A geração de modelos orientado a *features* pode ser mais aprofundada e melhor explorada com uma nova técnica. Desta maneira é possível ampliar as pesquisas sobre a geração de modelos orientados a *features*. No Seção 1.2 as questões de pesquisa deste trabalho serão apresentadas.

## 1.2 Questões de pesquisa

Diante da falta de técnicas capazes de realizar engenharia reversa orientada a *features*, bem como estudos que avaliem os benefícios do uso de modelos orientado a *features* em manutenção de projetos de software, este trabalho busca investigar duas lacunas que são formuladas através de duas Questões de Pesquisa (QP) a seguir:

- **QP-1:** O uso de modelos orientados a *features* reduz o esforço dos desenvolvedores e aumenta a corretude das atividades de manutenção de software?
- **QP-2:** Como realizar a engenharia reversa orientada a *features* de forma automática?

### 1.3 Objetivos

O objetivo geral deste trabalho é desenvolver uma *técnica capaz de realizar engenharia reversa orientada a features de diagrama de classes da UML*. Com a criação de uma técnica orientada a *features*, esta técnica busca dar suporte à geração de diagramas que representam aspectos estruturais da UML. A representação destes modelos será a partir do código-fonte com o conceito de engenharia reversa, gerando, assim, modelos como os diagramas de classes da UML. Os objetivos específicos deste trabalho são:

- **Reduzir o esforço na manutenção de desenvolvimento de software.** Este trabalho levanta a hipótese de que o uso de modelos orientados a *features* possa reduzir o esforço na manutenção. Com o uso do modelo orientado a *features*, busca-se também gerar uma análise comparativa com o uso tradicional de modelos UML através da perspectiva do desenvolvedor no contexto de manutenção de software.
- **Propor um algoritmo para dar suporte à engenharia reversa orientada a features.** Este objetivo visa propor a criação de um algoritmo que suporte uma técnica de engenharia reversa orientada a *features*. Com a criação deste algoritmo será possível identificar as *features* do sistema e a partir destas *features* identificadas gerar os modelos orientados a *features*.
- **Propor uma ferramenta que implemente a técnica.** O objetivo de desenvolver uma ferramenta que execute a técnica de engenharia reversa orientada a *features*. Ao desenvolver esta ferramenta, compreende-se que o uso poderá reduzir o esforço de manutenção de diagramas, proporcionando um ganho também em qualidade e produtividade das atividades.
- **Avaliar a técnica.** Este objetivo visa realizar análise de modelos gerados pela técnica e comparar com os modelos ideais para validar a precisão da técnica. Também como parte da avaliação da técnica, estudos qualitativos serão realizados com a participação de desenvolvedores de software.
- **Produzir conhecimento empírico sobre engenharia reversa orientada a features.** Este objetivo visa produzir conhecimento empírico sobre a geração de modelos orientados a *features*. Especificar e evidenciar os erros e acertos produzidos nos modelos gerados pela técnica e mensurar os dados a partir dos experimentos que serão realizados.

### 1.4 Metodologia

Como metodologia deste trabalho, primeiro foi realizada uma pesquisa sobre as ferramentas que usam a engenharia reversa como forma de gerar modelos UML. Esta pesquisa visa encontrar e analisar essas ferramentas que usam o conceito de engenharia reversa. A pesquisa foi importante para relacionar as diferenças entre as ferramentas escolhidas com os critérios estabelecidos para a escolha dos trabalhos. A pesquisa tem como objetivo usar a metodologia deste trabalho para contribuir com a análise dos trabalhos relacionados explorando conceitos e funcionalidades para desenvolvimento de melhores técnicas. (GONÇALVES, 2014)

No segundo momento deste trabalho foi averiguado os aspectos de comparação dos trabalhos relacionados. A análise dos trabalhos relacionados realizada com pesquisas qualitativas e quantitativas identificaram os trabalhos escolhidos para a comparação. Essas análises influenciaram nos comparativos entre os trabalhos obtendo assim resultados na etapa de avaliação deste trabalho. A pesquisa experimental está relacionada em contribuir com novas ideias, novos estudos de prototipação, estudos em laboratórios e amostras criteriosas (JUNG, 2004).

O terceiro passo deste trabalho consiste em apresentar uma técnica de engenharia reversa orientada a *features* como solução do problema apresentado. Também é apresentado nesta etapa os aspectos de desenvolvimento da técnica e a validação da técnica. Na última etapa da metodologia adotada para este trabalho corresponde em analisar a execução da técnica buscando o objetivo de reduzir esforço da atualização de modelos e da qualidade do código.

### **1.5 Organização do trabalho**

Esse trabalho está dividido em capítulos. Cada capítulo irá tratar um tema sobre esta pesquisa. No Capítulo 2, a fundamentação teórica é apresentada, com referências bibliográficas, assuntos que envolvem este trabalho e conceitos sobre engenharia reversa orientada a *features*. No Capítulo 3, por sua vez, os trabalhos relacionados serão apresentados. Neste capítulo é realizado o comparativo entre as ferramentas, técnicas e conceitos que são referências para o assunto deste trabalho. Juntamente neste capítulo é abordado uma descrição detalhada dos trabalhos relacionados.

No Capítulo 4, a técnica de engenharia reversa orientada a *features* é apresentada. O capítulo descreve os conceitos utilizados, frameworks, algoritmos e detalhes de implementação da proposta deste trabalho. No Capítulo 5, o experimento controlado é apresentado. No Capítulo 6, a técnica é avaliada através de um estudo de caso. No Capítulo 7, por fim, algumas considerações finais e trabalhos futuros são apresentados.



## 2 FUNDAMENTAÇÃO TEÓRICA

Este Capítulo apresenta os conceitos fundamentais para o entendimento da proposta deste trabalho. A Seção 2.1 apresenta o conceito de engenharia reversa. A Seção 2.2 descreve o método de modelos de *features*. A Seção 2.3 descreve as informações referente a UML. A Seção 2.4 apresenta a Programação Orientada a Aspecto (POA) e o Aspectj.

### 2.1 Engenharia reversa

Em (PRESSMAN; MAXIM, 2016) relatam que a engenharia reversa no processo de desenvolvimento de software tem a finalidade de gerar informações a partir de códigos já existentes e criar novos artefatos para contribuir com o projeto. O método de engenharia reversa para códigos fontes, podem também ajudar na construção de novos métodos e códigos a partir de modelos UML (KAPPLER et al., 2008). Algumas abordagens de sistemas de software e de algumas ferramentas executam trechos de código para a geração de artefatos ou documentos com o uso da engenharia reversa. Segundo (CHOUAMBE et al., 2008) a engenharia reversa pode ser definida como uma nova arquitetura de software compostas de módulos ou classes. Esta nova arquitetura pode ser criada a partir de um componente de software ou ferramentas que podem gerar novas arquiteturas de software de forma semiautomática ou de forma totalmente automática.

Algumas arquiteturas ou até pequenos projetos de software, não criam documentos referentes ao desenvolvimento realizado. A engenharia reversa pode ajudar na criação de diagramas de classes a partir de sistemas já implementados. Também como método de criar novos modelos para o projeto, os novos modelos gerados podem ajudar na compreensão e no entendimento do sistema. Porém, hoje não há ferramentas que auxiliam a criação e a atualização destes diagramas. Um ponto negativo das ferramentas de geração de diagramas de classes e a metodologia de engenharia reversa. O processo de atualização de novos diagramas não existe, fazendo com que seja maior a rejeição o uso dessas ferramentas. Diferentemente do mundo do hardware, a engenharia reversa de software não é tão utilizada na abstração de informações de um produto concorrente (SCHACH, 2010).

Pressman e Maxim (2016) definem que a engenharia reversa tem a atividade de reverter um código-fonte de software através de suas definições mais abstratas. Dessa maneira é possível identificar as informações pertinentes as alterações específicas de cada código. Tais informações que fazem o entendimento dos sistemas sejam essenciais para a manutenção do software. Abaixo na Figura 3 é apresentando uma ilustração do processo da técnica de engenharia reversa a partir de um código-fonte.

**Figura 3 – Representação de engenharia reversa vs código-fonte**

Fonte: Elaborado pelo autor.

A Figura 3 apresenta uma ideia geral de engenharia reversa com o código-fonte realizando a transformação para um diagrama de classes. Em casos eficientes, a engenharia reversa realiza transformações, de forma que partes do modelo sejam modificadas, em vez de todo o modelo. Muller e outros (2000), destacam a ideia da exploração de informações extraídas pela engenharia reversa no processo de desenvolvimento, ou seja, algumas informações ou artefatos, visões arquiteturais, diagramas de projeto, ligações de rastreamento podem ser úteis para os desenvolvedores através do uso das técnicas de engenharia reversa.

A UML no processo de manutenção de software depende de dois sub processos: a engenharia reversa e a análise do modelo (RIVA et al., 2006). Em estudos empíricos com alguns participantes de pesquisas realizadas, observaram que após adotarem a UML no desenvolvimento de software, os participantes que utilizavam os modelos na manutenção de software tinham mais facilidade no entendimento da tarefa (PETRE, 2013). No entanto, os modelos não atualizados passam a ficar obsoletos, não ajudando na execução das novas tarefas e ocasionando em muitas vezes uma documentação antiga e desatualizada para o projeto. Isso vai contra o uso adequado dos diagramas UML (ANA et al., 2013).

A engenharia reversa realizar um processo de transformação, isto é, código-fonte existente é utilizado para que seja gerado um novo componente a partir do uso deste código. Com o uso da engenharia reversa, é possível obter informações de um programa em que já existe uma operação. Com esta técnica de reversão, é possível até melhorar a operação que já existe. A engenharia reversa reduz o risco e o custo do desenvolvimento de software (HASSAN et al., 2015).

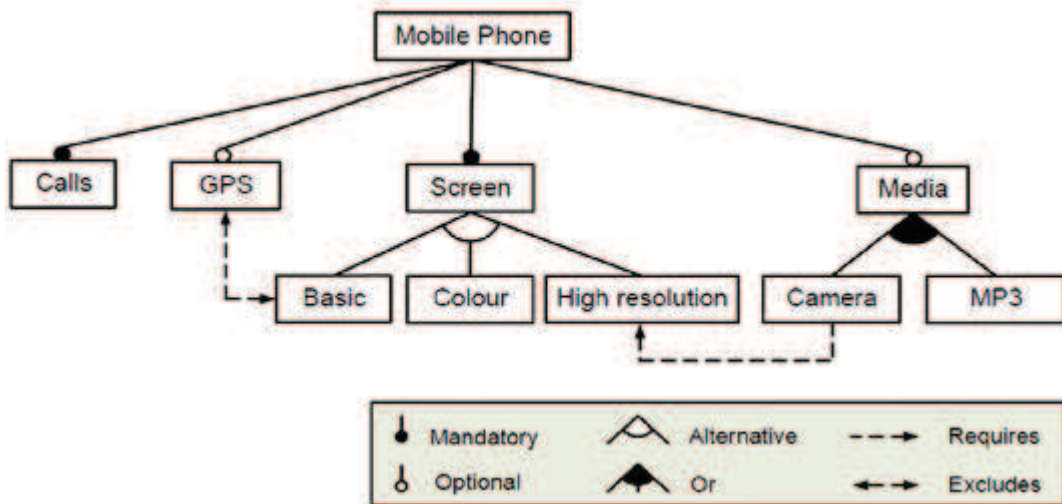
Atualmente, o processo de engenharia reversa é muito útil pois visa definir o comum e a variabilidade da linha de produtos de software. O objetivo da engenharia de domínio em processos de software consiste na alta modificação do código-fonte. Esta é a razão pelo qual os autores decidiram usar a engenharia reversa no processo de desenvolvimento (MALTA; CENTENERA; GONZALEZ-BLANCO, 2017). A Seção 2.2 descreve o conceito de modelos de *features* no desenvolvimento de software.

## 2.2 Modelos de *features*

No desenvolvimento de software, um modelo de *features* é uma representação de uma funcionalidade do software, podendo ser tratado como uma nova característica do software. A modelagem de *features* é a atividade de modelar as propriedades comuns e variáveis de conceitos, e das interdependências entre essas propriedades, organizando-as num modelo coerente denominado modelo de *features*. Um diagrama de *features* consiste num conjunto de nós, um conjunto de segmentos direcionais e decorações nesses segmentos. A raiz de um diagrama de *features* representa um conceito. Há *features* mandatórias, alternativas e opcionais. Uma argumentação detalhada sobre as vantagens da construção de diagramas de *features* para expressar conhecimento sobre um domínio.

O modelo de *features* é uma representação hierárquica (forma de árvore) das propriedades e os relacionamentos entre *features* e sub-*features* (pais e filhos). Adicionalmente, existem relacionamentos cruzados (entre sub-árvores) entre as *features*, representando restrições entre elas (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010). Os conceitos de *features* e de modelagem de *features* foram introduzidos pelo método *Feature-Oriented Domain Analysis* (FODA), criado pelo *Software Engineering Institute* (SEI) para a modelagem de domínio. O propósito da modelagem de domínio é selecionar e definir o domínio em foco, e coletar informação relevante sobre o domínio e integrar essa informação num modelo de domínio.

Figura 4 – Exemplo de modelo de *features*



Fonte: (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010).

A Figura 4 apresenta um modelo de *features* para dispositivos móveis ilustrando as possíveis características do sistema. No modelo, a representação de um telefone celular que obrigatoriamente possui suporte para chamadas (*feature Calls*), possui uma tela (*feature Screen*) e opcionalmente pode conter capacidade de mídia (*features: Media, Câmera e MP3*), sendo que essas últimas *features* é um relacionamento inclusivo no qual indica que qualquer combinação de recursos de multimídia é válida.

## 2.3 UML

*Unified Modeling Language* (UML) é uma linguagem de modelagem de sistemas de informação mantida pela OMG. A principal facilidade do uso da UML no desenvolvimento de software é o entendimento, o mapeamento da estrutura e a arquitetura do software. Existem outros benefícios que são consequência do uso da UML, por exemplo, a geração de documentação para o projeto e a facilidade de comunicação entre desenvolvedores e clientes (OMG-UML, 2015).

Segundo Dobing e Parsons (2006), ao contrário do que diz a literatura popular, os desenvolvedores parecem acreditar que os diagramas UML podem ser entendidos pelos clientes. Clientes estão mais envolvidos com narrativas de casos de uso e diagramas de atividade. A UML tem apoiado a modelagem de dados, a modelagem de negócios, a modelagem de objetos e a modelagem de componentes (UML, 2011). A perspectiva dos objetivos da UML são:

- modelar qualquer tipo de sistema, em termos de diagramas utilizando os conceitos da orientação a objetos;
- estabelecer uma união fazendo com que métodos conceituais sejam também executáveis;
- criar uma linguagem de modelação utilizável tanto pelo homem quanto pela máquina.

A UML atende todos os requisitos de especificação dentro de um processo, desde a fase de análise até a fase de testes e implementação do sistema (CHANTRE et al., 2008). O uso da UML no processo de desenvolvimento do software é benéfico, pois ajuda na compreensão e no entendimento da atividade que o desenvolvedor realizado no processo de desenvolvimento de software. Pensar que antes de realizar a codificação do projeto, o uso de um modelo faz com que os erros e as correções futuras diminuam. Diagramas atualizados e bem consolidados trazem para novos integrantes e novos colaboradores a compreensão e entendimento pelo grupo.

Os resultados de uma experiência controlada para avaliar o impacto dos modelos de design UML na manutenção de software, relatam que o uso da UML tem um ganho na qualidade do software desenvolvido (SCANNIELLO, 2014). Quando ocorre alguma nova implementação ou uma nova *features* no código-fonte, os diagramas UML não refletem as alterações nos modelos. Isso vai contra um uso adequado dos diagramas, visto que após uma modificação do código, o modelo deve ser atualizado (FERNÁNDEZ-SÁEZ et al., 2013).

No entanto, o uso de diagrama dentro do processo de desenvolvimento é bastante importante e utilizado dentro das atividades de desenvolvimento de software o diagrama mais usado é o diagrama de classes (PETRE, 2013). O diagrama de classes foi o modelo UML mais usado em um experimento onde participantes do experimento desenvolviam tarefas de manutenção de software. A necessidade de realizar a atualização dos modelos após a manutenção do código-fonte é vital para o sistema, porém devido ao tempo na demora na atualização dos modelos e da falta de ferramentas que atualizam os modelos fazem com que isto não seja priorizado dentro das empresas e das indústrias. Com isso, a engenharia reversa pode contribuir para a atualização dos modelos. A atualização automática das alterações realizadas no código-fonte faz com que a engenharia reversa orientada a *features* seja diferencial.

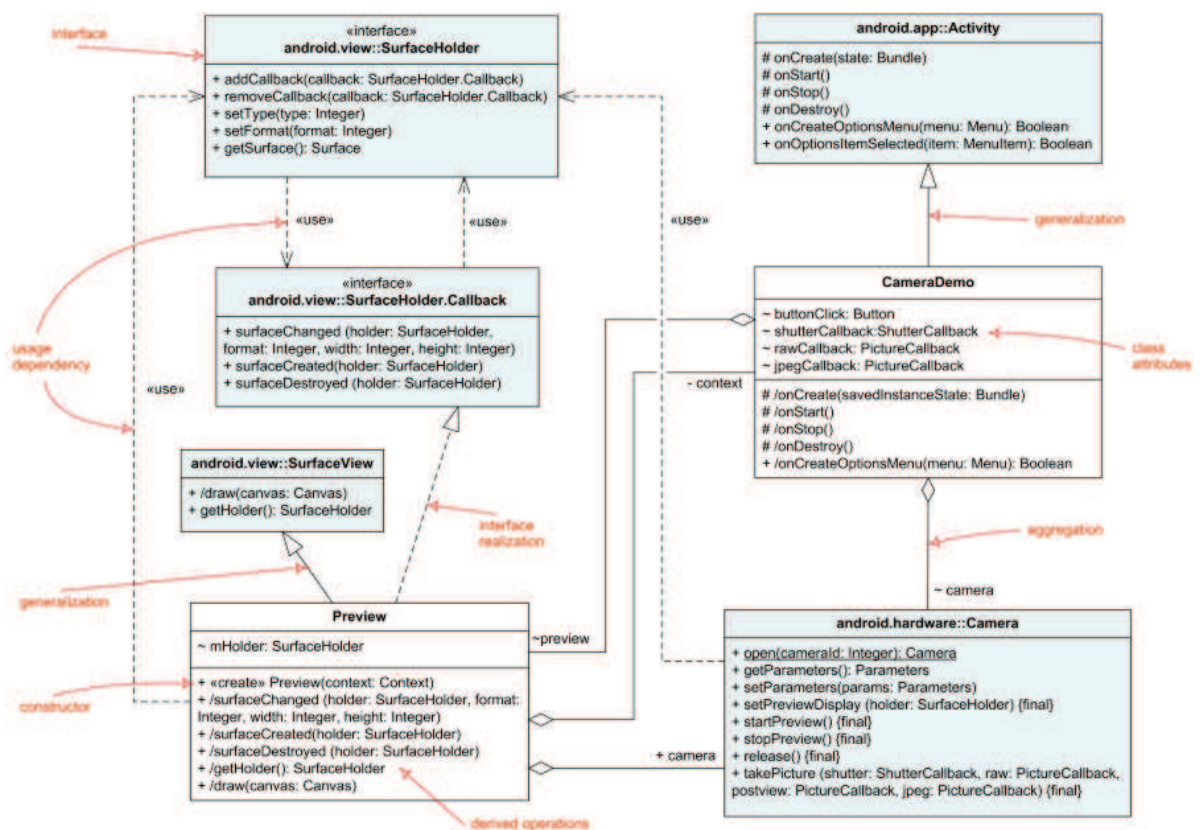


### 2.3.1 Diagrama de classes

O diagrama de classes apresenta a arquitetura das classes do sistema e pode conter diferentes níveis de detalhamento das informações relacionada, dependendo do propósito com que foi construído e de acordo com a fase do projeto. Segundo Petre (2013), o diagrama mais usado em atividades de manutenção de código é o diagrama de classes. Um diagrama de classes pode conter as classes importantes de um sistema, seus atributos, métodos, e principalmente as associações entre as classes.

Para Fowler (2011), o diagrama de classes representa o sistema, descrevendo os tipos de objetos e os tipos de relacionamentos estáticos que existem entre eles. Como ele serve de base para vários outros diagramas e é considerado como um dos principais, muitas vezes ele é utilizado como precursor no design do software. Um diagrama de classes pode representar o sistema completo com classes, atributos e métodos. A Figura 5 ilustra os elementos típicos que compõem um diagrama de classes.

**Figura 5 – Implementação de elementos do diagrama de classes**



Fonte: uml-diagrams, 2017.

Segundo Faitelson e Tyszberowicz (2017), grandes modelos UML são complicados de serem compreendidos. Como forma de melhorar o uso de diagramas complexos seria de agrupar entidades em pacotes, desta forma seria mais fácil o entendimento dos modelos complexos dos sistemas. Para os autores, modelos mais abstratos têm uma maior aceitação por pessoas com pouco conhecimento em UML.

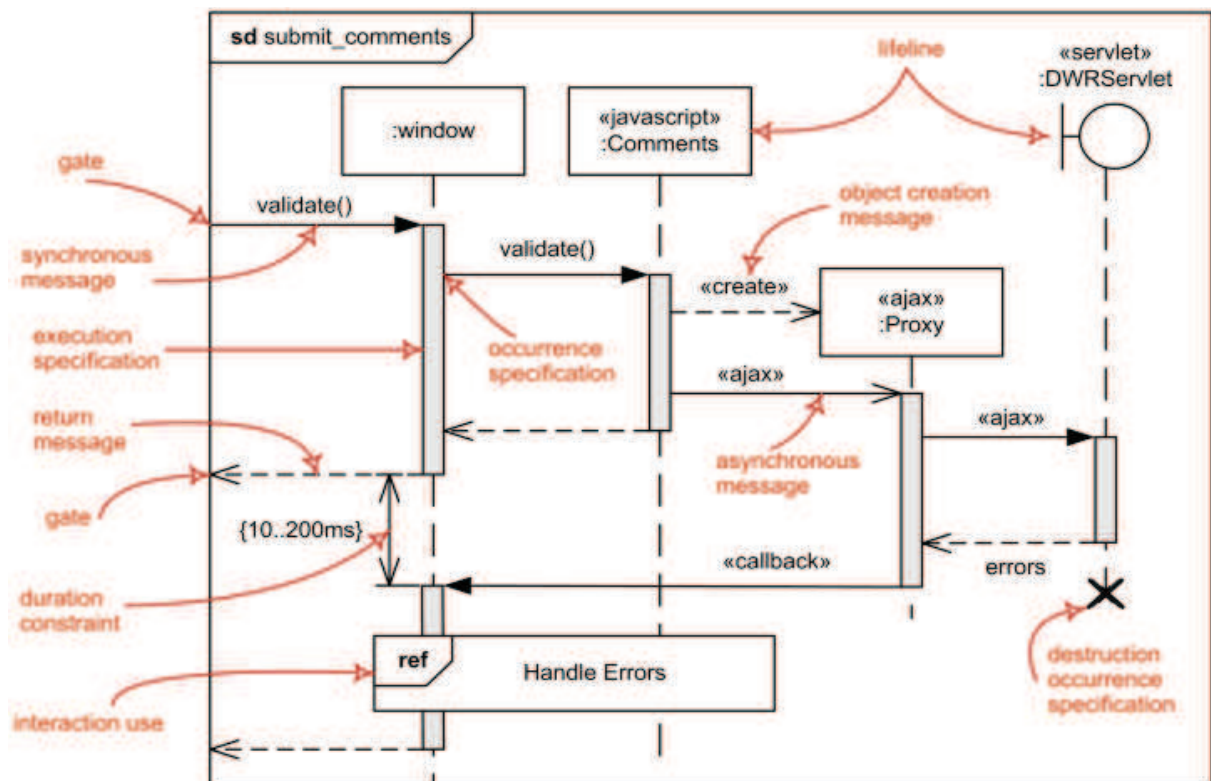
### 2.3.2 Diagrama de sequência

Diagramas de sequência UML fazem parte de um modelo UML e existem somente em projetos de modelagem UML. Um diagrama de sequência descreve uma interação focando a sequência de mensagens que são trocadas, juntamente com suas correspondentes. O tipo mais comum é o diagrama de sequência, que se concentra no intercâmbio de mensagens entre suas interações. (MSDN, 2017).

Os diagramas de sequência UML modelam o fluxo de lógica em seu sistema de forma visual, permitindo que ambos documentem e validem sua lógica, e são comumente usados para fins de análise e design. Os diagramas de sequência são o artefato UML mais popular para modelagem dinâmica, que se concentra na identificação do comportamento no seu sistema. Os diagramas de sequência, juntamente com diagramas de classes, são os modelos de nível de projeto mais importantes para o desenvolvimento de aplicativos comerciais modernos (Agile Modeling, 2017).

Normalmente, um diagrama de sequência captura o comportamento de um único cenário. O diagrama mostra uma série de objetos de exemplo e as mensagens que são passadas entre esses objetos dentro do caso de uso (Safari Book Online, 2017). A Figura 6 apresenta um diagrama de sequência.

**Figura 6 – Representação do diagrama de sequência**



Fonte: UML Sequence Diagrams, 2017.

O diagrama de sequência é o tipo de diagrama de interação mais comum, que se concentra no intercâmbio de mensagens. O diagrama de sequência descreve uma interação concentrando-se na sequência de mensagens que são trocadas, juntamente com suas

especificações de ocorrência (UML Sequence Diagrams, 2017). A seguir a Seção 2.4 descreve o conceito de programação orientada a aspectos.

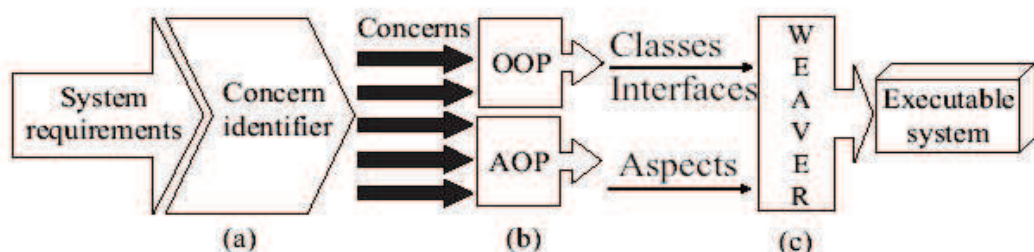
## 2.4 Programação orientada a aspectos

A Programação Orientada a Aspectos (POA) surgiu com o objetivo de estender a Orientação a Objetos (OO) com recursos para dar suporte à modularização de interesses transversais (LADDAD, 2003). Tipos de interesses como requisitos funcionais e requisitos não funcionais. A Programação Orientada a Aspectos surge como uma proposta para a construção de sistemas complexos através da separação de interesses, como aqueles que não se aplicam à lógica de negócios como, por exemplo, segurança, desempenho, persistência, integridade de dados e tratamento de erros, dentre outros (GOETTEN, 2006).

A POA é dividida em um sistema de partes para uma melhor compreensão das fronteiras que definem suas funções. Toda parte relacionada a um interesse é transportada para uma localização física separada das demais, de forma a permitir maior usabilidade e do software (KICZALES, 2001). As linguagens de POA já suportam esta funcionalidade através de construções linguísticas bem definidas. Usando eventos baseados em POA, é possível a criação de pontos que escritos combinam com um histórico sequência de eventos.

Os paradigmas atuais não atendem às necessidades para a implementação dos requisitos de um sistema completo. O desenvolvimento orientado a aspectos promove a separação avançada desde o nível da implementação até os outros estágios do processo de desenvolvimento, incluindo especificação de requisitos, análise e desenvolvimento de projeto (LADDAD, 2003). A Figura 7 apresenta o fluxo de desenvolvimento orientado a aspectos dividido em 3 fases: a primeira etapa realiza a identificação e caracteriza os requisitos (a); a segunda etapa realiza a implementação de requisitos não-transversais em classes e a implementação de requisitos transversais em aspectos (b). a última etapa (c) é chamada a etapa de Weaver que configura a ferramenta e combina o código de orientação do objetivo com o código de orientação de aspectos para execução do sistema (SOARES; BORBA, 2005).

**Figura 7 – Desenvolvimento orientado a aspectos**



Fonte: SOARES; BORBA, 2005.

### 2.4.1 Pontos de junção (*Join points*)

*Join points* são pontos bem definidos na execução do programa. Os *join points* são os pontos dentro do fluxo de execução da aplicação que são candidatos a terem os *advices* aplicados. Um *join point* é um ponto na execução da aplicação que pode sofrer a introdução

de um *advice*. É importante ressaltar que essa introdução preserva o código-fonte intacto, pois ela acontece somente em tempo de compilação ou execução, e adiciona um novo comportamento no ponto especificado (WAND, 2004).

Um *join point* (ponto de junção) é um ponto identificável do fluxo de um programa. Pode ser uma chamada de método ou a configuração do valor de uma variável. Variadas implementações da orientação a aspectos suportam variados tipos de *join point* (Java Framework Portal, 2017).

