UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RICARDO THEIS GERALDI

SMartyCheck: uma Técnica de Inspeção baseada em *Checklist* para Diagramas de Casos de Uso e de Classes da Abordagem *SMarty*

Maringá
2015

RICARDO THEIS GERALDI

SMartyCheck: uma Técnica de Inspeção baseada em *Checklist* para Diagramas de Casos de Uso e de Classes da Abordagem *SMarty*

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Edson A. Oliveira Júnior

Maringá
2015

Dados Internacionais de Catalogação na Publicação (CIP)
(Biblioteca Central - UEM, Maringá, PR, Brasil)

G354s

Geraldi, Ricardo Theis
SMartyCheck: uma técnica de inspeção baseada em checklist para diagramas de casos de uso e de classes da abordagem *SMarty* / Ricardo Theis Geraldi. -- Maringá, 2015.
241 f. : il., figs., tabs. + Apêndices

Orientador: Prof. Dr. Edson Alves de Oliveira Júnior.
Dissertação (mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2015.

1. Inspeção de software. 2. Linha de Produto de Software. 3. Gerenciamento de variabilidade. 4. *SMartyCheck* - Estudos empíricos. 5. UML (Unified Modeling Language). I. Oliveira Júnior, Edson Alves de, orient. II. Universidade Estadual de Maringá. Centro de Tecnologia. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 21.ed. 004.21

MN-001981

FOLHA DE APROVAÇÃO

RICARDO THEIS GERALDI

SMartyCheck: uma técnica de inspeção baseada em *checklist* para diagramas de casos de uso e de classes da abordagem *SMarty*

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Prof. Dr. Edson Alves de Oliveira Junior
Universidade Estadual de Maringá – DIN/UEM

Profa. Dra. Itana Maria de Souza Gimenes
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Valter Vieira de Camargo
Universidade Federal de São Carlos – DC/UFSCar

Aprovada em: 20 de fevereiro de 2015.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

AGRADECIMENTOS

Agradeço primeiramente a **Deus** por ter me abençoado nesta caminhada, seja nas dificuldades, bem como nas felicidades durante todo este período. Um imenso obrigado pela compreensão de todas as pessoas que contribuíram para a realização deste trabalho. **Em especial:**

A minha família pelo carinho, amor, paciência, incentivo e dedicação com relação a mim: Ao meu amado Pai **Adenis Aparecido Geraldi**, a minha amada Mãe **Vera Lúcia Theis Geraldi** e ao meu grande parceiro e Irmão **Marcelo Theis Geraldi**. Agradecimentos nunca medirão o que estas pessoas fazem e já fizeram por mim! Meu Muito Obrigado a minha família que me tornou uma pessoa melhor frente ao aprendizado, ao mundo e em todas as lições de vida.

Ao meu orientador e amigo Dr. Edson Alves de Oliveira Júnior por sempre me guiar nas dificuldades, no auxílio em diversos trabalhos, além de me motivar e sugerir a busca por novos horizontes no desenvolvimento desta pesquisa. Ao meu primeiro orientador, Sérgio Roberto Pereira da Silva (*in memoriam*) por acreditar em mim e pela oportunidade oferecida no início do Mestrado, que descanse em paz.

Aos parceiros, pesquisadores e professores que tive a grande oportunidade de conhecê-los: Thelma Lopes, Itana Gimenes, Igor Steinmacher, Ana Chaves, Lucio Valentin, Tayana Conte, Dante Medeiros Filho, Anderson Marcolino, Munif Gebara Júnior, Yandre Maldonado, Nardênio Martins, Valéria Feltrim, Ademir Constantino e Elisa Huzita. Além disso, aos professores amigos que nunca mediram esforços para me ajudar, os quais conheço desde a minha graduação como Bacharel em Sistemas de Informação: Claudete Werner (orientadora) obrigado por sempre me incentivar, Daniela Flôr, Késsia Marchi e Fabiano Utiyama.

Aos amigos, em especial, do grupo de pesquisa GSII (Grupo de Sistemas Interativos Inteligentes): Douglas Toledo, Emanuel Duarte, Frank Cardoso, Leandro Lago e Lucas Nanni. Também, especialmente, aos amigos do meu grupo de pesquisa atual GRSSE (Grupo de Pesquisa em Reuso Sistemático de Software e Experimentação): Márcio Bera, Jaime Dias, André Felipe, Ana Allian e Maicon Pazin. Agradeço também aos amigos, que compartilharam seus conhecimentos, dificuldades e vitórias comigo durante o Mestrado: Alisson Chiquitto, Ariel Zuquello, Rômulo Beninca, Maurício Begnini, Felipe Perez, Alexandre Giron, Geazzy Zanoni, Paulo Oliveira e Pietro Martins.

A Inês, secretária do curso de Mestrado, pela cordialidade, simplicidade e disposição em ajudar sempre a todos.

Por fim, agradeço a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concedido a esta pesquisa.

SMartyCheck: uma Técnica de Inspeção baseada em *Checklist* para Diagramas de Casos de Uso e de Classes da Abordagem *SMarty*

RESUMO

Linha de Produto de Software (LPS) é uma abordagem que está sendo consolidada para a reutilização de artefatos de software com base em um domínio específico. A abordagem de LPS permite a customização de produtos com o auxílio efetivo do gerenciamento de variabilidades, satisfazendo as necessidades específicas dos clientes. No entanto, para que a qualidade de uma LPS seja garantida é necessário realizar atividades de verificação e validação, de análise estática e dinâmica, as quais estão contidas nos processos de garantia e controle da qualidade de software. Neste cenário, a revisão de software é aplicada no processo de controle de qualidade, no intuito de garantir a qualidade no processo de engenharia de software, eliminando defeitos em cada fase do processo. Assim, dentre as técnicas de revisão de software existentes está a técnica de leitura baseada em *checklist*. Tal técnica permite a detecção e remoção de diferentes tipos de defeitos, para melhorar, consequentemente, a qualidade de um produto de sofware. Neste contexto, esta dissertação apresenta uma proposta de técnica de inspeção de software baseada em *checklist*, denominada *SMartyCheck*. O principal objetivo da *SMartyCheck* é possibilitar a inspeção de diagramas da *Unified Modeling Language* (UML) de casos de uso e classes de uma LPS com base na abordagem *Stereotype-based Management of Variability* (*SMarty*). Para avaliar empiricamente a *SMartyCheck*, foi utilizada a estratégia exploratória sequencial baseada em *mixed-methods*, buscando analisar por meio de um estudo empírico qualitativo, a viabilidade da *SMartyCheck*. Após isso, um estudo quantitativo foi conduzido nesse cenário, no intuito de avaliar a eficiência, a eficácia e a efetividade da *SMartyCheck* em comparação com a técnica *Ad hoc*. Os resultados obtidos com a condução destes estudos empíricos permitiram aprimorar a *SMartyCheck*, fornecendo evidências de que a técnica é viável para inspeção de diagramas *SMarty* de LPS, além de ser eficiente, eficaz e efetiva em comparação com a técnica *Ad hoc*.

Palavras-chave: Estudos Empíricos. Gerenciamento de Variabilidades. Inspeção de Software. Linha de Produto de Software. *Mixed-Methods*. *SMarty*. *SMartyCheck*. UML.

SMartyCheck: a Checklist-based Inspection Technique for Use Case and Class Diagrams of *SMarty* Approach

ABSTRACT

Software Product Line (SPL) is a promising approach for specific domain software artifacts reuse, which has been consolidated. The SPL approach enables customization of products with an effective support of variability management, satisfying specific needs of customers. However, in order to assure SPL quality it is essential performing activities for verification and validation, static and dynamic analysis, which are encompassed by quality assurance and quality control software processes. In this scenario, software revision is applied to the quality control process in order to ensure the quality of a software engineering process, removing defects at each stage. Thus, amongst software revision techniques, exist the checklist-based reading technique. Such a technique allows the detection and removal of several types of defects, whereas improves the quality of the software products. In this context, this master thesis proposes a software inspection technique based on checklist, called SMartyCheck. The main objective of SMartyCheck technique is to enable the inspection of Unified Modeling Language (UML) use case and class SPL diagrams based on the Stereotype-based Management of Variability (SMarty) apporach. In order to empirically evaluate SMartyCheck it was used the sequential exploratory strategy based on mixed-methods, aiming at analyzing the SMartyCheck feasibility throughout a qualitative study. Furthermore, a quantitative study was conducted in this scenario to evaluate the efficiency, efficacy and effectiveness of SMartyCheck compared with an Ad hoc technique. The results obtained with the execution of such empirical studies allowed improving SMartyCheck, providing evidence that the technique is feasible for inspecting SMarty SPL diagrams, besides being efficient and effective compared to the Ad hoc technique.

Keywords: Empirical Studies. Mixed-Methods. Software Inspection. Software Product Line. SMarty. SMartyCheck. UML. Variability Management.

LISTA DE FIGURAS

Figura 1.1 Etapas da Metodologia de Desenvolvimento de Pesquisa. 20

Figura 2.1 *Framework* de Engenharia de LPS. Traduzido de (Pohl *et al.*, 2005). 24
Figura 2.2 O Processo de Gerenciamento de Variabilidades *SMartyProcess* e sua Interação entre as Atividades com o Processo de Desenvolvimento de LPS, traduzido de (OliveiraJr *et al.*, 2005; OliveiraJr, 2010). 30

Figura 3.1 Os Subprocessos da Engenharia de Domínio de LPS e a Técnica *SMartyCheck*. Adaptado e traduzido de (Pohl *et al.*, 2005). 47
Figura 3.2 Exemplo Tipo de Defeito *SMartyCheck* - Ambiguidade. 48
Figura 3.3 Exemplo Tipo de Defeito *SMartyCheck* - Incompleto. 49
Figura 3.4 Exemplo Tipo de Defeito *SMartyCheck* - Inconsistência. 49
Figura 3.5 Exemplo Tipo de Defeito *SMartyCheck* - Incorreto. 49
Figura 3.6 Exemplo Tipo de Defeito *SMartyCheck* - Instável. 50
Figura 3.7 Exemplo Tipo de Defeito *SMartyCheck* - Imodificável. 50
Figura 3.8 Exemplo Tipo de Defeito *SMartyCheck* - Fato Incorreto. 51
Figura 3.9 Exemplo Tipo de Defeito *SMartyCheck* - Informação Estranha. . 51
Figura 3.10 Exemplo Tipo de Defeito *SMartyCheck* - Desvio Intencional. . . . 51
Figura 3.11 Exemplo Tipo de Defeito *SMartyCheck* - Inviável. 52
Figura 3.12 Exemplo Tipo de Defeito *SMartyCheck* - Omissão. 52
Figura 3.13 Exemplo Tipo de Defeito *SMartyCheck* - Anomalia. 53
Figura 3.14 Taxonomia de Tipos de Defeitos da Técnica *SMartyCheck*. 53
Figura 3.15 Taxonomia com Tipos de Defeitos Equivalentes da Técnica *SMartyCheck*. 56
Figura 3.16 Diagrama de Casos de Uso *SMarty* Oráculo da LPS AGM (Apêndice C). 62
Figura 3.17 *Toy Example* - Diagrama de Casos de Uso *SMarty* da LPS AGM com Defeitos Incorporados. 63
Figura 3.18 Diagrama de Classes *SMarty* Oráculo da LPS AGM (Apêndice C). 71
Figura 3.19 *Toy Example* - Diagrama de Classes *SMarty* da LPS AGM com Defeitos Incorporados. 72

Figura 4.1 Metodologia de Pesquisa do Estudo Empírico Qualitativo. 78
Figura 4.2 Representação Gráfica com as Associações relacionadas à Categoria “Viável para a Inspeção”. 93

Figura 4.3 Representação Gráfica com as Associações relacionadas à Categoria "Possíveis Melhorias". 95

Figura 5.1 Etapas do Estudo Empírico Quantitativo. 106
Figura 5.2 Conjunto de Variáveis do Estudo Empírico Quantitativo. 113
Figura 5.3 *Box Plots* Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Casos de Uso. 118
Figura 5.4 Escala de Correlação de Spearman (Spearman, 1987). 125
Figura 5.5 *Box Plots* Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Classes. 129
Figura 5.6 Taxonomia de Tipos de Defeitos da *SMartyCheck* após a Análise do Estudo Empírico Quantitativo. 140

Figura 1.1 O Processo do Estudo Secundário seguido (adaptado de Petersen *et al.* (2008a)). 170
Figura 1.2 O Processo de Pesquisa (adaptado de Barney *et al.* (2012)). . . . 172
Figura 1.3 Estágios de Busca dos Estudos Primários (adaptado dos estudos de Mota Silveira Neto *et al.* (2011) e Mohabbati *et al.* (2013)). . . 174
Figura 1.4 Esquema de Classificação (adaptado de Petersen *et al.* (2008a)). . 176
Figura 1.5 Total de Estudos Primários por Fonte/Base de Dados Eletrônica. 178
Figura 1.6 *Word Cloud* da Distribuição dos Estudos por Evento. 180
Figura 1.7 *Word Cloud* da Distribuição dos Estudos por Periódico. 181
Figura 1.8 Número de Estudos por Tipo de Local. 181
Figura 1.9 Distribuição Cronológica dos Estudos Primários por Ano de Publicação. 182
Figura 1.10 Autores Top 30 (filtro #1). 183
Figura 1.11 Autores Top 10 (filtro #2). 183
Figura 1.12 Questões de Pesquisa (QP1 e QP2) Respondidas com base nos Estudos Selecionados por Fontes/Bases de Dados Eletrônicas e Tipos de Pesquisa. 184
Figura 1.13 Questões de Pesquisa (QP1 e QP2) Respondidas com base nos Estudos Selecionados por Tipos de Defeitos e Tipos de Pesquisa. . 186
Figura 1.14 *Word Cloud* dos Tipos de Defeitos mais Relevantes. 186
Figura 1.15 Tipos de Técnicas de Inspeção de Software baseadas nos Estudos mais Relevantes por Questões de Pesquisa (QPs) e Tipos de Pesquisa. 187

Figura 2.1 Elementos para a Extensão da UML por meio de Perfis, adaptado de (Marcolino, 2014; OMG, 2011). 197
Figura 2.2 Exemplo Diagrama de Casos de Uso, adaptado de (OMG, 2011). . 199
Figura 2.3 Exemplo Diagrama de Classes, adaptado de (OMG, 2011). 202

Figura 3.1 Diagrama de Casos de Uso da LPS AGM. Adaptado de (OliveiraJr, 2010). 205
Figura 3.2 Diagrama de Classes da LPS AGM. Adaptado de (OliveiraJr, 2010).210
Figura 3.3 Tela do Jogo Brickles 211
Figura 3.4 Tela do Jogo Pong 211
Figura 3.5 Tela do Jogo Bowling 211

Figura 4.1 Diagrama de Casos de Uso da LPS *Mobile Media*. Adaptado de (Santos *et al.*, 2008). 213
Figura 4.2 Diagrama de Casos de Uso da LPS *Mobile Media* com base na *SMarty*. Adaptado de (Contieri Junior, 2010). 215

Figura 5.1 1º Parte - Exemplo de Questionário Eletrônico de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.221
Figura 5.2 2º Parte - Exemplo de Questionário Eletrônico de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.222
Figura 5.3 3º Parte - Exemplo de Questionário Eletrônico de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.223
Figura 5.4 1º Parte - Exemplo de Questionário Eletrônico de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. . . 224
Figura 5.5 2º Parte - Exemplo de Questionário Eletrônico de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. . . 225
Figura 5.6 3º Parte - Exemplo de Questionário Eletrônico de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. . . 226
Figura 5.7 Exemplos de Condições (Sim ou Não) de Inspeção definidas para os itens do *Checklist* da *SMartyCheck*. 227
Figura 5.8 Exemplo do Questionário Eletrônico criado no LimeSurvey sobre a técnica *SMartyCheck* e o Estudo Empírico Qualitativo. 228
Figura 5.9 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 230

Figura 5.10 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 231
Figura 5.11 3º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 232
Figura 5.12 4º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 233
Figura 5.13 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 234
Figura 5.14 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 235
Figura 5.15 3º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 236
Figura 5.16 4º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*. 237
Figura 5.17 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*. 238
Figura 5.18 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*. 239
Figura 5.19 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*. 240
Figura 5.20 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*. 241

LISTA DE TABELAS

Tabela 2.1 Estudos de Taxonomias e Classificações de Tipos de Defeitos . . . 37

Tabela 3.1 Tipos de Defeitos Adotados para a Técnica *SMartyCheck*. 53
Tabela 3.2 Exemplo Inspeção Casos de Uso da LPS AGM *SMarty*. 64
Tabela 3.3 Exemplo Inspeção Classes da LPS AGM *SMarty*. 73

Tabela 4.1 Dados de Caracterização Detalhados dos Especialistas do Estudo. 82

Tabela 5.1 Dados de Caracterização Detalhados dos Participantes do Estudo. 115
Tabela 5.2 Resultados Obtidos e Estatística Descritiva da Técnica *SMarty-Check* para Diagrama de Casos de Uso. 117
Tabela 5.3 Resultados Obtidos e Estatística Descritiva da Técnica *Ad hoc* para Diagrama de Casos de Uso. 117
Tabela 5.4 Teste de Normalidade Shapiro-Wilk das Amostras da Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Casos de Uso. 119
Tabela 5.5 Ranque Mann-Whitney Wilcoxon - Diagrama de Casos de Uso. . 123
Tabela 5.6 Experiência e Nível de Conhecimento dos Participantes da Técnica *SMartyCheck*. 124
Tabela 5.7 Experiência e Nível de Conhecimento dos Participantes da Técnica *Ad hoc*. 124
Tabela 5.8 Correlação de Spearman entre a Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* e o Nível de Conhecimento de UML, LPS e Inspeção de Software - Diagrama de Casos de Uso. 125
Tabela 5.9 Resultados Obtidos e Estatística Descritiva da Técnica *SMarty-Check* para Diagrama de Classes. 127
Tabela 5.10 Resultados Obtidos e Estatística Descritiva da Técnica *Ad hoc* para Diagrama de Classes. 128
Tabela 5.11 Teste de Normalidade Shapiro-Wilk das Amostras da Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Classes. 130
Tabela 5.12 Ranque Mann-Whitney Wilcoxon - Diagrama de Classes. 134
Tabela 5.13 Correlação de Spearman entre a Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* e o Nível de Conhecimento de UML, LPS e Inspeção de Software - Diagrama de Classes. 135

Tabela 5.14 Resumo dos Resultados do Estudo Empírico Quantitativo. 136

Tabela 1.1 Número de Estudos por Filtro e Fontes/Bases de Dados Eletrônicas. 173
Tabela 1.2 Palavras-chave e *Strings* de Busca da *Query* de Busca. 174
Tabela 1.3 Estudos Recuperados com base nos Critérios de Inclusão e Exclusão (filtro #2). 179
Tabela 1.4 Lista de Eventos dos Estudos Selecionados 180
Tabela 1.5 Lista de Periódicos dos Estudos Selecionados 181
Tabela 1.6 Estudos mais Relevantes - Ordenados por Questões de Pesquisa (QP1 e QP2) 188

Tabela 2.1 Elementos Gráficos de Diagramas de Casos de Uso e suas Descrições, adaptado de (OMG, 2011). 198
Tabela 2.2 Elementos Gráficos de Diagramas de Classes e suas Descrições, adaptado de (OMG, 2011). 200
Tabela 2.3 Caminhos Gráficos de Diagramas de Classes e suas Descrições, adaptado de (OMG, 2011). 201

Tabela 3.1 Pacotes e Descrição das Classes da LPS AGM 209

LISTA DE SIGLAS E ABREVIATURAS

AGM: *Arcade Game Maker*
AGRT: *Actor Goal Reading Technique*
CBR: *Checklist-Based Reading*
GT: *Grounded Theory*
GV: Gerenciamento de Variabilidade
LPS: Linha de Produto de Software
PBR: *Perspective-Based Reading*
PLP: *Product Line Practice*
SEI: *Software Engineering Institute*
SMarty: *Stereotype-based Management of Variability*
SPL: *Software Product Line*
SPLIT: *Software Product Line Inspection Techniques*
TUCCA: *Technique for Use Case Model Construction and Construction-based Requirements Document Analysis*
UCRT: *Use Case Reading Technique*
UML: *Unified Modeling Language*
VM: *Variability Management*

SUMÁRIO

1 Introdução **16**
1.1 Contextualização . . . 16
1.2 Motivação . . . 18
1.3 Objetivos . . . 19
1.4 Metodologia de Desenvolvimento da Pesquisa . . . 19
1.5 Organização do Trabalho . . . 21

2 Fundamentação Teórica **22**
2.1 Considerações Iniciais . . . 22
2.2 Linha de Produto de Software (LPS) e Gerenciamento de Variabilidades . . . 23
2.2.1 A Abordagem ***SMarty*** para Gerenciamento de Variabilidades . . . 26
2.3 Inspeção de Software . . . 33
2.3.1 Taxonomias e Classificações de Tipos de Defeitos . . . 35
2.3.2 Técnicas de Detecção de Defeitos . . . 39
2.3.3 Técnicas de Inspeção de Software para LPS . . . 42
2.4 Considerações Finais . . . 44

3 *SMartyCheck 2.0*: uma Técnica de Inspeção para Diagramas UML de LPS **45**
3.1 Considerações Iniciais . . . 45
3.2 Caracterização da Técnica ***SMartyCheck 2.0*** . . . 46
3.3 Taxonomia de Tipos de Defeitos da ***SMartyCheck 2.0*** . . . 48
3.4 ***SMartyCheck 2.0***: Diagramas UML de LPS Inspecionados . . . 57
3.4.1 Inspeção de Diagramas ***SMarty*** de Casos de Uso . . . 58
3.4.2 Inspeção de Diagramas ***SMarty*** de Classes . . . 67
3.5 Considerações Finais . . . 76

4 Estudo Empírico Qualitativo da *SMartyCheck 2.0* **77**
4.1 Considerações Iniciais . . . 77
4.2 Definição do Estudo . . . 79
4.3 Planejamento do Estudo . . . 79
4.4 Execução do Estudo . . . 81
4.5 Análise e Interpretação dos Resultados . . . 88
4.6 Avaliação de Validade do Estudo . . . 96
4.6.1 Ameaças à Validade de Conclusão . . . 96

4.6.2 Ameaças à Validade de ***Constructo*** 96
4.6.3 Ameaças à Validade Interna 96
4.6.4 Ameaças à Validade Externa 97
4.7 Refinamento da ***SMartyCheck 2.0*** com os Resultados Obtidos 98
4.8 Considerações Finais 104

5 Estudo Empírico Quantitativo da *SMartyCheck 2.1* **105**
5.1 Considerações Iniciais 105
5.2 Definição do Estudo 107
5.3 Planejamento do Estudo 108
5.4 Execução do Estudo 113
5.5 Análise e Interpretação dos Resultados 115
5.5.1 Análise e Interpretação dos Resultados para Diagramas de Casos de Uso 116
5.5.2 Análise e Interpretação dos Resultados para Diagramas de Classes . 126
5.5.3 Análise e Interpretação dos Resultados Gerais Obtidos 136
5.6 Avaliação de Validade do Estudo 138
5.6.1 Ameaças à Validade de Conclusão 138
5.6.2 Ameaças à Validade de ***Constructo*** 138
5.6.3 Ameaças à Validade Interna 138
5.6.4 Ameaças à Validade Externa 139
5.7 Aprimoramento da ***SMartyCheck 2.1*** com os Resultados Obtidos 140
5.8 Considerações Finais 145

6 Conclusão **147**
6.1 Contribuições 148
6.2 Limitações 150
6.3 Trabalhos Futuros 151

REFERÊNCIAS **153**

A Apêndice A - Estudo Secundário sobre Tipos de Defeitos em Técnicas de Inspeção de Software **169**
A.1 O Processo do Estudo Secundário 169
A.1.1 Definição das Questões de Pesquisa 170
A.1.2 O Processo de Pesquisa 170

A.1.3 Definição das Fontes/Bases Eletrônicas, Palavras-Chave e ***Strings*** de Busca . . . 172
A.1.4 Definição dos Critérios de Inclusão e Exclusão . . . 174
A.1.5 Esquema de Classificação . . . 175
A.1.6 Extração e Agregação dos Dados . . . 177
A.2 Discussão e Resultados do Estudo Secundário . . . 177
A.2.1 Visão Geral do Estudo Secundário . . . 177
A.2.2 Metadados dos Estudos Primários . . . 178
A.2.3 Discussão dos Principais Estudos Selecionados . . . 184
A.2.4 Discussão dos Estudos Mais Relevantes . . . 188
A.3 Ameaças à Validade . . . 193
A.4 Considerações Finais . . . 194

B Apêndice B - Perfis UML: Diagramas de Casos de Uso e Classes 196
B.1 Perfis UML: Diagramas de Casos de Uso e Classes . . . 196
B.1.1 Fundamentos sobre Perfis UML . . . 196
B.1.2 Diagramas de Casos de Uso . . . 197
B.1.3 Diagramas de Classes . . . 199

C Apêndice C - A Linha de Produto de Software *Arcade Game Maker* 203
C.1 Introdução . . . 203
C.2 Similaridades e Variabilidades . . . 203
C.3 Atores e Casos de Uso . . . 204
C.4 Classes . . . 208
C.5 Telas dos Jogos . . . 211

D Apêndice D - A Linha de Produto de Software *Mobile Media* 212

E Apêndice E - Questionários Eletrônicos *SMartyCheck* 216
E.1 Introdução . . . 216
E.2 Questionário de Caracterização dos Participantes . . . 216
E.3 Questionários Eletrônicos - Estudo Empírico Qualitativo de Viabilidade . . . 219
E.4 Questionários Eletrônicos - Estudo Empírico Quantitativo de Efetividade . . . 229

1

Introdução

"O passado é história, o futuro é mistério, e hoje é uma dádiva. Por isso é chamado de presente!"

Provérbio Chinês

1.1 Contextualização

A reutilização de software é uma técnica empregada para melhorar a produtividade e a qualidade no ciclo de desenvolvimento de software. Neste cenário, a fim de apoiar a reutilização de software de forma sistemática e não oportunística, surge o conceito de Linha de Produto de Software (LPS), o qual está se difundindo cada vez mais com o passar do tempo na indústria e na academia, segundo Pohl *et al.* (2005), Linden *et al.* (2007) e Ahnassay *et al.* (2013).

Pode-se definir LPS como um conjunto de sistemas de software que compartilham características comuns, as quais podem ser gerenciadas, satisfazendo as necessidades de um segmento em particular de mercado ou missão (Clements e Northrop, 2002; Linden *et al.*, 2007; Northrop, 2002).

Dentre os principais benefícios obtidos com a adoção de LPS estão: a melhoria da qualidade, a redução dos custos de desenvolvimento, bem como a redução dos custos do tempo de produção, a diminuição de riscos, além dos benefícios que afetam diretamente a qualidade do produto de software entregue ao cliente de forma adaptada às suas

necessidades (Pohl *et al.*, 2005). Assim, as organizações perceberam que a adoção de LPS é de fato uma forma sistemática de reutilizar artefatos de software para construir produtos customizados, de forma rápida e com qualidade (Ahnassay *et al.*, 2013; Gomaa, 2004; Linden *et al.*, 2007; SEI, 2012a).

Além dos benefícios proporcionados para as organizações com a adoção de LPS, diversas atividades são essenciais, como o gerenciamento de variabilidades. Por definição, uma variabilidade é a forma como os membros de uma família de produtos podem se diferenciar entre si, sendo descrita por pontos de variação, variantes e restrições entre variantes (Halmans e Pohl, 2003; Linden *et al.*, 2007; Pohl *et al.*, 2005). Neste sentido, o gerenciamento de variabilidades permite a identificação, a representação e o rastreamento de variabilidades de uma LPS (Pohl *et al.*, 2005). Segundo estudos secundários realizados por Chen *et al.* (2009), Galster *et al.* (2013) e Thurimella e Bruegge (2012), várias abordagens de gerenciamento de variabilidades são encontradas na literatura.

A abordagem *Stereotype-based Management of Variability* (*SMarty*) permite o gerenciamento de variabilidades em LPS, baseado em modelos *Unified Modeling Language* (UML) (OliveiraJr *et al.*, 2010, 2013). *SMarty* é composta por um perfil UML, o *SMartyProfile*, e um processo, o *SMartyProcess*, formado por atividades e diretrizes bem definidas para identificar e rastrear variabilidades em modelos UML de uma LPS utilizando estereótipos propostos em seu perfil. A efetividade da *SMarty* é evidenciada pelas avaliações experimentais realizadas nos estudos de Marcolino *et al.* (2013), Marcolino *et al.* (2014a) e Marcolino *et al.* (2014b).

SMarty vem sendo consolidada nos últimos anos, com base na sua adoção por diversos projetos e trabalhos acadêmicos como em Fragal *et al.* (2013), Rodrigues *et al.* (2012), Oizumi *et al.* (2012), Contieri Junior *et al.* (2011). A presença de um perfil UML e um processo sistemático a diferencia das demais abordagens existentes. SMarty, ao contrário de demais abordagens existentes como, Ziadi *et al.* (2004), Korherr e List (2007), Gomaa (2004), permite o gerenciamento variabilidades de LPSs por meio da identificação e representação de variabilidades em modelos UML de caso de uso, sequência, classe, componente e atividade.

Neste sentido, os artefatos de uma LPS podem ser inspecionados, como um modo de prevenir e garantir que defeitos, não venham a ocorrer durante o ciclo de vida e nos produtos específicos que podem ser gerados a partir de uma LPS. Por definição, a inspeção de software é uma técnica de avaliação envolvida, em particular, com um tipo de revisão específica, na qual pode ser aplicada aos artefatos de software por meio de um processo sistemático e bem planejado de detecção de defeitos (Fagan, 1986; Kalinowski e Travassos, 2004; Mello *et al.*, 2012). Assim, é necessária a elaboração de uma boa documentação,

além de avaliar a conformidade da especificação de diferentes artefatos e requisitos, a fim de suprir as necessidades e expectativas dos usuários (Ackerman *et al.*, 1989; Cunha *et al.*, 2012; Travassos *et al.*, 1999). Deste modo, a inspeção de software vem sendo timidamente utilizada por engenheiros de software no intuito de garantir e controlar a qualidade de LPSs e produtos específicos.

1.2 Motivação

Um software pode incorporar distintos tipos de defeitos, que devem ser detectados e removidos em um período curto de tempo, pois do contrário, isto pode gerar altos custos no desenvolvimento e manutenção do produto (Boehm e Basili, 2001). Portanto, técnicas de inspeção de software podem ser utilizadas para detectar defeitos e minimizar os custos no desenvolvimento de software, especialmente em estágios iniciais.

Dentre os benefícios diretos e indiretos proporcionados pela aplicação planejada da inspeção de software, estão: a redução de custos que impactam o gerenciamento, o ciclo de vida e o nível de defeitos de um software, além do aumento da produtividade em equipe e redução da escala de tempo de desenvolvimento, bem como o tempo de testes (Alshazly *et al.*, 2014; Brykczynski *et al.*, 1994; Fagan, 2002; Gilb e Graham, 1993; Hayes *et al.*, 2006; Jones, 1996; Rombach *et al.*, 2008).

Há muitos relatos que comprovam a eficácia e o sucesso das inspeções por meio de resultados positivos obtidos com seu uso, além dos benefícios retratados por empresas como: a IBM (Fagan, 2002, 1986; Rombach *et al.*, 2008), a NASA (Hayes, 2003; Hayes *et al.*, 2006; Rombach *et al.*, 2008), a Motorola, a Allianz (Kristiansen, 2010; Rombach *et al.*, 2008), a AT&T Bell Labs (Godfrey, 1986), a Bell Northern Research (BNR) (Russell, 1991), a Bull HN Information Systems (Weller, 1993) e a Jet Propulsion Laboratory (Kelly *et al.*, 1992).

De acordo com os benefícios proporcionados pela inspeção de software e com um número reduzido de técnicas atuais de inspeção para LPS na literatura (Cunha *et al.*, 2012; Mello *et al.*, 2014), a inspeção de diagramas UML contendo variabilidade não é suportada. Assim, a adoção e o diferencial da abordagem *SMarty* favorecem a aplicação de inspeções em seus diagramas UML, pois pode auxiliar na melhoria da detecção e redução de possíveis defeitos em LPSs.

1.3 Objetivos

Conforme o contexto e a motivação apresentados, este trabalho tem como objetivo principal propor a técnica *SMartyCheck* para detectar defeitos em LPSs com base em diagramas UML *SMarty* contendo variabilidade por meio de um *checklist*. Assim, a técnica *SMartyCheck* pode contribuir com a melhoria da qualidade de diagramas UML *SMarty* por meio da inspeção desses diagramas na etapa de Engenharia de Domínio (Pohl *et al.*, 2005).

Para alcançar o objetivo geral deste trabalho, os seguintes objetivos específicos são listados:

- identificar na literatura os principais tipos de defeitos de técnicas de inspeção de software com suporte a diagramas UML;
- definir um *checklist* para a técnica proposta considerando tipos de defeitos para diagramas UML *SMarty* contendo variabilidade em casos de uso e classes;
- avaliar empiricamente a técnica *SMartyCheck*.

1.4 Metodologia de Desenvolvimento da Pesquisa

Para o desenvolvimento deste trabalho foram utilizadas duas abordagens: teórica e empírica. Assim, esta pesquisa está acerca da abordagem teórica, com base na revisão bibliográfica (Capítulo 2) apoiada por um estudo secundário da literatura (Apêndice A) sobre a área de pesquisa. O teor empírico desta pesquisa leva em consideração estudos qualitativos e quantitativos.

A abordagem teórica é estabelecida como base a fim de permitir a compreensão do estado da arte e propor a técnica *SMartyCheck* no contexto apropriado, permitindo o refinamento e a evolução da técnica por meio de estudos empíricos. A Figura 1.1 apresenta as atividades e as etapas da metodologia utilizada neste trabalho.

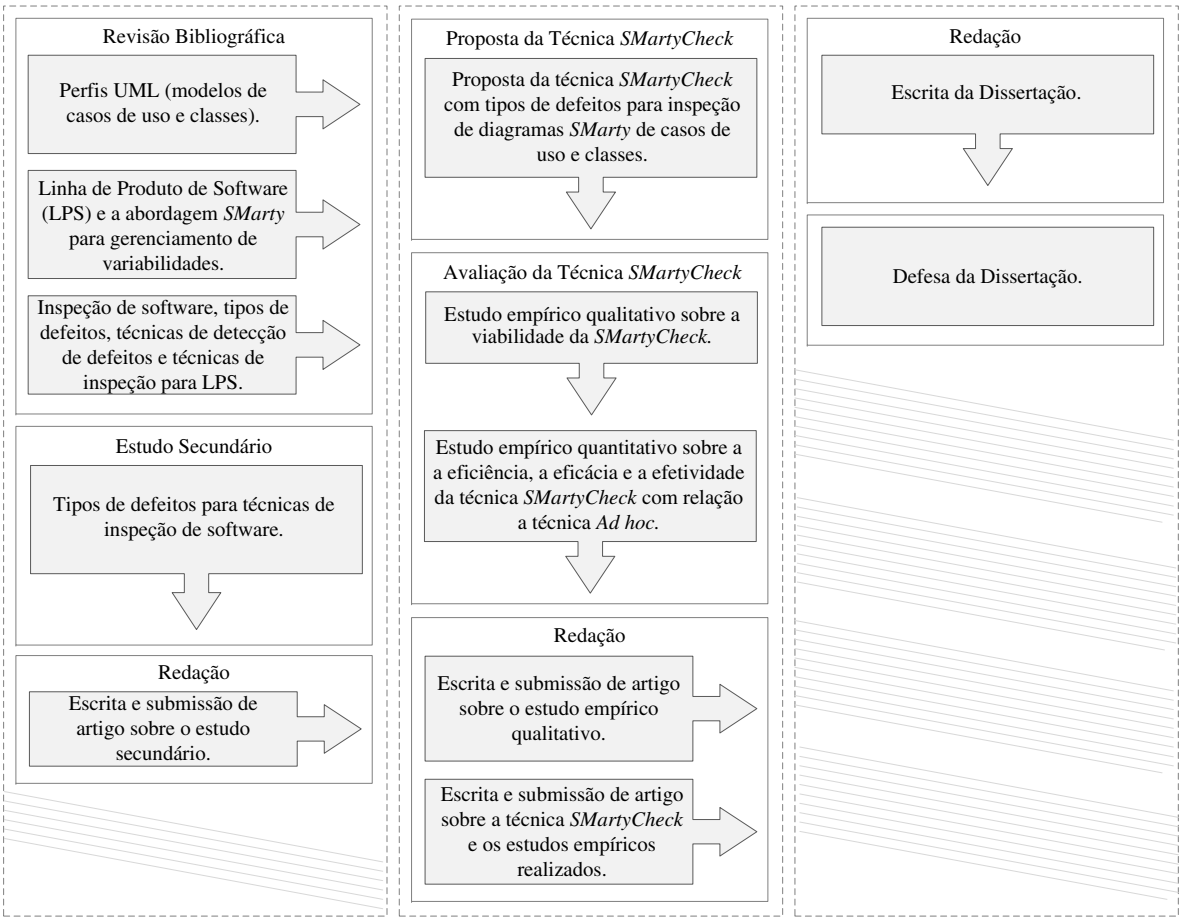


Figura 1.1: Etapas da Metodologia de Desenvolvimento de Pesquisa.

A seguir é apresentada uma breve descrição das etapas da metodologia aplicadas à esta pesquisa:

- **Revisão Bibliográfica:** engloba os conceitos essenciais e o estudo dos seguintes temas: perfis UML, diagramas de casos de uso e de classes (Apêndice B), Linha de Produto de Software (LPS) e gerenciamento de variabilidades (Seção 2.2), a abordagem *SMarty* (Seção 2.2.1), inspeção de software (Seção 2.3), taxonomias e classificações de tipos de defeitos (Seção 2.3.1), técnicas de detecção de defeitos (Seção 2.3.2) e técnicas de inspeção de software para LPS (Seção 2.3.3).

- **Estudo Secundário:** consiste em um mapeamento sistemático de estudos primários, de acordo com o Apêndice A por meio do respectivo tópico: tipos de defeitos para técnicas de inspeção de software. A análise dos estudos do mapeamento realizado permitiu a identificação de tipos de defeitos no escopo da *SMartyCheck*.

- **Proposta da Técnica *SMartyCheck*:** refere-se a proposta da técnica *SMartyCheck* com tipos de defeitos para a inspeção de diagramas *SMarty* de casos de uso e de classes.

- **Avaliação Empírica da Técnica *SMartyCheck*:** realização de avaliações por meio de dois estudos empíricos conduzidos com a finalidade de analisar a viabilidade da técnica *SMartyCheck*, bem como evidenciar a sua eficiência, eficácia e efetividade em comparação com a técnica *Ad hoc*. Os estudos empíricos foram projetados e conduzidos seguindo as abordagens de *mixed-methods*, especificamente a estratégia sequencial exploratória Creswell e Clark (2010). Para o estudo qualitativo (Dybå *et al.*, 2011; Seaman, 1999) foram realizadas análises de conteúdo por meio de procedimentos da *Grounded Theory* (GT) com base em conceitos de *Coding* (*Open Coding* e *Axial Coding*) (Corbin e Strauss, 2008). Já o estudo quantitativo foi apoiado por técnicas de análise de dados coletados, como a estatística descritiva e estatística inferencial (Juristo e Moreno, 2010; Wohlin *et al.*, 2012).

- **Redação:** trata-se da escrita e submissão de artigos sobre a técnica proposta e os estudos empíricos conduzidos, bem como a escrita e submissão de artigo sobre o estudo secundário realizado, além da escrita e defesa da dissertação.

1.5 Organização do Trabalho

Neste capítulo foram apresentados o contexto no qual esta dissertação está inserida, a motivação, os objetivos e a metodologia de desenvolvimento de pesquisa. O Capítulo 2 apresenta os conceitos básicos sobre Linha de Produto de Software (LPS) e gerenciamento de variabilidades, inspeção de software, taxonomias e classificações de tipos de defeitos, técnicas de detecção de defeitos e técnicas de inspeção de software para LPS. O Capítulo 3 descreve a proposta da técnica *SMartyCheck*, bem como a sua caracterização e taxonomia de tipos de defeitos. Os Capítulos 4 e 5 apresentam e discutem os dois estudos empíricos conduzidos com o intuito de analisar a viabilidade e evidenciar a eficiência, a eficácia e a efetividade da *SMartyCheck*. No Capítulo 6 são apresentadas as conclusões sobre esta pesquisa, as contribuições, as limitações observadas e direções para trabalhos futuros.

2

Fundamentação Teórica

"O que sabemos é uma gota; o que ignoramos é um oceano."

Isaac Newton (1642 - 1727), Cientista

2.1 Considerações Iniciais

A existência de poucas técnicas de inspeção de software para LPS na literatura motiva a proposta de novas técnicas. Logo, este capítulo revisa os principais tipos de defeitos e técnicas existentes para inspeção de software essenciais ao desenvolvimento deste trabalho.

Considerando o foco e o objetivo deste trabalho, as técnicas de detecção de defeitos baseadas em *checklist*, além da técnica *Ad hoc* são discutidas a fim de posicionar o escopo desta pesquisa permeando a proposta da técnica *SMartyCheck* (Capítulo 3).

Na sequência, técnicas de inspeção de software para LPS identificadas na literatura e por meio do estudo secundário (Apêndice A) realizado são discutidas na Seção 2.3.3.

Assim, a fundamentação teórica elementar é apresentada neste capítulo para propiciar a compreensão dos principais conceitos acerca de Perfis UML (Apêndice B), Linha de Produto de Software (LPS) e gerenciamento de variabilidades (Seção 2.2), a abordagem *SMarty* (Seção 2.2.1), inspeção de software (Seção 2.3) e taxonomias e classificações de tipos de defeitos (Seção 2.3.1).

2.2 Linha de Produto de Software (LPS) e Gerenciamento de Variabilidades

Linha de Produto de Software (LPS) é uma abordagem que representa um conjunto de sistemas de software que compartilham características comuns, as quais podem ser gerenciadas, satisfazendo as necessidades de um segmento em particular de mercado ou missão (Bosch e Lee, 2010; Clements e Northrop, 2002; Linden *et al.*, 2007; Northrop, 2002).

Em LPS, o conjunto de sistemas de software é chamado de família de produtos, sendo que os membros desta família são produtos específicos desenvolvidos a partir de um núcleo de artefatos (*Core assets*). Com isso, uma LPS se difere do processo de desenvolvimento de software tradicional, pois considera uma família de produtos, ao contrário, do método tradicional que considera cada sistema de forma individual (Linden *et al.*, 2007).

O núcleo de artefatos é constituído por um conjunto de características comuns (similaridades) e variáveis (variabilidades), formando a base de uma LPS (Linden *et al.*, 2007). As variabilidades estão relacionadas às características variáveis dos produtos específicos gerados com artefatos reutilizáveis e com a possibilidade de serem instanciados de um núcleo (Capilla *et al.*, 2013).

Neste cenário, o *framework* de engenharia de LPS criado por Pohl *et al.* (2005) tem o objetivo de incorporar os conceitos centrais da engenharia de linha de produto tradicional e facilitar a reutilização de artefatos, além de permitir a customização em massa por meio de variabilidades.

A Figura 2.1 ilustra o *framework* de engenharia de LPS, bem como seus processos e subprocessos (Pohl *et al.*, 2005).

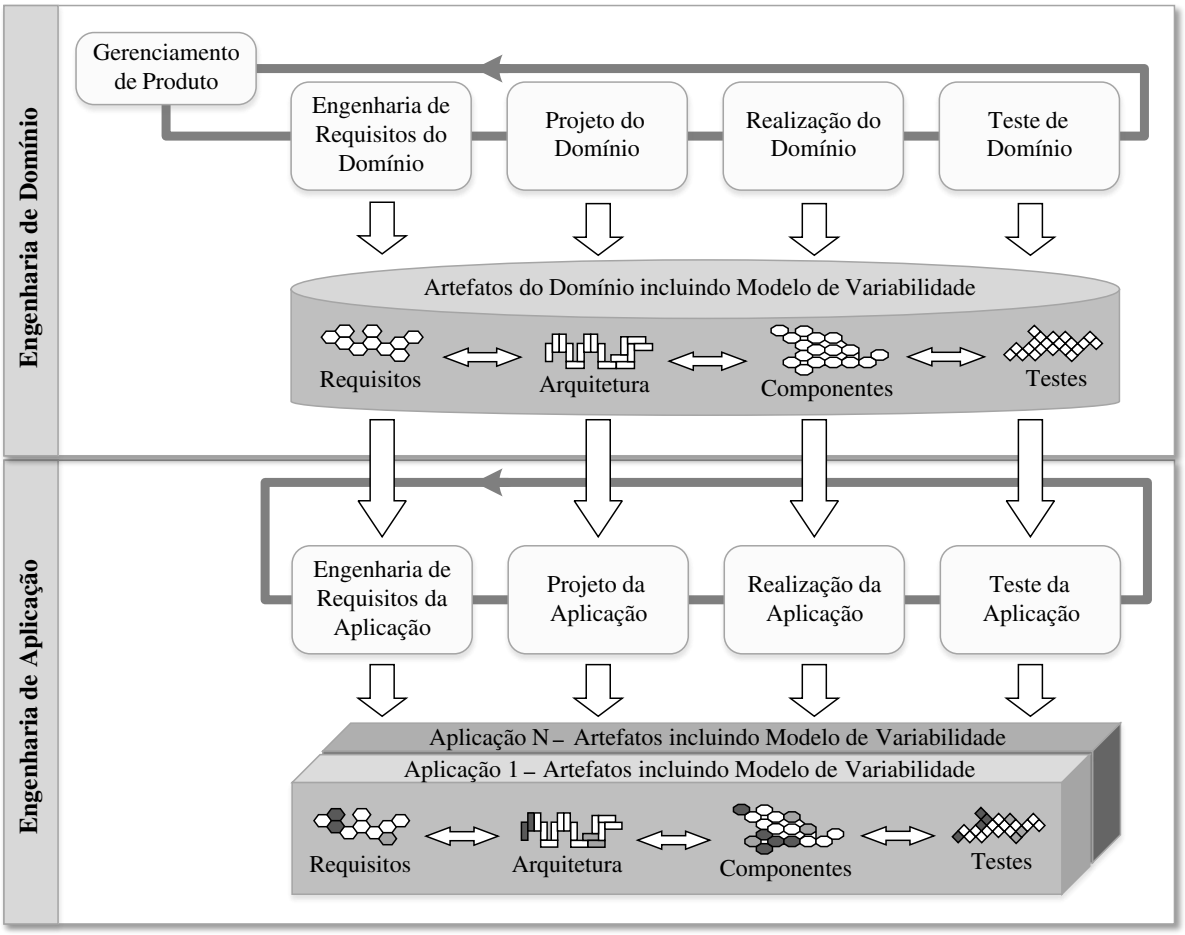


Figura 2.1: *Framework* de Engenharia de LPS. Traduzido de (Pohl *et al.*, 2005).

No contexto deste *framework*, a engenharia de LPS é separada em dois processos, os quais contém diferentes subprocessos (Pohl *et al.*, 2005):

- **Engenharia de Domínio**: subprocesso em que os aspectos comuns e as variabilidades das LPSs são identificados(as) e apresentados(as);
- **Engenharia de Aplicação**: subprocesso em que as aplicações de uma LPS são construídas por meio da reutilização de artefatos de domínio, explorando as variabilidades de uma linha.

Segundo Pohl *et al.* (2005), além de testes, técnicas para qualidade de software podem ser empregadas nos processos de Engenharia de Domínio e de Engenharia de Aplicação, tais como revisão de software, inspeção de software e *walkthroughs*. Estas técnicas especializadas requerem adaptações segundo relatórios de experiência identificados na literatura (veja Apêndice A).

No sentido de visar produtividade e qualidade, LPS vem sendo consolidada como uma abordagem adotada por grandes organizações, as quais estão presentes no *hall* da fama do SEI (2012b). Dentre as empresas globalmente conhecidas que utilizam LPS, estão: Boeing, Bosch Group, Hewlett-Packard (HP), Nokia, Philips, Siemens, Toshiba e a U.S. Naval Research Laboratory. Perante a isto, o sucesso na adoção de LPS se deve principalmente a atividades de gerenciamento técnico e organizacional bem planejados. Um exemplo é o gerenciamento de variabilidades (Capilla *et al.*, 2013; Chen *et al.*, 2009; Galster *et al.*, 2013; Thurimella e Bruegge, 2012).

O gerenciamento de variabilidades possui quatro atividades fundamentais: a identificação de variabilidade, a delimitação de variabilidade, a implementação de variabilidade e o gerenciamento das variantes (Capilla *et al.*, 2013; Chen *et al.*, 2009; Galster *et al.*, 2013).

A variabilidade é de suma importância para uma LPS, pois é a forma como os membros de uma família de diversos produtos podem se distinguir entre si (Weiss e Lai, 1999). De acordo com Linden *et al.* (2007), "uma variabilidade em LPS é uma característica que pode ser comum a todos os produtos, mas não para todos". Assim, uma variabilidade pode ser modelada para permitir o desenvolvimento de produtos personalizados por meio da configuração e ajuste de artefatos reutilizáveis para um determinado contexto peculiar (Pohl *et al.*, 2005; Thurimella e Bruegge, 2012).

Segundo Halmans e Pohl (2003), quando certas decisões de projeto de produtos de software são postergadas, é provável o surgimento de um número maior de variabilidades. Neste contexto, as variabilidades podem ser resolvidas com base em um tempo de resolução (tempo de compilação, de ligação, de execução e de atualiação) (Linden *et al.*, 2007). Desta forma, a resolução das variabilidades se faz por meio da associação de uma ou mais variantes relacionadas a um ponto de variação em específico (Gurp *et al.*, 2001).

Uma variabilidade é descrita por meio de pontos de variação, variantes e restrições entre variantes (Halmans e Pohl, 2003; Linden *et al.*, 2007; Pohl *et al.*, 2005):

- **Pontos de Variação**: segundo Jacobson *et al.* (1997), "um ponto de variação identifica uma ou mais localidades em que a variação irá ocorrer". Assim, um ponto de variação pode ocorrer em diferentes níveis de abstração de artefatos genéricos de uma LPS. Deste modo, é permitida a resolução de suas variabilidades em um ou mais locais por meio das suas variantes associadas;

- **Variantes**: representam possíveis elementos ou unidades, os quais podem ser escolhidos para resolver um ponto de variação ou uma variabilidade específica; e

- **Restrições entre variantes**: estabelecem os relacionamentos entre uma ou mais variantes com o objetivo de resolver seus respectivos pontos de variação ou variabilidade em um dado tempo de resolução (Linden *et al.*, 2007).

O gerenciamento de variabilidades em LPS se tornou de fato relevante na comunidade científica, conforme estudos secundários de literatura realizadas por Chen *et al.* (2009) e Galster *et al.* (2013). Além disso, diversas abordagens utilizam com sucesso o gerenciamento de variabilidades, dentre as quais estão o método PLUS de Gomaa (2004) e a abordagem de Ziadi *et al.* (2004), além da abordagem *SMarty* de OliveiraJr *et al.* (2010) apresentada em detalhes no Seção 2.2.1 por ser a abordagem mais relevante à esta dissertação.

2.2.1 A Abordagem *SMarty* para Gerenciamento de Variabilidades

De acordo com os conceitos essenciais apresentados sobre LPS (Seção 2.2) e gerenciamento de variabilidades, a abordagem *Stereotype-based Management of Variability* (*SMarty*) foi proposta no estudo de OliveiraJr *et al.* (2010), na qual possui suporte, em sua versão 4.0, a modelos UML de casos de uso, classes, atividades e componentes.

Tal abordagem (*SMarty*) tem o principal objetivo de permitir que as variabilidades de uma LPS possam ser gerenciadas de maneira efetiva em modelos UML. Para isto, a *SMarty* foi avaliada por meio de um conjunto de experimentos conduzidos e apresentados nos estudos realizados por Marcolino *et al.* (2013), Marcolino *et al.* (2014a), (Marcolino *et al.*, 2014b), OliveiraJr *et al.* (2013) e OliveiraJr *et al.* (2010).

As avaliações experimentais conduzidas por Marcolino *et al.* (2014a, 2013) permitiram que a *SMarty* fosse extendida de forma efetiva para suportar modelos UML de sequência em seu escopo, como proposto no estudo de Marcolino (2014). Deste modo, a *SMarty* evoluiu para a versão 5.1.

Para cada modelo UML suportado pela *SMarty* uma versão (1.0) é acrescentada, ou seja, a *SMarty* suporta cinco modelos UML: casos de uso (1.0), classes (2.0), atividades (3.0), componentes (4.0) e sequência (5.0), os quais equivalem a versão 5.1 da *SMarty*.

A abordagem *SMarty* é composta de um Perfil UML, o *SMartyProfile*, e um Processo, o *SMartyProcess* (Fiori *et al.*, 2012; OliveiraJr *et al.*, 2010, 2013):

- ***SMartyProfile***: permite a representação gráfica de variabilidades por meio da UML e seu mecanismo de perfil, o qual é formado por um conjunto de estereótipos e meta-atributos (*tagged values*) utilizados para representar variabilidades em modelos UML de LPS; e

- ***SMartyProcess***: é um processo sistemático, no qual possui um conjunto de diretrizes associadas com o objetivo de guiar o usuário na identificação, representação, e rastreamento de variabilidades de uma LPS (OliveiraJr *et al.*, 2010).

O Perfil *SMartyProfile*

O *SMartyProfile* baseia-se no inter-relacionamento dos principais conceitos de LPS, no qual engloba o gerenciamento de variabilidades (Seção 2.2). Com base na relação entre estes conceitos de gerenciamento de variabilidades e os modelos UML, é permitido que estes conceitos sejam utilizados aos elementos de interesse do metamodelo da UML. Deste modo, o perfil UML *SMartyProfile* 5.1 possui um conjunto de estereótipos e meta-atributos (*tagged values*), os quais são listados e descritos de forma sucinta a seguir:

- <<**variability**>> representa o conceito de variabilidade, no qual é uma extensão da metaclasse UML *Comment*. Este estereótipo é aplicado apenas em notas UML e é constituído dos seguintes meta-atributos:

  - `name`, nome de referência para uma variabilidade;
  - `minSelection`, denota a quantia mínima de variantes a serem selecionadas para resolver um ponto de variação ou variabilidade;
  - `maxSelection`, denota a quantia máxima de variantes a serem selecionadas para resolver um ponto de variação ou variabilidade;
  - `bindingTime`, delimita o instante no qual uma variabilidade será resolvida. Este tempo no qual acontecerá está representado pela classe de enumeração *BindingTime*;
  - `allowsAddingVar`, define se existe a possibilidade de inserção de novas variantes após uma variabilidade ser resolvida;
  - `variants`, apresenta a coleção de instâncias associada à variabilidade; e
  - `realizes`, representa a coleção de variabilidades de modelos de nível inferior, no qual é possível realizar a variabilidade.

- <<**variationPoint**>> representa o conceito de ponto de variação, no qual é uma extensão das metaclasses UML: *DecisionNode*, *Actor*, *UseCase*, *Interface*, e *Class*. Sendo assim, tal estereótipo é empregado somente a nós de decisão, atores, casos de uso, interfaces e classes. Este estereótipo é constituído dos seguintes meta-atributos:

  - `numberOfVariants`, indica o número de variantes associadas que podem ser selecionadas para resolver esse ponto de variação;

  - `bindingTime`, define o instante no qual um ponto de variação será resolvido. Este tempo no qual ocorrerá está representado pela classe de enumeração *BindingTime*;
  - `allowsAddingVar`, define se existe a possibilidade de inserção de novas variantes após uma variabilidade ser resolvida;
  - `variants`, representa a coleção de instâncias das variantes, as quais estão associadas a esse ponto de variação; e
  - `variabilities`, representa a coleção de variabilidades, nas quais esse ponto de variação está associado.

- <<**variant**>> engloba o conceito de variante, no qual é uma extensão abstrata das metaclasses UML: *Actor*, *UseCase*, *Interface*, *Class*, *Action*, *ActivityPartition*, *Component*, *Dependency* e *Comment*. Tal estereótipo é abstrato, o qual não pode ser aplicado em nenhum elemento UML. No entanto, suas especializações não abstratas podem ser aplicadas em atores, casos de uso, interfaces, classes, ações, partição de atividade, componente, dependência, comentário, fragmento combinado, mensagem e pacotes. Portanto, suas especializações, não abstratas abrangem os seguintes estereótipos: <<mandatory>>, <<optional>>, <<alternative_OR>> e <<alternative_XOR>>. O estereótipo <<variant>> é constituído dos seguintes meta-atributos:
  - `rootVP`, representa o ponto de variação ao qual está associado; e
  - `variabilities`, coleção de variabilidades com as quais esta variante está associada.

- <<**mandatory**>> representa uma variante obrigatória que está presente em todos os produtos de uma LPS;

- <<**optional**>> representa uma variante que pode ser escolhida para resolver uma variabilidade ou um ponto de variação;

- <<**alternative_OR**>> representa uma variante que faz parte de um grupo de variantes inclusivas. Assim, diferentes combinações destas variantes podem resolver pontos de variação de diferentes modos, gerando desta forma, produtos distintos;

- <<**alternative_XOR**>> denota uma variante que pertence há um grupo de variantes exclusivas, na qual somente uma variante do grupo pode ser selecionada para resolver um ponto de variação;

- <<**mutex**>> significa o conceito de restrição "Exclusão Mútua", sendo um relacionameto mutuamente exclusivo entre variantes. Isto significa que para uma variante ser selecionada, a variante relacionada não pode ser selecionada;

- <<**requires**>> manifesta o conceito de restrição "Complemento", sendo um relacionamento entre variantes, no qual a variante escolhida requer a seleção da variante relacionada; e

- <<**variable**>> é uma extensão da metaclasse UML *Component*. Tal estereótipo aponta que um componente é formado por um conjunto de classes com variabilidades explícitas. Este estereótipo possui o meta-atributo `classSet`, no qual é a coleção de instâncias das classes variáveis que formam o componente.

O perfil *SMartyProfile* possui características próprias para representar relações de cada modelo UML suportado entre pontos de variação e suas variantes, nas quais, estas relações entre as variantes, são representadas por meio da utilização de meta-atributos. Além disso, a *SMarty* possibilita a extensão dos elementos que pertencem ao metamodelo da UML, diferentemente de outras abordagens, que não o fazem.

Desta forma, o perfil *SMartyProfile* aplica os conceitos de pontos de extensão da UML em seus modelos, o que o torna compatível com os metamodelos da UML, bem como no uso de ferramentas automatizadas ou de apoio, nas quais manipulam arquivos XML *Metadata Interchange* (XMI), abrangendo modelos UML.

O Processo *SMartyProcess*

O *SMartyProcess* adota suas atividades comuns relacionadas às definidas no processo de desenvolvimento de LPS (Pohl *et al.*, 2005) (SEI, 2012a). Este relacionamento é representado na Figura 2.2 por meio de um diagrama de atividades da UML, no qual mostra o processo genérico de Desenvolvimento de Linha de Produto, ilustrado pelas atividades alinhadas no lado esquerdo e o *SMartyProcess* simbolizado pelas atividades do retângulo à direita (OliveiraJr *et al.*, 2005).

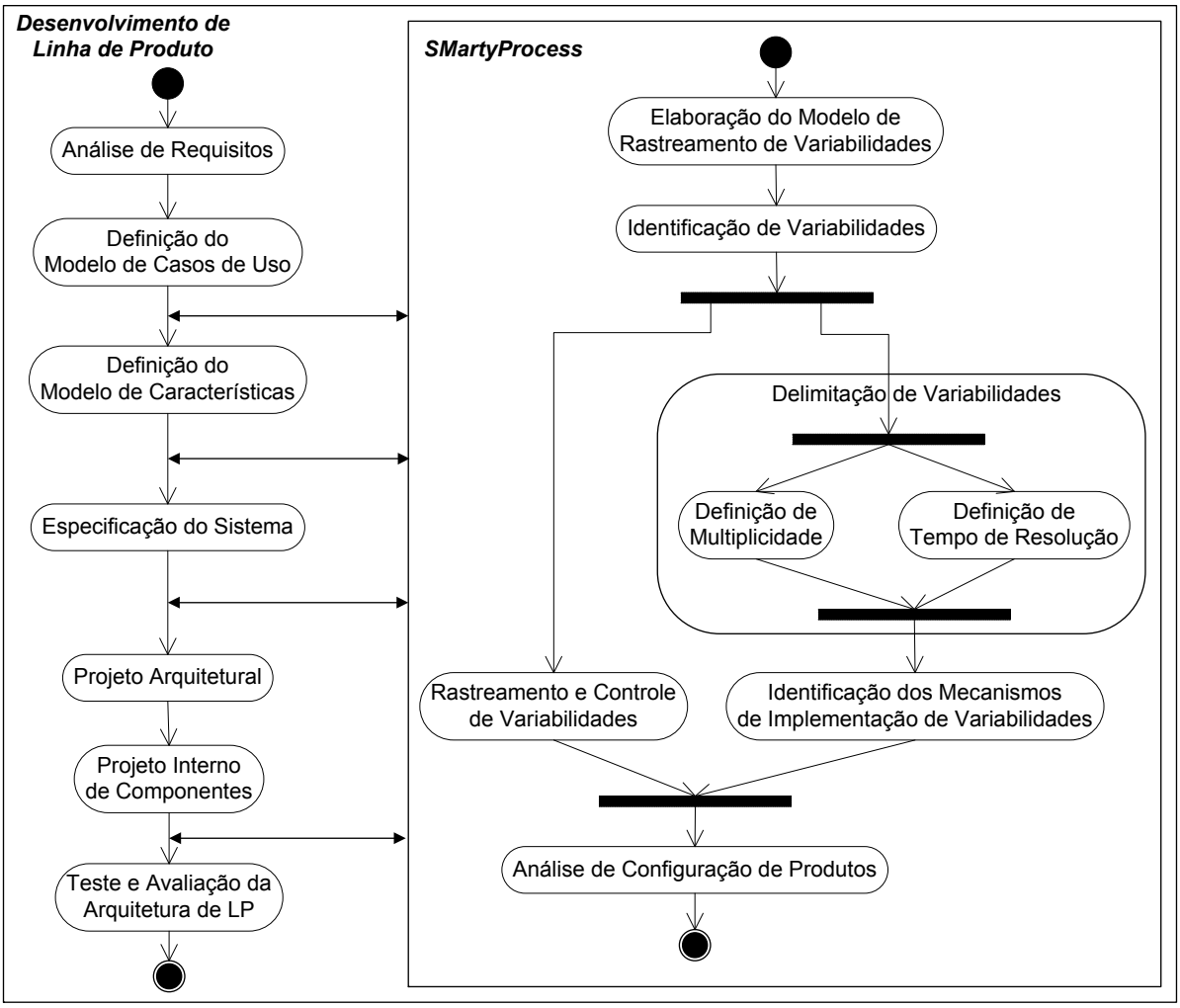


Figura 2.2: O Processo de Gerenciamento de Variabilidades *SMartyProcess* e sua Interação entre as Atividades com o Processo de Desenvolvimento de LPS, traduzido de (OliveiraJr *et al.*, 2005; OliveiraJr, 2010).