#### 2.4.2 Pontos de atuação (*Pointcuts*)

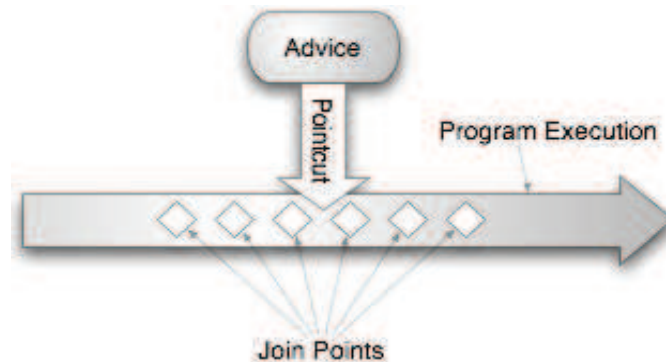
Um *pointcut* define os pontos de interesse na execução de um programa, que deve ser associado ao *Advice*, que será explicado no próximo tópico. O aconselhamento pode ser suportado pela estrutura de programação orientada a aspectos. *Pointcuts* permitem especificar onde que se deseja ser aplicada. Muitas vezes é possível especificar esses *pointcuts* usando nomes de classes, métodos explícitos ou através de expressões regulares que definem padrões de classe correspondente ao método utilizado (KICZALES, 2001). Algumas estruturas de programação orientada a aspectos (POA) permitem criar *pointcuts* dinâmicos que determinam se ao aplicar o conselho com base em decisões de tempo de execução, tais como o valor dos parâmetros do método.

Um *pointcut* é uma construção que seleciona *join points*. Depois de capturar um *join point* é possível especificar as regras nesses nos pontos de junção, como também executar determinada ação antes ou depois da execução dos pontos de junção. Um *pointcut* pode selecionar um *join point* que é uma chamada de um método especificado. *Pointcuts* especificam as regras de junção (onde devem ser feitas), e *join points* são as situações que satisfazem essas regras. É possível criar um *pointcut* para todos os métodos chamados checar, o *pointcut* definiu a regra: métodos chamados checar. Cada método chamado checar será um *join point* desse *pointcut*

#### 2.4.3 Adendo (*Advice*)

Os adendos são compostos de duas partes: a primeira delas é o ponto de atuação, que define as regras de captura dos pontos de junção; a segunda é o código que será executado quando ocorrer o ponto de junção definido pela primeira parte. O adendo é um mecanismo bastante similar a um método (comparado a programação orientada a objetos), cuja função é declarar o código que deve ser executado a cada ponto de junção em um ponto de atuação. Possui o código que será executado no ponto de junção capturado por um *pointcut*. Este é um objeto que incluem chamadas de API às preocupações globais do sistema que representam a ação a ser executada em um *join point* especificado por um ponto. *Advice* é a implementação lógica transversal. O interesse transversal que precisa ser aplicado nos *pointcuts* definem em quais *join points* o *advice* será aplicado. O conceito chave é que os *pointcuts* definem em quais *join points* são aplicados os *advices* (KICZALES, 2001). A Figura 8 ilustra o funcionamento da programação orientada a aspectos.

**Figura 8 – Funcionamento da programação orientada a aspectos**



Fonte: The AspectJ Team, 2017.

A Figura 8 apresenta os conceitos da POA. O *advice* contém o interesse transversal que precisa ser aplicado. Os *join points* são os pontos dentro do fluxo de execução da aplicação que são candidatos a terem os *advices* aplicados. Os *pointcuts* definem em quais *join points* o *advice* será aplicado. O conceito chave dos *pointcuts* é de definir em quais *join points* são aplicados os *advices*.

#### 2.4.4 AspectJ - Orientação a Aspectos com Java

O AspectJ (The AspectJ Team, 2017) é uma extensão para a linguagem de programação Java. Por isso, houve ainda uma preocupação pela compatibilidade de quatro itens extremamente importantes e essenciais:

- **compatibilidade total** - todo programa Java válido é também um programa AspectJ válido;
- **compatibilidade de plataforma** - todo programa AspectJ pode ser executado em uma máquina virtual Java (JVM);
- **compatibilidade de ferramentas** - deve ser possível estender ferramentas existentes para suportar o AspectJ de uma forma natural; isto inclui IDE's de ferramentas de documentação e ferramentas de projeto;
- **compatibilidade para o programador** - Ao programar com AspectJ, o programador deve sentir-se como se estivesse utilizando uma extensão da linguagem Java.

Há uma divisão de duas partes na programação orientada a aspectos são elas: a linguagem de especificação e a linguagem de implementação. A parte da linguagem de especificação define a linguagem na qual o código é escrito; com aspectos, os interesses funcionais são implementados em Java, e para implementação da combinação de interesses sistêmicos são utilizadas as extensões disponibilizadas pelo próprio AspectJ. A parte da linguagem de implementação fornece ferramentas para compilação, debug e integração com ambientes integrados de desenvolvimento.

AspectJ (The AspectJ Team, 2017) suporta dois tipos de implementações de requisitos transversais: A dinâmica que permite definir implementação adicional em pontos bem definidos do programa. E a estática que permite alterar as assinaturas estáticas

das classes e interfaces de um programa Java (LADDAD, 2003). No capítulo a seguir será apresentado as pesquisas relacionadas ao estado da prática e ao estado da arte deste trabalho.

### 3 TRABALHOS RELACIONADOS

Este Capítulo apresenta a relação dos trabalhos que exploram o assunto deste trabalho. A Seção 3.1 apresenta a metodologia para a seleção dos trabalhos. Na Seção 3.2 são apresentadas as análises dos trabalhos do estado da arte. A Seção 3.3 apresenta a análise das ferramentas que foram selecionadas para o estado da prática deste trabalho. A Seção 3.4 descreve a comparação dos trabalhos juntamente com os critérios definidos para a análise comparativa entre os trabalhos relacionados. Por fim, na Seção 3.5 são apresentadas as oportunidades de pesquisa deste trabalho.

#### 3.1 Metodologia para seleção dos trabalhos

O tema de engenharia reversa orientada a *features* envolve duas questões principais: a engenharia reversa e os modelos UML de *features*. A partir da engenharia reversa, é possível gerar modelos UML a partir do código-fonte. Uma funcionalidade do sistema pode ser capturada através da programação orientada a aspectos. Com o uso da engenharia reversa e das funcionalidades do sistema, é possível gerar novos modelos UML orientados a *features*. Com isto, a engenharia reversa e o conceito de *features* são as questões principais deste trabalho.

Para isso, foram consultadas as principais bibliotecas digitais de publicações de pesquisa como, por exemplo, ACM, Springer, Elsevier e IEEE. Além dos repositórios citados, foram pesquisados, também, trabalhos de pesquisa em nível de Pós-Graduação em Universidades nacionais e estrangeiras. Diversas palavras-chaves foram utilizadas para as pesquisas dos trabalhos. As palavras-chaves são referências nos trabalhos do estado da arte e que foram essenciais para as buscas dos trabalhos relacionados a este trabalho, por exemplo, UML, engenharia reversa, engenharia reversa orientada a *features*, programação orientada a aspectos, diagrama de classes, modelagem, manutenção de software, estudos empíricos, entre outros.

Diversos trabalhos foram encontrados com base nas pesquisas realizadas. No campo da engenharia reversa, alguns trabalhos científicos do estado da arte foram selecionados onde apresentam conceitos de engenharia reversa gerando modelos UML completos, por exemplo, Modisco (BRUNELIERE et al., 2014), scrYUML (DECKER et al., 2016), ForUML (NANTHAAMORNPHONG et al., 2015) e fREX (BERGMAYR et al., 2016). Também foram selecionadas algumas ferramentas comerciais que usam a engenharia reversa de código-fonte, por exemplo, Astah (ASTAH, 2017), Argo UML (ARGO UML, 2017) e Visual Paradigm (VISUAL PARADIGN, 2016). As pesquisas realizadas foram importantes para as análises detalhadas de cada uma das ferramentas e para elaboração dos critérios de avaliação entre os trabalhos e ferramentas.

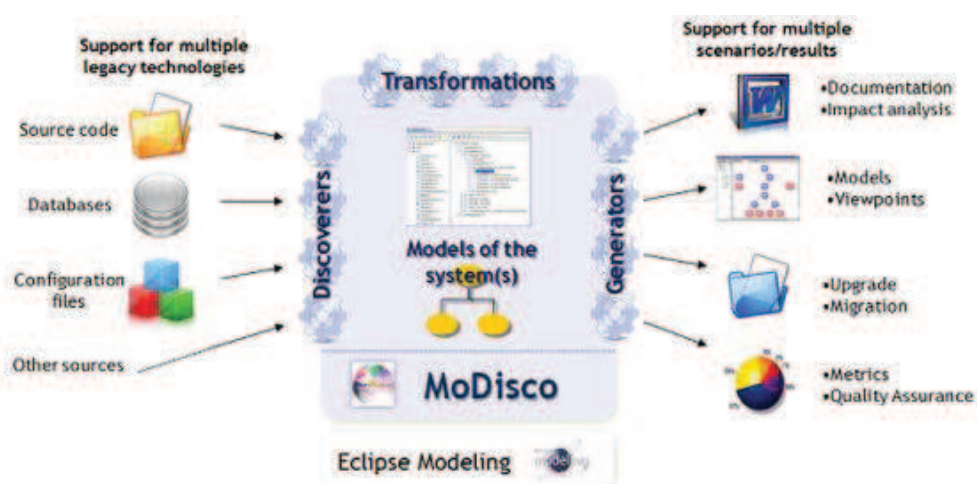
Com a pesquisa elaborada de trabalhos científicos e de ferramentas estudadas, foram selecionados 8 trabalhos para a análise do estado da arte e 5 ferramentas para a análise do estado da prática. Esses trabalhos que foram selecionados, serão apresentados nas seções a seguir para compor um estudo mais detalhado do objetivo proposto desta pesquisa. Os trabalhos pesquisados contribuíram para o desenvolvimento desta pesquisa, com a análise e comparação de técnicas de engenharia reversa do estado prática ou para o estado da arte. Estes trabalhos contribuíram para o conhecimento utilizados como base teórica nas seções seguintes deste trabalho. A seguir, na Seção 3.2 será apresentada a análise do estado da arte.

### 3.2 Análise do estado da arte

Com o estudo realizado sobre trabalhos relacionados sobre o tema desta pesquisa, foram selecionados alguns trabalhos do estado da arte. Os trabalhos selecionados serão detalhados com uma análise aprofundada observando algumas características dos trabalhos. Abaixo serão apresentados os trabalhos.

**MoDisco** (BRUNELIERE et al., 2014). O MoDisco é uma ferramenta com técnicas de engenharia reversa de MDE e que busca facilitar a construção de modelos e soluções baseadas em sistemas herdados de engenharia reversa. A utilização do MoDisco em projeto de nível superior de modelagem de software ajuda na compreensão e nas fases do processo de desenvolvimento de software. O projeto oficialmente parte da Eclipse Foundation que está integrado no projeto de nível superior de modelagem que promove técnicas de MDE e de seus desenvolvimentos dentro da comunidade Eclipse. Atualmente é reconhecido como uma das principais ferramentas para uma solução industrializada graças a uma colaboração ativa pelo OMG fornecendo implementações de referência e ferramentas de algumas empresas de software.

**Figura 9 – Visão geral do projeto Eclipse-MDT MoDisco**



Fonte: BRUNELIERE et al., 2014

Conforme ilustrado na Figura 9, os tipos de artefatos legados possíveis (por exemplo, código-fonte, bases de dados, arquivos de configuração, documentação, entre outros) estão disponíveis através da ferramenta Modisco. O MoDisco tem como objetivo fornecer recursos necessários para criar as representações de modelos correspondentes e permitir a análise dos modelos.

O Eclipse Modeling Framework (EMF, 2017) oferece vários componentes, transformações e geradores. Como resultado, o framework visa a produção nos sistemas legados, dependendo do uso dos resultados de engenharia reversa (por exemplo, modernização de software, refatoração, documentação e análise de qualidade). Um dos principais objetivos do MoDisco é permanecer adaptável a muitos cenários diferentes, facilitando assim a sua adoção por uma base maior de usuários.

Através do pacote de ferramentas de modelagem do Eclipse, o MoDisco suporta parte dos cenários industriais de engenharia reversa geralmente que envolvem diferentes tipos de



entradas. Os cenários são considerados reversão de código-fonte, documentação ou dados brutos, bem como diferentes tipos de resultados que podem ser extraídos através do MoDisco. É possível também ter como resultado código-fonte, uma nova documentação, também modelos (visualizações) e métricas. O framework MoDisco já foi aplicado e implantado em vários casos de uso real com os objetivos subjacentes, por exemplo, validando sua capacidade de gerenciar alguns cenários industriais.

**SrcYUML** (DECKER et al.,2016). É uma ferramenta escrita em C++ para engenharia reversa de diagramas de classes UML. O srcYUML é uma representação em XML do código-fonte com os *tokens* (por exemplo, identificadores, palavras-chave, operadores). A ferramenta permite uma conversão extremamente rápida do código-fonte para os projetos do tipo srcYUML. Uma vez que um projeto é convertido é gerado o diagrama de classes UML que representa o código-fonte. Como o srcML é uma instância de XML, qualquer XML padrão pode ser aplicado para realizar a engenharia reversa na ferramenta.

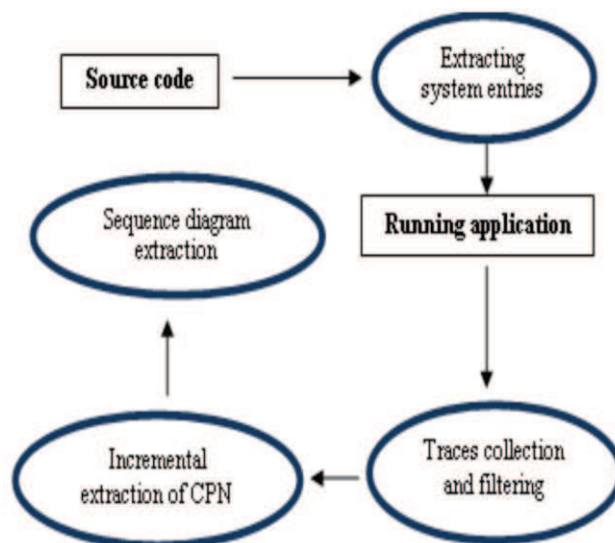
SrcYUML é uma ferramenta de linha de comando que atualmente somente funciona para a plataforma Linux. A ferramenta é altamente eficiente e com uma alta precisão para gerar diagramas de classes. A ferramenta usa mapeamentos conforme definidos para chegar a uma as classes do projeto. Atualmente, o srcYUML suporta os seguintes recursos:

- tipos de classe: classe e interface (sem tipos de dados definidos).
- detalhes do atributo: multiplicidade, tipo e visibilidade;
- detalhes da operação;
- detalhes dos parâmetros, incluindo direção;
- associações incluindo generalização, agregação, composição;

Além disso, a srcYUML suporta outras linguagens de programação (por exemplo, Java e C#). Ao apoiar outras linguagens adicionais, o srcYUML também permite modelar projetos em diferentes linguagens.

**Reverse Engineering of UML Sequence Diagram** (BAIDADA; JAKIMI, 2016). Com a criação da ferramenta de geração de diagrama de sequência A engenharia reversa ajuda a automatizar o processo de criação de novos modelos UML. Com o uso da engenharia reversa do código-fonte é possível ter documentação atualizada de acordo com o desenvolvimento do código. Isso pode ser alcançado observando o código. A documentação enfrenta problemas relacionados à natureza altamente dinâmica de alguma aplicação. Mesmo com a estrutura definida de alguns objetos da aplicação, pode ser definido dinamicamente no tempo de execução. A engenharia reversa é o processo de análise de um assunto onde o sistema identifica componentes e suas inter-relações. Também busca criar representações do sistema em uma outra forma ou em níveis superiores de abstração. Abaixo na Figura 10 é apresentada a visão geral da técnica.

**Figura 10 – Visão geral do processo proposto**



Fonte: BAIDADA; JAKIMI, 2016

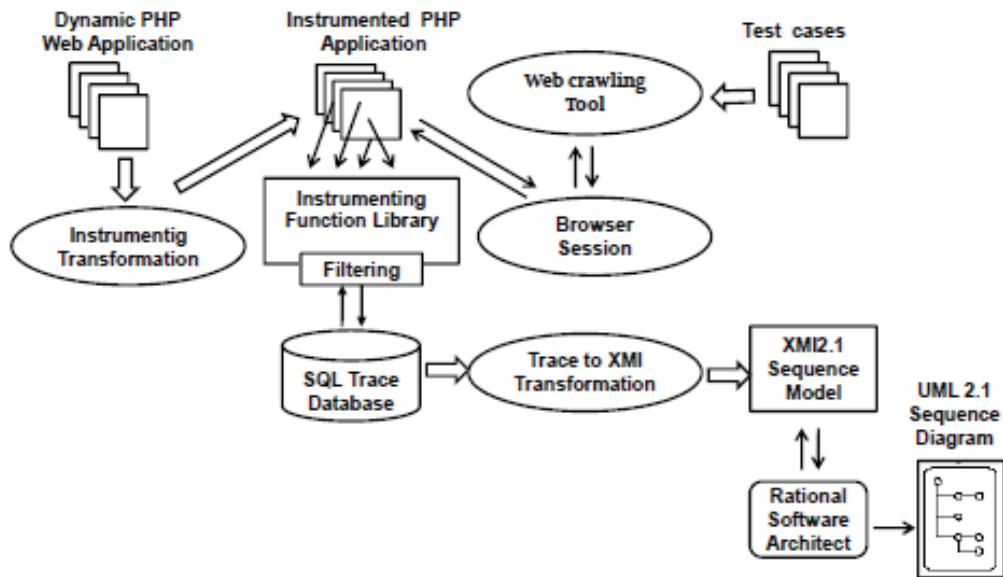
A engenharia reversa consiste em extrair a informação do código para gerar os modelos de alto nível que ajudam a entender o comportamento do sistema. A abordagem proposta para a engenharia reversa dos diagramas de sequência UML é definida em quatro etapas principais, são elas: Extração de valores de entrada do sistema do código-fonte; coleção e filtragem; vestígios transformação em redes; diagrama UML.

**PHP2XMI** (ALALFI; CORDY; DEAN, 2009). O PHP2XMI é parte essencial de um framework destinado a teste de conformidade de aplicativos Web dinâmicos. A engenharia reversa de diagramas de sequência para aplicativos da Web foi implementado usando linguagens de script, como a linguagem PHP. O projeto buscou identificar os elementos de interação na plataforma web. Em geral, o diagrama de sequência facilita o processo de compreensão do comportamento de interação do sistema, e o PHP2XMI ajudou a deduzir as permissões para cada usuário de cada página Web.

A elaboração dos objetivos das aplicações Web muitas vezes têm múltiplos pontos de entrada e comportamento de exibição, isso é difícil de detectar até o tempo de execução do projeto. Com o uso do PHP2XMI é possível recuperar automaticamente diagramas de sequência do usuário. Algumas características como representação do conjunto completo de comportamentos, mudanças de estado, estado na terminologia do diagrama de sequência são elaboradas não são atendidas. As aplicações Web têm vários componentes que podem ser afetados por uma única execução de página. A aplicação usa o banco de dados e as variáveis de sessão e de cookie. Com isto, a ferramenta utiliza a busca de informação do banco de dados para automatizar o processo da prática de instrumentos web dinâmicos. Com o uso da engenharia reversa é possível recuperar a informação das páginas, salvar esta informação em banco de dados e posteriormente gerar o diagrama de sequência.

A ferramenta gera diagramas de sequência a partir da engenharia reversa de informação gerada na Web baseados na linguagem PHP. O resultado pode ser importado e visualizado em qualquer conjunto de ferramentas UML 2.1.

**Figura 11 – Arquitetura da ferramenta PHP2XMI**



Fonte: ALALFI; CORDY; DEAN, 2009.

PHP2XMI é uma ferramenta desenvolvida com a engenharia reversa voltada para recuperação de diagramas de sequência de forma dinâmica baseada em PHP. A abordagem utilizada no PHP2XMI envolve três etapas, são elas:

- instrumentação dinâmica: núcleo de captura automática do código-fonte e informações dinâmicas, como endereço de página, variáveis de protocolos, sessões e cookies.
- filtragem e armazenamento: durante as sessões interativas do navegador, os traços de execução gerados pelas capturas são filtrados, as informações redundantes são descartadas e as que não são, posteriormente são armazenadas em um banco de dados para análise futura.
- análise de banco de dados e geração de modelo: a execução os traços armazenados no banco de dados são transformados em elementos de meta-modelo de sequência UML.

Os diagramas gerados pela ferramenta representam a forma da web presente no código-fonte. Porém, será realizada no futuro usando uma técnica de cobertura de instrumentação para garantir a exatidão e integridade do diagrama de sequência gerado. Também está sendo planejado a integração de diagramas de sequência de diferentes sessões para gerar um completo diagrama de sequência para todo o projeto de aplicação Web.

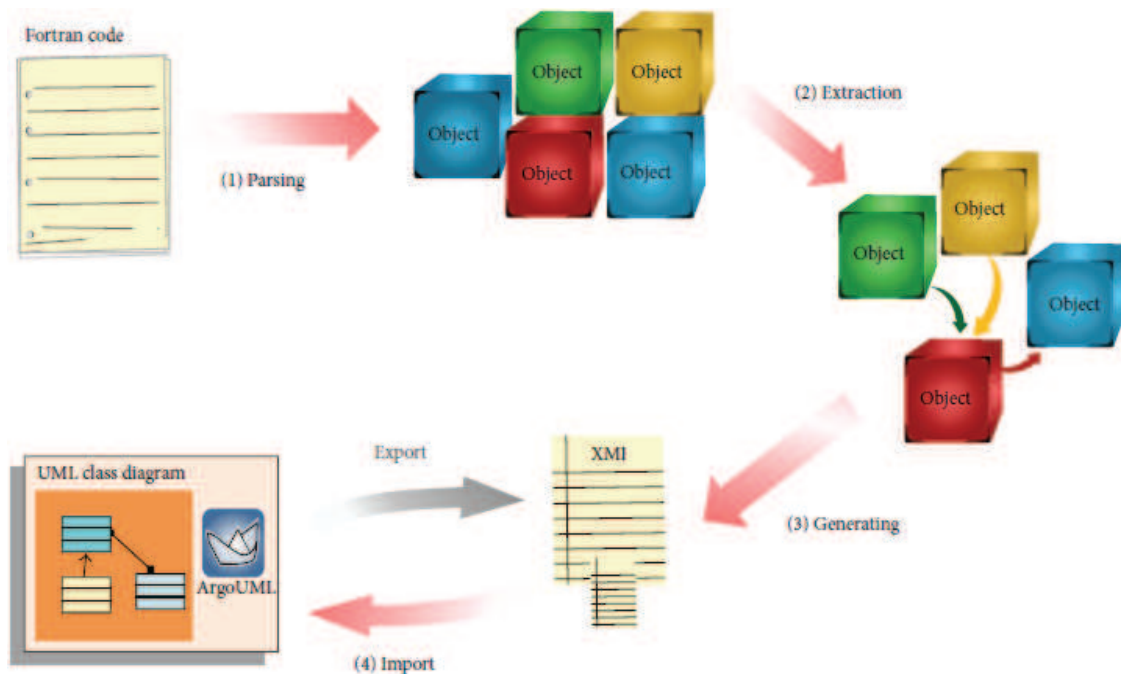
**ForUML** (NANTHAAMORNPHONG et al., 2015). A tecnologia ForUML foi criada com uma abordagem utilizando a engenharia reversa para transformar o código-fonte Fortran em modelos UML. O processo de transformação no ForUML é baseado no Formato XMI, que fornece um método padrão de mapeamento em um modelo de objeto em XML. A ferramenta exibe os resultados da análise usando o ArgoUML (ARGO UML, 2017) ferramenta de modelagem. O ForUML é independente do compilador e capaz de gerar UML para todos os tipos de código OO Fortran. A avaliação da precisão da técnica usando cinco pacotes de

software que usam recursos orientados a objetos dos padrões compilador Fortran 95, 2003 e 2008. O documento fornecendo mais informações de fundo e mais detalhes sobre a transformação processo em ForUML. As contribuições desta ferramenta são as seguintes:

- o ForUML *Tool* ajudará os desenvolvedores a extrair diagramas UML do código que permita e tomem boas decisões sobre o software nas tarefas de desenvolvimento e manutenção;
- descrição do processo de transformação usado para desenvolver o ForUML, o que pode ajudar outros autores de ferramentas criar ferramentas para a comunidade;
- feedback da oficina que deve ajudar a desenvolver melhores práticas e ferramentas adequadas para uso;

As características dos desenvolvedores sobre se beneficiar de uma ferramenta que cria documentação do sistema com um menor esforço é satisfatória. Com a comunidade da engenharia de software normalmente usa engenharia reversa para solucionar este problema é válido ter uma ferramenta que realiza a criação de documentação para o software. Este trabalho destina-se principalmente a desenvolvedores da comunidade de engenharia de software que desenvolve em Fortran. Os diagramas de classes UML suportam a manutenção e evolução de software. Os desenvolvedores podem usar os diagramas UML para ilustrar conceitos de design de software. A Figura 12 apresenta o fluxo de execução da ferramenta ForUML.

**Figura 12 – O processo de transformação de FORUML**



Fonte: NANTHAAMORNPHONG et al., 2015

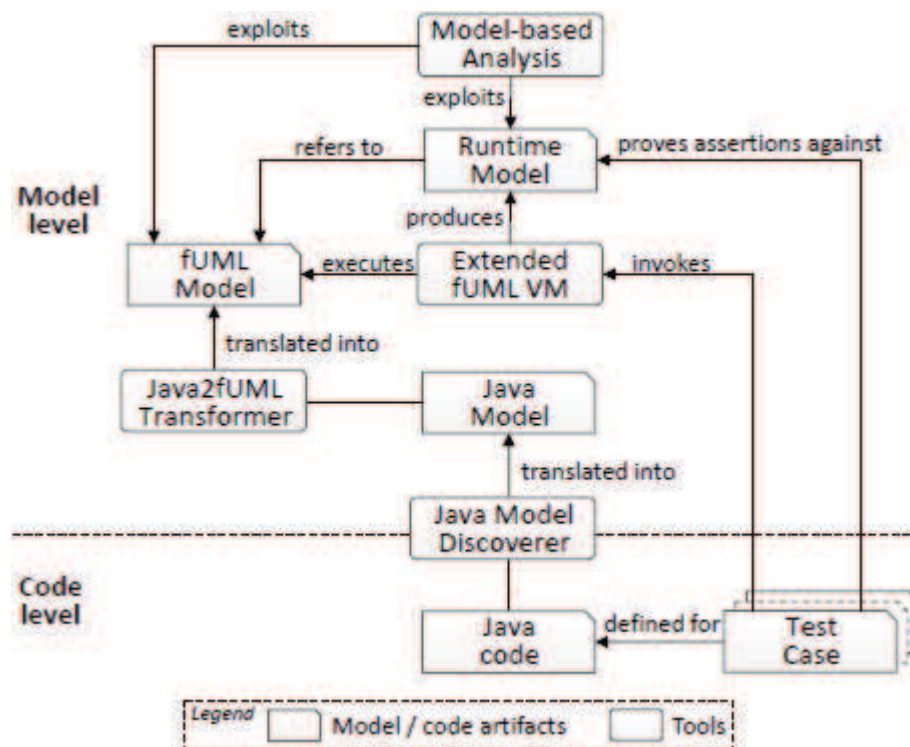
Além disso, diagramas UML podem ajudar os desenvolvedores a examinar relacionamentos entre objetos para identificar problemas e soluções no software com o uso do ForUML. Como as ferramentas de engenharia de software geralmente melhoram a produtividade, o ForUML pode reduzir o tempo de aprendizagem e com uma curva menor

necessária para aplicar práticas no desenvolvimento do software. Por exemplo, o FORUML ajudará os desenvolvedores realizarem atividades de refatoração.

ForUML conseguiu transformar automaticamente o código-fonte em diagramas UML corretos. Uma das contribuições do ForUML é a sua capacidade de código Fortran de engenharia reversa. ForUML integra a capacidade de visualização na ferramenta ArgoUML para o diagrama de classes. O ForUML pode ser usado por pessoas durante o processo de desenvolvimento de software, especialmente para desenvolvedores. No entanto, o ForUML tem algumas limitações que podem ser melhoradas com algumas novas implementações e melhorias na ferramenta.

**fRex** (BERGMAYR et al., 2016). A ferramenta fRex tem uma dupla contribuição nos seguintes aspectos, são eles: Gerar modelos automaticamente a partir de aplicativos existentes e realizar um mapeamento entre modelos UML para diagramas de atividade/classe e os recursos de linguagem principal do Java, colocando o foco em aspectos comportamentais e sua execução a nível de modelo. Para obter as informações em tempo de execução, o fUML utiliza uma máquina virtual dedicada (VM) que foi estendida para fornecer traços de execução como um modelo de tempo de execução. A Figura 13 apresenta a arquitetura geral da ferramenta fRex.

**Figura 13 – Arquitetura geral do fRex**



Fonte: fRex, 2016

O fUML fornece conceitos para definir classes com atributos e operações, herança múltipla e enumerações bem como um sistema de tipo extensível. Como a abordagem baseada em UML é uma linguagem com várias visualizações interessantes. O diagrama de sequência UML pode ser produzido usando análise de rastreamento, convertendo a VM fUML produzindo traços para diagramas de sequência UML. Derivando parcialmente as representações do comportamento do software, que permitiram a quantidade certa de

informações a serem transmitidas a cada *stakeholder* envolvido no sistema. Finalmente, também é possível verificar a potencial aplicação de linguagens específicas de domínio para análise comportamental, destacando aspectos que não são diretos para representar em modelos UML puros.

Esta ferramenta apresenta uma abordagem de MDRE que permite a análise dinâmica de comportamentos de software. Obtido um modelo fUML adequado, pode ser diretamente executado pela VM fUML em uma fase de compreensão do modelo. A ideia geral da ferramenta é realizar a análise comportamental a nível do modelo gerado beneficiando assim as características interessantes em termos de genéricos, reutilizáveis e extensibilidade do modelo. A abordagem é particularmente útil quando os aspectos dinâmicos do software em um determinado processo de engenharia reversa são necessários. Obviamente, o escopo também pode ser estendido para outras aplicações práticas como a extração de carga de trabalho em sistemas de banco de dados em larga escala.

O mapeamento em Java para fUML é interessante por ser uma linguagem gratuita de uma perspectiva de UML que também existe ferramentas gratuitas. Em alguns aspectos da linguagem Java estão atualmente a representação dos modelos como distribuição dinâmica, genéricos (para classes e interfaces), exceções e asserções, bibliotecas externas e aspectos de reflexão correspondentes. Esses conceitos atualmente não são suportados diretamente pelo fUML.

O fUML tem como objetivo ser estendido para fornecer um conjunto mais completo de conceitos que podem mapear mais linguagem de programação diretamente para fUML. É importante investigar sobre como o mapeamento entre anotações em nível de código e nível de modelo podem ser estendidas a fim de incorporar também aspectos comportamentais. A ideia é explorar perfis UML descobertos automaticamente fornecendo correspondência estereótipos de anotação. Também são incluídos comportamentos e aspectos em uma forma que eles são diretamente utilizáveis pela VM fUML para fins de execução. Além disso, a versão atual do framework fREX têm suporte para mais de uma linguagem.

**FED-CASE** (SADAF; ATHAR; AZAM,2016). A ferramenta FED-CASE buscou avaliar o uso de critérios e de números de medidas para sugerir ferramentas para o desenvolvimento automatizado de modelos UML. É possível através da ferramenta identificar alguns conceitos de engenharia reversa automática que geram modelo de diagrama de classes UML. O estudo mostra claramente que a automação do modelo conduzido no desenvolvimento é mais eficiente do que o desenvolvimento simples sem o uso do modelo. A ferramenta fornece uma maneira fácil e eficiente de modelar as especificações do sistema.

A ferramenta é criada na plataforma Visual Studio 2010 usando a programação com a linguagem Visual Basic. A tarefa de implementação do modelo é realizada através da interface com o usuário. O sistema converte os componentes do diagrama de classe em engenharia estrutural. O diagrama de classes será gerado através da codificação. As funções do modelo são convertidas como inicializações da ferramenta. O VB é uma linguagem usada por sua facilidade e eficiência. As classes e funções integradas adequadas para gerenciar gráficos são necessários para a visualização do modelo gerado. A tarefa incluída na implementação foi lidar com gráficos que é principalmente necessário para desenhar o modelo correto UML. A engenharia reversa também pode ser feita automaticamente.

O diagrama consta com ícones no painel vertical esquerdo representam os componentes do modelo de diagrama de classes que podem ser arrastados para serem utilizados no modelo. Desta forma, a ferramenta fornece uma maneira fácil e eficiente de modelar o sistema de software. Sempre que o usuário arrastar e soltar o ícone do diagrama de

classes irá aparecer uma janela para que seja inserido o nome do diagrama que está sendo criado. A seguir na Seção 3.3 será apresentada a análise do estado da prática.

### 3.3 Análise do estado da prática

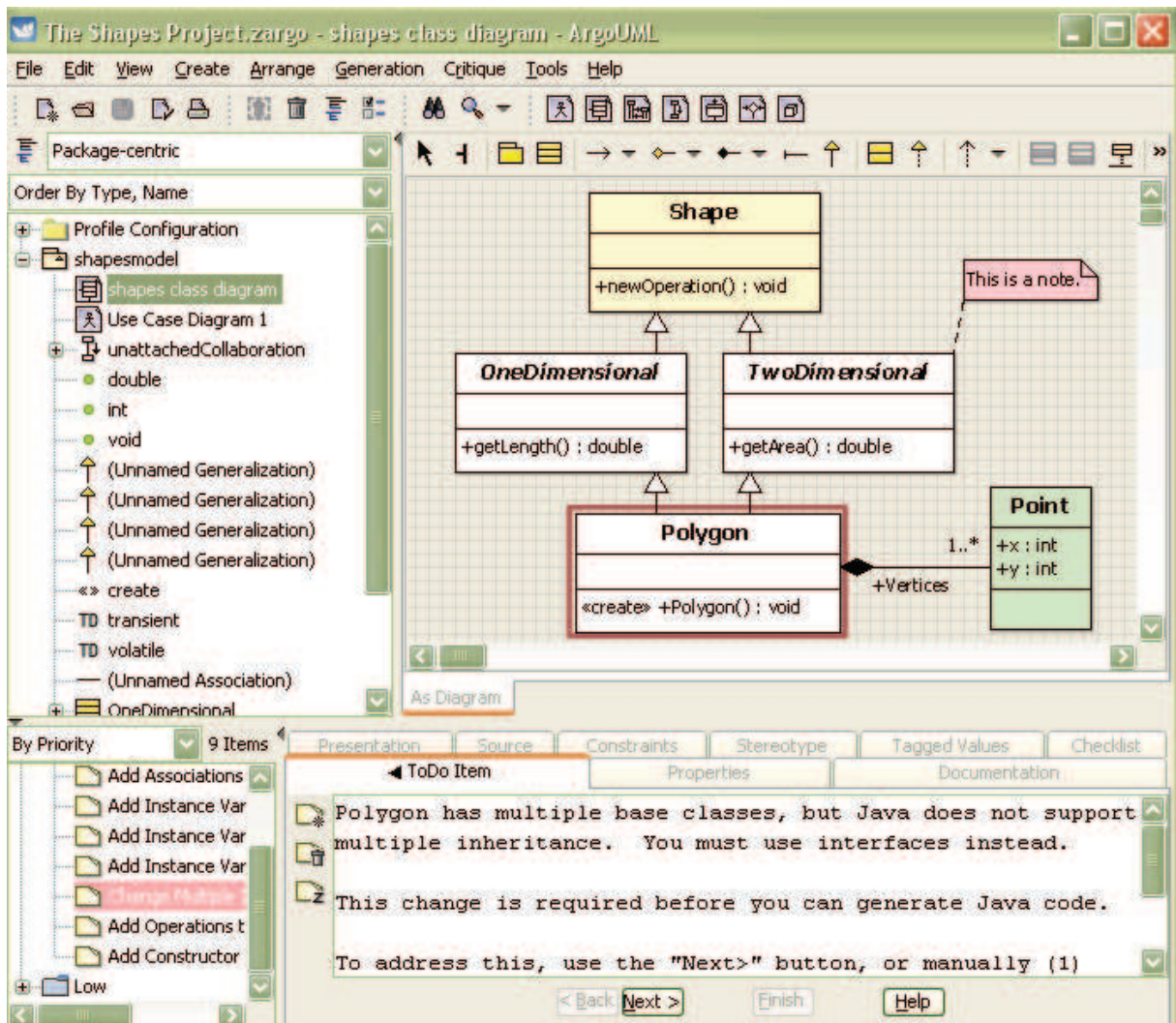
**ArgoUML** (ARGO UML, 2017). É uma ferramenta de modelagem UML. É disponibilizada em dez idiomas. A ferramenta ArgoUML foi concebida como uma ferramenta com ambiente para uso de análise e de *design* de sistemas de software orientados a objetos. É um software de modelagem UML gratuito e suporta os diagramas do padrão UML 1.4 como diagrama de classes, diagrama de caso de uso, diagrama de atividade, diagrama de sequência e diagrama de implantação. A ferramenta ArgoUML foi criada na plataforma Java. Através da ferramenta é possível a geração de arquivos XMI. É um formato de arquivo padrão para projetos UML que serão suportados por outra ferramenta, como a ferramenta SD *Metrics* (ALI; BANGOURA, 2015) que será utilizada como ferramenta de apoio para avaliação e análise deste trabalho.

De acordo com Vieira (2004), a ferramenta ArgoUML é uma das principais ferramentas de modelagem UML e inclui suporte para todos os seguintes diagramas da UML: Diagrama de classes, Diagrama de estados, Diagrama de Atividades, Diagrama de casos de uso, Diagramas de colaboração, Diagrama de Componentes e Diagrama de sequência. Segundo o mesmo autor, a ferramenta a que se refere possui uma interface completa, tornando-o complexo de o manipular. Contudo, ela é capaz de:

- desenhar e imprimir diagramas UML;
- declarações de classes Java;
- exportar documentação para páginas web em Java;
- gerar arquivos gráficos em formato .GIF;
- gerar comandos SQL com auxílio de outros softwares, etc.;

Um diferencial é a visualização de propriedade dos elementos UML quando selecionados, exibindo uma espécie de rastreabilidade entre elementos. A geração do diagrama de classes não é realizada de forma automática. Para visualizar os dados no diagrama basta adicionar um novo diagrama e arrastar as informações dos elementos. A Figura 14 apresenta a interface da ferramenta ArgoUML.

Figura 14 – Ferramenta ArgoUML



Fonte: ARGO UML, 2017.

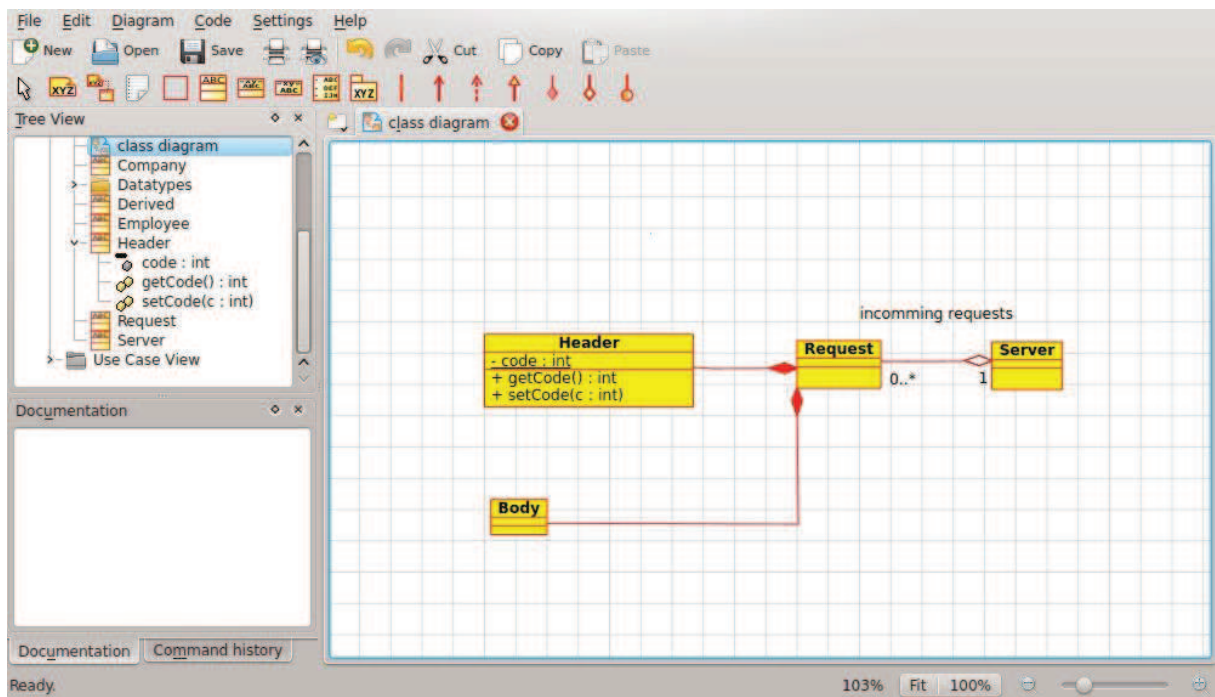
A ferramenta fornece geração de código para Java, C ++, C# e PHP. A geração do código Java com a engenharia reversa fornece arquivos e novos modelos a partir do código-fonte ou de diagrama já existente. O ArgoUML fornece uma estrutura modular de engenharia reversa. Atualmente, o código-fonte Java é fornecido por padrão e existem módulos para importação de arquivos de classe. Existe suporte muitas funções de edição de diagramas que ajudam a edição de diagramas UML (ARGO UML, 2017).

**Umbrello Uml Modeler** (UMBRELLO, 2016). É um software de modelagem desenvolvido por um grupo de programadores. A ferramenta gera código na plataforma Java. Além disso, também gera arquivos gráficos de imagens, realiza a engenharia reversa de classes, exporta arquivos no padrão XMI, dentre outras funcionalidades. A ferramenta não gera automaticamente diagramas UML a partir da importação do código-fonte. Para isso, deve-se arrastar os elementos para um novo diagrama, para que seja possível visualizar seus relacionamentos.



A Figura 15 ilustra a interface da ferramenta Umbrello UML Modeler.

**Figura 15 – A ferramenta Umbrello UML Modeler**



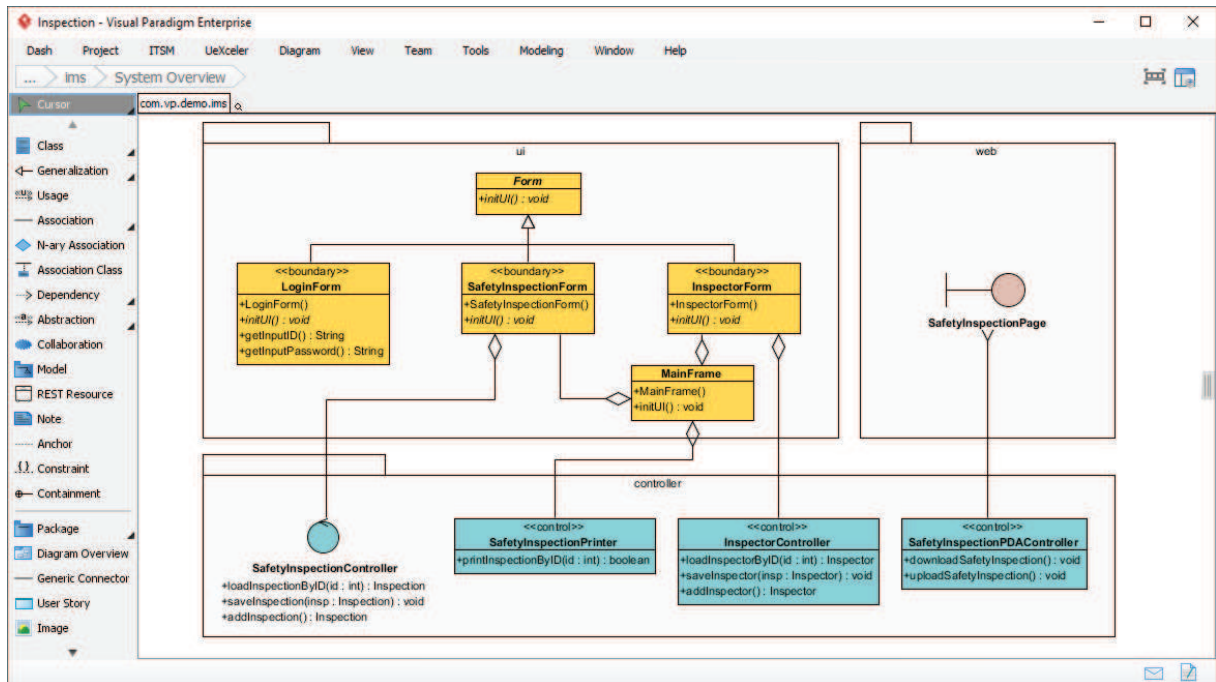
Fonte: Umbrello Uml Modeler, 2016.

O Umbrello UML Modeller ajuda a comunicar erros ou sugerir melhorias no software. Também corrige erros, adiciona funcionalidades e gera uma boa documentação possível ser gerada em outros idiomas. Umbrello UML Modeller pode copiar os objetos em formato de imagens como, por exemplo, em formato PNG. Com esta ferramenta é possível imprimir separadamente os diagramas gerados.

**Visual Paradigm** (VISUAL PARADIGM, 2016). O Visual Paradigm é uma ferramenta que suporta os diagramas definidos na UML 2.0. É uma ferramenta de modelagem de entidade e relacionamento de banco de dados e de modelagem de software. O Visual Paradigm é possível gerar diagramas de classes UML ou criar o código-fonte a partir de um modelo já existente. A ferramenta é capaz de gerar código-fonte Java que reflete o modelo de classes UML já existente. O modelo UML reflete as alterações que transforma em código-fonte. A engenharia reversa ajuda a manter o código-fonte Java e design de software sincronizadas.

Ao gerar o código ou atualizar o modelo UML, as alterações serão atualizadas no modelo ou no código respectivamente. Com a ferramenta também é possível capturar os requisitos funcionais de um caso de uso. O uso de um diagrama de caso de uso representa uma função do sistema de alto nível que produz um resultado mensurável para o sistema. Atores são relacionados com os casos de uso para representar os papéis que interagem com as funções (VISUAL PARADIGM, 2016). A Figura 16 ilustra a interface do usuário com a ferramenta Visual Paradigm.

Figura 16 – A ferramenta Visual Paradigm



Fonte: Visual Paradigm, 2016.

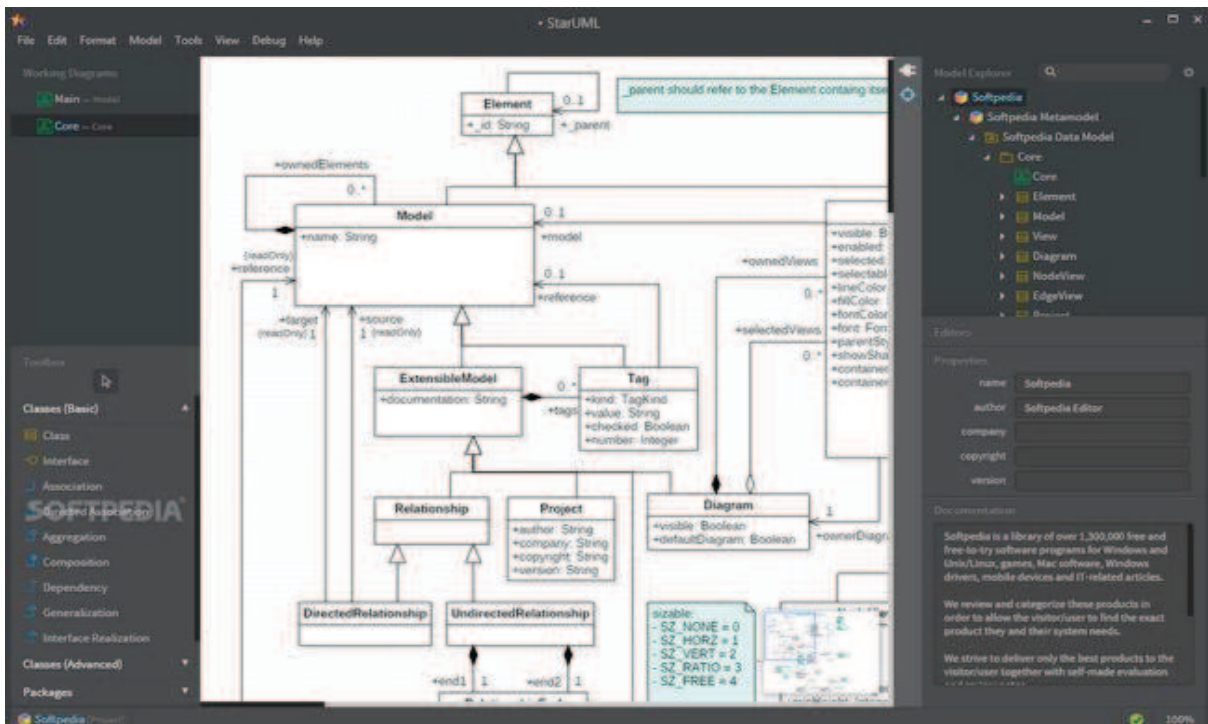
**StarUML** (StarUML, 2017). StarUML é uma ferramenta gratuita para desenvolvimento rápido, flexível, completo e disponível gratuitamente na plataforma do sistema operacional Windows. O StarUML é customizável e tem uma alta extensibilidade em suas funcionalidades. A ferramenta adere estritamente o padrão UML especificado pelo OMG para modelagem de software. O StarUML é baseado na UML versão 1.4 e fornece anotações UML na versão 2.0 em onze diferentes diagramas UML (ALI; BANGOURA, 2015).

O StarUML suporta linguagens de programação como o Java, C # e C ++. É possível gerar códigos-fonte dos modelos já criados anteriormente ou criar um modelo a partir do código-fonte através de engenharia reversa. As extensões que fornecem engenharia de código podem ser instaladas via uma ferramenta de extensão. Por ser uma ferramenta gratuita, é possível encontrar fontes no repositório Github (Java, C #, C ++). Abaixo é apresentado a interface da ferramenta StarUML. A ferramenta StarUML suporta os seguintes diagramas UML:

- diagrama de casos de uso,
- diagrama de classe,
- diagrama de sequência,
- diagrama de atividades.

A Figura 17 apresenta a ferramenta StarUML.

Figura 17 – A Ferramenta StarUML



Fonte: StarUML, 2017

Também suporta a maioria dos diagramas especificados na UML 2.0. O conjunto muito rico de recursos e opções de formatação fazem uma das principais ferramentas dentro da modelagem de software. É possível gerar código-fonte a partir do diagrama UML e vice-versa. A engenharia reversa de código-fonte existente em alguns dos diagramas UML. As linguagens suportadas pela ferramenta são: C, C# e Java. O tempo de execução da ferramenta é rápida em comparação com outras ferramentas UML. Suporta os principais diagramas e exporta imagens em formatos JPG.

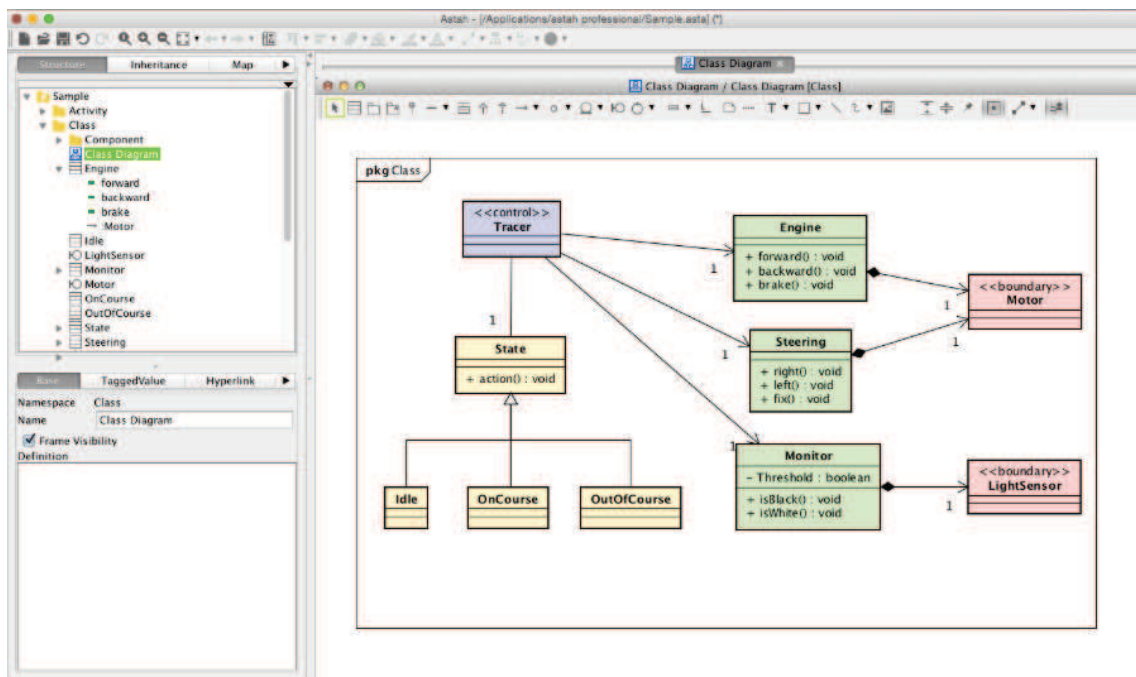
**ASTAH** (ASTAH, 2017). O Astah é uma ferramenta para criação de diagramas UML possuindo uma versão gratuita, o Astah Community. A versão gratuita que pode ser obtida no site da ferramenta com link para download da ferramenta. A ferramenta possui algumas restrições de funções que só estão liberadas para o usuário na versão vendida comercialmente. As funções disponíveis na versão gratuita são suficientes para o uso e criação de modelos UML. A ferramenta é utilizada para modelar os diagramas da UML e também gerar diagramas a partir do código. Anteriormente, o nome da ferramenta Astah era denominada de JUDE.

A ferramenta Astah é código livre e 100% desenvolvida em Java. Permite a geração de código em Java e que realiza a engenharia reversa em código Java. Também é possível realizar a exportação dos modelos em formato de imagens como PNG/JPEG. A ferramenta gera os códigos-fonte de seus modelos ou também é possível construir um modelo UML a partir do código-fonte com o uso de engenharia reversa. É compatível com o padrão UML 2.0 e suporta praticamente todos os diagramas definidos pela OMG. Características gerais, o editor de diagramas UML que incorpora outros recursos é modificada de acordo com a distribuição da ferramenta utilizada.

Abaixo são apresentados alguns dos benefícios do Astah:

- salva/exporta diagramas em formatos PNG/JPEG/EMF/SVG,
- importação e exportação de código-fonte Java;
- salva arquivos em formato XML/XMI;

**Figura 18 – Interface da ferramenta Astah**



Fonte: ASTAH, 2017

A Figura 18 ilustra a interface da ferramenta Astah. Um dos softwares mais conceituados para a criação de diagramas de UML é o Astah (ASTAH, 2017). Na versão grátis alguns dos recursos não estão liberados, mas mesmo assim, é possível utilizar muito bem as funcionalidades da ferramenta. O software é bem intuitivo e com apenas alguns minutos de exploração é possível criar um modelo UML na ferramenta. A Seção 3.4 irá apresentar os critérios de comparação que foram definidos para este trabalho e apresentar o comparativo entre os trabalhos relacionados detalhados até o momento.

### 3.4 Comparação dos trabalhos relacionados

Esta seção detalha os critérios de comparação entre os trabalhos relacionados. Os critérios foram definidos com o objetivo de classificar e identificar possíveis diferenças e pontos em comum entre as ferramentas apresentadas neste trabalho. Com isto, abaixo são apresentados os critérios que foram definidos para a comparação entre os trabalhos, são eles:

- **Engenharia reversa orientada a *features* (C1).** A engenharia reversa é o conceito central nas ferramentas relacionadas neste trabalho. O método da engenharia reversa realizado pelas ferramentas apresentadas é no formato tradicional, isto é, a engenharia reversa realiza a transformação de código-fonte em diagramas UML de todo o código-fonte do sistema. Nas ferramentas apresentadas no estado da prática neste trabalho, por exemplo, Umbrello, ArgoUML, Visual Paradigm, StarUML e Astah, a reversão de código é realizada da forma tradicional. Ferramentas como Visual Paradigm, Astah e ArgoUML são muito semelhantes na forma de criação e geração do diagrama UML. Todas ferramentas citadas buscam usar a engenharia reversa como forma de automatizar o processo de criação de diagramas.

A engenharia reversa orientada a *features* é realizada a captura de informações e métodos a partir de uma funcionalidade do sistema. Estas funcionalidades implementadas no código-fonte, fazem com que a captura destas informações possa gerar um novo modelo UML a partir do código capturado. Com as informações armazenadas, o processo de geração do modelo é realizado automaticamente pela ferramenta. As ferramentas não disponibilizam a geração de modelos orientados a *features*. Este critério é importante para este trabalho pois ilustra a limitação das ferramentas estudadas no estado da prática.

- **Mecanismo de geração de modelos automática (C2).** Este critério é definido pela automatização da geração dos modelos. Algumas ferramentas disponibilizam a geração do modelo de forma automática de todo o sistema, gerando modelos de todo o projeto. Porém, nem todas as ferramentas são automáticas. Algumas é necessário que o usuário realize manualmente a seleção do arquivo ou até mesmo adicionar o arquivo na ferramenta para que seja realizada a engenharia reversa do código-fonte. No mecanismo de geração dos modelos com a engenharia reversa, as ferramentas buscam automatizar este processo facilitando para o usuário não precisar importar o arquivo.

Com isto, na forma automática não é necessária a interação do usuário para que a ferramenta realize a criação de um modelo UML. Todas ferramentas selecionadas neste trabalho suportam a geração de modelos UML. Para a avaliação dos critérios utilizados do nível de automação do mecanismo das ferramentas, não foram classificadas como totalmente automática. A automação das ferramentas foi parcialmente atendida em algumas ferramentas que foram possíveis realizar a engenharia reversa.