No estudo de OliveiraJr (2010), definições formais e estereótipos foram propostos para a *SMarty* por meio do *SMartyProfile*. Com isso, de maneira orquestrada, a *SMarty* faz a combinação do perfil *SMartyProfile* e do processo *SMartyProcess*, formando uma abordagem guiada por diretrizes para gerenciar sistematicamente variabilidades de LPS.

O processo *SMartyProcess* é realizado pelo engenherio de LPS e é um processo iterativo e incremental. Assim, quando o *SMartyProcess* ocorre após a execução de cada atividade do desenvolvimento de LPS, o mesmo é iterativo e quando a quantidade de variabilidades tende a crescer, à medida que as atividades do *SMartyProcess* são executadas, o mesmo é incremental.

O *SMartyProcess* absorve os elementos do núcleo de artefatos de uma LPS, bem como o alimenta com outros. Assim, por meio de cada ciclo, entre o Desenvolvimento de LPS e o *SMartyProcess*, pode-se identificar progressivamente as variabilidades associadas para os modelos de casos de uso, classes, atividades, componentes e sequência, utilizando a atividade de identificação de variabilidades, na qual recebe como entrada cada um destes modelos.

Entretanto, para realizar a identificação de variabilidades é necessário um conhecimento específico de analistas e gerentes de LPS, pois tal atividade que depende fortemente do domínio. Perante a isto, o *SMartyProcess* fornece um conjunto de diretrizes com o objetivo de apoiar e realizar com sucesso a atividade de gerenciamento de variabilidades (Fiori *et al.*, 2012; OliveiraJr *et al.*, 2010, 2013).

Por conseguinte, as diretrizes gerais do *SMartyProcess* são apresentadas para a identificação e representação de variabilidades aplicando o *SMartyProfile*.

É importante mencionar que tais diretrizes, específicas para casos de uso e classes, foram transcritas na íntegra dos estudos de Marcolino (2014) e OliveiraJr (2010), com o objetivo de manter a descrição correta, bem como a organização e o padrão estabelecido para estas diretrizes.

- **Diretrizes para Identificação e Representação de Variabilidades (RV)**

  RV.1 Variabilidades com variantes opcionais (<<optional>>) possuem multiplicidade `minSelection = 0` e `maxSelection = 1`;

  RV.2 Variabilidades com variantes exclusivas (<<alternative_XOR>>) possuem multiplicidade `minSelection = maxSelection = 1`;

  RV.3 Variabilidades com variantes inclusivas (<<alternative_OR>>) possuem multiplicidade `minSelection = 1` e `maxSelection = 1 = size(variants)` em que `size(x)` é uma função que retorna a quantidade de elementos da coleção `x`;

  RV.4 O valor `bindingTime` deve ser definido escolhendo-se um dos valores da classe de enumeração *BindingTime*, que são: DESIGN_TIME, LINK_TIME, COMPILE_TIME e RUNTIME;

  RV.5 O valor booleano do atributo `allowsAddingVar` deve ser analisado com base na possibilidade de manter o ponto de variação aberto (`true`) ou fechado (`false`); e

  RV.6 O valor da coleção variantes (`variants`) é o conjunto formado pelas instâncias das variantes associadas ao ponto de variação ou variabilidade.

- **Diretrizes para Modelos de Casos de Uso (UC)**

UC.1 Elementos de modelos de casos de uso relacionados aos mecanismos de extensão e de pontos de extensão sugerem pontos de variação (<<variationPoint>>) com variantes associadas, as quais podem ser inclusivas (<<alternative_OR>>) ou exclusivas (<<alternative_XOR>>);

UC.2 Modelos de casos de uso com o relacionamento de inclusão (<<include>>) ou associados a atores sugerem variantes obrigatórias (<<mandatory>>) ou opcionais (<<optional>>);

UC.3 Variantes que, ao serem selecionadas para fazer parte de um produto, exigem a presença de outra(s) determinada(s) variante(s) devem ter seu relacionamentos de dependência marcados com o estereótipo <<requires>>;

UC.4 Variantes mutuamente exclusivas para um determinado produto devem ter seus relacionamentos de dependência marcados com o estereótipo <<mutex>>.

- **Diretrizes para Modelos de Classes (CL)**

CL.1 Em modelos de classes, pontos de variação e suas variantes são identificadas nos seguintes relacionamentos:

a) generalização, os classificadores mais gerais são os pontos de variação, enquanto os mais específicos são as variantes;

b) realização de interface, os "*suppliers*" (especificações) são os pontos de variação e as implementações (clientes) são as variantes;

c) agregação, as instâncias tipadas com losangos não preenchidos são os pontos de variação e as instâncias associadas são as variantes; e

d) composição, as instâncias tipadas com losangos preenchidos são os pontos de variação e as instâncias associadas são as variantes.

CL.2 Elementos de modelos de classes, relacionados a associações nas quais os seus atributos `aggregationKind` possuem valor *none*, ou seja, não representam, nem agregação, nem composição, sugerem variantes obrigatórias ou opcionais;

CL.3 Variantes em modelos de classes, que ao serem selecionadas para fazer parte de um produto, exigem a presença de outras determinada(s) variante(s) devem ter seus relacionamentos de dependência marcados com o estereótipo <<requires>>;

CL.4 Variantes em modelos de classes, mutuamente exclusivas para um determinado produto devem ter seus relacionamentos de dependência marcados com o estereótipo <<mutex>>.

2.3 Inspeção de Software

Um software pode incorporar diferentes tipos de defeitos que devem ser detectados e removidos. Tais defeitos podem aumentar consideravelmente os custos de desenvolvimento e manutenção de um projeto de software (Boehm e Basili, 2001; Cunha *et al.*, 2012; Mello *et al.*, 2014).

Neste contexto, atividades de verificação e validação e análise estática e dinâmica são essenciais para contribuir com o controle da qualidade de um projeto de software (Pressman, 2010; Sommerville, 2011). Adicionalmente, as inspeções de software podem aumentar tal controle de qualidade de software de maneira plausível.

Além disso, em muitos contextos, as inspeções de software vêm sendo e têm sido importantes para garantir a qualidade de software, como uma parte, sob esforços empregados em distintos produtos de software (Alshazly *et al.*, 2014; Carver, 2004; Rombach *et al.*, 2008).

Assim, quando a inspeção de software é realizada nas fases iniciais do projeto de software, são detectados entre 60% a 90% dos defeitos principais nas fases iniciais do ciclo de vida do software, com uma média de detecção em torno de 80% (Boehm e Basili, 2001; Denger e Shull, 2007; Fagan, 2002, 1986; Gilb e Graham, 1993).

Portanto, o estado da arte da inspeção de software possui um histórico de diversos casos de sucesso na indústria (Carver, 2004; Rombach *et al.*, 2008), com resultados positivos obtidos e benefícios, que comprovam a sua eficácia, sendo retratada em várias publicações por meio de empresas como: a IBM (Fagan, 2002, 1986; Rombach *et al.*, 2008), a NASA (Hayes, 2003; Hayes *et al.*, 2006; Rombach *et al.*, 2008), a Motorola, a Allianz (Kristiansen, 2010; Rombach *et al.*, 2008), a AT&T Bell Labs (Godfrey, 1986), a Bell Northern Research (BNR) (Russell, 1991), a Bull HN Information Systems (Weller, 1993) e a Jet Propulsion Laboratory (JPL) (Kelly *et al.*, 1992) (Gilb e Graham, 1993).

Os benefícios da inspeção de software cresceram ao longo do tempo e englobam: redução do nível de defeitos; aumento na satisfação do cliente; aumento na produtividade de desenvolvimento, reduzindo a entrega do software; redução de custos de gerenciamento e do ciclo de vida do software; melhora contínua dos processos por meio da remoção de defeitos sistemáticos; melhor programação de horários para as reuniões de inspeção; treinamento mais rápido para desenvolvedores de novos produtos; e melhor experiência dos inspetores ao longo do tempo (Brykczynski *et al.*, 1994; Fagan, 2002, 1986; Gilb e Graham, 1993; Jones, 1996; Weller, 1993).

Com a apresentação do contexto acerca da área de inspeção de software e, a sua realidade benéfica de inspeções bem planejadas por grandes organizações, é importante

distinguir as definições sobre defeito, erro, falha e problema. Isto é necessário, antes da apresentação dos processos, técnicas e conceitos essenciais sobre a área, a fim de esclarecer que somente defeitos são tratados e, consequentemente, detectados durante o processo de inspeção de artefatos de software.

Tais definições seguem a terminologia da norma IEEE 1044-2009 - *Classification for Software Anomalies* (IEEE, 2010):

- **Problema**: dificuldade ou incerteza experimentada por uma ou mais pessoas, que resulta de um encontro insatisfatório com um sistema em uso. O significado de problema também pode estar relacionado com uma situação negativa para superar;

- **Erro**: uma ação humana que produz um resultado incorreto;

- **Falha**: cessação da capacidade de um produto para realizar uma função obrigatória ou a sua incapacidade de realizar dentro dos limites especificados previamente. O significado de falha também pode estar relacionado com um evento em que um sistema ou componente do sistema não realiza uma função obrigatória dentro dos limites especificados;

- **Defeito**: uma imperfeição ou deficiência em um produto de trabalho em que tal produto não atende seus requisitos ou especificações e precisa ser reparado ou substituído. Adicionalmente, um defeito é um tipo especial de falha, segundo (Alshazly *et al.*, 2014; Munson *et al.*, 2006).

De modo geral, pode-se definir a inspeção de software como um tipo específico de revisão de software (Ciolkowski *et al.*, 2003) aplicada em artefatos de software por meio de um processo sistemático e bem planejado de detecção de defeitos (Fagan, 2002, 1986; Kalinowski e Travassos, 2004; Mello *et al.*, 2012).

Por definição da norma IEEE 729-1983 - *Glossary of Software Engineering Terminology* (IEEE, 1983), inspeção de software é definida como uma técnica de avaliação formal em que os requisitos de software, projeto ou código são examinados em detalhes por uma pessoa ou um grupo com o objetivo de detectar defeitos, falhas e violações de padrões de desenvolvimento (Cheng e Jeffery, 1996; Gilb e Graham, 1993).

Logo, os objetivos da inspeção de software são definidos na norma IEEE 1028-1997 - *Software Reviews* (Anda e Sjøberg, 2002; Gilb e Graham, 1993; IEEE, 1998c): (i) verificar se um produto de software satisfaz a especificação; (ii) verificar se um produto de software satisfaz os atributos de qualidade especificados; (iii) verificar se um produto de software atende as regulamentações aplicáveis, normas, diretrizes, planos e procedimentos; (iv)

identificar desvios com relação a normas e especificações; (v) coletar dados da engenharia de software, por exemplo, dados sobre anomalias e esforços; e (vi) utilizar dados da engenharia de software para melhorar o processo de inspeção e a documentação de apoio, por exemplo, *checklists*.

Além da definição e dos objetivos da inspeção de software apresentados, a área é composta de diversos processos e técnicas de detecção de defeitos (Seção 2.3.2) para artefatos de software.

Dentre os principais processos existentes na literatura, estão: *Fagan Defect-Free Process*™ (Fagan, 2002, 1986), *Inspection / Agile Specification Quality Control* (*Spec QC*) (Gilb, 2005, 2009) e *IBM Orthogonal Defect Classification* (ODC) (Chillarege *et al.*, 1992; Munson *et al.*, 2006).

Um dos processos mais importantes historicamente foi proposto por Fagan (1986) de acordo com Gilb e Graham (1993). Definido por meio de papéis (moderador, inspetor e autor) e atividades (avaliação de artefatos), tal processo foi reorganizado e atualizado posteriormente por Kalinowski e Travassos (2004); Sauer *et al.* (2000).

Neste sentido, os estágios do processo de inspeção de Fagan (1986) são usualmente retratados nos estudos de Ebenau e Strauss (1994); Kollanus e Koskinen (2009); Shull *et al.* (2000), no qual possui os seguintes estágios definidos: Planejamento; Detecção ou Descoberta; Coleção; Discriminação; Retrabalho e Continuação.

Dentre os estágios mencionados com relação ao processo de inspeção de Fagan, esta pesquisa engloba somente os estágios de Planejamento e Detecção, a fim de contribuírem com a proposta de técnica presente nesta dissertação.

Além da existência de processos de inspeção de software, bem como de técnicas de detecção de defeitos, a próxima Seção 2.3.1, apresenta diversas taxonomias e classificações de tipos de defeitos presentes em estudos da literatura.

2.3.1 Taxonomias e Classificações de Tipos de Defeitos

A literatura apresenta diversas propostas de taxonomias de tipos de defeitos, as quais normalmente são criadas e adaptadas por meio de outros estudos que obtiveram resultados eficazes (Alshazly *et al.*, 2014; Basili *et al.*, 1995; Boehm e Basili, 2001; Hayes *et al.*, 2006; Kelly *et al.*, 1992; Rombach *et al.*, 2008; Russell, 1991; Shull *et al.*, 2000; Weller, 1993) em inspeções de artefatos de software por meio da utilização de tipos distintos de defeitos. Segundo Denger e Shull (2007), os inspetores devem definir um conjunto de alto nível de categorias de defeitos, escolhidos de acordo com as qualidades dos produtos finais desejados.

Neste sentido, com a finalidade de estabelecer padrões, o *Institute of Electrical and Electronics Engineers* (IEEE) é mantenedor de diversas normas, as quais definem procedimentos, recomendam práticas e definem regras para serem seguidas com o intuito de manter a organização e a qualidade. Assim, dentre as normas IEEE, existem as seguintes: IEEE 1012-2012 - *System and Software Verification and Validation* (IEEE, 2012) e a IEEE 830-1998 - *Recommended Practice for Software Requirements Specifications* (IEEE, 1998a).

A norma 1012-2012, define processos sobre verificação e validação de um sistema ou software. Tal norma cita a inspeção de software e alguns tipos de defeitos utilizados nas fases de concepção, de projeto, de requisitos, de testes e de código fonte na inspeção: *Complete*, *Correct*, *Omissions*, *Ambiguities*, *Consistent*, *Traceable* e *Testable*.

Já a norma 830-1998, recomenda práticas para especificações de requisitos de software, além de apresentar algumas características para uma boa especificação de requisitos (Wiegers, 2006). Esta especificação deve ser: *Correct*, *Unambiguous*, *Complete*, *Consistent*, *Ranked for importance and/or stability*, *Verifiable*, *Modifiable* e *Traceable*.

As normas mencionadas são relevantes no intuito de colaborarem com a taxonomia de tipos de defeitos da técnica proposta nesta dissertação (Seção 3.3). A Tabela 2.1 apresenta estudos com taxonomias e classificações adaptadas a partir de tais normas.

Tabela 2.1: Estudos de Taxonomias e Classificações de Tipos de Defeitos

Estudo	**Tipos de Defeitos**
(Kelly *et al.*, 1992)	Clarity, Correctness/Logic, Completeness, Consistency, Functionality, Compliance, Maintenance, Level of Detail, Traceability, Reliability e Performance.
(Hayes *et al.*, 2006; Miller *et al.*, 1995)	Requirements, Incompleteness, Omitted/Missing, Incorrect, Ambiguous, Infeasible, Inconsistent, Over-specification, Non-Treaceble, Non-Verifiable, Misplaced, Intentional Deviation e Redundant.
(Travassos *et al.*, 2001, 1999)	Omission, Incorrect Fact, Inconsistency, Ambiguity e Extraneous Information.
(Anda e Sjøberg, 2002)	Omissions, Incorrect Facts, Inconsistencies, Ambiguities, Extraneous Information e Consequences.
(Denger e Paech, 2004)	Incorrectness, Incompleteness, Inconsistency, Ambiguity, Incomprehensibility/Intraceability, Intestability, Inchangeability, Infeasibility e Over-specification.
(Lamsweerde, 2009; Souza *et al.*, 2013)	Incompleteness or Omission, Ambiguity, Incorrectness or Inadequacy, Inconsistency or Contradiction, Non-Traceability or Opacity, Incomprehensibility or Unintelligibility, Non-Organization or Poor structuring, Unnecessary Information or Over Specification e Business Rule.
(Alshazly *et al.*, 2014; Margarido *et al.*, 2009)	Missing or Incomplete, Incorrect Information, Inconsistent, Ambiguous or Unclear, Misplaced, Infeasible or Non-verificable, Redundant or Duplicate, Typo or Formatting e Not Relevant or Extraneous.

Os experimentos conduzidos por Kelly *et al.* (1992), durante as 203 inspeções realizadas ao longo de três anos no laboratório da *Jet Propulsion Laboratory* (JPL), fundado pela NASA, indicam uma melhora geral na qualidade do software, além de reduzir o retrabalho e os esforços de manutenção. Deste modo, as análises feitas nas investigações de Kelly *et al.* (1992) relacionam a densidade de defeitos encontrados com o tipo das inspeções que foram conduzidas.

Com base nos tipos de defeitos existentes, Miller *et al.* (1995) propõem diretrizes para verificação e validação de softwares especialistas ou softwares convencionais. Além das diretrizes, uma quantidade considerável de tipos de defeitos para software é proposta neste estudo, os quais foram utilizados e adaptados em uma taxonomia de falhas específica para inspeções de softwares da NASA *International Space Station* (Estação Espacial Internacional) com sucesso (Hayes, 2003; Hayes *et al.*, 2006).

Segundo Travassos *et al.* (1999), os artefatos de software são norteados por fatores externos com múltiplas interpretações, por conhecimentos gerais e específicos de domínios diferentes, além de requisitos. Com isso, os tipos de defeitos advém de tais fontes de informações consideradas relevantes.

Neste contexto, uma taxonomia de tipos de defeitos foi proposta por Travassos *et al.* (1999) com definições específicas para projetos orientados a objetos, visando inspecionar diagramas UML (Travassos *et al.*, 2001), como por exemplo: Diagramas de Classes, de Pacotes e de Interação. A taxonomia proposta por Travassos *et al.* (1999), foi adaptada da taxonomia de defeitos de requisitos de Basili *et al.* (1995) por ser eficaz, como afirmam Travassos *et al.* (1999).

Além disso, Anda e Sjøberg (2002) propõem uma taxonomia de tipos de defeitos, a qual foi adaptada da taxonomia de defeitos de requisitos proposta no estudo de Shull *et al.* (2000). Deste modo, a taxonomia de Anda e Sjøberg (2002) pode ser utilizada na inspeção de artefatos de software, além de definir uma especificação para inspecionar diagramas UML de caso de uso.

De maneira complementar ao estudo de Anda e Sjøberg (2002), o trabalho realizado por Denger e Paech (2004) propõe também uma taxonomia de tipos de defeitos para a inspeção de diagramas UML de casos de uso, a qual foi adaptada de acordo com a norma IEEE (1998a).

Outro estudo significativo é o realizado por Lamsweerde (2009), no qual propôs uma classificação sobre *non-conformities* (não conformidades) (Card, 1998), defeitos classificados em nove tipos baseados na especificação de requisitos (IEEE, 1998a,c) com o intuito de inspecionar características de LPSs (Souza *et al.*, 2013). Em adição, o estudo

realizado por Salinesi e Mazo (2012) busca propor alternativas de como identificar defeitos em LPS, mas nenhuma taxonomia é apresentada ou definida para este fim.

A revisão da literatura realizada por Margarido *et al.* (2009), permitiu a classificação dos tipos de defeitos de especificações de requisitos por meio do estudo de taxonomia de tipos de defeitos, identificando a frequência da ocorrência e uso dos mesmos (Alshazly *et al.*, 2014).

2.3.2 Técnicas de Detecção de Defeitos

Dentre as técnicas de revisão de software existentes, estão as principais técnicas de detecção de defeitos, também conhecidas como técnicas de leitura, as quais são:

- Técnica de Leitura Baseada em *Checklist* (*Checklist-Based Reading (CBR)*) (Alshazly *et al.*, 2014; Anda e Sjøberg, 2002; Biffl e Halling, 2000; Brykczynski, 1999; Cunha *et al.*, 2012; Laitenberger, 2002; Mello *et al.*, 2014, 2012);
- Técnica de Leitura Baseada em Perspectivas (*Perspective-Based Reading (PBR)*) (He e Carver, 2006; Laitenberger, 2002; Yang, 2007);
- Técnica *Ad hoc* (Basili *et al.*, 1996; Cox *et al.*, 2004a; Rombach *et al.*, 2008);

Os principais conceitos das técnicas CBR e *Ad hoc* são apresentados a seguir. A técnica CBR é a base da técnica proposta neste trabalho, enquanto a técnica *Ad hoc* é utilizada na avaliação empírica da *SMartyCheck*.

Técnica de Detecção de Defeitos baseada em Checklist

A Técnica de Leitura Baseada em *Checklist* (*Checklist-Based Reading (CBR)*) é uma técnica não-sistemática, a qual não fornece instruções ou mecanismos sobre "como" a inspeção deve ser realizada, somente apresenta aos inspetores o "que" deve ser inspecionado por meio de orientações utilizando-se de um documento no formato de uma lista de verificação, um *checklist* (Alshazly *et al.*, 2014; Brykczynski, 1999; He e Carver, 2006).

Os inspetores recebem um *checklist* com os respectivos itens para verificação descritos em forma de perguntas ou afirmações, no intuito de procurar e detectar possíveis defeitos (Staron *et al.*, 2005). Normalmente, o revisor/inspetor faz a leitura deste documento (*checklist*) respondendo a cada questionamento em uma lista de questões assertivas (sim/não) por meio de um *check*, no qual cada questão deve ser clara para ser respondida por meio de uma única verificação perante o conhecimento prévio dos inspetores (Lanubile e Visaggio, 2000).

Recomenda-se que durante a elaboração do *checklist*, o mesmo deva ser realista e preciso, além da possibilidade de ser baseado em um ou dois cenários, segundo Eliza e Lagares (2012):

- ***Checklist* Baseado no Histórico**: quais defeitos foram registrados pelos inspetores nas últimas iterações; e

- ***Checklist* Baseado na Experiência**: conforme a experiência dos inspetores envolvidos em determinada inspeção.

A técnica CBR possui restrições, vantagens e desvantagens como toda técnica em geral (Alshazly *et al.*, 2014; Aurum *et al.*, 2002; Brykczynski, 1999). Uma grande vantagem da CBR é fornecer orientações de maneira sucinta sobre o "que" deve ser inspecionado (Ciolkowski, 2009). O *checklist* também auxilia na identificação de quais informações são relevantes sobre determinadas tarefas e o problema, o qual está sendo tratado (Alshazly *et al.*, 2014; Cunha *et al.*, 2012; Mello *et al.*, 2014, 2012). Além disso, a técnica CBR é uma das técnicas mais adotadas e difundidas para análise de artefatos de software e requisitos (Fagan, 2002; Kristiansen, 2010; Rombach *et al.*, 2008).

Segundo o estudo de Rombach *et al.* (2008), o *checklist* é utilizado pelas organizações ou empresas em média de 50% das vezes, já a técnica *Ad hoc* fica com o percentual em torno de 40%, enquanto técnicas de leitura mais complexas possuem 10%. Nestes percentuais estão presentes organizações, que utilizam o *checklist*, como a IBM (Fagan, 1986; Rombach *et al.*, 2008), a NASA (Hayes, 2003; Hayes *et al.*, 2006), a Motorola e a Allianz (Rombach *et al.*, 2008), além de outras organizações que aplicam o *checklist* com sucesso como a principal técnica de leitura para inspeções. Estas empresas obtiveram na prática taxas de 95% de eficácia na redução de defeitos antes do teste, reduzindo custos de desenvolvimento de código em torno de 50%, bem como na redução do tempo de entrega do software, com o percentual na faixa de 50% (Kristiansen, 2010; Rombach *et al.*, 2008).

Diversos benefícios são evidenciados na literatura com relação a eficiência da técnica CBR, quando somada à procedimentos para a realização de uma inspeção de software bem planejada (Anda e Sjøberg, 2002; Basili *et al.*, 1996; Biffl e Halling, 2000; Brykczynski, 1999; Cox *et al.*, 2004a; Cunha *et al.*, 2012; Denger e Paech, 2004; Gilb e Graham, 1993; Kelly *et al.*, 1992; Mello *et al.*, 2014, 2012; Rombach *et al.*, 2008; Sabaliauskaite *et al.*, 2004; Staron *et al.*, 2005):

- redução do nível de defeitos de um software de modo relevante e considerável. A simplicidade e a precisão de questões bem elaboradas e objetivas permite almejar este benefício;

- redução de custos referente ao ciclo de vida e gerenciamento do software. Os custos são minimizados com a detecção de defeitos e, consequentemente, obtendo a redução no nível (quantidade) de vários tipos de defeitos detectados;

- redução no tempo de teste e de entrega do software. Isto ocorre, pois os defeitos foram identificados antes mesmo do software ser executado e/ou na fase de codificação (desenvolvimento), além de diminuir algum tipo de adaptação ou manutenção futura no mesmo software; e

- melhora da produtividade em equipe, reduzindo a escala de tempo de desenvolvimento. Considerando que o software tenha menos defeitos a equipe se concentra em outras fases, tarefas ou atividades com relação ao projeto do software.

Contudo, a técnica CBR possui algumas limitações, as quais são listadas a seguir (Alshazly *et al.*, 2014; He e Carver, 2006; Laitenberger, 2002; Laitenberger *et al.*, 2001; Yang, 2007):

- o *checklist* é de caráter geral na maioria dos casos. A simplicidade e a má adaptação de algum tipo de *checklist* de outras fontes da literatura pode fazer com que o mesmo não seja específico;

- o *checklist* é limitado em tipos específicos de defeitos. Assim, o inspetor pode se concentrar em encontrar tipos de defeitos inexistentes e adquirir certo "vício" em utilizar os mesmos tipos de defeitos que acha mais plausíveis para as mesmas situações durante a inspeção;

- o *checklist* tem caráter repetitivo. O inspetor pode desencadear certo esforço por utilizar uma mesma lista de verificação para várias inspeções, causando certa repetição, fadiga e, consequentemente, diminuir a sua produtividade e os resultados obtidos; e

- o *checklist* não garante a qualidade de software por meio de algum princípio. O inspetor não sabe "como" a inspeção deve ser conduzida, apenas o "que" deve ser inspecionado.

Técnica *Ad hoc*

A técnica *Ad hoc* não fornece qualquer tipo de orientação, não possui procedimentos ou mecanismos claros para os inspetores realizarem a leitura dos artefatos de software de maneira correta (Aurum *et al.*, 2002; Sabaliauskaite *et al.*, 2004). Desta forma, a inspeção depende do ponto de vista do inspetor, do seu conhecimento e experiência na detecção de defeitos em artefatos de software (Aurum *et al.*, 2002).

Contudo, tal técnica é uma das técnicas mais difundidas por ser "informal" segundo Rombach *et al.* (2008), ou seja, sem critérios sistemáticos para realizar inspeções em artefatos de um software em específico, diferentemente das técnicas CBR e PBR.

Estudos indicam que de fato a técnica *Ad hoc* não se aproxima da eficácia, na maioria das vezes, das técnicas como a CBR ou a PBR (Basili *et al.*, 1996; Cox *et al.*, 2004a; Lanubile e Visaggio, 2000; Mello *et al.*, 2014, 2012).

2.3.3 Técnicas de Inspeção de Software para LPS

SPLIT: uma Técnica de Inspeção para LPS

Um dos estudos recuperados e selecionados no estudo secundário (Apêndice A) realizado e considerado relevante a este trabalho é o de Cunha *et al.* (2012). Neste estudo é proposto um conjunto de técnicas de inspeção baseadas em *checklist* denominado *Software Product Line Inspection Techniques* (SPLIT) com o objetivo de avaliar modelos de LPS. Tal técnica possui como proposta a comparação do Documento de Requisitos de Software, Mapa de Produtos e Modelo de Características (Cunha *et al.*, 2012). SPLIT engloba:

- a técnica 1, que analisa o mapa de produtos comparando com o documento dos requisitos de software conforme os seguintes tipos de defeitos:
  - Redundância: o mapa de produtos apresenta produtos que tenham o mesmo conjunto de características; e
  - Inconsistência: o mapa de produtos apresenta inconsistências quando ele não corresponde aos produtos que estão no documento de requisitos.
- a técnica 2, que analisa o mapa de produtos e o modelo de características, comparando com o documento de requisitos de software. Apenas um tipo de defeito é considerado:
  - Inconsistência: os artefatos da LPS (mapa de produtos e modelo de características) apresentam produtos que não correspondem ao documento de requisitos. Essa técnica difere da primeira devido à informação proveniente estritamente do modelo de características.
- a técnica 3, que analisa o modelo de características buscando por defeitos, os quais são:
  - Redundância: se pelo menos uma informação semântica é modelada de várias maneiras;

  - Anomalia: se configurações em potencial estão sendo perdidas, embora estas configurações deveriam ser possíveis; e
  - Inconsistência: se o modelo de características possui informações contraditórias.

Porém, percebe-se que um conjunto reduzido de tipos de defeitos é considerado pela técnica SPLIT. Por sua vez, *SMartyCheck* engloba 11 tipos distintos de defeitos (Seção 5.7) para diagramas UML. Além disso, uma diferença importante identificada entre as técnicas *SMartyCheck* e a SPLIT de Cunha *et al.* (2012), foi com relação a aplicação da inspeção em artefatos diferentes. *SMartyCheck* faz a inspeção de diagramas UML *SMarty* de casos de uso e de classes de LPS. Diferentemente, a técnica SPLIT compara o Documento de Requisitos de Software, o Mapa do Produto e o Modelo de Características de uma LPS.

FMCheck: uma Técnica de Inspeção para Modelos de Características

Outro estudo significativo é o de Mello *et al.* (2012), o qual propõe uma técnica de inspeção baseada em *checklist* para apoiar a identificação de defeitos em Modelos de Características de LPS, denominada FMCheck. Tal técnica possui um conjunto de 3 atividades:

- a atividade 1 é realizada pelo analista do domínio. Consiste no preenchimento do questionário de caracterização do modelo, que visa auxiliar na elaboração do *checklist* para aplicação em um contexto específico;
- a atividade 2 é realizada pelo moderador da inspeção. Consiste na configuração do *checklist*, levando em consideração o questionário preenchido e uma tabela de rastreabilidade definida na técnica; e
- a atividade 3 consiste na realização de pelo menos uma ou várias inspeções individuais e, consequentemente, no preenchimento do relatório de discrepâncias correspondente a cada inspeção.

O estudo realizado por Mello *et al.* (2012) detectou, por meio da técnica FMCheck, uma quantidade de defeitos consideráveis com relação a técnica de inspeção *ad hoc*. A técnica FMCheck detectou 53 defeitos distintos, os quais não foram detectados pelas inspeções *ad hoc*, sendo que, as inspeções *ad hoc* detectaram apenas 11 defeitos distintos, sendo que estes, não foram também detectados pela técnica FMCheck. No entanto, foi observado que os itens de verificação da técnica FMCheck cobrem 10 dos 11 defeitos relatados apenas nas inspeções *ad hoc*, o que sugere uma grande cobertura dos itens de

verificação da técnica FMCheck quanto ao contexto avaliado. Observou-se também que a técnica FMCheck contribui, principalmente, para a detecção dos seguintes tipos de defeitos: omissões, fatos incorretos e ambiguidades (Mello *et al.*, 2014, 2012).

Entretanto, a técnica FMCheck possui uma taxonomia de 5 tipos de defeitos em comparação com a *SMartyCheck* que engloba 11 tipos distintos de defeitos para diagramas UML (Seção 5.7). Além disso, existe uma diferença significativa na inspeção entre as técnicas FMCheck de Mello *et al.* (2014, 2012) e a *SMartyCheck* de acordo em qual artefato se aplica o *checklist*, sendo que, na FMCheck, o mesmo é aplicado no Modelo de Características e na *SMartyCheck* é aplicado em diagramas UML *SMarty* de casos de uso e de classes de LPS.

2.4 Considerações Finais

Este capítulo apresentou os fundamentos teóricos necessários para o desenvolvimento deste trabalho, bem como para o posicionamento da técnica proposta: a *SMartyCheck* (Capítulo 3). Neste sentido, acerca dos temas apresentados e revisados neste capítulo, a compreensão das definições sobre perfis UML, diagramas de casos de uso e de classes (Apêndice B) auxiliam a compreender a proposta e o poder da abordagem *SMarty* (Seção 2.2.1), na qual foi avaliada por meio de experimentos, indicando sua efetividade para identificação e representação de variabilidades de LPS. Portanto, a viabilidade de propor a técnica *SMartyCheck* para inspecionar diagramas UML *SMarty* de LPS se torna interessante e necessário, pois poucos trabalhos sobre técnicas de inspeção de software para LPS foram identificados na literatura.

Além disso, os principais conceitos sobre inspeção de software, taxonomias e classificações de tipos de defeitos foram apresentados. Isto possibilitou compreender o panorama da área com o intuito de propor a técnica *SMartyCheck*. Ainda, um estudo secundário (Apêndice A) foi realizado com o objetivo de identificar e selecionar estudos primários relevantes sobre os tipos de defeitos existentes na literatura, a fim de adaptá-los na proposta da *SMartyCheck*. Contudo, nenhum dos estudos primários mapeados têm tipos de defeitos segmentados e/ou possibilita a inspeção de diagramas UML de casos de uso e de classes de LPS. Assim, optou-se por propor a técnica *SMartyCheck*.

Em síntese, pode-se observar, por meio dos estudos apresentados e discutidos sobre técnicas de detecção de defeitos, que as mesmas são totalmente dependentes do cenário em que uma inspeção de software é realizada, seja por meio de experimentos conduzidos em diferentes contextos ou por meio de estudos de casos planejados.

3

SMartyCheck 2.0: uma Técnica de Inspeção para Diagramas UML de LPS

"Transportai um punhado de terra todos os dias e fareis uma montanha."

Confúcio (551 - 479 a.C), Filósofo Chinês

3.1 Considerações Iniciais

Neste capítulo é apresentada a proposta da técnica *SMartyCheck* para a inspeção de software baseada em *checklist* para diagramas *SMarty*. Essa técnica faz o uso da abordagem *SMarty* com o intuito de inspecionar diagramas UML de variabilidade de LPS por meio da combinação de um *checklist* com uma taxonomia de tipos de defeitos adotados de estudos relevantes mapeados e selecionados da literatura.

O estudo secundário realizado sobre tipos de defeitos em técnicas de inspeção de software (Apêndice A) recuperou diversos estudos primários. Alguns estudos possibilitaram a adoção dos tipos de defeitos para a proposta da taxonomia de tipos de defeitos da técnica *SMartyCheck* (Seção 3.3).

A proposta inicial da *SMartyCheck* apresentada neste capítulo foi avaliada por meio da condução de dois estudos empíricos, um qualitativo (Capítulo 4) e um quantitativo

(Capítulo 5). Portanto, após a análise e interpretação dos resultados do estudo quantitativo o *checklist* da *SMartyCheck* foi aprimorado, sendo considerado o *checklist* final da técnica. Entretanto, tal *checklist* aprimorado não foi avaliado, necessitando de estudos empíricos no futuro.

Poucos estudos que propõem técnicas de inspeção de software para LPS foram identificados na literatura. Além disso, as técnicas atuais de inspeção de LPS (Cunha *et al.*, 2012; Mello *et al.*, 2014) não suportam a detecção de diferentes tipos de defeitos em diagramas UML. Tais fatos motivaram o desenvolvimento desta pesquisa. Assim, acredita-se que uma das principais contribuições desta pesquisa é a proposta da técnica *SMartyCheck*, com o objetivo de detectar vários tipos de defeitos nos diagramas UML *SMarty* de variabilidade de casos de uso e de classes.

Este capítulo apresenta a técnica *SMartyCheck*, que visa apoiar a inspeção de qualidade e melhorar a detecção de defeitos em diagramas UML *SMarty*. A Seção 3.2 apresenta a caracterização da técnica *SMartyCheck*. Por sua vez, a Seção 3.3 apresenta a taxonomia de tipos de defeitos da *SMartyCheck*. A Seção 3.4 descreve como a técnica é aplicada a diagramas UML *SMarty* de casos de uso e de classes. Já a Seção 3.5 discute os poucos trabalhos relacionados existentes na literatura.

3.2 Caracterização da Técnica *SMartyCheck 2.0*

Considerando o *framework* de engenharia de LPS (Seção 2.2), a técnica *SMartyCheck* está posicionada de acordo com as entradas e saídas de cada subprocesso da etapa da Engenharia de Domínio. Em tal etapa os diagramas *SMarty* de variabilidade alimentam a técnica *SMartyCheck*. Dentre os diagramas UML *SMarty* (Seção 2.2.1), *SMartyCheck* foi projetada para inspecionar casos de uso e classes.

A Figura 3.1 ilustra os dois subprocessos da Engenharia de Domínio de LPS com os diagramas *SMarty* de variabilidade, os quais são inspecionados por meio da *SMartyCheck*, conforme segue:

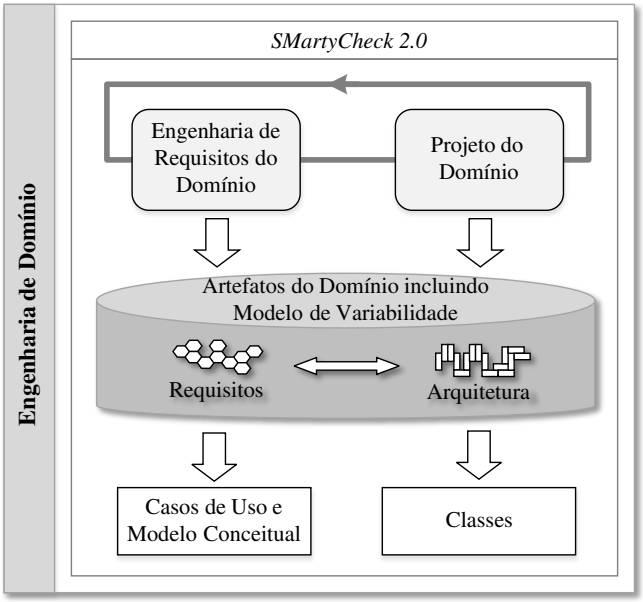


Figura 3.1: Os Subprocessos da Engenharia de Domínio de LPS e a Técnica *SMartyCheck*. Adaptado e traduzido de (Pohl *et al.*, 2005).

- **Engenharia de Requisitos do Domínio**: inspeção de diagramas de casos de uso e classes.
- **Projeto do Domínio**: inspeção de diagramas de classes em nível de projeto.

A técnica *SMartyCheck* considera somente a Engenharia de Domínio por conter variabilidade no núcleo de artefatos produzido. Diferentes versões de LPSs derivadas a partir de um oráculo de uma LPS podem ser inspecionadas. No entanto, a *SMartyCheck* não permite a inspeção e geração de produtos específicos destas LPS. Assim, não envolve a Engenharia de Aplicação.

A técnica *SMartyCheck* foi projetada com base na Técnica de Leitura Baseada em *Checklist* (*Checklist-Based Reading (CBR)*) (Seção 2.3.2), a fim de detectar defeitos em diagramas UML de casos de uso e de classes da *SMarty* de variabilidade de LPSs por meio de um *checklist*. Entretanto, cada diagrama possui um *checklist* para que a inspeção seja organizada e direcionada (Seções 3.4.1 e 3.4.2).

SMartyCheck apresenta ao inspetor "o quê" deve ser inspecionado. Para tanto, são fornecidas orientações ao inspetor, utilizando-se de um documento no formato de uma lista de verificação (*checklist*). O *checklist* contém os tipos de defeitos pré-definidos, os seus respectivos itens descritos, os locais para responder aos questionamentos por meio de um *check* (sim/não), além de um espaço para que observações sejam escritas pelos

inspetores de acordo com o defeito identificado. O formato do *checklist* é apresentado na Seção 3.4.

Neste âmbito, o *checklist* da *SMartyCheck* foi elaborado com a análise, adaptação e adição de tipos de defeitos extraídos dos estudos apresentados na Seção 2.3.1. Deste modo, uma taxonomia de tipos de defeitos foi projetada para técnica *SMartyCheck*, a qual é descrita em detalhes na Seção 3.3.

3.3 Taxonomia de Tipos de Defeitos da *SMartyCheck 2.0*

Na Seção 2.3.1 foram apresentadas as taxonomias e classificações existentes sobre tipos de defeitos utilizados por técnicas de inspeção de software.

Esta seção apresenta os tipos de defeitos encontrados na literatura (Apêndice A) adotados para projetar a taxonomia de tipos de defeitos para a técnica *SMartyCheck*.

De modo ordenado, os estudos são apresentados primeiramente e, na sequência, cada tipo de defeito é descrito com seu objetivo, bem como exemplificado. Deste modo, é possível compreender "o quê" é inspecionado nos diagramas UML *SMarty* de variabilidade:

- Tipos de defeitos adotados da norma IEEE 1012-2012 - *System and Software Verification and Validation* (IEEE, 2012) para a técnica *SMartyCheck*:

  - **Ambiguidade**: elementos de casos de uso ou de classes apresentam mais de uma ou várias interpretações. Isto requer que cada elemento tenha um nome diferente com relação ao seu objetivo. Por exemplo, com base na Figura 3.2, a classe *Size* não está de acordo, a qual foi alterada para *Big Size*.

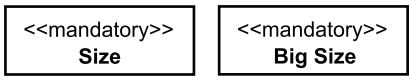


Figura 3.2: Exemplo Tipo de Defeito *SMartyCheck* - Ambiguidade.

  - **Incompleto**: ausência de um ou mais elementos obrigatórios de casos de uso ou de classes. Por exemplo, se um diagrama contém os casos de uso ou classes obrigatórias. De acordo com a Figura 3.3, a classe *Size* é obrigatória, mas não está especificada (cor cinza).

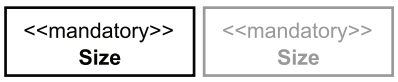


Figura 3.3: Exemplo Tipo de Defeito *SMartyCheck* - Incompleto.

- **Inconsistência**: refere-se à falta de consistência interna em que um subconjunto de elementos individuais de casos de uso ou de classes que têm conflitos nos diagramas *SMarty*. Por exemplo, se um elemento de casos de uso ou de classes não cumpre as restrições de modelagem dos diagramas *SMarty*. De acordo com a Figura 3.4, o caso de uso *Play Selected Game* possui quatro variantes, sendo que deveria possuir no máximo três, pois o meta-atributo $maxSelection = 3$.

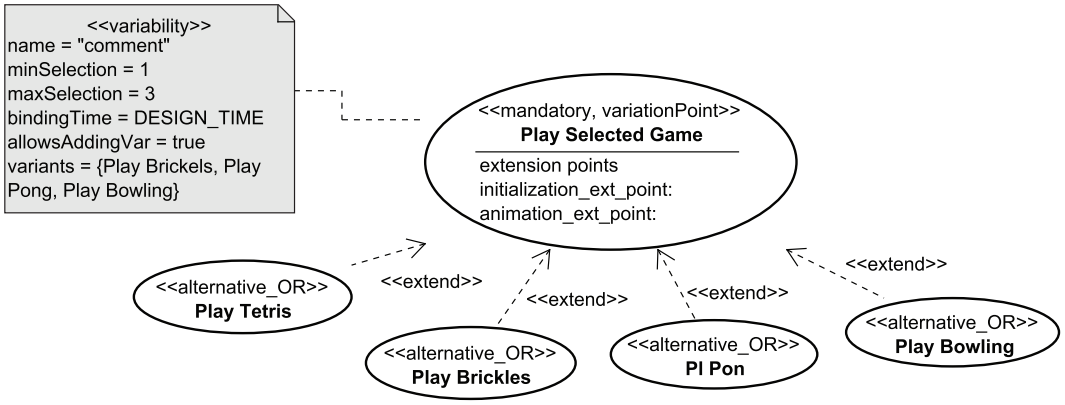


Figura 3.4: Exemplo Tipo de Defeito *SMartyCheck* - Inconsistência.

- **Incorreto**: cada elemento de casos de uso ou de classes deve atender aos relacionamentos definidos pela abordagem *SMarty*. Por exemplo, se uma associação pode ocorrer ou não entre elementos de casos de uso ou de classes. De acordo com a Figura 3.5, a classe *GameSprite* está associada com a classe *Menu*.

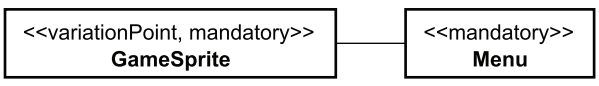


Figura 3.5: Exemplo Tipo de Defeito *SMartyCheck* - Incorreto.

- Tipos de defeitos adotados da norma IEEE 830-1998 - *Recommended Practice for Software Requirements Specifications* (IEEE, 1998a,c) para a técnica *SMartyCheck*:

- **Instável**: cada um dos elementos de casos de uso ou de classes têm um identificador único. Por exemplo, cada elemento deve ser identificado com um nome singular. De acordo com a Figura 3.6, as classes *MovableSprite* e *GameSprite* possuem o mesmo meta-atributo *name* = "comment".

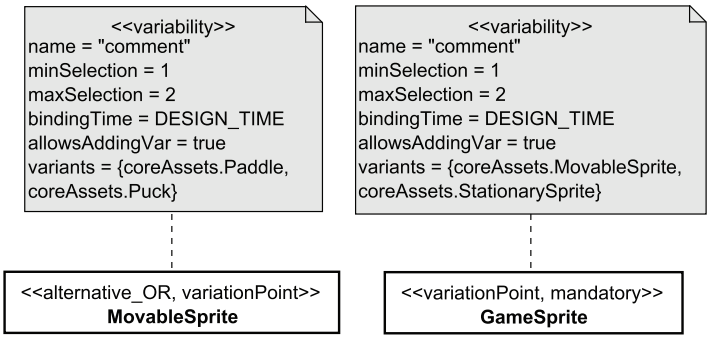


Figura 3.6: Exemplo Tipo de Defeito *SMartyCheck* - Instável.

- **Imodificável**: quando a estrutura de um ou vários elementos de casos de uso ou de classes pode ser mantida por meio de quaisquer mudanças feitas nos elementos. Por exemplo, os elementos devem ser especificados de maneira separada para serem coerentes e não redundantes. De acordo com a Figura 3.7, o caso de uso *Play Selected Game* possui variantes escolhidas e combinadas - *variants = Play Brickles*, *Play Pong*, *Play Bowling*. As variantes *Play Tetris* e *Pl Pong* foram especificadas de forma errônea.

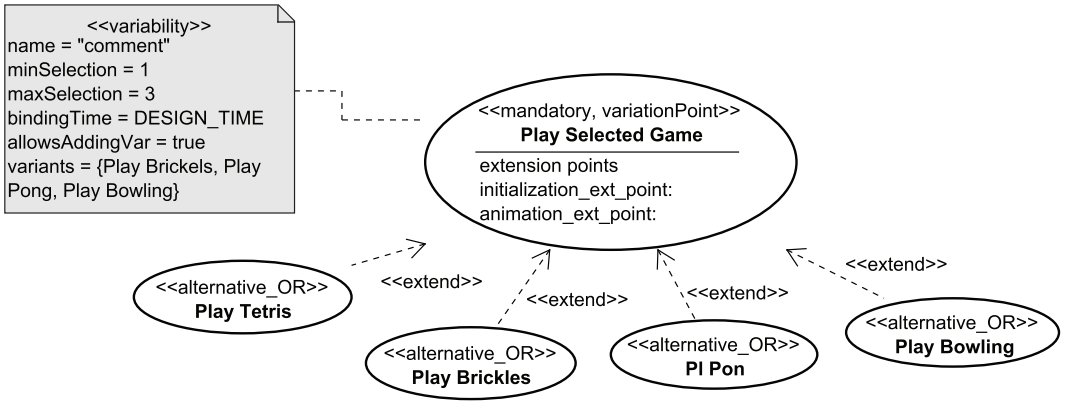


Figura 3.7: Exemplo Tipo de Defeito *SMartyCheck* - Imodificável.

- Tipos de defeitos adotados do estudo de Travassos *et al.* (1999):

  - **Fato Incorreto**: os nomes de um ou mais elementos de casos de uso ou de classes são contraditórios. Por exemplo, o nome de um ou mais elementos de

casos de uso ou de classes está incorreto. De acordo com a Figura 3.8, o caso de uso *Play Pong* possui o nome *Pl Pon*.

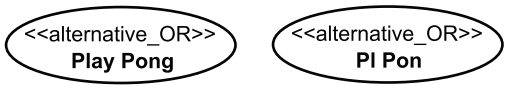


Figura 3.8: Exemplo Tipo de Defeito *SMartyCheck* - Fato Incorreto.

- **Informação Estranha**: os elementos de casos de uso ou de classes são redundantes. Por exemplo, podem existir elementos de casos de uso ou de classes duplicados ou específicados além dos diagramas *SMarty*. De acordo com a Figura 3.9, a classe *Velocity* está duplicada.

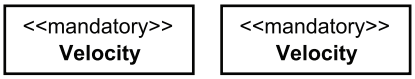


Figura 3.9: Exemplo Tipo de Defeito *SMartyCheck* - Informação Estranha.

- Tipos de defeitos adotados dos estudos de Miller *et al.* (1995) *apud* Hayes *et al.* (2006):

  - **Desvio Intencional**: elementos de casos de uso ou de classes requerem ou dependem de sua associação com outros elementos de acordo com os diagramas *SMarty* de variabilidade. Por exemplo, um elemento de casos de uso ou de classes depende de outro elemento. De acordo com a Figura 3.10, o caso de uso *Check Previous Best Score* exige a seleção do caso de uso *Save Score*.

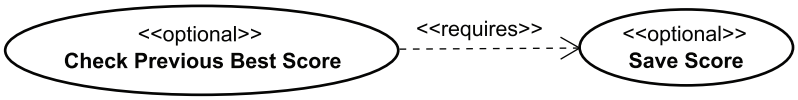


Figura 3.10: Exemplo Tipo de Defeito *SMartyCheck* - Desvio Intencional.

  - **Inviável**: elementos de casos de uso ou de classes podem ou não serem incluídos de acordo com as restrições definidas para as variantes com base na abordagem *SMarty*. Por exemplo, a adição de novos elementos de casos de uso ou de classes nos diagramas *SMarty*. De acordo com a Figura 3.11, o caso de uso *Play Selected Game* possui o meta-atributo $allowsAddingVar = \text{false}$, proibindo a inclusão de novas variantes, sendo que deveria ter três ($maxSelection = 3$).

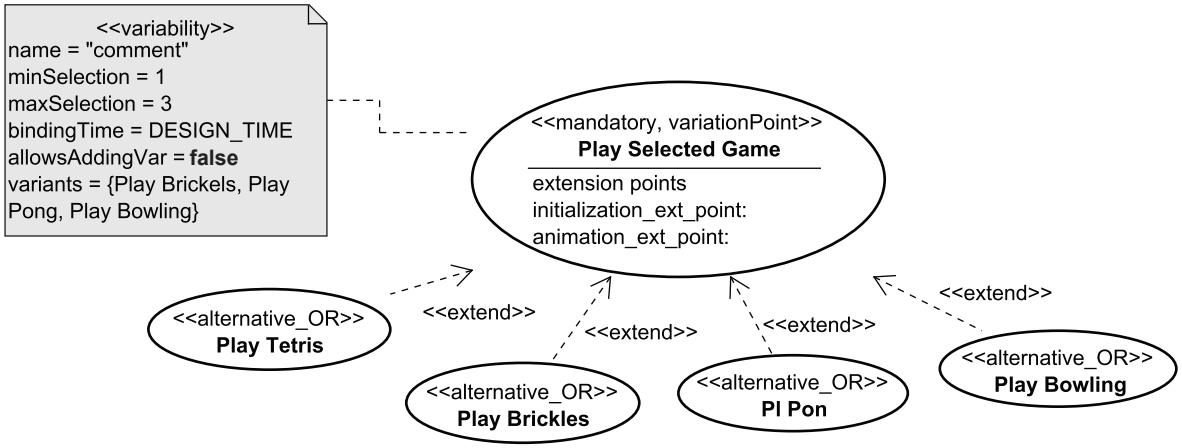


Figura 3.11: Exemplo Tipo de Defeito *SMartyCheck* - Inviável.

- Tipos de defeitos adotados dos estudos de Lamsweerde (2009) *apud* Souza *et al.* (2013):

  - **Omissão**: ausência ou falta de um ou vários elementos obrigatórios de casos de uso ou de classes. Por exemplo, um elemento não está presente na especificação das funcionalidades obrigatórias de acordo com os diagramas *SMarty*. De acordo com a Figura 3.12, os casos de uso obrigatórios *Exit Game*, *Initialization* e *Animation Loop* não estão especificados (cor cinza).

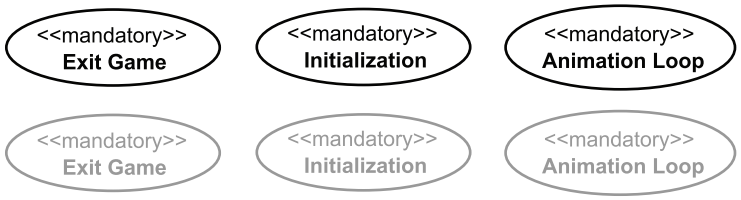


Figura 3.12: Exemplo Tipo de Defeito *SMartyCheck* - Omissão.

  - **Regras de Negócio**: Situação em que a definição das regras de negócio de um domínio específico de uma LPS pode ser incorretamente descrita e especificada de acordo com os diagramas *SMarty*. Por exemplo, os elementos de casos de uso ou de classes não cumprem com as funcionalidades com base nas regras de negócio pré-definidas.

- Tipo de defeito adotado de Cunha *et al.* (2012):

  - **Anomalia**: um ou mais elementos de casos de uso ou de classes estão associados(as) com outros elementos de forma incomum. Por exemplo, uma

associação incorreta entre os elementos. De acordo com a Figura 3.13, a classe *StationarySprite* está associada com a classe *Board*, a qual não é um ponto de variação.

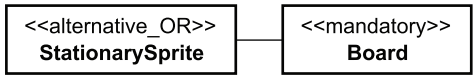


Figura 3.13: Exemplo Tipo de Defeito *SMartyCheck* - Anomalia.

A Tabela 3.1 relaciona os tipos de defeitos descritos, mostrando a fonte de onde foram adotados na taxonomia e no *checklist* da *SMartyCheck*:

Tabela 3.1: Tipos de Defeitos Adotados para a Técnica *SMartyCheck*.

Estudos	Tipos de Defeitos
IEEE (2012)	Ambiguidade, Completo (adotado como Incompleto), Consistência (adotado como Inconsistência) e Correto (adotado como Incorreto).
IEEE (1998a,c)	Estável (adotado como Instável) e Modificável (adotado como Imodificável).
Travassos *et al.* (1999)	Fato Incorreto e Informação Estranha.
Miller *et al.* (1995) *apud* Hayes *et al.* (2006)	Desvio Intencional e Inviável.
Lamsweerde (2009) *apud* Souza *et al.* (2013)	Regras de Negócio e Omissão.
Cunha *et al.* (2012)	Anomalia.

A Figura 3.14 ilustra a taxonomia de tipos de defeitos proposta para o *checklist* da técnica *SMartyCheck*.

Figura 3.14: Taxonomia de Tipos de Defeitos da Técnica *SMartyCheck*.

A próxima seção apresenta e justifica a equivalência entre os tipos de defeitos da *SMartyCheck*.

Equivalência entre os Tipos de Defeitos

Durante o projeto da taxonomia da *SMartyCheck*, bem como a adaptação dos itens do *checklist* da técnica, identificou-se que vários tipos de defeitos possuíam semelhanças e conflitos quando a inspeção dos diagramas *SMarty* era aplicada por meio dos itens do *checklist* da *SMartyCheck*. Diante disto, diversos tipos de defeitos foram mantidos com adaptações e/ou excluídos, com o intuito de aumentar a acurácia do *checklist*.

Assim, cada tipo de defeito é justificado e relacionado diretamente com sua respectiva equivalência sob outros tipos de defeitos, ora identificados quando o *checklist* da *SMartyCheck* foi elaborado:

- Tipos de defeitos equivalentes da norma IEEE 1012-2012 - *System and Software Verification and Validation* (IEEE, 2012):
  - **Ambiguidade**: mantido e adotado na taxonomia da *SMartyCheck* e possui os seguintes tipos de defeitos identificados como equivalentes: (i) Incompreensibilidade (Lamsweerde, 2009) *apud* (Souza *et al.*, 2013); (ii) Não Verificável (Miller *et al.*, 1995) *apud* (Hayes *et al.*, 2006); e (iii) Não Organizado (Lamsweerde, 2009) *apud* (Souza *et al.*, 2013). Os conflitos gerados por esses tipos de defeitos não permitiram que os mesmos fossem inseridos na taxonomia da *SMartyCheck*.
  - **Completo**: mantido e adotado na taxonomia da *SMartyCheck*, o qual foi definido e denominado como **Incompleto**. Não foram identificadas equivalências com outros tipos de defeitos.
  - **Consistência**: mantido e adotado na taxonomia da *SMartyCheck*, o qual foi definido e denominado como **Inconsistência** e possui o seguinte tipo de defeito identificado como equivalente: (i) Consequência (Anda e Sjøberg, 2002). As contradições geradas por esse tipo de defeito não permitiram que o mesmo fosse inserido na taxonomia da *SMartyCheck*, pois o tipo de defeito causava confusão e mal entendimento referente a leitura sobre o item e o que deveria ser inspecionado.
  - **Correto**: mantido e adotado na taxonomia da *SMartyCheck*, o qual foi definido e denominado como **Incorreto**. Não foram identificadas equivalências com outros tipos de defeitos.

- Tipos de defeitos equivalentes da norma IEEE 830-1998 - *Recommended Practice for Software Requirements Specifications* (IEEE, 1998a,c):

  - **Estável**: mantido e adotado na taxonomia da *SMartyCheck*, o qual foi definido e denominado como **Instável**. Não foram identificadas equivalências com outros tipos de defeitos.

  - **Modificável**: mantido e adotado na taxonomia da *SMartyCheck*, o qual foi definido e denominado como **Imodificável**. Não foram identificadas equivalências com outros tipos de defeitos.

- Tipos de defeitos equivalentes do estudo de Travassos *et al.* (1999):

  - **Fato Incorreto**: mantido e adotado na taxonomia da *SMartyCheck* e possui os seguintes tipos de defeitos identificados como equivalentes: (i) Incompreensibilidade (Lamsweerde, 2009) *apud* (Souza *et al.*, 2013); (ii) Intestável (Denger e Paech, 2004); (iii) Muito Especificado (Lamsweerde, 2009) *apud* (Souza *et al.*, 2013); (iv) Não Verificável (Miller *et al.*, 1995) *apud* (Hayes *et al.*, 2006); e (v) Testável (IEEE, 2012). As descrições não faziam sentido ou não eram objetivas o bastante sobre o que realmente deveria ser inspecionado.

  - **Informação Estranha**: mantido e adotado na taxonomia da *SMartyCheck* e possui os seguintes tipos de defeitos identificados como equivalentes: (i) Redundância (Cunha *et al.*, 2012); e (ii) Extraviado (Miller *et al.*, 1995) *apud* (Hayes *et al.*, 2006). As descrições estavam repetidas e não eram objetivas sobre o que realmente deveria ser inspecionado.

- Tipos de defeitos equivalentes dos estudos de Miller *et al.* (1995) *apud* Hayes *et al.* (2006):

  - **Desvio Intencional**: mantido e adotado na taxonomia da *SMartyCheck*. Não foram identificadas equivalências com outros tipos de defeitos.

  - **Inviável**: mantido e adotado na taxonomia da *SMartyCheck*. Não foram identificadas equivalências com outros tipos de defeitos.

- Tipos de defeitos equivalentes dos estudos de Lamsweerde (2009) *apud* Souza *et al.* (2013):

  - **Omissão**: mantido e adotado na taxonomia da *SMartyCheck* e possui o seguinte tipo de defeito identificado como equivalente, denominado Rastreabi-

lidade (IEEE, 2012). A descrição deste tipo de defeito se apresentava repetida quando comparada ao tipo de defeito Omissão.

- **Regras de Negócio**: mantido e adotado na taxonomia da *SMartyCheck*. Não foram identificadas equivalências com outros tipos de defeitos.

- Tipo de defeito equivalente do estudo de Cunha *et al.* (2012):

  - **Anomalia**: mantido e adotado na taxonomia da *SMartyCheck*. Não foram identificadas equivalências com outros tipos de defeitos.

A Figura 3.15 ilustra a taxonomia com os vinte e três (23) tipos de defeitos equivalentes mantidos com adaptações e/ou excluídos no *checklist* da *SMartyCheck*.

Figura 3.15: Taxonomia com Tipos de Defeitos Equivalentes da Técnica *SMartyCheck*.

A equivalência foi discutida a fim de mostrar o processo de adaptação dos tipos de defeitos mapeados da literatura para a técnica *SMartyCheck*. Portanto, tal equivalência é fundamental para as próximas seções.

3.4 *SMartyCheck 2.0*: Diagramas UML de LPS Inspecionados

SMartyCheck foi elaborada para inspecionar diagramas UML *SMarty* de casos de uso e de classes sob a dimensão de LPSs. Portanto, a *SMartyCheck* pode ser utilizada em dois momentos: (i) para inspecionar artefatos de uma LPS por meio de diagramas UML *SMarty* de casos de uso ou de classes com possíveis defeitos incorporados; ou (ii) para inspecionar as possíveis versões de uma LPS que pode ser modelada e comparada com base em uma LPS original (oráculo), da qual foi derivada.

Para tanto, foram definidos *releases*[1] (IEEE, 1998b) de quantos e quais diagramas *SMarty* a *SMartyCheck* suporta. Assim, para cada tipo de diagrama *SMarty* uma versão é somada à versão precedente. Neste sentido, foi definida a versão 1.0 para casos de uso e a versão 2.0 para classes, estabelecendo a versão atual da *SMartyCheck 2.0*.

Os estereótipos contidos nas diretrizes da abordagem *SMarty* (Seção 2.2.1) para casos de uso (UC.1, UC.2, UC.3 e UC.4) e classes (CL.1, CL.2, CL.2.1, CL.3, CL.4) possibilitam a comunicação entre *SMartyCheck* e *SMarty*. Tais estereótipos são utilizados no *checklist* da técnica para a inspeção de diagramas *SMarty* de variabilidade.

O *checklist* da *SMartyCheck* foi elaborado considerando a taxonomia de tipos de defeitos definida (Seção 3.3). Deste modo, o *checklist* da *SMartyCheck* possui a seguinte estrutura:

- **Id**: identificador numérico único referente a cada tipo de defeito;
- **Tipos de Defeitos**: contém o nome e a sigla referente a cada tipo de defeito;
- **Itens do *Checklist***: contém as descrições pré-definidas relacionadas a cada tipo de defeito. Tais descrições fornecem instruções por meio de questões/itens sobre "o quê" deve ser inspecionado nos diagramas *SMarty*;
- **Resposta (Sim/Não)**: dois campos permitem que o inspetor assinale com um "X" a resposta associada ao defeito inspecionado e possivelmente detectado. Quando a resposta for assinalada com um "X" no campo "Sim", a mesma deve ser justificada no campo "Defeito Identificado", do contrário, não é necessária justificá-la;
- **Defeito Identificado**: contém as observações do inspetor caso o defeito seja detectado e, consequentemente, o campo de resposta "Sim" seja assinalado com um "X".

[1] Notificações e distribuições formais de cada versão aprovada (IEEE, 1998b).

Um *checklist* foi elaborado para cada diagrama *SMarty* inspecionado, os quais possuem a mesma taxonomia de tipos de defeitos da *SMartyCheck*. Porém, a diferença principal está presente nas descrições dos itens pré-definidos relacionados para cada tipo de defeito atribuído para cada diagrama *SMarty*.

Outra característica importante nas descrições dos itens dos *checklists*, de forma comum para cada diagrama, são os estereótipos (Seção 2.2.1). Portanto, tais estereótipos oriundos da abordagem *SMarty* foram aplicados a partir das diretrizes *SMarty* existentes para a adaptação dos tipos de defeitos dos *checklists*. As diretrizes *SMarty* são essenciais, pois guiaram o elaboração dos itens do *checklist* da técnica *SMartyCheck*, apoiadas de acordo com os estereótipos que auxiliam e melhoram de fato a qualidade de inspeções de diagramas UML quando bem especificados (Fernández-Sáez *et al.*, 2013; Staron *et al.*, 2005).

Em suma, para a realização da inspeção dos diagramas *SMarty* de casos de uso e de classes de uma LPS o inspetor deve:

- responder cada questionamento/item do *checklist*, assinalando com um "X" apenas um campo ("Sim" ou "Não"); e
- quando o campo "Sim" for assinalado com um "X", deve-se justificar a resposta no campo "Defeito Identificado".

As próximas seções apresentam o formato do *checklist* para cada diagrama *SMarty* de casos de uso e de classes da técnica *SMartyCheck 2.0*.

3.4.1 Inspeção de Diagramas *SMarty* de Casos de Uso

O diagrama de casos de uso com possíveis defeitos de uma LPS, sem comparação com o oráculo, contém casos de uso que supostamente podem:

- conter um nome igual a outros casos de uso, mas com resposabilidades diferentes (homônimo) ou um nome que não reflete seu objetivo real;
- ser obrigatórios e não estarem especificados; e
- ter sua funcionalidade duplicada no diagrama.

O diagrama de casos de uso com possíveis defeitos de uma LPS, quando comparado com o oráculo, contém casos de uso que supostamente podem:

- apresentar inconsistências entre a comparação dos diagramas inspecionados;

- estar associados a partir de outros casos de uso de forma incorreta;

- requerer a escolha obrigatória de associação a partir de outros casos de uso;

- estar fora do escopo dos diagramas comparados;

- não estar especificados nos diagramas;

- ser verificados se possuem uma funcionalidade comum entre os diagramas;

- ser combinados ou escolhidos entre os diagramas conforme o cumprimento das regras impostas pelos estereótipos da *SMarty* (Seção 2.2.1); e

- ser escolhidos de acordo com suas funcionalidades com base em um domínio específico, ora definido para os diagramas de uma LPS qualquer.

Assim, os itens a seguir apresentam os tipos de defeitos e os itens do *checklist* da *SMartyCheck* divididos em duas categorias para a inspeção de diagramas *SMarty* de casos de uso de LPS:

– **Sem o Oráculo (inspecionar apenas o diagrama com defeitos incorporados)**

01. Ambiguidade (Am)

- **Am.1** Existe algum caso de uso em que seu nome não reflete o seu objetivo real?
- **Am.2** Existe algum caso de uso em que seu nome é igual a outro caso de uso de modo a possuir uma interpretação duplicada?

02. Anomalia (An)

- **An.1** Existe algum caso de uso especificado por meio do estereótipo <<`alternative_OR`>> associado com um caso de uso que não esteja relacionado com o estereótipo <<`variationPoint`>>?

03. Incompleto (Incom)

- **Incom.1** Todos os casos de uso com <<`mandatory`>>, ou seja, obrigatórios não estão especificados no diagrama de casos de uso da LPS?

05. Incorreto (Incor)

- **Incor.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>> que está associado com um caso de uso na LPS que não seja <<`alternative_OR`>>?

- **Incor.2** Existe algum caso de uso com o estereótipo <<`mandatory`>> que está associado com um caso de uso na LPS marcado com o estereótipo <<`mandatory`>>?

07. Instável (Ins)

- **Ins.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>>, no qual possui associado o estereótipo <<`variability`>> cujo meta-atributo *name* seja igual na LPS?

09. Informação Estranha (IE)

- **IE.2** Existe algum caso de uso com sua funcionalidade duplicada na LPS?

10. Inviável (Inv)

- **Inv.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas na LPS não permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar*?

– **Com Oráculo (inspecionar comparando os diagramas sem defeitos e com defeitos incorporados)**

04. Inconsistência (Incons)

- **Incons.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior que o definido em *maxSelection* da variabilidade associada?
- **Incons.2** Existe algum caso de uso com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é menor que o definido em *minSelection* da variabilidade associada?
- **Incons.3** Existe alguma variabilidade (<<`variability`>>) especificada no diagrama de casos de uso da LPS cujo meta-atributo *bindingTime* seja DESIGN_TIME (tempo de projeto)?

06. Desvio Intencional (DI)

- **DI.1** Existe algum caso de uso que exige a seleção de outro (<<`requires`>>) e esse outro não está especificado na LPS?
- **DI.2** Existe algum caso de uso que exige a não seleção de outro (<<`mutex`>>) e esse outro está especificado na LPS?

08. Fato Incorreto (FI)

- **FI.1** Existe algum caso de uso com seu nome incorreto na LPS?

- **FI.2** Existe um ou mais casos de uso que não é possível comparar seu nome na LPS?

09. Informação Estranha (IE)

- **IE.1** Existe algum caso de uso especificado além dos casos de uso já existentes na LPS?
- **IE.2** Existe algum caso de uso com sua funcionalidade duplicada na LPS?

11. Imodificável (Imo)

- **Imo.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>>, no qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas, afetando a coerência da LPS, de acordo a com o meta-atributo *variants*?

12. Omissão (Om)

- **Om.1** Existe algum caso de uso especificado como obrigatório por meio do estereótipo <<`mandatory`>> que não esteja especificado na LPS?

13. Regras de Negócio (RN)

- Este tipo de defeito deve ser aplicado baseado na definição de um domínio específico incluso em uma LPS específica modelada a partir da LPS oráculo. O Domínio/Regras de Negócio Pré-definidos devem ser descritos neste espaço. **RN.1** Os casos de uso não são claros com o objetivo e as funcionalidades desejadas com base no domínio definido?

No intuito de propiciar uma melhor compreensão da *SMartyCheck* em um caso prático e concreto, o diagrama de casos de uso da LPS *Arcade Game Maker* (AGM) (SEI, 2009) (Apêndice C) de acordo com *SMarty* (Figura 3.16) é utilizado. Para que isto seja possível, um diagrama de casos de uso da LPS AGM foi criado com alguns defeitos incorporados (*Toy example* - Figura 3.17). O exemplo de inspeção desses diagramas é apresentado por meio do *checklist* da *SMartyCheck* de acordo com a Tabela 3.2.

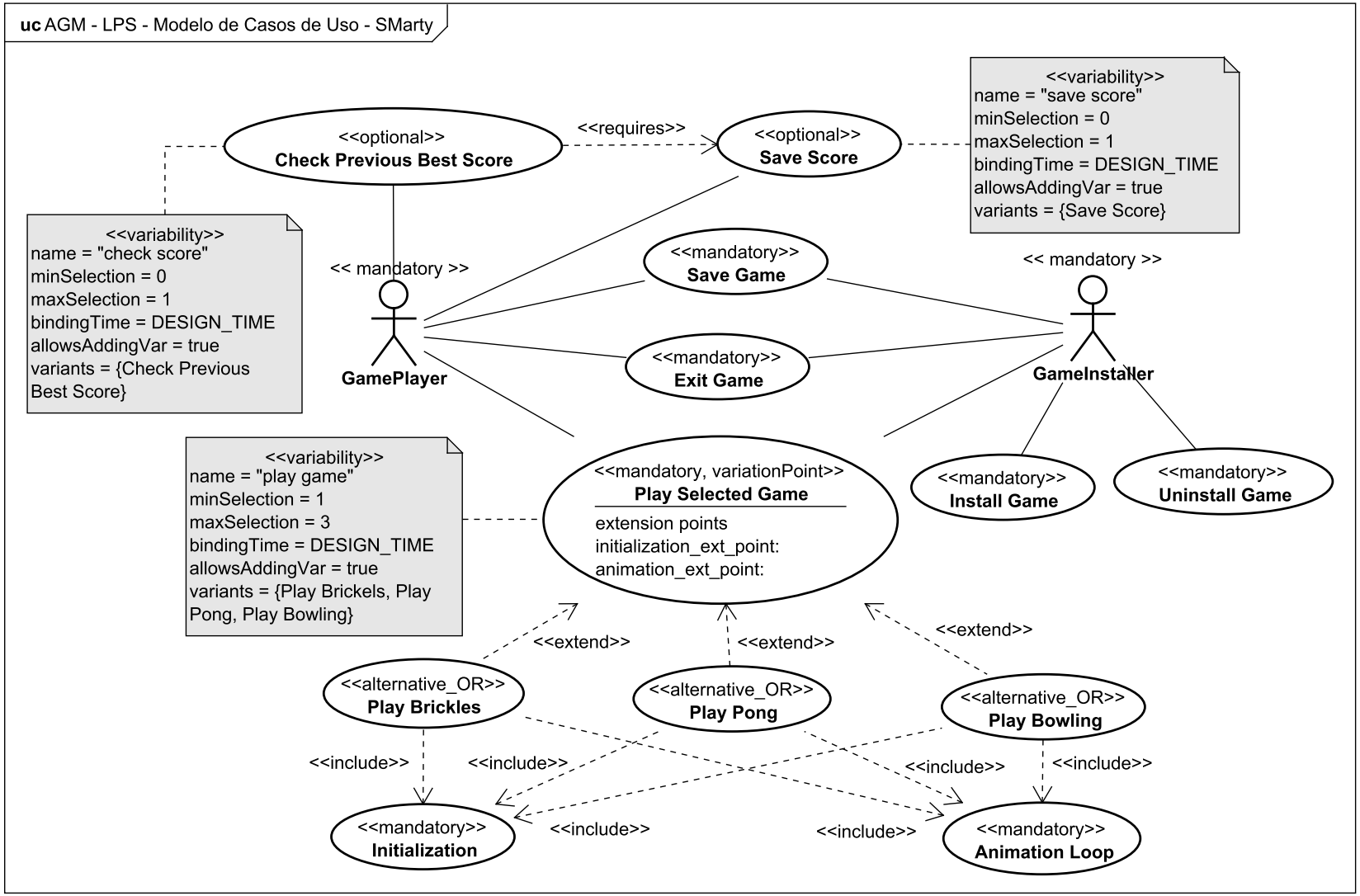


Figura 3.16: Diagrama de Casos de Uso *SMarty* Oráculo da LPS AGM (Apêndice C).

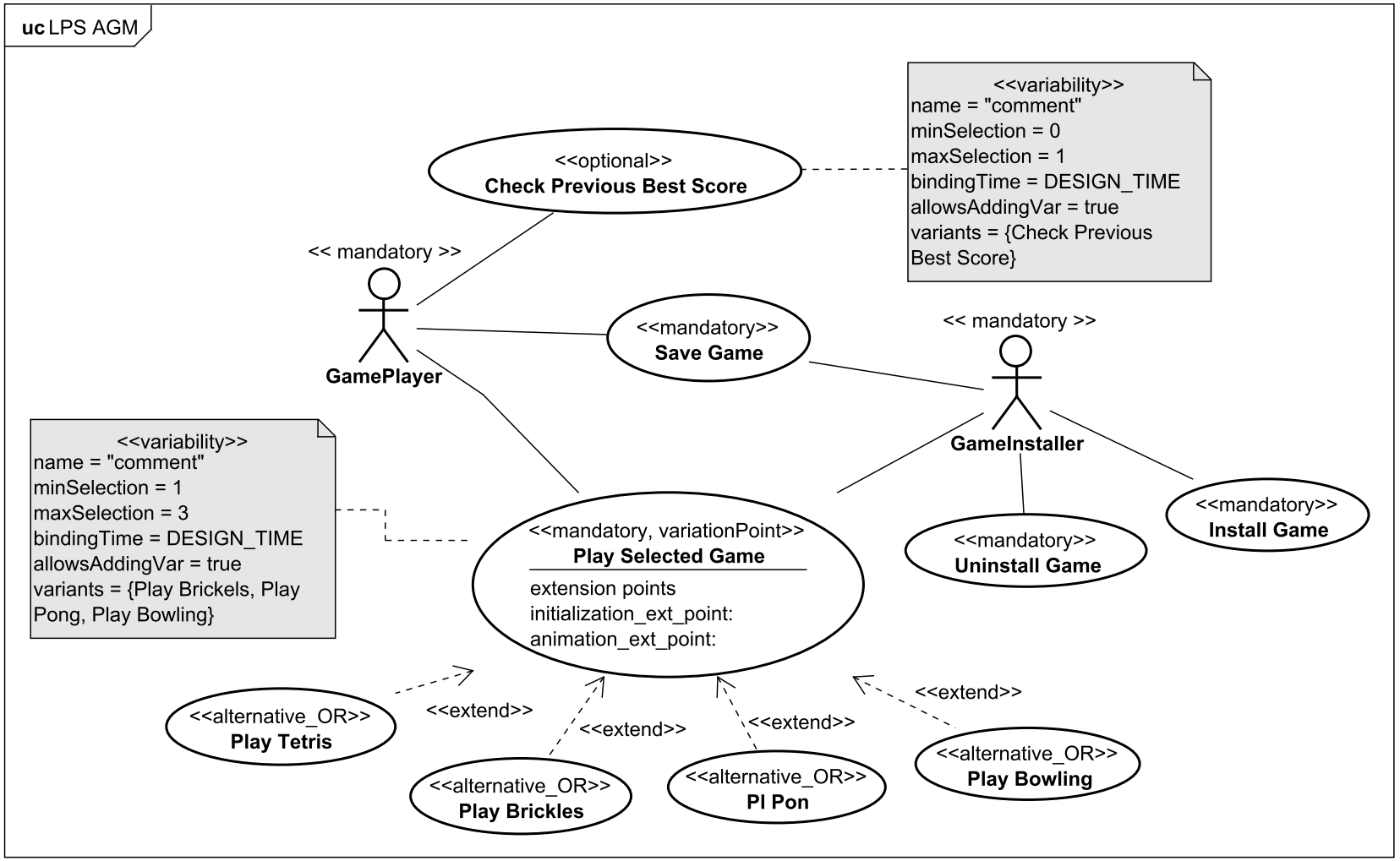


Figura 3.17: *Toy Example* - Diagrama de Casos de Uso *SMarty* da LPS AGM com Defeitos Incorporados.

Tabela 3.2: Exemplo Inspeção Casos de Uso da LPS AGM *SMarty*.

Id	Tipos de Defeitos	Itens do *Checklist*	Sim	Não	Defeito Identificado
		Sem o Oráculo - Inspeção do diagrama com defeitos incorporados			
01	**Ambiguidade (Am)**	**Am.1** Existe algum caso de uso em que seu nome não reflete o seu objetivo real?		X	
		Am.2 Existe algum caso de uso em que seu nome é igual a outro caso de uso de modo a possuir uma interpretação duplicada?		X	
02	**Anomalia (An)**	**An.1** Existe algum caso de uso especificado por meio do estereótipo <<`alternative_OR`>> associado com um caso de uso que não esteja relacionado com o estereótipo <<`variationPoint`>>?		X	
03	**Incompleto (Incom)**	**Incom.1** Todos os casos de uso com <<`mandatory`>>, ou seja, obrigatórios não estão especificados no diagrama de casos de uso da LPS?		X	
05	**Incorreto (Incor)**	**Incor.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>> que está associado com um caso de uso na LPS que não seja <<`alternative_OR`>>?		X	
		Incor.2 Existe algum caso de uso com o estereótipo <<`mandatory`>> que está associado com um caso de uso na LPS marcado com o estereótipo <<`mandatory`>>?		X	
07	**Instável (Ins)**	**Ins.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>>, no qual possui associado o estereótipo <<`variability`>> cujo meta-atributo *name* seja igual na LPS?	X		Os casos de uso *Play Selected Game* e *Check Previous Best Score* possuem o mesmo meta-atributo *name* = "comment".
09	**Informação Estranha (IE)**	**IE.2** Existe algum caso de uso com sua funcionalidade duplicada na LPS?		X	
10	**Inviável (Inv)**	**Inv.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas na LPS não permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar*?		X	
		Com Oráculo - Inspeção comparando os diagramas sem defeitos e com defeitos incorporados			

Continua na próxima página

Tabela 3.2 – *Continuação da página anterior*

Id	Tipos de Defeitos	Itens do *Checklist*	Sim	Não	Defeito Identificado
04	**Inconsistência (Incons)**	**Incons.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior que o definido em *maxSelection* da variabilidade associada?	X		O caso de uso *Play Selected Game* possui quatro variantes, sendo que deveria possuir no máximo três.
		Incons.2 Existe algum caso de uso com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é menor que o definido em *minSelection* da variabilidade associada?		X	
		Incons.3 Existe alguma variabilidade (<<`variability`>>) especificada no diagrama de casos de uso da LPS cujo meta-atributo *bindingTime* seja DESIGN_TIME (tempo de projeto)?		X	
06	**Desvio Intencional (DI)**	**DI.1** Existe algum caso de uso que exige a seleção de outro (<<`requires`>>) e esse outro não está especificado na LPS?	X		O caso de uso *Check Previous Best Score* exige a seleção do caso de uso *Save Score*.
		DI.2 Existe algum caso de uso que exige a não seleção de outro (<<`mutex`>>) e esse outro está especificado na LPS?		X	
08	**Fato Incorreto (FI)**	**FI.1** Existe algum caso de uso com seu nome incorreto na LPS?	X		O caso de uso *Play Pong* possui o nome *Pl Pon*.
		FI.2 Existe um ou mais casos de uso que não é possível comparar seu nome na LPS?	X		Os casos de uso *Play Pong* e *Pl Pon* não podem ser comparados.
09	**Informação Estranha (IE)**	**IE.1** Existe algum caso de uso especificado além dos casos de uso já existentes na LPS?	X		O caso de uso *Pl Pon*.
		IE.2 Existe algum caso de uso com sua funcionalidade duplicada na LPS?		X	
11	**Imodificável (Imo)**	**Imo.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>>, no qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas, afetando a coerência da LPS, de acordo a com o meta-atributo *variants*?	X		O caso de uso *Play Selected Game* possui variantes escolhidas e combinadas: *Play Tetris*, *Play Brickles*, *Pl Pong* e *Play Bowling*. As variantes *Play Tetris* e *Pl Pong* foram especificadas de forma errônea. Já as variantes - *variants*: [*Play Brickles*, *Play Pong* e *Play Bowling*], não afetam a coerência da LPS.

Continua na próxima página

Tabela 3.2 – *Continuação da página anterior*

Id	Tipos de Defeitos	Itens do *Checklist*	Sim	Não	Defeito Identificado
12	**Omissão (Om)**	**Om.1** Existe algum caso de uso especificado como obrigatório por meio do estereótipo <<`mandatory`>> que não esteja especificado na LPS?	X		Os casos de uso obrigatórios *Exit Game*, *Initialization* e *Animation Loop* não estão especificados.
13	**Regras de Negócio (RN)**	Domínio/Regras de Negócio Pré-definidos: O sistema deve ser capaz de permitir a instalar e desinstalar um *game*, além de permitir escolher um *game* para jogar, salvar o estado do *game*, sair do *game*, definir as opções do *game* e disponibilizar ao jogador um modo de salvar seu *score* e checar seu melhor *score* anterior.			
		RN.1 Os casos de uso não são claros com o objetivo e as funcionalidades desejadas com base no domínio definido?	X		O sistema não é capaz permitir sair do *game*, definir as opções do *game* e disponibilizar ao jogador um modo de salvar seu *score*.

3.4.2 Inspeção de Diagramas *SMarty* de Classes

O diagrama de classes com possíveis defeitos de uma LPS, sem comparação com o oráculo, contém classes que supostamente podem:

- conter um nome igual a outras classes, mas com resposabilidades diferentes (homônimo) ou um nome que não reflete seu objetivo real;

- ser obrigatórias e não estarem especificadas; e

- ter sua funcionalidade duplicada no diagrama.

O diagrama de classes com possíveis defeitos de uma LPS, quando comparado com o oráculo, contém classes que supostamente podem:

- apresentar inconsistências entre a comparação dos diagramas inspecionados;

- estar associadas a partir de outras classes de forma incorreta;

- requerer a escolha obrigatória de associação a partir de outras classes;

- estar fora do escopo dos diagramas comparados;

- não estar especificadas nos diagramas;

- ser verificadas se possuem uma funcionalidade comum entre os diagramas;

- ser combinadas ou escolhidas entre os diagramas conforme o cumprimento das regras impostas pelos estereótipos da *SMarty* (Seção 2.2.1); e

- ser escolhidas de acordo com suas funcionalidades com base em um domínio específico, ora definido para os diagramas de uma LPS qualquer.

Assim, os itens a seguir apresentam os tipos de defeitos e os itens do *checklist* da *SMartyCheck* divididos em duas categorias para a inspeção de diagramas *SMarty* de classes de LPS:

– **Sem o Oráculo (inspecionar apenas o diagrama com defeitos incorporados)**

01. Ambiguidade (Am)

- **Am.1** Existe alguma classe em que seu nome não reflete o seu objetivo real?
- **Am.2** Existe alguma classe em que seu nome é igual a outra classe de modo a possuir uma interpretação duplicada?

02. Anomalia (An)

- **An.1** Existe alguma classe especificada por meio do estereótipo <<`alternative_OR`>> associada com uma classe que não esteja relacionada com o estereótipo <<`variationPoint`>>?

03. Incompleto (Incom)

- **Incom.1** Todas as classes com <<`mandatory`>>, ou seja, obrigatórias não estão especificadas no diagrama de classes da LPS?

05. Incorreto (Incor)

- **Incor.1** Existe alguma classe com o estereótipo <<`variationPoint`>> que está associada com uma classe na LPS que não seja <<`alternative_OR`>>?
- **Incor.2** Existe alguma classe com o estereótipo <<`mandatory`>> que está associada com uma classe na LPS marcada com o estereótipo <<`mandatory`>>?

07. Instável (Ins)

- **Ins.1** Existe alguma classe com o estereótipo <<`variationPoint`>>, na qual possui associada o estereótipo <<`variability`>> cujo meta-atributo *name* seja igual na LPS?

09. Informação Estranha (IE)

- **IE.2** Existe alguma classe com sua funcionalidade duplicada na LPS?

10. Inviável (Inv)

- **Inv.1** Existe alguma classe com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas na LPS não permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar*?

– **Com Oráculo (inspecionar comparando os diagramas sem defeitos e com defeitos incorporados)**

04. Inconsistência (Incons)

- **Incons.1** Existe alguma classe com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior que o definido em *maxSelection* da variabilidade associada?
- **Incons.2** Existe alguma classe com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é menor que o definido em *minSelection* da variabilidade associada?

- **Incons.3** Existe alguma variabilidade (<<`variability`>>) especificada no diagrama de classes da LPS cujo meta-atributo *bindingTime* seja DESIGN_TIME (tempo de projeto)?

06. Desvio Intencional (DI)

- **DI.1** Existe alguma classe que exige a seleção de outra (<<`requires`>>) e essa outra não está especificada na LPS?
- **DI.2** Existe alguma classe que exige a não seleção de outra (<<`mutex`>>) e essa outra está especificada na LPS?

08. Fato Incorreto (FI)

- **FI.1** Existe alguma classe com seu nome incorreto na LPS?
- **FI.2** Existe uma ou mais classes que não é possível comparar seu nome na LPS?

09. Informação Estranha (IE)

- **IE.1** Existe alguma classe especificada além das classes já existentes na LPS?
- **IE.2** Existe alguma classe com sua funcionalidade duplicada na LPS?

11. Imodificável (Imo)

- **Imo.1** Existe alguma classe com o estereótipo <<`variationPoint`>>, na qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas, afetando a coerência da LPS, de acordo a com o meta-atributo *variants*?

12. Omissão (Om)

- **Om.1** Existe alguma classe especificada como obrigatória por meio do estereótipo <<`mandatory`>> que não esteja especificado na LPS?

13. Regras de Negócio (RN)

- Este tipo de defeito deve ser aplicado baseado na definição de um domínio específico incluso em uma LPS específica modelada a partir da LPS oráculo. O Domínio/Regras de Negócio Pré-definidos devem ser descritos neste espaço. **RN.1** As classes não são claras com o objetivo e as funcionalidades desejadas com base no domínio definido?

No intuito de propiciar uma melhor compreensão da *SMartyCheck* em um caso prático e concreto, o diagrama de classes da LPS *Arcade Game Maker* (AGM) (SEI, 2009) (Apêndice C) de acordo com *SMarty* (Figura 3.18) é utilizado. Para que isto seja possível,

um diagrama de classes da LPS AGM foi criado com alguns defeitos incorporados (*Toy example* - Figura 3.19). O exemplo de inspeção desses diagramas é apresentado por meio do *checklist* da *SMartyCheck* de acordo com a Tabela 3.3.

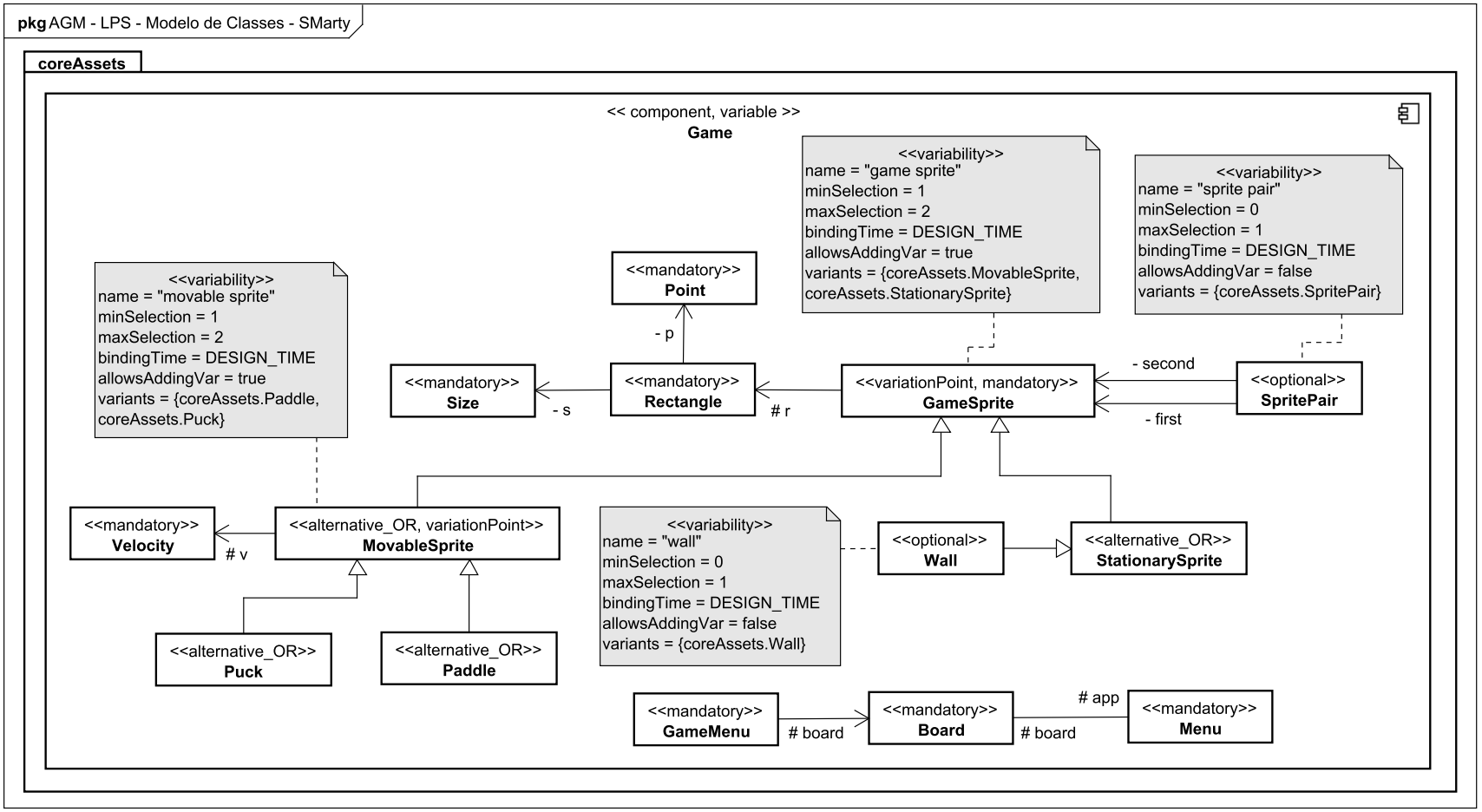


Figura 3.18: Diagrama de Classes *SMarty* Oráculo da LPS AGM (Apêndice C).

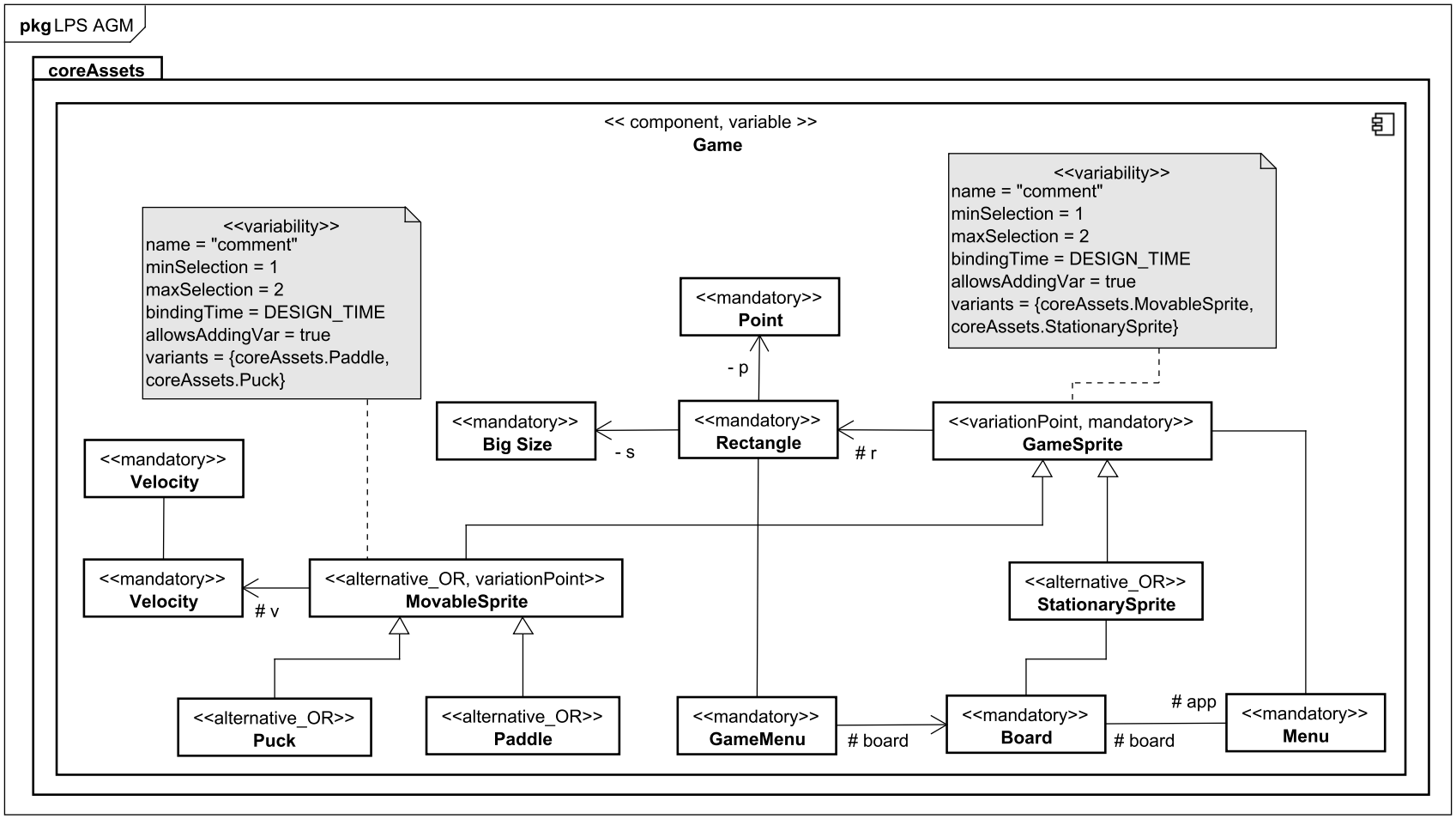


Figura 3.19: *Toy Example* - Diagrama de Classes *SMarty* da LPS AGM com Defeitos Incorporados.

Tabela 3.3: Exemplo Inspeção Classes da LPS AGM *SMarty*.

Id	Tipos de Defeitos	Itens do *Checklist*	Sim	Não	Defeito Identificado
		Sem o Oráculo - Inspeção do diagrama com defeitos incorporados			
01	**Ambiguidade (Am)**	**Am.1** Existe alguma classe em que seu nome não reflete o seu objetivo real?	X		A classe *Size* não está de acordo, a qual foi alterada para *Big Size*.
		Am.2 Existe alguma classe em que seu nome é igual a outra classe de modo a possuir uma interpretação duplicada?	X		A classe *Velocity* possui a mesmo nome e está associada com ela mesma.
02	**Anomalia (An)**	**An.1** Existe alguma classe especificada por meio do estereótipo <<`alternative_OR`>> associada com uma classe que não esteja relacionada com o estereótipo <<`variationPoint`>>?	X		A classe *StationarySprite* está associada com a classe *Board*, a qual não é um ponto de variação.
03	**Incompleto (Incom)**	**Incom.1** Todas as classes com <<`mandatory`>>, ou seja, obrigatórias não estão especificadas no diagrama de classes da LPS?		X	
05	**Incorreto (Incor)**	**Incor.1** Existe alguma classe com o estereótipo <<`variationPoint`>> que está associada com uma classe na LPS que não seja <<`alternative_OR`>>?	X		A classe *GameSprite* está associada com a classe *Menu*.
		Incor.2 Existe alguma classe com o estereótipo <<`mandatory`>> que está associada com uma classe na LPS marcada com o estereótipo <<`mandatory`>>?		X	(i) A classe *Rectangle* está associada com a classe *GameMenu*; (ii) a classe *StationarySprite* está associada com a classe *Board*; e (iii) a classe *GameSprite* está associada com a classe *Menu*.
07	**Instável (Ins)**	**Ins.1** Existe alguma classe com o estereótipo <<`variationPoint`>>, na qual possui associada o estereótipo <<`variability`>> cujo meta-atributo *name* seja igual na LPS?	X		As classes *MovableSprite* e *GameSprite* possuem o mesmo meta-atributo *name* = "comment".
09	**Informação Estranha (IE)**	**IE.2** Existe alguma classe com sua funcionalidade duplicada na LPS?	X		A classe *Velocity* está duplicada na LPS.
10	**Inviável (Inv)**	**Inv.1** Existe alguma classe com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas na LPS não permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar*?		X	
		Com Oráculo - Inspeção comparando os diagramas sem defeitos e com defeitos incorporados			

Continua na próxima página

Tabela 3.3 – *Continuação da página anterior*

Id	Tipos de Defeitos	Itens do *Checklist*	Sim	Não	Defeito Identificado
04	**Inconsistência (Incons)**	**Incons.1** Existe alguma classe com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior que o definido em *maxSelection* da variabilidade associada?		X	
		Incons.2 Existe alguma classe com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é menor que o definido em *minSelection* da variabilidade associada?		X	
		Incons.3 Existe alguma variabilidade (<<`variability`>>) especificada no diagrama de classes da LPS cujo meta-atributo *bindingTime* seja DESIGN_TIME (tempo de projeto)?		X	
06	**Desvio Intencional (DI)**	**DI.1** Existe alguma classe que exige a seleção de outra (<<`requires`>>) e essa outra não está especificada na LPS?		X	
		DI.2 Existe alguma classe que exige a não seleção de outra (<<`mutex`>>) e essa outra está especificada na LPS?		X	
08	**Fato Incorreto (FI)**	**FI.1** Existe alguma classe com seu nome incorreto na LPS?	X		A classe *Size* possui o nome *Big Size*.
		FI.2 Existe uma ou mais classes que não é possível comparar seu nome na LPS?	X		As classes *Size* e *Big Size* não podem ser comparadas.
09	**Informação Estranha (IE)**	**IE.1** Existe alguma classe especificada além das classes já existentes na LPS?	X		A classe *Big Size*.
		IE.2 Existe alguma classe com sua funcionalidade duplicada na LPS?	X		A classe *Velocity* está duplicada.
11	**Imodificável (Imo)**	**Imo.1** Existe alguma classe com o estereótipo <<`variationPoint`>>, na qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas, afetando a coerência da LPS, de acordo a com o meta-atributo *variants*?		X	
12	**Omissão (Om)**	**Om.1** Existe alguma classe especificada como obrigatória por meio do estereótipo <<`mandatory`>> que não esteja especificada na LPS?	X		A classe obrigatória *Size* não está especificada.

Continua na próxima página

Tabela 3.3 – *Continuação da página anterior*

Id	Tipos de Defeitos	Itens do *Checklist*	Sim	Não	Defeito Identificado
13	**Regras de Negócio (RN)**	Domínio/Regras de Negócio Pré-definidos: O sistema deve permitir que elementos se movimentem no jogo e contenha elementos dos jogos com os quais o jogador interage. Além disso, deve definir no projeto do jogo um retângulo que demarca sua área, bem como o tamanho deste retângulo. Por fim, deve permitir personalizar cores para novos ambientes dos jogos.			
		RN.1 As classes não são claras com o objetivo e as funcionalidades desejadas com base no domínio definido?	X		O sistema não é capaz de permitir a personalização de cores para novos ambientes dos jogos.

3.5 Considerações Finais

Este capítulo apresentou a proposta da técnica de inspeção de software denominada *SMartyCheck*, com o objetivo principal de inspecionar diagramas *SMarty* de casos de uso e classes de LPS. Além disso, foi apresentada a motivação acerca da sua proposta e foram descritos os tipos de defeitos adotados da literatura que constituem a taxonomia da *SMartyCheck*, além de sua versão 2.0, na qual permite inspecionar os diagramas *SMarty* de casos de uso e classes de LPS.

Foram criados dois exemplos ilustrativos que apresentam a aplicação da *SMartyCheck* em diagramas *SMarty* da LPS AGM. Tais exemplos são importantes para entender como os estudos empíricos dos Capítulos 4 e 5 foram realizados.

Os próximos capítulos abrangem um estudo qualitativo (Capítulo 4) e um estudo quantitativo (Capítulo 5), a fim de avaliar a real viabilidade e a eficiência, a eficácia e efetividade da técnica *SMartyCheck*, de acordo com a metodologia de experimentação empregada por meio de *mixed-methods* (métodos mistos) (Creswell e Clark, 2010) e engenharia de software experimental (Juristo e Moreno, 2010; Wohlin *et al.*, 2012).

É importante ressaltar que a proposta inicial do *checklist* da *SMartyCheck* apresentada neste capítulo foi avaliada por meio de tais estudos empíricos. Assim, somente após a análise e interpretação dos resultados do estudo quantitativo tal *checklist* da *SMartyCheck* foi aprimorado, sendo considerado o *checklist* final da técnica (Seção 5.7). Contudo, esse *checklist* aprimorado não foi avaliado, possibilitando a condução de novos estudos empíricos no futuro.

4

Estudo Empírico Qualitativo da *SMartyCheck 2.0*

"A dúvida é o principio da sabedoria."

Aristóteles (384 - 322 a.C.), Filósofo Grego

4.1 Considerações Iniciais

Este Capítulo apresenta um estudo empírico qualitativo com a finalidade de avaliar a viabilidade da técnica *SMartyCheck 2.0*. A realização deste estudo foi essencial, pois foi possível analisar o *feedback*/respostas de especialistas para refinar a técnica *SMartyCheck*.

A fim de ilustrar a metodologia de pesquisa aplicada neste estudo, a Figura 4.1 apresenta as respectivas etapas seguidas.

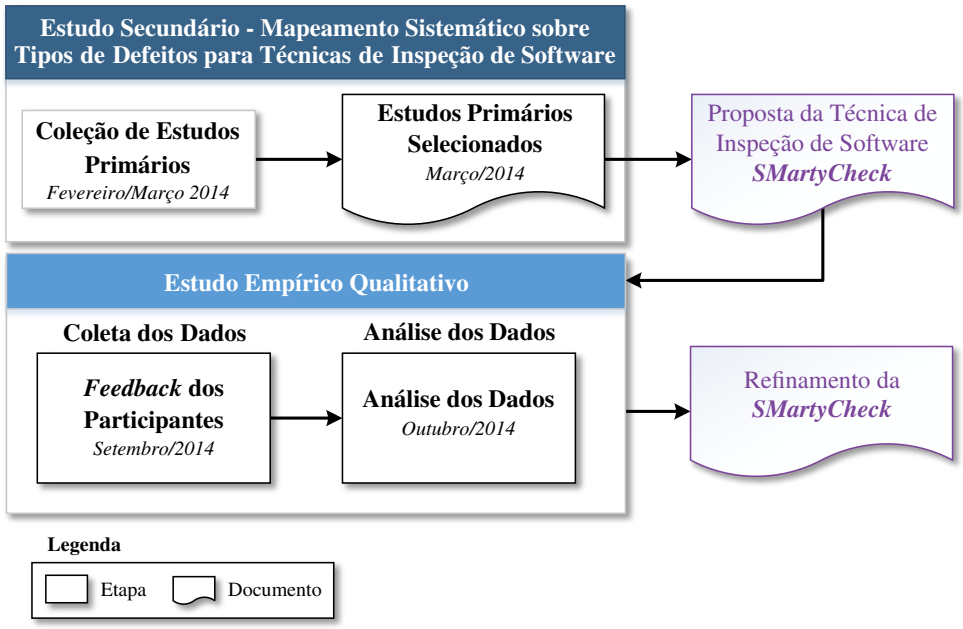


Figura 4.1: Metodologia de Pesquisa do Estudo Empírico Qualitativo.

Este estudo se baseia em procedimentos de *Grounded Theory* e *Coding* de acordo com Corbin e Strauss (2008), além da estratégia sequencial exploratória de métodos mistos (*mixed-methods*) conforme Creswell e Clark (2010); Dybå *et al.* (2011); Seaman (1999).

Um estudo qualitativo é essencialmente qualquer tipo de pesquisa que retorne resultados que não podem ser medidos por meio de métodos estatísticos em qualquer tipo de escala numérica (Corbin e Strauss, 2008).

Segundo Corbin e Strauss (2008), a importância da *Grounded Theory* é garantida por meio do pesquisador que deve compreender e caminhar em conjunto com sua pesquisa. As técnicas e os métodos qualitativos fornecem os meios para transformar a visão do pesquisador em realidade. Tanto a teoria como a análise dos dados envolvem interpretação baseada em uma investigação feita de modo sistemático. Portanto, a análise ocorre pela interação entre os especialistas e os dados (Corbin e Strauss, 2008).

Basicamente, há três componentes principais na pesquisa qualitativa, apoiada por *Grounded Theory* (Corbin e Strauss, 2008):

- dados, advindos de várias fontes, tais como questionários (aplicados neste estudo);
- procedimentos, os quais os pesquisadores podem utilizar para organizar e interpretar os dados em termos de suas propriedades e dimensões; e
- conceitualizar, para reduzir, elaborar e relacionar sempre são referidos como ***Coding***.

Para analisar os dados deste estudo foram utilizados *Open Coding* e *Axial Coding* (Corbin e Strauss, 2008):

- ***Open Coding*** é o processo analítico por meio do qual os conceitos são identificados e suas propriedades e dimensões são descobertas a partir dos dados; e

- ***Axial Coding*** é o processo de relacionar categorias às suas subcategorias. Chamado de "axial" porque ocorre em torno do eixo de uma categoria, associando categorias ao nível de propriedades e dimensões.

Neste estudo, representações de ocorrência dos códigos e outras relações são apresentadas de acordo com as análises/interpretações realizadas com base nas respostas/*feedback* de cada participante.

4.2 Definição do Estudo

O principal propósito deste estudo foi responder a questão: "**A técnica *SMartyCheck* é viável para a detecção de defeitos em diagramas *SMarty* de casos de uso e de classes?**".

Deste modo, o objetivo deste estudo (Basili *et al.*, 1994) é:

Analisar *a técnica SMartyCheck*

Com o propósito de *refiná-la*

Em relação à *viabilidade de detecção de defeitos em diagramas SMarty de casos de uso e de classes*

Do ponto de vista *de Pesquisadores da Técnica*

No contexto de acadêmicos de pós-graduação e docentes das seguintes instituições: Instituto Federal do Paraná (IFPR), Universidade Paranaense (UNIPAR), Universidade de São Paulo (ICMC/USP) e Universidade Tecnológica Federal do Paraná (UTFPR).

4.3 Planejamento do Estudo

As etapas correspondentes ao planejamento deste estudo empírico qualitativo são apresentadas a seguir.

Projeto Piloto: um estudo piloto foi realizado com o objetivo de avaliar a instrumentação utilizada, para adequá-la. Para tanto, dois especialistas, ambos com doutorado, um com conhecimento da área de engenharia de software, notação UML e LPS e, o outro,

com conhecimento na área de engenharia de software e inspeção de software. Os dados obtidos no projeto piloto não foram utilizados como resultados deste estudo empírico.

Treinamento: para este estudo foi necessário treinar os especialistas na aplicação da técnica *SMartyCheck*. Foram apresentados aos especialistas exemplos de inspeção de diagramas *SMarty* de casos de uso e de classes de LPS por meio da explicação dos tipos de defeitos contidos no *checklist* da *SMartyCheck* utilizando a LPS *Mobile Media*.

Especialistas: a maioria dos especialistas são da área de engenharia de software, com dez anos de experiência em média. Tais especialistas são: 4 doutorandos (50%), 2 mestrandos (25%), 1 doutor (12,5%) e 1 mestre (12,5%). Tais especialistas são docentes, acadêmicos de pós-graduação e profissionais, os quais trabalham diretamente com a área ou pesquisadores que têm conhecimento sobre verificação e validação de software ao longo dos últimos anos em diferentes universidades no Brasil.

Instrumentação: todos os especialistas do estudo receberam um conjunto de documentos: (i) uma cópia do Termo de Consentimento Livre e Esclarecido (TCLE); (ii) uma cópia do Questionário de Caracterização de Participante, no qual o especialista indicará sua formação acadêmica, seu nível de experiência com modelagem (notação UML), seu conhecimento sobre LPS e inspeção de software; (iii) uma cópia de um documento com uma síntese sobre os Conceitos de Linha de Produto de Software (LPS); (iv) uma cópia de um documento resumido sobre os Conceitos de Inspeção de Software e a Técnica *SMartyCheck*; (v) uma cópia de um documento sobre o treinamento aplicado referente a técnica *SMartyCheck* com exemplos do *checklist* da mesma, utilizando a LPS *Mobile Media* (Apêndice D); (vi) uma cópia de um documento com a descrição geral da LPS *Arcade Game Maker* (AGM), idêntico ao contido no Apêndice C; e (vii) uma cópia de um documento com a descrição geral dos estereótipos do perfil *SMartyProfile* (Seção 2.2.1) da abordagem *SMarty*, os quais são utilizados no *checklist* elaborado da técnica *SMartyCheck* (Capítulo 3).

A instrumentação apresentada poderia ser utilizada pelos especialistas para consulta, com a finalidade de auxiliá-los à responderem os formulários eletrônicos criados no software LimeSurvey[1] (Apêndice E). Para isto, foram criados seis formulários eletrônicos, os quais foram distribuídos e enviados por email aleatoriamente para os especialistas. Cada especialista recebeu por email, dentre os seis formulários, um formulário contendo diagramas de casos de uso e um formulário contendo diagramas de classes, que deveriam

[1]LimeSurvey - *http://www.limesurvey.org*

ser inspecionados. Além disso, um formulário eletrônico a respeito do estudo empírico qualitativo e sobre a técnica *SMartyCheck* foi enviado aos especialistas.

4.4 Execução do Estudo

Nesta Seção são apresentadas as etapas executadas para a obtenção dos resultados referentes à aplicação deste estudo empírico qualitativo.

Procedimentos de Participação: os itens a seguir, números em ordem cronológica, apresentam os procedimentos relacionados a participação para tal estudo:

1. o especialista recebe documentos necessários ao estudo para preencher e assinar;
2. o especialista faz a leitura do termo, esclarece possíveis dúvidas e assina o Termo de Consentimento Livre e Esclarecido (TCLE);
3. o experimentador/pesquisador anota o Id (gerado no software LimeSurvey) do especialista no Questionário de Caracterização de Participante;
4. o especialista faz a leitura do Questionário de Caracterização de Participante, esclarece possíveis dúvidas, preenche suas informações e o assina;
5. o experimentador/pesquisador ministra o treinamento;
6. após o treinamento, o experimentador/pesquisador envia a instrumentação para cada especialista, a qual pode ser utilizada para consulta ao responder os questionários eletrônicos;
7. o experimentador/pesquisador explica o formato dos formulários (questionários) eletrônicos enviados a cada especialista, com o objetivo de esclarecer possíveis dúvidas de preenchimento e quanto ao envio dos mesmos;
8. o especialista esclarece possíveis dúvidas com relação ao formato, preenchimento e envio dos formulários recebidos em seu email; e
9. o especialista responde e envia os formulários eletrônicos.

É importante mencionar que um prazo de 20 dias foi estipulado e combinado em conjunto com os especialistas para responderem aos questionários. Contudo, alguns especialistas solicitaram que o prazo fosse estendido por mais 7 dias por causa de

compromissos profissionais. Este cenário permitiu uma situação confortável e motivadora de resposta para cada especialista que se disponibilizou a colaborar com tal estudo.

As tarefas foram realizadas em igual número pelos especialistas. Portanto, a Tabela 4.1 apresenta as informações sobre cada especialista deste estudo, os quais foram ordenados (de forma crescente) de acordo com sua formação acadêmica. Os especialistas foram classificados com base nos seguintes fatores:

- **Formação Acadêmica**: Mestrando(a) (Mn), Mestre (Ms), Doutorando(a) (Dn), Doutor(a) (Dr);
- **Experiência com UML**: Nenhuma (N), Básica (B), Moderada (M), Avançada (A);
- **Experiência com LPS e Variabilidade**: Nenhuma (N), Básica (B), Moderada (M), Avançada (A); e
- **Experiência com Inspeção de Software**: Nenhuma (N), Básica (B), Moderada (M), Avançada (A).

Tabela 4.1: Dados de Caracterização Detalhados dos Especialistas do Estudo.

Nº e Id do Especialista	**Formação Acadêmica**	**Experiência com UML**	**Experiência com LPS**	**Experiência com Inspeção de Software**
Nº 4 - Id 4	Mn(X) Ms() Dn() Dr()	N() B() M() A(X)	N() B() M(X) A()	N() B(X) M() A()
Nº 5 - Id 5	Mn(X) Ms() Dn() Dr()	N() B(X) M() A()	N() B() M() A(X)	N(X) B() M() A()
Nº 6 - Id 7	Mn() Ms(X) Dn() Dr()	N() B() M(X) A()	N() B(X) M() A()	N() B(X) M() A()
Nº 2 - Id 2	Mn() Ms() Dn(X) Dr()	N() B(X) M() A()	N(X) B() M() A()	N(X) B() M() A()
Nº 3 - Id 3	Mn() Ms() Dn(X) Dr()	N() B() M() A(X)	N() B() M() A(X)	N() B(X) M() A()
Nº 7 - Id 11	Mn() Ms() Dn(X) Dr()	N() B() M() A(X)	N() B(X) M() A()	N() B(X) M() A()
Nº 8 - Id 13	Mn() Ms() Dn(X) Dr()	N() B() M() A(X)	N(X) B() M() A()	N(X) B() M() A()
Nº 1 - Id 1	Mn() Ms() Dn() Dr(X)	N() B() M() A(X)	N(X) B() M() A()	N(X) B() M() A()

Pode-se observar que os especialistas possuem em geral um conhecimento avançado em UML, moderado em LPS e básico em inspeção de software.

Os itens a seguir apresentam os dados obtidos de acordo com as respostas fornecidas por cada especialista do estudo.

— Os especialistas responderam individualmente às seguintes questões:

(a) Você considera o processo de inspeção da técnica *SMartyCheck* adequado com relação à identificação de defeitos em Diagramas *SMarty* de casos de uso e de classes? Justifique.

* **Resposta Especialista Nº 1 (Id 1)**: "Sim, achei o processo interessante e confere maior grau de corretude ao diagrama."
* **Resposta Especialista Nº 2 (Id 2)**: "Sim."
* **Resposta Especialista Nº 3 (Id 3)**: "Sim, pois permitiu identificar diferentes tipos de defeitos nos diagramas UML, apoiando a correção dos diagramas e evitando assim o desenvolvimento de produtos incorretos."
* **Resposta Especialista Nº 4 (Id 4)**: "Sim, acredito que se automatizado junto a um processo de desenvolvimento sistemático, como é o caso da LPS, pode agregar muito no quesito qualidade."
* **Resposta Especialista Nº 5 (Id 5)**: "Com base nas LPSs do experimento, considero a técnica com algumas inspeções redundantes."
* **Resposta Especialista Nº 6 (Id 7)**: "Sim, pois é possível através do mapeamento identificar as falhas no processo."
* **Resposta Especialista Nº 7 (Id 11)**: "Em parte. Grande parte dos itens do *checklist* poderiam ser automatizados sem perda. Verificação de existência de valores, elementos com mesmo nome, associações obrigatórias/indevidas podem ser obtidas por mecanismos automatizados. As questões subjetivas, de entendimento, escolhas e, etc., devem sim ser mantidas e utilizadas por meio de técnicas de inspeção."
* **Resposta Especialista Nº 8 (Id 13)**: "Sim, pois uma lista de verificação guia o revisor no processo de identificação de possíveis inconsistências do diagrama. Muito diferente se o revisor tiver que realizar seu trabalho efetuando uma comparação *Ad hoc*."

(b) **Você considera que o formato do *checklist* da técnica *SMartyCheck* contribui de maneira clara, objetiva e precisa para a inspeção de Diagramas *SMarty* de casos de uso e de classes? Justifique.**

* **Resposta Especialista Nº 1 (Id 1)**: "Sim, mas alguns itens causaram um pouco de confusão a princípio."
* **Resposta Especialista Nº 2 (Id 2)**: "Sim."
* **Resposta Especialista Nº 3 (Id 3)**: "Sim. A *SMartyCheck* contribuiu de maneira clara. Contudo, algumas das questões estão formuladas de forma a dificultar o entendimento. Talvez alguns poucos ajustes podem melhorá-las ainda mais."

* **Resposta Especialista Nº 4 (Id 4)**: "Sim, porém algumas perguntas me pareceram um tanto recorrentes, o que pode ser considerada uma ameaça a validade de uma proposta como esta. Talvez seja interessante confrontar algumas delas e manter apenas as com maior relevância."
* **Resposta Especialista Nº 5 (Id 5)**: "O *checklist* precisa ser de alguma forma mais resumido e mais claro. Isso deve ser resolvido com treinamento e tempo de uso da técnica."
* **Resposta Especialista Nº 6 (Id 7)**: "Sim, a ferramenta contribui para inspeção visto que os elementos são mapeados e delimitados pelas suas inconsistências."
* **Resposta Especialista Nº 7 (Id 11)**: "Sim. Entretanto existem algumas questões duplicadas e ambíguas. Em minha opinião FI.1 é igual FI.2; e IE.1 é igual IE.2 é igual Am.2. Em An.1 é preciso deixar mais claro o tipo de associação, a questão leva à marcação incorreta. Em Incons.1 é preciso deixar mais clara a implicação disso. Nos materiais não fica claro se o *maxSelection* restringe a escolha e se o diagrama especificado deve conter apenas (no máximo) aquela quantidade, ou se ainda pode haver a escolha."
* **Resposta Especialista Nº 8 (Id 13)**: "O formato contribuiu, mas as questões são bastante ambíguas e de difícil interpretação sobre o que realmente está verificando. Muitas vezes não se sabe se a pergunta se refere ao diagrama com ou sem defeito. O uso de termos associados e relacionados também deve ser melhor empregado."

(c) Você considera que os tipos de defeitos são coerentes com a descrição dos itens do *checklist* da técnica *SMartyCheck* para inspeção de Diagramas *SMarty* de casos de uso e de classes? Justifique.

* **Resposta Especialista Nº 1 (Id 1)**: "O defeito identificado como "Imodificável" me pareceu confuso quanto a sua descrição."
* **Resposta Especialista Nº 2 (Id 2)**: "Sim."
* **Resposta Especialista Nº 3 (Id 3)**: "Sim, são coerentes pois permitiram analisar em diferentes elementos possíveis erros e permitir a identificação desses para correção."
* **Resposta Especialista Nº 4 (Id 4)**: "Sim, acredito que os tipos de defeitos estão bem segmentados. Mas talvez alguns deles possam ser retirados seguindo a linha da resposta anterior."

* **Resposta Especialista Nº 5 (Id 5)**: "Alguns dão duplo sentido, levando o inspetor a interpretar de maneira errada o item do *checklist*. O item 4 (caso do DESIGN_TIME) leva a entender que usar o DESING_TIME é incorreto."

* **Resposta Especialista Nº 6 (Id 7)**: "Acredito que sim, pois expressam o problema a ser mapeado."

* **Resposta Especialista Nº 7 (Id 11)**: "Sim, os tipos de defeitos são claros e objetivos. Entretanto, poderiam ser agrupados e colocados em ordem mais natural."

* **Resposta Especialista Nº 8 (Id 13)**: "Alguns tipos não parecem necessariamente um defeito. Talvez pela descrição ambígua, não tenha entendido realmente a questão o item de verificação."

(d) Você considera que os estereótipos da *SMarty* são elementos importantes para propiciar uma melhor inspeção por meio da técnica *SMartyCheck* em Diagramas *SMarty* de casos de uso e de classes? Justifique.

* **Resposta Especialista Nº 1 (Id 1)**: "Sim, achei a especificação de estereótipos interessante e acho que a sua utilização é imprescindível para a correta leitura do diagrama."

* **Resposta Especialista Nº 2 (Id 2)**: "Sim."

* **Resposta Especialista Nº 3 (Id 3)**: "Sim, pois eles representam os elementos chave das variabilidades, pontos de variação e variantes."

* **Resposta Especialista Nº 4 (Id 4)**: "Sim, os esteriótipos são essenciais, pois apresentam de forma clara e intuitiva elementos relacionados às variabilidades de uma LPS."

* **Resposta Especialista Nº 5 (Id 5)**: "Sim, são elementos chave para a inspeção, já que ela também é baseada em sua maioria nestes estereótipos."

* **Resposta Especialista Nº 6 (Id 7)**: "Sim, pois através desta técnica, facilita o mapeamento das inconsistências de maneira mais rápida e eficiente."

* **Resposta Especialista Nº 7 (Id 11)**: "Sim. Parte das restrições impostas pelos estereótipos, entretanto, podem ser checados automaticamente."

* **Resposta Especialista Nº 8 (Id 13)**: "Sim, os estereótipos são essenciais, mas há algumas inconsistências no diagrama que o tornam confuso, principalmente sobre a anotação que possuem atributos redundantes (*name*, *variants*)

que podem ser identificados a partir de outros elementos do diagrama e o problema com os atributos "*minSelection*" e "*maxSelection*" que não são claros quanto ao seu papel: *minSelection* induz a pensar que você deve selecionar no mínimo o número de opções definidos. *maxSelection* induz a pensar que você pode selecionar no máximo o número de opções definidos. O diagrama (sem defeito) já apresenta quantas variações existem para um."

(e) Quais tipos de defeitos você entende que poderiam ser alterados e/ou removidos do *checklist* da técnica *SMartyCheck*? Justifique.

* **Resposta Especialista Nº 1 (Id 1)**: "Não compreendi exatamente os objetivos do item: Incons.3 Existe alguma variabilidade (<<`variability`>>) especificada no diagrama de classes da LPS cujo meta-atributo *bindingTime* seja DESIGN_TIME (tempo de projeto)? Já que nos dois diagramas a resposta foi sim, fiquei sem entender..."

* **Resposta Especialista Nº 2 (Id 2)**: "Nenhum."

* **Resposta Especialista Nº 3 (Id 3)**: "Nenhum. Há apenas a necessidade de rever algumas das questões que levam a inspeção."

* **Resposta Especialista Nº 4 (Id 4)**: "Incons.3: Acredito que todo diagrama seja em DESIGN_TIME. DI.2: Não sei se seria passível de remoção, mas achei interessante citá-lo, pois nunca vi o esteriótipo em nenhum diagrama *SMarty*."

* **Resposta Especialista Nº 5 (Id 5)**: "O tipo de defeito 11 verifica o *max* e *min Selection*. Existe um outro defeito que trata esta situação. O defeito 5 (Incor.2) eu ainda não entendi porque relacionar 2 itens mandatórios esta incorreto."

* **Resposta Especialista Nº 6 (Id 7)**: "Imodificável pois passa impressão de IMPOSSÍVEL."

* **Resposta Especialista Nº 7 (Id 11)**: "Como dito anteriormente, FI.1 é igual FI.2; IE.1 é igual IE.2 é igual Am.2 Ademais, Incons.1, Incom.1, An.1, Incons.3, Incons.2 (e alguns outros que não tomei nota) podem ser feitos automaticamente por análise sintática do diagrama."

* **Resposta Especialista Nº 8 (Id 13)**: "Todos os items que negativamente positivos! "Todos os itens com estereótipo, ou seja obrigatório, não estão...". Na hora de responder o revisor nunca sabe se responde: sim, eles não estão, ou não, todos estão. Ou seja, confirmo a negação ou nego afirmando. Viu,

nunca fica claro. Use questões que a pessoa responsa sim ou não. "Todos os itens encontram seu correspondente no diagrama?" "Há algum item faltando ou ausente?" É diferente perguntar: "Todos os itens não estão ausentes???!!!"