- **Notação UML 2.0 (C3).** Neste critério serão analisadas as ferramentas que são capazes de atender a notação UML 2.0. Caso a ferramenta não suporte à notação UML 2.0 poderá atender parcialmente com funcionalidades da UML 1.X. Caso não o modelo não tenha padrão de notação UML, será classificado como não atende a notação UML.
- **Criação de modelos de classes (C4).** O quarto critério é relacionado aos modelos que a ferramenta disponibiliza. Este critério será atendido pela ferramenta quando a mesma disponibilizar a geração de modelos de classes. A ferramenta que suportar a geração de diagramas de classes será classificada como atende este critério. Este critério foi incluído devido ao modelo de classes ser um dos principais modelos utilizado dentro do processo de

desenvolvimento do software. O diagrama de classes é o modelo mais utilizado no desenvolvimento e na manutenção do software.

- **Engenharia reversa de código Java (C5).** Realizar engenharia reversa a partir do código-fonte na linguagem Java. A geração de modelos UML pode ser gerada a partir de qualquer linguagem. Este critério foi utilizado para identificar as ferramentas que geram os modelos na plataforma Java. Caso a ferramenta crie os modelos a partir de outra linguagem, mas suporte a plataforma Java é classificada como atende parcialmente.
- **Uso de aspectos (*trace*) (C6).** O sexto critério é referente ao uso de programação orientada a aspectos na execução da ferramenta. Algumas ferramentas podem utilizar o uso de *traces* no processo de geração dos modelos UML. A ferramenta pode utilizar qualquer algoritmo de execução de aspectos no processo de geração dos modelos.

Após os critérios definidos e detalhados, abaixo é apresentada a tabela comparativa com os trabalhos pesquisados do estado da arte e do estado da prática. Os critérios são destacados na primeira linha da tabela com a identificação de cada critério. Cada critério está apresentado na ordem em que foi detalhado acima. A letra ‘C’ seguido do número de cada critério criado, por exemplo, o critério 1 é denominado de C1. Confira a seguir a Tabela 1 que apresenta os trabalhos relacionados.

**Tabela 1 – Tabela de comparação entre trabalhos relacionados**

	C1	C2	C3	C4	C5	C6
<b>TechREF</b>	+	+	+	+	+	+
ArgoUML (ArgoUML,2017)	-	+	+	+	+	-
Astah (ASTAH, 2017)	-	+	+	+	+	-
FED-CASE (SADAF; ATHAR; AZAM,2016)	-	~	~	+	-	-
ForUML. (NANTHAAMORNPHONG et al., 2015)	-	~	-	+	-	-
fRex (BERGMAYR et al., 2016)	-	~	-	+	-	-
MoDisco (BRUNELIERE et al., 2014)	-	+	~	+	-	-
PHP2XMI (ALALFI; CORDY; DEAN, 2009)	-	~	-	-	-	-
ScrYUML. (DECKER et al.,2016)	-	+	-	+	-	+
SequenceTools (BAIDADA; JAKIMI, 2016)	-	+	~	-	~	+
StarUML (StarUML, 2017)	-	+	+	+	+	-
Umbrello Uml Modeler. (UMBRELLO, 2016)	-	+	+	+	+	-
Visual Paradigm (VISUAL PARADIGN,2016)	-	+	+	+	+	-

Legenda:  + Atende,  ~ Atende parcialmente,  - Não atende

Fonte: Elaborado pelo autor.

Após a apresentação da tabela comparativa é possível observar as lacunas existentes na comparação entre os trabalhos e ferramentas avaliadas. Em todas as ferramentas que foram

estudadas, em nenhuma obtém informações sobre possíveis casos de melhora na execução do método de engenharia reversa orientada a *features* e não há nenhuma forma de modificar este formato que é gerado os modelos UML.

Os experimentos mostram que a discussão do uso de modelos UML dentro do processo de desenvolvimento de software é ampla. Uma grande questão envolve os artigos realizados com experimentos controlados com o uso de modelos UML no processo de desenvolvimento do software. O custo e o grau de esforço para atualização dos modelos são um grande entrave para que a UML seja vista e utilizada larga escala. Por isto, existe uma grande oportunidade de pesquisa neste cenário, afim de observar novos resultados e apresentar um novo método de realizar engenharia reversa automática capaz de ajudar a compreender o uso de modelos UML dentro do processo de desenvolvimento e manutenção de software.

### 3.5 Oportunidades de pesquisa

Através da comparação dos trabalhos relacionados apresentada na Tabela 1, foi possível avaliar que existem algumas lacunas para novas discussões e elaborações de novos trabalhos empíricos. Estas lacunas serão detalhadas a seguir como oportunidades de pesquisa e serão exploradas como forma de contribuição para este tema. De acordo com a análise da tabela comparativa acima apresentada, serão explorados os seguintes tópicos:

- **Engenharia reversa orientada a *features*:** A técnica TechREF traz o conceito de geração de diagramas UML com a proposta de reverter o código apenas a partir das funcionalidades do sistema. Como este processo é diferenciado dos demais trabalhos que foram apresentados na comparação dos trabalhos relacionados, abre-se uma oportunidade de pesquisa para este assunto. Conforme observado na Tabela 1, as ferramentas relacionadas na comparação, somente a técnica TechREF suporta a engenharia reversa orientada a *features*. As demais ferramentas suportam apenas a engenharia reversa sem *features*, ou seja, na forma tradicional. Com base nas informações da Tabela 1, aproximadamente 84% das ferramentas, isto é, das seis ferramentas analisadas, cinco delas não suportam engenharia reversa orientada a *features*. Com esta análise, verifica-se a falta de ferramentas orientadas a *features*. Para apresentar este conceito de engenharia reversa orientada a *features*, no próximo Capítulo 4, será apresentado os processos de implementação da técnica e os aspectos de implementação e algoritmos desenvolvidos.
- **Automatização das ferramentas:** Um dos critérios de avaliação aplicados na relação dos trabalhos conforme é apresentado na Tabela 1, é a forma de automatização do processo de engenharia reversa nas ferramentas. O processo de execução de geração de um diagrama UML nas ferramentas é avaliado de três maneiras, são elas: Manual, Semiautomática e Automática. Das ferramentas analisadas, quatro ferramentas são classificadas nesta avaliação como semiautomáticas, ou seja, aproximadamente 67% das ferramentas geram diagramas UML de forma semiautomática. As demais suportam parcialmente a geração dos diagramas UML de forma semiautomática. Para o critério de geração de forma automática somente uma ferramenta é capaz de gerar de forma automática os modelos UML. Desta forma, a análise da Tabela 1 ilustra

a falha na automatização das ferramentas em não realizar geração automática de diagramas UML. Para verificar isto, é possível observar na Tabela 1, que apenas uma das ferramentas é automática. Este percentual é de 16% das ferramentas, ou seja, 5/6 das ferramentas não são automáticas, isto identifica que as ferramentas não são automáticas na geração de diagramas UML.

- **Produção de conhecimento empírico:** Os experimentos contribuem com novas técnicas e aumentam a discussão do uso da UML. Novas ideias podem ajudar a ampliar esta área de pesquisa. Os trabalhos apresentados na Tabela 1 mostram soluções criadas com técnicas que ajudam e aumentam as discussões sobre o uso de modelos UML dentro do processo de desenvolvimento de software. Com isto, é interessante que novas soluções e novas ideias possam surgir para fomentar ainda mais a discussão do uso da modelagem do software.

No Capítulo 4 será apresentado a solução de engenharia reversa orientada a *features* criada neste trabalho.



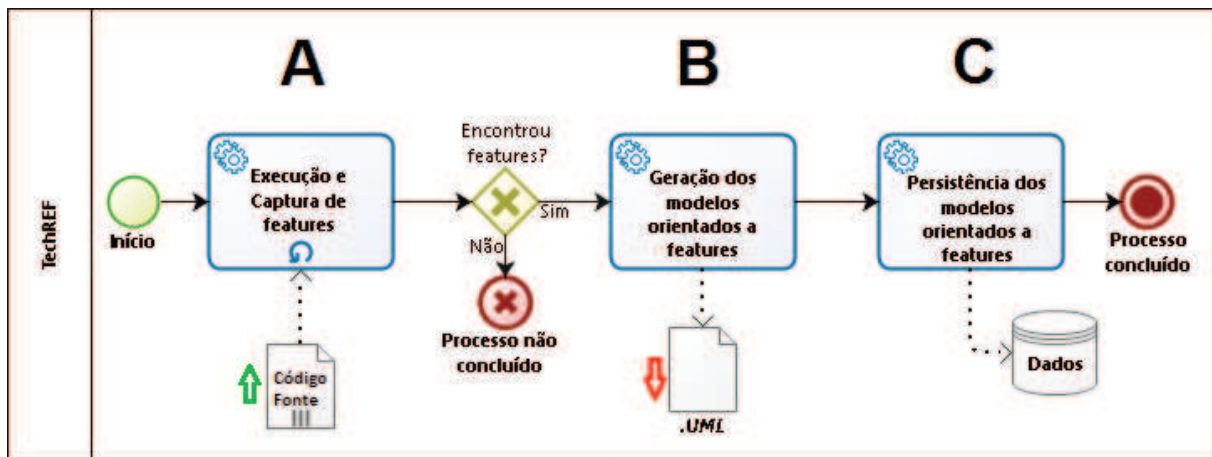
## 4 TÉCNICA DE ENGENHARIA REVERSA

Neste Capítulo será apresentada a técnica TechREF e sua arquitetura. Na Seção 4.1, uma visão geral da técnica e seus processos elaborados são apresentados. Na Seção 4.2, o processo de engenharia da técnica é apresentado. Por fim, na Seção 4.3, são apresentados os aspectos de implementação da técnica.

### 4.1 Visão geral da técnica

A nomenclatura criada para a técnica foi de TechREF. Essa ideia foi criada por causa do termo Técnica em inglês que é ‘Tech’, seguido das iniciais de *Reverse Engineering oriented Features* que é ‘REF’. Com isto, a união das duas palavras Tech e REF originou-se o nome TechREF. O grande diferencial da TechREF é o uso da engenharia reversa com o conceito de orientação a *features*. Esta nova forma de identificar as funcionalidades do sistema caracteriza um diferencial na técnica, ou seja, permite que o usuário elabore novos modelos a partir de funcionalidades implementadas no sistema. A técnica busca transformar a informação das funcionalidades do sistema em modelos UML, criando, assim, novos diagramas e melhorando o entendimento para quem utiliza o sistema. A Figura 19 ilustra o fluxo do processo de execução da TechREF.

Figura 19 – Processo de execução da técnica TechREF



Fonte: elaborado pelo autor.

Os processos da técnica TechREF foram divididos em 3 cenários. Cada processo é referente a um processo de execução da técnica. Abaixo serão explicados os processos que compõem a técnica TechREF conforme apresentado na Figura 19. Os processos da TechREF são:

- **Execução e Captura das *features*.** A execução e captura dos pontos de *features* é realizada na primeira etapa do processo de execução da técnica TechREF. Este processo é realizado para capturar a informação das *features* do sistema. Ao inicializar a TechREF este processo inicial da técnica é acionado, onde é realizada a coleta das informações das funcionalidades que o usuário deseja criar o novo modelo orientado a *features*.

- **Geração dos modelos de *features*.** A segunda etapa do processo da TechREF é a geração dos modelos orientados a *features*. Posteriormente ao armazenamento das informações que foram capturadas no primeiro processo, este segundo processo será realizar a geração do novo modelo. Esse modelo irá consumir as informações que foram coletadas na primeira etapa do processo da técnica.
- **Persistência dos modelos de *features*.** A última etapa do processo da técnica realiza a persistência do modelo que foi gerado na segunda etapa. Nesta etapa é possível escolher o nome do arquivo para persistir o modelo em disco. A persistência do modelo é através da notação UML. Esta terceira etapa é específica para armazenar os novos modelos UML gerados pela técnica TechREF.

#### 4.1.1 Execução e captura de *features*

O processo de execução e captura de *features* é o primeiro processo realizado pela técnica TechREF. Conforme ilustrado na Figura 19, a letra “A” no diagrama representa a primeira etapa do processo. Esta primeira etapa captura a *feature* a partir do código do sistema. Para que seja capturado o código é necessário que a inicialização de *pointcut* seja criada através da programação orientada a aspectos. Esses *pointcuts* são identificados através da inicialização da técnica neste primeiro processo.

Após a inicialização da técnica, será realizada o processo de armazenamento das informações que foram capturadas. O processo de captura é realizado através da execução do usuário no sistema. Ao utilizar o sistema após a inicialização da TechREF, as informações das capturadas são incluídas em arquivos em formato .xml para que posteriormente em outro processo possa ser utilizada. Quando uma nova funcionalidade é identificada pela técnica é realizada uma nova captura das informações referente a funcionalidade do sistema. Com isto, é armazenada uma nova informação em um arquivo de formato .xml, assim é possível que seja gerado modelos de *features* conforme será apresentado na Seção 4.1.2 com os requisitos do processo.

Para exemplificar o funcionamento de execução e captura das *features*, na Figura 20 abaixo é apresentado dois trechos de códigos, onde é representada uma funcionalidade do sistema. Em destaque na cor azul na imagem, é possível observar as duas classes com dois métodos sendo utilizados nesta *feature*. Com isto, para exemplificar esta captura, foi incluído um método A1 na classe A e um método B1 na classe B. No método da classe A1 foi implementada uma chamada para um método da classe B. Na Figura 20 é possível observar que o método A1 instância um método da classe B.

**Figura 20 – Trecho de código de uma *feature***

<code>class A {</code>	<code>class B {</code>
<code>  metodoA1() {</code> <code>    B.metodoB1();</code> <code>  }</code>	<code>  metodoB1() {</code> <code>    ...</code> <code>  }</code>
<code>}</code>	<code>}</code>

Legenda:

Feature A

Fonte: elaborado pelo autor.

Para exemplificar o processo de desenvolvimento de software, será criada uma nova *feature* no sistema criado como exemplo, para a inclusão de novos métodos na classe A e na classe B que já existem, a nova classe C conforme ilustrada na Figura 21 faz parte da nova funcionalidade. Esta nova funcionalidade irá refletir como é o desenvolvimento de novas funcionalidades dentro do processo de desenvolvimento do software. As novas funcionalidades são implementadas dentro do processo de desenvolvimento de software, ocorrendo manutenções nos sistemas já em andamento, isto é, há novas implementações e manutenção de software sendo criadas juntamente como o andamento do projeto de software. As novas *features* que foram implementadas nas classes estão destacadas em tom verde conforme ilustra a Figura 21.

Conforme é apresentado na imagem abaixo, a nova funcionalidade na classe A, em destaque na cor verde, é o novo método A2 que foi adicionado posteriormente comparado com a Figura 20 na classe A. Continuando a implementação, na classe B, foi adicionado um novo método B2, que não existia na classe B. Finalizando, a classe C foi implementada com um novo método C1 que é instanciado através da chamada de execução do método B2 da classe B, e que por sua vez é instanciado dentro do método A2 da classe A. Abaixo em tom verde a nova implementação realizada que envolve a segunda *feature*.

**Figura 21 – Trecho de código de *features***

<code>class A {</code>	<code>class B {</code>	<code>class C {</code>
<code>  metodoA1() {</code> <code>    B.metodoB1();</code> <code>  }</code>	<code>  metodoB1() {</code> <code>    ...</code> <code>  }</code>	<code>  metodoC1() {</code> <code>    ...</code> <code>  }</code>
<code>  metodoA2() {</code> <code>    B.metodoB2();</code> <code>  }</code>	<code>  metodoB2() {</code> <code>    C.metodoC1()</code> <code>  }</code>	<code>}</code>
<code>}</code>	<code>}</code>	

Legenda:

Feature A     Feature B

Fonte: elaborado pelo autor.

De acordo com os exemplos das *features* que foram implementadas acima nas ilustrações das Figuras 20 e 21, o processo de identificação e captura das *features* no código é realizado através do mecanismo de identificação das classes que participam da funcionalidade do sistema. Esta identificação é realizada através de programação orientada a aspectos. Abaixo um exemplo do algoritmo que captura as funcionalidades do código de acordo com as

assinaturas das classes e operações. O Algoritmo 1 realiza a captura das classes através da execução do sistema e armazena as informações capturadas em formato XML. Com isto, as informações das classes envolvidas no primeiro processo da técnica são armazenadas para que no segundo processo da técnica seja possível a geração dos modelos orientados a *features*.

---

**Algoritmo 1** – Captura de *features*

---

**Entrada:** Classe acionada pelo sistema.

---

1. **DocumentoXML** xml;
  2. objeto = aspecto.ObterAssinaturas();
  3. classe = objeto.NomeClasse();
  4. **Se** classe.ObterNome() <> “ “ **Então**
  5.     xml = IniciaTemplateXML();
  6.     xmlClasse = xmlClasse + classe.ObterNome();
  7.     objetoMetodo = classe.ObterMetodos();
  8.     **Enquanto** existe objetoMetodo **Faça**
  9.         xmlMetodo = xmlMetodo + objetoMetodo.NomeMetodo();
  10.     **Fim Enquanto**
  11.     objetoAtributo = classe.ObterAtributos();
  12.     **Enquanto** existe objetoAtributo **Faça**
  13.         xmlAtributo = xmlAtributo + objetoAtributo.NomeAtributo();
  14.     **Fim Enquanto**
  15.     xml = FechaTemplateXML();
  16. **Fim Se**
- 

O Algoritmo 1 conforme é apresentado acima, descreve o processo de captura de uma funcionalidade do sistema, ou seja, enquanto estiver sendo executada a técnica, o algoritmo irá identificar as informações das classes através da programação orientada a aspectos. Nesta captura, as informações das *features* executadas são armazenadas em arquivos XML. As informações dos métodos são armazenadas bem como os atributos das classes. Assim que é inicializada a técnica, as capturas das *features* do projeto são armazenadas até o momento em que o usuário finaliza a captura. Para finalizar a captura das informações, o usuário deverá parar a captura. Após a conclusão da captura pelo usuário, o sistema dispara automático o processo seguinte da técnica. No item 4.1.2 será apresentado um detalhamento de como são gerados os modelos de *features* que foram capturados na primeira etapa do processo da técnica.

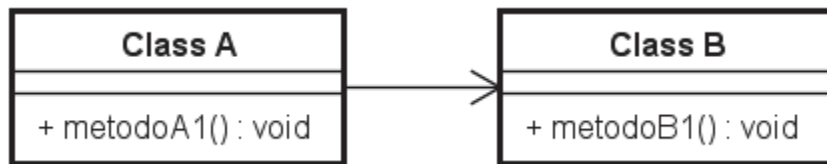
#### 4.1.2 Geração dos modelos orientados a *features*

O segundo processo da técnica é representado com a letra “B” na Figura 19 deste trabalho. Este processo é denominado de geração dos modelos orientados a *features*. Esta

etapa utiliza os dados armazenados na etapa anterior de captura das funcionalidades. Se o processo de execução armazenou alguma funcionalidade, então este processo de geração obtém essas informações e posteriormente gera o modelo UML de acordo com as *features* que foram encontradas. O novo modelo gerado terá as informações referentes às *features* que o usuário capturou.

Após o primeiro processo ser acionado da técnica TechREF, o segundo processo de geração do modelo irá ser inicializado, isto é, de acordo com as *features* armazenadas no primeiro processo, a técnica cria um modelo UML com as informações que foram capturadas. O novo modelo gerado irá representar a funcionalidade que foi capturada pelo sistema. Esse novo modelo gerado pela técnica TechREF será um modelo atualizado contendo todas as informações do sistema que foi capturado. A Figura 22 apresenta um exemplo de diagrama de classes orientado a *features* com as classes apresentadas na Figura 20.

**Figura 22 – Modelo UML gerado de uma *feature***



Fonte: elaborado pelo autor.

No diagrama da Figura 22, é possível identificar as informações que foram capturadas da *feature*. O sistema capturou as classes A e B e os métodos A1 e B1. A relação entre as classes A e B criou uma associação entre as duas, gerando o modelo UML de classes ilustrado na Figura 22. Com isto é possível criar modelos UML a partir de informações capturadas do código-fonte. Para que o novo modelo seja gerado com sucesso, todas as informações das funcionalidades do sistema precisam ser capturadas. O Algoritmo 2 apresenta a geração dos modelos UML.

---

**Algoritmo 2 – Geração do modelo UML orientado a *features***

---

**Entrada:** XML com informações das classes.

**Saída:** Modelo UML.

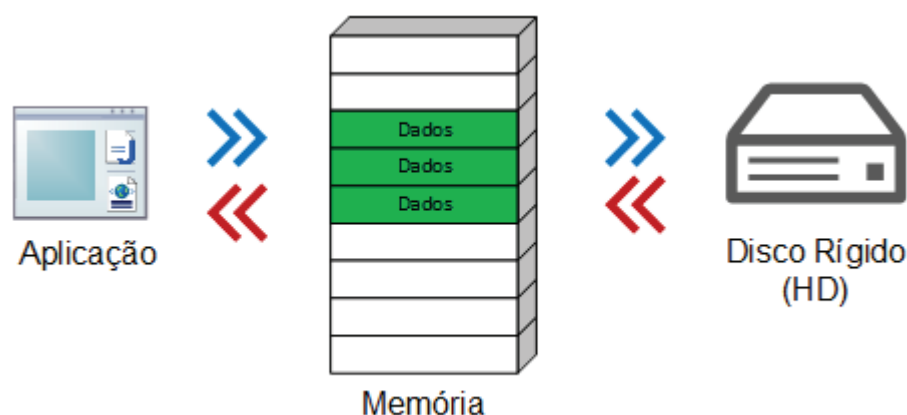
1. ModeloUML modelo = CriaModeloUML("Model");
  2. listaXML = Documento.abrirXML("CapturaClasse.xml");
  3. **Enquanto** listaXML não for o fim **Faça**
  4.   elemento = listaXML.ObterDados();
  5.   **Se** elemento.ObterNome() <> "" **Então**
  6.     modelo = CriaClasse(elemento.ObterNome());
  7.   **Fim Se**
  8. **Fim Enquanto**
  9. **Retorna** modelo.Salvar("Modelo.uml");
-

Quando a geração dos modelos de *features* é finalizada o segundo processo está concluído. Após a finalização deste processo o usuário realiza a última etapa denominada de persistência dos modelos orientados a *features*. Este processo é realizado na terceira etapa do processo que foi dividido em 3 etapas conforme apresentado pela Figura 19.

#### 4.1.3 Persistência dos modelos orientados a *features*

Na etapa anterior, foi realizado a geração do modelo orientado a *features*. Após a geração do modelo, é realizada a última etapa do processo da técnica TechREF que é denominado de persistência dos modelos de *features*. Esta etapa caracteriza por salvar o arquivo em disco. Até a geração do novo modelo, as informações ficam armazenadas em memória. Essas informações podem ser perdidas quando a técnica foi parada. Com o objetivo de persistir os dados, sempre que a técnica for finalizada, é informado ao usuário sobre a parada de execução e posteriormente os dados são salvos. Por padrão, o sistema operacional Windows armazena em cache dados de arquivos que são lidos de discos e gravados em discos. As operações de gravação escrevem dados de arquivo no cache de arquivos do sistema em vez de no disco, e esse tipo de cache é referido como um cache de *write-back*. O cache é gerenciado por objeto de arquivo (MSDN, 2017).

**Figura 23 – Processo de gravação em disco**



Fonte: elaborado pelo autor.

Conforme ilustrado na Figura 23, o fluxo de gravação dos novos modelos de *features* envolve os processos de inicialização e finalização da técnica, que envolvem o sistema de cache em memória do Windows e a gravação do modelo em disco. Este fluxo representa o que ocorre na técnica TechREF, onde após a captura das funcionalidades, das informações que são alocadas em memória para geração dos modelos e posteriormente as informações salvas em disco. Com o arquivo armazenado em disco, é possível a execução e visualização do modelo em aplicação que tenha compatibilidade com o formato UML. No tópico a seguir, será apresentado o processo de engenharia reversa orientado a *features*.

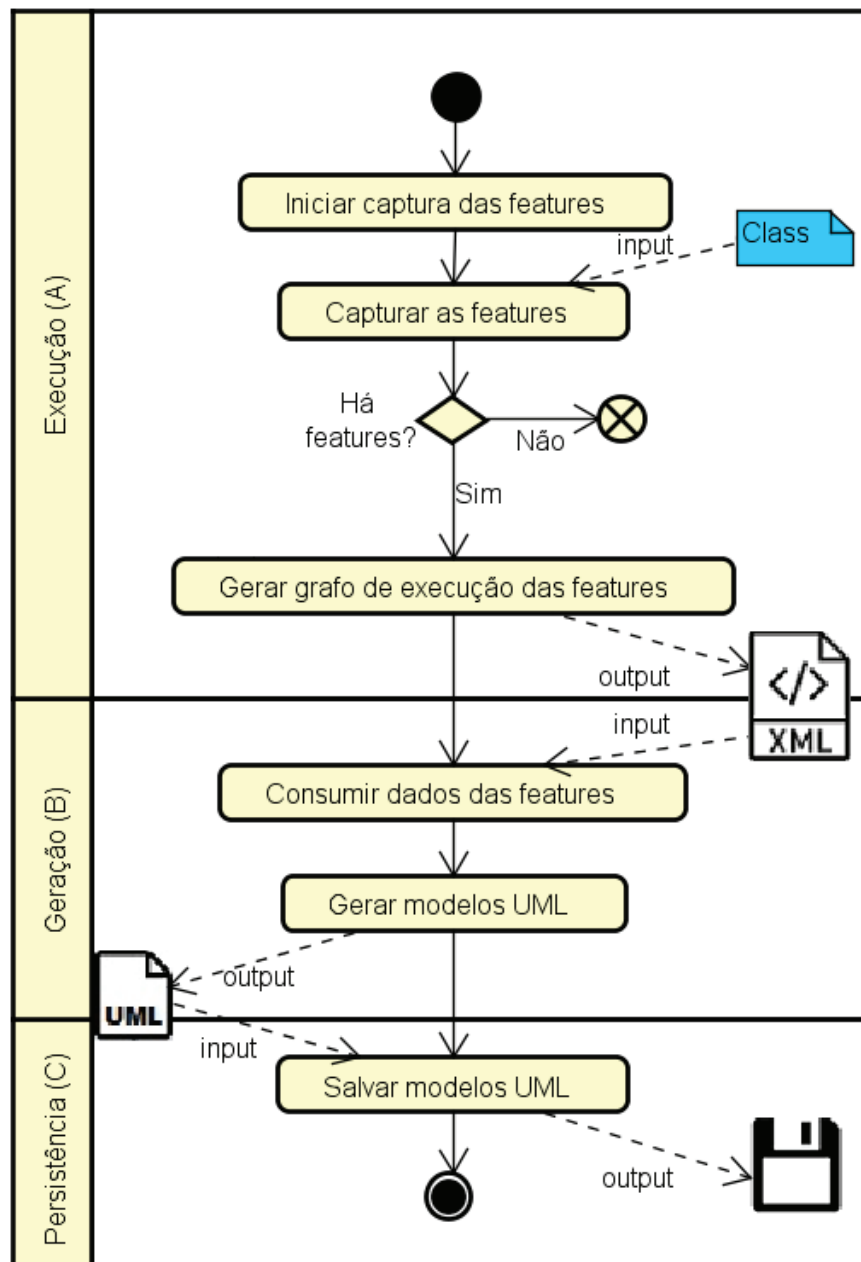
## 4.2 Processo de engenharia reversa orientado a *features*

Os processos da técnica TechREF estão vinculadas as *features*. Essas funcionalidades do sistema serão capturadas através dos aspectos implementados na técnica. Ao inicializar a técnica, são identificados os pontos de captura como classes e métodos. Essas informações irão ser consumidas no processo de geração de modelos, gerando novos modelos de classes. Também são armazenadas informações das relações entre as classes e seus métodos. Como forma de ilustrar o fluxo de atividade da técnica TechREF, será apresentado abaixo um digrama de atividade contendo os passos de atividades na geração de um novo modelo orientado a *features*.

O diagrama de atividades ilustrado na Figura 24 apresenta o fluxo de atividades da técnica apresentando o comportamento de execução da TechREF. Na primeira partição do diagrama de atividades, denominado de inicialização, é elaborada a atividade inicial da técnica ou a inicialização da técnica. Posteriormente na segunda partição do diagrama, denominada execução da técnica, é onde suas atividades de captura são realizadas. Neste segundo fluxo são encontradas as novas tarefas que foram criadas e o processo de captura dessas *features*. Caso exista alguma captura de uma funcionalidade, a execução passa para atividade de armazenamento dessas informações. Caso não exista nenhuma funcionalidade, a execução chega na etapa de finalização sem armazenar nenhuma informação.


Na terceira parte do diagrama de atividades, é realizada a geração dos modelos de *features*. Estas atividades são referentes as informações armazenadas e geração dos arquivos no formato XML e UML para que na última etapa os modelos sejam persistidos. Por fim na última partição do diagrama de atividades, é denominado o processo de persistência, onde o processo da técnica é finalizado e com a persistência em formato UML dos arquivos que foram gerados.

Figura 24 – Diagrama de atividades da TechREF




Legenda:

 arquivo .xml

 código fonte

 arquivo .uml

 salvar arquivo

Fonte: elaborado pelo autor.

O diagrama de atividades acima ilustra as atividades do processo de execução da técnica TechREF. Algumas das funcionalidades da TechREF, por exemplo, o armazenamento de informações das *features* e a geração do diagrama UML orientado a *features* podem ser identificadas no diagrama conforme ilustrado na imagem acima. Essas são algumas das



atividades da técnica TechREF, que foram apresentadas. A Seção 4.3 descreve os aspectos de implementação da técnica de engenharia reversa orientada a *features*.