(f) Sugira "tipos de defeitos", os quais desejaria que fossem incluídos ou adotados no *checklist* da técnica *SMartyCheck*? Justifique.

* **Resposta Especialista Nº 1 (Id 1)**: "De forma geral senti falta de perguntas relacionadas a casos de uso/classes com o estereótipo diferente de <<`mandatory`>>."
* **Resposta Especialista Nº 2 (Id 2)**: "Não tenho."
* **Resposta Especialista Nº 3 (Id 3)**: "Não sugiro tipos de defeitos, contudo, questiono como seria a aplicação do *SMartyCheck* sem o diagrama correto da LPS. Neste sentido seria interessante também uma avaliação nesta perspectiva."
* **Resposta Especialista Nº 4 (Id 4)**: "Talvez seja possível identificar itens relacionados a manutenibilidade dos diagramas, identificando se o mesmo está estruturado de forma viável em termos de manutenção."
* **Resposta Especialista Nº 5 (Id 5)**: "Nenhum a adicionar."
* **Resposta Especialista Nº 6 (Id 7)**: "Acredito que tais defeitos poderiam ter menos variantes. Penso ficar complicado mapear caso tenhamos uma gama muito grande de opções."
* **Resposta Especialista Nº 7 (Id 11)**: "Não tenho sugestões."
* **Resposta Especialista Nº 8 (Id 13)**: "Nenhum, no momento."

(g) Faça uma classificação, em ordem de prioridade, dos tipos de defeitos da técnica *SMartyCheck* considerados mais relevantes para você.

* **Resposta Especialista Nº 1 (Id 1)**: 1º Regras de Negócio, 2º Incompleto, 3º Inconsistência, 4º Incorreto, 5º Anomalia, 6º Omissão, 7º Ambiguidade, 8º Fato Incorreto, 9º Instável, 10º Inviável, 11º Imodificável, 12º Informação Estranha e 13º Desvio Intencional.
* **Resposta Especialista Nº 2 (Id 2)**: 1º Regras de Negócio, 2º Omissão, 3º Inconsistência, 4º Ambiguidade, 5º Imodificável, 6º Incorreto, 7º Anomalia, 8º Instável, 9º Incompleto, 10º Inviável, 11º Desvio Intencional, 12º Informação Estranha e 13º Fato Incorreto.

* **Resposta Especialista Nº 3 (Id 3)**: 1º Ambiguidade, 2º Inconsistência, 3º Incompleto, 4º Incorreto, 5º Instável, 6º Inviável, 7º Regras de Negócio, 8º Omissão, 9º Fato Incorreto, 10º Anomalia, 11º Informação Estranha, 12º Imodificável e 13º Desvio Intencional.

* **Resposta Especialista Nº 4 (Id 4)**: 1º Incorreto, 2º Incompleto, 3º Inconsistência, 4º Omissão, 5º Instável, 6º Ambiguidade, 7º Imodificável, 8º Anomalia, 9º Desvio Intencional, 10º Informação Estranha, 11º Inviável, 12º Fato Incorreto e 13º Regras de Negócio.

* **Resposta Especialista Nº 5 (Id 5)**: 1º Ambiguidade, 2º Anomalia, 3º Inconsistência, 4º Desvio Intencional, 5º Omissão, 6º Regras de Negócio, 7º Imodificável, 8º Inviável, 9º Instável, 10º Incorreto, 11º Incompleto, 12º Informação Estranha e 13º Fato Incorreto.

* **Resposta Especialista Nº 6 (Id 7)**: 1º Incompleto, 2º Ambiguidade, 3º Informação Estranha, 4º Regras de Negócio, 5º Fato Incorreto, 6º Instável, 7º Incorreto, 8º Inconsistência, 9º Imodificável, 10º Omissão, 11º Desvio Intencional, 12º Anomalia e 13º Inviável.

* **Resposta Especialista Nº 7 (Id 11)**: 1º Regras de Negócio, 2º Anomalia, 3º Inconsistência, 4º Ambiguidade, 5º Incorreto, 6º Incompleto, 7º Omissão, 8º Informação Estranha, 9º Fato Incorreto, 10º Inviável, 11º Desvio Intencional, 12º Instável e 13º Imodificável.

* **Resposta Especialista Nº 8 (Id 13)**: 1º Regras de Negócio, 2º Inconsistência, 3º Incorreto, 4º Incompleto, 5º Fato Incorreto, 6º Informação Estranha, 7º Inviável, 8º Anomalia, 9º Omissão, 10º Ambiguidade, 11º Instável, 12º Imodificável e 13º Desvio Intencional.

4.5 Análise e Interpretação dos Resultados

Os itens a seguir descrevem os resultados obtidos com base nas respostas/*feedback* fornecidas por cada especialista de acordo com a execução deste estudo, bem como sobre a técnica *SMartyCheck*.

- **Resultados/*Feedback* Obtidos - Especialista Nº 1 (Id 1)**: Foi possível observar que o processo para a detecção de defeitos da técnica *SMartyCheck* é interessante e suporta a corretude com relação aos diagramas *SMarty* de casos de uso e de classes inspecionados. Além disso, o especialista afirmou que a especificação dos estereótipos

da *SMarty* é interessante e imprescindível para a correta leitura do diagrama. Já os itens do *checklist* da *SMartyCheck* causaram um pouco de confusão. Além disso, a descrição do item do *checklist* da *SMartyCheck*, relacionado ao tipo de defeito denominado "Imodificável", se mostrou confusa. Ainda sobre os tipos de defeitos, o especialista não compreendeu o objetivo do item do *checklist* da *SMartyCheck* relacionado ao tipo de defeito denominado Inconsistência (apenas o Incons.3). Por fim, como sugestão de melhoria da *SMartyCheck*, o especialista sugeriu que fossem adicionados mais itens no *checklist* da *SMartyCheck* com questões (perguntas) relacionadas a casos de uso/classes com o estereótipo diferente de <<`mandatory`>>.

- **Resultados/*Feedback* Obtidos - Especialista Nº 2 (Id 2)**: Todas as respostas fornecidas por este especialista foram "Sim", o qual afirmou que o processo de detecção de defeitos, o formato do checklist da *SMartyCheck*, a coerência dos tipos de defeitos e os estereótipos da *SMarty* são viáveis. Além disso, o especialista não se opôs sobre os tipos de defeitos especificados no *checklist* da *SMartyCheck* e não sugeriu nenhum tipo de defeito para possível melhora da *SMartyCheck*.

- **Resultados/*Feedback* Obtidos - Especialista Nº 3 (Id 3)**: O processo de inspeção da técnica *SMartyCheck* é adequado para detecção de diferentes tipos de defeitos em diagramas *SMarty* de casos de uso e de classes, apoiando a corretude e evitando a geração de produtos incorretos no futuro a partir de uma LPS já inspecionada por meio da *SMartyCheck*. Com relação ao formato do *checklist* da *SMartyCheck*, possui itens claros e contribui com a inspeção. Contudo, alguns itens devem ser revisados e ajustados com o intuito de melhorar a qualidade da *SMartyCheck* de modo geral. Acerca dos tipos de defeitos, o especialista afirmou que são coerentes, pois é possível analisar diferentes elementos que podem estar propensos a erros, contribuindo para que sejam corrigidos. O especialista não deseja que nenhum tipo de defeito seja alterado ou removido do *checklist* da *SMartyCheck*, além de não sugerir nenhum tipo de defeito. Porém, o especialista sugere aplicar a *SMartyCheck* sem a presença de um diagrama UML *SMarty* de LPS original, utilizado na comparação e inspeção de defeitos com base em outro diagrama UML *SMarty* de LPS com defeitos incorporados. Por fim, o especialista considerou que os estereótipos são elementos chave para propiciar uma melhor inspeção.

- **Resultados/*Feedback* Obtidos - Especialista Nº 4 (Id 4)**: Com base nas respostas de tal especialista, o processo de inspeção da técnica *SMartyCheck* é adequado para detecção de diferentes tipos de defeitos em diagramas *SMarty* de casos de uso e de classes. O especialista sugeriu a automatização do processo de inspeção da *SMartyCheck*

em conjunto com um processo de desenvolvimento sistemático, o qual destacou o cenário de LPS, ressaltando que isto poderia agregar na melhora da qualidade. Sobre os itens do *checklist* da *SMartyCheck*, o especialista os achou claros e objetivos, mas com redundâncias, sugerindo confrontar e considerar apenas os itens de maior relevância, no intuito de melhorar a proposta da *SMartyCheck*. O especialista afirmou que os tipos de defeitos estão bem segmentados, mas sugeriu tomar as mesmas providências com relação aos tipos de defeitos com base nos itens do *checklist* da *SMartyCheck*. Ainda sobre os tipos de defeitos, o especialista deseja que o item do *checklist* da *SMartyCheck* relacionado ao tipo de defeito denominado Inconsistência (apenas o Incons.3) seja removido, afirmando que todo diagrama possui o meta-atributo DESIGN_TIME, além de sugerir uma possível remoção do tipo de defeito denominado Desvio Intencional (apenas o DI.2), pois os diagramas inspecionados pelo especialista, até então, não continham o estereótipo <<mutex>>. Por fim, o especialista não sugeriu nenhum tipo de defeito para ser adotado ao *checklist* da *SMartyCheck*, mas sugeriu itens relacionados a verificar a manutenabilidade dos diagramas, bem como a estrutura dos mesmos.

- **Resultados/*Feedback* Obtidos - Especialista Nº 5 (Id 5)**: Analisando a respostas fornecidas por tal especialista, o processo de inspeção da técnica *SMartyCheck* possui redundâncias para detecção de diferentes tipos de defeitos em diagramas *SMarty* de casos de uso e de classes. Já sobre os itens do *checklist* da *SMartyCheck*, o mesmo afirmou que o *checklist* deve ser mais simplificado e mais claro, sugerindo que talvez isto seja resolvido com mais tempo de treinamento e familiaridade com a *SMartyCheck*. Além disso, o especialista afirmou que os tipos de defeitos dão duplo sentido, o que pode atrapalhar a interpretação do item e, consequentemente, levando o inspetor a uma resposta errada. Assim sendo, o especialista cita a dificultade em entender a descrição do item que contém o meta-atributo DESIGN_TIME, possibilitando uma interpretação errônea sobre o mesmo. Já sobre os tipos de defeitos, o especialista possui dúvidas a respeito de existir ou não outro item que substitua a descrição do tipo de defeito denominado Inconsistência (Incons.1 e Incons.2) relacionado, respectivamente, aos meta-atributos *maxSelection* e do *minSelection*. Além disso, o especialista alega o não entendimento e por que está errado associar dois itens que possuem o estereótipo <<mandatory>>, ressaltando que seja necessário uma passível alteração ou remoção do item denominado Incorreto (apenas o Incor.2) especificado no *checklist* da *SMartyCheck*. Ademais, o especialista não sugeriu nenhum tipo de defeito para ser analisado e/ou adotado no *checklist* da *SMartyCheck*. Enfim, o especialista afirmou que os estereótipos da *SMarty* são elementos chave para a inspeção.

- **Resultados/*Feedback* Obtidos - Especialista Nº 6 (Id 7)**: Interpretando as respostas de tal especialista, o processo de inspeção da técnica *SMartyCheck* é adequado para detecção de diferentes tipos de defeitos em diagramas *SMarty* de casos de uso e de classes. Sobre o formato do *checklist*, o especialista acredita que a técnica contribui para a inspeção e detecção de defeitos inconsistentes de acordo os itens do *checklist* da *SMartyCheck*. O especialista acredita que os tipos de defeitos são coerentes, pois deixam claro os defeitos a serem detectados. Ainda sobre os tipos de defeitos, o especialista deseja que o tipo de defeito denominado "Imodificável" seja removido do *checklist* da *SMartyCheck*, pois deixa à entender que seja "Impossível". Adicionalmente, o especialista sugere que os tipos de defeitos especificados, na forma de itens do *checklist* da *SMartyCheck*, tenham menos variantes, pois podem gerar complicações ou confusões no momento da inspeção dos diagramas *SMarty* de casos de uso e de classes de LPS. Enfim, o especialista afirma que o uso dos estereótipos da *SMarty* facilitam a detecção de insconsistências de forma rápida e eficiente.

- **Resultados/*Feedback* Obtidos - Especialista Nº 7 (Id 11)**: O processo de inspeção da técnica *SMartyCheck* é adequado em parte para detecção de diferentes tipos de defeitos, pois alguns itens do *checklist* da *SMartyCheck* poderiam ser automatizados e outros mantidos para a inspeção de forma manual. Este especialista sugeriu que os itens do *checklist* poderiam ser automatizados por mecanismos de análise sintática a partir dos diagramas *SMarty* de casos de uso e de classes de LPS inspecionados. Neste sentido, as restrições impostas pelos estereótipos também poderiam ser, de alguma forma, automatizadas. Com relação aos tipos de defeitos, o especialista afirmou que são claros e objetivos, mas podem ser agrupados em uma ordem mais natural. Contudo, alguns itens do *checklist* foram identificados pelo especialista como duplicados e ambíguos. Por fim, de acordo com a opinião deste especialista, os tipos de defeitos FI.1 e FI.2 são iguais; IE.1 é igual IE.2, que é igual Am.2; An.1 não está claro o tipo de associação; e Incons.1 não está claro sobre se o meta-atributo *maxSelection* da *SMarty* restringe a escolha e se o diagrama especificado deve conter apenas (no máximo) a quantidade definida, ou se ainda pode haver a escolha.

- **Resultados/*Feedback* Obtidos - Especialista Nº 8 (Id 13)**: De acordo com as respostas deste especialista, o processo de inspeção da técnica *SMartyCheck* é adequado para detecção de diferentes tipos de defeitos, bem como inconsistências em diagramas *SMarty* de casos de uso e de classes em comparação com uma técnica *Ad hoc*. Além disso, o formato do *checklist* contribuiu para a inspeção dos diagramas. Contudo, o especialista achou as questões (itens do *checklist*) ambíguas (redundantes) e de difícil

interpretação para serem verificadas, afirmando que existem questões ambíguas referente aos diagramas com ou sem defeitos. Adicionalmente, o especialista sugeriu maior clareza nos itens do *checklist* para diferenciar o uso dos termos "associados" e "relacionados". Sobre os tipos de defeitos, o especialista afirmou que alguns não parecem defeitos, ou seja, são incoerentes e/ou ambíguos. Portanto, o especialista sugere que alguns itens do *checklist*, negativamente positivos (que contém o "não", exemplo: "Todos os itens não estão ausentes"), sejam removidos do *checklist* para facilitar a sua compreensão. O especialista não sugere a adição/adaptação de nenhum tipo de defeito. Por fim, com relação aos estereótipos da *SMarty*, tal especialista achou que os meta-atributos *maxSelection* e *minSelection* referente ao estereótipo <<variability>>, não estão claros com relação ao seus papéis, pois induzem o inspetor a pensar em selecionar no máximo (*maxSelection*) o número de opções definidas e, no caso, do mínimo (*minSelection*) o número de opções definidas.

A ferramenta Dedoose[2] foi utilizada para organizar e codificar os dados coletados dos especialistas por meio dos conceitos de *Open Coding*. Assim, foi possível categorizar os códigos por meio de conceitos de *Axial Coding*, permitindo criar as representações gráficas dessas categorias com a utilização do software de diagramação Microsoft Visio[3].

Deste modo, durante o processo de *coding*, 3 categorias emergiram com relação a técnica *SMartyCheck*: "Viável para a Inspeção", "Possíveis Melhorias" e "Automatização da *SMartyCheck*".

Logo, a descrição das categorias é apresentada a seguir de acordo com seus códigos relacionados, bem como os trechos dos dados coletados e a categorização dos códigos para avaliar as respostas das perguntas sobre a técnica, as quais contribuiram para o refinamento da *SMartyCheck*.

Viável para a Inspeção: esta categoria avalia a viabilidade da técnica *SMartyCheck*, fornecendo indícios que algumas melhorias são essencias. A Figura 4.2 apresenta uma representação gráfica dos fatores que fazem parte desta categoria.

[2]Dedoose - http://www.dedoose.com
[3]Microsoft Visio - http://office.microsoft.com/visio/

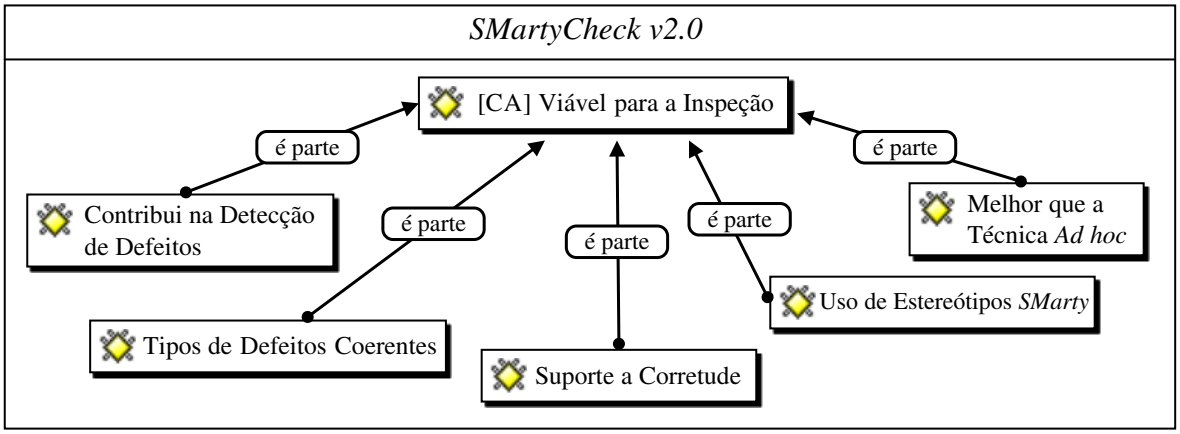


Figura 4.2: Representação Gráfica com as Associações relacionadas à Categoria "Viável para a Inspeção".

Analisando as respostas dos especialistas sobre a *SMartyCheck*, a técnica suporta a corretude com base no código Suporte a Corretude atribuído de acordo com a resposta do especialista Nº 1: "*...achei o processo interessante e confere maior grau de corretude ao diagrama.*"; e do Nº 3: "*...apoiando a correção dos diagramas.*". Além disso, a *SMartyCheck* detecta inconsistências nos diagramas *SMarty* com base na resposta do especialista Nº 6: "*...facilita o mapeamento das inconsistências.*"; e do Nº 8: "*...uma lista de verificação guia o revisor no processo de identificação de possíveis inconsistências do diagrama.*".

A *SMartyCheck* contribuiu para a detecção de defeitos de acordo com o código Contribui na Detecção de Defeitos atribuído com base nas respostas dos especialistas Nº 1, Nº 2 e Nº 4: "*Sim.*"; do Nº 3: "*Sim, pois permitiu identificar diferentes tipos de defeitos nos diagramas UML.*"; e do Nº 6: "*Sim, pois é possível através do mapeamento identificar falhas no processo.*".

Adicionalmente, a *SMartyCheck* possui tipos de defeitos coerentes com base no código Tipos de Defeitos Coerentes atribuído de acordo com a resposta do especialista Nº 3: "*Sim, são coerentes pois permitiram analisar em diferentes elementos possíveis erros e permitir a identificação desses para correção.*"; do Nº 4: "*Sim, acredito que os tipos de defeitos estão bem segmentados.*"; do Nº 6: "*Acredito que sim, pois expressam o problema a ser mapeado.*"; e do Nº 7: "*Sim, os tipos de defeitos são claros e objetivos. Entretanto, poderiam ser agrupados e colocados em ordem mais natural.*".

Além disso, alguns especialistas afirmaram que os estereótipos da *SMarty* são elementos essenciais para uma melhor inspeção de acordo com o código Uso Estereótipos *SMarty* atribuído com base na resposta do especialista Nº 1: "*Sim, achei a especificação de es-*

tereótipos interessante e acho que a sua utilização é imprescindível para a correta leitura do diagrama."; do Nº 3: "*Sim, pois eles representam os elementos chave das variabilidades, pontos de variação e variantes.*"; do Nº 4: "*Sim, os esteriótipos são essenciais, pois apresentam de forma clara e intuitiva elementos relacionados às variabilidades de uma LPS.*"; do Nº 5: "*Sim, são elementos chave para a inspeção, já que ela também é baseada em sua maioria nestes estereótipos.*"; e do Nº 8: "*Sim, os estereótipos são essenciais.*".

Ademais, a *SMartyCheck* é diferente e pode ser melhor quando comparada com a técnica *Ad hoc*, com base no código Melhor que a técnica *Ad hoc*, atribuído de acordo com a resposta do especialista Nº 8: "*Muito diferente se o revisor tiver que realizar seu trabalho efetuando uma comparação Ad hoc.*".

Possíveis Melhorias: a técnica *SMartyCheck* requer possíveis melhorias, pois têm algumas limitações, considerando redundâncias relatadas pelos especialistas sobre os itens contidos no seu *checklist*.

A *SMartyCheck* possui poucas redundâncias nos itens do *checklist* com base no código Redundância nos Itens do Checklist atribuído de acordo com a resposta do especialista Nº 4: "*...algumas perguntas me pareceram um tanto recorrentes, o que pode ser considerada uma ameaça a validade de uma proposta como esta.*"; do Nº 5: "*...considero a técnica com algumas inspeções redundantes.*"; e do Nº 7: "*...Os tipos de defeitos, Incons.1, Incom.1, An.1, Incons.3, Incons.2 podem ser feitos automaticamente por análise sintática do diagrama.*".

Além disso, a *SMartyCheck* possui descrições confusas relacionadas aos itens do *checklist*, as quais foram evidenciadas de acordo com o código Confusão nos Itens do Checklist atribuído com base na resposta do especialista Nº 1: "*alguns itens causaram um pouco de confusão a princípio.*"; do Nº 3: "*...algumas das questões estão formuladas de forma a dificultar o entendimento.*"; e do Nº 8: "*Todos os items que negativamente positivos!... o revisor nunca sabe se responde: sim, eles não estão, ou não, todos estão.*".

Os especialistas responderam que alguns tipos de defeitos são incoerentes, ambíguos e de difícil interpretação com base no código Tipos de Defeitos Incoerentes atribuído de acordo com a resposta do especialista Nº 1: "*O defeito identificado como "Imodificável"me pareceu confuso quanto a sua descrição.*"; do Nº 5: "*O defeito 5 (Incor.2) eu ainda não entendi porque relacionar 2 itens mandatorios esta incorreto.*"; do Nº 7: "*Em An.1 é preciso deixar mais claro o tipo de associação, a questão leva à marcação incorreta. Em Incons.1 é preciso deixar mais clara a implicação disso.*"; e do Nº 8: "*Alguns tipos não parecem necessariamente um defeito. Talvez pela descrição ambígua, não tenha entendido realmente a questão o item de verificação.*".

Automatização da *SMartyCheck*: analisando as respostas dos especialistas foi sugerida a automatização dos itens do *checklist* da *SMartyCheck* no futuro de acordo com o código Automatização da *SMartyCheck* atribuído com base na resposta do especialista Nº 4: "*acredito que se automatizado junto a um processo de desenvolvimento sistemático como é o caso da LPS, pode agregar muito no quesito qualidade.*".

Portanto, é claro que alguns ajustes são necessários para refinar a técnica SMartyCheck de acordo com os problemas apresentados conforme as categorias "**Possíveis Melhorias**" e "**Automatização da *SMartyCheck***". A Figura 4.3 apresenta uma representação gráfica dos fatores, que são parte da categoria "**Possíveis Melhorias**" e os tipos de defeitos sugeridos pelos especialistas que podem ser automatizados por meio das respectivas melhorias apresentadas na categoria "Automatização da *SMartyCheck*".

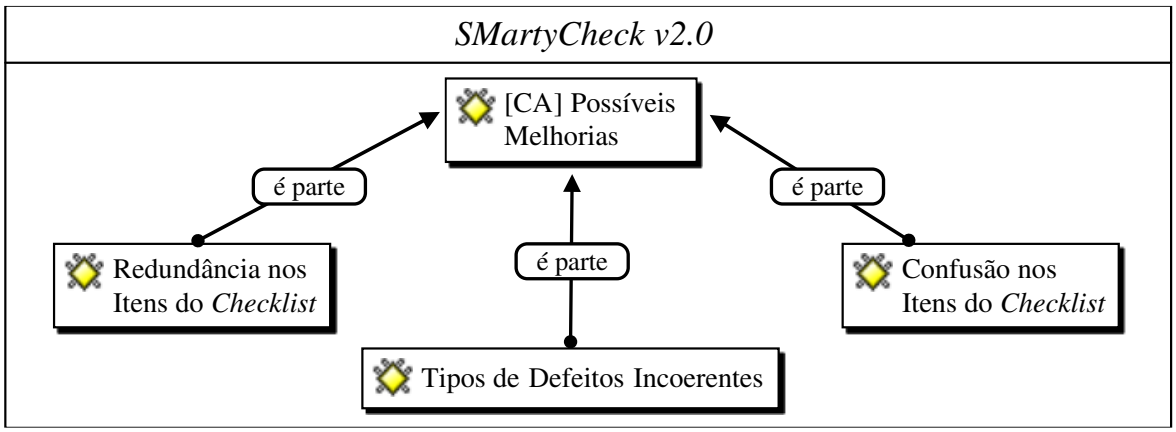


Figura 4.3: Representação Gráfica com as Associações relacionadas à Categoria "Possíveis Melhorias".

A Figura 4.3 apresenta as possíveis melhorias, por meio dos códigos definidos relacionados a *SMartyCheck*, com base em alguns itens do *checklist* identificados como redundantes (Redundância nos Itens do *Checklist*) e confusos (Confusão nos Itens do *Checklist*). Estes itens foram refinados, melhorados e apresentados na Seção 4.7. Os tipos de defeitos incoerentes também estão presentes na *SMartyCheck* e são representados por meio do código Tipos de Defeitos Incoerentes. Tais problemas compõem a categoria "Possíveis Melhorias", os quais foram revisados, melhorando e refinando a *SMartyCheck* (Seção 4.7).

Analisando os resultados obtidos de forma geral, é possível fornecer indícios de que a *SMartyCheck* é viável com base na experiência dos especialistas e na instrumentação utilizada, bem como: (i) suporta a corretude de diagramas; (ii) contribui explicitamente para a detecção de defeitos; (iii) possui tipos de defeitos coerentes; (iv) utiliza correta-

mente os estereótipos da *SMarty*, possibilitando a detecção de defeitos mais intuitiva e objetiva; e (v) parece ser mais adequada quando comparada com a técnica *Ad hoc*.

Além disso, de acordo com os resultados obtidos por meio das respostas/*feedback* dos especialistas, as quais foram analisadas/interpretadas, isto permitiu que a técnica *SMartyCheck* fosse refinada. Neste sentido, a Seção 4.7 descreve em detalhes as melhorias aplicadas na técnica *SMartyCheck*, buscando torná-la mais adequada para o contexto no qual está situada.

4.6 Avaliação de Validade do Estudo

As ameaças consideradas relevantes e identificadas com relação aos impactos para tal estudo são apresentadas nesta seção.

4.6.1 Ameaças à Validade de Conclusão

A principal ameaça para este estudo está relacionada ao número de participantes/especialistas (8). Este número é relativamente pequeno, mas o conhecimento dos especialistas que colaboraram com este estudo foi significativo. Além disso, pela característica própria do estudo, preza-se pela qualidade dos especialistas.

4.6.2 Ameaças à Validade de *Constructo*

A viabilidade foi definida e inicialmente testada com base no projeto piloto. Portanto, a instrumentação foi adequada, antes de ser aplicada no estudo real. Da mesma forma, a técnica *SMartyCheck* e a LPS *Arcade Game Maker* (AGM) também foram avaliadas no projeto piloto. Os dados obtidos em tal projeto foram descartados.

Já o nível de conhecimento exigido sobre os conceitos de modelagem UML, de LPS e de inspeção de software demonstraram-se satisfatórios para análise da técnica e consequente refinamento.

4.6.3 Ameaças à Validade Interna

As seguintes dificuldades foram identificadas:

- **Diferenças entre os especialistas**. Como a amostra de tal estudo foi pequena, algumas poucas variações com relação às habilidades dos especialistas estavam evidentes, sendo assim, os especialistas não foram divididos em grupos. Logo, as tarefas executadas pelos especialistas foram realizadas de maneira similar.

- **Acurácia das respostas dos especialistas**. Uma vez treinados sobre os conceitos básicos de LPS, variabilidades e inspeção de software, além da entendimento e compreensão da técnica *SMartyCheck*, somando-se ao nível de conhecimento que os especialistas possuíam, considera-se que a inspeção e a detecção de defeitos dos diagramas *SMarty*, por meio do *checklist* da *SMartyCheck*, é realmente válida.

- **Efeitos de fadiga**. O treinamento do estudo teve duração de 50 minutos, em média. Após isso, os formulários eletrônicos, foram enviados aos especialistas para serem respondidos durante um período de até 27 dias. Deste modo, os efeitos de fadiga foram reduzidos, pois os especialistas poderiam responder os questionários eletrônicos com certa tranquilidade com relação a adequarem seu tempo, diminuindo a pressão de entrega das respostas.

- **Outros fatores importantes**. A influência entre os especialistas foi mitigada, pois os questionários eletrônicos enviados para serem respondidos pelos mesmos foram distribuídos de forma individual e direcionados de maneira aleatória para cada especialista. Além disso, é provável que cada especialista tenha respondido aos questionários eletrônicos em datas e horários distintos, além de locais diferentes.

4.6.4 Ameaças à Validade Externa

Duas ameaças foram identificadas:

- **Instrumentação**. As LPSs pedagógicas utilizadas no treinamento (LPS *Mobile Media*) e no estudo empírico (LPS *Arcade Game Maker* (AGM)) podem colocar em risco a validade externa. Portanto, tentou-se utilizar documentos padronizados a técnica *SMartyCheck* e à LPS AGM. Contudo, por estas LPSs não serem comerciais, algumas suposições podem ser feitas sobre esta ameaça. Desta forma, outros estudos empíricos adicionais podem ser conduzidos, a fim de obter resultados diferentes na indústria por meio de outras LPSs.

- **Participantes**. A obtenção de especialistas qualificados foi uma das grandes dificuldades encontradas para este estudo. Assim, especialistas do meio acadêmico com um nível de conhecimento relevante em engenharia de software foram convidados para tentar amenizar tal ameaça.

4.7 Refinamento da *SMartyCheck 2.0* com os Resultados Obtidos

De acordo com as análises e a interpretação dos resultados obtidos no estudo empírico qualitativo de viabilidade realizado conforme a Seção 4.5, a técnica *SMartyCheck 2.0* foi refinada, evoluindo para a versão *SMartyCheck 2.1*. Deste modo, esta seção descreve em detalhes o que foi alterado e/ou removido da *SMartyCheck 2.0*. Portanto, os itens a seguir apresentam estas modificações e, ao final desta seção, o novo *checklist* da *SMartyCheck 2.1* refinado para diagramas *SMarty* de casos de uso e de classes é apresentado.

Alguns itens do *checklist* da *SMartyCheck* foram alterados, corrigidos ou removidos, na qual a técnica ficou com dezoito (18) itens especificados. Assim, as seguintes modificações foram feitas no *checklist* da *SMartyCheck*:

- O item do *checklist* da *SMartyCheck* relacionado com o tipo de defeito "Incorreto (Incor.2)" foi removido do *checklist*, pois seu objetivo não era claro e estava descrito de maneira errônea. A resposta do especialista Nº 5 permitiu identificar este problema;

- O item do *checklist* da *SMartyCheck* relacionado com o tipo de defeito "Inconsistência (Incons.3)" for removido, pois seu questionamento estava ambíguo e sem nenhum sentido. As respostas dos especialistas Nº 1, Nº 4 e Nº 5 permitiram identificar este problema;

- As respostas dos participantes Nº 1 e Nº 7 alegavam a dificuldade em compreeder a descrição do item do *checklist* relacionado com o tipo de defeito "Imodificável (Imo.1)", sendo alterado para "Imutável (Imu.1)" com sua descrição alterada no intuito de ficar claro para a leitura e interpretação;

- O item do *checklist* da *SMartyCheck* relacionado com o tipo de defeito "Instável (Ins.1)" foi alterado pelo autor, pois identificou-se que sua descrição dificultava a leitura, causando uma má interpretação para os inspetores;

- O item do *checklist* da *SMartyCheck* relacionado com o tipo de defeito "Inviável (Inv.1)" foi alterado pelo autor, pois identificou-se que sua descrição dificultava a leitura, causando uma má interpretação para os inspetores;

- Buscou-se remover a redundância e confusão dos itens do *checklist* com a finalidade de deixar claro qual LPS deve ser inspecionada de acordo com o tipo de defeito, ou seja, a LPS (oráculo - sem defeitos), a LPS (com defeitos incorporados) ou as duas LPS devem

ser comparadas para detectar os defeitos de acordo com os questionamentos dos itens do *checklist* da *SMartyCheck*. A resposta do especialista Nº 3 permitiu identificar este problema;

- A fim de correção, os termos "relacionado(os)(a)(as)" foram alterados todos para "associado(os)(a)(as)" no intuito de facilitar a leitura e compreensão dos itens do *checklist* da *SMartyCheck* durante a inspeção. A resposta do especialista Nº 10 permitiu identificar este problema; e

- Também a fim de correção, os termos "marcado(os)(a)(as)" foram alterados todos para "especificado(os)(a)(as)" no intuito de facilitar a leitura e compreensão dos itens do *checklist* da *SMartyCheck* durante a inspeção. Este problema foi identificado pelo autor deste trabalho, com base no problema reportado anteriormente sobre os termos "relacionado" e "associado".

Além disso, os especialistas classificaram os tipos de defeitos do *checklist* da *SMartyCheck* por ordem de prioridade. A partir das 8 respostas dos especialistas, a maior frequência de cada tipo de defeito com base nas respectivas posições de classificação foi considerada para a classificação final:

- **Classificação Final dos Tipos de Defeitos**: 1º Regras de Negócio, 2º Incompleto, 3º Inconsistência, 4º Incorreto, 5º Fato Incorreto, 6º Ambiguidade, 7º Imodificável, 8º Anomalia, 9º Instável, 10º Inviável, 11º Omissão, 12º Informação Estranha e 13º Desvio Intencional.

Portanto, com base nas modificações apresentadas e na classificação final dos tipos de defeitos, os *checklists* da *SMartyCheck 2.1* para inspeção de diagramas *SMarty* de casos de uso e de classes de LPS são apresentados de modo refinado com as devidas modificações de acordo com os resultados obtidos no estudo empírico qualitativo.

Assim, os itens a seguir apresentam os tipos de defeitos e os itens do *checklist* da *SMartyCheck 2.1* refinado para a inspeção de diagramas *SMarty* de casos de uso de LPS:

– **Com Oráculo (inspecionar comparando os diagramas sem defeitos e com defeitos incorporados)**

01. Regras de Negócio (RN)

- Este tipo de defeito deve ser aplicado de acordo com a definição de um domínio específico com base em uma LPS (oráculo/original) previamente escolhida.
  O Domínio/Regras de Negócio Pré-definidos para uma possível nova LPS,

neste caso modificada, devem ser descrito(as) neste espaço.

RN.1 Os casos de uso não são claros com o objetivo e as funcionalidades desejadas com base no domínio definido?

03. Inconsistência (Incons)

- **Incons.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior que o definido em *maxSelection* da variabilidade (<<`variability`>>) associada?
- **Incons.2** Existe algum caso de uso com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é menor que o definido em *minSelection* da variabilidade (<<`variability`>>) associada?

05. Fato Incorreto (FI)

- **FI.1** Existe algum caso de uso com seu nome incorreto na LPS?
- **FI.2** Existe um ou mais casos de uso que não é possível comparar seu nome na LPS?

07. Imutável (Imu)

- **Imu.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>>, no qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas de acordo as variantes especificadas no meta-atributo *variants* na LPS?

11. Omissão (Om)

- **Om.1** Existe algum caso de uso especificado como obrigatório por meio do estereótipo <<`mandatory`>> que não esteja especificado na LPS?

12. Informação Estranha (IE)

- **IE.1** Existe algum caso de uso especificado além dos casos de uso já existentes na LPS?
- **IE.2** Existe algum caso de uso com sua funcionalidade duplicada na LPS?

13. Desvio Intencional (DI)

- **DI.1** Existe algum caso de uso que exige a seleção de outro (<<`requires`>>) e esse outro não está especificado na LPS?
- **DI.2** Existe algum caso de uso que exige a não seleção de outro (<<`mutex`>>) e esse outro está especificado na LPS?

– **Sem o Oráculo (inspecionar apenas o diagrama com defeitos incorporados)**

02. Incompleto (Incom)

- **Incom.1** Todos os casos de uso com <<`mandatory`>>, ou seja, obrigatórios não estão especificados no diagrama de casos de uso da LPS?

04. Incorreto (Incor)

- **Incor.1** Existe algum caso de uso com o estereótipo <<`variationPoint`>> que está associado com um caso de uso na LPS que não seja <<`alternative_OR`>>?

06. Ambiguidade (Am)

- **Am.1** Existe algum caso de uso em que seu nome não reflete no seu objetivo real?
- **Am.2** Existe algum caso de uso em que seu nome é igual a outro caso de uso de modo a possuir uma interpretação duplicada?

08. Anomalia (An)

- **An.1** Existe algum caso de uso especificado por meio do estereótipo <<`alternative_OR`>> associado com um caso de uso que não esteja especificado com o estereótipo <<`variationPoint`>>?

09. Instável (Ins)

- **Ins.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>>, o qual possui associado o estereótipo <<`variability`>> cujo meta-atributo *name* seja igual a de outros casos de uso especificados com o estereótipo <<`variationPoint`>>?

10. Inviável (Inv)

- **Inv.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas não (false) permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar*?

12. Informação Estranha (IE)

- **IE.2** Existe algum caso de uso com sua funcionalidade duplicada na LPS?

Assim, os itens a seguir apresentam os tipos de defeitos e os itens do *checklist* da *SMartyCheck 2.1* refinado para a inspeção de diagramas *SMarty* de classes de LPS:

– **Com Oráculo (inspecionar comparando os diagramas sem defeitos e com defeitos incorporados)**

01. Regras de Negócio (RN)

- Este tipo de defeito deve ser aplicado de acordo com a definição de um domínio específico com base em uma LPS (oráculo/original) previamente escolhida.
  O Domínio/Regras de Negócio Pré-definidos para uma possível nova LPS, neste caso modificada, devem ser descrito(as) neste espaço.
  RN.1 As classes não são claras com o objetivo e as funcionalidades desejadas com base no domínio definido?

03. Inconsistência (Incons)

- **Incons.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior que o definido em *maxSelection* da variabilidade (<<`variability`>>) associada?
- **Incons.2** Existe alguma classe com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é menor que o definido em *minSelection* da variabilidade (<<`variability`>>) associada?

05. Fato Incorreto (FI)

- **FI.1** Existe alguma classe com seu nome incorreto na LPS?
- **FI.2** Existe uma ou mais classes que não é possível comparar seu nome na LPS?

07. Imutável (Imu)

- **Imu.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>>, na qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas de acordo as variantes especificadas no meta-atributo *variants* na LPS?

11. Omissão (Om)

- **Om.1** Existe alguma classe especificada como obrigatória por meio do estereótipo <<`mandatory`>> que não esteja especificada na LPS?

12. Informação Estranha (IE)

- **IE.1** Existe alguma classe especificada além das classes já existentes na LPS?
- **IE.2** Existe alguma classe com sua funcionalidade duplicada na LPS?

13. Desvio Intencional (DI)

- **DI.1** Existe alguma classe que exige a seleção de outra (<<`requires`>>) e essa outra não está especificada na LPS?
- **DI.2** Existe alguma classe que exige a não seleção de outra (<<`mutex`>>) e essa outra está especificada na LPS?

– **Sem o Oráculo (inspecionar apenas o diagrama com defeitos incorporados)**

02. Incompleto (Incom)

- **Incom.1** Todos as classes com <<`mandatory`>>, ou seja, obrigatórias não estão especificadas no diagrama de classes da LPS?

04. Incorreto (Incor)

- **Incor.1** Existe alguma classe com o estereótipo <<`variationPoint`>> que está associada com uma classe na LPS que não seja <<`alternative_OR`>>?

06. Ambiguidade (Am)

- **Am.1** Existe alguma classe em que seu nome não reflete no seu objetivo real?
- **Am.2** Existe alguma classe em que seu nome é igual a outra classe de modo a possuir uma interpretação duplicada?

08. Anomalia (An)

- **An.1** Existe alguma classe especificada por meio do estereótipo <<`alternative_OR`>> associada com uma classe que não esteja especificada com o estereótipo <<`variationPoint`>>?

09. Instável (Ins)

- **Ins.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>>, na qual possui associada o estereótipo <<`variability`>> cujo meta-atributo *name* seja igual a de outras classes especificadas com o estereótipo <<`variationPoint`>>?

10. Inviável (Inv)

- **Inv.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas não (false) permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar*?

12. Informação Estranha (IE)

- **IE.2** Existe alguma classe com sua funcionalidade duplicada na LPS?

4.8 Considerações Finais

Os resultados obtidos a partir da análise qualitativa baseada no *feedback* dos especialistas permitiu identificar 3 categorias e seus respectivos códigos (*codings*) para avaliar a viabilidade da técnica *SMartyCheck*. Desta forma, os resultados qualitativos contribuiram de fato para refinar a *SMartyCheck* auxiliando na identificação de tipos de defeitos que necessitam de ajuste e, consequentemente, aplicando correção nos itens do *checklist* considerados ambíguos e difícil compreensão.

Além disso, o *feedback* dos especialistas forneceram tanto indícios, quanto evidências de que a *SMaryCheck* é viável, pois afirmaram que a técnica suporta a corretude de diagramas *SMarty* de LPS e a identificação de inconsistências. Em adição, a *SMartyCheck* contribui com a inspeção por meio dos estereótipos da *SMarty*, os quais são elementos chaves essenciais para melhorar a detecção de defeitos. Ademais, a *SMartyCheck* possui tipos de defeitos coerentes, sendo diferente em comparação com uma técnica *Ad hoc*, na qual não possui uma estrutura bem definida.

Deve-se reconhecer que novos estudos empíricos devem ser conduzidos no ambiente acadêmico e na indústria para avaliar de forma mais precisa a *SMartyCheck*. O corpo de conhecimento produzido por este estudo nos permite construir um estudo quantitativo acerca da eficiência, eficácia e efetividade da *SMartyCheck* com relação a outro tipo de técnica de inspeção, como sugerido pelos especialistas. Assim, o próximo capítulo apresenta em detalhes um estudo quantitativo conduzido com o intuito de avaliar a eficiência, eficácia e efetividade da *SMartyCheck*.

5

Estudo Empírico Quantitativo da *SMartyCheck 2.1*

"Os números governam o mundo."

Pitágoras (570 - 490 a.C.), Filósofo e Matemático Grego

5.1 Considerações Iniciais

Este Capítulo apresenta e descreve, o planejamento, a execução e a análise dos dados do estudo empírico quantitativo, o qual tem o objetivo de avaliar a eficiência, a eficácia e a efetividade da técnica *SMartyCheck 2.1* em comparação com a técnica *Ad hoc*. Para tanto, diversas métricas/equações foram utilizadas para coletar os dados obtidos com as inspeções realizadas pelos participantes.

A Figura 5.1 apresenta uma síntese das etapas que compreendem desde a definição do objetivo, o projeto, a execução e a análise de dados deste estudo.

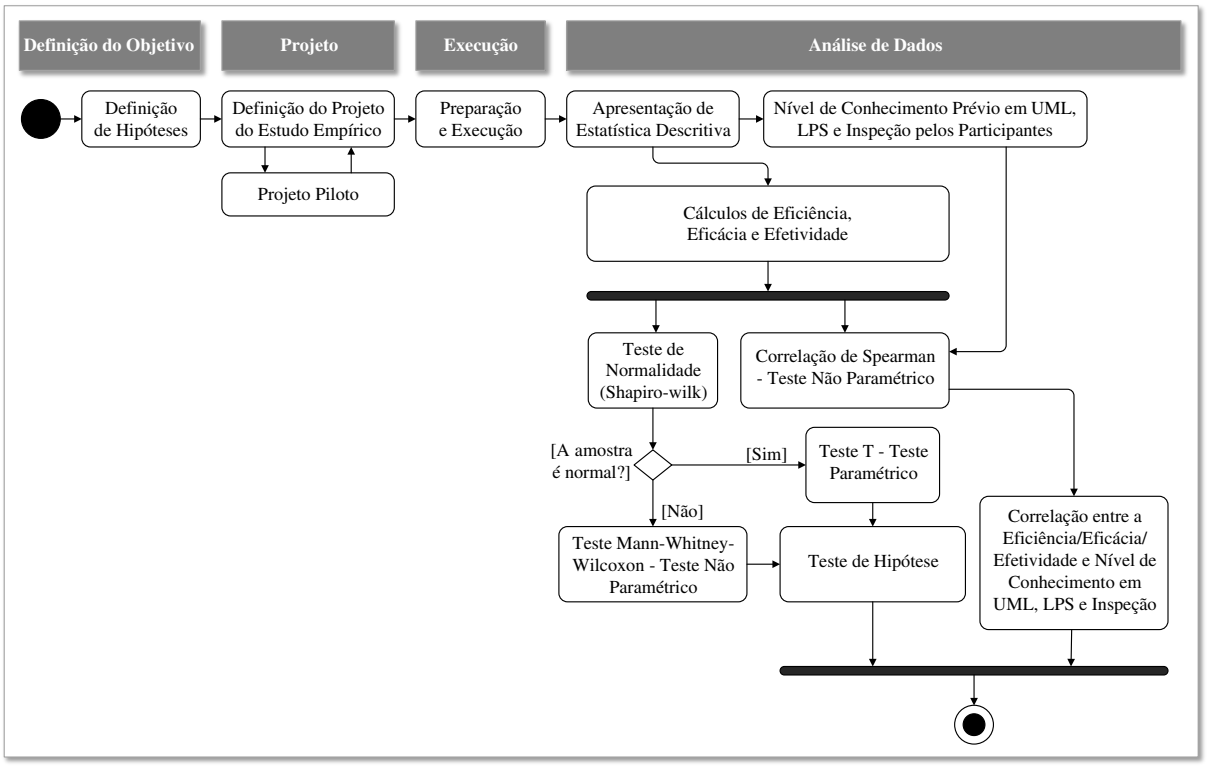


Figura 5.1: Etapas do Estudo Empírico Quantitativo.

A princípio, os dados coletados de acordo com as inspeções realizadas por cada participante foram tabulados, sendo aplicado o teste de normalidade Shapiro-Wilk (Shapiro e Wilk, 1965). Após isso, com base nos resultados obtidos por meio de tal teste foi possível verificar se a amostra era normal ou não. De acordo com a amostra foram aplicados os seguintes testes estatísticos para comparar a eficiência, a eficácia e a efetividade com relação a diferença estatística obtida entre a técnica *SMartyCheck* e a técnica *Ad hoc*: o Teste T - Paramétrico (Wohlin *et al.*, 2012) e o Teste Mann-Whitney-Wilcoxon - Não Paramétrico (Hole, 2011; Juristo e Moreno, 2010; Wohlin *et al.*, 2012). Assim, foi possível testar as hipóteses (Seção 5.3) definidas para este estudo.

Paralelamente aos testes[1] de hipótese foram realizadas as correlações de Spearman para eficiência, eficácia e efetividade com relação ao nível de conhecimento dos participantes em UML, LPS e inspeção de software. Para tanto, foram atribuídos pesos de 1-5 para respostas de UML, pesos de 1-5 para respostas de LPS, e de 1-5 para respostas de inspeção de software com base na escala de Likert (1932). Com isso, a partir desses pesos, a

[1]Os seguintes softwares foram utilizados para auxiliar nos cálculos: Microsoft Excel 2013 (http://office.microsoft.com/excel/), StatSoft Statistica 10 (http://www.statsoft.com/), Wolfram Mathematica 9 (http://www.wolfram.com/mathematica/) e o IBM SPSS Statistics 22 (http://www-01.ibm.com/software/analytics/spss/).

correlação de Spearman (Spearman, 1987; Wohlin *et al.*, 2012) foi aplicada para indicar o nível de correlação e analisar possíveis indícios do nível de conhecimento do participante com relação ao resultado obtido com base na aplicação de determinada técnica de inspeção.

O planejamento deste estudo empírico, bem como a sua execução seguem as recomendações e *templates* experimentais de Juristo e Moreno (2010); Wohlin *et al.* (2012).

5.2 Definição do Estudo

O principal propósito deste estudo empírico quantitativo foi responder a questão: "**A técnica *SMartyCheck 2.1* é eficiente, eficaz e efetiva para inspeção de diagramas *SMarty* de casos de uso e de classes em comparação com a técnica *Ad hoc*?**".

Deste modo, o objetivo do estudo (Basili *et al.*, 1994) é:

Analisar *a técnica SMartyCheck*

Com o propósito de *caracterizar*

Em relação à *sua eficiência, eficácia e efetividade em comparação com a técnica Ad hoc*

Do ponto de vista *de inspetores de artefatos de software de LPS*

No contexto de *acadêmicos de graduação e pós-graduação da área de Engenharia de Software da Universidade Estadual de Maringá (UEM), Universidade Paranaense (UNIPAR) e da Universidade Estadual do Oeste do Paraná (UNIOESTE), além de profissionais que atuam no meio industrial.*

Assim, métricas de acordo com os estudos de Gopalakrishnan Nair *et al.* (2012); Karg *et al.* (2011), foram adotadas nas inspeções realizadas por cada participante com o objetivo retornar a eficiência, eficácia e a efetividade na detecção de defeitos. Tais métricas são apresentadas e definidas a seguir:

- a Equação 5.1 que calcula a eficiência (Gopalakrishnan Nair *et al.*, 2012) de determinada técnica t na detecção de defeitos é definida por meio do número total de defeitos detectados durante inspeções realizadas, dividido pelo tempo total despendido das inspeções.

$$\textbf{\textit{Eficiência(t)}} = \frac{Total\ de\ defeitos\ detectados}{Tempo\ total\ de\ inspecao} \tag{5.1}$$

- para calcular a eficácia das inspeções realizadas por cada participante, os estudos de Mello *et al.* (2014); Sabaliauskaite (2004); Sabaliauskaite *et al.* (2002a) apresentam a métrica de Fagan (1976) denominada *Error Detection Efficiency*, a qual foi adotada

neste estudo para permitir calcular a eficácia das inspeções. Deste modo, a Equação 5.2 retorna o resultado de *Eficácia(t)*. Portanto, para calcular a eficácia é considerado o número total de defeitos detectados durante inspeções realizadas, dividido pelo número total de defeitos existentes antes das inspeções para uma determinada técnica *t*. A Equação 5.2 apresenta tal relação.

$$\textbf{\textit{Eficácia(t)}}\ (\%) = \frac{Total\ de\ defeitos\ detectados}{Total\ de\ defeitos\ existentes} * 100 \tag{5.2}$$

- o cálculo da efetividade (Equação 5.3) foi adotado com base nos estudos de Basili e Selby (1987); Marcolino *et al.* (2014a). Tal cálculo é utilizado para o número de elementos modelados corretamente subtraído ao número de elementos modelados incorretamente, sendo identificados em variabilidades de LPS. Assim, o cálculo foi adaptado para este estudo considerando o número de defeitos detectados corretamente ($nDefC$) subtraído ao número de defeitos detectados incorretamente ($nDefI$) para uma determinada técnica *t*.

$$\textbf{\textit{Efetividade(t)}} = \begin{cases} nDefC, & se\ \ nDefI = 0 \\ nDefC - nDefI, & se\ \ nDefI > 0 \end{cases} \tag{5.3}$$

5.3 Planejamento do Estudo

As etapas correspondentes ao planejamento deste estudo empírico quantitativo são apresentadas a seguir.

Projeto Piloto: um estudo piloto foi realizado com o objetivo de avaliar a instrumentação utilizada, para adequá-la. Para tanto, um docente com doutorado, e com conhecimento da área de engenharia de software, avaliou a instrumentação de tal estudo. Os dados obtidos no projeto piloto não foram utilizados como resultados deste estudo empírico.

Treinamento: foram apresentados aos participantes exemplos de inspeção de diagramas *SMarty* de casos de uso e de classes por meio da explicação dos tipos de defeitos contidos no *checklist* da *SMartyCheck* utilizando a LPS *Mobile Media* para exemplificá-los. Para a explicação da técnica *Ad hoc* os participantes foram treinados sobre os tipos de defeitos por meio de exemplos de inspeção utilizando a LPS *Mobile Media*.

Participantes: a maioria dos participantes são da área de engenharia de software. Tais participantes são: 3 graduandos (21,5%), 1 graduado (7,5%) e 10 mestrandos (71%). Tais participantes são acadêmicos e profissionais, os quais trabalham diretamente com a área ou que têm conhecimento sobre verificação e validação de software ao longo dos últimos anos em diferentes universidades no Brasil.

Instrumentação: todos os participantes do estudo receberam um conjunto de documentos: (i) uma cópia do Termo de Consentimento Livre e Esclarecido (TCLE); (ii) uma cópia do Questionário de Caracterização de Participante, no qual o participante indicará sua formação acadêmica, seu nível de experiência com modelagem (notação UML), seu conhecimento sobre LPS e inspeção de software; (iii) uma cópia de um documento com uma síntese sobre os Conceitos de Linha de Produto de Software (LPS); (iv) uma cópia de um documento com a descrição geral da LPS *Arcade Game Maker* (AGM), idêntico ao contido no Apêndice C; e (v) uma cópia de um documento com a descrição geral dos estereótipos do perfil *SMartyProfile* (Seção 2.2.1) da abordagem *SMarty*, os quais são utilizados no *checklist* elaborado da técnica *SMartyCheck* (Capítulo 3).

Além disso, os participantes foram divididos em dois grupos, sendo que, um grupo recebeu os seguintes documentos com relação a técnica *Ad hoc*: (i) uma cópia de um documento resumido sobre os Conceitos de Inspeção de Software e sobre a Técnica *Ad hoc*; e (ii) uma cópia de um documento sobre o treinamento aplicado referente a técnica *Ad hoc* com exemplos, utilizando a LPS *Mobile Media* (Apêndice D).

O outro grupo recebeu os seguintes documentos com relação a técnica *SMartyCheck*: (i) uma cópia de um documento resumido sobre os Conceitos de Inspeção de Software e sobre a Técnica de Inspeção baseada em *Checklist*; e (ii) uma cópia de um documento sobre o treinamento aplicado referente a técnica *SMartyCheck* com exemplos do *checklist* da mesma, utilizando a LPS *Mobile Media* (Apêndice D).

A instrumentação apresentada pode ser utilizada pelos participantes para consulta, com a finalidade de auxiliá-los a responderem os formulários eletrônicos criados no software LimeSurvey[2] (Apêndice E) com relação as inspeções que devem ser realizadas por meio do *checklist* da técnica *SMartyCheck* ou por meio da utilização da técnica *Ad hoc*. Para tanto, os participantes utilizaram a LPS AGM, pois é uma LPS pedagógica para jogos, criada pelo (SEI, 2009) para apoiar o aprendizado e a experimentação de conceitos de

[2]LimeSurvey - *http://www.limesurvey.org*

LPS. Assim, foram criados doze formulários eletrônicos, os quais foram distribuídos e enviados por email aleatoriamente para os participantes. Estes formulários (questionários) eletrônicos são descritos a seguir:

- Formulários da instrumentação para *SMartyCheck*:
  - três formulários foram criados, aleatorizados e enviados por email a cada participante do estudo. Cada formulário contendo dois diagramas *SMarty* de casos de uso da LPS AGM, um diagrama sem defeitos (oráculo) e um outro diagrama com defeitos, a fim de serem comparados e inspecionados, no intuito de permitirem aos participantes detectar defeitos por meio do *checklist* da técnica *SMartyCheck*, no qual está presente, no mesmo formulário, logo em seguida dos diagramas a serem inspecionados em cada formulário; e
  - três formulários foram criados, aleatorizados e enviados por email a cada participante do estudo. Cada formulário contendo dois diagramas *SMarty* de classes da LPS AGM, um diagrama sem defeitos (oráculo) da LPS AGM e um outro diagrama com defeitos, a fim de serem comparados e inspecionados, no intuito de permitirem aos participantes detectar defeitos por meio do *checklist* da técnica *SMartyCheck*, no qual está presente, no mesmo formulário, logo em seguida dos diagramas a serem inspecionados em cada formulário.
- Formulários da instrumentação para *Ad hoc*:
  - três formulários foram criados, aleatorizados e enviados por email a cada participante do estudo. Cada formulário contendo dois diagramas *SMarty* de casos de uso da LPS AGM, um diagrama sem defeitos (oráculo) e um outro diagrama com defeitos, a fim de serem comparados e inspecionados, no intuito de permitirem aos participantes detectar defeitos por meio da técnica *Ad hoc*, no qual está presente, no mesmo formulário, logo em seguida dos diagramas a serem inspecionados em cada formulário; e
  - três formulários foram criados, aleatorizados e enviados por email a cada participante do estudo. Cada formulário contendo dois diagramas *SMarty* de classes da LPS AGM, um diagrama sem defeitos (oráculo) da LPS AGM e um outro diagrama com defeitos, a fim de serem comparados e inspecionados, no intuito de permitirem aos participantes detectar defeitos por meio da técnica *Ad hoc*, no qual está presente, no mesmo formulário, logo em seguida dos diagramas a serem inspecionados em cada formulário.

Formulação das Hipóteses: as seguintes hipóteses foram definidas para este estudo empírico quantitativo:

- **Hipótese sobre Eficiência**:

  - **Hipótese Nula (H_i0)**: Não há diferença entre a eficiência das inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* e com a técnica *Ad hoc*.

$$H_i0 : \mu((Eficiencia(SMartyCheck)) = \mu((Eficiencia(Adhoc)) \quad (5.4)$$

  - **Hipótese Alternativa (H_i1)**: A eficiência nas inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* é, em média, maior do que as inspeções realizadas com a técnica *Ad hoc*.

$$H_i1 : \mu((Eficiencia(SMartyCheck)) > \mu((Eficiencia(Adhoc)) \quad (5.5)$$

  - **Hipótese Alternativa (H_i2)**: A eficiência nas inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* é, em média, menor do que as inspeções realizadas com a técnica *Ad hoc*.

$$H_i2 : \mu((Eficiencia(SMartyCheck)) < \mu((Eficiencia(Adhoc)) \quad (5.6)$$

- **Hipótese sobre Eficácia**:

  - **Hipótese Nula (H_a0)**: Não há diferença entre a eficácia das inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* e com a técnica *Ad hoc*.

$$H_a0 : \mu((Eficacia(SMartyCheck)) = \mu((Eficacia(Adhoc)) \quad (5.7)$$

  - **Hipótese Alternativa (H_a1)**: A eficácia nas inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* é, em média, maior do que as inspeções realizadas com a técnica *Ad hoc*.

$$H_a1 : \mu((Eficacia(SMartyCheck)) > \mu((Eficacia(Adhoc)) \quad (5.8)$$

- **Hipótese Alternativa (H_a2)**: A eficácia nas inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* é, em média, menor do que as inspeções realizadas com a técnica *Ad hoc*.

$$H_a2 : \mu((Eficacia(SMartyCheck)) < \mu((Eficacia(Adhoc)) \quad (5.9)$$

- **Hipótese sobre Efetividade**:

  - **Hipótese Nula (H_e0)**: Não há diferença entre a efetividade das inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* e com a técnica *Ad hoc*.

$$H_e0 : \mu((Efetividade(SMartyCheck)) = \mu((Efetividade(Adhoc)) \quad (5.10)$$

  - **Hipótese Alternativa (H_e1)**: A efetividade nas inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* é, em média, maior do que as inspeções realizadas com a técnica *Ad hoc*.

$$H_e1 : \mu((Efetividade(SMartyCheck)) > \mu((Efetividade(Adhoc)) \quad (5.11)$$

  - **Hipótese Alternativa (H_e2)**: A efetividade nas inspeções realizadas em diagramas *SMarty* de casos de uso e de classes com a técnica *SMartyCheck* é, em média, menor do que as inspeções realizadas com a técnica *Ad hoc*.

$$H_e2 : \mu((Efetividade(SMartyCheck)) < \mu((Efetividade(Adhoc)) \quad (5.12)$$

Variáveis Independentes: a técnica de inspeção corresponde ao fator técnica, com dois tratamentos: técnica *SMartyCheck 2.1* e técnica *Ad hoc*. A LPS AGM é representada por uma variável de entrada com valor fixo, conforme ilustra a Figura 5.2.

Variáveis Dependentes: a eficiência, a eficácia e a efetividade são calculadas para cada técnica de inspeção com base nos diagramas *SMarty* de casos de uso e de classes inspecionados. Logo, a LPS AGM foi utilizada no estudo de acordo com a capacidade aleatória aplicada nos diagramas *SMarty* de casos de uso e de classes distribuídos para os participantes, conforme a instrumentação definida para tal estudo.

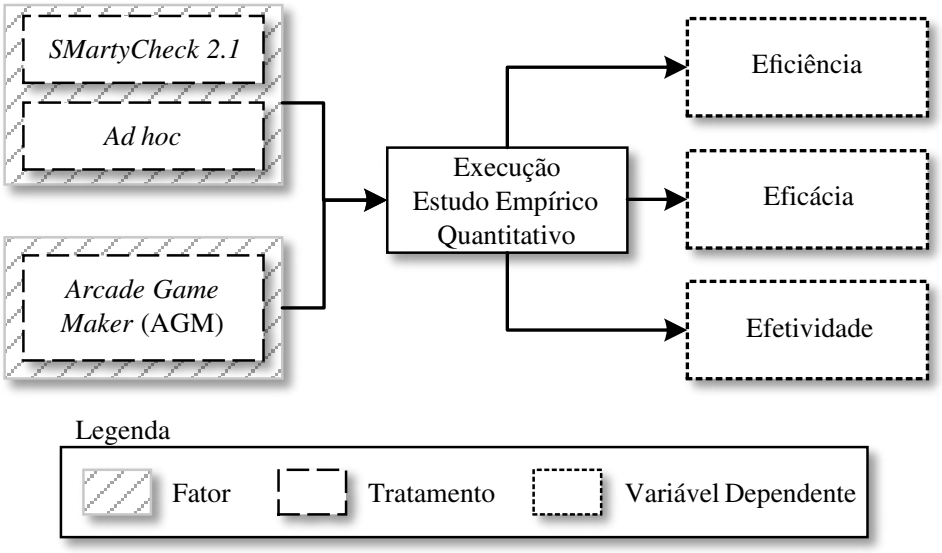


Figura 5.2: Conjunto de Variáveis do Estudo Empírico Quantitativo.

Capacidade Aleatória: a seleção de participantes não ocorreu de forma aleatória sob o universo de candidatos, já que tal universo é bem restrito. A capacidade aleatória foi exercida com relação à distribuição dos objetos do estudo (diagramas *SMarty* de casos de uso e de classes) e da distribuição das técnicas *SMartyCheck* e *Ad hoc* para os participantes.

Classificação em Bloco: um grupo (bloco) de participantes realizou inspeções de diagramas *SMarty* de casos de uso e de classes por meio do *checklist* da *SMartyCheck* e, o outro grupo, realizou inspeções por meio da técnica *Ad hoc*.

Balanceamento: as respectivas tarefas foram distribuídas igualmente para todos os participantes.

5.4 Execução do Estudo

Nesta seção são apresentadas as etapas executadas para a obtenção dos resultados referentes a aplicação do estudo empírico quantitativo.

Procedimentos de Participação: os itens a seguir, números em ordem cronológica, apresentam os procedimentos relacionados a participação adequada para tal estudo:

1. o participante recebe documentos necessários ao estudo para preencher e assinar;

2. o participante faz a leitura do termo, esclarece possíveis dúvidas e assina o Termo de Consentimento Livre e Esclarecido (TCLE);

3. o experimentador/pesquisador anota o Id (gerado no software LimeSurvey) do participante no Questionário de Caracterização de Participante;

4. o participante faz a leitura do Questionário de Caracterização de Participante, esclarece possíveis dúvidas, preenche suas informações e assina o questionário;

5. o experimentador/pesquisador ministra o treinamento sobre a técnica *Ad hoc* para o outro grupo de participantes, permitindo aos mesmos esclarecem dúvidas sobre a mesma;

6. o experimentador/pesquisador pede cordialmente aos participantes do treinamento sobre a técnica *Ad hoc* para que se retirem do local, permitindo que o treinamento sobre a técnica *SMartyCheck* seja iniciado com o outro grupo, garantindo que não nenhum tipo de viés ocorra durante a condução deste estudo empírico de maneira geral;

7. o experimentador/pesquisador ministra o treinamento sobre a técnica *SMartyCheck* para um grupo de participantes, permitindo aos mesmos esclarecem dúvidas sobre a mesma;

8. após a realização dos treinamentos, o experimentador/pesquisador envia a instrumentação para cada participante, a qual pode ser utilizada para consulta ao responder os questionários eletrônicos;

9. o experimentador/pesquisador explica o formato dos formulários (questionários) eletrônicos enviados por email a cada participante, com o objetivo de esclarecer possíveis dúvidas de preenchimento e quanto ao envio dos mesmos;

10. o experimentador/pesquisador define um prazo de 3 dias, combinado com todos os participantes do estudo para o preenchimento e envio das respostas. O tempo para envio das respostas foi controlado por meio do software *LimeSurvey*; e

11. o participante responde e envia os formulários eletrônicos.

As tarefas foram realizadas em igual número pelos participantes. Portanto, a Tabela 5.1 apresenta as informações sobre cada participante deste estudo, os quais foram ordenados (de forma crescente) de acordo com sua formação acadêmica. Os participantes foram classificados com base na escala definida na Seção 5.1:

- **Formação Acadêmica**: Graduando(a) (Gn), Graduado(a) (Gr), Mestrando(a) (Mn), Mestre (Ms), Doutorando(a) (Dn), Doutor(a) (Dr);

- **Experiência com UML**: Nenhuma (1), Pouca (2), Básica (3), Moderada (4), Avançada (5);

- **Experiência com LPS e Variabilidade**: Nenhuma (1), Pouca (2), Básica (3), Moderada (4), Avançada (5); e

- **Experiência com Inspeção de Software**: Nenhuma (1), Pouca (2), Básica (3), Moderada (4), Avançada (5).

Tabela 5.1: Dados de Caracterização Detalhados dos Participantes do Estudo.

Id do Participante	**Formação Acadêmica**	**Experiência com UML**	**Experiência com LPS**	**Experiência com Inspeção de Software**
Id 6 (*SMartyCheck*)	Gn(X) Gr() Mn() Ms() Dn() Dr()	3	1	3
Id 7 (*SMartyCheck*)	Gn(X) Gr() Mn() Ms() Dn() Dr()	3	1	3
Id 4 (*SMartyCheck*)	Gn(X) Gr() Mn() Ms() Dn() Dr()	4	4	3
Id 7 (*Ad hoc*)	Gn() Gr(X) Mn() Ms() Dn() Dr()	4	2	1
Id 3 (*SMartyCheck*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	3	2	2
Id 6 (*Ad hoc*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	3	2	2
Id 4 (*Ad hoc*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	3	2	3
Id 2 (*Ad hoc*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	4	2	2
Id 5 (*Ad hoc*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	4	2	2
Id 5 (*SMartyCheck*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	4	4	2
Id 1 (*Ad hoc*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	5	4	2
Id 1 (*SMartyCheck*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	5	4	2
Id 3 (*Ad hoc*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	5	4	3
Id 2 (*SMartyCheck*)	Gn() Gr() Mn(X) Ms() Dn() Dr()	5	5	2

Pode-se observar que os participantes possuem em geral um conhecimento moderado em UML, básico em LPS e pouco em inspeção de software.

De acordo com dados coletados das inspeções realizadas por cada participante, os mesmos são analisados e interpretados na próxima seção como resultados obtidos de tal estudo.

5.5 Análise e Interpretação dos Resultados

As análises e interpretações feitas de acordo com os dados coletados em conjunto com a aplicação dos testes estatísticos são apresentadas nas próximas seções.

5.5.1 Análise e Interpretação dos Resultados para Diagramas de Casos de Uso

Os resultados obtidos para diagramas de casos de uso de acordo com a aplicação da técnica *SMartyCheck* e da técnica *Ad hoc* para a LPS *Arcade Game Maker* (AGM) foram analisados com base nas seguintes etapas:

- análise e interpretação dos dados coletados para as técnicas *SMartyCheck* e *Ad hoc*, apresentados na Tabela 5.2 e na Tabela 5.3, por meio do teste de normalidade Shapiro-Wilk, do teste T (para amostras normais) e do teste Mean-Whitney Wilcoxon (para amostras não normais); e

- análise e interpretação da correlação entre a eficiência, eficácia e efetividade das técnicas (*SMartyCheck* e *Ad hoc*) e as respostas fornecidas pelos participantes utilizando o Questionário de Caracterização, por meio da aplicação do teste de correlação de Spearman.

Para as técnicas *SMartyCheck* e *Ad hoc* foram feitos cálculos de eficiência, eficácia e efetividade para diagramas de casos de uso utilizando a LPS AGM. Para tanto, cada participante recebeu um *checklist* com diagramas de casos de uso para serem inspecionados (Apêndice E). Assim, os defeitos detectados corretamente e os defeitos detectados incorretamente foram coletados de cada participante, aplicados diretamente para o cálculo de efetividade (Equação 5.3), bem como para a realização dos cálculos de eficácia e eficiência.

Tais diagramas de casos de uso inspecionados por cada participante têm uma quantidade total de defeitos existentes que podem ou não ser detectados corretamente, os quais foram utilizados para calcular a eficácia (Equação 5.2) das inspeções. Para o cálculo relacionado a eficiência (Equação 5.1) das inspeções, o tempo total de inspeção foi calculado de acordo com as inspeções realizadas por cada participante.

Buscou-se equilibrar o mesmo nível de complexidade dos diagramas de casos de uso, distribuídos em 6 formulários diferentes de maneira aleatória para os participantes. Para tanto, uma quantidade diferente de defeitos foram mutados artificialmente em 3 diagramas de casos de uso distintos para evitar o viés deste estudo. Portanto, 2 diagramas de casos de uso têm 9 defeitos e 1 diagrama de casos de uso possui 6 defeitos.

A Tabela 5.2 e Tabela 5.3 apresentam os resultados obtidos de acordo com as inspeções para diagramas de casos de uso utilizando as técnicas *SMartyCheck* e *Ad hoc*.

Tabela 5.2: Resultados Obtidos e Estatística Descritiva da Técnica *SMartyCheck* para Diagrama de Casos de Uso.

Técnica *SMartyCheck* (Inspeção LPS AGM) - Diagrama de Casos de Uso								
Id do Participante	Total de Defeitos Existentes	Tempo Total de Inspeção (min:seg)	Eficiência	Eficácia	Defeitos Detectados Corretamente	Defeitos Existentes Não Detectados	Defeitos Detectados Incorretamente	Efetividade
1	6	17:81	0,34	100%	6	0	2	4
2	9	32:15	0,28	100%	9	0	1	8
3	9	12:50	0,72	100%	9	0	1	8
4	6	70:05	0,09	100%	6	0	5	1
5	9	19:16	0,47	100%	9	0	0	9
6	6	28:29	0,18	83%	5	1	4	1
7	9	36:00	0,19	78%	7	2	2	5
Média	7,71	30:85	0,32	94%	7,29	0,43	2,14	5,14
Desvio Padrão	1,48	17:79	0,20	9%	1,58	0,73	1,64	3,09
Mediana	9,00	28:29	0,28	100%	7,00	0,00	2,00	5,00

Tabela 5.3: Resultados Obtidos e Estatística Descritiva da Técnica *Ad hoc* para Diagrama de Casos de Uso.

Técnica *Ad hoc* (Inspeção LPS AGM) - Diagrama de Casos de Uso								
Id do Participante	Total de Defeitos Existentes	Tempo Total de Inspeção (min:seg)	Eficiência	Eficácia	Defeitos Detectados Corretamente	Defeitos Existentes Não Detectados	Defeitos Detectados Incorretamente	Efetividade
1	9	19:22	0,26	56%	5	4	4	1
2	6	30:08	0,13	67%	4	2	2	2
3	9	17:17	0,23	44%	4	5	5	-1
4	9	12:01	0,17	22%	2	7	7	-5
5	6	34:49	0,14	83%	5	1	1	4
6	9	16:31	0,18	33%	3	6	6	-3
7	9	133:06	0,05	67%	6	3	3	3
Média	8,14	37:48	0,17	53%	4,14	4,00	4,00	0,14
Desvio Padrão	1,36	39:72	0,06	20%	1,25	2,00	2,00	3,04
Mediana	9,00	19:22	0,17	56%	4,00	4,00	4,00	1,00

Analisando os resultados obtidos na Tabela 5.2 e Tabela 5.3, foi possível observar que a eficiência, eficácia e efetividade das inspeções realizadas por meio da técnica *SMartyCheck* é, em média, maior em comparação com a técnica *Ad hoc*. Com isso, visando comparar as técnicas para diagramas de casos de uso, a Figura 5.3 apresenta os comparativos por meio dos *Box Plots* com os respectivos valores de eficiência, eficácia e efetividade (Tabela 5.2 e Tabela 5.3) para diagramas de casos de uso de cada uma das técnicas (*SMartyCheck* e *Ad hoc*).

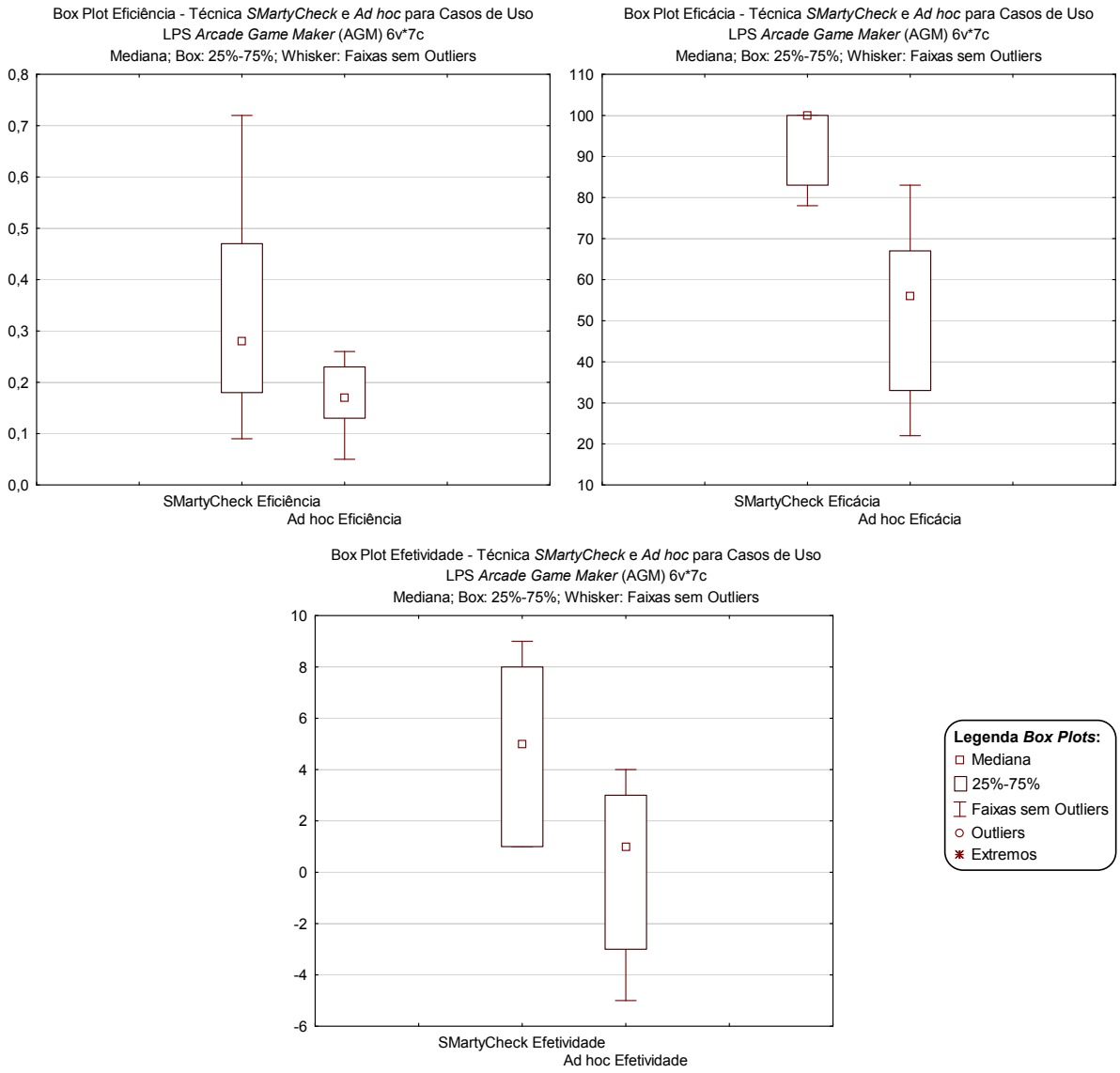


Figura 5.3: *Box Plots* Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Casos de Uso.

Além disso, os itens do *checklist* para diagrama de casos de uso da técnica *SMartyCheck* e/ou os tipos de defeitos da técnica *Ad hoc* foram analisados de forma individual com base nas inspeções realizadas pelos participates em tal estudo. Assim, os itens e/ou tipos de defeitos para diagrama de casos de uso com maior eficiência, eficácia e efetividade são apresentados a seguir:

- Técnica *SMartyCheck* - Itens do *checklist* para inspeção de diagrama de casos de uso:

- itens eficazes e efetivos: Desvio Intencional (DI.1), Informação Estranha (IE.1) e Omissão (Om.1); e
- itens eficientes: Desvio Intencional (DI.1), Informação Estranha (IE.1) e Omissão (Om.1).

- Técnica *Ad hoc* - Tipos de defeitos para inspeção de diagrama de casos de uso:
  - tipos de defeitos eficazes e efetivos: Desvio Intencional (DI.1) e Informação Estranha (IE.2); e
  - tipos de defeitos eficientes: Inconsistência (Incons.1), Desvio Intencional (DI.1) e Fato Incorreto (FI.1 e FI.2).

Eficiência, Eficácia e Efetividade da *SMartyCheck versus* Técnica *Ad hoc* para Casos de Uso

Teste de Normalidade dos Dados - Shapiro-Wilk: De acordo com as hipóteses definidas na Seção 5.3 o teste de normalidade de Shapiro e Wilk (1965) foi aplicado para a eficiência, a eficácia e a efetividade, as quais foram calculadas para os diagramas de casos de uso da LPS *Arcade Game Maker* (AGM) inspecionados pelos participantes, conforme apresentado na Tabela 5.4. Portanto, os resultados obtidos indicam para cada técnica:

Tabela 5.4: Teste de Normalidade Shapiro-Wilk das Amostras da Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Casos de Uso.

Técnica *SMartyCheck* - Diagrama de Casos de Uso			
Shapiro-Wilk	**Normalidade** (α = 0,05)		
	SW-W	W Crítico	*p-value*
Eficiência	0,9190	0,8030	0,4617
Eficácia	0,6401		0,0008
Efetividade	0,8800		0,2255

Técnica *Ad hoc* - Diagrama de Casos de Uso			
Shapiro-Wilk	**Normalidade** (α = 0,05)		
	SW-W	W Crítico	*p-value*
Eficiência	0,9738	0,8030	0,9252
Eficácia	0,9692		0,8933
Efetividade	0,9500		0,7306

- ***Técnica SMartyCheck (N=7):***
  - ***Eficiência*** para a LPS AGM: para a média (μ) 0,32, desvio padrão (σ) 0,20, a eficiência para a aplicação da técnica *SMartyCheck* na LPS AGM resultou em $p = 0{,}4617$, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

    O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância ($\alpha = 0{,}05$), $p = 0{,}4617$ ($0{,}4617 > 0{,}05$) e valor de $W_{calculado} = 0{,}9190$

$> W_{critico} = 0{,}8030$, que a amostra é considerada normal. Logo, a hipótese definida como nula $H_i0 : \mu((Eficiencia(SMartyCheck)) = \mu((Eficiencia(Adhoc))$ é aceita.

No entanto, é possível observar que a média (μ) 0,32 de eficiência obtida por meio da *SMartyCheck* é maior que a média (μ) 0,17 de eficiência obtida por meio da técnica *Ad hoc*. Portanto, a *SMartyCheck* é mais eficiente que a técnica *Ad hoc* para diagramas de casos de uso. Logo, a hipótese definida como alternativa $H_i1 : \mu((Eficiencia(SMartyCheck)) > \mu((Eficiencia(Adhoc))$ é aceita.

- ***Eficácia*** para a LPS AGM: para a média (μ) 0,94, desvio padrão (σ) 0,09, a eficácia para a aplicação da técnica *SMartyCheck* na LPS AGM resultou em p = 0,0008, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,0008 (0,0008 < 0,05) e valor de $W_{calculado}$ = 0,6401 < $W_{critico}$ = 0,8030, que a amostra é considerada não normal. Assim, a hipótese definida como nula $H_a0 : \mu((Eficacia(SMartyCheck)) = \mu((Eficacia(Adhoc))$ não é aceita, permitindo que a hipótese definida como alternativa $H_a1 : \mu((Eficacia(SMartyCheck)) > \mu((Eficacia(Adhoc))$ seja aceita. Portanto, a *SMartyCheck* possui uma eficácia maior, em média, que a técnica *Ad hoc* para diagramas de casos de uso.

- ***Efetividade*** para a LPS AGM: para a média (μ) 5,14, desvio padrão (σ) 3,09, a efetividade para a aplicação da técnica *SMartyCheck* na LPS AGM resultou em p = 0,2255, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,2255 (0,2255 > 0,05) e valor de $W_{calculado}$ = 0,8800 > $W_{critico}$ = 0,8030, que a amostra é considerada normal. Logo, a hipótese definida como nula $H_e0 : \mu((Efetividade(SMartyCheck)) = \mu((Efetividade(Adhoc))$ é aceita.

  Contudo, é possível observar que a média (μ) 5,14 de efetividade obtida por meio da *SMartyCheck* é maior que a média (μ) 0,14 de efetividade obtida por meio da técnica *Ad hoc*. Portanto, a *SMartyCheck* é mais efetiva que a técnica *Ad hoc* para diagramas de casos de uso. Logo, a hipótese definida como alternativa $H_e1 : \mu((Efetividade(SMartyCheck)) > \mu((Efetividade(Adhoc))$ é aceita

- ***Técnica Ad hoc (N=7):***

- ***Eficiência*** para a LPS AGM: para a média (μ) 0,17, desvio padrão (σ) 0,06, a eficiência para a aplicação da técnica *Ad hoc* na LPS AGM resultou em p = 0,9252, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,9252 (0,9252 > 0,05) e valor de $W_{calculado}$ = 0,9738 > $W_{critico}$ = 0,8030, que a amostra é considerada normal.

- ***Eficácia*** para a LPS AGM: para a média (μ) 0,53, desvio padrão (σ) 0,20, a eficácia para a aplicação da técnica *Ad hoc* na LPS AGM resultou em p = 0,8933, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,8933 (0,8933 > 0,05) e valor de $W_{calculado}$ = 0,9692 > $W_{critico}$ = 0,8030, que a amostra é considerada normal.

- ***Efetividade*** para a LPS AGM: para a média (μ) 0,14, desvio padrão (σ) 3,04, a efetividade para a aplicação da técnica *Ad hoc* na LPS AGM resultou em p = 0,7306, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,7306 (0,7306 > 0,05) e valor de $W_{calculado}$ = 0,9500 > $W_{critico}$ = 0,8030, que a amostra é considerada normal.

Teste T para Eficiência e Efetividade de *SMartyCheck* e *Ad hoc*: Neste estudo a maioria das amostras são independentes e foram identificadas como normais para a verificação das hipóteses por meio do teste T (paramétrico) (Wohlin *et al.*, 2012). Entretanto, apenas a amostra referente a eficácia da técnica *SMartyCheck* foi considerada não normal após a aplicação do teste de Shapiro-Wilk, sendo assim, a hipótese da mesma é verificada por meio do teste de Mann-Whitney Wilcoxon (não paramétrico).

Na sequência, os valores de T foram calculados e analisados de acordo com a eficiência e a efetividade de cada técnica (*SMartyCheck* e *Ad hoc*) para diagramas de casos de uso:

- ***Teste T - Eficiência***. De acordo com a média da amostra da técnica *SMartyCheck* ($\mu = 0,32$) e da técnica *Ad hoc* ($\mu = 0,17$), bem como seus respectivos desvios padrões ($\sigma = 0,20$ e $\sigma = 0,06$), e o tamanho da amostra ($N = 7$) igual a 7 para cada uma das amostras, foi obtido o grau de liberdade de $df = 12$ e o valor de $T_{calculado}$ = 1,87.

  Portanto, o grau de liberdade $df = 12$ para $N = 7$, foi combinado com o valor de $T_{calculado}$ = 1,87, indicando o valor $T_{critico}$ que deve ser selecionado na tabela t com o intuito de realizar o teste de hipóteses. Logo, o valor $T_{critico}$ = 2,17, de

acordo com a tabela *t*, com o nível de significância ($\alpha = 0,05$). Assim, o valor $T_{calculado} = 1{,}87 < T_{critico} = 2{,}17$, causando a rejeição da hipótese nula $H_i0 : \mu((Eficiencia(SMartyCheck)) = \mu((Eficiencia(Adhoc))$ do estudo, e aceitando a hipótese alternativa $H_i1 : \mu((Eficiencia(SMartyCheck)) > \mu((Eficiencia(Adhoc))$, que evidencia maior eficiência para técnica *SMartyCheck*.

- ***Teste T - Efetividade***. De acordo com a média da amostra da técnica *SMartyCheck* ($\mu = 5,14$) e da técnica *Ad hoc* ($\mu = 0,14$), bem como seus respectivos desvios padrões ($\sigma = 3,09$ e $\sigma = 3,04$), e o tamanho da amostra ($N = 7$), foi obtido o grau de liberdade de $df = 12$ e o valor de $T_{calculado} = 2{,}82$.

  Portanto, o grau de liberdade $df = 12$ para $N = 7$, foi combinado com o valor de $T_{calculado} = 2{,}82$, indicando o valor $T_{critico}$ que deve ser selecionado na tabela *t* com o intuito de realizar o teste de hipóteses. Logo, o valor $T_{critico} = 2{,}17$, de acordo com a tabela *t*, com o nível de significância ($\alpha = 0,05$). Assim, o valor $T_{calculado} = 2{,}82 > T_{critico} = 2{,}17$, causando a rejeição da hipótese nula $H_e0 : \mu((Efetividade(SMartyCheck)) = \mu((Efetividade(Adhoc))$ do estudo, e aceitando a hipótese alternativa $H_e1 : \mu((Efetividade(SMartyCheck)) > \mu((Efetividade(Adhoc))$, que evidencia maior efetividade para técnica *SMartyCheck*.

Teste Mann-Whitney Wilcoxon para Eficácia de *SMartyCheck* e *Ad hoc*: Como mencionado, apenas a amostra referente a eficácia da técnica *SMartyCheck* foi considerada não normal após a aplicação do teste de Shapiro-Wilk. Assim, o teste Mann-Whitney Wilcoxon (não paramétrico) (Hole, 2011; Juristo e Moreno, 2010) pode ser aplicado em amostras independentes como é o caso das amostras das técnicas *SMartyCheck* e *Ad hoc*, na qual a primeira foi identificada como não normal e a segunda normal, sendo esta última, como alternativa para o teste T (paramétrico).

Logo, o teste foi calculado de acordo com os pesos (valores) da Tabela 5.5, com o valor obtido para $p = 0{,}00330$ em comparação com o nível de significância ($\alpha = 0,05$). Portanto, o valor obtido $p = 0{,}00330 < \alpha = 0,05$, causando a rejeição da hipótese $H_a0 : \mu((Eficacia(SMartyCheck)) = \mu((Eficacia(Adhoc))$ do estudo, e aceitando a hipótese alternativa $H_a1 : \mu((Eficacia(SMartyCheck)) > \mu((Eficacia(Adhoc))$, que evidencia maior eficácia para técnica *SMartyCheck* para diagramas de casos de uso.

Tabela 5.5: Ranque Mann-Whitney Wilcoxon - Diagrama de Casos de Uso.

Ranque Mann-Whitney Wilcoxon				
Amostra para Diagrama de Casos de Uso				
SMartyCheck (N = 7)			*Ad hoc* (N = 7)	
Id	**Amostra Eficácia**		***SMartyCheck***	***Ad hoc***
1	22%	*Ad hoc*		1
2	33%	*Ad hoc*		2
3	44%	*Ad hoc*		3
4	56%	*Ad hoc*		4
5	67%	*Ad hoc*		5,5
6	67%	*Ad hoc*		5,5
7	78%	*SMartyCheck*	7	
8	83%	*SMartyCheck*	8,5	
9	83%	*Ad hoc*		8,5
10	100%	*SMartyCheck*	12	
11	100%	*SMartyCheck*	12	
12	100%	*SMartyCheck*	12	
13	100%	*SMartyCheck*	12	
14	100%	*SMartyCheck*	12	
Total			75,5	29,5
U(Técnica)			1,5	47,5
U(MIN(SMartyCheck,Ad hoc)			1,5	
p-value			0,00330	

Além disso, foi possível observar a diferença estatística dos pesos atribuídos (Tabela 5.5) para os valores de eficácia entre as técnicas (*SMartyCheck* e *Ad hoc*), rejeitando a hipótese nula H_a0 e aceitando a hipótese alternativa H_a1.

Correlação entre a Eficiência, Eficácia e a Efetividade e o Nível de Conhecimento dos Participantes para Casos de Uso

A correlação entre o nível de conhecimento dos participantes sobre UML, LPS e inspeção de software e a eficiência, eficácia e efetividade do diagrama de casos de uso é realizada, no intuito de verificar a possível influência do nível de conhecimento dos participantes em comparação com os resultados obtidos.

A Tabela 5.6 e a Tabela 5.7 apresentam o nível de conhecimento em UML, LPS e inspeção de software dos participantes. Assim, foi possível observar que os participantes possuem, em média, um nível de conhecimento básico para moderado em UML, básico em LPS e pouco para básico em inspeção de software.

Tabela 5.6: Experiência e Nível de Conhecimento dos Participantes da Técnica *SMarty-Check*.

Técnica *SMartyCheck*				
Experiência e Nível de Conhecimento				
Id do Participante	Experiência na Área (Anos)	Experiência com UML (1 a 5)	Experiência com LPS (1 a 5)	Experiência com Inspeção de Software (1 a 5)
1	8	5	4	2
2	1	5	5	2
3	6	3	2	2
4	2	4	4	3
5	0,8	4	4	2
6	0,15	3	1	3
7	2	3	1	3
Média	2,85	3,86	3,00	2,43
Desvio Padrão	2,75	0,83	1,51	0,49
Mediana	2,00	4,00	4,00	2,00

Tabela 5.7: Experiência e Nível de Conhecimento dos Participantes da Técnica *Ad hoc*.

Técnica *Ad hoc*				
Experiência e Nível de Conhecimento				
Id do Participante	Experiência na Área (Anos)	Experiência com UML (1 a 5)	Experiência com LPS (1 a 5)	Experiência com Inspeção de Software (1 a 5)
1	2	5	4	2
2	0,9	4	2	2
3	4	5	4	3
4	0,3	3	2	3
5	2	4	2	2
6	2	3	2	2
7	6	4	2	1
Média	2,46	4,00	2,57	2,14
Desvio Padrão	1,80	0,76	0,90	0,64
Mediana	2,00	4,00	2,00	2,00

É importante mencionar que tanto os participantes mais experientes, quanto os participantes menos experientes possuem, em média, o mesmo nível de conhecimento para a utilização da técnica *SMartyCheck* quanto para a técnica *Ad hoc*.

Os participantes considerados mais experientes não tiveram um nível de detecção de defeitos nas inspeções maior ou menor em comparação com os participantes menos experientes. Portanto, ambos permaneceram equiparados com relação ao nível de detecção de defeitos de acordo com os resultados indicados de tal estudo.

Em seguida, com base na escala de Likert (1932), a Tabela 5.8 apresenta os dados aplicados no teste não paramétrico da correlação de Spearman (1987) para verificar se existe correlação entre os valores de eficiência, eficácia e efetividade das técnicas

SMartyCheck e *Ad hoc* de acordo com o nível de conhecimento dos participantes de tal estudo.

Tabela 5.8: Correlação de Spearman entre a Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* e o Nível de Conhecimento de UML, LPS e Inspeção de Software - Diagrama de Casos de Uso.

Técnica *SMartyCheck* - Diagrama de Casos de Uso (*Corr.1*)			
Correlação de Spearman - Nível de Conhecimento			
-	UML	LPS	Inspeção de Software
Eficiência	0,125	0,259	-0,625
Eficácia	0,732	0,830	-0,196
Efetividade	0,196	0,268	-0,482

Técnica *Ad hoc* - Diagrama de Casos de Uso (*Corr.2*)			
Correlação de Spearman - Nível de Conhecimento			
-	UML	LPS	Inspeção de Software
Eficiência	0,411	0,813	0,580
Eficácia	0,420	0,071	-0,455
Efetividade	0,411	0,063	-0,509

Analisando os resultados obtidos (Tabela 5.8) por meio da escala de correção de Spearman (Figura 5.4), as seguintes análises foram realizadas:

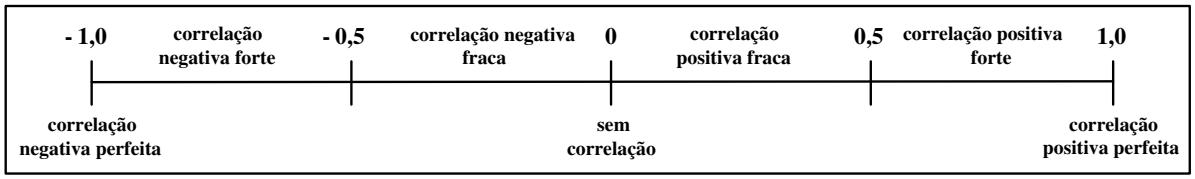


Figura 5.4: Escala de Correlação de Spearman (Spearman, 1987).

- Correlação técnica *SMartyCheck* ***(Corr.1)***:

  - ***Eficiência***. Evidências são fornecidas de o que o nível de conhecimento em UML (positiva fraca) e LPS (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *SMartyCheck* para a inspeção de diagramas de casos de uso. Ao contrário, o nível de conhecimento em inspeção de software (negativa forte) dos participantes fornece indícios da não influencia na aplicação da técnica *SMartyCheck*.

  - ***Eficácia***. Indícios são fornecidos de o que o nível de conhecimento em UML (positiva forte) e LPS (positiva forte) dos participantes influenciam na aplicação da técnica *SMartyCheck*. Contudo, o nível de conhecimento em inspeção de software (negativa fraca) dos participantes fornece evidências da não influencia na aplicação da técnica *SMartyCheck*.

  - ***Efetividade***. Evidências são fornecidas de o que o nível de conhecimento em UML (positiva fraca) e LPS (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *SMartyCheck*. Em contrapartida, o nível

de conhecimento em inspeção de software (negativa fraca) dos participantes fornece indícios da não influencia na aplicação da técnica *SMartyCheck*.

- Correlação técnica *Ad hoc* ***(Corr.2)***:

  - ***Eficiência***. Evidências são fornecidas de o que o nível de conhecimento em UML (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *Ad hoc* para a inspeção de diagramas de casos de uso. Ao contrário, o nível de conhecimento em LPS (positiva forte) e inspeção de software (positiva forte) dos participantes fornecem indícios de uma forte influencia na aplicação da técnica *Ad hoc*.
  - ***Eficácia***. Indícios são fornecidos de o que o nível de conhecimento em UML (positiva fraca) e LPS (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *Ad hoc*. Em contrapartida, o nível de conhecimento em inspeção de software (negativa fraca) dos participantes fornece evidências da não influencia na aplicação da técnica *Ad hoc*.
  - ***Efetividade***. Evidências são fornecidas de o que o nível de conhecimento em UML (positiva fraca) e LPS (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *Ad hoc*. Contudo, o nível de conhecimento em inspeção de software (negativa forte) dos participantes fornece indícios da não influencia na aplicação da técnica *Ad hoc*.

5.5.2 Análise e Interpretação dos Resultados para Diagramas de Classes

Os resultados obtidos para diagramas de classes de acordo com a aplicação da técnica *SMartyCheck* e da técnica *Ad hoc* para a LPS *Arcade Game Maker* (AGM) foram analisados com base nas seguintes etapas:

- análise e interpretação dos dados coletados para as técnicas *SMartyCheck* e *Ad hoc*, apresentados na Tabela 5.9 e na Tabela 5.10, por meio do teste de normalidade Shapiro-Wilk, do teste T (para amostras normais) e do teste Mean-Whitney Wilcoxon (para amostras não normais); e

- análise e interpretação da correlação entre a eficiência, eficácia e efetividade das técnicas (*SMartyCheck* e *Ad hoc*) e as respostas fornecidas pelos participantes utilizando o Questionário de Caracterização, por meio da aplicação do teste de correlação de Spearman.

Para as técnicas *SMartyCheck* e *Ad hoc* foram feitos cálculos de eficiência, eficácia e efetividade para diagramas de classes utilizando a LPS AGM. Para tanto, cada participante recebeu um *checklist* com diagramas de classes para serem inspecionados (Apêndice E). Assim, os defeitos detectados corretamente e os defeitos detectados incorretamente foram coletados de cada participante, aplicados diretamente para o cálculo de efetividade (Equação 5.3), bem como para a realização dos cálculos de eficácia e eficiência.

Tais diagramas de classes inspecionados por cada participante têm uma quantidade total de defeitos existentes que podem ou não ser detectados corretamente, os quais foram utilizados para calcular a eficácia (Equação 5.2) das inspeções. Para o cálculo relacionado a eficiência (Equação 5.1) das inspeções, o tempo total de inspeção foi calculado de acordo com as inspeções realizadas por cada participante.

Buscou-se equilibrar o mesmo nível de complexidade dos diagramas de classes, distribuídos em 6 formulários diferentes de maneira aleatória para os participantes. Para tanto, uma quantidade diferente de defeitos foram mutados artificialmente em 3 diagramas de classes distintos para evitar o viés deste estudo. Portanto, 1 diagrama de classes possui 7 defeitos, 1 diagrama de classes contém 9 defeitos e 1 diagrama de classes tem 6 defeitos.

A Tabela 5.9 e Tabela 5.10 apresentam os resultados obtidos de acordo com as inspeções para diagramas de classes utilizando as técnicas *SMartyCheck* e *Ad hoc*.

Tabela 5.9: Resultados Obtidos e Estatística Descritiva da Técnica *SMartyCheck* para Diagrama de Classes.

Técnica *SMartyCheck* (Inspeção LPS AGM) - Diagrama de Classes								
Id do Participante	Total de Defeitos Existentes	Tempo Total de Inspeção (min:seg)	Eficiência	Eficácia	Defeitos Detectados Corretamente	Defeitos Existentes Não Detectados	Defeitos Detectados Incorretamente	Efetividade
1	6	11:45	0,52	100%	6	0	1	5
2	7	18:81	0,37	100%	7	0	0	7
3	9	18:75	0,48	100%	9	0	2	7
4	6	57:00	0,09	83%	5	1	4	1
5	7	15:10	0,46	100%	7	0	0	7
6	6	23:34	0,13	50%	3	3	6	-3
7	7	46:00	0,13	86%	6	1	4	2
Média	6,86	27:21	0,31	88%	6,14	0,71	2,43	3,71
Desvio Padrão	0,99	16:00	0,17	17%	1,73	1,03	2,13	3,57
Mediana	7,00	18:81	0,37	100%	6,00	0,00	2,00	5,00

Tabela 5.10: Resultados Obtidos e Estatística Descritiva da Técnica *Ad hoc* para Diagrama de Classes.

Técnica *Ad hoc* (Inspeção LPS AGM) - Diagrama de Classes								
Id do Participante	Total de Defeitos Existentes	Tempo Total de Inspeção (min:seg)	Eficiência	Eficácia	Defeitos Detectados Corretamente	Defeitos Existentes Não Detectados	Defeitos Detectados Incorretamente	Efetividade
1	7	10:56	0,38	57%	4	3	3	1
2	6	58:18	0,05	50%	3	3	3	0
3	7	17:51	0,17	43%	3	4	4	-1
4	9	9:59	0,42	44%	4	5	5	-1
5	6	6:06	0,66	67%	4	2	2	2
6	6	3:34	0,30	17%	1	5	6	-5
7	9	96:48	0,08	89%	8	1	1	7
Média	7,14	28:82	0,29	52%	3,86	3,29	3,43	0,43
Desvio Padrão	1,25	32:60	0,20	21%	1,96	1,39	1,59	3,37
Mediana	7,00	10:56	0,30	50%	4,00	3,00	3,00	0,00

Analisando os resultados obtidos na Tabela 5.9 e Tabela 5.10, foi possível observar que a eficiência, eficácia e efetividade das inspeções realizadas por meio da técnica *SMartyCheck* é, em média, maior em comparação com a técnica *Ad hoc*. Com isso, visando comparar as técnicas para diagramas de classes, a Figura 5.5 apresenta os comparativos por meio dos *Box Plots* com os respectivos valores de eficiência, eficácia e efetividade (Tabela 5.9 e Tabela 5.10) para diagramas de classes de cada uma das técnicas (*SMartyCheck* e *Ad hoc*).

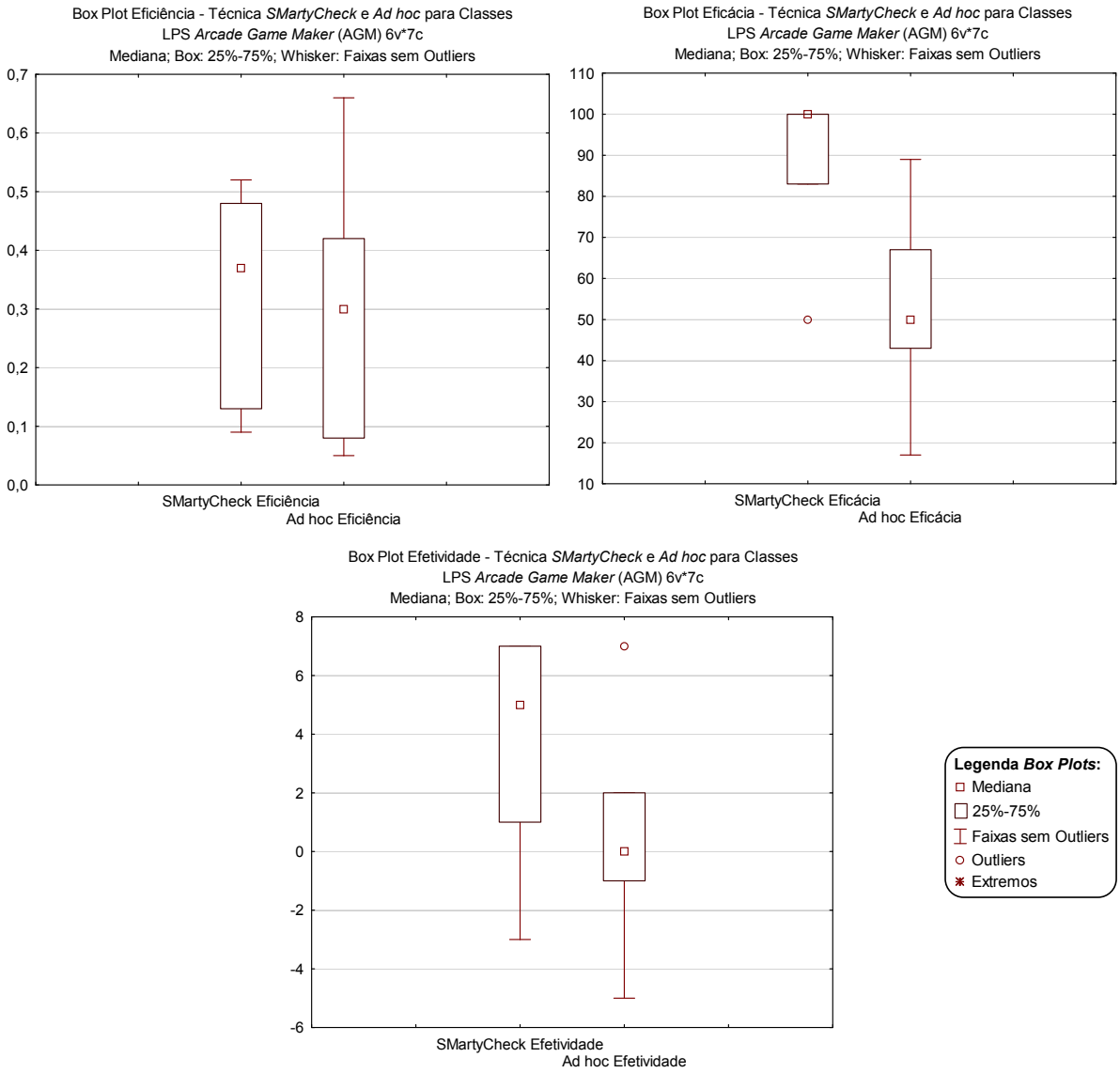


Figura 5.5: *Box Plots* Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Classes.

Além disso, os itens do *checklist* para diagrama de classes da técnica *SMartyCheck* e/ou os tipos de defeitos da técnica *Ad hoc* foram analisados de forma individual com base nas inspeções realizadas em tal estudo. Assim, os itens e/ou tipos de defeitos para diagrama de classes com maior eficiência, eficácia e efetividade são apresentados a seguir:

- Técnica *SMartyCheck* - Itens do *checklist* para inspeção de diagrama de classes:
  - itens eficazes e efetivos: Informação Estranha (IE.1) e Omissão (Om.1); e
  - itens eficientes: Informação Estranha (IE.1 e IE.2) e Omissão (Om.1).

- Técnica *Ad hoc* - Tipos de defeitos para inspeção de diagrama de classes:
  - tipos de defeitos eficazes e efetivos: Fato Incorreto (FI.1), Informação Estranha (IE.2) e Omissão (Om.1); e
  - tipos de defeitos eficientes: Inconsistência (Incons.1), Fato Incorreto (FI.1), Imodificável (Imo.1) e Omissão (Om.1).

Eficiência, Eficácia e Efetividade da *SMartyCheck versus* Técnica *Ad hoc* para Classes

Teste de Normalidade dos Dados - Shapiro-Wilk: De acordo com as hipóteses definidas na Seção 5.3 o teste de normalidade de Shapiro e Wilk (1965) foi aplicado para a eficiência, a eficácia e a efetividade, as quais foram calculadas para os diagramas de classes da LPS *Arcade Game Maker* (AGM) inspecionados pelos participantes, conforme apresentado na Tabela 5.11. Portanto, os resultados obtidos indicam para cada técnica:

Tabela 5.11: Teste de Normalidade Shapiro-Wilk das Amostras da Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* - Diagrama de Classes.

Técnica *SMartyCheck* - Diagrama de Classes			
Shapiro-Wilk	**Normalidade** (α = 0,05)		
	SW-W	W Crítico	*p-value*
Eficiência	0,8332	0,8030	0,0848
Eficácia	0,7157		0,0053
Efetividade	0,8571		0,1414

Técnica *Ad hoc* - Diagrama de Classes			
Shapiro-Wilk	**Normalidade** (α = 0,05)		
	SW-W	W Crítico	*p-value*
Eficiência	0,9459	0,8030	0,6928
Eficácia	0,9736		0,9243
Efetividade	0,9427		0,6636

- ***Técnica SMartyCheck (N=7):***
  - ***Eficiência*** para a LPS AGM: para a média (μ) 0,31, desvio padrão (σ) 0,17, a eficiência para a aplicação da técnica *SMartyCheck* na LPS AGM resultou em $p = 0{,}0848$, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

    O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância ($\alpha = 0{,}05$), $p = 0{,}0848$ ($0{,}0848 > 0{,}05$) e valor de $W_{calculado} = 0{,}8332 > W_{critico} = 0{,}8030$, que a amostra é considerada normal. Logo, a hipótese definida como nula $H_i0 : \mu((Eficiencia(SMartyCheck)) = \mu((Eficiencia(Adhoc))$ é aceita.

    Contudo, é possível observar que a média (μ) 0,31 de eficiência obtida por meio da *SMartyCheck* é maior que a média (μ) 0,29 de eficiência obtida por meio da técnica *Ad hoc*. Portanto, a *SMartyCheck* é mais eficiente que a técnica

Ad hoc para diagramas de classes. Logo, a hipótese definida como alternativa $H_i1 : \mu((Eficiencia(SMartyCheck)) > \mu((Eficiencia(Adhoc))$ é aceita.

- ***Eficácia*** para a LPS AGM: para a média (μ) 0,88, desvio padrão (σ) 0,17, a eficácia para a aplicação da técnica *SMartyCheck* na LPS AGM resultou em p = 0,0053, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,0053 (0,0053 < 0,05) e valor de $W_{calculado}$ = 0,7157 < $W_{critico}$ = 0,8030, que a amostra é considerada não normal. Assim, a hipótese definida como nula $H_a0 : \mu((Eficacia(SMartyCheck)) = \mu((Eficacia(Adhoc))$ não é aceita, permitindo que a hipótese definida como alternativa $H_a1 : \mu((Eficacia(SMartyCheck)) > \mu((Eficacia(Adhoc))$ seja aceita. Portanto, a *SMartyCheck* possui uma eficácia maior, em média, que a técnica *Ad hoc* para diagramas de classes.

- ***Efetividade*** para a LPS AGM: para a média (μ) 3,71, desvio padrão (σ) 3,57, a efetividade para a aplicação da técnica *SMartyCheck* na LPS AGM resultou em p = 0,1414, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,1414 (0,1414 > 0,05) e valor de $W_{calculado}$ = 0,8571 > $W_{critico}$ = 0,8030, que a amostra é considerada normal. Logo, a hipótese definida como nula $H_e0 : \mu((Efetividade(SMartyCheck)) = \mu((Efetividade(Adhoc))$ é aceita.

  No entanto, é possível observar que a média (μ) 3,71 de efetividade obtida por meio da *SMartyCheck* é maior que a média (μ) 0,43 de efetividade obtida por meio da técnica *Ad hoc*. Portanto, a *SMartyCheck* é mais efetiva que a técnica *Ad hoc* para diagramas de classes. Logo, a hipótese definida como alternativa $H_e1 : \mu((Efetividade(SMartyCheck)) > \mu((Efetividade(Adhoc))$ é aceita.

- ***Técnica Ad hoc (N=7):***

  - ***Eficiência*** para a LPS AGM: para a média (μ) 0,29, desvio padrão (σ) 0,20, a eficiência para a aplicação da técnica *Ad hoc* na LPS AGM resultou em p = 0,6928, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

    O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,6928 (0,6928 > 0,05) e valor de $W_{calculado}$ = 0,9459 > $W_{critico}$ = 0,8030, que a amostra é considerada normal.

- ***Eficácia*** para a LPS AGM: para a média (μ) 0,52, desvio padrão (σ) 0,21, a eficácia para a aplicação da técnica *Ad hoc* na LPS AGM resultou em p = 0,9243, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,9243 (0,9243 > 0,05) e valor de $W_{calculado}$ = 0,9736 > $W_{critico}$ = 0,8030, que a amostra é considerada normal.

- ***Efetividade*** para a média (μ) 0,43, desvio padrão (σ) 3,37, a efetividade para a aplicação da técnica *Ad hoc* na LPS AGM resultou em p = 0,6636, ao ser aplicado ao teste de normalidade Shapiro-Wilk.

  O teste indicou, para uma amostra (N) de tamanho 7 com 95% de nível de significância (α = 0,05), p = 0,6636 (0,6636 > 0,05) e valor de $W_{calculado}$ = 0,9427 > $W_{critico}$ = 0,8030, que a amostra é considerada normal.

Teste T para Eficiência e Efetividade de *SMartyCheck* e *Ad hoc*: Neste estudo a maioria das amostras são independentes e foram identificadas como normais para a verificação das hipóteses por meio do teste T (paramétrico) (Wohlin *et al.*, 2012). Entretanto, apenas a amostra referente a eficácia da técnica *SMartyCheck* foi considerada não normal após a aplicação do teste de Shapiro-Wilk, sendo assim, a hipótese da mesma é verificada por meio do teste de Mann-Whitney Wilcoxon (não paramétrico).

Em seguida, os valores de T foram calculados e analisados de acordo com a eficiência e a efetividade de cada técnica (*SMartyCheck* e *Ad hoc*) para diagramas de classes:

- ***Teste T - Eficiência***. De acordo com a média da amostra da técnica *SMartyCheck* ($\mu = 0,31$) e da técnica *Ad hoc* ($\mu = 0,29$), bem como seus respectivos desvios padrões ($\sigma = 0,17$ e $\sigma = 0,20$), e o tamanho da amostra ($N = 7$) igual a 7 para cada uma das amostras, foi obtido o grau de liberdade de $df = 12$ e o valor de $T_{calculado}$ = 0,15.

  Portanto, o grau de liberdade $df = 12$ para $N = 7$, foi combinado com o valor de $T_{calculado}$ = 0,15, indicando o valor $T_{critico}$ que deve ser selecionado na tabela t com o intuito de realizar o teste de hipóteses. Logo, o valor $T_{critico}$ = 2,17, de acordo com a tabela t, com o nível de significância ($\alpha = 0,05$). Assim, o valor $T_{calculado}$ = 0,15 < $T_{critico}$ = 2,17, causando a rejeição da hipótese nula H_i0 : $\mu((Eficiencia(SMartyCheck)) = \mu((Eficiencia(Adhoc))$ do estudo, e aceitando a hipótese alternativa $H_i1 : \mu((Eficiencia(SMartyCheck)) > \mu((Eficiencia(Adhoc))$, que evidencia maior eficiência para técnica *SMartyCheck*.

- ***Teste T - Efetividade***. De acordo com a média da amostra da técnica *SMartyCheck* ($\mu = 3,71$) e da técnica *Ad hoc* ($\mu = 0,43$), bem como seus respectivos desvios padrões ($\sigma = 3,57$ e $\sigma = 3,37$), e o tamanho da amostra ($N = 7$), foi obtido o grau de liberdade de $df = 12$ e o valor de $T_{calculado} = 1{,}63$.

  Portanto, o grau de liberdade $df = 12$ para $N = 7$, foi combinado com o valor de $T_{calculado} = 1{,}63$, indicando o valor $T_{critico}$ que deve ser selecionado na tabela t com o intuito de realizar o teste de hipóteses. Logo, o valor $T_{critico} = 2{,}17$, de acordo com a tabela t, com o nível de significância ($\alpha = 0,05$). Assim, o valor $T_{calculado} = 1{,}63 < T_{critico} = 2{,}17$, causando a rejeição da hipótese nula $H_e0 : \mu((Efetividade(SMartyCheck)) = \mu((Efetividade(Adhoc))$ do estudo, e aceitando a hipótese alternativa $H_e1 : \mu((Efetividade(SMartyCheck)) > \mu((Efetividade(Adhoc))$, que evidencia maior efetividade para técnica *SMartyCheck*.

Teste Mann-Whitney Wilcoxon para Eficácia de *SMartyCheck* e *Ad hoc*: Como mencionado, apenas a amostra referente a eficácia da técnica *SMartyCheck* foi considerada não normal após a aplicação do teste de Shapiro-Wilk. Assim, o teste Mann-Whitney Wilcoxon (não paramétrico) (Hole, 2011; Juristo e Moreno, 2010) pode ser aplicado em amostras independentes como é o caso das amostras das técnicas *SMartyCheck* e *Ad hoc*, na qual a primeira foi identificada como não normal e a segunda normal, sendo esta última, como alternativa para o teste T (paramétrico).

Logo, o teste foi calculado de acordo com os pesos (valores) da Tabela 5.12, com o valor obtido para $p = 0{,}01520$ em comparação com o nível de significância ($\alpha = 0,05$). Portanto, o valor obtido $p = 0{,}01520 < \alpha = 0,05$, causando a rejeição da hipótese $H_a0 : \mu((Eficacia(SMartyCheck)) = \mu((Eficacia(Adhoc))$ do estudo, e aceitando a hipótese alternativa $H_a1 : \mu((Eficacia(SMartyCheck)) > \mu((Eficacia(Adhoc))$, que evidencia maior eficácia para técnica *SMartyCheck* para diagramas de classes.

Tabela 5.12: Ranque Mann-Whitney Wilcoxon - Diagrama de Classes.

Ranque Mann-Whitney Wilcoxon				
Amostra para Diagrama de Classes				
SMartyCheck (N = 7)			*Ad hoc* (N = 7)	
Id	**Amostra Eficácia**		***SMartyCheck***	***Ad hoc***
1	17%	*Ad hoc*		1
2	43%	*Ad hoc*		2
3	44%	*Ad hoc*		3
4	50%	*Ad hoc*		4,5
5	50%	*SMartyCheck*	4,5	
6	57%	*Ad hoc*		6
7	67%	*Ad hoc*		7
8	83%	*SMartyCheck*	8	
9	86%	*SMartyCheck*	9	
10	89%	*Ad hoc*		10
11	100%	*SMartyCheck*	12,5	
12	100%	*SMartyCheck*	12,5	
13	100%	*SMartyCheck*	12,5	
14	100%	*SMartyCheck*	12,5	
Total			71,5	33,5
U(Técnica)			5,5	43,5
U(MIN(SMartyCheck,Ad hoc)			5,5	
p-value			0,01520	

Além disso, foi possível observar a diferença estatística dos pesos atribuídos (Tabela 5.12) para os valores de eficácia entre as técnicas (*SMartyCheck* e *Ad hoc*), rejeitando a hipótese nula H_a0 e aceitando a hipótese alternativa H_a1.

Correlação entre a Eficiência, Eficácia e a Efetividade e o Nível de Conhecimento dos Participantes para Classes

A correlação entre o nível de conhecimento dos participantes sobre UML, LPS e inspeção de software e a eficiência, eficácia e efetividade do diagrama de classes é realizada, no intuito de verificar a possível influência do nível de conhecimento dos participantes em comparação com os resultados obtidos.

A Tabela 5.6 e a Tabela 5.7 apresentam o nível de conhecimento em UML, LPS e inspeção de software dos participantes. Assim, foi possível observar que os participantes possuem, em média, um nível de conhecimento básico para moderado em UML, básico em LPS e pouco para básico em inspeção de software.

É importante mencionar que tanto os participantes mais experientes, quanto os participantes menos experientes possuem, em média, o mesmo nível de conhecimento para a utilização da técnica *SMartyCheck* quanto para a técnica *Ad hoc*.

Os participantes considerados mais experientes não tiveram um nível de detecção de defeitos nas inspeções maior ou menor em comparação com os participantes menos experientes. Portanto, ambos permaneceram equiparados com relação ao nível de detecção de defeitos de acordo com os resultados indicados de tal estudo.

Em seguida, com base na escala de Likert (1932), a Tabela 5.13 apresenta os dados aplicados no teste não paramétrico da correlação de Spearman (1987) para verificar se existe correlação entre os valores de eficiência, eficácia e efetividade das técnicas *SMartyCheck* e *Ad hoc* de acordo com o nível de conhecimento dos participantes de tal estudo.

Tabela 5.13: Correlação de Spearman entre a Eficiência, Eficácia e Efetividade para as Técnicas *SMartyCheck* e *Ad hoc* e o Nível de Conhecimento de UML, LPS e Inspeção de Software - Diagrama de Classes.

Técnica *SMartyCheck* - Diagrama de Classes (*Corr.1*)			
Correlação de Spearman - Nível de Conhecimento			
-	UML	LPS	Inspeção de Software
Eficiência	0,366	0,464	-0,616
Eficácia	0,580	0,679	-0,536
Efetividade	0,411	0,527	-0,589

Técnica *Ad hoc* - Diagrama de Classes (*Corr.2*)			
Correlação de Spearman - Nível de Conhecimento			
-	UML	LPS	Inspeção de Software
Eficiência	-0,125	0,188	0,384
Eficácia	0,321	0,063	-0,491
Efetividade	0,420	0,134	-0,482

Analisando os resultados obtidos (Tabela 5.13) por meio da escala de correção de Spearman (Figura 5.4), as seguintes análises foram realizadas:

- Correlação técnica *SMartyCheck* ***(Corr.1)***:

  - ***Eficiência***. Evidências são fornecidas de o que o nível de conhecimento em UML (positiva fraca) e LPS (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *SMartyCheck* para a inspeção de diagramas de classes. Ao contrário, o nível de conhecimento em inspeção de software (negativa forte) dos participantes fornece indícios da não influencia na aplicação da técnica *SMartyCheck*.

  - ***Eficácia***. Indícios são fornecidos de o que o nível de conhecimento em UML (positiva forte) e LPS (positiva forte) dos participantes influenciam na aplicação da técnica *SMartyCheck*. Em contrapartida, o nível de conhecimento em inspeção de software (negativa forte) dos participantes fornece evidências da não influencia na aplicação da técnica *SMartyCheck*.

  - ***Efetividade***. Evidências são fornecidas de o que o nível de conhecimento em UML (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *SMartyCheck*. No entanto, o nível de conhecimento em LPS (positiva

forte) dos participantes fornece indícios da influencia na aplicação da técnica *SMartyCheck*. Contudo, o nível de conhecimento em inspeção de software (negativa forte) dos participantes fornece indícios da não influencia na aplicação da técnica *SMartyCheck*.

- Correlação técnica *Ad hoc* ***(Corr.2)***:

  - ***Eficiência***. Evidências são fornecidas de o que o nível de conhecimento em UML (negativa fraca) dos participantes não influenciam na aplicação da técnica *Ad hoc* para a inspeção de diagramas de classes. Ao contrário, o nível de conhecimento em LPS (positiva fraca) e inspeção de software (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *Ad hoc*.
  - ***Eficácia***. Indícios são fornecidos de o que o nível de conhecimento em UML (positiva fraca) e LPS (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *Ad hoc*. Em contrapartida, o nível de conhecimento em inspeção de software (negativa fraca) dos participantes fornece evidências da não influencia na aplicação da técnica *Ad hoc*.
  - ***Efetividade***. Evidências são fornecidas de o que o nível de conhecimento em UML (positiva fraca) e LPS (positiva fraca) dos participantes influenciam pouco na aplicação da técnica *Ad hoc*. Contudo, o nível de conhecimento em inspeção de software (negativa fraca) dos participantes fornece evidências da não influencia na aplicação da técnica *Ad hoc*.

5.5.3 Análise e Interpretação dos Resultados Gerais Obtidos

Os resultados obtidos com o estudo empírico quantitativo realizado são resumidos na Tabela 5.14.

Tabela 5.14: Resumo dos Resultados do Estudo Empírico Quantitativo.

Resultados				
Item	**Estudo Empírico Quantitativo**			
	Casos de Uso		**Classes**	
Técnica	***SMartyCheck***	***Ad hoc***	***SMartyCheck***	***Ad hoc***
Eficiência	0,32	0,17	0,31	0,29
	SMartyCheck foi mais eficiente.		*SMartyCheck* foi mais eficiente.	
Eficácia	0,94	0,53	0,88	0,52
	SMartyCheck foi mais eficaz.		*SMartyCheck* foi mais eficaz.	
Efetividade	5,14	0,14	3,71	0,43
	SMartyCheck foi mais efetiva.		*SMartyCheck* foi mais efetiva.	

Analisando os resultados obtidos na Tabela 5.14 por meio das interpretações realizadas na Seção 5.5, é possível inferir nas seguintes evidências:

- a técnica *SMartyCheck* é mais eficiente, eficaz e efetiva em comparação com a *Ad hoc* para a inspeção de diagramas *SMarty* de casos de uso e de classes, de acordo com as indícios fornecidos neste estudo quantitativo;

- acredita-se que a diferença estatística da eficácia obtida para a técnica *SMartyCheck* para diagramas de casos de uso e de classes, ocorreu devido ao fato de que a técnica *Ad hoc* não possui critérios sistemáticos para a inspeção, sendo assim, não garante a precisão na detecção de defeitos;

- a técnica *Ad hoc* pode ser evidênciada, no cenário de tal estudo, como uma técnica que pode ser eficiente ou efetiva, se e somente se, os inspetores possuírem um alto grau de experiência e conhecimento em diversas áreas (UML, LPS e inspeção de software). No entanto, isso não garante que seja eficiente, eficaz ou efetiva, pois sem a devida condução de novos estudos empíricos não há como obter evidências para tal fato; e

- os testes realizados (Shapiro-Wilk, T e Mann-Whitney Wilcoxon) foram aplicados de acordo com as distribuições das amostras (normal ou não normal), a fim de garantir a rejeição ou aceitação das hipóteses definidas para tal estudo de forma correta.

Com base nos resultados obtidos, tal estudo permitiu obter evidências iniciais relevantes sobre a qualidade da técnica *SMartyCheck*, possibilitando construir um corpo de conhecimento para que pesquisadores possam adotá-la para a realização de novas pesquisas, de novos estudos empíricos, além de colaborar para a evolução da *SMartyCheck*. Além disso, a indústria também pode se beneficiar desta pesquisa, considerando um possível interesse em técnicas de inspeção de software para a detecção de defeitos em diagramas UML de LPS, apoiados por meio da abordagem *SMarty*.