### 4.3 Aspectos de implementação

A implementação da TechREF foi realizada na linguagem Java. Esta definição teve uma forte tendência devido ser uma linguagem de código livre, possibilitando a realização também da execução de aplicações/projetos de códigos livres. A linguagem Java também é uma das linguagens mais utilizada no mundo, por apresentar bibliotecas, novos complementos e projetos abertos para o consumo de informações. Com isto, foi escolhida esta linguagem para a implementação da TechREF.

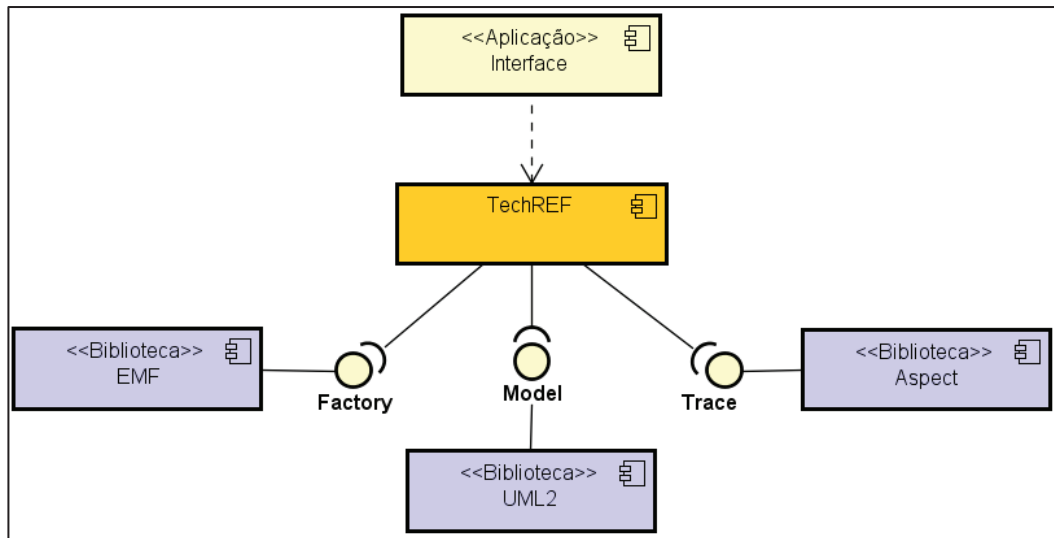
Para ajudar no desenvolvimento de códigos e funcionalidades da técnica, a linguagem Java disponibiliza um acervo de bibliotecas que ajudam e auxiliam na implementação de funcionalidades e customização do código. Com isto, é possível usar bibliotecas de apoio na implementação da técnica. As bibliotecas são chamadas de plug-ins. Algumas bibliotecas já disponibilizam funcionalidades que ajudam na implementação e contribuem para o tempo de desenvolvimento. A implementação da TechREF utilizou bibliotecas que auxiliaram e ajudaram a desenvolver a solução da técnica. Um plug-in que foi utilizado no desenvolvimento da TechREF é o *Eclipse Modeling Framework* (EMF, 2017). Este framework disponibiliza componentes que facilitam a criação de modelos UML.

Esta funcionalidade é utilizada a partir de códigos já existentes de bibliotecas que podem ser adicionadas no projeto. As informações necessárias para que seja gerado um novo modelo UML são capturadas através das classes e métodos. Os modelos serão criados através de códigos pré-definidos que geram um esboço de metamodelo UML. O formato do arquivo que é gerado pela biblioteca EMF é no formato *Unified Modeling Language* (UML). A Seção 4.3.1 apresenta as informações sobre a arquitetura da técnica.

#### 4.3.1 Arquitetura

A arquitetura da técnica é apresentada através do diagrama de componentes da TechREF. Este diagrama ilustra os módulos independentes que fazem parte da técnica. Cada módulo contém elementos importantes para o funcionamento da técnica. O diagrama de componentes ilustra um conjunto de elementos importantes para o processo de engenharia reversa orientado a *features*. Com o uso dos componentes apresentado na Figura 25 foi possível criar modelos orientados a *features*. Os componentes utilizados na implementação da TechREF foram EMF (EMF, 2017), UML2 (UML2, 2017) e AspectJ (The AspectJ Team, 2017). Esses componentes foram essenciais para o desenvolvimento da técnica.

**Figura 25 – Diagrama de componentes da TechREF**



Fonte: elaborado pelo autor.

O módulo TechREF possui o principal comportamento e a principal funcionalidade sobre os demais módulos da técnica. Ele é responsável pelo núcleo de atividades da técnica, ou seja, este módulo representa a implementação da técnica de engenharia reversa orientada a *features*. É este módulo o responsável por realizar todo o processo de tratamento das informações que são utilizadas para a geração de um modelo UML. Os componentes UML2 e EMF, são módulos responsáveis pela elaboração dos modelos e pela forma gráfica de representar o padrão de geração de modelos UML. As funções de visualização dos modelos e dos frameworks envolvidos na captura das *features* para a consistência de informações dos diagramas válidos é realizada através de módulos UML2 e EMF.

Para que a geração dos modelos orientados a *features* de forma automática, existe um componente responsável importante denominado de AspectJ. Este componente utiliza programação orientada a aspectos (POA) para capturar os elementos necessários para a geração de modelos orientados a *features*. O início da captura é realizado através da programação orientada a aspectos, a qual é acionada através do usuário que utiliza a TechREF. Por isto, a ação do módulo de *Interface* da técnica requer a inicialização da aplicação. Desta maneira a ação de captura dos elementos só é realizada quando a técnica é inicializada. Ao concluir a técnica, o usuário deve para a captura das *features* para que seja possível gerar os modelos orientados a *features*. A Seção 4.3.2 irá apresentar o detalhamento da implementação dos componentes da TechREF.

#### 4.3.2 Implementação dos componentes

O conceito de programação orientada a aspectos foi usado durante o desenvolvimento da TechREF, como mencionado anteriormente. Quando a técnica é inicializada, as informações das classes são capturadas através de métodos que foram implementados na programação orientada a aspectos. Esse conceito de programação orientada a aspectos permite que sejam configurados os pontos de execução, podendo assim obter informações dos métodos e dos atributos das classes envolvidas nas *features* do sistema. Com isto, é possível obter informações das classes, métodos e atributos dos projetos em que está sendo utilizada a TechREF.

A Figura 26 apresenta o código de programação orientada a aspectos. Este código apresenta a captura das informações das classes. Essas informações capturadas são armazenadas em arquivos XML. Assim que for identificada uma chamada de uma classe do sistema, as informações são coletadas e armazenadas em arquivo XML. Caso exista novas *features*, o usuário deverá para a captura da *feature*, salvar o modelo e inicializar outra captura. Confira abaixo código-fonte, com o trecho de código dos pontos de configuração a partir da programação orientada a aspectos.

**Figura 26 – Trecho do código-fonte dos aspectos implementados para a TechREF**

```

public aspect Capture {
    pointcut traceMethods()
        : (execution(* *.*(..))
        || execution(*.new(..))) &&
        !execution(* *.*$*(..)) &&
        !within(Capture) && if(running);

    before() : traceMethods() {
        Object target = thisJoinPoint.getTarget();
        Class<?> type = target.getClass();
        root = false;

        if (!type.getSimpleName().equalsIgnoreCase("")) {
            if (before.size() > 0) {
                for (int i = 0; i < before.size(); i++) {
                    if (type.getSimpleName().equals(before.get(i))) {
                        root = true;
                    }
                }
            }
            if (!root) {
                str.append("<class name=\"" + type.getSimpleName() + "\">\n");
                for (Method m : type.getDeclaredMethods()) {
                    strMethod.append("<method name=\"" + m.getName() + "\" type=\"" +
                        m.getReturnType().getSimpleName() + "\"></method>\n");
                }
                Field[] f = type.getDeclaredFields();
                for (Field field : f) {
                    Class typeName = field.getType();
                    field.setAccessible(true);
                    strAtrib.append("<atrib name=\"" + field.getName().toString() +
                        "\" type=\"" + typeName.getSimpleName() + "\"></atrib>\n");
                }
                str.append("</class>");
                before.add(type.getSimpleName().toString());
            }
        }
    }

    after() : traceMethods() {
}
}

```

Fonte: elaborado pelo autor.

Com a captura da *feature*, o segundo processo da técnica é a geração do modelo orientado a *features*. Porém, as informações que foram armazenadas nos arquivos XML para as *features* do sistema não são suficientes para gerar um modelo UML. É necessário o uso de um componente com elementos que auxiliam na montagem de um *template* de modelo UML. Este componente é um plug-in da plataforma Eclipse chamado de UML2, o qual possui uma biblioteca com elementos suficientes para criar a estrutura necessária para novos modelos UML. Com a UML2 foi possível implementar um *template* no formato de um modelo UML. As informações armazenadas na primeira etapa da captura das *features* será utilizada na montagem do novo modelo UML. Para gerar o modelo UML com as informações da captura, foi implementado métodos para criar classes, criar métodos e criar atributos do novo modelo. A Figura 27 apresenta o método criado para adicionar os métodos das classes capturadas no novo modelo orientado a *features*.

Figura 27 – Método para adicionar operações da classe

```

public void addMethodModel(Model model, org.eclipse.uml2.uml.Class classe) {
    //Upload captured xml
    File f = new File("../VirtualPets/files/CaptureMethod.xml");
    SAXBuilder sb = new SAXBuilder();
    Document doc = null;
    try {
        doc = sb.build(f);
    } catch (JDOMException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    //Get element root
    Element root = (Element) doc.getRootElement();
    List elements = root.getChildren();
    int tam = elements.size();
    //Iterable elements xml root
    for (int j = 0; j < tam; j++) {
        Element atr = (Element) elements.get(j);
        //Checks whether the class is equal to xml
        if (classe.getName().equals(atr.getAttribute("name").getValue())) {
            //List elements (operations) xml
            List elements2 = atr.getChildren();
            int tam2 = elements2.size();
            for (int v = 0; v < tam2; v++) {
                //Get element
                Element atr2 = (Element) elements2.get(v);
                //Create type of method
                PrimitiveType primitivetype =
                    createPrimitiveType(model, atr2.getAttribute("type").getValue());
                //Created operation of the class capture(methods)
                createOperation(classe,
                    atr2.getAttribute("name").getValue(), primitivetype);
            }
        }
    }
}

```

Fonte: elaborado pelo autor.

O código apresentado na Figura 27 ilustra a inserção das operações que foram capturadas das classes. Além das informações de operações das classes, outras informações também são adicionadas no novo modelo. Todas as informações são adicionadas no novo modelo UML. As informações de atributos e tipos dos atributos também são incluídas juntamente com as classes e suas relações. Com todas as informações adicionadas ao novo modelo orientado a *feature*. Após todas as inserções, a etapa do processo é concluída. Abaixo um exemplo de como é o *template* UML criado pela TechREF.

Figura 28 – Modelo orientado a *features*

```

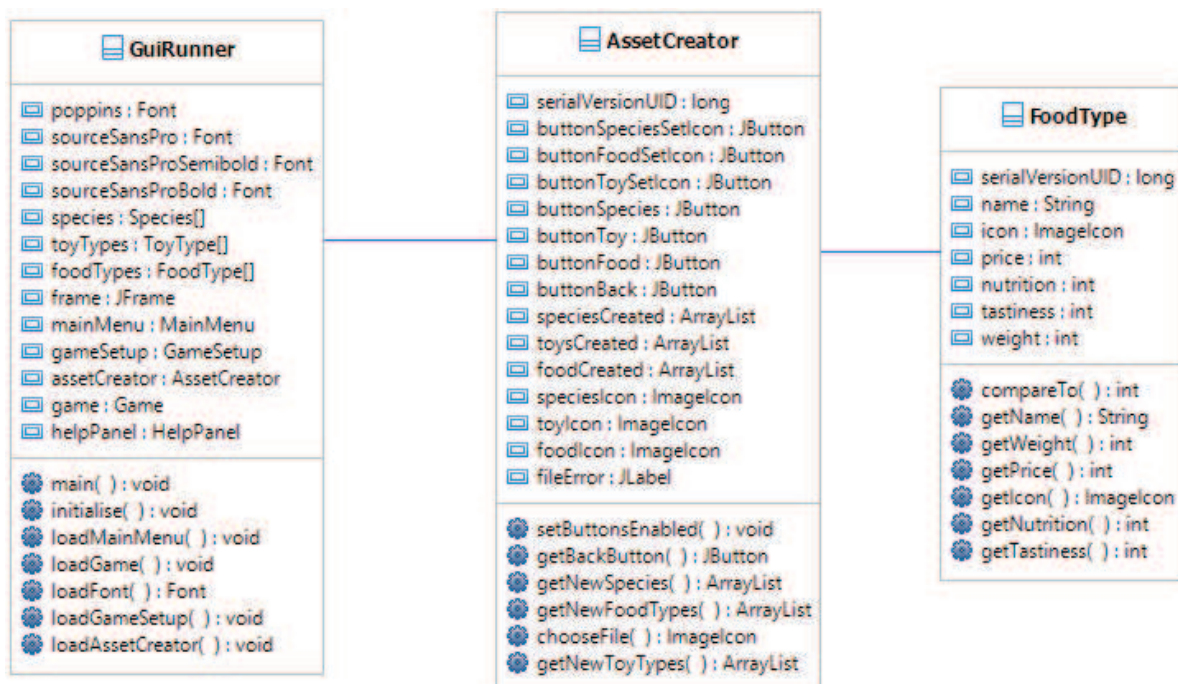
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
  xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
  xmi:id="_EZFnMOB1Eee4SvpME1bHCA"
  name="Case01">
  <packagedElement xmi:type="uml:Class" xmi:id=" EZcMgOB1Eee4SvpME1bHCA" name="GuiRunner">
  </packagedElement>
  <packagedElement xmi:type="uml:Class" xmi:id=" EZdaceB1Eee4SvpME1bHCA" name="AssetCreator">
  </packagedElement>
  <packagedElement xmi:type="uml:Class" xmi:id="_EZeBueB1Eee4SvpME1bHCA" name="FoodType">
  <ownedAttribute xmi:id=" EZeBuuB1Eee4SvpME1bHCA" name="serialVersionUID" type="_EZeB0eB1Eee4SvpME1bHCA"/>
  <ownedAttribute xmi:id=" EZeBu-B1Eee4SvpME1bHCA" name="name" type="_EZeB0uB1Eee4SvpME1bHCA"/>
  <ownedAttribute xmi:id=" EZeBvOB1Eee4SvpME1bHCA" name="icon" type="_EZeB0-B1Eee4SvpME1bHCA"/>
  <ownedAttribute xmi:id=" EZeBveB1Eee4SvpME1bHCA" name="price" type="_EZeB1OB1Eee4SvpME1bHCA"/>
  <ownedAttribute xmi:id=" EZeBvuB1Eee4SvpME1bHCA" name="nutrition" type="_EZeB1eB1Eee4SvpME1bHCA"/>
  <ownedAttribute xmi:id=" EZeBv-B1Eee4SvpME1bHCA" name="tastiness" type="_EZeB1uB1Eee4SvpME1bHCA"/>
  <ownedAttribute xmi:id=" EZeBwOB1Eee4SvpME1bHCA" name="weight" type="_EZeB1-B1Eee4SvpME1bHCA"/>
  <ownedOperation xmi:id=" EZeBweB1Eee4SvpME1bHCA" name="compareTo">
  </ownedOperation>
  <ownedOperation xmi:id=" EZeBxeB1Eee4SvpME1bHCA" name="getName">
  </ownedOperation>
  <ownedOperation xmi:id=" EZeBx-B1Eee4SvpME1bHCA" name="getWeight">
  </ownedOperation>
  <ownedOperation xmi:id=" EZeBxeB1Eee4SvpME1bHCA" name="getPrice">
  </ownedOperation>
  <ownedOperation xmi:id=" EZeBy-B1Eee4SvpME1bHCA" name="getIcon">
  </ownedOperation>
  <ownedOperation xmi:id=" EZeBzeB1Eee4SvpME1bHCA" name="getNutrition">
  </ownedOperation>
  <ownedOperation xmi:id=" EZeBz-B1Eee4SvpME1bHCA" name="getTastiness">
  </ownedOperation>
  </packagedElement>
  <packagedElement xmi:type="uml:Association" xmi:id=" EZeoweB1Eee4SvpME1bHCA"
  </packagedElement>
  <packagedElement xmi:type="uml:Association" xmi:id=" EZeoyOB1Eee4SvpME1bHCA"
  </packagedElement>
</uml:Model>

```

Fonte: elaborado pelo autor.

Após a geração do modelo orientando a *features*, o usuário pode visualizar o modelo UML com a ajuda do EMF. O componente EMF disponibiliza frameworks para edição e visualização de modelagem de dados. A Figura 28 apresentou um arquivo UML no formato texto, isto é, apresentou o template de um arquivo XML e suas informações. Porém, há plugins do EMF que possibilitam ao usuário visualizar o diagrama de classes na forma gráfica.

Figura 29 – Diagrama de classes orientado a *features*



Fonte: elaborado pelo autor.

A Figura 29 ilustra um diagrama de classes gerado pela técnica TechREF. Após o modelo UML ser salvo pelo usuário, é possível visualizar o diagrama de classes que a técnica gerou através de componentes gráficos EMF. Desta forma, é possível representar o diagrama de classes orientado a *features* gerado pela TechREF.

No Capítulo 5 deste trabalho será apresentado o experimento controlado realizado para avaliar o uso de modelos orientados a *features*.

## 5 EXPERIMENTO CONTROLADO

Este Capítulo apresenta um experimento controlado que foi realizado como parte da avaliação do uso de modelos orientados a *features* no processo de manutenção do software. Nesta primeira avaliação o experimento controlado responde à primeira questão de pesquisa (QP-1) elaborada na Seção 1.2 apresentada neste trabalho. Na Seção 5.1 será apresentado o detalhamento do experimento controlado.

### 5.1 Experimento controlado

O experimento controlado foi realizado com o propósito de responder a primeira questão de pesquisa (QP-1). Este experimento faz parte de uma primeira avaliação deste trabalho. Com isto, pretende-se avaliar o uso da UML em atividades de manutenção de software com um grupo de pessoas. Os participantes realizaram individualmente, três atividades de manutenção de software para o mesmo sistema. Na Seção 5.1.1, serão apresentados os objetivos e as questões de pesquisa do experimento controlado. A Seção 5.1.2 descreve a formulação de hipóteses do experimento controlado. A Seção 5.1.3 apresenta as variáveis de estudo. A Seção 5.1.4 é detalhado a seleção de participantes para o experimento. A Seção 5.1.5 apresenta os detalhes da execução do experimento controlado. A Seção 5.1.6 descreve a análise do estudo. Por fim, a Seção 5.1.7 apresenta os resultados do experimento controlado.

#### 5.1.1 Objetivo e questões de pesquisa

O objetivo do experimento controlado é avaliar o uso de modelos UML orientados a *features* em atividades de manutenção de software. Nesta avaliação, duas variáveis serão controladas: esforço e corretude. Estas variáveis são descritas na Seção 5.1.3 deste capítulo. Este estudo busca avaliar o tempo de execução das atividades de manutenção usando diagramas de classes UML tradicionais e diagramas de classes UML orientados a *features*. Busca-se também entender se o número de atividades de manutenção executadas corretamente é maior ou não, usando modelo tradicional ou orientado a *features*. Com isto, o objetivo deste estudo é indicado com base no modelo GQM (BASILI et al., 1994), conforme segue:

**Analisar** uso da modelagem UML orientada a *features*  
**com o propósito** investigar os efeitos  
**com respeito ao** esforço e a corretude  
**através da perspectiva de** desenvolvedores de software  
**no contexto de** manutenção de software.

Em particular, este estudo pretende revelar se os modelos UML orientados a *features*, em comparação com os modelos UML tradicionais completos do sistema, podem reduzir o esforço investido para as tarefas de manutenção. Aumentando a corretude das mudanças

realizadas nas atividades pelos desenvolvedores. Assim, concentra-se em explorar duas questões de pesquisa:

- **Q1:** O uso de modelos UML orientados a *features* reduz o esforço do desenvolvedor em tarefas de manutenção de software?
- **Q2:** Qual é o efeito do uso de modelos UML orientados a *features* sobre a corretude das tarefas de manutenção de software?

### 5.1.2 Formulação de hipóteses

As duas questões de pesquisa deste experimento procuram responder sobre a corretude das tarefas e o esforço que será dedicado ao experimento controlado. As questões de pesquisa apresentadas procuram superar as expectativas comparando os modelos de cada método aplicado no experimento. Com o ideal de ajudar a contribuir para as discussões sobre o uso de modelos UML no desenvolvimento de software.

A questão da pesquisa 1 procura avaliar o esforço dos participantes que usaram o modelo orientado a *features* e o modelo tradicional completo do sistema para responder as tarefas do experimento. Portanto, a primeira hipótese avalia se o uso de modelos orientados a *features* requer um esforço maior ou igual ao modelo tradicional.

A hipótese nula será verdadeira quando não houver redução no esforço das tarefas que usam modelos UML orientados a *features*. A hipótese alternativa é válida quando houve uma redução na execução das tarefas. Com base nesta afirmação, é possível indicar as hipóteses nulas e alternativas da seguinte forma:

**Hipótese Nula 1 - H<sub>1-0</sub>** - Não houve redução no esforço do tempo de execução das tarefas em comparação com o modelo tradicional.

**H<sub>1-0</sub>:**  $\text{Esforço}_{(\text{feature})} > \text{Esforço}_{(\text{tradicional})}$

**Hipótese Alternativa 1 - H<sub>1-1</sub>** - Houve uma redução no esforço em relação ao tempo de execução das tarefas em comparação com o modelo tradicional.

**H<sub>1-1</sub>:**  $\text{Esforço}_{(\text{feature})} \leq \text{Esforço}_{(\text{tradicional})}$

A hipótese 2 apresenta o uso de modelos UML orientados a *features* com benefícios para a manutenção de tarefas dentro do desenvolvimento de software. Esta hipótese avalia a corretude das tarefas realizadas por cada participante, verificando se a hipótese nula é inválida. A hipótese 2 será verdadeira quando a ocorrência do número de correções de atividades com uso de modelos orientados a *features* for maior ao modelo UML tradicional.

Na hipótese alternativa desta questão, o uso dos modelos orientados a *features* deve obter no experimento um número mais satisfatório que indica a hipótese alternativa como verdadeira. Assim, indica que as tarefas que foram respondidas utilizando o modelo orientado a *features* obteve um número melhor de atividades corretas comparadas as atividades que usaram modelos tradicionais. Com isso, as hipóteses são formuladas da seguinte forma:

**Hipótese Nula 2 - H<sub>2-0</sub>** - O uso de modelos UML orientados a *features* não produziu um número menor de atividades executadas corretamente.



**H<sub>2-0</sub>:**  $\text{Corretude}_{(\text{feature})} > \text{Corretude}_{(\text{tradicional})}$

**Hipótese Alternativa 2 - H<sub>2-1</sub>** - O uso de modelos UML orientados a *features* produziu um número menor ou igual de atividades executadas corretamente.

**H<sub>2-1</sub>:**  $\text{Corretude}_{(\text{feature})} \leq \text{Corretude}_{(\text{tradicional})}$

### 5.1.3 Variáveis de estudo

A principal razão pela qual essas variáveis são importantes para este experimento é avaliar o uso de modelos UML no processo de desenvolvimento de software. O uso de modelos em atividades desempenhadas pelos desenvolvedores pode ser relevantes e indicar um ganho significativo do uso de modelos UML. As variáveis dependentes do experimento controlado comparam o esforço e a corretude das tarefas que foram realizadas pelos participantes, é possível entender que o uso de modelos orientados a *features* pode ajudar nas tarefas de manutenção de software, ou seja, essa métrica torna possível avaliar a diferença entre o uso de modelos orientado a *features* comparado com a forma tradicional.

A variável independente das hipóteses 1 e 2 é o conhecimento de modelos UML no uso de modelos orientados a *features*. Como mencionado anteriormente, o uso de modelos UML orientados a *features* podem apresentar bons resultados dentro do processo de manutenção das atividades do experimento.

### 5.1.4 Contexto e seleção de participantes

O experimento foi realizado com 10 alunos com experiência profissional na área de desenvolvimento de software. Os participantes selecionados eram estudantes do curso de mestrado do Programa de Pós-Graduação em Computação Aplicada e com um diploma de bacharel (ou equivalente) nas áreas de tecnologia da informação. Todos os selecionados possuíam conhecimentos consideráveis em modelagem de software e linguagem de programação. Os participantes também foram convidados a assistir uma pequena explicação de como o experimento deve funcionar e como as atividades que iriam ser realizadas no experimento controlado.

Os critérios de seleção para a participação do experimento foram os seguintes: experiência em desenvolvimento de software e experiência em modelagem de software. Os participantes tinham idade entre 25 e 35 anos. Os participantes escolhidos para o experimento tinham mais de 2 anos de experiência em modelagem de software. Todos os participantes tinham mais de 2 anos de experiência em desenvolvimento de software. O experimento controlado fez parte de uma atividade curricular do curso de pós-graduação e foi realizado como exercícios laboratoriais práticos. Cada participante foi exposto ao mesmo nível de treinamento que está sendo avaliado.

### 5.1.5 Detalhes do experimento

Este experimento caracteriza o uso de modelos UML orientados a *features* e também modelos UML na forma tradicional. Os participantes realizaram 3 tarefas simulando a manutenção de um sistema de software. A Tabela 2 apresenta as atividades que foram

realizadas no experimento. As tarefas são típicas da manutenção de software. Os modelos da forma tradicional foram gerados a partir de ferramentas que disponibilizam a engenharia reversa de código na forma tradicional, ou seja, de todo o sistema. Para as atividades com o uso de modelos orientados a *features*, os modelos foram modificados para que contenham apenas as informações necessárias para a execução das atividades. As tarefas realizadas no experimento controlado são apresentadas abaixo.

**Tabela 2 – Atividades do experimento**

<b>Tarefas</b>	<b>Feature</b>	<b>Descrição</b>
T1	Alterar Senha	Alteração no método de validação de senha;
T2	Login / Autenticação	Implementar a validação de usuários/perfis;
T3	Exportação	Criar a funcionalidade de exportação de senhas;

Fonte: Elaborado pelo autor.

As tarefas foram criadas em critérios de complexidades e dificuldades diferentes. Cada tarefa consiste em adicionar novas funcionalidades de valor prático e características novas em partes do sistema. Pelo menos uma classe em cada caso de uso deve ser modificada. Um resumo das tarefas que foram aplicadas no experimento controlado é apresentado na Tabela 3.

**Tabela 3 – Descrição das atividades do experimento**

<b>Tarefas</b>	<b>Modificações necessárias</b>
1	Campos habilitados - Será necessário criar um método que receba por parâmetro a informação do componente criado com o valor de entrada. Este método irá verificar a possibilidade de habilitar os campos do formulário.
2	Usuário logado - Para que seja possível implementar mais esta nova funcionalidade será necessária a inclusão de uma classe e um novo método para a mensagem de que o usuário conectou/entrou e está podendo receber mensagens no formulário.
3	Salvar informações - Será necessário a criação de um método que faça a gravação das mensagens enviadas em arquivo texto no diretório da máquina do usuário. O método receberá por parâmetro a mensagem e irá gravar no arquivo.

Fonte: Elaborado pelo autor.

A Tarefa 1 verifica se o valor contém as informações ativas (ativadas) para que o participante possa ativar os campos na tela; caso contrário o participante não poderá habilitar os campos do formulário. Na Tarefa 2, consiste em informar todos os usuários que estão conectados em um sistema, que um novo usuário apenas conectado. Na Tarefa 3 as lojas enviaram mensagens. A opção de salvar as mensagens no arquivo de texto deve ser criada para dizer que é possível salvar as mensagens informadas ao usuário se quiser gravar as mensagens. Se o usuário seleciona para gravar, as mensagens devem ser salvas no arquivo de texto. O nome do arquivo e a forma de implementar a gravação do arquivo de texto são a critério do desenvolvedor.

O participante pode escolher entre duas maneiras de realizar as atividades: a primeira maneira é realizar as tarefas da experiência com o uso de diagramas criados a partir da engenharia reversa tradicional e a segunda maneira é usar os diagramas orientados a *features*.

Os modelos utilizados no estudo foram diagramas de classes UML com cerca de 8 classes e 7 relacionamentos para atividades tradicionais de engenharia reversa e com cerca de 4 classes e 3 relacionamentos para atividades com engenharia reversa orientada a *features*. Este é o tamanho médio dos modelos ao realizar uma experiência controlada como estudar como está em conformidade com os diagramas anteriores. Além disso, dadas as restrições de tempo de experiências controladas, os participantes não poderiam ser expostos a grandes modelos devido à complexidade e ao tempo das atividades.

#### 5.1.6 Análise do estudo

A análise quantitativa foi realizada de forma superficial com dados brutos, realizando um teste estatístico com amostras da experiência controlada. Essas análises responderam as questões de pesquisa sobre os efeitos do uso de modelos de *features* e modelos tradicionais.