É importante mencionar, que os indícios de eficácia obtidos para a técnica *SMartyCheck* podem estar relativamente relacionados a efetividade da abordagem *SMarty*, a qual possui um perfil UML bem definido, o *SMartyProfile*, além de um processo com diretrizes que guiam o usuário, o *SMartyProcess*. Deste modo, os estereótipos contidos no perfil *SMartyProfile* foram essenciais para o sucesso das inspeções de diagramas *SMarty* de casos de uso e de classes de LPS por meio da utilização da técnica *SMartyCheck*.

5.6 Avaliação de Validade do Estudo

As ameaças consideradas relevantes e identificadas com relação aos impactos para tal estudo são apresentadas nesta seção.

5.6.1 Ameaças à Validade de Conclusão

A principal ameaça para este estudo está relacionada ao número de participantes (14). Este número é relativamente pequeno, mas o conhecimento dos participantes que contribuiram com este estudo foi significativo. Entretanto, pela característica própria do estudo, uma quantidade maior de participantes possibilitaria uma melhor generalização dos resultados.

5.6.2 Ameaças à Validade de *Constructo*

Neste estudo empírico três variáveis dependentes foram definidas e testadas com base no projeto piloto realizado. Portanto, a instrumentação foi adequada, antes de ser aplicada no estudo real. Assim, foi possível verificar se as métricas aplicadas sob estas três variáveis realmente estão de acordo com a avaliação quantitativa proposta em tal estudo. Desse modo, da mesma forma que a avaliação das variáveis dependentes ocorreu, as variáveis independentes também foram avaliadas no projeto piloto, tais como a técnica *SMartyCheck*, a técnica *Ad hoc* e a LPS *Arcade Game Maker* (AGM). Os dados obtidos em tal projeto foram descartados.

Já o nível de conhecimento exigido sobre os conceitos de modelagem UML, LPS e de inspeção de software demonstraram-se satisfatórios para análise das técnicas e, consequente aprimoramento da técnica *SMartyCheck*. Vale ressaltar que a escala (Likert, 1932) utilizada para classificar o nível de conhecimento dos participantes não é completamente precisa, podendo ser uma ameaça quanto a influência entre o nível de conhecimento dos participantes e as técnicas de inspeção utilizadas em tal estudo.

5.6.3 Ameaças à Validade Interna

As seguintes dificuldades foram identificadas:

- **Diferenças entre os participantes**. Como a amostra de tal estudo foi pequena, algumas poucas variações com relação às habilidades dos participantes estavam evidentes. No entanto, os participantes foram divididos dois grupos, nos quais cada grupo recebeu formulários (questionários eletrônicos) específicos para inspeção dos diagramas *SMarty*

utilizando a técnica *SMartyCheck* ou a técnica *Ad hoc*. Logo, as tarefas executadas pelos participantes foram realizadas em igual número.

- **Acurácia das respostas dos participantes**. Uma vez que os treinamentos foram ministrados para cada grupo de participantes, um treinamento sobre a técnica *SMartyCheck* e outro sobre a técnica *Ad hoc*, somando-se ao nível de conhecimento que os participantes possuíam considera-se que a inspeção e a detecção de defeitos dos diagramas *SMarty*, por meio das técnicas *SMartyCheck* e *Ad hoc*, é realmente válida.

- **Efeitos de fadiga**. O treinamento ministrado para cada grupo teve duração de 45 minutos, em média. Após isso, os formulários eletrônicos foram enviados aos participantes para serem respondidos durante um período de até 7 dias. Deste modo, os efeitos de fadiga foram reduzidos, pois os participantes poderiam responder os questionários eletrônicos com certa tranquilidade com relação a adequarem seu tempo, diminuindo a pressão de entrega das respostas.

- **Outros fatores importantes**. A influência entre os participantes foi mitigada, pois os questionários eletrônicos enviados para serem respondidos pelos mesmos foram distribuídos de forma individual e direcionados de maneira aleatória para cada participante. Além disso, é provável que cada participante tenha respondido aos questionários eletrônicos em datas e horários distintos, além de locais diferentes.

5.6.4 Ameaças à Validade Externa

Duas ameaças foram identificadas:

- **Instrumentação**. As LPSs pedagógicas utilizadas no treinamento (LPS *Mobile Media*) e no estudo empírico (LPS *Arcade Game Maker* (AGM)) podem colocar em risco a validade externa. Portanto, tentou-se utilizar documentos padronizados a técnica *SMartyCheck* e à LPS AGM, bem como para a técnica *Ad hoc*. Contudo, por estas LPSs não serem comerciais, algumas suposições podem ser feitas sobre esta ameaça. Desta forma, outros estudos empíricos adicionais podem ser conduzidos, a fim de obter resultados diferentes por meio de outras LPSs.

- **Participantes**. A obtenção de participantes qualificados foi uma das grandes dificuldades encontradas para este estudo. Assim, participantes do meio acadêmico com um nível de conhecimento relevante em engenharia de software foram convidados para tentar amenizar tal ameaça.

5.7 Aprimoramento da *SMartyCheck 2.1* com os Resultados Obtidos

De acordo com as análises e a interpretação dos resultados obtidos no estudo empírico quantitativo realizado conforme a Seção 5.5, a técnica *SMartyCheck 2.1* foi aprimorada, evoluindo para a versão *SMartyCheck 2.2*. Deste modo, esta seção descreve as modificações que foram feitas na *SMartyCheck 2.1*. Assim, ao final desta seção os novos *checklists* da *SMartyCheck 2.2*, para a inspeção de diagramas *SMarty* de casos de uso e de classes são apresentados.

Originalmente, a taxonomia da *SMartyCheck* foi proposta permitindo a detecção de treze (13) tipos de defeitos, os quais foram avaliados, refinados e mantidos após a análise e interpretação dos resultados obtidos no estudo empírico qualitativo (Seção 4.5). Porém, após a realização deste estudo quantitativo, os tipos de defeitos "Incompleto" e "Incorreto" foram excluídos da taxonomia, pois os resultados das inspeções evidenciavam que a maioria dos participantes não conseguiram interpretar tais tipos de defeitos por meio da realização de inspeções incorretas.

A Figura 5.6 ilustra a taxonomia de tipos de defeitos da *SMartyCheck 2.2* aprimorada com onze (11) tipos de defeitos após a realização de tal estudo.

Figura 5.6: Taxonomia de Tipos de Defeitos da *SMartyCheck* após a Análise do Estudo Empírico Quantitativo.

Alguns itens do *checklist* da *SMartyCheck 2.1* foram alterados, corrigidos ou removidos. Portanto, após a exclusão dos tipos de defeitos "Incompleto" e "Incorreto", o

checklist da *SMartyCheck 2.2* ficou com vinte (17) itens diferentes especificados. Assim, as seguintes modificações foram feitas no *checklist* da *SMartyCheck 2.1*:

- Os itens do *checklist* da *SMartyCheck*, que tiveram suas descrições alteradas para facilitar o entendimento, estão relacionados com os seguintes tipos de defeitos: Regras de Negócio (RN.1), Ambiguidade (Am.1), Inconsistência (Incons.1), Inconsistência (Incons.2), Anomalia (An.1), Instável (Ins.1), Inviável (Inv.1) e Desvio Intencional (DI.2); e

- Novos itens no *checklist* da *SMartyCheck* foram adicionados de acordo com as necessidades detectadas pelos participantes durante a condução de tal estudo. Portanto, os novos itens a seguir foram inseridos no *checklist*, os quais estão relacionados com os tipos de defeitos: Fato Incorreto (FI.3) e Anomalia (An.2).

Com base nas modificações apresentadas, os tipos de defeitos foram classificados novamente e ordenados crescentemente. Portanto, os *checklists* da *SMartyCheck 2.1* para inspeção de diagramas *SMarty* de casos de uso e de classes de LPS são apresentados de modo aprimorado com as devidas modificações de acordo com os resultados obtidos no estudo empírico quantitativo.

Os *checklists* aprimorados evoluiram a *SMartyCheck 2.1* para a *SMartyCheck 2.2*. No entanto, os checklists da *SMartyCheck 2.2* não foram avaliados empiricamente. Assim, tais *checklists* podem ser avaliados por meio da condução de novos estudos empíricos, importantes para possibilitar uma melhor generalização dos resultados.

Assim, os itens a seguir apresentam os tipos de defeitos e os itens do *checklist* da *SMartyCheck 2.2* aprimorado para a inspeção de diagramas *SMarty* de casos de uso de LPS:

– **Com Oráculo (inspecionar comparando os diagramas sem defeitos e com defeitos incorporados)**

01. Regras de Negócio (RN)

- Este tipo de defeito deve ser aplicado de acordo com a definição de um domínio específico de uma LPS previamente escolhida.
  O Domínio/Regras de Negócio Pré-definidos para uma possível nova LPS, neste caso modificada, devem ser descrito(as) neste espaço.
  RN.1 Os casos de uso não são claros com o objetivo e as funcionalidades desejadas com base no domínio definido?

02. Inconsistência (Incons)

- **Incons.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior ou menor que o definido em *maxSelection* ou *minSelection* da variabilidade (<<`variability`>>) associada?
- **Incons.2** Existe algum caso de uso especificado com o estereótipo <<`optional`>> cujo número de variantes especificadas é maior ou menor que o definido em *maxSelection* ou *minSelection* da variabilidade (<<`variability`>>) associada?

03. Fato Incorreto (FI)

- **FI.1** Existe algum caso de uso com seu nome incorreto na LPS?
- **FI.2** Existe um ou mais casos de uso que não é possível comparar seu nome na LPS?
- **FI.3** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>> que está associado com um caso de uso na LPS que não seja <<`alternative_OR`>>?

05. Imutável (Imu)

- **Imu.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>>, no qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas de acordo as variantes já especificadas no meta-atributo *variants* na LPS?

09. Omissão (Om)

- **Om.1** Existe algum caso de uso especificado como obrigatório por meio do estereótipo <<`mandatory`>> que não esteja especificado na LPS?

10. Informação Estranha (IE)

- **IE.1** Existe algum caso de uso especificado além dos casos de uso já existentes na LPS?
- **IE.2** Existe algum caso de uso com sua funcionalidade duplicada na LPS?

11. Desvio Intencional (DI)

- **DI.1** Existe algum caso de uso que exige a seleção de outro (<<`requires`>>) e esse outro não está especificado na LPS?
- **DI.2** Existe algum caso de uso que exige a não seleção de outro (<<`mutex`>>) e esse outro não está especificado na LPS?

– **Sem o Oráculo (inspecionar apenas o diagrama com defeitos incorporados)**

04. Ambiguidade (Am)

- **Am.1** Existe algum caso de uso em que seu nome é igual a outro caso de uso de modo a possuir uma interpretação duplicada?

06. Anomalia (An)

- **An.1** Existe algum caso de uso especificado como <<`alternative_OR`>> que estende <<`extend`>> um caso de uso que não esteja especificado como <<`variationPoint`>>?
- **An.2** Existe algum caso de uso especificado como <<`alternative_XOR`>> que estende <<`extend`>> um caso de uso que não esteja especificado como <<`variationPoint`>>?

07. Instável (Ins)

- **Ins.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>>, o qual possui associado à uma variabilidade <<`variability`>> cujo meta-atributo *name* seja igual a de outros casos de uso especificados com o estereótipo <<`variationPoint`>>?

08. Inviável (Inv)

- **Inv.1** Existe algum caso de uso especificado com o estereótipo <<`variationPoint`>> que não permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar* = *false*?

10. Informação Estranha (IE)

- **IE.2** Existe algum caso de uso com sua funcionalidade duplicada na LPS?

Assim, os itens a seguir apresentam os tipos de defeitos e os itens do *checklist* da *SMartyCheck 2.2* aprimorado para a inspeção de diagramas *SMarty* de classes de LPS:

– **Com Oráculo (inspecionar comparando os diagramas sem defeitos e com defeitos incorporados)**

01. Regras de Negócio (RN)

- Este tipo de defeito deve ser aplicado de acordo com a definição de um domínio específico de uma LPS previamente escolhida.
  O Domínio/Regras de Negócio Pré-definidos para uma possível nova LPS, neste caso modificada, devem ser descrito(as) neste espaço.

RN.1 As classes não são claras com o objetivo e as funcionalidades desejadas com base no domínio definido?

02. Inconsistência (Incons)

- **Incons.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>> cujo número de variantes especificadas é maior ou menor que o definido em *maxSelection* ou *minSelection* da variabilidade (<<`variability`>>) associada?
- **Incons.2** Existe alguma classe especificada com o estereótipo <<`optional`>> cujo número de variantes especificadas é maior ou menor que o definido em *maxSelection* ou *minSelection* da variabilidade (<<`variability`>>) associada?

03. Fato Incorreto (FI)

- **FI.1** Existe alguma classe com seu nome incorreto na LPS?
- **FI.2** Existe uma ou mais classes que não é possível comparar seu nome na LPS?
- **FI.3** Existe alguma classe especificada com o estereótipo <<`variationPoint`>> que está associada com uma classe na LPS que não seja <<`alternative_OR`>>?

05. Imutável (Imu)

- **Imu.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>>, na qual as variantes associadas (<<`optional`>>, <<`alternative_OR`>> ou <<`alternative_XOR`>>) não podem ser combinadas ou escolhidas de acordo as variantes já especificadas no meta-atributo *variants* na LPS?

09. Omissão (Om)

- **Om.1** Existe alguma classe especificada como obrigatória por meio do estereótipo <<`mandatory`>> que não esteja especificada na LPS?

10. Informação Estranha (IE)

- **IE.1** Existe alguma classe especificada além das classes já existentes na LPS?
- **IE.2** Existe alguma classe com sua funcionalidade duplicada na LPS?

11. Desvio Intencional (DI)

- **DI.1** Existe alguma classe que exige a seleção de outra (<<`requires`>>) e essa outra não está especificada na LPS?

- **DI.2** Existe alguma classe que exige a não seleção de outra (<<`mutex`>>) e essa outra não está especificada na LPS?

– **Sem o Oráculo (inspecionar apenas o diagrama com defeitos incorporados)**

04. Ambiguidade (Am)

- **Am.1** Existe alguma classe em que seu nome é igual a outra classe de modo a possuir uma interpretação duplicada?

06. Anomalia (An)

- **An.1** Existe alguma classe especificada como <<`alternative_OR`>> que estende <<`extend`>> uma classe que não esteja especificada como <<`variationPoint`>>?
- **An.2** Existe alguma classe especificada como <<`alternative_XOR`>> que estende <<`extend`>> uma classe que não esteja especificada como <<`variationPoint`>>?

07. Instável (Ins)

- **Ins.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>>, a qual possui associada à uma variabilidade <<`variability`>> cujo meta-atributo *name* seja igual a de outras classes especificadas com o estereótipo <<`variationPoint`>>?

08. Inviável (Inv)

- **Inv.1** Existe alguma classe especificada com o estereótipo <<`variationPoint`>> que não permite incluir novas variantes conforme definido no meta-atributo *allowsAddingVar* = *false*?

10. Informação Estranha (IE)

- **IE.2** Existe alguma classe com sua funcionalidade duplicada na LPS?

5.8 Considerações Finais

Com a condução do estudo empírico quantitativo, a técnica *SMartyCheck* foi considerada a princípio eficiente, eficaz e efetiva, na qual foi aprimorada de acordo com os resultados obtidos. Assim, tal estudo forneceu evidências de que a *SMartyCheck* pode evoluir consideravelmente por meio da condução de novos estudos empíricos. Vale ressaltar a importância dos estudos empíricos, como um meio de avaliar o potencial de uma técnica ou ferramenta.

Durante a condução deste estudo, poucos participantes se disponibilizaram a colaborar, sendo assim, isto foi considerado como uma ameaça a validade deste estudo. Deve-se reconhecer que a quantidade de participantes podem influenciar, de forma relevante, os resultados obtidos neste estudo.

O estudo empírico qualitativo realizado colaborou diretamente com o planejamento deste estudo quantitativo. Portanto, uma lição importante aprendida com relação aos estudos (qualitativo e quantitativo), foi a melhora contínua da técnica *SMartyCheck* aperfeiçoada neste estudo quantitativo.

Por fim, visando a transferência de tecnologia, a contribuição com o meio científico, o compartilhamento dos resultados dos estudos empíricos realizados para possíveis replicações e a aplicação de procedimentos experimentais (Brooks *et al.*, 1996; Shull *et al.*, 2004), a instrumentação utilizada nos estudos foi empacotada e está disponível em: http://www.din.uem.br/s̃marty/smartycheck_estudos_empiricos.zip.

6

Conclusão

"A imaginação é mais importante que a ciência, porque a ciência é limitada, ao passo que a imaginação abrange o mundo inteiro."

Albert Einstein (1879 - 1955), Físico

Com a aplicação de técnicas de reutilização de artefatos de software no meio acadêmico, bem como em organizações é possível almejar resultados mais efetivos com base na abordagem de LPS. Dentre estes resultados, diversos benefícios e contribuições do uso de LPS permitem o desenvolvimento de software de forma mais efetiva, além de minimizar o tempo de testes, de riscos, de custos e o *time to market*. Neste contexto, a abordagem de LPS é apoiada pelo gerenciamento de variabilidades. Diversas abordagens permitem o gerenciamento de variabilidades. Sendo assim, a abordagem *SMarty* foi proposta e avaliada. *SMarty* faz o uso da notação UML que é amplamente conhecida e suporta mecanismos de extensão, como perfis.

De acordo com as definições apresentadas nesta pesquisa sobre LPS, gerenciamento de variabilidades e a abordagem *SMarty*, foi proposta a técnica *SMartyCheck*. A *SMarty-Check* é uma técnica de inspeção de software baseada em *checklist*, a qual contribui com a detecção de defeitos em diagramas *SMarty* de casos de uso e de classes. Para tanto, tal técnica se baseia em um *checklist* proposto a partir dos resultados obtidos em um

estudo secundário (Apêndice A) realizado para garantir a qualidade na adoção de uma taxonomia de tipos de defeitos adaptados para LPS.

Sob este panorama, uma proposta inicial da técnica *SMartyCheck* para a inspeção de diagramas *SMarty* foi desenvolvida e avaliada por meio da condução de um estudo empírico qualitativo seguido de um estudo empírico quantitativo. Em síntese, no estudo qualitativo foi evidenciado que *SMartyCheck* é viável, sendo refinada de acordo com o *feedback* dos participantes. Em seguida, *SMartyCheck* foi aprimorada por meio dos resultados obtidos no estudo quantitativo, o qual forneceu evidências de que a técnica *SMartyCheck* é mais eficiente, eficaz e efetiva em comparação com a técnica *Ad hoc*.

Com base nos estudos realizados corrobora-se que a necessidade de realização de estudos empíricos é um fator primordial nos âmbitos acadêmico e industrial. Deste modo, estudos empíricos devem ser conduzidos e compartilhados para que a ciência evolua. Não distante a isso, as áreas de LPS e de inspeção de software carecem desses tipos de estudos, permitindo fornecer indícios para a transferência de tecnologia.

Pode-se concluir, portanto, que a técnica *SMartyCheck* contribui para a inspeção de diagramas *SMarty* de casos de uso e de classes com base neste contexto e a partir dos estudos empíricos realizados, fornecendo evidências de que a técnica é viável, eficiente, eficaz e efetiva em comparação com a técnica *Ad hoc*. Assim, as próximas seções apresentam as principais contribuições acerca da técnica *SMartyCheck*, as limitações desta pesquisa e os trabalhos futuros.

6.1 Contribuições

O resultado principal desta pesquisa foi a técnica *SMartyCheck* para inspecionar diagramas *SMarty* de casos de uso e de classes, e suas respectivas avaliações empíricas. Assim, resumimos as seguintes contribuições:

- contribuições relevantes para o meio acadêmico, pois a técnica *SMartyCheck* possibilitou a inspeção de artefatos de software em conjunto com a abordagem *SMarty*, conforme evidenciaram os estudos empíricos (Capítulos 4 e 5);

- usuários podem se beneficiar com a técnica *SMartyCheck*, pois é possível detectar defeitos de diagramas *SMarty* antes mesmo que um produto de LPS seja gerado;

- outra contribuição importante é o fato de que a técnica *SMartyCheck* melhora o processo de inspeção de software de diagramas *SMarty*. Para tanto, essa contribuição pode minimizar defeitos em LPS e, consequentemente, possibilitar que novos

produtos específicos sejam gerados com defeitos reduzidos, acelerando a derivação de produtos com qualidade. Entretanto, o mecanismo de geração e configuração de produtos não são tratados por este trabalho; e

- vale ressaltar que, com a proposta de uma nova técnica de inspeção para LPS, como é o caso da *SMartyCheck*, foi possível contribuir com o meio científico e com o meio industrial com possível transferência de tecnologia.

Resultados e Contribuições quanto às Avaliações Empíricas da Técnica *SMartyCheck*

Os estudos empíricos foram planejados e conduzidos no escopo dos métodos mistos (*mixed-methods*) para avaliar *SMartyCheck*. De acordo com tal estratégia, os itens a seguir apresentam as contribuições alcançadas de acordo com tais estudos empíricos realizados:

- a *SMartyCheck* contribui para a inspeção de diagramas *SMarty* por meio de tipos de defeitos coerentes e à corretude baseada em diagramas a partir de uma estrutura bem definida, sendo apoiada por meio de estereótipos da abordagem *SMarty*, os quais são elementos chaves essenciais para melhorar a detecção de defeitos;

- a *SMartyCheck* contribui fornecendo tipos de defeitos coerentes, uma estrutura bem definida, além do *checklist* que deve ser seguido com um critério sistemático para conduzir as inspeções. Portanto, tais características mencionadas diferem *SMartyCheck* em comparação com uma técnica *Ad hoc* para LPS; e

- as evidências iniciais dos estudos corroboraram sobre a relevância da qualidade da *SMartyCheck* no ambiente acadêmico. Com isso, é possível estabelecer uma base para que pesquisadores possam adotá-la para a realização de novas investigações, bem como a indústria pode ser beneficiada por meio da condução de novos estudo empíricos.

Publicações dos Resultados e Contribuições:

- GERALDI, R. T.; OLIVEIRAJR, E. A.; CONTE, T. U.; STEINMACHER, I. F. Checklist-based Inspection of SMarty Variability Models: Proposal and Empirical Feasibility Study (Aceito - Proc. Int. Conf. on Enterprise Information Systems (ICEIS 2015) - Conferência).

- GERALDI, R. T.; OLIVEIRAJR, E. Investigating Defect Types of Software Inspection Techniques: a Systematic Mapping Study (Submetido - ELSEVIER The Journal of Systems and Software (JSS) - Periódico).

- GERALDI, R. T.; OLIVEIRAJR, E. On the Evidence of UML SMarty Model Inspections (Em preparação para submissão - ELSEVIER Information and Software Technology (INFSOF (IST)) - Periódico).

6.2 Limitações

É importante mencionar, que as principais dificuldades identificadas estão relacionadas a encontrar participantes que se disponibilizem a contribuir com os estudos empíricos e encontrar participantes qualificados dentro de um universo específico de conhecimento. Um fator que limitou a avaliação empírica da *SMartyCheck* foi a condução da sua avaliação apenas no meio acadêmico. Ao contrário, organizações contactadas não se mostraram propícias em participar dos estudos empíricos, alegando problemas de falta de tempo hábil para a participação, além do consumo de capital intelectual (colaboradores) da empresa para tarefas extras dentro do horário de trabalho dos colaboradores. Neste sentido, esta pesquisa apresentou algumas limitações que são pontuadas e discutidas nos itens a seguir:

- amostras pequenas de participantes nos estudos qualitativo e quantitativo. A escassez de participantes foi considerada uma das ameaças para a avaliação empírica desta pesquisa. Porém, esta limitação pode ser reduzida por meio da obtenção de amostras maiores de participantes com o planejamento e condução de novos estudos empíricos e, na condução de replicações dos estudos empíricos realizados nesta pesquisa;

- a técnica *SMartyCheck* não foi avaliada no ambiente industrial como mencionado. Acredita-se que empresas de tecnologia de informação poderiam contribuir, em muito, com a evolução da técnica *SMartyCheck*. Contudo, esta limitação pode ser mitigada por meio da condução de estudos empíricos mais simplificados e ágeis, com o intuito de permitir a participação de colaboradores de diferentes empresas, já que o principal problema observado está relacionado a adequação do tempo;

- a dificuldade em encontrar LPSs reais (comerciais), além da falta destas LPSs, são limitações evidentes para a condução de avaliações empíricas. A existência ou disponibilização de LPS comerciais, em parceria com o meio industrial, seria

fundamental para corroborar com a condução dos estudos empíricos, a fim de avaliar a técnica *SMartyCheck* em ambiente corporativo;

- a ausência de técnicas de inspeção para LPS é uma limitação presente na literatura. Poucas técnicas de inspeção para diagramas UML de LPS foram propostas. Portanto, levando em consideração a limitação de estudos empíricos sobre técnicas de inspeção na literatura, este é um cenário motivador e desafiador para propor novas técnicas de inspeção para diagramas UML de LPS, como é o caso da *SMartyCheck*;

- a falta e a limitação de tipos de defeitos mais segmentados, bem como direcionados para técnicas de inspeção na literatura. Assim, esta limitação foi minimizada por meio da realização de um estudo secundário (Apêndice A) sobre tipos de defeitos em técnicas de inspeção de software. Desta forma, foi possível adaptar os tipos de defeitos para o desenvolvimento de uma taxonomia peculiar no escopo da proposta da técnica *SMartyCheck*; e

- *SMartyCheck* suporta somente a inspeção de diagramas *SMarty* de casos de uso e de classes. Porém, é possível vislumbrar a extensão da *SMartyCheck* para outros diagramas *SMarty* como atividades, sequência e componentes.

6.3 Trabalhos Futuros

De acordo com a experiência adquirida com o desenvolvimento da técnica *SMartyCheck* e com a condução dos estudos empíricos, é possível propor como trabalhos futuros:

- investigar a possível existência de mais tipos de defeitos por meio da realização de outro estudo secundário, no intuito de adaptá-los, se viável, na taxonomia de tipos de defeitos da técnica *SMartyCheck*;

- permitir a inspeção de diferentes diagramas *SMarty*, tais como diagramas de atividades, sequência e componentes;

- automatizar parcial ou totalmente o processo de inspeção da *SMartyCheck* por meio do desenvolvimento de uma ferramenta de análise sintática para detectar defeitos em diagramas *SMarty* de LPS;

- adaptar *SMartyCheck* para detectar defeitos em diagramas de características e em diagramas UML distintos que contenham estereótipos e meta-atributos, assim como os especificados na abordagem *SMarty*;

- planejar e conduzir mais estudos empíricos para aprimorar *SMartyCheck*, bem como corroborar e generalizar os resultados obtidos;
- compartilhar o desenvolvimento da *SMartyCheck* por meio da criação de uma versão adaptada de ferramenta de acordo com um projeto *Open Source*, com o princípio de permitir que pesquisadores interessados na técnica contribuam com a sua evolução;
- propor a técnica *SMartyPerspect* para inspeção de diagramas *SMarty* de variabilidade com base na técnica PBR; e
- comparar *SMartyCheck* com a técnica baseada em perspectivas a ser desenvolvida.

REFERÊNCIAS

Ackerman, A. F.; Buchwald, L. S.; Lewski, F. H. Software Inspections: An Effective Verification Process. *IEEE Software*, v. 6, n. 3, p. 31–36, 1989.

Ahnassay, A.; Bagheri, E.; Gasevic, D. *Empirical Evaluation in Software Product Line Engineering*. Relatório técnico, Ryerson University, 2013.
Disponível em `http://ls3.rnet.ryerson.ca/wp-content/uploads/2013/08/TR-LS3-130084R4T.pdf`

Alshazly, A. a.; Elfatatry, A. M.; Abougabal, M. S. Detecting defects in software requirements specification. *Alexandria Engineering Journal*, v. 53, n. 3, p. 513–527, 2014.

Amoui, M.; Kaushik, N.; Al-Dabbagh, A.; Tahvildari, L.; Li, S.; Liu, W. Search-based Duplicate Defect Detection: an Industrial Experience. In: *Proc. Int. Conf. on Mining Software Repositories (MSR)*, 2013, p. 173–182.

Anda, B.; Sjøberg, D. I. K. Towards an inspection technique for use case models. In: *Proc. Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, New York, NY, USA: ACM, 2002, p. 127–134.

Aurum, A.; Petersson, H.; Wohlin, C. State-of-the-art: software inspections after 25 years. *Software Testing, Verification and Reliability*, v. 12, n. 3, p. 133–154, 2002.

Bailey, J.; Budgen, D.; Turner, M.; Kitchenham, B. A.; Brereton, P.; Linkman, S. Evidence relating to Object-Oriented software design: A survey. *Int. Symposium on Empirical Software Engineering and Measurement (ESEM)*, p. 482–484, 2007.

Barney, S.; Petersen, K.; Svahnberg, M.; Aurum, A.; Barney, H. Software quality trade-offs: A systematic map. *Information and Software Technology (INFSOF)*, v. 54, n. 7, p. 651–662, 2012.

Basili, V.; Caldiera, G.; Lanubile, F.; Shull, F. Studies on Reading Techniques. In: *Proc. on Twenty-First Annual Software Engineering Workshop*, 1996, p. 002.

Basili, V. R.; Caldiera, G.; Rombach, H. D. The GQM approach. *Encyclopedia of Software Engineering (Wiley)*, 1994.

Basili, V. R.; Green, S.; Laitenberger, O.; Shull, F.; Sørumgård, S.; Zelkowitz, M. V. *The empirical investigation of perspective-based reading.* Relatório Técnico, College Park, MD, USA, 1995.
Disponível em `http://www.cs.umd.edu/users/mvz/handouts/emp_pbr.pdf`

Basili, V. R.; Selby, R. W. Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering*, v. SE-13, n. 12, p. 1278–1296, 1987.

Belgamo, A.; Fabbri, S.; Maldonado, J. C. TUCCA: Improving the Effectiveness of Use Case Construction and Requirement Analysis. In: *Proc. Int. Symposium on Empirical Software Engineering (ESEM)*, IEEE, 2005, p. 257–266.

Biffl, S. Evaluating defect estimation models with major defects. *Journal of Systems and Software (JSS)*, v. 65, n. 1, p. 13–29, 2003.

Biffl, S.; Freimut, B.; Laitenberger, O. Investigating the cost-effectiveness of reinspections in software development. In: *Proc. Int. Conf. on Software Engineering (ICSE)*, IEEE Computer Society, 2001, p. 155–164.

Biffl, S.; Halling, M. Software Product Improvement with Inspection. A Large-scale Experiment on the Influence of Inspection Processes on Defect Detection in Software Requirements Documents. In: *Proc. of the Euromicro Conf. Informatics: Inventing the Future*, IEEE Computer Society, 2000, p. 262–269.

Biffl, S.; Halling, M. Investigating the defect detection effectiveness and cost benefit of nominal inspection teams. *Transactions Software Engineering (TSE)*, v. 29, n. 5, p. 385–397, 2003.

Boehm, B.; Basili, V. R. Software Defect Reduction Top 10 List. *IEEE Computer*, v. 34, n. 1, p. 135–137, 2001.

Bosch, J.; Lee, J. Software Product Lines: Going Beyond. In: *Proc. Int. Conf. on Software Product Line Conference (SPLC)*, Jeju Island, South Korea, 2010, p. 548.

Brooks, A.; Daly, J.; Miller, J.; Roper, M.; Wood, M. Replication of experimental results in software engineering. *IEEE Transactions on Software Engineering*, p. 1–37, 1996.

Brykczynski, B. A Survey of Software Inspection Checklists. *ACM SIGSOFT Software Engineering Notes*, v. 24, n. 1, p. 82, 1999.

Brykczynski, B.; Meeson, R.; Wheeler, D. A. Software Inspection: Eliminating Software Defects. In: *Proc. Annual Software Technology Conf.*, 1994, p. 12.

Capilla, R.; Bosch, J.; Kang, K.-C. *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer Berlin Heidelberg, 319 p., 2013.

Card, D. Learning from Our Mistakes with Defect Causal Analysis. *IEEE Software*, v. 15, n. 1, p. 56–63, 1998.

Carver, J. The Impact of Background and Experience on Software Inspections. *Empirical Software Engineering (ESE)*, v. 9, n. 3, p. 259–262, 2004.

Chen, L.; Ali Babar, M.; Ali, N. Variability Management in Software Product Lines: A Systematic Review. In: *Proc. Int. Software Product Line Conf. (SPLC)*, Pittsburgh, PA, USA: Carnegie Mellon University, 2009, p. 81–90.

Cheng, B.; Jeffery, R. Comparing Inspection Strategies for Software Requirement Specifications. In: *Proc. Conf. on Australian Software Engineering*, IEEE Computer Society, 1996, p. 203–211.

Chillarege, R.; Bhandari, I.; Chaar, J.; Halliday, M.; Moebus, D.; Ray, B.; Wong, M.-Y. Orthogonal Defect Classification - A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering (TSE)*, v. 18, n. 11, p. 943–956, 1992.

Ciolkowski, M. What Do We Know About Perspective-Based Reading? An Approach for Quantitative Aggregation in Software Engineering. *Int. Symposium on Empirical Software Engineering and Measurement*, p. 133–144, 2009.

Ciolkowski, M.; Laitenberger, O.; Biffl, S. Software Reviews: The State of the Practice. *IEEE Software*, v. 20, n. 6, p. 46–51, 2003.

Clements, P.; Northrop, L. *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing, 2002.

Contieri Junior, A. C. *Aplicação de Métricas em Arquiteturas de Linhas de Produto de Software*. Trabalho de conclusão de curso, Universidade Estadual de Maringá, 2010.

Contieri Junior, A. C.; Correia, G. G.; Colanzi, T. E.; Gimenes, I. M. S.; Oliveira Junior, E. A.; Ferrari, S.; Masiero, P. C.; Garcia, A. F. Extending UML Components to Develop Software Product-Line Architectures: Lessons Learned. In: *Proc. European Conf. on Software Architecture*, Springer Berlin Heidelberg, 2011, p. 130–138 (*Lectures Notes in Computer Science*, v.6903).

Cooper, K.; Liddle, S.; Dascalu, S. Experiences Using Defect Checklists in Software Engineering Education. In: *Proc. Int. Conf. on Computer Applications in Industry and Engineering (CAINE)*, 2005, p. 402–409.

Corbin, J. M.; Strauss, A. L. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. 3 ed. Sage Publications, 2008.

Cox, K.; Aurum, A.; Jeffery, R. An Experiment in Inspecting the Quality of Use Case Descriptions. *Journal of Research and Practice in Information Technology*, v. 36, n. 4, p. 211–229, 2004a.

Cox, K.; Jeffery, R.; Aurum, A. *A Use Case Description Inspection Experiment*. Relatório técnico, University of New South Wales - School of Computer Science and Engineering, 2004b.
Disponível em `http://pandora.nla.gov.au/pan/32869/20040603-0000/ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/0414.pdf`

Creswell, J. W.; Clark, V. L. P. *Designing and Conducting Mixed Methods Research*. 2nd ed. SAGE, 488 p., 2010.

Cunha, R.; Conte, T. U.; de Almeida, E. S.; Maldonado, J. C. A Set of Inspection Technique on Software Product Line Models. In: *Proc. Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, 2012, p. 657–662.

Denger, C.; Ciolkowski, M.; Lanubile, F. Investigating the Active Guidance Factor in Reading Techniques for Defect Detection. In: *Proc. Int. Conf. on Empirical Software Engineering (ESEM - ISESE)*, 2004, p. 219–228.

Denger, C.; Paech, B. An Integrated Quality Assurance Approach for Use Case Based Requirements. In: *Modellierung*, 2004, p. 59–74.

Denger, C.; Shull, F. J. A Practical Approach for Quality-Driven Inspections. *IEEE Software*, v. 24, n. 2, p. 79–86, 2007.

Dunsmore, A.; Roper, M.; Wood, M. The role of comprehension in software inspection. *Journal of Systems and Software (JSS)*, v. 52, p. 121–129, 2000.

Dybå T.; Prikladnicki, R.; Rönkkö, K.; Seaman, C.; Sillito, J. Qualitative research in software engineering. *Empirical Software Engineering*, v. 16, n. 4, p. 425–429, 2011.

Ebenau, R. G.; Strauss, S. H. *Software Inspection Process.* New York, NY, USA: McGraw-Hill, Inc., 1994.

Eliza, R.; Lagares, V. Adotando Checklist no Teste de Software. *Java Magazine 87*, p. 53–61, 2012.

Fagan, M. Software pioneers. cáp. A History of Software Inspections, New York, NY, USA: Springer-Verlag Inc., p. 562–573, 2002.

Fagan, M. E. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, v. 15, n. 3, p. 182–211, 1976.

Fagan, M. E. Advances in Software Inspections. *IEEE Transactions on Software Engineering*, v. SE-12, n. 7, p. 744–751, 1986.

Fernández-Sáez, A. M.; Genero, M.; Chaudron, M. R. Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. *Information and Software Technology (INFSOF)*, v. 55, n. 7, p. 1119–1142, 2013.

Fiori, D. R.; Gimenes, I. M. S.; Maldonado, J. C.; Oliveira Junior, E. A. Variability Management in Software Product Line Activity Diagrams. In: *Proc. Int. Conf. on Distributed Multimedia Systems*, 2012, p. 89–94.

Fragal, V. H.; Silva, R. F.; Gimenes, I. M. S.; Oliveira Júnior, E. A. Application Engineering for Embedded Systems - Transforming SysML Specification to Simulink within a Product-Line based Approach. In: *Proc. Int. Conf. on Enterprise Information Systems*, SciTePress - Science and and Technology Publications, 2013, p. 94–101.

Freimut, B.; Laitenberger, O.; Biffl, S. Investigating the impact of reading techniques on the accuracy of different defect content estimation techniques. In: *Proc. Int. Symposium on Software Metrics (ESEM - METRIC)*, 2001, p. 51–62.

Galster, M.; Weyns, D.; Tofan, D.; Michalik, B.; Avgeriou, P. Variability in Software Systems - A Systematic Literature Review. *IEEE Transactions on Software Engineering*, v. 40, n. 3, p. 282–306, 2013.

Gilb, T. Agile Specification Quality Control. *Cutter IT Journal*, p. 35–39, 2005.

Gilb, T. Agile Specification Quality Control: Shifting emphasis from cleanup to sampling defects. *Testing Experience*, p. 87–93, 2009.

Gilb, T.; Graham, D. *Software Inspection*. Boston, MA, USA: Addison-Wesley, 471 p., 1993.

Godfrey, A. B. Report: The History and Evolution of Quality in AT&T. *AT&T Technical Journal*, v. 65, n. 2, p. 9–20, 1986.

Gomaa, H. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Redwood City, CA, USA: Addison-Wesley, 2004.

Gopalakrishnan Nair, T.; Suma, V.; Kumar Tiwari, P. Significance of depth of inspection and inspection performance metrics for consistent defect management in software industry. *IET Software*, v. 6, n. 6, p. 524, 2012.

Grunbacher, P.; Halling, M.; Biffl, S. An empirical study on groupware support for software inspection meetings. In: *Proc. Int. Conf. on Automated Software Engineering (ASE)*, 2003, p. 4–11.

Gurp, J.; Bosch, J.; Svahnberg, M. On the notion of variability in software product lines. *Proc. Working IEEE/IFIP Conf. on Software Architecture*, p. 45–54, 2001.

Halmans, G.; Pohl, K. Communicating the variability of a software-product family to customers. *Software and Systems Modeling*, v. 2, n. 1, p. 15–36, 2003.

Hayes, J. Building a requirement fault taxonomy: experiences from a NASA verification and validation research project. *Proc. Int. Symposium on Software Reliability Engineering (ISSRE)*, p. 49–59, 2003.

Hayes, J.; Raphael, I.; Holbrook, E.; Pruett, D. A Case History of International Space Station Requirement Faults. In: *Proc. IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS)*, IEEE Computer Society, 2006, p. 10.

He, L.; Carver, J. PBR vs. Checklist: A Replication in the N-Fold Inspection Context. In: *Proc. ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)*, New York, NY, USA: ACM Press, 2006, p. 95.

Hole, G. Mann-Whitney Test Handout Version 1.0. Acessado em 12 de Nov. 2014, 2011.
Disponível em `http://www.sussex.ac.uk/Users/grahamh/RM1web/MannWhitneyHandout2011.pdf`

Hungerford, B.; Hevner, A.; Collins, R. Reviewing software diagrams: a cognitive study. *Transactions Software Engineering (TSE)*, v. 30, n. 2, p. 82–96, 2004.

IEEE Glossary of Software Engineering Terminology, Standard 729-1983. 1983.
Disponível em `http://standards.ieee.org/findstds/standard/729-1983.html`

IEEE Recommended Practice for Software Requirements Specifications, Standard 830-1998. 1998a.
Disponível em `http://standards.ieee.org/findstds/standard/830-1998.html`

IEEE Software Configuration Management Plans, Standard 828-1998, p. 22. 1998b.
Disponível em `http://standards.ieee.org/findstds/standard/828-1998.html`

IEEE Software Reviews, Standard 1028-1997. 1998c.
Disponível em `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=663254`

IEEE Classification for Software Anomalies, Standard 1044-2009. 2010.
Disponível em `http://standards.ieee.org/findstds/standard/1044-2009.html`

IEEE System and Software Verification and Validation, Standard 1012-2012. 2012.
Disponível em `http://standards.ieee.org/findstds/standard/1012-2012.html`

Jacobson, I.; Griss, M.; Jonsson, P. *Software reuse: Architecture, process and organization for business success.* New York, NY, USA: ACM Press/Addison-Wesley Publishing, 1997.

Jones, C. Software defect-removal efficiency. *IEEE Computer*, v. 29, n. 4, p. 94–95, 1996.

Juristo, N.; Moreno, A. M. *Basics of Software Engineering Experimentation*. Madrid: Springer, 367 p., 2010.

Kalinowski, M.; Travassos, G. H. A Computational Framework for Supporting Software Inspections. In: *Proc. Int. Conf. on Automated Software Engineering (CASE)*, IEEE, 2004, p. 46–55.

Karg, L. M.; Grottke, M.; Beckhaus, A. A systematic literature review of software quality cost research. *Journal of Systems and Software (JSS)*, v. 84, n. 3, p. 415–427, 2011.

Kelly, D.; Shepard, T. A case study in the use of defect classification in inspections. In: *Proc. Int. Conf. Centre for Advanced Studies on Collaborative Research (CASCON)*, IBM Press, 2001, p. 7.

Kelly, D.; Shepard, T. An experiment to investigate interacting versus nominal groups in software inspection. In: *Proc. Int. Conf. on Centre for Advanced Studies on Collaborative Research (CASCON)*, IBM Press, 2003, p. 122–134.

Kelly, J. C.; Sherif, J. S.; Hops, J. An Analysis of Defect Densities Found During Software Inspections. *Journal of Systems and Software (JSS)*, v. 17, n. 2, p. 111–117, 1992.

Kitchenham, B. A. Guidelines for Performing Systematic Literature Reviews in Software Engineering. In: *Proc. Int. Conf. on Software Engineering (ICSE)*, New York, USA: ACM Press, 2007, p. 1051.

Kitchenham, B. A.; Budgen, D.; Pearl Brereton, O. Using mapping studies as the basis for further research - A participant-observer case study. *Information and Software Technology (INFSOF)*, v. 53, n. 6, p. 638–651, 2010.

Kollanus, S.; Koskinen, J. Survey of Software Inspection Research. *The Open Software Engineering Journal*, v. 3, p. 15–34, 2009.

Korherr, B.; List, B. A UML 2 Profile for Variability Models and their Dependency to Business Processes. In: *Proc. Int. Conf. on Database and Expert Systems Applications (DEXA)*, IEEE, 2007, p. 829–834.

KRISTIANSEN, J. M. W. *Software Defect Analysis - An Empirical Study of Causes and Costs in the Information Technology Industry.* Dissertação de mestrado, Norwegian University of Science and Technology, 2010.

LAITENBERGER, O. A Survey of Software Inspection Technologies. In: *Handbook on Software Engineering and Knowledge Engineering*, Kaiserslautern, Germany, 2002, p. 517–555.

LAITENBERGER, O.; ATKINSON, C.; SCHLICH, M.; EMAM, K. E. An experimental comparison of reading techniques for defect detection in {UML} design documents. *Journal of Systems and Software (JSS)*, v. 53, n. 2, p. 183–204, 2000.

LAITENBERGER, O.; DEBAUD, J.-M. An encompassing life cycle centric survey of software inspection. *Journal of Systems and Software (JSS)*, v. 50, n. 1, p. 5–31, 2000.

LAITENBERGER, O.; EL EMAM, K.; HARBICH, T. G. An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective Based Reading of Code Documents. *IEEE Transactions on Software Engineering (TSE)*, v. 27, n. 5, p. 387–421, 2001.

LAMSWEERDE, A. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons, 2009.

LANGE, C. F. J.; CHAUDRON, M. R. V. Effects of Defects in UML Models: an Experimental Investigation. In: *Proc. Int. Conf. on Software Engineering (ICSE)*, New York, NY, USA: ACM, 2006, p. 401–411.

LANUBILE, F.; MALLARDO, T.; CALEFATO, F.; DENGER, C.; CIOLKOWSKI, M. Assessing the Impact of Active Guidance for Defect Detection: a Replicated Experiment. In: *Proc. Int. Conf. on Software Metrics (ESEM - METRIC)*, 2004, p. 269–278.

LANUBILE, F.; VISAGGIO, G. Evaluating Defect Detection Techniques for Software Requirements Inspections. In: *International Software Engineering Research Network (ISERN)*, Bari, Italy: University of Bari, 2000, p. 24.

LIKERT, R. A Technique for the Measurement of Attitudes. *Archives of Psychology*, v. 22, n. 140, p. 1–55, 1932.

LINDEN, F.; SCHMID, K.; ROMMES, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer, 340 p., 2007.

Marcolino, A. *Avaliação experimental da abordagem SMarty para gerenciamento de variabilidade em linhas de produto de software baseadas em UML.* Dissertação de mestrado, Universidade Estadual de Maringá (UEM), 2014.

Marcolino, A.; OliveiraJr, E.; Gimenes, I. Towards the Effectiveness of the SMarty Approach for Variability Management at Sequence Diagram Level. In: *Proc. Int. Conf. on Enterprise Information Systems (ICEIS)*, Lisboa, Portugal, 2014a, p. 249–256.

Marcolino, A.; OliveiraJr, E.; Gimenes, I.; Barbosa, E. Empirically Based Evolution of a Variability Management Approach at UML Class Level. In: *Proc. Int. Conf. Computers, Software & Applications (COMPSAC)*, Vasteras, Sweden, 2014b, p. 354–363.

Marcolino, A.; OliveiraJr, E.; Gimenes, I.; Maldonado, J. Towards the Effectiveness of a Variability Management Approach at Use Case Level. In: *Proc. Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, 2013, p. 214–219.

Margarido, I. L.; Faria, J. a. P.; Vidal, R. M.; Vieira, M. Classification of Defect Types in Requirements Specifications: Literature Review, Proposal and Assessment. 2009.

Mello, R. M.; Pereira, W. M.; Travassos, G. H. Activity Diagram Inspection on Requirements Specification. *Proc. of the Brazilian Symposium on Software Engineering*, p. 168–177, 2010.

Mello, R. M.; Teixeira, E. N.; Schots, M.; Werner, C. M. L. Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections. *Journal of Universal Computer Science (JUCS)*, v. 20, n. 5, p. 720–745, 2014.

Mello, R. M.; Teixeira, E. N.; Schots, M.; Werner, C. M. L.; Travassos, G. H. Checklist-Based Inspection Technique for Feature Models Review. In: *Proc. of the Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, IEEE, 2012, p. 140–149.

Mendeley Mendeley Desktop v1.13.1. Acessado em 24 de Janeiro, 2015. Disponível em `http://www.mendeley.com`

Miller, J.; Wood, M.; Roper, M. Further experiences with scenarios and checklists. *Empirical Software Engineering*, v. 3, n. 1, p. 37–64, 1998.

Miller, J.; Yin, Z. Adding diversity to software inspections. In: *Proc. Int. Conf. on Cognitive Informatics (ICCI)*, 2003, p. 81–86.

Miller, L. A.; Groundwater, E. H.; Hayes, J. E.; Mirsky, S. M. *Guidelines for the Verification and Validation of Expert System Software and Conventional Software*. US Nuclear Regulatory Commission, 1995.

Mishra, D.; Mishra, A. Simplified software inspection process in compliance with international standards. *Computer Standards & Interfaces*, v. 31, n. 4, p. 763–771, 2009.

Mohabbati, B.; Asadi, M.; Gašević, D.; Hatala, M.; Müller, H. a. Combining service-orientation and software product line engineering: A systematic mapping study. *Information and Software Technology (INFSOF)*, v. 55, n. 11, p. 1845–1859, 2013.

Mota Silveira Neto, P. A.; Carmo Machado, I.; McGregor, J. D.; Almeida, E. S.; Lemos Meira, S. R. A systematic mapping study of software product lines testing. *Information and Software Technology (INFSOF)*, v. 53, n. 5, p. 407–423, 2011.

Munson, J. C.; Nikora, A. P.; Sherif, J. S. Software faults: A quantifiable definition. *Advances in Engineering Software*, v. 37, n. 5, p. 327–333, 2006.

Northrop, L. M. SEI's Software Product Line Tenets. *IEEE Software*, v. 19, n. 4, p. 32–40, 2002.

Novais, R. L.; Torres, A.; Mendes, T. S.; Mendonça, M.; Zazworka, N. Software evolution visualization: A systematic mapping study. *Information and Software Technology (INFSOF)*, v. 55, n. 11, p. 1860–1883, 2013.

Oizumi, W. N.; Correia, G. G.; Colanzi, T. E.; Ferrari, S.; Gimenes, I. M. S.; Oliveira, E. A.; Garcia, A. F.; Masiero, P. C. On the Proactive Design of Product-Line Architectures with Aspects: An Exploratory Study. In: *Proc. IEEE Annual Computer Software and Applications Conf.*, IEEE, 2012, p. 273–278.

OliveiraJr, E.; Gimenes, I.; Huzita, E.; Maldonado, J. A Variability Management Process for Software Product Lines. In: *Proc. Int. Conf. of the Centre for Advanced Studies on Collaborative Research (CASCON)*, IBM Press, 2005, p. 225–241.

OliveiraJr, E.; Gimenes, I.; Maldonado, J. Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science (JUCS)*, v. 16, n. 17, p. 2374–2393, 2010.

OliveiraJr, E.; Gimenes, I.; Maldonado, J.; Masiero, P. Systematic Evaluation of Software Product Line Architectures. *Journal of Universal Computer Science (JUCS)*, v. 19, n. 1, p. 25–52, 2013.

OliveiraJr, E. A. *SystEM-PLA: um método sistemático para avaliaçao de arquitetura de linha de produto de software baseada em UML.* Tese de Doutoramento, Universidade de São Paulo (USP) - São Carlos, 2010.

OMG Software & Systems Process Engineering Meta-Model Specification (OMG SPEM), Version 2.0. Acessado em 10 de Jul. 2014, 2008.
Disponível em `http://www.omg.org/spec/SPEM/2.0/PDF/`

OMG Systems Modeling Language (OMG SysML), Version 1.2. Acessado em 10 de Jul. 2014, 2010.
Disponível em `http://www.omg.org/spec/SysML/1.2/PDF/`

OMG Unified Modeling Language (UML), Superstructure Version 2.4.1. Acessado em 10 de Jul. 2014, 2011.
Disponível em `http://www.omg.org/spec/UML/2.4.1/PDF/`

OMG Unified Modeling Language (UML) Testing Profile (UTP), Version 1.2. Acessado em 05 de Jun. 2014, 2013.
Disponível em `http://www.omg.org/spec/UTP/1.2/PDF/`

Petersen, K.; Feldt, R.; Mujtaba, S.; Mattsson, M. Systematic Mapping Studies in Software Engineering. In: *Proc. Int. Conf. on Evaluation and Assessment in Software Engineering (EASE)*, Swinton, UK: British Computer Society, 2008a, p. 68–77.

Petersen, K.; Rönkkö, K.; Wohlin, C. The Impact of Time Controlled Reading on Software Inspection Effectiveness and Efficiency: A Controlled Experiment. In: *Proc. Int. Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA: ACM, 2008b, p. 139–148.

Pohl, K.; Böckle, G.; Linden, F. *Software Product Line Engineering: Foundations, Principles, and Techniques.* Springer, 2005.

Porter, A.; Siy, H.; Mockus, A.; Votta, L. Understanding the sources of variation in software inspections. *ACM Transaction Software Engineering Methodology (ACM TOSEM)*, v. 7, n. 1, p. 41–79, 1998.

Pressman, R. S. *Software Engineering - A Practitioner's Approach.* 7th ed. McGraw-Hill, 930 p., 2010.

Rodrigues, E. M.; Zorzo, A. F.; Oliveira Junior, E. A.; Souza Gimenes, I. M.; Maldonado, J. C.; Domingues, A. R. P. PlugSPL: An Automated Environment for Supporting Plugin-based Software Product Lines. In: *SEKE*, Knowledge Systems Institute Graduate School, 2012, p. 647–650.

Rombach, D.; Ciolkowski, M.; Jeffery, R.; Laitenberger, O.; McGarry, F.; Shull, F. Impact of research on practice in the field of inspections, reviews and walkthroughs. *ACM SIGSOFT Software Engineering Notes*, v. 33, n. 6, p. 26, 2008.

Roper, M.; Wood, M.; Miller, J. An empirical evaluation of defect detection techniques. *Information and Software Technology (INFSOF)*, v. 39, n. 11, p. 763–775, 1997.

Russell, G. W. Experience with Inspection in Ultralarge-Scale Developments. *IEEE Software*, v. 8, n. 1, p. 25–31, 1991.

Sabaliauskaite, G. *Investigating Defect Detection in Object-Oriented Design and Cost-Effectiveness of Software Inspection.* Tese de doutorado, Osaka University, 2004.

Sabaliauskaite, G.; Kusumoto, S.; Inoue, K. Extended Metrics to Evaluate Cost Effectiveness of Software Inspections. *IEICE Transactions Fundamentals/Commun./Electron./INF. & SYST.*, v. E85-A/B/C/D, n. 1, p. 1–6, 2002a.

Sabaliauskaite, G.; Kusumoto, S.; Inoue, K. Assessing defect detection performance of interacting teams in object-oriented design inspection. *Information and Software Technology (INFSOF)*, v. 46, n. 13, p. 875–886, 2004.

Sabaliauskaite, G.; Matsukawa, F.; Kusumoto, S.; Inoue, K. An Experimental Comparison of Checklist-based Reading and Perspective-Based Reading for UML Design Document Inspection. *Proc. Int. Symposium on Empirical Software Engineering (ESEM)*, p. 148–157, 2002b.

Sabaliauskaite, G.; Matsukawa, F.; Kusumoto, S.; Inoue, K. An Experimental Comparison of Checklist-based Reading and Perspective-Based Reading for UML Design Document Inspection. *Proc. Int. Symposium on Empirical Software Engineering (ISESE)*, p. 148–157, 2002c.

Sabaliauskaite, G.; Matsukawa, F.; Kusumoto, S.; Inoue, K. Further investigations of reading techniques for object-oriented design inspection. *Information and Software Technology (INFSOF)*, v. 45, n. 9, p. 571–585, 2003.

Salinesi, C.; Mazo, R. Defects in Product Line Models and How to Identify Them. In: Elfaki, A. O., ed. *Software Product Line - Advanced Topic*, InTech, p. 27, 2012.

Santos, J. a.; Moreira, A.; Araújo, J. a.; Amaral, V.; Alf érez, M.; Kulesza, U. Generating Requirements Analysis Models from Textual Requirements. In: *Proc. Int. Workshop on Managing Requirements Knowledge*, IEEE, 2008, p. 32–41.

Sauer, C.; Jeffery, D. R.; Land, L.; Yetton, P. The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research. *IEEE Transactions on Software Engineering (TSE)*, v. 26, n. 1, p. 1–14, 2000.

Seaman, C. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering (TSE)*, v. 25, n. 4, p. 557–572, 1999.

SEI *Software Engineering Institute Arcade Game Maker (AGM) Pedagogical Product Line*. Carnegie Mellon University, 2009.
Disponível em `http://www.sei.cmu.edu/productlines/ppl/`

SEI *Software Engineering Institute A Framework for Software Product Line Practice - Version 5.0*. Carnegie Mellon University, 2012a.
Disponível em `http://www.sei.cmu.edu/productlines/frame_report/index.html`

SEI *Software Engineering Institute Hall of Fame*. Carnegie Mellon University, 2012b.
Disponível em `http://www.sei.cmu.edu/productlines/plp_hof.html`

Shapiro, S. S.; Wilk, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, v. 52, n. 3/4, p. 591–611, 1965.

Shull, F.; Mendonça, M. G.; Basili, V.; Carver, J.; Maldonado, J. C.; Fabbri, S.; Travassos, G. H.; Ferreira, M. C. Knowledge-Sharing Issues in Experimental Software Engineering. *Empirical Software Engineering*, v. 9, n. 1/2, p. 111–137, 2004.

Shull, F.; Rus, I.; Basili, V. How Perspective-Based Reading Can Improve Requirements Inspections. *IEEE Computer*, v. 33, n. 7, p. 73–79, 2000.

Simidchieva, B. I.; Clarke, L. A.; Osterweil, L. J. Representing Process Variation with a Process Family. In: *Proc. Int. Conf. Software Process (ICSP)*, Springer-Verlag, 2007, p. 109–120.

Sommerville, I. *Software Engineering*. 9th ed. Addison-Wesley, 790 p., 2011.

Souza, I. S.; da Silva Gomes, G. S.; da Mota Silveira Neto, P. A.; do Carmo Machado, I.; de Almeida, E. S.; de Lemos Meira, S. R. Evidence of software inspection on feature specification for software product lines. *Journal of Systems and Software (JSS)*, v. 86, n. 5, p. 1172–1190, 2013.

Spearman, C. The Proof and Measurement of Association Between Two Things. *The American Journal of Psychology*, v. 100, n. 3-4, p. 441–471, 1987.
Disponível em `http://view.ncbi.nlm.nih.gov/pubmed/3322052`

Staron, M.; Kuzniarz, L.; Thurn, C. An Empirical Assessment of Using Stereotypes to Improve Reading Techniques in Software Inspections. *Proc. on Workshop on Software Quality (WoSQ)*, p. 1–7, 2005.

Thelin, T.; Runeson, P.; Wohlin, C. Prioritized Use Cases as a Vehicle for Software Inspections. *IEEE Software*, v. 20, n. 4, p. 30–33, 2003.

Thurimella, A. K.; Bruegge, B. Issue-based variability management. *Information and Software Technology (INFSOF)*, v. 54, n. 9, p. 933–950, 2012.

Tørner, F.; Ivarsson, M.; Pettersson, F.; Öhman, P. Defects in Automotive Use Cases. In: *Proc. Int. Symposium on Empirical Software Engineering (ESEM - ISESE)*, New York, NY, USA: ACM, 2006, p. 115–123.

Travassos, G. H.; Shull, F.; Carver, J. Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language. *Advances in Computers*, v. 54, p. 35–98, 2001.

Travassos, G. H.; Shull, F. J.; Fredericks, M.; Basili, V. R. Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. In: *Proc. of the ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, New York, USA: ACM Press, 1999, p. 47–56.

Wagner, S. A Model and Sensitivity Analysis of the Quality Economics of Defect-detection Techniques. In: *Proc. Int. Symposium on Software Testing and Analysis (ISSTA)*, New York, NY, USA: ACM, 2006, p. 73–84.

Walia, G. S.; Carver, J. C. A systematic literature review to identify and classify software requirement errors. *Information and Software Technology (INFSOF)*, v. 51, n. 7, p. 1087–1109, 2009.

Weilkiens, T.; Oestereich, B. *Uml 2 certification guide: Fundamental & intermediate exams (the omg press).* San Francisco, CA, USA: Morgan Kaufmann Publishers, 2006.

Weiss, D. M.; Lai, C. T. R. *Software product-line engineering: A family-based software development process.* Boston, MA, USA: Addison-Wesley Longman Publishing, 1999.

Weller, E. F. Lessons from Three Years of Inspection Data (software development). *IEEE Software*, v. 10, n. 5, p. 38–45, 1993.

Wiegers, K. E. *More About Software Requirements: Thorny Issues and Practical Advice.* Microsoft Press, 219 p., 2006.

Wieringa, R.; Maiden, N.; Mead, N.; Rolland, C. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, v. 11, n. 1, p. 102–107, 2005.

Winkler, D.; Thurnher, B.; Biffl, S. Early Software Product Improvement with Sequential Inspection Sessions: an Empirical Investigation of Inspector Capability and Learning Effects. In: *Proc. Int. Conf. on Software Engineering and Advanced Applications*, 2007, p. 245–254.

Wohlin, C.; Runeson, P.; Host, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in Software Engineering*, v. 44. Springer, 248 p., 2012.

Yang, X. M. *Towards a Self-Evolving Software Defect Detection Process.* Dissertação de mestrado, University of Saskatchewan, 2007.

Young, T. J. *Using AspectJ to Build a Software Product Line for Mobile Devices.* Dissertação de mestrado, University of British Columbia, 2005.

Ziadi, T.; Loïc, H.; Jézéquel, J.-M. Towards a UML Profile for Software Product Lines. *Product Familly Engineering Conf.*, p. 129–139, 2004.

Apêndice A - Estudo Secundário sobre Tipos de Defeitos em Técnicas de Inspeção de Software

A.1 O Processo do Estudo Secundário

O Estudo Secundário realizado tem como objetivo fornecer aos pesquisadores uma visão geral dos estudos primários sobre tipos de defeitos em atividades de inspeção de software (Kitchenham *et al.*, 2010; Petersen *et al.*, 2008a). Pesquisas realizadas por meio de mapeamentos sistemáticos permitem reunir um grande número de estudos primários, pois não há restrições impostas, com exceção de prazo, quando aplicável.

O estudo de Petersen *et al.* (2008a), comparou diferentes métodos para a realização de revisões e mapeamentos sistemáticos, no qual estabeleceram cinco estágios como estratégia para elaboração e condução de mapeamentos sistemáticos:

- Definição do protocolo;

- Definição das questões de pesquisa;

- Condução da pesquisa para estudos primários;

- Elaboração e aplicação de critérios de inclusão/exclusão nos estudos primários;

- Classificação dos estudos/publicações;

- Estratégia de extração e agregação dos dados.

Os estágios de conduzidos para a realização deste estudo secundário estão de acordo com a Figura 1.1. Assim, esta seção descreve como cada estágio foi planejado e conduzido, os quais são: questões de pesquisa (Seção A.1.1), processo de pesquisa (Seção A.1.2),

fontes/bases de dados eletrônicas, palavras-chave e *strings* de busca (Seção A.1.3), critérios de inclusão e exclusão (Seção A.1.4), esquema de classificação (Seção A.1.5) e extração e agregação dos dados (Seção A.1.6).

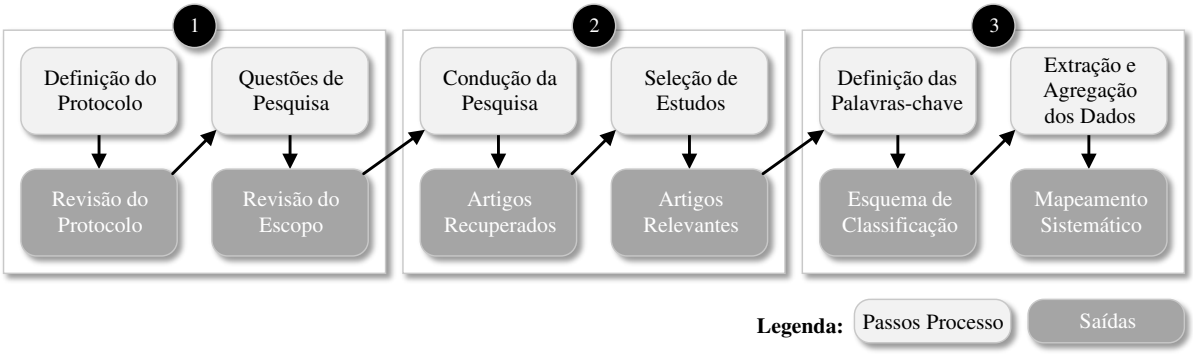


Figura 1.1: O Processo do Estudo Secundário seguido (adaptado de Petersen *et al.* (2008a)).

A.1.1 Definição das Questões de Pesquisa

Os principais objetivos deste estudo secundário foram estabelecidos com o objetivo de: (i) identificar na literatura os tipos de defeitos utilizados em técnicas de inspeção de software; e (ii) apresentar ao leitor uma visão geral dos estudos primários e fornecer evidências da existência de técnicas/abordagens de inspeção de software. Portanto, as seguintes questões de pesquisa (QP1 e QP2) foram definidas:

- ***Questão de Pesquisa (QP1)***. *Quais são os tipos de defeitos que têm sido considerados por técnicas de inspeção de software?*

- ***Questão de Pesquisa (QP2)***. *Quais estudos fornecem evidências sobre a existência de técnicas/abordagens de inspeção de software?*

A.1.2 O Processo de Pesquisa

O processo de pesquisa seguido baseia-se em critérios e diretrizes propostas por Kitchenham (2007); Kitchenham *et al.* (2010) e Petersen *et al.* (2008a) com relação a realização de mapeamentos sistemáticos. Em adição, o corpo de conhecimento foi constituído a partir de vários estudos de diferentes áreas, tal como Barney *et al.* (2012), Mota Silveira Neto *et al.* (2011), Mohabbati *et al.* (2013) e Novais *et al.* (2013), servindo como base para fornecer direções de como conduzir este estudo secundário. Portanto, os seguintes

procedimentos (Figura 1.2) para o processo de pesquisa foram estabelecidos de acordo com:

- a seleção de fontes/bases de dados eletrônicas relevantes e mecanismos de busca indexados (Seção A.1.3);
- foram definidas palavras-chave relevantes para compor a *query* principal e as *strings* de busca aplicadas nas fontes/bases de dados eletrônicas e mecanismos para recuperar estudos primários. As seguintes palavras-chave foram utilizadas: "software inspection" and "defect type" (Seção A.1.3);
- a composição dessas palavras-chave e suas variações, foi estabelecida utilizando os conectivos lógicos "AND" e "OR" (Seção A.1.3);
- a aplicação das *strings* de busca definidas para as fontes/bases de dados eletrônicas e nos mecanismos de busca indexados. Uma lista com um número considerável de estudos primários foi recuperada. A partir disso, critérios de inclusão e exclusão foram aplicados para filtar e selecionar os estudos mais relevantes para este estudo secundário. Além disso, estudos duplicados e potenciais conflitos, por exemplo, foram revisados para gerar uma lista atualizada (Seção A.1.4);
- a definição de um esquema de classificação baseado em categorias (Seção A.1.5); e
- a extração e agregação dos dados (Seção A.1.6) por meio de técnicas de visualização (gráficos, *bubble plots*, etc.) a fim de apresentar os resultados obtidos. Assim, uma breve discussão foi feita sobre os assuntos relacionados a este SM (Seção A.2).

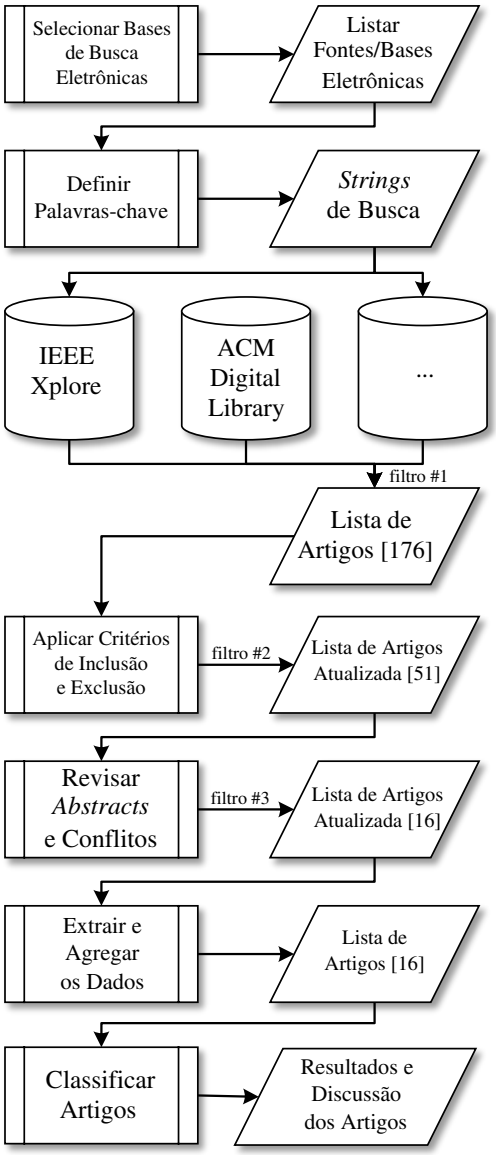


Figura 1.2: O Processo de Pesquisa (adaptado de Barney *et al.* (2012)).

A.1.3 Definição das Fontes/Bases Eletrônicas, Palavras-Chave e *Strings* de Busca

Uma vez que as questões de pesquisa foram definidas, o próximo passo foi definir as fontes/bases de dados eletrônicas e mecanismos de busca indexados para permitir a busca e identificação de estudos primários relevantes. Portanto, o seguinte conjunto de fontes/bases de dados eletrônicas utilizados neste estudo secundário foram:

- IEEE Xplore[1] - fonte para a descoberta e acesso a conteúdos científicos e técnicos publicados pelo *Institute of Electrical and Electronics Engineers* (IEEE) e seus parceiros de publicação;

- ACM Digital Library[2] - artigos publicados pela ACM e citações bibliográficas das principais editoras em computação;

- Elsevier ScienceDirect[3] - revisão de texto científico completo revisado por pares, além de conteúdo técnico e médico; e

- Google Scholar[4] - amplo mecanismo de busca de artigos, relatórios técnicos, dissertações e teses.

O próximo passo foi definir as palavras-chave apropriadas para construir as *strings* de busca com a finalidade de estabelecer uma *query* de busca geral. Esta *query* pode ser adaptada para cada busca, de modo iterativo, na qual foi aplicada nas fontes/bases de dados eletrônicas. Assim, as palavras-chave foram aplicadas por meio de cada *query* de busca, na qual foram modificadas de acordo com cada mecanismo de busca presente em cada fonte/base de dados eletrônica.

A Tabela 1.1 apresenta os filtros aplicados com base nos estudos primários recuperados e selecionados. Adicionalmente, a Figura 1.3 ilustra os filtros em detalhes, bem como o número de estudos primários selecionados de acordo com a atividade de cada filtro. Já, a Tabela 1.2 apresenta uma lista das *strings* de busca e suas respectivas palavras-chave.

Tabela 1.1: Número de Estudos por Filtro e Fontes/Bases de Dados Eletrônicas.

Fontes/Bases de Dados Eletrônicas	**Filtro #1**	**Filtro #2**	**Filtro #3**
ACM Digital Library	27	16	5
ELSEVIER ScienceDirect	14	11	2
Google Scholar	105	7	3
IEEE Xplore	30	17	6
Total	**176**	**51**	**16**

[1] `ieeexplore.ieee.org/Xplore`
[2] `http://dl.acm.org`
[3] `http://www.elsevier.com/online-tools/sciencedirect`
[4] `http://scholar.google.com`

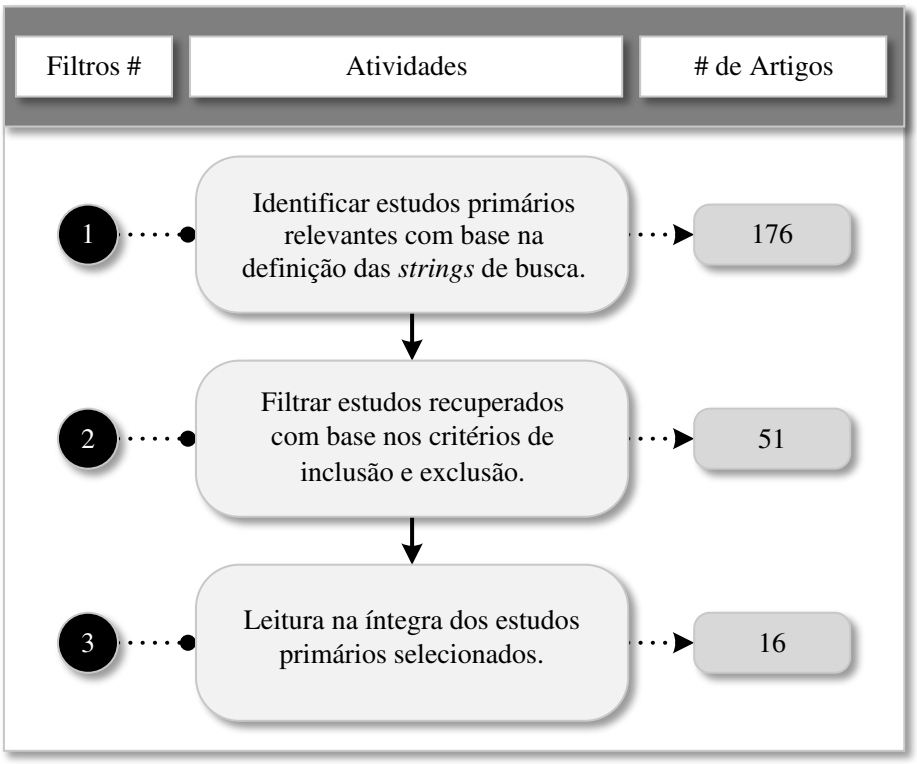


Figura 1.3: Estágios de Busca dos Estudos Primários (adaptado dos estudos de Mota Silveira Neto *et al.* (2011) e Mohabbati *et al.* (2013)).

Tabela 1.2: Palavras-chave e *Strings* de Busca da *Query* de Busca.

***Query* de Busca (QB)**
(‘‘software’’) AND ((‘‘inspection’’ AND (‘‘technique’’ OR ‘‘activity’’ OR ‘‘strategy’’)) OR ((‘‘defect type’’ OR ‘‘type of defects’’ OR ‘‘defect detection’’ OR ‘‘requirements defect’’ OR ‘‘fault detection’’)))

A.1.4 Definição dos Critérios de Inclusão e Exclusão

Com o objetivo de selecionar os estudos mais relevantes e contribuir para responder as questões de pesquisa deste estudo secundário, os seguintes critérios de inclusão e exclusão foram definidos:

Critérios de Inclusão. Para cada questão de pesquisa, QP1 e QP2, foram definidos os seguintes critérios de inclusão:

- ***QP1***: estudos que apresentam tipos de defeitos relacionados com técnicas de inspeção de software;

- ***QP2***: estudos que propõem técnicas de inspeção ou abordagens.

Critérios de Exclusão. Para cada questão de pesquisa, QP1 e QP2, foram definidos os seguintes critérios de exclusão:

- ***QP1***: estudos que não apresentam tipos de defeitos relacionados com técnicas de inspeção de software;

- ***QP2***: estudos que não propõem técnicas de inspeção ou abordagens.

Em adição, outros critérios de exclusão foram estabelecidos:

- estudos em diferentes idiomas, além do Inglês;
- estudos, que não estão em um dos seguintes formatos de arquivo: PDF, DOC or ODT;
- estudos duplicados, por exemplo, estudos recuperados a partir de mais de uma das fontes/bases de dados eletrônicas definidas;
- estudos indisponíveis, por exemplo, uma URL indisponível; e
- estudos com menos de quatro páginas.

As buscas nas fontes/bases de dados eletrônicas foram realizadas utilizando a *query* de busca com suas respectivas palavras-chave (Filtro #1), como apresentado na Seção A.1.3. Em seguida, uma seleção preliminar dos estudos foi realizada por meio da leitura do título, *abstract*, introdução e conclusão de cada estudo recuperado, bem como os critérios de inclusão e exclusão foram aplicados (Filtro #2). Portanto, os estudos primários selecionados foram lidos na íntegra, de acordo com o Filtro #3 (Figura 1.3), no intuito de classificá-los (Seção A.1.5), bem como para proporcionar uma análise construtiva (Seção A.2.4).

A.1.5 Esquema de Classificação

Para classificar os tipos de pesquisa deste estudo secundário, o método de classificação de Wieringa *et al.* (2005) foi adotado. Tal método é sugerido por Petersen *et al.* (2008a), pois tem uma estrutura de classificação bem definida.

O método de classificação proposto por Wieringa *et al.* (2005) permite a classificação dos tipos de pesquisa por meio de seis categorias:

- Pesquisa de Validação: permite avaliar técnicas geralmente realizadas em um ambiente acadêmico. De acordo com Wieringa *et al.* (2005), os métodos utilizados para avaliar diferentes pesquisas são: experimentos, simulações, construções de protótipos e análises matemáticas;

- Pesquisa de Avaliação: avalia técnicas geralmente realizadas na indústria. De acordo com Wieringa *et al.* (2005), esta categoria está focada em avaliar uma pesquisa, um problema de pesquisa ou a aplicação prática de uma técnica;

- Proposta de Solução: de acordo com Wieringa *et al.* (2005), esta categoria se concentra em propostas de novas técnicas e/ou revisões de técnicas baseadas no problema de pesquisa;

- Estudos Filosóficos: esta categoria tem o objetivo de apresentar novos conceitos que podem ser explorados em pesquisas;

- Estudos de Opinião: apresenta opiniões positivas ou negativas do autor sobre determinadas técnicas, experimentos, estudos de caso, dentre outros;

- Estudos de Experiência: apresenta as experiências do autor com relação a determinada pesquisa verificada na prática.

A classificação dos estudos recuperados foi feita por meio da leitura do abstract, da introdução e da conclusão de cada estudo recuperado, bem como foi feita a leitura de alguns trechos, a fim de certificar-se a categoria adequada, na qual cada estudo realmente pertence. A Figura 1.4 ilustra tal esquema de classificação.

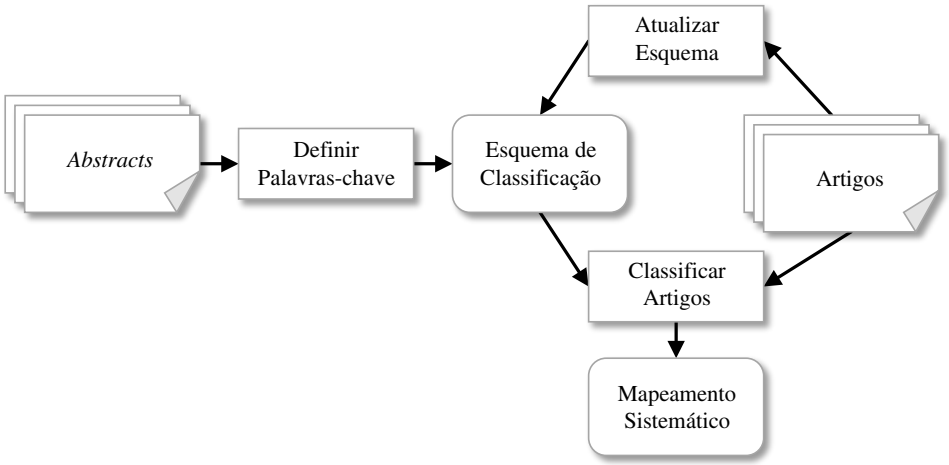


Figura 1.4: Esquema de Classificação (adaptado de Petersen *et al.* (2008a)).

Complementarmente, a Seção A.2.4 apresenta o escopo de cada estudo lido na íntegra.

A.1.6 Extração e Agregação dos Dados

A extração dos dados resume os dados relevantes com relação ao conjunto final dos estudos primários selecionados (Bailey *et al.*, 2007). Portanto, as seguintes informações, consideradas essenciais, foram extraídas de cada estudo:

- **Fontes/Bases de dados eletrônicas**: IEEE Xplore, ACM Digital Library e ELSEVIER ScienceDirect, bem como o mecanimo de busca eletrônico Google Scholar;
- **Título**;
- **Autores e afiliações**: a Figura 1.10 e a Figura 1.11 apresentam os principais autores;
- **Ano de publicação**: foi considerado o intervalo entre 1996 até Março/2014. Em um projeto piloto realizado, o estudo mais antigo era datado do ano de 1996;
- **Tipo de publicação**: evento, Periódico, *Workshop*, Capítulo de Livro, Dissertação de Mestrado, Tese de Doutorado, Relatório Técnico;
- **Acrônimo do tipo de publicação**.

A ferramenta Mendeley (Mendeley, 2015) foi adotada para proporcionar uma melhor organização das referências bibliográficas[5].

A.2 Discussão e Resultados do Estudo Secundário

Esta seção fornece uma discussão com relação aos resultados obtidos neste estudo secundário. Assim, a Seção A.2.1 apresenta uma visão geral dos estudos recuperados com base nos resultados a partir da aplicação do filtro #1 (Figura 1.2), a Seção A.2.2 fornece represetações gráficas dos estudos a partir do filtro #2, e as Seções A.2.3 e A.2.4, apresentam discussões acerca dos estudos mais relevantes a partir do filtro #3.

A.2.1 Visão Geral do Estudo Secundário

Este estudo secundário foi realizado entre Fevereiro até Abril de 2014. De acordo com este período foram obtidos um total de 176 estudos primários por meio da aplicação das *strings* de busca com base nas fontes/bases de dados eletrônicas definidas.

[5]Um pacote deste estudo contendo um arquivo bibtex e o conjunto dos estudos selecionados por fonte está disponível em: `http://www.din.uem.br/~smarty/SMartyCheck/SM-Defect-Types/SM_Defect_Types_2014.rar`

Por meio da utilização do mecanimo de busca eletrônico Google Scholar, foram recuperados 105 estudos, incluindo estudos duplicados, nos quais foram recuperados também na fonte/base IEEE Xplore e ACM Digital Library. Tal mecanismo de busca (Google Scholar) representa cerca de 60% do total de estudos primários recuperados. No entanto, alguns poucos estudos do Google Scholar foram selecionados para a leitura na íntegra.

Na busca realizada na IEEE Xplore foram recuperados 30 (17%) estudos, enquanto que na ACM Digital Library foram recuperados 27 (15%) estudos. Em adição, na ELSEVIER ScienceDirect foram recuperados 14 (8%) estudos.

A Figura 1.5 apresenta a distribuição total de estudos primários recuperados por fonte/base de dados eletrônica.

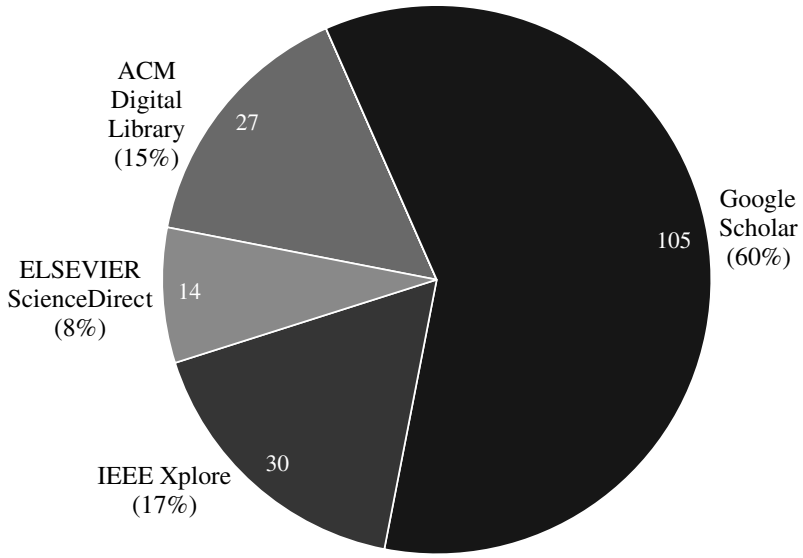


Figura 1.5: Total de Estudos Primários por Fonte/Base de Dados Eletrônica.

A.2.2 Metadados dos Estudos Primários

Esta seção discute os resultados obtidos com base no filtro #2 a partir dos 51 estudos, apresentados na Tabela 1.3.

Tabela 1.3: Estudos Recuperados com base nos Critérios de Inclusão e Exclusão (filtro #2).

Ord.	Autor(es)	Título	Ano	Tipo de Publicação	Local de Publicação	Fonte/Base de Dados Eletrônica	Tipo de Pesquisa
1	Cox *et al.* (2004b)	A Use Case Description Inspection Experiment	2004	Capítulo de Livro	N/A	Google Scholar	Pesquisa de Validação
2	Travassos *et al.* (2001)	Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language	2001	Capítulo de Livro	Advances in Computers	ELSEVIER ScienceDirect	Proposta de Solução
3	Amoui *et al.* (2013)	Search-Based Duplicate Defect Detection: An Industrial Experience	2013	Evento	MSR	ACM Digital Library	Pesquisa de Avaliação
4	Mello *et al.* (2012)	Checklist-based Inspection Technique for Feature Models Review	2012	Evento	SBCARS	IEEE Xplore	Proposta de Solução
5	Cunha *et al.* (2012)	A Set of Inspection Technique on Software Product Line Models	2012	Evento	SEKE	Google Scholar	Proposta de Solução
6	Mello *et al.* (2010)	Activity Diagram Inspection on Requirements Specification	2010	Evento	SBES	IEEE Xplore	Proposta de Solução
7	Chen *et al.* (2009)	Variability Management in Software Product Lines: A Systematic Review	2009	Evento	SPLC	ACM Digital Library	Pesquisa de Validação
8	Petersen *et al.* (2008b)	The Impact of Time Controlled Reading on Software Inspection Effectiveness and Efficiency: A Controlled Experiment	2008	Evento	ESEM	ACM Digital Library	Pesquisa de Validação
9	Simidchieva *et al.* (2007)	Representing Process Variation with a Process Family	2007	Evento	ICSP	Google Scholar	Pesquisa de Avaliação
10	Winkler *et al.* (2007)	Early Software Product Improvement with Sequential Inspection Sessions: An Empirical Investigation of Inspector Capability and Learning Effects	2007	Evento	SEAA (EUROMICRO)	IEEE Xplore	Pesquisa de Validação
11	Tørner *et al.* (2006)	Defects in Automotive Use Cases	2006	Evento	ESEM (ISESE)	ACM Digital Library	Pesquisa de Avaliação
12	He e Carver (2006)	PBR vs. Checklist: A Replication in the N-Fold Inspection Context	2006	Evento	ESEM (ISESE)	ACM Digital Library	Pesquisa de Validação
13	Lange e Chaudron (2006)	Effects of Defects in UML Models - An Experimental Investigation	2006	Evento	ICSE	ACM Digital Library	Pesquisa de Validação
14	Wagner (2006)	A Model and Sensitivity Analysis of the Quality Economics of Defect-Detection Techniques	2006	Evento	ISSTA	ACM Digital Library	Proposta de Solução
15	Cooper *et al.* (2005)	Experiences Using Defect Checklists in Software Engineering Education	2005	Evento	CAINE (ISCA)	Google Scholar	Estudo de Experiência
16	Belgamo *et al.* (2005)	TUCCA Improving the Effectiveness of Use Case Construction and Requirement Analysis	2005	Evento	ESEM (ISESE)	IEEE Xplore	Proposta de Solução
17	Staron *et al.* (2005)	An Empirical Assessment of Using Stereotypes to Improve Reading Techniques in Software Inspections	2005	Evento	WoSQ	ACM Digital Library	Pesquisa de Validação
18	Denger *et al.* (2004)	Investigating the Active Guidance Factor in Reading Techniques for Defect Detection	2004	Evento	ESEM (ISESE)	IEEE Xplore	Pesquisa de Validação
19	Lanubile *et al.* (2004)	Assessing the Impact of Active Guidance for Defect Detection: A Replicated Experiment	2004	Evento	ESEM (METRIC)	IEEE Xplore	Pesquisa de Validação
20	Denger e Paech (2004)	An Integrated Quality Assurance Approach for Use Case Based Requirements	2004	Evento	LNI, GI	Google Scholar	Proposta de Solução
21	Grunbacher *et al.* (2003)	An Empirical Study on Groupware Support for Software Inspection Meetings	2003	Evento	ASE	IEEE Xplore	Pesquisa de Validação
22	Kelly e Shepard (2003)	An Experiment to Investigate Interacting versus Nominal Groups in Software Inspection	2003	Evento	CASCON	ACM Digital Library	Pesquisa de Validação
23	Miller e Yin (2003)	Adding Diversity to Software Inspections	2003	Evento	ICCI	IEEE Xplore	Proposta de Solução
24	Sabaliauskaite *et al.* (2002b)	An Experimental Comparison of Checklist-Based Reading and Perspective-Based Reading for UML Design Document Inspection	2002	Evento	ESEM	IEEE Xplore	Pesquisa de Validação
25	Sabaliauskaite *et al.* (2002c)	An Experimental Comparison of Checklist-Based Reading and Perspective-Based Reading for UML Design Document Inspection	2002	Evento	ISESE	ACM Digital Library	Pesquisa de Validação
26	Anda e Sjøberg (2002)	Towards an Inspection Technique for Use Case Models	2002	Evento	SEKE	ACM Digital Library	Proposta de Solução
27	Kelly e Shepard (2001)	A Case Study in the Use of Defect Classification in Inspections	2001	Evento	CASCON	ACM Digital Library	Pesquisa de Avaliação
28	Freimut *et al.* (2001)	Investigating the Impact of Reading Techniques on the Accuracy of Different Defect Content Estimation Techniques	2001	Evento	ESEM (METRIC)	IEEE Xplore	Pesquisa de Validação
29	Biffl *et al.* (2001)	Investigating the Cost-Effectiveness of Reinspections in Software Development	2001	Evento	ICSE	ACM Digital Library	Proposta de Solução
30	Biffl e Halling (2000)	Software Product Improvement with Inspection - A Large-scale Experiment on the Influence of Inspection Processes on Defect Detection in Software Requirements Documents	2000	Evento	EUROMICRO	IEEE Xplore	Pesquisa de Validação
31	Travassos *et al.* (1999)	Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality	1999	Evento	OOPSLA	ACM Digital Library	Proposta de Solução
32	Cheng e Jeffery (1996)	Comparing Inspection Strategies for Software Requirement Specifications	1996	Evento	ASWEC	IEEE Xplore	Pesquisa de Validação
33	Mishra e Mishra (2009)	Simplified software inspection process in compliance with international standards	2009	Periódico	CSI	ELSEVIER ScienceDirect	Proposta de Solução
34	Walia e Carver (2009)	A systematic literature review to identify and classify software requirement errors	2009	Periódico	INFSOF (IST)	ELSEVIER ScienceDirect	Pesquisa de Validação
35	Munson *et al.* (2006)	Software faults: A quantifiable definition	2006	Periódico	ADVENGSOFT	ELSEVIER ScienceDirect	Pesquisa de Avaliação
36	Sabaliauskaite *et al.* (2004)	Assessing defect detection performance of interacting teams in object-oriented design inspection	2004	Periódico	INFSOF (IST)	ELSEVIER ScienceDirect	Pesquisa de Validação
37	Cox *et al.* (2004a)	An Experiment in Inspecting the Quality of Use Case Descriptions	2004	Periódico	JRPIT	Google Scholar	Pesquisa de Validação
38	Hungerford *et al.* (2004)	Reviewing Software Diagrams: A Cognitive Study	2004	Periódico	TSE	IEEE Xplore	Pesquisa de Validação
39	Thelin *et al.* (2003)	Prioritized Use Cases as a Vehicle for Software Inspections	2003	Periódico	IEEE Software	IEEE Xplore	Pesquisa de Validação
40	Ciolkowski *et al.* (2003)	Software Reviews: The State of the Practice	2003	Periódico	IEEE Software	IEEE Xplore	Pesquisa de Avaliação
41	Sabaliauskaite *et al.* (2003)	Further investigations of reading techniques for object-oriented design inspection	2003	Periódico	INFSOF (IST)	ELSEVIER ScienceDirect	Pesquisa de Avaliação
42	Biffl (2003)	Evaluating defect estimation models with major defects	2003	Periódico	JSS	ELSEVIER ScienceDirect	Pesquisa de Avaliação
43	Biffl e Halling (2003)	Investigating the Defect Detection Effectiveness and Cost Benefit of Nominal Inspection Teams	2003	Periódico	TSE	IEEE Xplore	Pesquisa de Validação
44	Laitenberger *et al.* (2001)	An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-Based Reading of Code Documents	2001	Periódico	TSE	IEEE Xplore	Pesquisa de Validação
45	Laitenberger e DeBaud (2000)	An encompassing life cycle centric survey of software inspection	2000	Periódico	JSS	ELSEVIER ScienceDirect	Pesquisa de Validação
46	Laitenberger *et al.* (2000)	An experimental comparison of reading techniques for defect detection in UML design documents	2000	Periódico	JSS	ELSEVIER ScienceDirect	Pesquisa de Validação
47	Dunsmore *et al.* (2000)	The role of comprehension in software inspection	2000	Periódico	JSS	ELSEVIER ScienceDirect	Pesquisa de Validação
48	Brykczynski (1999)	A Survey of Software Inspection Checklists	1999	Periódico	ACM SIGSOFT	ACM Digital Library	Pesquisa de Avaliação
49	Porter *et al.* (1998)	Understanding the Sources of Variation in Software Inspections	1998	Periódico	ACM TOSEM	ACM Digital Library	Pesquisa de Validação
50	Miller *et al.* (1998)	Further Experiences with Scenarios and Checklists	1998	Periódico	Empirical Software Engineering	Google Scholar	Pesquisa de Avaliação
51	Roper *et al.* (1997)	An empirical evaluation of defect detection techniques	1997	Periódico	INFSOF (IST)	ELSEVIER ScienceDirect	Pesquisa de Validação

Locais de Publicação

Os estudos publicados referente as áreas investigadas neste estudo secundário, abrangem os tipos de defeitos e técnicas de inspeção/abordagens, no qual as publicações foram identificados em 28 locais diferentes. A Figura 1.6 apresenta uma *word cloud* (nuvem de palavras ou nuvem de *tags*) de acordo com a distribuição das publicações nas conferências. Deste modo, foi possível observar na Figura 1.6, que há uma diferença significativa entre o número de estudos publicados no evento Empirical Software Engineering and Measurement (ESEM) (9 studies) em comparação com os demais eventos. Assim, a Tabela 1.4 apresenta os acrônimos e os respectivos nomes dos eventos.

Figura 1.6: *Word Cloud* da Distribuição dos Estudos por Evento.

Tabela 1.4: Lista de Eventos dos Estudos Selecionados

Acrônimo	Nome do Evento
ESEM	Empirical Software Engineering and Measurement
EUROMICRO	Euromicro Conference
ISESE	International Symposium on Empirical Software Engineering
OOPSLA	Object-Oriented Programming, Systems, Languages and Applications
SBCARS	Brazilian Symposium on Software Components, Architectures and Reuse
SBES	Brazilian Symposium on Software Engineering
SEKE	International Conference on Software Engineering and Knowledge Engineering
WoSQ	Workshop on Software Quality

Na sequência, analisando a Figura 1.7, foi possível observar que a maioria dos estudos publicados em periódicos estão concentrados nos seguintes periódicos:

- 4 estudos na Information and Software Technology (INFSOF/IST);
- 4 estudos no Journal of Systems and Software (JSS);
- 3 estudos na IEEE Transactions on Software Engineering (TSE); e
- 2 estudos na IEEE Software.

Figura 1.7: *Word Cloud* da Distribuição dos Estudos por Periódico.

A Tabela 1.5 apresenta os acrônimos e os respectivos nomes dos periódicos.

Tabela 1.5: Lista de Periódicos dos Estudos Selecionados

Acrônimo	**Nome dos Periódicos**
ACM SIGSOFT	ACM Special Interest Group on Software Engineering
ADVENGSOFT	ELSEVIER Advances in Engineering Software
INFSOF/IST	ELSEVIER Information and Software Technology
IEEE Software	
TSE	IEEE Transactions on Software Engineering
JRPIT	Journal of Research and Practice in Information Technology

A Figura 1.8 apresenta o número de publicações por local com base em eventos, periódicos e capítulos de livros. Assim, a maioria dos estudos selecionados foram publicados em eventos (30 estudos), seguido por publicações em periódicos (19 estudos), além de capítulos de livros (2 estudos), totalizando 51 estudos. Isso ocorre principalmente devido a peculiaridade da área de Ciência da Computação, em que os eventos são essenciais para fornecer evidências de teorias inovadoras, práticas, além do compartilhamento e da troca de conhecimento.

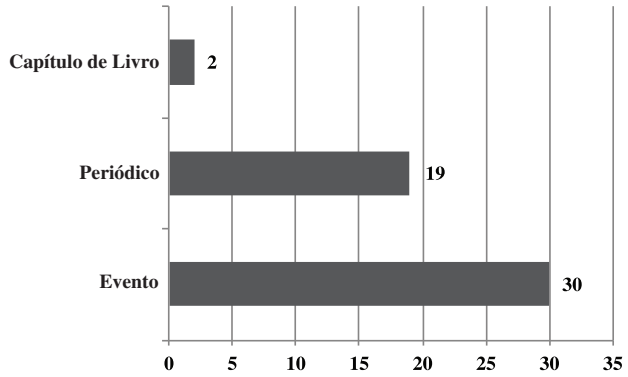


Figura 1.8: Número de Estudos por Tipo de Local.

Anos de Publicação e Principais Autores dos Estudos

Apesar da área de inspeção de software não ser uma área de pesquisa nova, estudos relevantes surgiram nos últimos anos. A Figura 1.9 fornece uma ilustração cronológica dos estudos primários entre o período de 1996 até 2013.

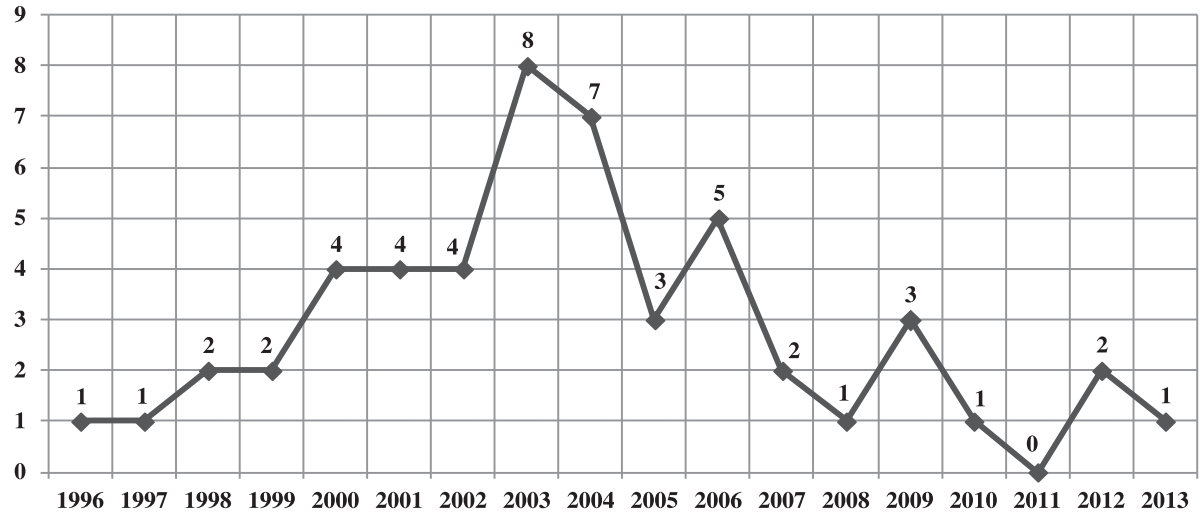


Figura 1.9: Distribuição Cronológica dos Estudos Primários por Ano de Publicação.

Portanto, levando em consideração a Figura 1.9, o ano de 2003 é de fato promissor para a área de inspeção de software. Além disso, foi possível descobrir que a partir de 2000 até 2006 essa área de pesquisa cresceu em termos de técnicas de inspeção em geral e novos tipos de defeitos. Em adição, desde 2009, provavelmente devido ao uso do emergente e *de facto* modelo baseado (*model-based*) por técnicas de reutilização de software, como Linhas de Produto de Software (LPS) e o Test-Driven Development (TDD) (Desenvolvimento Orientado a Testes) (veja na Seção A.2.4), a área de inspeção de software cresceu em seu número de publicações.

Por sua vez, foram mapeados os autores mais influentes da área de inspeção de software por meio de duas classificações: Top 30 (Figura 1.10) e Top 10 (Figura 1.11).

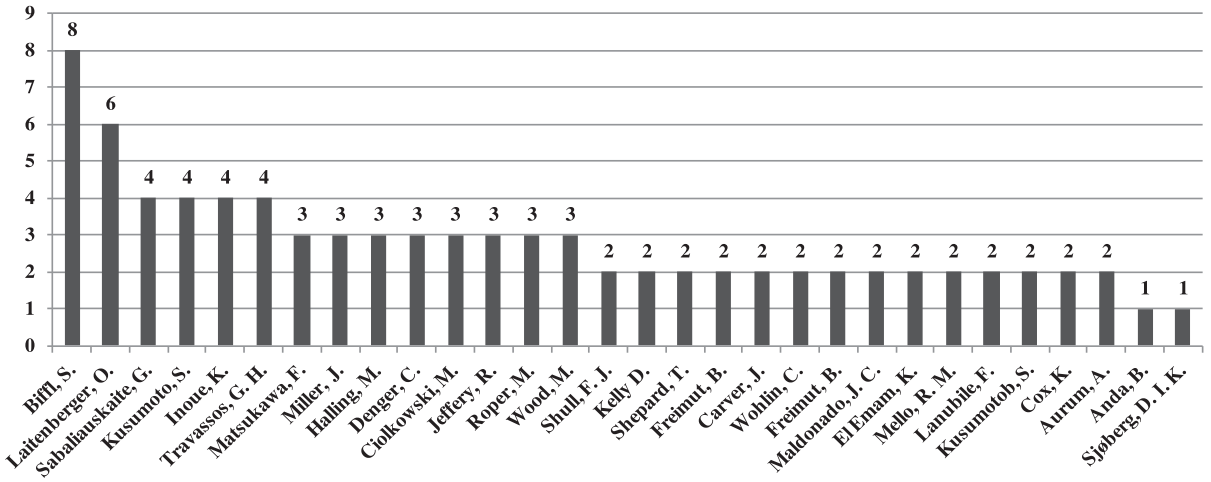


Figura 1.10: Autores Top 30 (filtro #1).

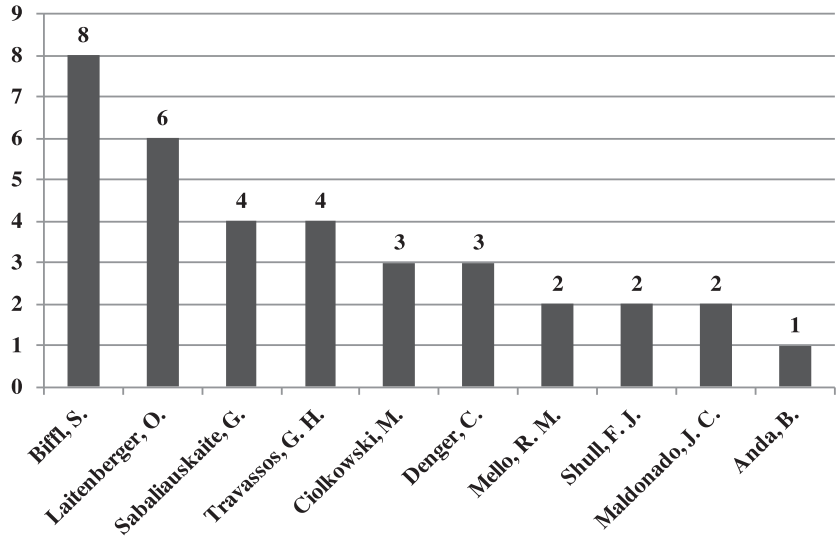


Figura 1.11: Autores Top 10 (filtro #2).

Os 30 autores mais influentes foram classificados após a aplicação do filtro #2 e antes da aplicação do filtro #3 (Figura 1.3) a fim de proporcionar uma visão geral de tais autores. Em seguida, após a aplicação do filtro #3, os 10 autores mais influentes das publicações mais relevantes para este estudo secundário foram identificados, permitindo assim, a classificação dos autores Top 10.

Analisando a Figura 1.10, Biffl possui 8 publicações em comparação com 6 de Laitenberger. Tais autores têm um alto número de publicações na área de inspeção de software. Por outro lado, autores, tais como Sabaliauskaite, Kusumoto e Inoue têm 4 publicações conjuntas. Em adição, Travassos possui 4 publicações relevantes para essa área.

Ademais, considerando os 16 estudos mais relevantes para este estudo secundário e analisando a Figura 1.11, os autores Biffl, Laitenberger, Sabaliauskaite e Travassos têm o mesmo número de publicações. Ademais, a classificação dos autores Top 10 também considera a proposta de novas taxonomias e classificações de tipos de defeitos, bem como estudos experimentais que fornecem um corpo de conhecimento sobre tais taxonomias e tipos de defeitos.

A.2.3 Discussão dos Principais Estudos Selecionados

As questões de pesquisa deste estudo secundário foram respondidas por meio dos 16 estudos primários com base na aplicação do filtro #3.

As próximas seções analisam os estudos mais relevantes com relação as fontes/bases de dados eletrônicas, tipos de pesquisa, questões de pesquisa (QPs), tipos de defeitos e técnicas de inspeção de software.

Fontes/Bases de Dados Eletrônicas x QPs x Tipos de Pesquisa

Aplicando o filtro #3 (Figura 1.3), foi obtido um total de 16 estudos relevantes. Tal filtro permitiu responder as questões de pesquisa QP1 e QP2 com base nas fontes/bases de dados eletrônicas e tipos de pesquisa (Figura 1.12).

O número de estudos a partir das fontes ACM Digital Library, IEEE Xplore, ELSEVIER ScienceDirect e Google Scholar está relacionado a responder a cada questão de pesquisa (QP1 e QP2). Em adição, é possível visualizar os tipos de pesquisa dos estudos primários de acordo com cada questão de pesquisa.

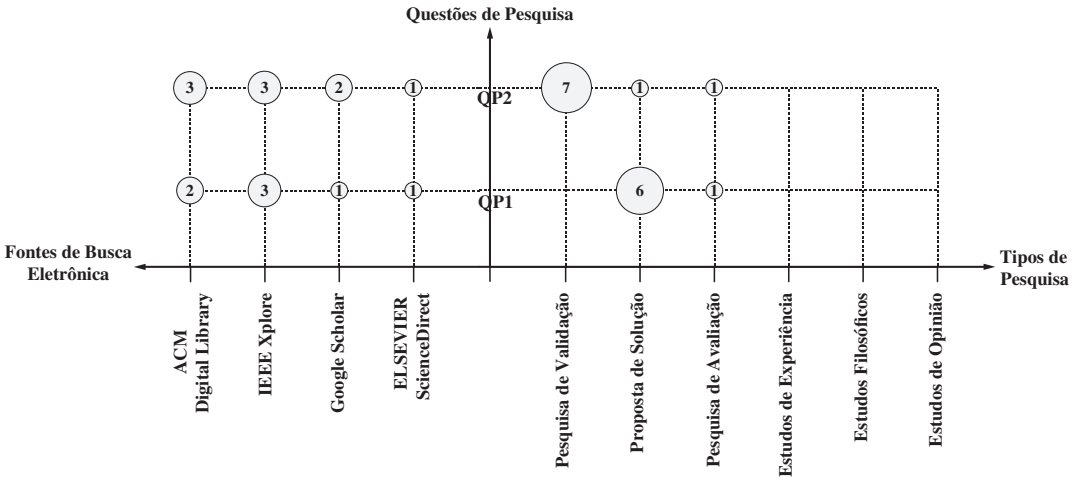


Figura 1.12: Questões de Pesquisa (QP1 e QP2) Respondidas com base nos Estudos Selecionados por Fontes/Bases de Dados Eletrônicas e Tipos de Pesquisa.

Além disso, considerando a Figura 1.12, os estudos selecionados mais relevantes estão nas fontes ACM Digital Library e IEEE Xplore, com 5 e 6 estudos, respectivamente. Já, a IEEE Xplore possui 3 estudos que responderam a questão de pesquisa QP1, sendo a maior parte classificada como Proposta de Solução. Adicionalmente, a questão de pesquisa QP2 foi respondida por meio dos 3 estudos com base na fonte ACM Digital Library, os quais foram classificados como Pesquisa de Validação.

Por conseguinte, 2 estudos da ELSEVIER ScienceDirect responderam as questões de pesquisa (QPs): 1 para a QP1 e 1 para a QP2. Ademais, 3 estudos do mecanismo de busca Google Scholar responderam as questões de pesquisa (QPs): 2 para a QP2 e 1 para a QP1.

Analisando de forma geral, a IEEE Xplore e a ACM Digital Library forneceram a maior parte de estudos relevantes sobre tipos de defeitos, enquanto que a IEEE Xplore, a ACM Digital Library e o Google Scholar permitiram identificar as técnicas de inspeção mais relevantes.

Já, a classificação denominada Pesquisa de Validação, permitiu evidenciar os tipos de defeitos existentes e as técnicas de inspeção identificadas neste estudo secundário.

Tipos de Defeitos x QPs x Tipos de Pesquisa

A Figura 1.13 apresenta os tipos de defeitos identificados, os quais foram classificados de acordo com os respectivos tipos de pesquisa. Assim, foi possível observar que a QP1 foi respondida com base na identificação de um número igual (3) de estudos primários para cada tipo de defeito. Em contraste, a QP2 foi respondida levando em consideração o alto número (7) de tipos de pesquisa, sendo a maior parte dos estudos classificados como Pesquisa de Validação, bem como vários estudos sobre técnicas de inspeção/abordagens.

Adicionalmente, com base na Figura 1.13, a engenharia de requisitos forneceu meios para identificar e adaptar tipos de defeitos a partir de várias estudos de diferentes técnicas/abordagens. Além disso, a maioria dos estudos possuem normas da IEEE como base, tais como a norma IEEE (1998a). Estas normas contém práticas recomendadas caracterizadas pela definição de um documento de requisitos bem definido, proporcionando assim, um meio para propor taxonomias de tipos de defeitos (see Seção A.2.4).

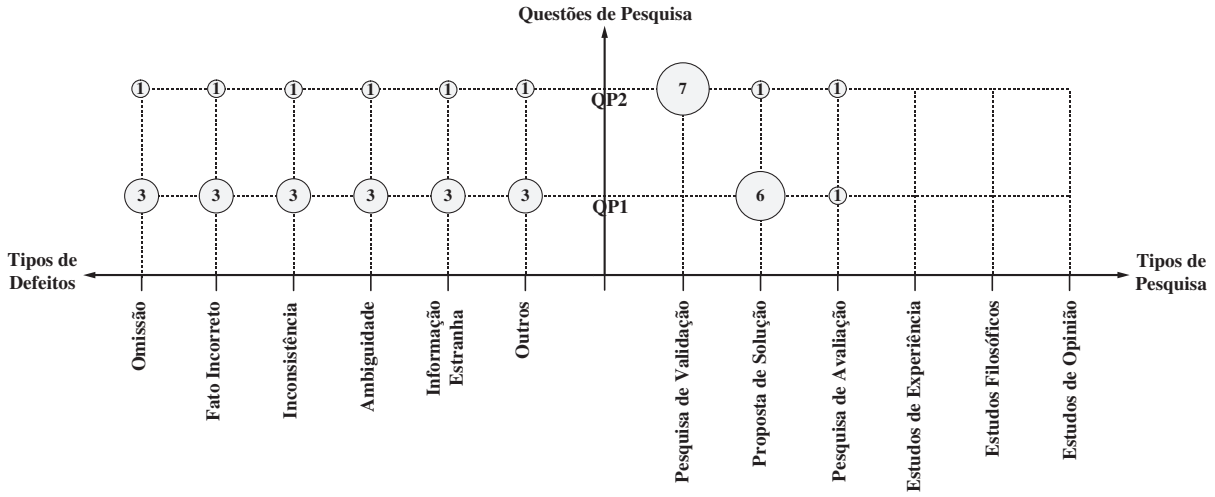


Figura 1.13: Questões de Pesquisa (QP1 e QP2) Respondidas com base nos Estudos Selecionados por Tipos de Defeitos e Tipos de Pesquisa.

Logo, a *word cloud* ilustrada na Figura 1.14, destaca os tipos de defeitos mais freqüentes identificados neste estudo secundário, tais como, Inconsistência e Fato Incorreto. Entretanto, os tipos de defeitos Ambiguidade e Omissão, têm praticamente o mesmo número de ocorrências. Ao contrário, o tipo de defeito Informação Estranha, quase não aparece em comparação com os principais tipos de defeitos. Ademais, os outros tipos de defeitos mal aparecem em taxonomias propostas dos estudos mais relevantes recuperados neste estudo secundário.

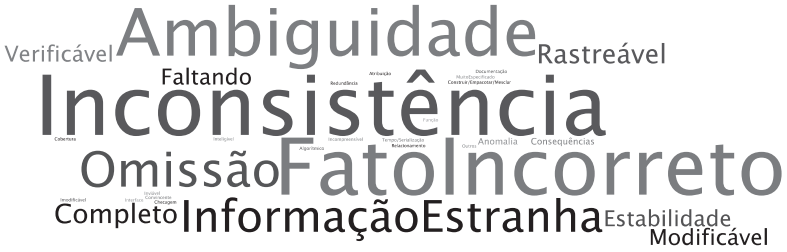


Figura 1.14: *Word Cloud* dos Tipos de Defeitos mais Relevantes.

Técnicas de Inspeção x QPs x Tipos de Pesquisa

A Figura 1.15 ilustra a distribuição de técnicas de inspeção com relação aos tipos de pesquisa e as questões de pesquisa (QP1 e QP2).

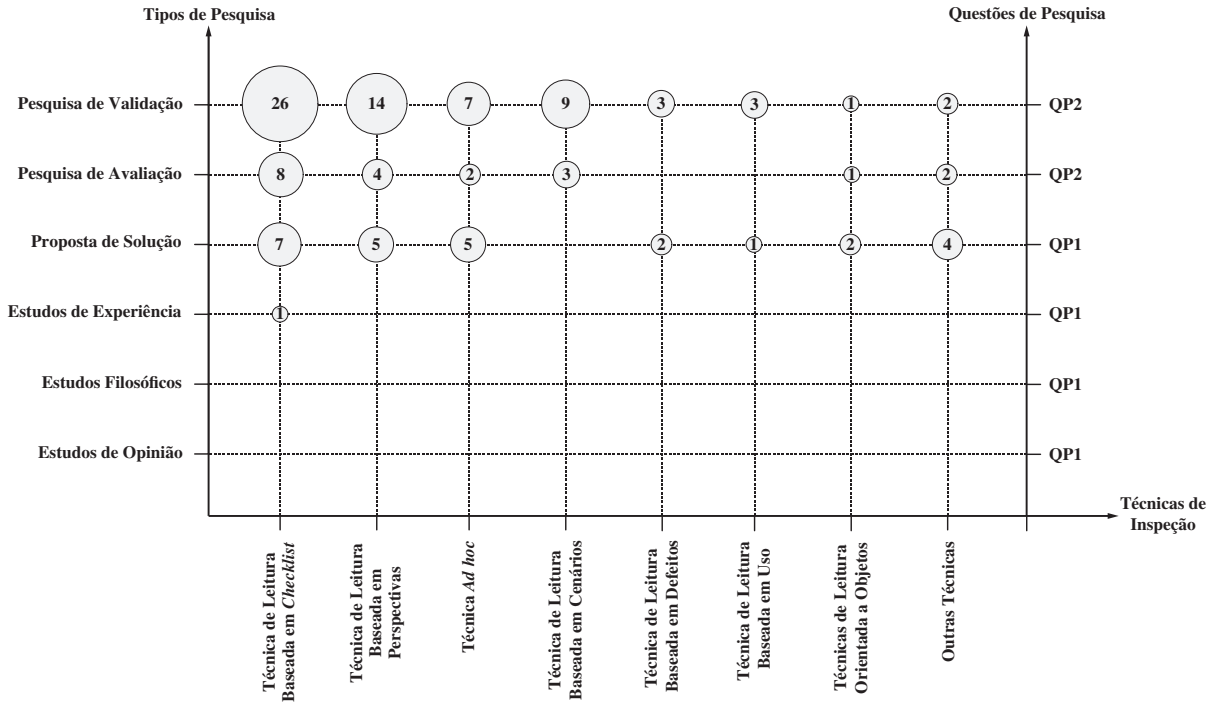


Figura 1.15: Tipos de Técnicas de Inspeção de Software baseadas nos Estudos mais Relevantes por Questões de Pesquisa (QPs) e Tipos de Pesquisa.

Analisando os estudos, foi possível observar que a maior parte das técnicas de inspeção de software estão diretamente relacionadas com as classificações denominadas Pesquisa de Validação e Pesquisa de Avaliação. Portanto, a QP2 foi respondida por estudos relacionados com experimentos controlados, estudos empíricos e casos de uso. Neste sentido, QP1 não possui nenhum estudo sobre técnicas de inspeção, pois possui o objetivo de identificar somente tipos de defeitos, além de técnicas/abordagens de inspeção de software.

De acordo com a Figura 1.15, os tipos de técnicas de inspeção mais frequentes são: Técnica de Leitura Baseada em *Checklist* (42 ocorrências), Técnica de Leitura Baseada em Perspectivas (23 ocorrências), Técnica *Ad hoc* (14 ocorrências) e Técnica de Leitura Baseada em Cenários (12 ocorrências).

Além disso, diferentes técnicas de inspeção foram mencionadas na Figura 1.15, aparecendo com menos frequência, as quais são: Técnica de Leitura Baseada em Defeitos, Técnica de Leitura Baseada em Uso e Técnica de Leitura Orientada a Objetos. Entretanto, todas essas técnicas/abordagens são consideradas essenciais para este campo de pesquisa, como pode ser visto na discussão dos estudos apresentada na próxima seção.

A.2.4 Discussão dos Estudos Mais Relevantes

Esta seção apresenta uma discussão acerca dos estudos mais relevantes deste estudo secundário. Portanto, a Tabela 1.6 lista cada um dos estudos relevantes, bem como as respectivas questões de pesquisa (QPs), autores, título do estudo, ano, tipos de pesquisa, fonte/base de dado eletrônica, o tipo e o local de publicação. Tal discussão é apresentada a seguir com base nas questões de pesquisa QP1 e QP2.

Tabela 1.6: Estudos mais Relevantes - Ordenados por Questões de Pesquisa (QP1 e QP2)

Ord.	QP	Autor(es)	Título	Ano	Tipo de Pesquisa	Fonte/Base de Dados Eletrônica	Tipo de Pesquisa	Local de Publicação
1	QP1	Anda e Sjøberg (2002)	Towards an Inspection Technique for Use Case Models	2002	Proposta de Solução	ACM Digital Library	Evento	SEKE
2	QP1	Travassos *et al.* (1999)	Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality	1999	Proposta de Solução	ACM Digital Library	Evento	OOPSLA
3	QP1	Belgamo *et al.* (2005)	TUCCA Improving the Effectiveness of Use Case Construction and Requirement Analysis	2005	Proposta de Solução	IEEE Xplore	Evento	ESEM (ISESE)
4	QP1	Cunha *et al.* (2012)	A Set of Inspection Technique on Software Product Line Models	2012	Proposta de Solução	Google Scholar	Evento	SEKE
5	QP1	Mello *et al.* (2010)	Activity Diagram Inspection on Requirements Specification	2010	Proposta de Solução	IEEE Xplore	Evento	SBES
6	QP1	Mello *et al.* (2012)	Checklist-based Inspection Technique for Feature Models Review	2012	Proposta de Solução	IEEE Xplore	Evento	SBCARS
7	QP1	Munson *et al.* (2006)	Software faults: A quantifiable definition	2006	Pesquisa de Avaliação	ELSEVIER ScienceDirect	Periódico	ADVENGSOFT
8	QP2	Brykczynski (1999)	A Survey of Software Inspection Checklists	1999	Pesquisa de Avaliação	ACM Digital Library	Periódico	ACM SIGSOFT
9	QP2	Staron *et al.* (2005)	An Empirical Assessment of Using Stereotypes to Improve Reading Techniques in Software Inspections	2005	Pesquisa de Validação	ACM Digital Library	Evento	WoSQ
10	QP2	Denger e Paech (2004)	An Integrated Quality Assurance Approach for Use Case Based Requirements	2004	Proposta de Solução	Google Scholar	Evento	LNI, GI
11	QP2	Biffl e Halling (2000)	Software Product Improvement with Inspection	2000	Pesquisa de Validação	IEEE Xplore	Evento	EUROMICRO
12	QP2	Laitenberger *et al.* (2001)	An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-Based Reading of Code Documents	2001	Pesquisa de Validação	IEEE Xplore	Periódico	TSE
13	QP2	Sabaliauskaite *et al.* (2002b)	An Experimental Comparison of Checklist-based Reading and Perspective-Based Reading for UML Design Document Inspection	2002	Pesquisa de Validação	ACM Digital Library	Evento	ESEM (ISESE)
14	QP2	Sabaliauskaite *et al.* (2004)	Assessing defect detection performance of interacting teams in object-oriented design inspection	2004	Pesquisa de Validação	ELSEVIER ScienceDirect	Periódico	INFSOF (IST)
15	QP2	Thelin *et al.* (2003)	Prioritized Use Cases as a Vehicle for Software Inspections	2003	Pesquisa de Validação	IEEE Xplore	Periódico	IEEE Software
16	QP2	Cox *et al.* (2004a)	An Experiment in Inspecting the Quality of Use Case Descriptions	2004	Pesquisa de Validação	Google Scholar	Periódico	JRPIT

- **Questão de Pesquisa 1 (QP1)**

1. Anda e Sjøberg (2002) propuseram uma taxonomia de tipos que pode ser utilizada em inspeções de software por meio da técnica de leitura baseada em *checklist*, especialmente para casos de uso. Tal taxonomia compreende os seguintes tipos de defeitos:

- `Omissão`, falta de um elemento ou funcionalidade obrigatória - por exemplo, a especificação e as variações de um caso de uso não estão presentes;
- `Fato Incorreto`, um caso de uso foi descrito de maneira incorreta;
- `inconsistência`, são problemas provenientes de casos de uso com seus objetivos e especificações mal elaborados - por exemplo, descrições entre os casos de uso, variações e terminologias;
- `Ambiguidade`, o caso de uso especificado não cumpre com seu objetivo - e.g., possui descrições com várias interpretações, ou seja, tem uma descrição ambígua;
- `Informação Estranha`, os casos de uso são redundantes - por exemplo, existem casos de uso duplicados e sem necessidade de especificação;
- `Consequências`, problemas inesperados na especificação dos casos de uso - por exemplo, a comunicação entre os analistas ou desenvolvedores é falha com relação aos objetivos do projeto e o que deve ou não ser feito.

Em adição, Anda e Sjøberg (2002) realizaram dois experimentos para validar a proposta, fornecendo resultados encorajadores com a finalidade de melhorar tal proposta, bem como evidenciar a real utilidade de inspeções por meio de técnicas de leitura baseada em *checklist* (*Checklist-Based Reading* (CBR)), no intuito de identificar defeitos em diagramas de casos de uso.

2. O estudo realizado por Travassos *et al.* (1999) merece atenção por destacar tipos de defeitos de software relacionados com a modelagem orientada a objetos em específico. Assim, Travassos *et al.* (1999) apresenta uma taxonomia de tipos de defeitos praticamente equivalente à taxonomia proposta por Anda e Sjøberg (2002). No entanto, a taxonomia proposta por Travassos *et al.* (1999) é aplicada em um conjunto de técnicas de leitura, denominada *Traceability-Based Reading* (TBR) (Técnica de Leitura baseada em Rastreabilidade), na qual foi avaliada experimentalmente em tal estudo. Neste sentido, os tipos de defeitos aplicados na técnica TBR são: `Omissão`, `Fato Incorreto`, `Inconsistência`, `Ambiguidade`, and `Informação Estranha`.
3. A técnica proposta por Belgamo *et al.* (2005), denominada *Technique for Use Case Model-based Construction and Construction Requirements Document Analysis* (TUCCA), engloba duas técnicas de leitura diferentes, nas quais são diferentes das técnicas de leitura baseada em *checklist*. Adicionalmente, um estudo de viabilidade conduzido por Belgamo *et al.* (2005) forneceu resultados interessantes quando a técnica TUCCA é comparada com as técnicas de leitura baseada em *checklist*. Assim, foi identificado tipos de defeitos que podem ser aplicados em técnicas de inspeção com base em técnicas de leitura baseada em *checklist*. Além disso, tal estudo menciona os

seguintes tipos de defeitos com relação à técnica utilizada, por meio dos requisitos da TUCCA para detectar discrepâncias:: `Omissão`, `Fato Incorreto`, `Inconsistência`, `Ambiguidade` and `Informação Estranha`.

4. Outro estudo relevante foi proposto por Cunha *et al.* (2012), no qual estabeleceram um conjunto de técnicas de inspeção, denominada *Software Product Lines Inspection Techniques* (SPLIT), com o objetivo de avaliar modelos de Linha de Produto de Software (LPS). Além disso, a técnica SPLIT leva em consideração a inspeção do Documento de Requisitos, o Mapa do Produto e o Modelo de Características para a detecção de defeitos. Neste contexto, a fim de avaliar este conjunto de técnicas de inspeção, um experimento foi realizado para comparar a abordagem de inspeção com base nos tipos de defeitos com relação à SPLIT. Os resultados do experimento favoreceram a SPLIT, pois uma grande quantidade de defeitos foi encontrada com relação a abordagem de inspeção com base nos tipos de defeitos. Portanto, os tipos de defeitos aplicados por meio da técnica SPLIT são: `Redundância`, `Anomalia` e `Inconsistência`.