As análises foram realizadas para testar as hipóteses individualmente para cada atividade, bem como para todas as tarefas do experimento. Os dados qualitativos foram coletados de algumas fontes: questionário, vídeo e registros de transcrição. Isso potencialmente ajudou a provar algumas evidências para explicar os resultados quantitativos e qualitativos que podem resultar em conclusões concretas.

Na seção a seguir, os resultados da experiência serão apresentados. Esses números foram extraídos do experimento realizado por participantes dedicados a completar as tarefas do experimento. A análise qualitativa dos dados foi obtida com dados do questionário e informações das gravações de vídeo.

#### 5.1.7 Resultados

Nesta seção serão apresentados os resultados obtidos através da realização do experimento. Os resultados serão apresentados de uma forma inicial com os valores brutos, visto que alguns valores não foram possíveis serem extraídos inicialmente por dados inválidos do experimento. Esta avaliação foi realizada com o uso de modelos UML tradicional para execução de atividades e com o uso de modelos UML orientado a *features*. Esses valores serão ilustrados através de tabelas e gráficos ao longo desta seção. Em cada atividade realizada no experimento controlado teve seu tempo registrado e as atividades que foram realizadas de forma semelhante em ambos os formatos do experimento foram todas registradas através de gravação. Abaixo na Tabela 4 são apresentados os valores de cada participante que usou os modelos UML no formato tradicional.

**Tabela 4 – Tempos de cada participante por tarefas do modelo tradicional**

	Tarefa 1	Tarefa 2	Tarefa 3
Participante 1	50	19	41
Participante 2	27	8	15
Participante 3	22	23	12
Participante 4	33	21	22
Participante 5	31	10	14

Fonte: Elaborado pelo autor.

Os resultados na Tabela 4 apresentam os dados dos participantes que realizaram o experimento com os modelos tradicionais. Esses participantes realizaram 3 atividades, cada atividade com o seu tempo de execução contabilizado em minutos. O tempo de esforço de atualização do UML foi adicionado ao tempo de execução da atividade. Os valores de esforços foram obtidos separadamente, o tempo apresentado na Tabela 4 já contabiliza o tempo gasto em decorrência da atualização dos modelos UML e da execução das tarefas de cada participante. Cada formato do experimento teve a participação de 5 integrantes no experimento. Os 5 participantes que participaram do experimento com modelo tradicional não participaram do modelo orientado a *features*, ou seja, outros 5 participantes é que participaram do experimento conforme ilustrado na Tabela 5 onde apresenta os tempos totais dos participantes que realizaram as atividades com modelos UML orientados a *features*.

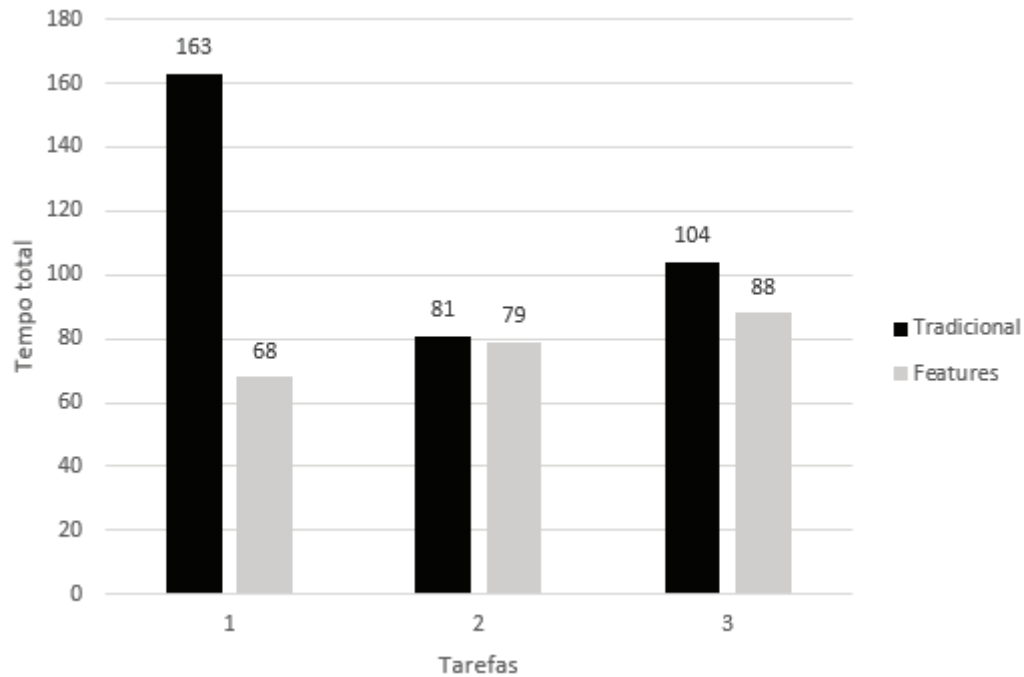
**Tabela 5 – Tempos de cada participante por tarefas do modelo *features***

	Tarefa 1	Tarefa 2	Tarefa 3
Participante 6	12	7	13
Participante 7	8	23	18
Participante 8	20	27	25
Participante 9	18	14	20
Participante 10	10	8	12

Fonte: Elaborado pelo autor.

Avaliando os resultados apresentados até o momento nas Tabelas 4 e 5 referentes ao esforço dos participantes em cada tarefa, na Figura 30 é apresentado um gráfico com o tempo total do esforço de cada atividade somado com o tempo de todos os participantes em cada atividade comparando as duas formas de execução do experimento. Em tom de cor preto é representado a forma tradicional do experimento com uso de modelo tradicional. Em tom de cor cinza é ilustrado a forma através da execução das tarefas no formato de *features*. Ao realizar uma análise básica sobre os dados brutos ilustrado na imagem, é possível observar nas três tarefas o esforço no tempo total gasto para a conclusão das tarefas foi maior para a realização do experimento utilizando a forma tradicional. Na execução das tarefas do experimento no formato de modelos de *features*, teve um menor esforço.

**Figura 30 – Tempo total de cada tarefa**



Fonte: Elaborado pelo autor.

Na Figura 30 acima é possível observar que o esforço da execução das atividades da forma tradicional comparado com a formato de modelos de *features* visualmente os valores apresentados indicam uma ligeira vantagem para no que diz respeito ao esforço dos participantes ao realizar as tarefas com modelos de *features*. O Experimento realizado com o modelo tradicional teve um esforço maior de 33% no total das horas trabalhadas comparado com o modelo orientado a *features*. A realização da tarefa 1 comparando os dois modelos tanto da forma tradicional como na *features*, teve um ganho no tempo da execução da tarefa de 58%. Após a realização do experimento, o modelo orientado a *features* apresentou números positivos do uso na manutenção de software.

Com a análise do tempo de cada participante em cada atividade desenvolvida, com os resultados apresentados nesta seção é possível medir o menor esforço dos participantes dedicados nas tarefas desenvolvidas com os modelos em comparação com a forma tradicional utilizada no experimento. Mas mesmo os gráficos e os dados mostrando essa tendência, um teste estatístico foi realizado para verificar, de fato, se os valores apresentados refletem em real um menor esforço na elaboração das atividades com modelo orientado a *features*.

Os valores apresentados pelo experimento indicam que os valores não são normalmente distribuídos, desta forma, foi escolhido um teste não paramétrico. Desta forma, o teste de Wilcoxon foi escolhido para analisar os valores. O teste é indicado para uma análise de duas amostras comparativas, para investigar a hipótese de cancelar a igualdade de dados, ou seja, o teste de Wilcoxon é realizado para avaliar a distribuição dos valores e investigar a possível diferença entre os valores e o possível ganho do uso de modelos orientados a *features* na manutenção do desenvolvimento de software.

O teste foi realizado com os valores extraídos pelo esforço dos participantes que realizaram o experimento. No total foram coletadas 30 amostras, que caracterizam o tempo que cada participante realizou a atividade. Na Tabela 6 é possível observar o valor de *p-value* <0,001, com intervalo de confiança de 95%.

**Tabela 6 – Resultado do teste de Wilcoxon**

<b>Test Wilcoxon Signed-Rank – Effort</b>	
<i>V</i>	105
<i>p-value (Two-tailed)</i>	< 0,001
<i>Alpha</i>	0,05
<i>Variance (V)</i>	253,000

Fonte: Elaborado pelo autor.

De acordo com as características do teste de Wilcoxon, o valor de *p-value* inferior a 0,05 é considerado um bom resultado para os casos de teste. O teste de Wilcoxon apresentou um resultado de 0,001 para o valor de *p-value*, isso ilustra evidências fortes e pistas significativas que as duas amostras apresentadas não são zero. Com este resultado, é suficientemente possível rejeitar a primeira hipótese nula (H1-0) do experimento controlado.

Com os resultados dos testes realizados com os valores de esforço, é possível avaliar que o esforço de manutenção no desenvolvimento do software com o uso do modelo orientados a *features* é menor que o esforço com o uso do modelo tradicional. Desta forma, reduzir o esforço de manutenção do software é uma grande conquista para os desenvolvedores que usaram os modelos orientados a *features*. Com isso, a primeira hipótese alternativa (H1-1) é verdadeira, contribuindo para responder à primeira questão de pesquisa deste trabalho (QP-1).

Com relação ao desempenho das corretudes das tarefas, foram analisadas as respostas das tarefas verificando se a resposta estava correta ou não que o participante realizou no experimento. A resposta de cada tarefa foi analisada pela assertividade, isto é, foi possível avaliar se as tarefas foram realizadas de forma correta ou incorreta. Alguns participantes responderam as questões, porém não conseguiram realizar de forma correta. Mesmo concluindo as tarefas do experimento controlado, alguns participantes não concluíram a tarefa de forma correta. Na Tabela 7 são apresentados os resultados das respostas de cada atividade por participante com o uso do modelo tradicional.

**Tabela 7 – Corretude das respostas do modelo tradicional**

	Tarefa 1	Tarefa 2	Tarefa 3
Participante 1	Incorreta	Incorreta	Incorreta
Participante 2	Correta	Correta	Incorreta
Participante 3	Incorreta	Correta	Correta
Participante 4	Correta	Incorreta	Incorreta
Participante 5	Incorreta	Correta	Correta

Fonte: Elaborado pelo autor.

Com a apresentação dos resultados das correções das tarefas realizadas por participantes do experimento controlado com os modelos tradicionais, é possível observar a complexidade das tarefas exigidas no experimento. Embora os participantes selecionados tivessem informado a sua experiência em desenvolvimento de software, observou-se uma grande dificuldade na elaboração das respostas e na assertividade de cada um dos participantes. Em alguns casos o participante não conseguiu ser assertivo em nenhuma das tarefas. Neste cenário é importante salientar que a relação entre o tempo do esforço do participante e a corretude das respostas, são totalmente desvinculadas para esta avaliação.

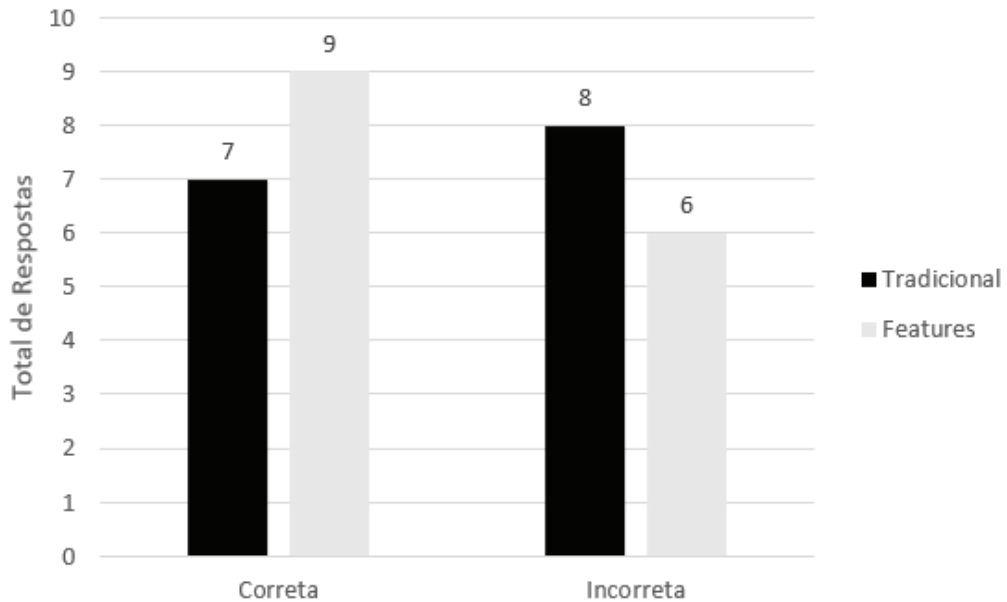
Na Tabela 8 são apresentados os resultados dos outros 5 participantes que realizaram o experimento controlado usando o modelo orientado a *features*. Neste cenário foi observado uma pequena vantagem na assertividade das questões entre o formato tradicional comparado ao formato de *features*. Pode-se levar em consideração o fato de que os modelos menores fizeram com que compreensão das atividades e das respostas das tarefas tivesse um resultado melhor comparado ao formato de modelos tradicional utilizado no experimento.

**Tabela 8 – Corretude das repostas do modelo orientado a *features***

	Tarefa 1	Tarefa 2	Tarefa 3
Participante 6	Correta	Incorreta	Correta
Participante 7	Correta	Incorreta	Incorreta
Participante 8	Correta	Correta	Incorreta
Participante 9	Incorreta	Incorreta	Correta
Participante 10	Correta	Correta	Correta

Fonte: Elaborado pelo autor.

Com base nos resultados dos participantes apresentados no que diz respeito a corretude das respostas das questões do experimento realizado, foram contabilizadas 15 respostas para o formato tradicional e 15 respostas para o formato do experimento com o uso de modelos orientados a *features*. Desta forma, no gráfico apresentado na Figura 31, ilustra o total das respostas que foram extraídas do experimento controlado. Para o método de aplicação de modelos tradicionais foram concluídas 7 tarefas de forma corretas e 8 tarefas de forma incorretas. Para o método aplicado no experimento com o uso de modelos orientados a *features* houve 9 tarefas respondidas corretamente e 6 tarefas que não foram respondidas corretamente.

**Figura 31 – Total de respostas das tarefas.**

Fonte: Elaborado pelo autor.

Com a apresentação dos dados na imagem acima é possível visualizar os resultados de uma forma abstrata. Com isto, foi realizado um teste para avaliar melhor os resultados apresentados. O teste de McNemar foi selecionado para ser aplicado neste resultado. A Tabela 9 apresenta os 60 pares de respostas do experimento controlado que foram realizadas através dos participantes foram relacionados. Esta relação de dados representa a frequência das duas amostras. As duas amostras foram emparelhadas com as características corretas e incorretas dos resultados do experimento controlado. Com isso, os valores de 16 corretos x corretos, 13 incorretos x corretos, 17 corretos x incorretos e 14 incorretos x incorretos. Os valores totalizaram 60 pares, como mostra na Tabela 9 abaixo.

**Tabela 9 – Relação das correções com cenários - teste McNemar**

		<i>Features</i>		Total
		Corretas	Incorretas	
<b>Tradicional</b>	Corretas	16	13	29
	Incorretas	17	14	31
Total		33	27	60

Fonte: Elaborado pelo autor.

O nível de confiança para este teste foi de 95%. O resultado do teste de McNemar mostrou um valor *p-value* de 0,584 que é maior que o valor de 0,05 para a rejeição da hipótese nula. Assim, uma vez que o valor *p-value* calculado é maior do que o nível de significância  $\alpha = 0,05$ , a hipótese nula não pode ser rejeitada. De acordo com o teste de



McNemar, o risco de rejeitar a hipótese nula H2-0 é de 58,39%. A Tabela 10 mostra os valores representados pelo teste.

**Tabela 10 – Teste corretude McNemar**

McNemar Test – Corretude	
Q	0,300
z  (valores observados)	0,548
z  (valores críticos)	1,960
<i>p-value</i>	0,584
<i>Alpha</i>	0,05

Fonte: Elaborado pelo autor.

Com os resultados apresentados através dos gráficos e testes realizados com os valores do experimento, é possível avaliar que o esforço da manutenção das atividades com o uso do modelo orientado a *features* é menor do que o esforço utilizando o modelo tradicional. Desta forma, a redução do esforço na manutenção de software é um ganho para os desenvolvedores que podem utilizar os modelos orientados a *features* gerados de forma automática através da ferramenta TechREF. Após a realização dos testes para ambas as hipóteses do experimento controlado, é respondida a primeira questão de pesquisa (QP-1) deste trabalho após avaliação.

É satisfatório concluir que os modelos orientados a *features* foram melhores absorvidos pelos participantes e que as tarefas foram melhores desenvolvidas com menos esforço para serem realizadas no experimento controlado. Não é possível afirmar com convicção que os modelos orientados a *features* obtiveram melhores resultados e com menor esforço dos participantes para a realização das atividades. Porém, os modelos orientados a *feature* têm benefícios no entendimento e na compreensão dos modelos e também para o desenvolvimento de novas atividades.



## 6 AVALIAÇÃO DO ESTUDO DE CASO

Com a finalidade de avaliar a técnica proposta no Capítulo 4, bem como atender aos objetivos específicos referentes à técnica que foram enumerados na Seção 1.3, foi realizada uma avaliação da técnica TechREF utilizando uma aplicação onde foram aplicadas métricas para avaliar os modelos gerados pela técnica. A realização deste estudo de caso responde à segunda questão de pesquisa (QP-2) elaborada na Seção 1.2 apresentada neste trabalho.

O Capítulo 6 foi organizado da seguinte maneira: A Seção 6.1 apresenta o funcionamento da TechREF, interface da técnica com uma aplicação, e suas funcionalidades juntamente com os aspectos de implementação. A Seção 6.2 apresenta a estrutura do estudo de caso. Os cenários do estudo de caso são apresentados na Seção 6.3. A Seção 6.4 apresenta os resultados. A análise dos resultados é realizada na Seção 6.5. Por fim, a Seção 6.6 apresenta as ameaças à validade do estudo.

### 6.1 Utilizando a TechREF

Para que a técnica desenvolvida neste trabalho fosse avaliada, foi necessário utilizá-la em um contexto de aplicação construída para que com as características de um projeto de mercado possa averiguar a precisão e a acurácia da TechREF. Com isto, foi pesquisado uma aplicação com as seguintes características: uma aplicação *open source*, criada e desenvolvida em tecnologia Java, com cenários que permitam aplicar o uso da técnica TechREF. Estas características são importantes para avaliar a técnica.

Após realizar algumas pesquisas em diretórios e repositórios de códigos fontes abertos como, por exemplo, GitHub (GitHub, 2017), diversas aplicações que contemplam as características citadas no parágrafo anterior, foi identificado na aplicação Virtual Pets (Virtual Pets, 2017) as necessidades para esta avaliação do trabalho. Esta aplicação foi criada com o objetivo de reproduzir a rotina de cuidar de um animal. É um jogo onde o usuário escolhe o animal que deseja cuidar virtualmente, alimenta e brinca. Com esta aplicação definida, será realizada a avaliação da TechREF.

A TechREF foi adicionada junto à aplicação Virtual Pets como uma forma avaliar os resultados. O desenvolvimento da TechREF não tem relação com a aplicação para a avaliação. Para que seja adicionado ao projeto, foi necessário importar o projeto da TechREF dentro da aplicação para realizar a captura das *features*. Com o objetivo de avaliar a técnica, foi criado cenários dentro da aplicação escolhida como forma de representar modificações dentro do sistema. Com a inclusão da TechREF dentro da aplicação Virtual Pets, será possível a captura dos modelos de *features*.

#### 6.1.1 Aplicando a TechREF

A TechREF foi adicionada na solução de desenvolvimento do projeto Virtual Pets (Virtual Pets, 2017). O projeto Virtual Pets foi escolhido para ser utilizado como forma de avaliação da TechREF por possuir vários cenários dentro do projeto. A aplicação Virtual Pets foi selecionada por constituir cenários variados que proporcionam a criação de casos de testes para a geração de modelos orientados a *features*. Com isto, foi criado cenários com fluxo de execução que serão apresentados nos próximos tópicos com o uso da TechREF.

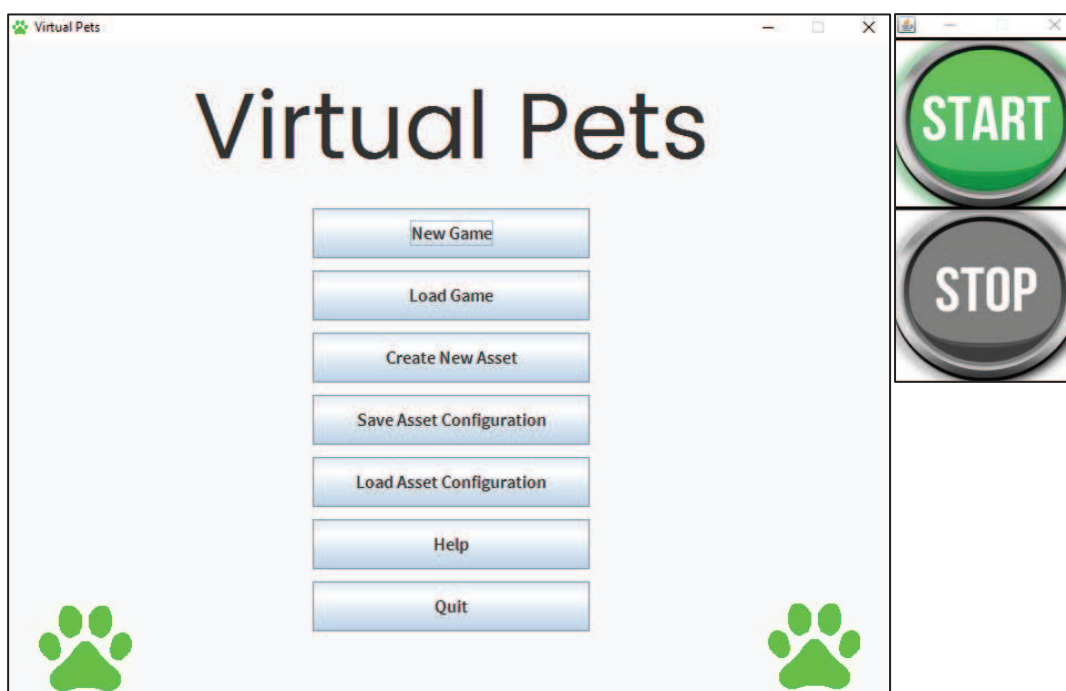
O Algoritmo 3 abaixo ilustra a inicialização da TechREF. Esta inicialização é realizada no método *initialize* da aplicação. Ao iniciar a aplicação, o usuário visualiza a tela inicial do projeto Virtual Pets e também a interface inicial da técnica desenvolvida, a TechREF. A interface da aplicação escolhida é exibida juntamente como a interface da TechREF. A interface da TechREF tem duas opções, a de inicialização “START e a de conclusão “STOP”. Caso o usuário não queira utilizar no primeiro momento a captura das *features*, o usuário pode minimizar as opções. Caso ele deseje começar a captura basta maximizar as opções. As funcionalidades da aplicação não têm relação com as funcionalidades da TechREF. Porém a funcionalidade da TechREF tem relação com as funcionalidades da aplicação em que foi adicionada. Com isto, o usuário poderá utilizar a aplicação sem nenhuma restrição mesmo utilizando a técnica de engenharia reversa.

É possível observar na Figura 32 abaixo que a opção “START está com tom de cor verde, e a opção “STOP está em tom cinza. Esta característica representa que ainda não foi inicializada nenhuma captura, ou não está sendo capturada nenhuma *features* até o momento. Esta funcionalidade foi desenvolvida para que quando o usuário inicialize uma captura de *features* ele deverá clicar no ícone em tom verde escrito “START”.

### Algoritmo 3: Inicialização

- 1: **Inicializa** *aplicação()*;
- 2: **Executar** *rodar(aplicação)*;
- 3: **Inicializa** *aplicaçãoTechREF()*;
- 4: **Executar** *rodar(aplicaçãoTechREF)*;
- 5: *aplicaçãoTechREF.janelastart()*;
- 6: *aplicaçãoTechREF.janelastop()*;

Figura 32 – Aplicação com o uso da TechREF



Fonte: Elaborado pelo autor.

### 6.1.2 Aspectos de implementação

**Inicialização da técnica.** Quando o usuário desejar capturar uma nova *feature*, ele deverá clicar na opção “START”. Enquanto não inicializar a captura, o ícone de “STOP” ficará sempre desabilitado, em tom cinza. Somente quando usuário inicializar uma captura é que o ícone de “STOP” apresentado na interface da TechREF irá ficar com a cor vermelha, ilustrando assim que existe uma *feature* em processo de captura. Ao clicar no ícone, será informado se o usuário deseja realmente começar a captura da *feature*. Caso a opção seja “SIM”, indicando que deseja começar a capturar a nova *feature*, o sistema irá iniciar a captura das informações necessárias para a captura das *features*. Caso o usuário não queira iniciar, basta cancelar a inicialização em “NÃO”.

No momento em que o usuário inicializa a TechREF, todas as ações do usuário serão coletadas. A captura de cada funcionalidade é responsabilidade do usuário de gerar uma informação consistente, isto é, o usuário é responsável para captura da *feature*. O usuário deve realizar o fluxo correto da *feature* para que o sistema com as informações que foram capturadas possa gerar um modelo correto. Quando o usuário realizar uma captura de uma nova *feature*, o fluxo de dados deverá realizado somente uma por vez. Na Figura 33 abaixo é possível observar a mensagem que é ilustrada para usuário no momento em que inicializa, ou clica no ícone “START” da interface da técnica para a captura das *features*.

Figura 33 – Tela de confirmação de captura de *features*

Fonte: Elaborado pelo autor.

**Inicialização da captura das *features*.** No momento em que o usuário informa que deseja realmente capturar as *features*, é criado um arquivo com formato .xml que irá armazenar as informações que serão capturadas e armazenadas para serem utilizadas na geração do novo modelo orientado a *features*. Este padrão contempla a relação das classes que compõem o fluxo do processo da *feature*. A Tabela 34 apresenta um exemplo retirado da aplicação escolhida para a avaliação da ferramenta.

Figura 34 – Modelo de .xml com as informações das classes capturadas

```

<?xml version="1.0" encoding="UTF-8"?>
<model>
  <class name="Game">
    <class name="Player"></class>
    <class name="ShopPanel">
      <class name="ShopFoodDisplayer">
        <class name="FoodType"></class>
      </class>
      <class name="ShopToyDisplayer">
        <class name="ToyType"></class>
      </class>
    </class>
  </class>
</model>

```

Fonte: Elaborado pelo autor.

Até que o usuário não conclua a captura das informações referentes a *feature*. As informações serão armazenadas e relacionadas enquanto estiver sendo capturadas. Quando o usuário desejar concluir a captura e após realizar todo o fluxo da *feature*, o usuário deverá concluir de forma que os passos realizados até então estejam corretos para representar todo o modelo referente da captura que ele realizou. A conclusão da captura da informação da *feature* será realizada quando o usuário clicar no ícone “STOP”. Na Figura 35 é ilustrado um exemplo onde o ícone “STOP” que está em tom de cor vermelho. Como característica de implementação da técnica, sempre que houver uma *feature* sendo capturada o ícone ficará em tom vermelho. Com isto o usuário irá clicar na imagem em vermelho realizando assim a parada da captura da *feature*. Ao clicar no ícone “STOP” da interface da TechREF será exibido uma mensagem de confirmação se o usuário deseja realmente realizar a conclusão da captura da *feature*.

Figura 35 – Tela de conclusão da captura da *feature*

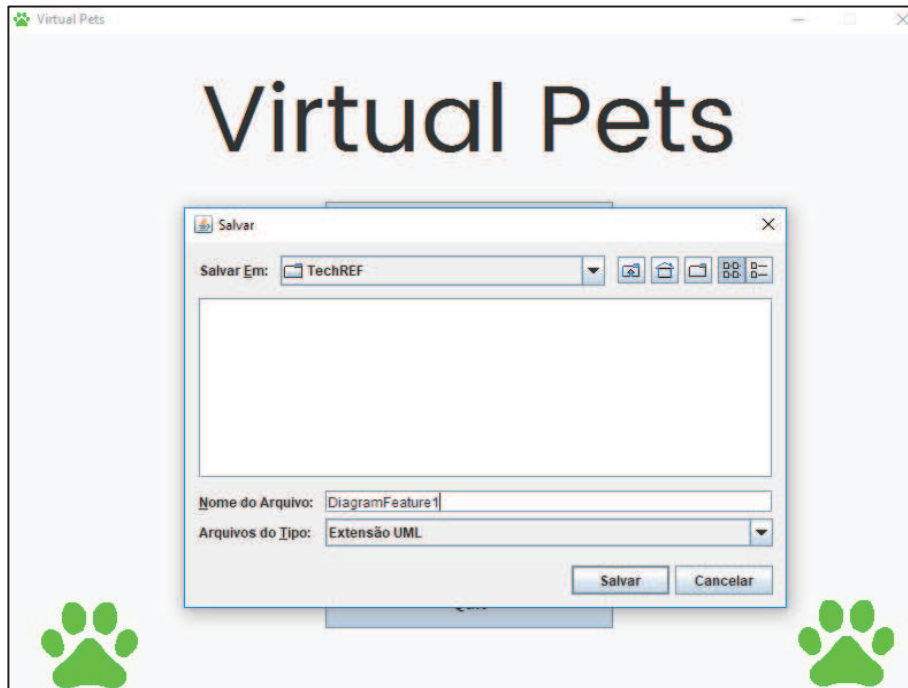


Fonte: Elaborado pelo autor.

**Geração do modelo orientado a *features*.** Após a conclusão do usuário, a TechREF irá solicitar ao usuário onde deseja salvar o modelo gerado. O usuário então deverá salvar o modelo gerado. Caso isso não ocorra, o modelo não é gerado. É importante o usuário selecionar um diretório ou salvar no diretório padrão da técnica. Foi criado um diretório padrão no diretório C:\ na pasta TechREF. Esta pasta é criada quando o usuário conclui uma captura de *features*. O novo modelo criado pode ser salvo no diretório padrão da técnica ou em um diretório em que deseja salvar o modelo. Para isto, basta selecionar o caminho na janela onde irá informar o nome do modelo orientado a *features* que deseja colocar. A Figura

36 apresenta a tela com a indicação do nome do arquivo e da confirmação do modelo orientado a *features* salvo com sucesso.

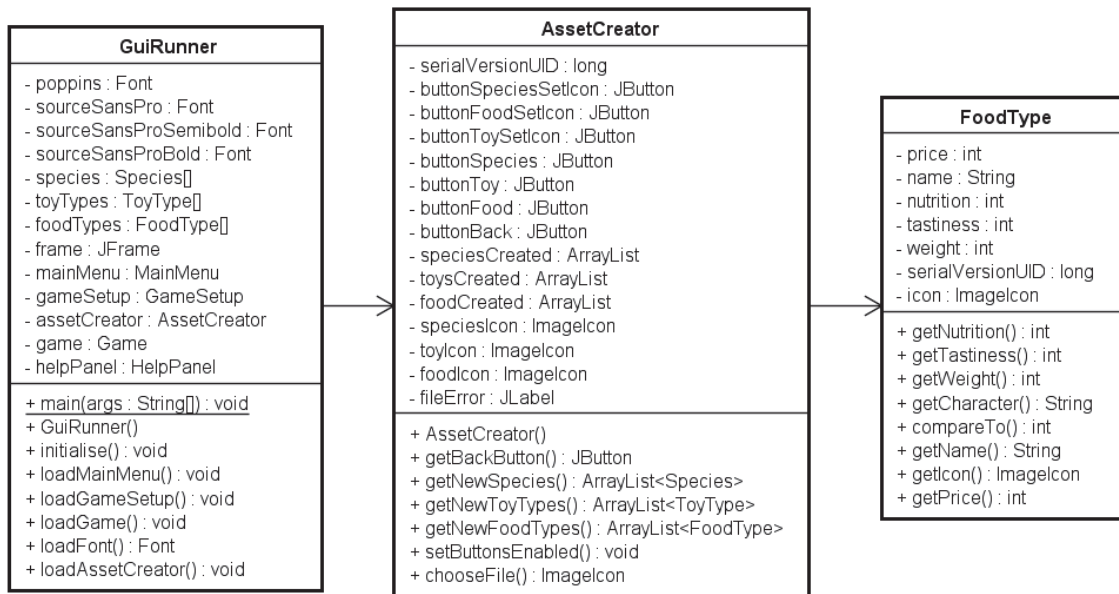
**Figura 36 – Tela para salvar o modelo orientado a *features***





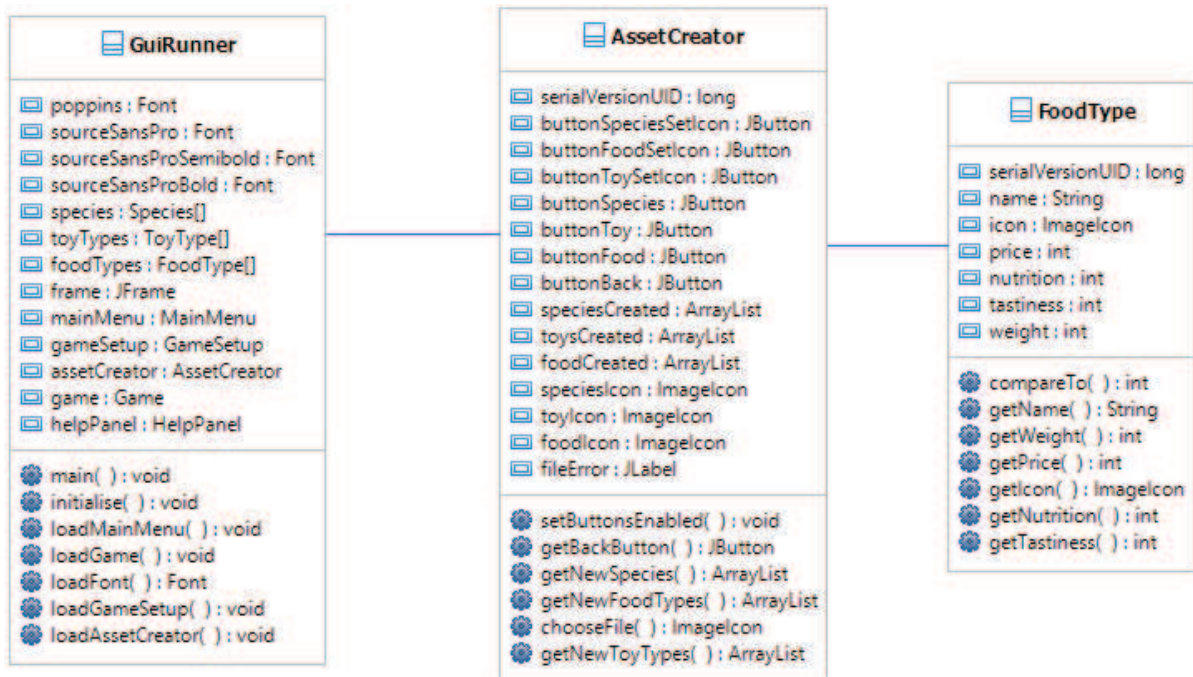
Após salvar os modelos gerados pela técnica TechREF, é possível visualizar os modelos de forma gráfica através dos componentes EMF. Abaixo é possível visualizar os dois modelos. Na Figura 37 é ilustrado o modelo ideal criado pelos participantes do estudo de caso e na Figura 38 é possível visualizar o modelo gerado pela TechREF.

**Figura 37 – Modelo ideal**



Fonte: Elaborado pelo autor

**Figura 38 – Modelo gerado pela TechREF**



Fonte: Elaborado pelo autor.

Nas Figuras 37 e 38 acima é possível visualizar a semelhança entre os modelos apresentados. A Seção 6.2 apresenta a estrutura do estudo de caso para avaliação da TechREF.

## 6.2 Estrutura do estudo de caso

Como forma de avaliar este trabalho na efetividade da técnica TechREF ao produzir modelos orientados a *features* com o uso da engenharia reversa do código-fonte. Este estudo utiliza métricas selecionadas através da execução do software SDMetrics (SDMetrics, 2017). Para realizar a comparação dos resultados da avaliação da técnica com modelos tradicionais, ou modelos originais do projeto, foram selecionados participantes, voluntários que desejaram participar desta avaliação do trabalho. Com isto, foram aplicadas métricas ao modelo gerado e ao modelo ideal para que seja possível avaliar a precisão (*precision*), recuperação (*recall*) e média ponderada (*f-measure*) dos diagramas.

### 6.2.1 Método de avaliação

Para avaliar os modelos gerados pela técnica TechREF foram criados 4 cenários. Estes cenários representam 4 funcionalidades do sistema que pertence a um projeto de sistemas. Estas funcionalidades foram elaboradas a partir do projeto escolhido para a avaliação deste trabalho. Com as funcionalidades definidas, foram selecionados participantes que desenvolveram modelos ideais para cada cenário. Ao todo foram 5 participantes que elaboraram os modelos em cada cenário. Os 5 participantes ao concluírem os modelos criados para cada um dos cenários, selecionaram o modelo que mais chegasse ao ideal entre todos que foram criados de cada uma das 4 funcionalidades/cenários, ou seja, cada funcionalidade do projeto que foi selecionado, teve um modelo ideal criado pelos 5 participantes do estudo de caso.

Após os participantes concluírem os modelos, foi gerado os modelos através da execução da técnica TechREF onde foi gerado os modelos de *features* de cada uma das funcionalidades criadas. A técnica TechREF gerou modelos para cada cenário criado. Com as métricas definidas e com os dois modelos criados, o modelo gerado e o modelo ideal, executase o software de avaliação com as métricas definidas. Realiza-se comparações entre as métricas e então são aplicadas as fórmulas de precisão, *recall* e *f-measure* para mensurar a eficácia do modelo gerado. Na Figura 39 é ilustrado o processo de execução do estudo de caso.

**Figura 39 – Processo de execução do estudo de caso**



Fonte: Elaborado pelo autor.

Os modelos gerados pela técnica TechREF serão denominados na avaliação de modelos gerado em *MG* e os modelos ideais criados pelos participantes do estudo de caso serão denominados de modelos ideais em *MI*. Assim na comparação dos modelos com as métricas, será utilizada esta nomenclatura na avaliação.

### 6.2.2 Cenários do estudo de caso

Com o objetivo de avaliar a técnica, foram criados 4 cenários para avaliar a eficiência da técnica implementada. Estas 4 implementações remetem a funcionalidades do sistema, isto é, representam *features* do sistema. Através da elaboração dessas implementações, os modelos ideais *MI* que serão desenvolvidos pelos participantes do estudo de caso e os modelos que serão gerados pela técnica serão avaliados com métricas. Cada cenário elaborado contém um modelo ideal *MI*. Desta forma é possível averiguar quanto a técnica se faz precisa na construção do modelo gerado *MG*. Abaixo são detalhados os cenários:

- **Cenário 1 – Adicionar característica da alimentação.** O primeiro cenário é referente a um novo atributo de classe. Esta nova característica será avaliada pela quantidade de atributos na classe. Com a inclusão deste novo atributo na classe, a nova informação será observada na avaliação se na geração do modelo o atributo estará contido no modelo que será gerado *MG*. Na implementação do cenário com o modelo ideal *MI*, o modelo deverá conter o novo atributo no projeto escolhido.
- **Cenário 2 – Incluir a idade nas informações do animal de estimação virtual.** O segundo cenário deverá ser adicionado um novo método nas classes que compõem as informações do animal. Esta nova informação deverá aparecer na tela do usuário com uma descrição “Idade”. Neste cenário será avaliado o número de operações das classes envolvidas, comparando o modelo gerado *MG* e o modelo ideal *MI*.

- **Cenário 3 – Modificar o tempo do jogo de dias para anos.** Neste cenário é realizada uma modificação de projeto onde irá impactar em atributos e métodos do sistema. O terceiro cenário deverá modificar o tempo de contagem do jogo de dias para anos, isto é, ao invés do jogar funcionar com a contagem do período de animal em dias, deverá ser anual, fazendo a contagem da idade do animal no decorrer do tempo. Com isto, este cenário irá avaliar a modificação dos elementos que compõem os modelos.
- **Cenário 4 – Visualizar os itens para compra.** Por fim, o último cenário tem como objetivo visualizar os itens disponíveis para comprar. Neste cenário o usuário irá visualizar todos os itens disponíveis na opção de shop do jogo. Para abrir a visualização dos itens, o usuário deverá clicar no ícone de compras. Os itens disponíveis para compra serão apresentados em duas categorias: brinquedos e comidas. Na apresentação dos itens, irá ser apresentado as características dos itens que foram inseridas no cenário 1. Com isto, é possível avaliar as modificações realizadas nas classes.

### 6.2.3 Perfis dos participantes

A seleção dos participantes foi realizada através de um questionário de participação com algumas questões sobre aspectos acadêmicos e profissionais. Dentre os que responderam as questões e que optaram em participar do estudo de caso deste trabalho, foram selecionados 5 participantes. Dos participantes selecionados 1 tem faixa etária entre 25 a 29 anos, 3 entre 30 a 34 e 1 entre 35 a 39 anos. A Tabela 11 ilustra a faixa etária dos participantes.

**Tabela 11 – Faixa etária dos participantes**

<b>Faixa etária</b>	<b>Participantes</b>
Entre 25 a 29 anos	1
Entre 30 a 34 anos	3
Entre 35 a 39 anos	1

Fonte: elaborado pelo autor.

Para realizar o estudo de caso, foram selecionados 5 participantes com formação superior completa em Ciência da Computação, Sistemas de Informação e Análise de sistemas, conforme está detalhado na Tabela 12, que representa o grau de escolaridade dos participantes.

**Tabela 12 – Formação acadêmica**

<b>Formação acadêmica</b>	<b>Participantes</b>
Sistema de informação	2 participantes
Ciências da computação	2 participantes
Análise de sistemas	1 participante

Fonte: elaborado pelo autor.

Todos participantes ocupam empregos profissionais em empresas de grande porte. Dos 5 participantes selecionados como voluntários para o estudo de caso, 2 atuam como programadores/desenvolvedores de sistemas, 1 atua como analista de sistemas, 2 atuam como arquiteto de sistemas, conforme descrito na Tabela 13.

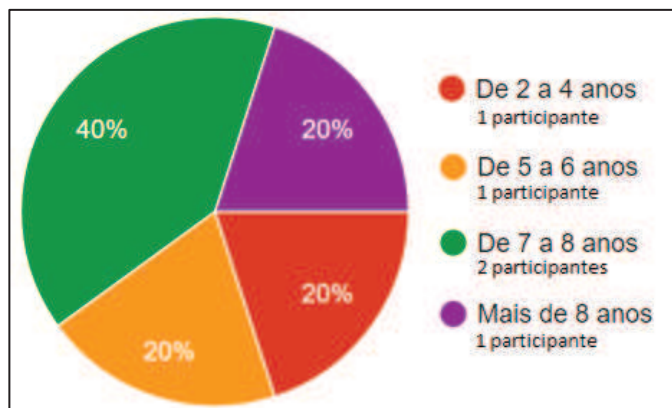
**Tabela 13 – Cargo dos participantes ou especialidades de ocupação**

Cargo/Especialidade	Participantes
Programador	2 participantes
Analista	1 participante
Arquiteto de software	2 participantes

Fonte: elaborado pelo autor.

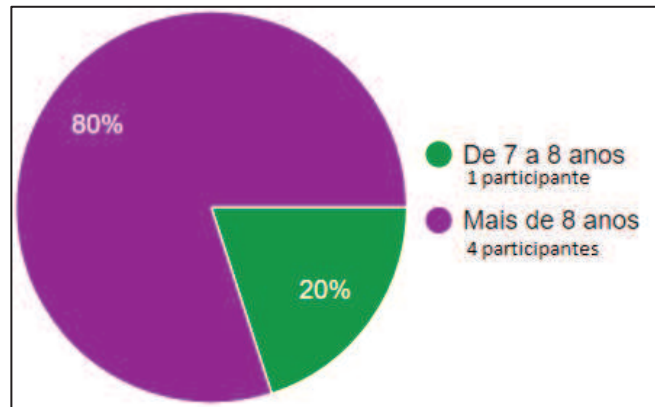
Os participantes do estudo de caso responderam sobre o tempo de estudos em universidades. 1 participante estudou entre 2 a 4 anos, 1 participante estudou em universidades entre 5 a 6 anos, 2 participantes estudaram de 7 a 8 anos e 1 participante estudou mais de 8 anos. Na Figura 40 abaixo é ilustrado o percentual dos participantes por tempo de estudos em universidades.

**Figura 40 – Tempo de estudos em universidades**



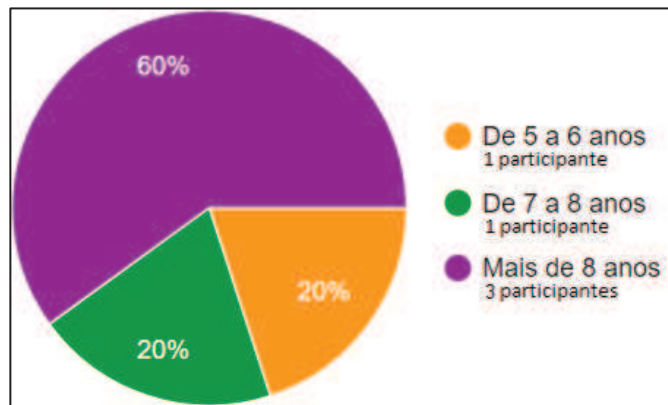
Fonte: elaborado pelo autor.

Sobre a experiência dos participantes em desenvolvimento de software apenas 1 participante dos que foram selecionados não tem mais do que 8 anos de experiência. Mesmo assim, este participante tem experiência entre 7 a 8 anos em desenvolvimento de software. 4 dos participantes tem mais de 8 anos de experiência em desenvolvimento de software. Conforme é ilustrado na Figura 41 abaixo, 80% dos integrantes que fizeram parte do estudo de caso tem experiência sênior em desenvolvimento de software.

**Figura 41 – Tempo de experiência em desenvolvimento de software**

Fonte: elaborado pelo autor.

Com relação a experiência dos participantes em modelagem de software, 1 participante tem experiência de 5 a 6 anos, 1 participante tem de 7 a 8 anos de experiência em modelagem de software. Os outros 3 participantes do estudo de caso têm mais de 8 anos de experiência em modelagem de software. Abaixo na Figura 42 é ilustrado os dados referentes aos participantes no que diz respeito a experiência em modelagem de software.

**Figura 42 – Tempo de experiência em modelagem de software**

Fonte: elaborado pelo autor.

Após fazer o levantamento dos perfis dos participantes, é possível destacar o grau de envolvimento dos participantes em relação a desenvolvimento de software e modelagem de software. É possível verificar que praticamente todos os participantes atuam em áreas da tecnologia da informação há muitos anos e que o nível de experiência dos participantes tanto para desenvolvimento como para modelagem de software é bastante significativo. Desta forma, caracteriza uma qualidade na seleção dos participantes e bons perfis para a avaliação do estudo de caso.

#### 6.2.4 Seleção das Métricas

A seleção das métricas foi realizada com o objetivo de avaliar as mudanças em novas *features*. As métricas foram selecionadas por enumerar aspectos das classes e características do código-fonte, exemplo, números de atributos de uma classe ou de operações. Com as

métricas possível encontrar as inconsistências nas classes geradas e identificar possíveis aspectos que não se assemelham entre o modelo gerado e o modelo ideal.

Com isto, foram selecionadas métricas referentes aos números de elementos das classes envolvidas nos modelos. Cada métrica diz respeito a uma avaliação. Foram selecionadas 6 métricas para avaliar a TechREF, logo abaixo estão ilustradas na tabela as métricas escolhidas e a nomenclatura das métricas que serão utilizadas como avaliação dos *MG* e no *MI*. A Tabela 14 apresenta a relação das métricas selecionadas para este estudo de caso.

**Tabela 14 – Tabela com a relação de métricas**

	<b>Métrica</b>	<b>Descrição</b>
1	NumClass	Lista o número de classes do modelo.
2	NumAttr	Lista o número de atributos da classe.
3	NumOps	Lista o número de operações da classe.
4	NumPubOps	Lista o número de operações públicas da classe.
5	Setters	Lista o número de operações com o nome “set”.
6	Getters	Lista o número de operações com o nome “get”, “is” ou “has”.

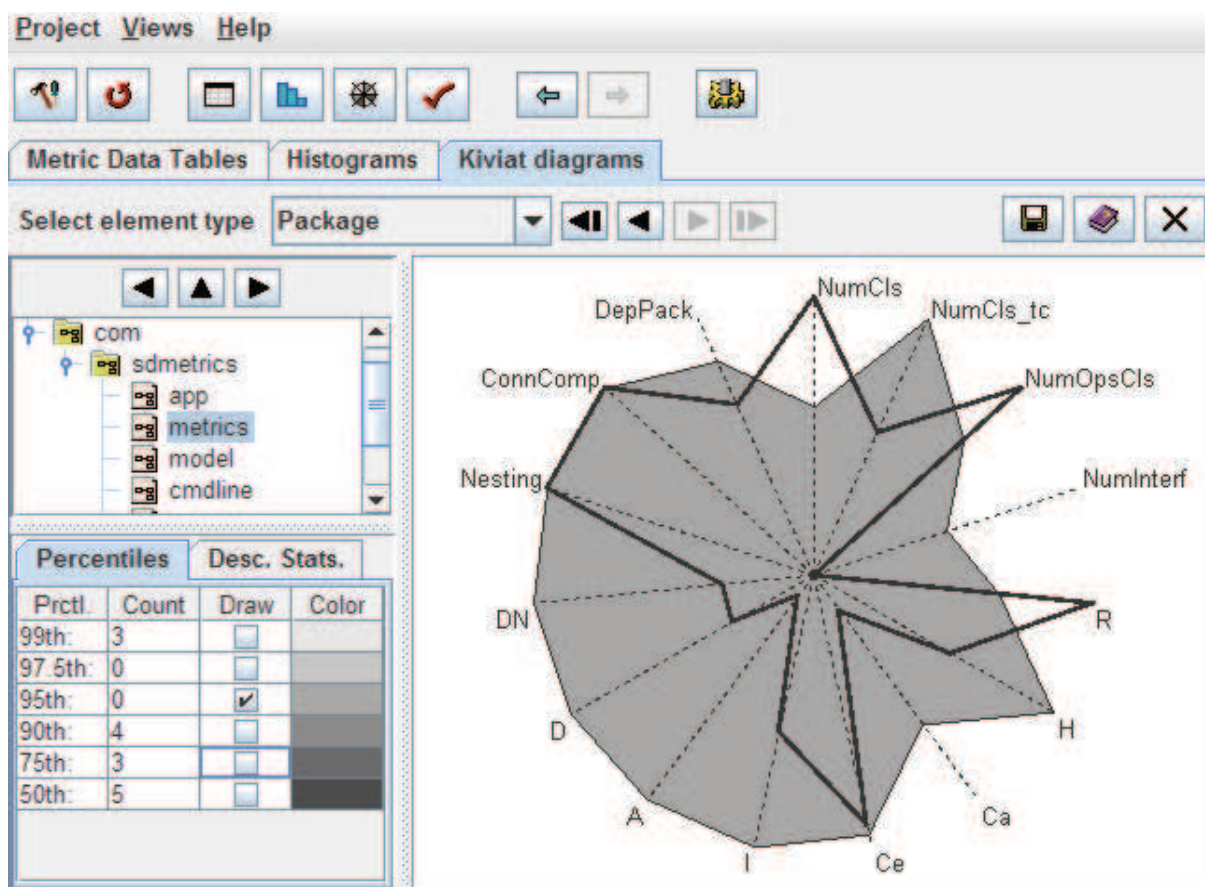
Fonte: elaborado pelo autor.

### 6.2.5 SDMetrics

SDMetrics é uma ferramenta de medição de qualidade de design de software para UML. O SDMetrics verifica as regras de design para detectar automaticamente o design incompleto ou incorreto. Medidas de propriedades estruturais são frequentemente aplicadas ao código-fonte para extrair os resultados das métricas. As métricas cobrem informações coletadas de diagramas de estrutura de classe, pacote, objeto e composição (SDMetrics, 2017). É uma ferramenta de medição de projeto muito completa, que analisa uma ampla gama de diagramas UML, incluindo diagrama de classes, caso de uso, diagrama de atividade e diagrama de estados, gerando várias métricas para cada tipo de diagrama. (FONTE, 2012)

O SDMetrics está desenvolvido na linguagem Java, e fornece uma interface de usuário gráfica para análise de arquivos *Xml Metadata Interchange* (XMI) (FONTE, 2012). A ferramenta permite acessar os resultados com diferentes visualizações, gráfica ou textual. O SDMetrics não avalia a qualidade do modelo, porém disponibiliza métricas para aplicadas no modelo. Estas métricas são detalhadas na documentação da ferramenta (SDMetrics, 2017). Abaixo é apresentado a interface da ferramenta SDMetrics.

Figura 43 – Ferramenta SDMetrics.



Fonte: SDMetrics, 2017

Para realizar a análise dos modelos gerados pela TechREF foram usadas formulas de precisão, recuperação e de média ponderada. A formula da precisão trata os valores positivos corretos previstos nos modelos gerados, isto é, a precisão calcula o número corretos dividido pelo número de todos os resultados. A recuperação trata com a exclusão dos valores que não foram gerados. A média harmônica ponderada é calculada com o valor da precisão e o valor calculado da recuperação (FAWCETT, 2006). Abaixo são ilustradas as formulas que foram usadas.

$$precisão = \frac{|MG \cap MI|}{|MG|}$$

$$recuperação = \frac{|MG \cap MI|}{|MI|}$$



$$f\text{-measure} = \frac{2 \cdot \text{precisão} \cdot \text{recuperação}}{\text{precisão} + \text{recuperação}}$$

### 6.3 Resultados do estudo de caso

Conforme apresentado o detalhamento da estrutura do trabalho na Seção 6.2, esta seção apresenta os resultados obtidos da TechREF. Com o apoio da ferramenta SD Metrics (SD Metrics, 2017) foi possível realizar a comparação entre o modelo gerado (MG) com o modelo ideal (MI). Para avaliação deste trabalho, foi usada como apoio a ferramenta SD Metrics. Esta ferramenta ajudou na automação das métricas de cada modelo. Após extrair as informações, os dados foram calculados em sua totalidade para avaliar cada métrica de cada cenário elaborado.

Cada métrica apresentada em cada cenário tem o valor total dos dados, por exemplo, a métrica 1 do cenário 1 tem a totalidade de classes do modelo gerado e do modelo ideal. Assim foi replicado para todas as demais métricas selecionadas para esta avaliação. Na Tabela 7 abaixo, são ilustrados os valores referentes aos modelos e os resultados brutos de cada modelo. A relação do modelo gerado (MG) com intersecção do modelo ideal (MI) é resultante dos valores encontrados em ambos modelos. Para medir a proporção dos elementos positivos corretamente previstos no modelo ideal foi aplicado a fórmula da precisão conforme apresentado na Seção 6.2.5. Também foi aplicado a fórmula de recuperação e média ponderada, que representam a recuperação dos elementos avaliados e média das duas fórmulas utilizadas na avaliação.

**Tabela 15 – Tabela com os resultados das métricas do cenário 1**

Cenário 1		MG	MI	MG ∩ MI	Precision	Recall	f-measure
	NumClass	3	3	3	1,00	1,00	1,00
NumAttr	35	35	35	1,00	1,00	1,00	
NumOps	22	23	22	0,96	1,00	0,98	
NumPubOps	16	16	16	1,00	1,00	1,00	
Setters	1	1	1	1,00	1,00	1,00	
Getters	10	10	10	1,00	1,00	1,00	

Fonte: elaborado pelo autor.

Com a apresentação da Tabela 15, percebe-se que houve uma grande similaridade entre os elementos do modelo gerado e os elementos do modelo ideal. É possível verificar que apenas a métrica 3 houve uma diferença entre o modelo gerado (MG) e o modelo ideal (MI). O segundo cenário deste trabalho, apresentou elementos em todas as métricas. Neste cenário as métricas 2, 3 e 4 não foram 100% precisas. Na Tabela 16 abaixo, é possível observar os elementos em todas as métricas. Igualmente ao cenário 1, o cenário 2 também houve uma grande similaridade dos dados entre os dois modelos.

**Tabela 16 – Tabela com os resultados das métricas do cenário 2**

Cenário 2		MG	MI	$MG \cap MI$	<i>Precision</i>	<i>Recall</i>	<i>F-Measure</i>
	NumClass	4	4	4	1,00	1,00	1,00
	NumAttr	19	20	19	0,95	1,00	0,97
	NumOps	37	41	37	0,90	1,00	0,95
	NumPubOps	31	35	31	0,89	1,00	0,94
	Setters	5	5	5	1,00	1,00	1,00
	Getters	16	16	16	1,00	1,00	1,00

Fonte: elaborado pelo autor.

A Tabela 17 abaixo ilustra os dados referente ao cenário 3. Com valores muito próximos aos valores do modelo idealizado dito como correto pelos participantes deste estudo, a geração do modelo pela TechREF apresentou bons resultados para o cenário 3. Abaixo é possível observar que somente a métrica 3 e 4 não obterão valores iguais a 1 na precisão dos modelos.

**Tabela 17 – Tabela com os resultados das métricas do cenário 3**

Cenário 3		MG	MI	$MG \cap MI$	<i>Precision</i>	<i>Recall</i>	<i>F-Measure</i>
	NumClass	2	2	2	1,00	1,00	1,00
	NumAttr	23	23	23	1,00	1,00	1,00
	NumOps	11	13	11	0,85	1,00	0,92
	NumPubOps	7	8	7	0,88	1,00	0,93
	Setters	1	1	1	1,00	1,00	1,00
	Getters	2	2	2	1,00	1,00	1,00

Fonte: elaborado pelo autor.

No último cenário deste estudo, observa-se a continuidade dos valores muito próximos aos valores idealizados. Os valores extraídos dos modelos gerados pela TechREF obtiveram valores muito precisos comparados aos valores dos modelos idealizados. Neste cenário não houve valores para a métrica 5, deixando a métrica sem resultado. Com isto, os valores ficaram zerados neste item para o cenário 4. Abaixo na Tabela 18 é ilustrado os valores do último cenário do estudo de caso.

**Tabela 18 – Tabela com os resultados das métricas do cenário 4**

Cenário 4		MG	MI	$MG \cap MI$	<i>Precision</i>	<i>Recall</i>	<i>F-Measure</i>
	NumClass	3	3	3	1,00	1,00	1,00
	NumAttr	14	14	14	1,00	1,00	1,00
	NumOps	12	15	12	0,80	1,00	0,89
	NumPubOps	12	15	12	0,80	1,00	0,89
	Setters	0	0	0	-	-	-
	Getters	9	9	9	1,00	1,00	1,00

Fonte: elaborado pelo autor.

Após a apresentação dos resultados do experimento controlado e dos resultados do estudo de caso, será realizada na próxima seção 6.4 a análise dos resultados referentes as questões de pesquisa deste trabalho. A análise da avaliação realizada neste trabalho será para responder as duas questões deste trabalho.

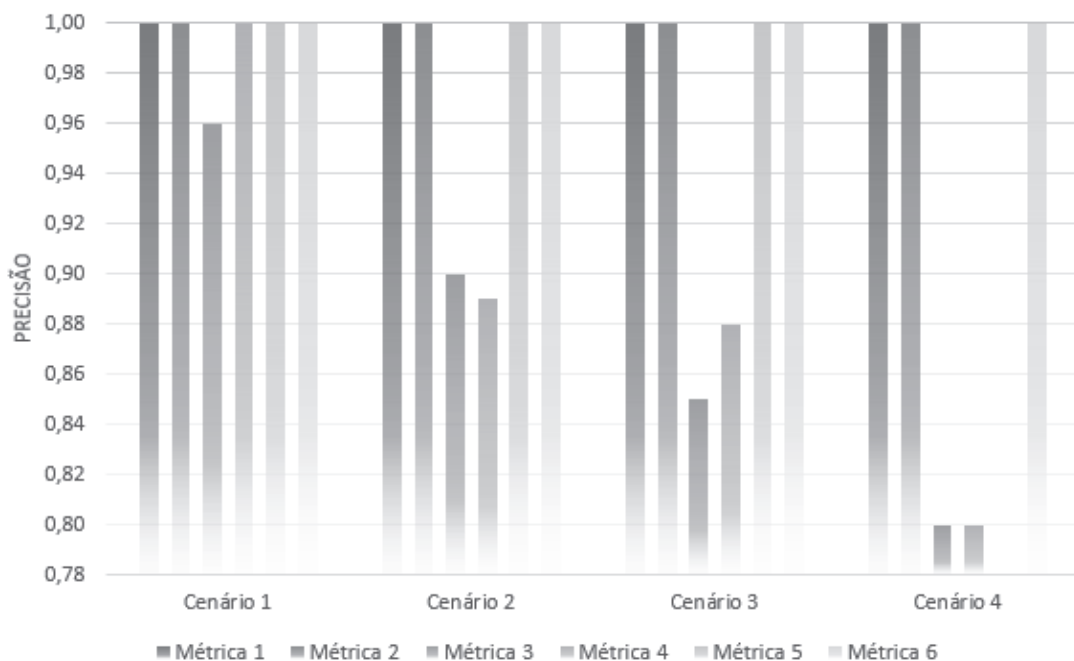
## 6.4 Análise dos resultados

Visto que a avaliação do experimento controlado que foi realizado na Seção 5.1, serão realizadas nesta seção as análises referentes as questões de pesquisa ao estudo de caso. Os resultados analisados buscam responder as questões referentes ao desenvolvimento de software e da automação de modelos orientados a *features* a partir da geração dos modelos pela TechREF. As análises das questões de pesquisa detalhadas na Seção 1.2 deste trabalho, buscam informações importantes e relevantes para as respostas que serão elaboradas dentro desta dissertação.

A segunda questão de pesquisa tem o objetivo de avaliar a forma automática de geração de modelos de *features*. Esta forma foi realizada através da criação de uma técnica que utiliza conceitos para gerar o modelo orientado a *features* através do código-fonte realizando uma engenharia reversa de código em modelo UML. Esta avaliação da técnica foi realizada com a ajuda de uma ferramenta juntamente com formulas de precisão, recuperação e média apresentada na Seção 6.2.5.

Após os resultados serem calculados, foi possível realizar a avaliação dos valores encontrados para a precisão dos modelos gerados pela TechREF. Na Figura 44 é possível observar os valores da precisão da técnica em cada cenário e pelas métricas que foram elaboradas.

**Figura 44 – Precisão das métricas por cenários**



Fonte: elaborado pelo autor.

Conforme apresentado na Figura 44 acima, é possível observar o volume alto do índice de precisão da técnica. Para a consideração de uma boa precisão os valores próximos

de 1 é representado um ótimo valor para caracterizar a precisão. Nos cenários apresentados no gráfico, em algumas métricas é possível analisar que a precisão da técnica apresentou bons resultados. Embora os gráficos ilustrem a tendência muito boa em relação a precisão da técnica e os resultados apresentados indicam também uma grande similaridade do modelo gerado com o modelo idealizado, foi realizado um novo teste avaliando os valores de precisão gerados através dos resultados obtidos da técnica TechREF.

Foi utilizado o *One-Sample Z-test*, este teste compara os valores de precisão ao valor hipotético para anular a hipótese. Neste teste, foi possível avaliar o valor de 1,00 para a hipótese nula. Com esse cenário foi obtido os seguintes valores para o estudo de caso. Abaixo a Tabela 19 apresenta os valores referente ao teste Z-test.

**Tabela 19 – Resultado precisão - Z-test**

Teste One-Sample Z-test	
<i>Difference</i>	-0,032
<i>z (Observed value)</i>	-2,244
<i> z  (Critical value)</i>	1,960
<i>p-value (Two-tailed)</i>	< 0,025
<i>Alpha</i>	0,05

Fonte: elaborado pelo autor.

Com o nível de confiança de 95%, o *p-value* apresentou um valor menor do que 0,05 no teste *One-Sample Z-test*. Desta forma, indica-se uma evidência de rejeição da hipótese nula. O valor apresentado ao realizar o teste *One-Sample Z-test* foi de 2,49% de chance para a hipótese seja verdade, ou seja, os valores apresentados ilustram a similaridade dos modelos gerados pela técnica TechREF. A questão de pesquisa relacionada a esta hipótese de automação dos modelos de *features* responde a segunda questão de pesquisa deste trabalho.

## 6.5 Avaliação final

Com os resultados apresentados juntamente com análise dos dados, foi possível responder as questões de pesquisa elaboradas para este trabalho. O experimento controlado produziu importantes resultados que obtiveram confirmação na análise realizada. Também na análise do estudo de caso indicaram bons resultados fazendo com que ambas questões de pesquisa tivessem a hipótese nula rejeitada.

Os testes estatísticos realizados dos dois cenários deste trabalho, o experimento controlado e o estudo de caso, indicaram valores que rejeitaram as hipóteses nulas das questões de pesquisas apresentadas na Seção 1.2 deste trabalho. A análise realizada sobre os resultados do experimento controlado, levantaram fortes evidências sobre os dados apresentados nas seções referentes aos testes estatísticos de Wilcoxon. Desta forma, os valores obtidos através dos testes foram importantes para consolidar os resultados da hipótese alternativa, salientando o ganho no esforço quando utilizado os modelos de *features* dentro do processo de manutenção do desenvolvimento de software.

Para a segunda questão de pesquisa deste trabalho, foi realizado o estudo de caso com a técnica de engenharia reversa orientada a *features*. Que através dos modelos gerados pela técnica obteve resultados importantes e indicativos sobre a geração de modelos similares aos

modelos idealizados de *features*. Com os testes de precisão foi possível ter indicativos de que os modelos gerados foram satisfatórios comparados aos modelos ideais. Os testes estáticos comprovaram a precisão da técnica, rejeitando a hipótese nula da segunda questão de pesquisa. Desta forma, a hipótese alternativa da segunda questão foi verdadeira na sua plenitude mostrando que a técnica pode ser utilizada como forma de geração de modelos de *features* de forma automática.

Com as questões de pesquisa respondidas e com os objetivos elaborados neste trabalho na Seção 1.4, que diz respeito a reduzir o esforço na manutenção de software, implementar uma técnica para a geração de modelos orientados a *features*, avaliar os resultados obtidos e produzir conhecimento empírico, comprova-se que o objetivo principal deste trabalho foi atingido.

Desta forma, ao atingir os objetivos definidos neste trabalho e concluir todas as etapas do processo de execução deste trabalho, é possível afirmar que o uso de modelos orientados a *features* na manutenção de software reduz o esforço dos usuários e o uso da técnica TechREF na geração de modelos orientados a *features* podem ajudar na manutenção dos sistemas dentro do processo de desenvolvimento do software e nas melhorias futuras.

## 6.6 Ameaças à validade do estudo

Ao realiza um experimento controlado é necessário validar os métodos e as escolhas utilizadas para o processo de execução do trabalho. Desta forma, foram aplicados e avaliados os resultados encontrados em quatro subcategorias de ameaças à validade do experimento, são elas:

- **Ameaças à validade interna:** Foram utilizados métodos para a confiabilidade das medidas utilizadas no experimento. A aplicação das tarefas foi realizada de forma controlada e elaboradas de forma transparente onde os participantes foram designados para a realização das tarefas de forma sequencial sem ordenação de indivíduos para grupos específicos. A separação dos indivíduos foi realizada de maneira classificatória para que houvesse o equilíbrio na proporção das tarefas realizadas. Os testes paramétricos foram aplicados com base em valores extraídos do experimento e sem a violação de dados ou alteração de valores. Os resultados apresentados foram os mesmos extraídos pelos testes realizados para responder as questões de pesquisa deste trabalho.
- **Ameaças à validade externa:** O experimento foi realizado em ambiente acadêmico com integrantes de programa de pós-graduação. Foram realizadas tarefas com tempos informados e cronometrados em cada tarefa. Antes do início da participação dos integrantes selecionados, foi realizada uma introdução explicativa de todas as atividades que seriam realizadas no experimento. As respostas dos integrantes que participaram das tarefas do experimento foram gravadas por vídeo, capturando a tela do computador.
- **Ameaças à validade de construção:** O uso de uma documentação sobre a linguagem de programação foi informado anteriormente ao início das atividades por forma de questionário. Todas as tarefas que foram realizadas, foram previamente discutidas e explicadas para serem implementadas. Cada uma das questões tinha uma breve descrição e acompanhado de modelos criados para auxiliar na conclusão da atividade.

- **Ameaças à validade de conclusão:** O baixo poder estatístico nos testes pode levar a conclusões falsas, ao não rejeitar a hipótese nula. O experimento foi realizado com um grupo de 10 pessoas, totalizando um número de 30 amostras. O estudo buscou eliminar esse risco na fase de projeto tentando aumentar o número de participantes e escolhendo os testes estatísticos mais adequados para os resultados.

## 7 CONCLUSÃO

Com as questões abordadas ao longo deste trabalho, foi apresentado que a modelagem de sistemas carece de melhores soluções sobre uso de modelos orientado a *features*. No entanto, as respostas que foram obtidas no experimento controlado e no estudo de caso mostram que o uso de modelos UML apresentam benefícios. Com esta pesquisa, não foi possível afirmar fortemente que os resultados apresentados mostrem uma grande vantagem no uso de modelos orientados a *features*. Porém, é possível afirmar que houve um ganho embora pequeno sobre o modelo tradicional. Muitos problemas das ferramentas de modelagem serão solucionados através de modelos gerados orientados a *features* de forma automática e capazes de agilizar a produtividade e reduzir o esforço da atualização e qualidade do software.

Assim, esta pesquisa pode ser encarada como uma referência que investiga o efeito do uso de técnicas orientadas a *features* para avaliar modelos gerados. Além disso estudos empíricos serão necessários para entender melhor se estes resultados que serão confirmados na medida da realização de replicações de experimentos controlados utilizando a ferramenta proposta neste trabalho. A realização de experimentos controlado fomenta várias questões que precisam ser abordadas em relação delineamento experimental, instrumentação, medição e execução prática.

Os resultados obtidos na validação do experimento controlado não foram significativamente positivos para uma avaliação conclusiva sobre o uso de modelos orientados a *features*. Os dados tendem a uma melhora relativa na compreensão dos modelos gerados a partir dos modelos gerados. Porém também não é fato de que essas pequenas e significativas contribuições possam mostrar que existe uma pequena vantagem no que diz respeito ao uso de modelos orientados a *features*. Com relação a corretude das tarefas, foi possível identificar uma similaridade entre os modelos aplicados no experimento, o que comprova no ponto de vista que ambos modelos utilizados trazem o mesmo benefício. Porém, é válido aplicar mais testes e com um maior número de participantes para averiguar o resultado em uma escala maior.

Por fim, o desejo é de aumentar os resultados e dados coletados para que possam ser mais relevantes para a conclusão e mais precisa análise sobre o uso de modelos orientados a *features*. A avaliação conclusiva deste trabalho é satisfatória, porém não conclusiva, isso acrescenta as condições de que é possível avançar ainda mais com novas formas de avaliar experimentos controlados, estudos de caso e novas métricas de avaliação que poderão comprovar as vantagens dos modelos orientados a *features*.

### 7.1 Contribuições

Este trabalho contribuiu para a pesquisa sobre o uso de modelos orientados a *features*. Com a implementação de uma ferramenta de engenharia reversa capaz de gerar de forma automática modelos orientados a *features*, foi possível avaliar e gerar resultados que contribuíram para a discussão sobre UML. Além da técnica que foi apresentada como contribuição neste trabalho, também foi possível contribuir para a pesquisa empírica de novas soluções onde técnica podem mensurar o esforço e a corretude de tarefas que utilizam modelos orientados a *features*.

Por fim, o estudo gerou conhecimento referente a novos experimentos controlados e a estudos de caso para avaliação de técnica de engenharia reversa orientada a *features*. O

conjunto de informações apresentadas neste trabalho reuniu dados de pesquisa qualitativas e quantitativas comprovando que esta pesquisa pode ser uma referência para demais trabalhos futuros.

## 7.2 Limitações e trabalhos futuros

Este trabalho abordou algumas questões envolvendo modelos UML orientados a *features*. As contribuições deste trabalho foram salientadas durante o trabalho apresentado, porém, é necessário que mais pesquisa que possam fomentar este assunto e comprovar os resultados que foram encontrados nesta pesquisa. A gama de possibilidades de criação de novos cenários para esta pesquisa é imensa, porém não há tempo hábil para que seja adicionada neste trabalho todas as possibilidades. Com isto, é apresentado abaixo as limitações desta pesquisa e pontos francos que possam ser discutidos em novas pesquisas futuras, são elas:

- **Incluir demais modelos UML na técnica.** A técnica TechREF foi desenvolvida com o objetivo de gerar diagramas de classes. Com o propósito de atender as métricas que foram definidas, foi desenvolvido apenas a geração automática dos diagramas de classes. De modo que para avaliar os cenários, a implementação dos demais modelos que fazem parte da definição OMG podem ser adicionadas e implementados na técnica. Porém novas métricas deverão ser adicionadas e novos testes deverão ser realizados para avaliar os resultados.
- **Atender outras definições da Notação UML 2.0.** As notações definidas na UML são importantes para a boa compreensão do projeto. Os diagramas de classes gerados pela técnica atenderam algumas definições. Pode ser adicionada à TechREF novas implementações e definições de notação UML 2.0.
- **Adicionar cenários de validação.** É possível a realização de novos testes paramétricos de não paramétricos dos resultados que foram extraídos nesta pesquisa. Diversos testes podem ser aplicados uma vez que cada teste é importante e avalia um cenário diferente. Para a avaliação desta pesquisa se utilizou os testes que foram importantes para o resultado esperado, podendo ser realizado outros testes.
- **Aumentar a escala de participantes.** A aplicação do experimento controlado foi realizada dentro da universidade com alunos de pós-graduação. Pode-se aumentar e estender o experimento para outras universidades ou até mesmo para o público em geral. Porém é necessário a realização de filtros e seleção mais apurada dos participantes para se manter o equilíbrio e a organização do experimento. É válido aumentar os resultados desta pesquisa para averiguar se o aumento da população impacta nos resultados obtidos.



## REFERÊNCIAS

- Agile Modeling.** UML 2 Sequence Diagrams: An Agile Introduction. Disponível em: <<http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>>. Acesso em: 15 jul 2017.
- ALALFI, Manar H.; CORDY, James R.; DEAN, Thomas R. **Automated reverse engineering of UML sequence diagrams for dynamic web applications.** In: Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on. IEEE, 2009. p. 287-294.
- ALI, Noor Azian Mohamad; BANGOURA, Sekou Oumar. **Analysis between argo uml and star uml.** International Journal Of Applied Science Engineering And Management. (IJASEM). 2015, v. 1.
- ARGO UML.** Disponível em: <<http://argouml.tigris.org>>. Acesso em: 15 abr. 2017.
- ASTAH,** Reference Manual. Disponível em :< <http://astah.net/tutorials/>>. Acesso em: 26 mai. 2017.
- BAIDADA, Chafik; JAKIMI, Abdeslam. **Towards new hybrid approach of the reverse engineering of UML sequence diagram.** In: Information Science and Technology (CiSt), 2016 4th IEEE International Colloquium on. IEEE, 2016. p. 164-168.
- BASIL, V. et al. **The Goal Question Metric Paradigm: Encyclopedia of software engineering.** Wiley, New York. 1994.
- BENAVIDES, David et al. **Automated analysis in feature modelling and product configuration.** In: International Conference on Software Reuse. Springer, Berlin, Heidelberg, 2013. p. 160-175.
- BENAVIDES, David; SEGURA, Sergio; RUIZ-CORTÉS, Antonio. **Automated analysis of feature models 20 years later: A literature review.** Information Systems, v. 35, n. 6, p. 615-636, 2010.
- BERGMAYR, Alexander et al. fREX: fUML-based reverse engineering of executable behavior for software dynamic analysis. In: **Modeling in Software Engineering (MiSE), 2016 IEEE/ACM 8th International Workshop on.** IEEE, 2016. p. 20-26.
- BRIAND, Lionel C.; LABICHE, Yvan; LEDUC, Johanne. **Toward the reverse engineering of UML sequence diagrams for distributed Java software.** IEEE Transactions on Software Engineering, v. 32, n. 9, p. 642-663, 2006.
- BRUNELIERE, Hugo et al. **Modisco: A model driven reverse engineering framework.** Information and Software Technology, v. 56, n. 8, p. 1012-1032, 2014.
- CHANTRE, A., et al. **Manual de Técnicas de Desenvolvimento de Sociedade de Informação.** Cabo Verde: Josefa Lopes, 2008.

CHAUDRON, Michel RV; HEIJSTEK, Werner; NUGROHO, Ariadi. **How effective is UML modeling? An empirical perspective on costs and benefits.** Software & Systems Modeling, v. 11, n. 4, p. 571-580, 2012.

CHOUAMBE, Landry; KLATT, Benjamin; KROGMANN, Klaus. **Reverse engineering software-models of component-based systems.** In: Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on. p. 93-102. IEEE, 2008.

DECKER, Michael John et al. **A Tool for Efficiently Reverse Engineering Accurate UML Class Diagrams.** In: Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on. IEEE, 2016. p. 607-609.

DZIDEK, Wojciech J.; ARISHOLM, Erik; BRIAND, Lionel C. **A realistic empirical evaluation of the costs and benefits of UML in software maintenance.** IEEE Transactions on software engineering, v. 34, n. 3, p. 407-432, 2008.

EMF. **Eclipse Modeling Framework (EMF).** Disponível em: <https://www.eclipse.org/modeling/emf/>. Acesso em: 15 de jul. 2017.

**Extended Feature models.** Disponível em <[https://www.utwente.nl/ewi/trese/b\\_referaat/broek1/](https://www.utwente.nl/ewi/trese/b_referaat/broek1/)>. Acesso em: 16 mar. de 2017.

FAITELSON, David; TYSZBEROWICZ, Shmuel. **UML diagram refinement (focusing on class-and use case diagrams).** In: Proceedings of the 39th International Conference on Software Engineering. p. 735-745. IEEE Press, 2017.

FARIAS, Kleinner et al. **Evaluating the effort of composing design models: a controlled experiment.** Software & Systems Modeling, v. 14, n. 4, p. 1349-1365, 2015.

FAWCETT, Tom. **An introduction to ROC analysis.** Pattern recognition letters, v. 27, n. 8, p. 861-874, 2006.

FERNÁNDEZ-SÁEZ, Ana M. et al. **Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: a controlled experiment.** In: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering. p. 60-71. ACM, 2013.

FERNÁNDEZ-SÁEZ, Ana M. et al. **Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study.** Information and Software Technology, v. 55, n. 7, p. 1119-1142, 2013.

FONTE, Daniela et al. **Modeling Languages: metrics and assessing tools.** arXiv preprint arXiv:1206.4477, 2012.

GLASS, Robert L. **Facts and fallacies of software engineering.** Addison-Wesley Professional, 2002.

GOETTEN, Junior; WINCK, Diogo. **AspectJ - Programação Orientada a Aspectos com Java.** São Paulo - SP: Novatec Editora, 2006.

HASSAN, Shoaib et al. **Software Reverse Engineering to Requirement Engineering for Evolution of Legacy System**. In: IT Convergence and Security (ICITCS), 2015 5th International Conference on. IEEE, 2015. p. 1-4.

**Java Framework Portal**. O que é Programação Orientada a Aspectos? Disponível em: <<http://www.javaframework.org/portal/2010/04/14/o-que-programacao-orientada-a-aspectos/>>. Acesso em: 22 jul 2017.

KAPPLER, Thomas et al. **Towards Automatic Construction of Reusable Prediction Models for Component-Based Performance Engineering**. Software Engineering, v. 121, p. 140-154, 2008.

KEELE, Staffs. **Guidelines for performing systematic literature reviews in software engineering**. In: Technical report, Ver. 2.3 EBSE Technical Report. EBSE. sn, 2007.

KICZALES, Gregor et al. **An overview of AspectJ**. In: European Conference on Object-Oriented Programming. p. 327-354. Springer Berlin Heidelberg, 2001.

KITCHENHAM, Barbara A.; DYBA, Tore; JORGENSEN, Magne. **Evidence-based software engineering**. In: Proceedings of the 26th international conference on software engineering. IEEE Computer Society, 2004. p. 273-281.

LADDAD, Ramnivas. **AspectJ in action: practical aspect-oriented programming**. Dreamtech Press, 2003.

LANGE, Christian FJ et al. **An experimental investigation of UML modeling conventions**. In: International Conference on Model Driven Engineering Languages and Systems. p. 27-41. Springer Berlin Heidelberg, 2006.

MALTA, Mariana Curado; CENTENERA, Paloma; GONZALEZ-BLANCO, Elena. **Using Reverse Engineering to Define a Domain Model: The Case of the Development**. Developing Metadata Application Profiles, p. 146, 2017.

MSDN, Microsoft. **File Caching**. Disponível em <[https://msdn.microsoft.com/en-us/library/windows/desktop/aa364218\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa364218(v=vs.85).aspx)>. Acesso em: 11 abr. 2017.

NANTHAAMORNPHONG, Aziz et al. **Extracting uml class diagrams from object-oriented fortran: Foruml**. Scientific Programming, v. 2015, p. 1, 2015.

NARANG, Rajesh. **Software Engineering—Principles and Practices**. McGraw-Hill Education, 2015.

NUGROHO, Ariadi; CHAUDRON, Michel RV. **Evaluating the impact of UML modeling on software quality: An industrial case study**. In: International Conference on Model Driven Engineering Languages and Systems. p. 181-195. Springer Berlin Heidelberg, 2009.

NUNES, Mauro; O'NEILL, Henrique. **Fundamental de UML**. 2004.

OSMAN, Hafeez; CHAUDRON, Michel RV. **Correctness and Completeness of CASE Tools in Reverse Engineering Source Code into UML Model**. GSTF Journal on Computing (JoC), v. 2, n. 1, 2014.

PETRE, Marian. **UML in practice**. In: Proceedings of the 2013 International Conference on Software Engineering. p. 722-731. IEEE Press, 2013.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software-8ª Edição**. McGraw Hill Brasil, 2016.

RIVA, Claudio et al. **UML-based reverse engineering and model analysis approaches for software architecture maintenance**. In: Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on. p. 50-59. IEEE, 2004.

SADAF, Sadia; ATHAR, Ali; AZAM, Farooque. Evaluation of FED-CASE-A Tool to Convert Class Diagram into Structural Coding. In: **Computer, Consumer and Control (IS3C), 2016 International Symposium on**. IEEE, 2016. p. 184-187.

**Safari Book Online**. Sequence Diagrams. Disponível em: <<http://my.safaribooksonline.com/book/software-engineering-and-development/uml/0321193687/sequence-diagrams/ch04>>. Acesso em: 10 jul 2017.

SCANNIELLO, Giuseppe et al. **On the impact of UML analysis models on source-code comprehensibility and modifiability**. ACM Transactions on Software Engineering and Methodology (TOSEM), v. 23, n. 2, p. 13, 2014.

SDMetrics. **SDMetrics: The Software Design Metrics tool for the UML**. Disponível em: <https://www.sdmetrics.com/down/SDMetricsPresentation.pdf>. Acesso em: 20 de out. 2017.

SOARES, Sérgio; BORBA, Paulo. **Implementing modular and reusable aspect-oriented concurrency control with aspectj**. In: WASP, Uberlandia, Brazil, 2005.

**StarUML**. Disponível em: < <http://staruml.io/>>. Acesso em: 10 jul 2017.

STEINMACHER, Igor et al. **GeCA: Uma Ferramenta de Engenharia Reversa e Geração Automática de Código**. Simpósio Brasileiro de Sistemas de Informação (SBSI) 2006 Proceedings (em cd). Curitiba, 2006.

**The AspectJ Team**. Disponível em: <<http://www.eclipse.org/aspectj/doc/released/progguide/index.html>>. Acesso em: 06 mai. 2017.

**Umbrello Uml Modeller**. Disponível em < <https://umbrello.kde.org/>>. Acesso em: 15 jun. 2016.

**UML Sequence Diagrams**. Disponível em: <<http://www.uml-diagrams.org/sequence-diagrams.html>>. Acesso em: 18 jul 2017.

**UML, OMG**. Formal Specifications: Unified Modeling Language, v 2.5, 2015.

**UML2. Model Development Tools (MDT)**. Disponível em: <http://www.eclipse.org/modeling/mdt/?project=uml2>. Acesso em: 20 de ago. 2017.

**UML-DIAGRAMS**. Disponível em: <https://www.uml-diagrams.org/class-diagrams-overview.html>. Acesso em: 09 de set. 2017.

**Vieira, Marcio Junior.** Modelagem UML com Software Livre. Disponível em <<http://www.ambientelivre.com.br/~marcio/UML5FISL.pdf>>. Acesso em: 17 mai. 2016

**VirtualPets**, Disponível em: <https://github.com/Andrew2789/SENG201-VirtualPets>. Acesso em: 10 ago 2017.

**Visual Paradigm**, What is Visual Paradigm? Disponível em <<https://www.visual-paradigm.com/features>>. Acesso em: 11 mai. 2016.

WAND, Mitchell; KICZALES, Gregor; DUTCHYN, Christopher. **A semantics for advice and dynamic join points in aspect-oriented programming.** ACM Transactions on Programming Languages and Systems (TOPLAS), v. 26, n. 5, p. 890-910, 2004.

WEIGERT, Thomas; WEIL, Frank. **Practical experiences in using model-driven engineering to develop trustworthy computing systems.** In: Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on. p. 8 pp. IEEE, 2006.



## ANEXO A – QUESTIONÁRIO DE SELEÇÃO DOS PARTICIPANTES.

### Questionário

O objetivo deste questionário é obter informações de pessoas que queiram participar de um experimento que está sendo realizado dentro do programa de mestrado em computação aplicada na área de engenharia de software. Esta pesquisa visa verificar a experiência na área de desenvolvimento de software e modelagem de software. Sua participação é muito importante para esta pesquisa! Muito Obrigado.

**\*Obrigatório**

1. Nome:

---

2. Idade: \*

---

3. Sexo: \*

*Marcar apenas uma oval.*

- Feminino  
 Masculino

4. Profissão: \*

---

5. Cargo: \*

---

6. Escolaridade (Maior grau completo): \*

*Marcar apenas uma oval.*

- Técnico  
 Graduação  
 Mestrado  
 Doutorado  
 Outro: \_\_\_\_\_

7. Formação acadêmica: \*

*Marcar apenas uma oval.*

- Sistemas de Informação  
 Ciências da Computação  
 Engenharia da Computação  
 Análise de Sistemas  
 Outro: \_\_\_\_\_

**8. Quanto tempo estudou (tem estudado) em universidades: \****Marcar apenas uma oval.*

- Menos de 2 anos
- De 2 a 4 anos
- De 5 a 6 anos
- De 7 a 8 anos
- Mais de 8 anos

**9. Seu cargo se encaixa em qual especialidade: \****Marcar apenas uma oval.*

- Programador
- Analista
- Arquiteto
- Gerente
- Outro: \_\_\_\_\_

**10. Quanto tempo você tem de experiência em DESENVOLVIMENTO de software: \****Marcar apenas uma oval.*

- Menos de 2 anos
- De 2 a 4 anos
- De 5 a 6 anos
- De 7 a 8 anos
- Mais de 8 anos

**11. Quanto tempo você tem de experiência com MODELAGEM de software: \****Marcar apenas uma oval.*

- Menos de 2 anos
- De 2 a 4 anos
- De 5 a 6 anos
- De 7 a 8 anos
- Mais de 8 anos
-