5. Já o estudo de Mello *et al.* (2010), permite a identificação de defeitos em diagramas de atividade, no qual técnicas de inspeção de software podem ser aplicadas. Assim, a pesquisa realizada por Mello *et al.* (2010) apresenta uma técnica de leitura baseada em *checklist*, bem como uma especificação para os defeitos. Tal especificação está concentrada em auxiliar a revisão de diagramas de atividade para atividades de especificação de requisitos. De acordo com Mello *et al.* (2010) os seguintes tipos de defeitos foram adaptados a partir da literatura: `Omissão`, `Ambiguidade`, `Inconsistência`, `Fato Incorreto` e `Informação Estranha`.

6. Outro estudo significante de Mello *et al.* (2012), propõe uma técnica de inspeção baseada em *checklist* para apoiar a identificação de defeitos em Modelos de Características de Linhas de Produto de Software (LPS), denominada FMCheck. A principal diferença entre a FMCheck e outras técnicas de inspeção de software se baseia em qual artefato o *checklist* é aplicado. Na FMCheck, o *checklist* é aplicado no Modelo de Características, enquanto que, diagramas UML são inspecionados por diferentes técnicas, como é o caso da *SMartyCheck* (Capítulo 3). Os seguintes tipos de defeitos aplicados por meio da técnica FMCheck foram adaptados a partir do estudo de Travassos *et al.* (1999): `Omissão`, `Fato Incorreto`, `Inconsistência`, `Ambiguidade` e `Informação Estranha`.

7. Por outro lado, em comparação com os estudos anteriormente mencionados, a pesquisa conduzida por Munson *et al.* (2006) realiza tentativas para quantificar as

falhas de software, definindo o defeito como um tipo especial de falha, de acordo com a avaliação de falhas realizada neste estudo por meio de um software específico de análise gramátical.

- **Questão de Pesquisa 2 (QP2)**

1. A pesquisa conduzida por Brykczynski (1999) visa identificar quais itens de *checklists* desenvolvidos por Michael Fagan Fagan (1976, 1986) entre as décadas de 70's e 80's, são considerados relevantes para o processo de inspeção de software. Assim, Brykczynski (1999) analisou 117 *checklists* a partir de 24 fontes diferentes, a fim de validar tal pesquisa. Portanto, isso permitiu Brykczynski (1999) observar que conduzir inspeções por meio de técnicas de leitura baseada em *checklist* é realmente efetivo. Em adição, tal técnica é amplamente adotada pela indústria para inspecionar artefatos por meio de *checklists* em diferentes contextos de software.
2. Neste contexto de qualidade, é importante ressaltar o estudo de Staron *et al.* (2005), no qual fornece evidências, baseado em vários experimentos, de que os estereótipos especificados em diagramas UML contribuem para garantir a qualidade no processo de inspeção de software, bem como reduz os defeitos aplicando técnicas de leitura basedas em *checklist* e *Perspective-Based Reading* (PBR) (Técnicas de Leitura baseda em Perspectivas). Assim, os resultados obtidos evidenciaram que a técnica CBR é mais eficiente e a PBR é mais efetiva. Portanto, os estereótipos são essenciais para garantir a qualidade em inspeções de software realizadas.
3. O estudo de Denger e Paech (2004) é interessante, pois apresenta uma abordagem integrada a fim de garantir a qualidade de casos de uso. Tal abordagem Tal abordagem combina *guidelines* e técnicas de inspeção para casos de uso. Essa abordagem é avaliada por meio de simulações. Assim, a combinação proposta é realizada com base na classificação de defeitos e classes de defeitos, nos quais foram identificados levando em consideração critérios de qualidade. Além disso, Denger e Paech (2004) somente avaliaram as técnicas CBR e PBR, sendo assim, ambas foram praticamente iguais na detecção de defeitos. Portanto, por meio de experimentos, os resultados obtidos forneceram evidências iniciais para aumentar a eficiência e a eficácia para garantir a qualidade de casos de uso.
4. O estudo realizado por Biffl e Halling (2000) investigou experimentalmente o efeito das técnicas de leitura CBR, SBR (*Scenario-Based Reading* (SBR) - Técnica de Leitura baseda em Cenários) e PBR em conjunto com a detecção de defeitos envolvendo o processo de inspeção de qualidade. Portanto, os resultados obtidos

corroboraram para medir a eficácia de tais técnicas. É importante mencionar, que a técnica CBR obteve uma melhor eficácia nos resultados quando comparada com a técnica PBR na maioria dos casos.

5. Os quasi-experimentos realizados por Laitenberger *et al.* (2001) também estabeleceram comparações no intuito de identificar a eficácia e o custo na detecção de defeitos na técnica PBR com relação a técnica CBR. Com o objetivo de avaliar este cenário, duas replicações foram realizadas com participantes da Bosch Telecom GmbH. Na sequência, os resultados obtidos em geral forneceram evidências que a PBR é mais efetiva que a CBR. Portanto, a técnica PBR reduziu o custo por defeitos encontrado e contribuiu para a detecção de um grande quantidade de defeitos durante as reuniões de inspeção. Em adição, a técnica conseguiu reduzir os custos na identificação de defeitos com base nos esforços dos participantes.

6. O estudo de Sabaliauskaite *et al.* (2002b) avaliou o desempenho de dois grupos de inspeção por meio de experimentos utilizando duas técnicas de leitura: a CBR e a PBR. Neste cenário, os resultados obtidos a partir da comparação destas técnicas não revelaram diferenças significativas entre as mesmas.

7. Outro estudo conduzido por Sabaliauskaite *et al.* (2004) também comparou as técnicas CBR e PBR, mas em um contexto diferente com base em um estudo anterior Sabaliauskaite *et al.* (2002b). Portanto, este novo estudo apresenta uma avaliação documentos UML. Os seguintes resultados foram obtidos: eficácia similar na detecção de defeitos em ambas as técnicas de inspeção; os revisores que utilizaram a técnica PBR gastaram menos tempo inspecionando artefatos em comparação com os revisores que utilizaram a técnica CBR; e o custo por defeitos encontrados utilizando a técnica CBR é menor do que utilizando a técnica PBR.

8. Outro estudo experimental relevante conduzido por Thelin *et al.* (2003) apresenta a técnica *Usage-Based Reading* (UBR) (Técnica de Leitura baseada no Uso), na qual foi experimentalmente avaliada por meio da comparação com a técnica CBR. Tal técnica (UBR) foi analisada por meio de três testes, no qual foi avaliada a eficiência (defeitos encontrados por hora), eficácia (porcentagem de defeitos encontrados) e preparação (tempo de inspeção em minutos). Assim, o experimento indicou que a técnica UBR é significativamente mais eficiente e efetiva quando comparada com a técnica CBR.

9. Já o estudo de Cox *et al.* (2004a) apresentou resultados experimentais relevantes na aplicação da inspeção baseada em *checklist* (CBR) no processo técnico de descrição ou especificação de casos de uso. Tal experimento comparou a técnica CBR e a técnica

Ad hoc com base em um grupo de participantes, que identificaram geralmente mais defeitos aplicando a técnica CBR.

Por meio de uma análise geral, é evidente a diversidade de tipos de estudos/pesquisas, os quais compreendem desde:

- propostas de taxonomias de tipos de defeitos, experimentalmente avaliadas para diagramas UML específicos;
- diferentes propostas e comparações de técnicas ou abordagens de inspeção de software e para a detecção de defeitos;
- vários experimentos sobre diferentes técnicas de inspeção de software; e
- estudos que avaliam a qualidade de diagramas UML por meio de estereótipos que podem ser utilizados em técnicas de inspeção de software.

É interessante mencionar, que os estudos recuperados e selecionados neste estudo secundário, apresentam uma perspectiva motivadora. Entre os estudos discutidos foi possível observar distintas pesquisas e questões/problemas pertinentes em vários cenários, principalmente experimentais. Portanto, este estudo secundário realizado é considerado essencial para identificar estudos que possam mitigar qualquer tipo de auxílio para a detecção de defeitos em técnicas de inspeção de software.

Além disso, diversos experimentos foram identificados com base nos estudos discutidos, os quais contribuiram para a compreensão de tipos de defeitos e técnicas de inspeção existentes na literatura. Isto, propiciou evidências essenciais para a evolução de técnicas de inspeção por meio da identificação, adaptação e aplicação de diferentes tipos de defeitos em contextos distintos.

A.3 Ameaças à Validade

As principais ameaças à validade deste estudo secundário são discutidas a seguir:

- `Questões de pesquisa`: duas questões de pesquisa foram definidas para conduzir este estudo secundário. Tais questões de pesquisa foram analisadas e refinadas antes de serem finalmente elaboradas. Acredita-se que mais palavras-chave genéricas retornaram mais estudos primários relevantes no escopo deste estudo. Entretanto, este estudo está focado apenas em tipos de defeitos e técnicas de inspeção de software.

- `Fontes/Bases de dados eletrônicas`: quatro fontes eletrônicas foram selecionadas, as quais são consideradas essenciais para a comunidade acadêmica com base em diversos mapeamentos sistemáticos e revisões sistemáticas realizadas na literatura. Contudo, se mais fontes de busca fossem consideradas para este estudo secundário, poderiam ser identificados um maior conjunto de estudos relevantes potenciais. Além disso, estudos secundários deveriam ser considerados para expandir a lista de fontes/bases de dados eletrônicas.

- `Viés de publicação`: não é possível garantir que todos os estudos primários relacionados com este estudo secundário foram recuperados. Este problema pode ocorrer devido ao fato de que as informações (dados) contidas nos motores de busca não são precisas como se deseja para o processamento e execução de *queries* (consultas) de acordo com as palavras-chaves definidas.

- `Falta de familiaridade com outras áreas`: a melhoria das palavras-chave e das *strings* de busca definidas para este estudo secundário poderia auxiliar e otimizar a busca dos estudos primários mais relevantes em outras áreas de pesquisa. Por exemplo, os estudos primários da área de segurança de informação poderiam ser investigados, a fim de identificar quais tipos de defeitos ocorrem nas infraestruturas de redes de computadores, levando em consideração os defeitos presentes em redes mais simples até redes mais complexas. Portanto, os possíveis tipos de defeitos presentes em tal área não foram considerados neste estudo secundário.

A.4 Considerações Finais

A principal motivação para o desenvolvimento deste estudo foi mapear os tipos de defeitos existentes na literatura da área de inspeção de software, com o objetivo de fornecer classificações interessantes e taxonomias para orientar pesquisadores interessados na realização de estudos voltados para este tópico. Assim, por meio deste estudo secundário, novas pesquisas podem ser realizadas, no intuito de evoluir tipos de defeitos existentes e técnicas de inspeção de software e/ou propondo novas abordagens para inspecionar artefatos de software em vários contextos diferentes.

Analisando os resultados obtidos neste estudo secundário, foi possível fornecer evidências de que a maioria dos estudos mapeados estão relacionados em diferentes contextos, tais como, experimentos, propostas de novas técnicas de inspeção e novas taxonomias de tipos de defeitos. Estudos foram conduzidos em grandes organizações/empresas (IBM, NASA, JPL, AT&T, Motorola, Nortel, Allianz, etc.), as quais relataram que se as inspeções de

software forem bem planejadas, com os respectivos tipos de defeitos adaptados a partir da literatura, podem contribuir para a detecção de defeitos com uma média de 80%, bem como aumentar a qualidade de software.

Na época em que este estudo secundário foi realizado, a literatura existente sobre o tópico de pesquisa em questão possuia carência de estudos sobre detecção de defeitos, considerando vários contextos diferentes, tais como, a inspeção baseada em modelos de Linhas de Produto de Software (LPSs), Arquiteturas de Referência, Arquitetura Orientada a Modelos (*Model-Driven Architecture*) e Sistemas de Sistemas (*System-of-Systems*)

Uma vez que este estudo secundário foi realizado, espera-se que os resultados obtidos possam orientar trabalhos futuros com ênfase nos tipos de defeitos já identificados e em técnicas de inspeção de software, bem como em novos estudos. Além disso, novos estudos secundários podem ser realizados levando-se em consideração as diferentes fontes/bases de dados eletrônicas e as palavras-chave utilizadas.

Apêndice B - Perfis UML: Diagramas de Casos de Uso e Classes

B.1 Perfis UML: Diagramas de Casos de Uso e Classes

A *Unified Modeling Language* (UML) (Linguagem de Modelagem Unificada) fornece aos engenheiros de software, arquitetos de sistemas e desenvolvedores de software uma notação padronizada para a modelagem de softwares, processos e negócios. Além disso, a UML e seus diagramas, são altamente disseminados e utilizados no desenvolvimento de software por meio de ferramentas de apoio para a análise, projeto e implementação de software (OMG, 2011).

Os principais conceitos e a forma de extensão semântica da UML por meio de perfis são apresentados na próxima Subseção B.1.1. Por conseguinte, a especificação dos principais elementos e caminhos gráficos dos diagramas de casos de uso e de classes, relevantes para este trabalho, é apresentada de forma sucinta com base no documento da OMG (2011).

B.1.1 Fundamentos sobre Perfis UML

A UML 2.0 permite a extensão de metaclasses com base em seus metamodelos (OMG, 2011). Uma metaclasse (`UseCase`, `Class`, `Component`, entre outras) pode ser instânciada por meio da criação de um diagrama UML, sendo assim, um modelo UML é uma instância de um metamodelo. Deste modo, um metamodelo UML permite criar novos elementos por meio de mecanismos de extensão (Weilkiens e Oestereich, 2006). Com isso, para cumprir novas especificações, os mecanismos de extensão permitem estender a UML de maneira semântica por meio do seu metamodelo.

Portanto, é possível estender a UML e projetos UML por meio do pacote *Profile* (<<`profile`>>), ou seja, um mecanismo de extensão denominado de perfil UML, conforme ilustrado na Figura 2.1, no qual permite o relacionamento entre pacotes de um projeto e um perfil (OMG, 2011; Weilkiens e Oestereich, 2006). Assim, o perfil UML

é composto por estereótipos (*stereotype*), meta-atributos (*tagged values*) dos estereótipos e restrições (*constraint*).

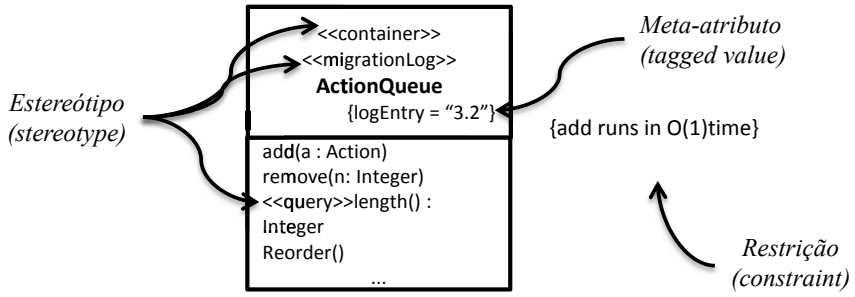


Figura 2.1: Elementos para a Extensão da UML por meio de Perfis, adaptado de (Marcolino, 2014; OMG, 2011).

Neste contexto, os estereótipos podem estar presentes em um perfil UML, os quais têm as seguintes características: estendem o poder da UML de maneira controlada, padronizada e semântica; descrevem como uma metaclasse pode ser estendida; podem possuir atributos (*tagged values*) e operandos; e permitem que um diagrama UML possa ser anotado com um novo estereótipo, pois o mesmo não pode ser extendido por meio do relacionamento de generalização (Weilkiens e Oestereich, 2006).

Sob o panorama de perfis UML, é possível citar alguns exemplos: *Testing Profile* (UTP) (OMG, 2013), SysML (OMG, 2010) e SPEM (OMG, 2008). Além disso, a abordagem *SMarty* (Seção 2.2.1), considerada para este trabalho, utiliza-se dos conceitos de perfil UML em seu escopo, a fim adequar uma semântica coesa para a identificação e representação de variabilidades em modelos UML. Assim, a *SMarty* possui seu próprio perfil, o *SMartyProfile*, de acordo com as especificações da UML (OMG, 2011).

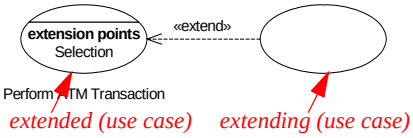
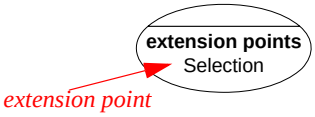
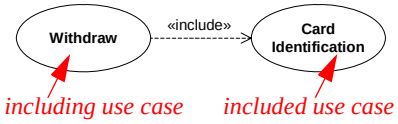
B.1.2 Diagramas de Casos de Uso

Segundo a OMG (2011), de maneira geral, os casos de uso são utilizados na captura e na especificação de requisitos e comportamentos obrigatórios exigidos por um sistema. Um diagrama de casos de uso é composto por atores, por casos de uso, bem como o sistema, nos quais os casos de uso se aplicam. Já os atores podem ser representados por usuários do sistema ou por outros sistemas externos. Assim, todos estes elementos são descritos em diagramas de casos de uso.

A Tabela 2.1 apresenta o conjunto de elementos gráficos de diagramas de casos de uso com suas respectivas notações, além das suas descrições. Além disso, a Tabela 2.1 ilustra possíveis caminhos (`paths`) gráficos (<<`extend`>> e <<`include`>>), ou seja,

relacionamentos utilizados entre os casos de uso. Deste modo, os elementos gráficos e os caminhos gráficos expressam quais elementos podem se interagir, além de como podem ser relacionados, no intuito de cumprirem os comportamentos exigidos por um sistema.

Tabela 2.1: Elementos Gráficos de Diagramas de Casos de Uso e suas Descrições, adaptado de (OMG, 2011).

Elemento	Notação	Descrição
Actor		A notação de ator é representada pelo ícone "*stick man*" com o nome do ator normalmente definido abaixo do ícone.
Extend		Este relacionamento específica que o comportamento do caso de uso pode ser estendido (*extended*) por meio do comportamento de outro caso de uso (*extending*).
ExtensionPoint		Identifica um ponto no comportamento do caso de uso, no qual o comportamento pode ser estendido (*extended*) por meio do comportamento de outro caso de uso (*extending*) especificado por meio do relacionamento *extend*.
Include		Define que um caso de uso contém o comportamento definido em outro caso de uso. O relacionamento ocorre entre dois casos de uso, nos quais o comportamento definido em um caso de uso (*including*) é incluído no comportamento do caso de uso base (*included*).
UseCase		Consiste em uma especificação de um conjunto de ações executadas por um sistema, retornando um resultado de valor para um ou mais atores do sistema.

A Figura 2.2 apresenta um exemplo de um diagrama de casos de uso (OMG, 2011), no qual representa, provavelmente, um sistema de vendas por telefone (*telemarketing*). Neste diagrama, foram modelados os seguintes elementos gráficos:

- Quatro atores (*actor*): *Customer* (Cliente), *Salesperson* (Vendedor), *Shipping Clerk* (Encarregado do Envio) e o *Supervisor* (Supervisor);

- Quatro casos de uso (*use case*): *Check Status* (Checar o Status), *Place Order* (Fazer Pedido), *Fill Order* (Preencher Pedido) e *Establish Credit* (Estabelecimento de Crédito);

- O sistema (*Telephone Catalog*) é representado por *subject* (Sistema), no qual contém os quatro casos de uso que recebem interações externas dos quatro atores.

Após a modelagem, os caminhos gráficos, ou seja, os relacionamentos foram definidos entre os atores e os casos de uso (Figura 2.2). Por exemplo: a função do ator *Salesperson* (Vendedor) é checar o status do pedido feito pelo *Customer* (Cliente) por meio do caso de uso *Check Status* (Checar o Status) e, fazer um pedido, quando solicitado pelo *Customer* (Cliente) por meio do caso de uso *Place Order* (Fazer Pedido). Entretanto, o *Customer* (Cliente) pode também: checar o status de um pedido feito por ele mesmo por meio do caso de uso *Check Status* (Checar o Status), fazer um pedido por meio do caso de uso *Place Order* (Fazer Pedido), além da possibilidade de verificar seu crédito por meio do caso de uso *Establish Credit* (Estabelecimento de Crédito).

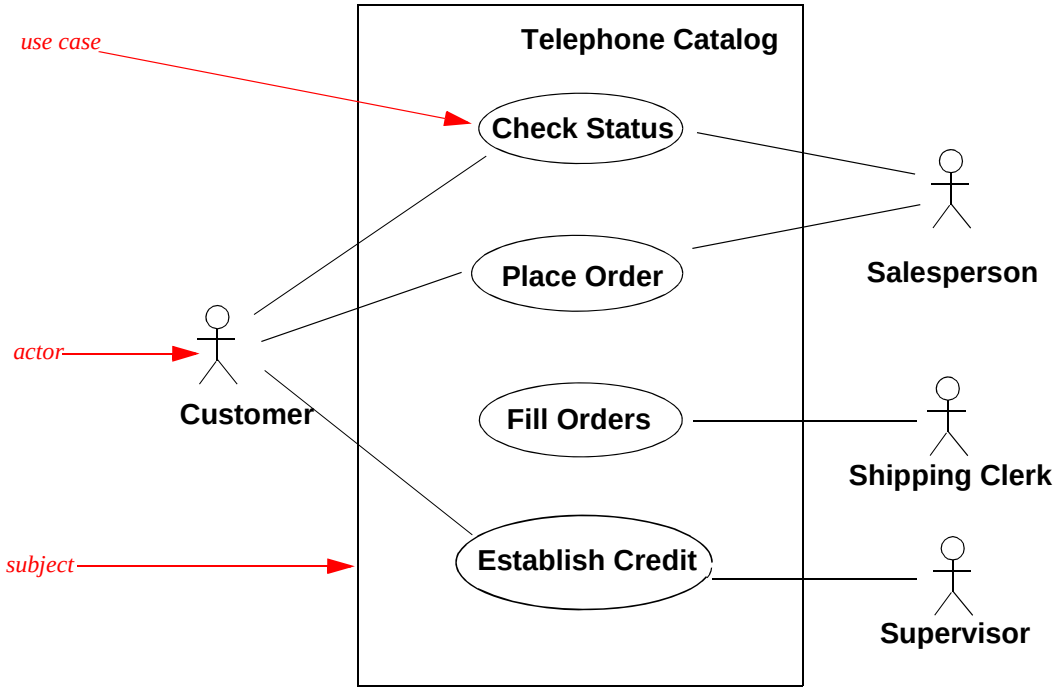


Figura 2.2: Exemplo Diagrama de Casos de Uso, adaptado de (OMG, 2011).

B.1.3 Diagramas de Classes

Segundo a OMG (2011), de forma geral, as classes descrevem um conjunto de objetos que compartilham as mesmas especificações. Um diagrama de classes é basicamente composto por classes e pacotes. As classes podem ser utilizadas para representar a estrutura de um sistema e seus relacionamentos por meio do compartilhamento de características (atributos

e operações), restrições e da semântica. Já os pacotes são utilizados para agrupar os elementos, ou seja, as classes. Assim, estes elementos são descritos em diagramas de classes.

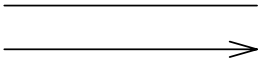
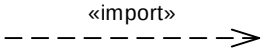
A Tabela 2.2 apresenta os elementos gráficos básicos (*Class* e *Package*) de diagramas de classes com suas respectivas notações, além das suas descrições.

Tabela 2.2: Elementos Gráficos de Diagramas de Classes e suas Descrições, adaptado de (OMG, 2011).

Elemento	Notação	Descrição
Class		Uma classe descreve um conjunto de objetos que compartilham as mesmas especificações de características (atributos e operações), restrições e semântica.
Package		Um pacote é utilizado para agrupar elementos, proporcionando uma descrição (*namespace*) para os elementos agrupados.

Além disso, a Tabela 2.3 ilustra possíveis caminhos (`paths`) gráficos, ou seja, relacionamentos utilizados entre as classes. Deste modo, os elementos gráficos e os caminhos gráficos expressam quais elementos podem se interagir, além de como podem ser relacionados, no intuito de cumprirem as especificações de um sistema, representando a sua estrutura geral.

Tabela 2.3: Caminhos Gráficos de Diagramas de Classes e suas Descrições, adaptado de (OMG, 2011).

Tipo de Caminho	Notação	Descrição
Aggregation (Agregação)		É um tipo de enumeração que especifica os literais para a definição do tipo de agregação de uma propriedade.
Association (Associação)		Uma associação descreve um conjunto de tuplas cujos valores referem-se a instâncias tipificadas. Uma instância de uma associação é chamada de *link*. Um *link* é uma tupla com um valor para cada extremidade da associação.
Composition (Composição)		É um tipo de agregação, mas com o valor literal igual à *composite*. Indica que a propriedade é agregada por composição, na qual o objeto composto é responsável pela existência e armazenamento dos objetos compostos (partes).
Dependency (Dependência)		É um relacionamento que significa que um único elemento ou um conjunto de elementos do diagrama requer outros elementos do diagrama para a sua especificação ou implementação.
Generalization (Generalização)		É uma relação taxonômica entre um classificador (classe ou elemento) mais geral e um classificador mais específico. Cada instância do classificador específico é também uma instância indireta do classificador mais geral. Assim, o classificador específico herda as características do classificador mais geral.
Realization (Realização)		É uma relacionamento de abstração especializado entre dois conjuntos de elementos do diagrama, um representando uma especificação (fornecedor) e o outro representa uma implementação do último (cliente).
PackageImport (public)		É um relacionamento direcionado, no qual identifica um pacote cujos membros devem ser importados por meio de um local (*namespace*).

A Figura 2.3 apresenta um exemplo de um diagrama de classes (OMG, 2011), representando um sistema botânico básico, no qual as árvores são organizadas por espécies. Neste diagrama, foram modelados os seguintes elementos gráficos (classes): *Tree* (Árvore), *Sugar Maple* (Espécie), *Apricot* (Espécie), *American Elm* (Espécie), *Saguaro* (Espécie), *Tree Species* (Espécies de Árvores), *Geographic Location* (Localização Geográfica) e *Leaf Pattern* (Padrão das Folhas). Assim, com a finalidade de exemplificar a Figura 2.3, pode-se observar neste diagrama, por exemplo: as classes *Sugar Maple* (Espécie), *Apricot* (Espécie), *American Elm* (Espécie) e *Saguaro* (Espécie) generalizam ou herdam (*Generalization* - Tabela 2.3) características da classe *Tree* (Árvore), sendo que, cada espécie é uma árvore.

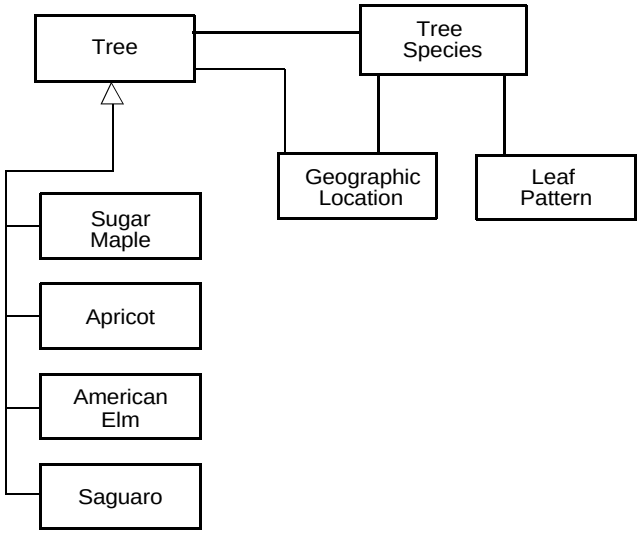


Figura 2.3: Exemplo Diagrama de Classes, adaptado de (OMG, 2011).

Apêndice C - A Linha de Produto de Software *Arcade Game Maker*

C.1 Introdução

A *Arcade Game Maker* (AGM) é uma LPS pedagógica para jogos, criada pelo *Software Engineering Institute* (SEI) (SEI, 2009) para apoiar o aprendizado e a experimentação de conceitos de LPS. Possui um conjunto completo de documentos e diagramas UML, bem como um conjunto de classes testadas e código-fonte para três jogos diferentes, os quais são: *Pong*, *Bowling* e *Brickles*. Apesar de não ser uma LPS comercial, a AGM tem sido utilizada para ilustrar os conceitos de várias abordagens diferentes de LPS, estudos de caso e avaliação de arquiteturas de LPS.

Os artefatos essenciais da AGM são: o modelo de características, o modelo de casos de uso, o modelo de classes, diagrama de casos de uso, diagrama de classes e a arquitetura lógica de componentes. Neste apêndice, são apresentados o diagrama de casos de uso e o diagrama de classes da LPS AGM, pois são relevantes com relação aos objetivos deste trabalho.

C.2 Similaridades e Variabilidades

A seguir são apresentadas as principais similaridades contidas na LPS AGM:

- todo jogo possui um conjunto de **Sprites**, nas quais são os elementos do jogo que os jogadores vêem e com os quais eles interagem;

- todo jogo possui um conjunto de **Rules**, nas quais são as regras que regem as ações dos jogos. Por exemplo, um jogo pode ter uma regra em que um objeto em movimento ao colidir com um objeto estático deve obedecer às leis da física; e

- todos os jogos envolvem movimentação.

Além das similaridades a LPS AGM possui as seguintes variabilidades:

- **Tipos de Regras**: é a maior diferença entre os jogos. Algumas regras estão relacionadas às leis da física (gravidade, colisões, etc.) e podem ser aplicáveis a múltiplos jogos. Outras regras estão especificamente relacionadas a um jogo e podem ser usadas em todas as implementações do jogo, mas não se aplicam a outros jogos; e

- **Tipos de Movimentação**: em alguns jogos a movimentação é inerente à operação do jogo. Isto acontece periodicamente e é orientada pelo tempo. Em outros jogos, o jogador escolhe e inicia a movimentação, sendo ações dirigidas pelo ator.

C.3 Atores e Casos de Uso

Os itens a seguir apresentam os atores e casos de uso da LPS AGM e as suas principais ações. A Figura 3.1 apresenta o diagrama de casos de uso da LPS AGM.

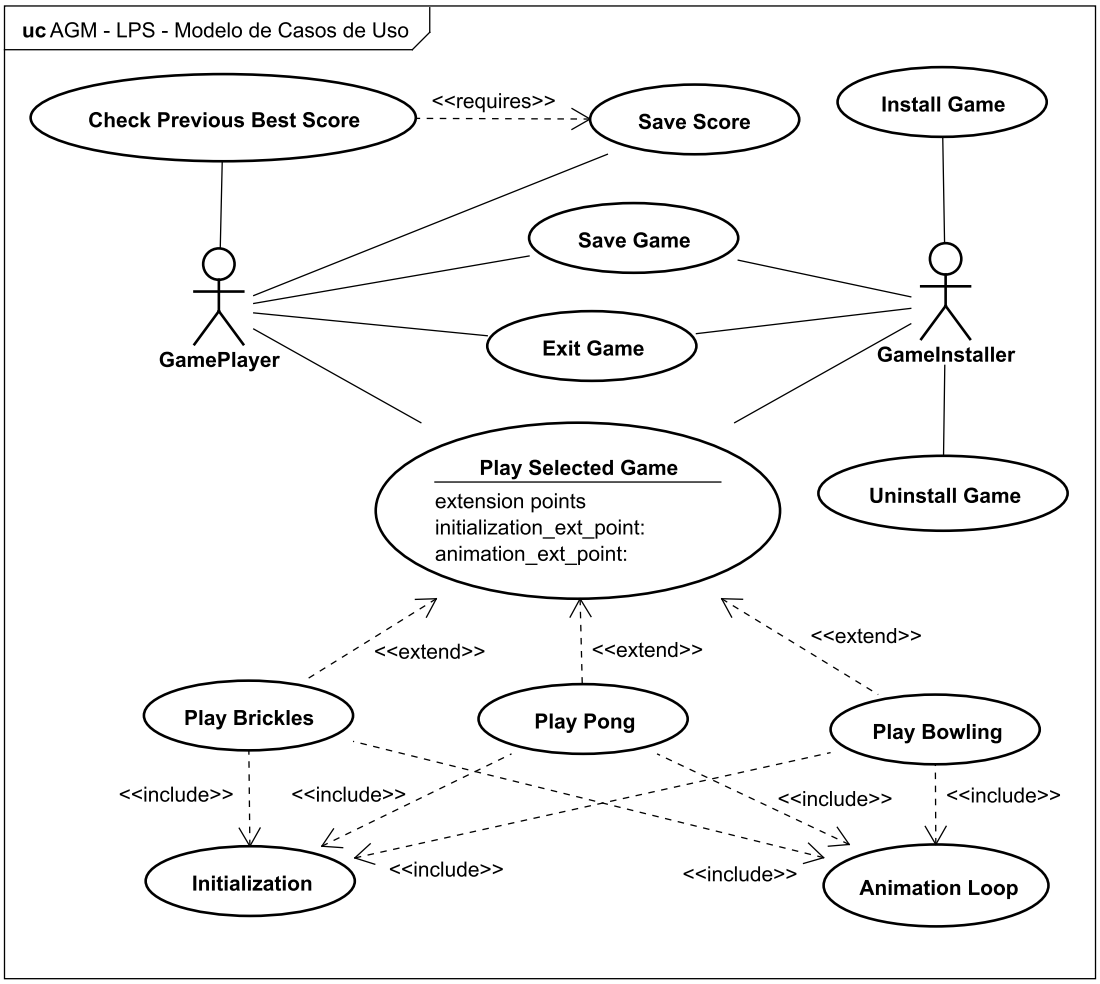


Figura 3.1: Diagrama de Casos de Uso da LPS AGM. Adaptado de (OliveiraJr, 2010).

1. ***GamePlayer***: ator responsável por executar as principais ações de um jogo produzido pela AGM.

2. ***GameInstaller***: ator responsável por ações de configuração dos jogos (instalação e desinstalação) produzidos pela AGM.

3. ***Check Previous Best Score***: verifica e apresenta a melhor pontuação registrada anteriormente.

   (a) Ator seleciona a opção *CHECK PREVIOUS BEST SCORE* do menu do sistema. Sistema pede para fornecer o nome do arquivo, lê o arquivo e retorna o placar (*score*) à caixa de diálogo.

   (b) Ator seleciona *OK* na caixa de diálogo para continuar. Sistema retorna ao estado anterior.

4. ***Save Score***: salva a pontuação corrente do jogador.

   (a) Ator seleciona *SAVE SCORE* no menu do sistema. Sistema pede um nome de arquivo (cria um novo se não existe), escreve o score no arquivo e retorna ao estado pré-salvo do jogo.

5. ***Save Game***: salva o jogo em andamento.

   (a) Ator seleciona a opção *SAVE GAME* no menu do sistema. Sistema permite ao ator especificar um nome de arquivo e, em seguida, escreve os dados do jogo no arquivo especificado, retornando ao estado pré-salvo do jogo.

6. ***Install Game***: instala o jogo escolhido.

   (a) Ator escolhe o instalador do jogo para ser executado. Sistema apresenta uma caixa de diálogo para escolher o diretório em que serão armazenados os arquivos do jogo.

   (b) Ator escolhe o diretório. Sistema armazena os arquivos no diretório escolhido.

7. ***Exit Game***: encerra o jogo em andamento.

   (a) Ator seleciona *EXIT* no Menu do sistema. Sistema apresenta a caixa de diálogo para salvar ou sair do jogo.

   (b) Ator salva o jogo. Sistema salva e sai ou cancela saída do jogo.

   (c) Sistema retorna à ação suspensa.

8. ***Uninstall Game***: remove o jogo selecionado.

   (a) Ator escolhe a opção *UNINSTALL* do menu do sistema. Sistema apresenta uma caixa de diálogo ao ator.

   (b) Ator seleciona o diretório do jogo a ser removido. Sistema exclui os arquivos do diretório e apresenta uma caixa de diálogo de remoção concluída.

   (c) Ator seleciona a opção OK da caixa de diálogo. Sistema fecha a caixa de diálogo.

9. ***Play Selected Game***: um ator seleciona o jogo e inicia a sua execução.

   (a) Ator seleciona a opção *PLAY* a partir do menu. Sistema inicializa o jogo e apresenta o *GameBoard*.

   (b) Ator clica com o botão esquerdo e inicia o jogo. Sistema inicia a ação do jogo.

(c) Ator clica com o botão esquerdo ou usa o teclado para enviar comandos. Sistema responde aos comandos.

(d) Ator responde à caixa de diálogo *Won/Lost/Even* clicando com o botão esquerdo. Sistema retorna o *GameBoard* para ser inicializado.

(e) A qualquer instante o ator pode selecionar *EXIT*, via menu.

10. ***Play Bowling***: inicia o jogo *Bowling*.

(a) Ator seleciona *PLAY* do Menu do sistema. Sistema inicializa o jogo e apresenta o *GameBoard*.

(b) Ator clica com o botão esquerdo para jogar. Sistema inicia a ação do jogo.

(c) Ator repete as ações seguintes dez (10) vezes mais um lance de bônus (opcional)

(d) Ator posiciona o mouse e clica com o botão esquerdo para lançar a *Bowling-Ball* pela *Alley* (pista). Sistema move a *BowlingBall* pela *Alley* usando um algoritmo randômico. Se há colisões com os *BowlingPins*, o sistema move os *BowlingPins*, segundo as leis físicas de colisão. Sistema conta o número de pinos derrubados. Sistema computa o *score*.

11. ***Play Brickles***: inicia o jogo *Brickles*.

(a) Ator seleciona *PLAY* do menu do sistema. Sistema inicializa o jogo e apresenta o *GameBoard*.

(b) Ator clica com o botão esquerdo para iniciar o jogo. Sistema inicia a ação do jogo.

(c) Ator usa o botão esquerdo ou o teclado para enviar comandos ao jogo. Sistema move o *Paddle* horizontalmente, seguindo o rastro do mouse. A cada movimento do *Puck*, o sistema verifica se houve colisão com outro objeto. Se o *Puck* colide com o *Ceiling* (teto) ou com uma *Wall*, o *Puck* volta para a área de jogo. Se o *Puck* colide com o *Floor*, deixa de existir. Se o número máximo de *Pucks* não foi alcançado, um novo é criado, caso contrário a caixa de diálogo *Lost* é apresentada. Se o *Puck* colide com um *Brick*, a ação a ser tomada depende do *Brick*. Se for o último *Brick*, a caixa de diálogo *Won* é apresentada.

(d) Ator responde à caixa de diálogo *Won/Lost* clicando com o botão esquerdo. Sistema retorna o *GameBoard* ao estado inicializado e pronto para jogar.

12. ***Play Pong***: inicia o jogo *Pong*.

(a) Ator seleciona *PLAY* do menu do sistema. Sistema inicializa o jogo e apresenta o *GameBoard*.

(b) Ator clica com o botão esquerdo para iniciar o jogo. Sistema inicia a ação do jogo.

13. ***Animation Loop***: executa as ações de animação dos jogos.

(a) Sistema gera periodicamente sinais e os envia para o jogo.

(b) Sistema move todos os objetos passo a passo de acordo com os seus algoritmos de movimentação.

(c) Sistema verifica se há colisões e executa os algoritmos de colisão dos objetos.

14. ***Initialization***: inicializa o jogo selecionado e apresenta o *GameBoard*.

(a) Sistema cria as instâncias-padrão para as classes requeridas.

(b) Sistema entra no estado *READY*.

C.4 Classes

A Tabela 3.1 apresenta as classes da LPS AGM e uma breve descrição sobre cada uma delas. A Figura 3.2 apresenta o diagrama de classes da LPS AGM.

Tabela 3.1: Pacotes e Descrição das Classes da LPS AGM

Pacote	**Classe**	**Descrição**
coreAssets	*Board*	Borda de um jogo
	GameMenu	Menu com as opções de um determinado jogo
	GameSprite	Elementos dos jogos com os quais o jogador interage
	Menu	Menu com as principais opções dos jogos
	MovableSprite	Elementos que se movem em um jogo
	Paddle	Elemento utilizado em um jogo para colocar um Puck em movimento
	Point	Determinado ponto em um retângulo
	Puck	Representa o principal elemento de um jogo como, por exemplo, a bola que derruba os BowlingPins no jogo Bowling, a bolinha que destrói os BrickPile no jogo Brickles, etc.
	Rectangle	Um retângulo que demarca uma área em um jogo
	Size	Tamanho de um retângulo
	SpritePair	Par de elementos de um jogo que reagem à uma ação
	StationarySprite	Elementos que não se movem em um jogo
	Velocity	Velocidade de um MovableSprite
	Wall	Representa as paredes de um jogo
bowl	*Bowling*	Classe com a inicialização do jogo
	BowlingBall	Bola de boliche
	BowlingBoard	Borda do jogo Bowling
	BowlingGameMenu	Menu com as opções específicas do jogo
	BowlingPin	Pino do jogo de boliche
	Edge	Limites esquerdo e direito da canaleta de boliche
	EndOfAlley	Fim da pista de boliche
	Gutter	Canaleta da pista de boliche
	Lane	Pista de boliche
	RackOfPins	Local onde os pinos são posicionados
pong	*BottomPaddle*	Elemento que movimenta a Puck do jogo, localizado na parte inferior da PongBoard
	Ceiling	Teto do jogo
	DividingLine	Linha divisória dos Paddles
	Floor	Chão do jogo
	LeftWall	Parede à esquerda do jogo
	Pong	Classe com a inicialização do jogo
	PongBoard	Borda do jogo Pong
	PongGameMenu	Menu com as opções específicas do jogo
	RightWall	Parede à direita
	ScoreBoard	Placar do jogo
	TopPaddle	Elemento que movimenta a Puck do jogo, localizado na parte superior da PongBoard
brickles	*Brick*	Tijolo a ser quebrado pelo elemento Puck
	Brickles	Classe com a inicialização do jogo
	BricklesBoard	Borda do jogo Pong
	BricklesGameMenu	Menu com as opções específicas do jogo
	BrickPile	Pilha de tijolos
	Ceiling	Teto do jogo
	Floor	Chão do jogo
	LeftWall	Parede à esquerda
	PuckSupply	Quantidade de Pucks que o jogador tem direito em um jogo
	RightWall	Parede à direita

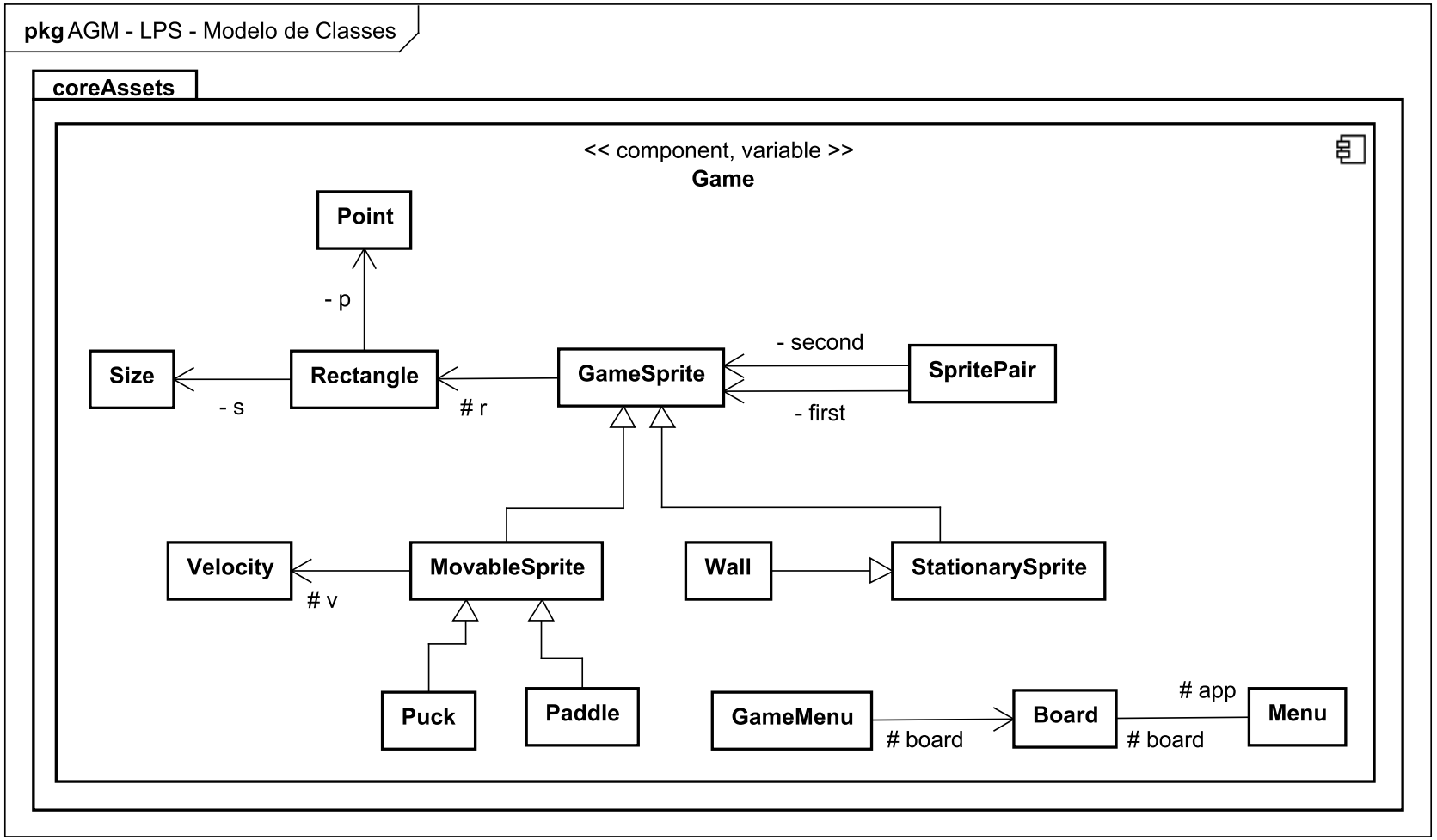


Figura 3.2: Diagrama de Classes da LPS AGM. Adaptado de (OliveiraJr, 2010).

C.5 Telas dos Jogos

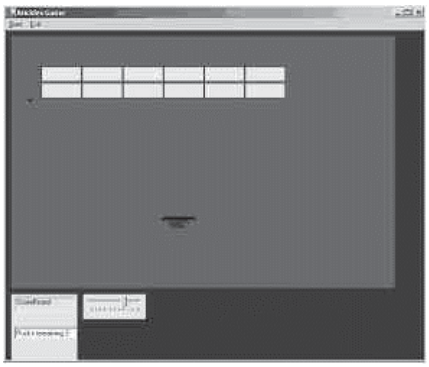


Figura 3.3: Tela do Jogo Brickles

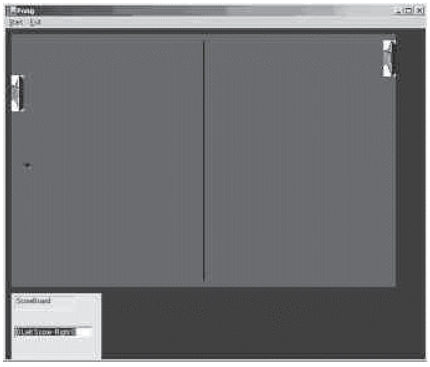


Figura 3.4: Tela do Jogo Pong

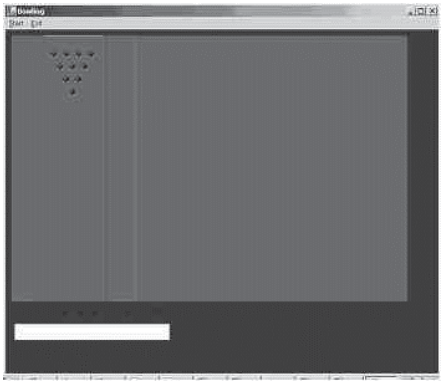


Figura 3.5: Tela do Jogo Bowling

Apêndice D - A Linha de Produto de Software *Mobile Media*

A *Mobile Media* é uma LPS composta por aplicações (produtos) que manipulam músicas, vídeos e fotos para dispositivos móveis, como *smartphones* e *tablets*. Provê suporte para gerenciar (criar, excluir, visualizar, executar e enviar) diferentes tipos de mídia (Young, 2005).

A *Mobile Media* surgiu da extensão de um LPS já existente denominada *Mobile Photo* (Young, 2005) por meio da inserção de novas propriedades multimídia, como manipulação de vídeos e músicas, que somente podem ser realizados em alguns tipos de aparelhos.

De certa forma, pode-se dizer que a inserção das características opcionais e alternativas a determinados aparelhos caracterizou o surgimento da *Mobile Media*. A Figura 4.1 apresenta o Diagrama de Casos de Uso da *Mobile Media*.

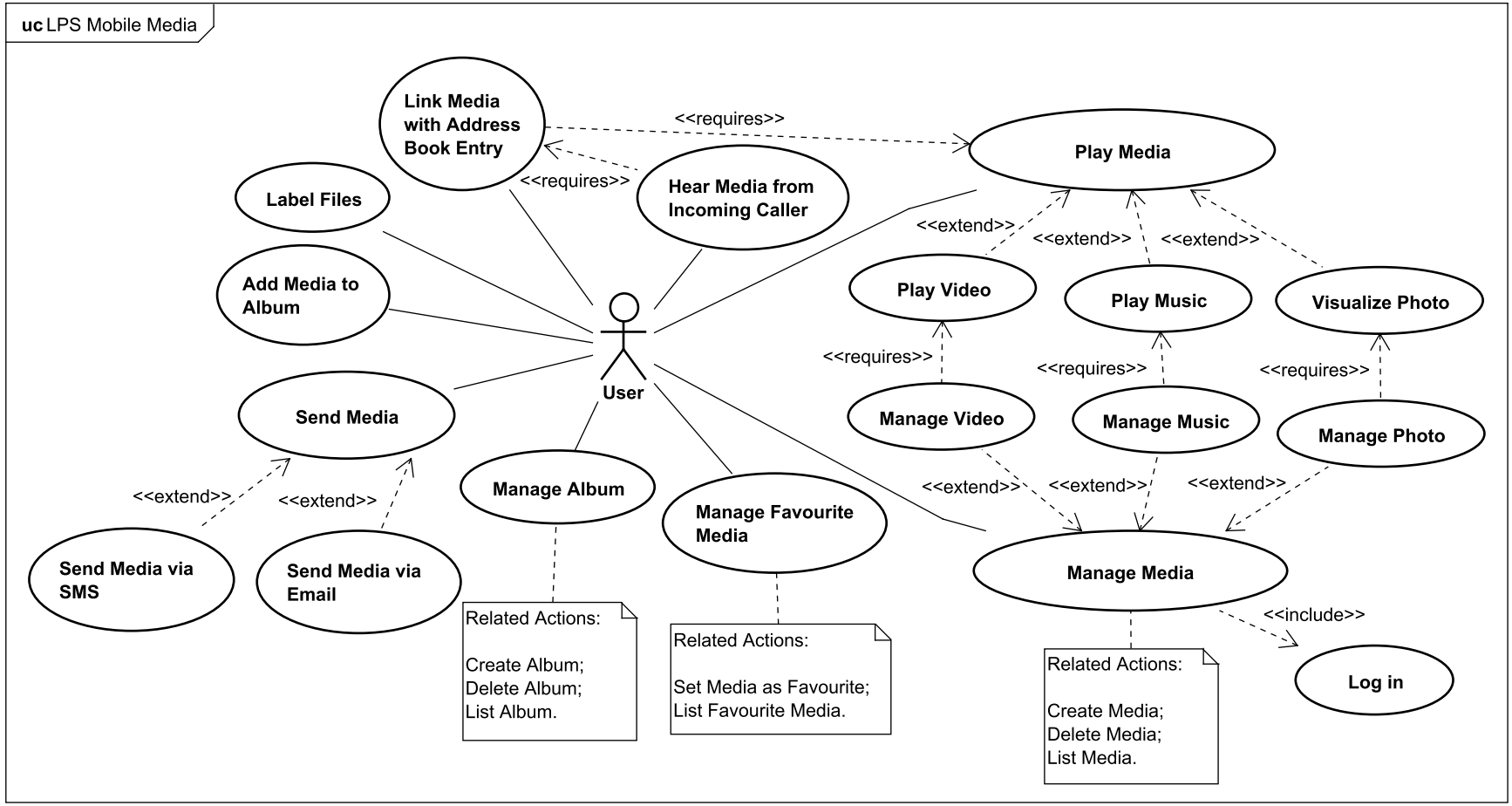


Figura 4.1: Diagrama de Casos de Uso da LPS *Mobile Media*. Adaptado de (Santos *et al.*, 2008).

Os itens a seguir apresentam as características básicas e as características variáveis da *Mobile Photo*, agora denominada *Mobile Media*:

1. **Criar Álbum de Fotos**: Permite ao usuário definir novos álbuns de fotos para armazenar categorias de fotos no dispositivo. A persistência da informação do álbum é realizada utilizando RMS (J2ME *Record Management System*).

2. **Armazenar Foto**: Gerenciar a conversão e persistência dos arquivos de foto para o sistema de arquivos do dispositivo utilizando RMS.

3. **Adicionar/Deletar Foto**: Permite ao usuário excluir fotos permanentemente do dispositivo ou adicionar novas fotos a álbuns definidos.

4. **Rotular Foto**: Permite ao usuário determinar um texto para uma foto. Os rótulos aparecerão na lista de exibição e podem ser utilizados para uma futura funcionalidade relacionada à busca.

5. **Visualizar Foto**: Mostra uma foto selecionada na tela do dispositivo.

6. **Enviar Foto via SMS**: Permite a um usuário enviar uma foto para outro via *Short Messaging Service*.

7. **Enviar Foto via Email**: Permite a um usuário enviar uma foto para outro via Email.

8. **Relacionar Foto com Registro na Agenda**: Permite ao usuário associar um registro na sua lista de contatos com a foto do álbum.

9. **Mostrar Foto nas Chamadas Recebidas**: Intercepta chamadas recebidas e mostra a foto associada ao contato.

10. **Tocar Melodia nas Chamadas Recebidas**: Intercepta chamadas recebidas e toca uma melodia personalizada para aquele contato.

De acordo com a Figura 4.1 da LPS *Mobile Media*, a Figura 4.2 apresenta um Diagrama de Casos de Uso da LPS *Mobile Media* com base na abordagem *SMarty*.

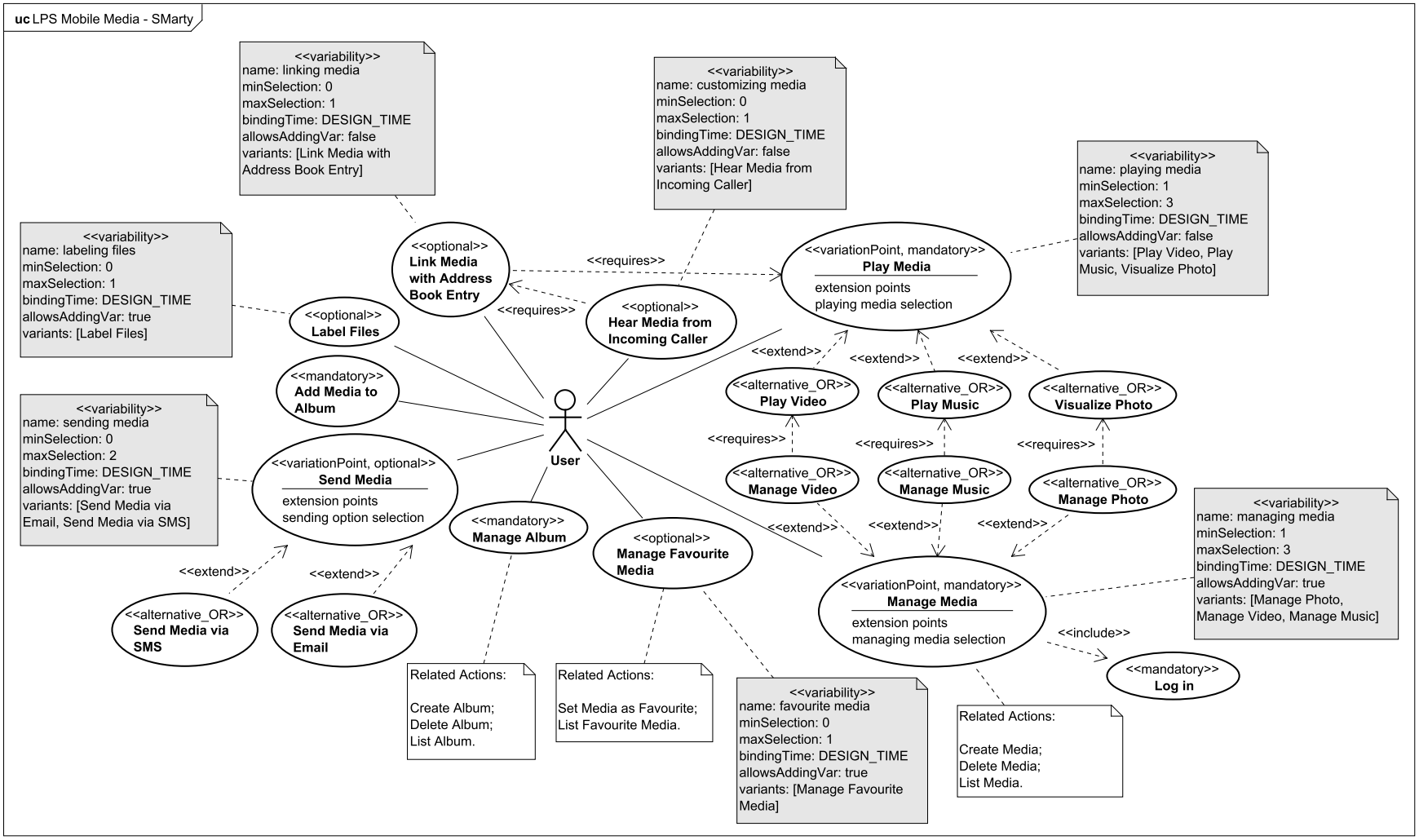


Figura 4.2: Diagrama de Casos de Uso da LPS *Mobile Media* com base na *SMarty*. Adaptado de (Contieri Junior, 2010).

Apêndice E - Questionários Eletrônicos *SMartyCheck*

E.1 Introdução

Primeiramente, um questionário de caracterização foi aplicado aos participantes referente ao nível de conhecimento e experiência sobre UML, LPS e inspeção de software (Seção E.2). Além disso, formulários eletrônicos apresentados nas próximas seções são exemplos dos quais foram criados no software LimeSurvey[1] com relação a técnica *SMartyCheck*.

Para o estudo empírico qualitativo de viabilidade foram criados seis formulários eletrônicos, os quais foram distribuídos e enviados por email aleatoriamente para os participantes. Em adição, um formulário eletrônico com algumas questões sobre a técnica *SMartyCheck* e o estudo empírico qualitativo foi enviado aos participantes. Tais exemplos dos formulários eletrônicos são apresentados em ordem na Seção E.3.

Já para o estudo empírico quantitativo de efetividade foram criados doze formulários eletrônicos, os quais foram distribuídos e enviados por email aleatoriamente para os participantes. Portanto, seis questionários eletrônicos foram criados com relação a técnica *SMartyCheck* e seis questionários eletrônicos foram criados com relação a técnica *Ad hoc*. Tais exemplos dos formulários eletrônicos são apresentados em ordem na Seção E.4.

E.2 Questionário de Caracterização dos Participantes

[1]LimeSurvey - *http://www.limesurvey.org*

Questionário de Caracterização de Participante em Estudo de Caso Empírico Quantitativo

"Avaliação de Efetividade da SMartyCheck: uma Técnica de Inspeção baseada em Checklist para Modelos SMarty de Variabilidade"

Nome - ID do Participante

Nas perguntas a seguir, quando duas ou mais alternativas forem válidas, marque a alternativa que mais se aplica ao seu caso.

1. Qual o seu nível de formação?

[ ] Graduando
[ ] Pós-graduando (Especialização)
[ ] Mestrando
[ ] Doutorando

[ ] Graduado
[ ] Pós-graduado (Especialização)
[ ] Mestre
[ ] Doutor

2. Em qual setor atua?

[ ] Acadêmico (ensino)

[ ] Industrial (empresarial/corporativo)

3. Qual o nome da empresa/universidade que atua?

_____

4. Quanto tempo possui de experiência na área que atua?

_____ meses ou _____ anos

5. Qual a sua experiência com a notação UML com relação aos diagramas de casos de uso e de classes?

[ ] Já lí, de forma superficial, algo a respeito de UML.

[ ] Eu **nunca** modelei um software usando a UML.

[ ] **Minha experiência com a notação UML é básica**.
Eu modelo software somente no nível dos elementos mais comuns da UML como casos de uso; classes e herança.

[ ] **Minha experiência com a notação UML é moderada**.
Eu modelo software no nível dos elementos da opção anterior, além de: relacionamentos de dependência *include* e *extend*, e *extension points* em diagramas de casos de uso; e polimorfismo, associação (uni e bi-direcionais), dependência, agregação e composição em classes.

[ ] **Minha experiência com a notação UML é avançada**.
Eu modelo software que exige a utilização de todos os elementos de diagramas de casos de uso e classes, além de outros diagramas da UML como, por exemplo, diagramas de colaboração, sequência, e componentes.

6. Qual a sua experiência com relação à abordagem de Linha de Produto de Software (LPS) e Gerenciamento de Variabilidade?

[ ] Eu **nunca** ouvi falar a respeito de LPS.

[ ] **Já lí**, de forma superficial, algo a respeito de LPS.

[ ] **Minha experiência com LPS é básica**.
Eu conheço os seguintes conceitos da abordagem: ciclo de desenvolvimento de LPS e suas atividades (engenharia de domínio e engenharia de aplicação). Porém, não tenho experiência com gerenciamento de variabilidades.

[ ] **Minha experiência com LPS é moderada**.
Eu conheço os conceitos da opção anterior, e com relação ao gerenciamento de variabilidades, eu sei o conceito de pontos de variação, variantes e os seus relacionamentos, além dos conceitos de resolução de variabilidades e tempos de resolução (*DESIGN_TIME*, *LINK_TIME*, *RUNTIME*, entre outros).

[ ] **Minha experiência com LPS é avançada**.
Eu conheço os conceitos da opção anterior, além de alguns processos existentes de desenvolvimento de LPS (FODA, PLP, PLUS, PuLSE, entre outros). Com relação ao gerenciamento de variabilidades, eu sei os conceitos da opção anterior, além de: modelos de resolução; abordagens existentes para o gerenciamento de variabilidades, e representação de variabilidades (usando a UML, modelos de características, entre outras).

7. Você já teve alguma experiência ou conhecimento à respeito da abordagem *Stereotype-based Management of Variability* (*SMarty*)?

[ ] Sim [ ] Não

8. Qual a sua experiência com relação à técnica de Inspeção de Software?

[ ] Eu **nunca** ouvi falar a respeito sobre Inspeção de Software.

[ ] **Já lí**, de forma superficial, algo a respeito sobre Inspeção de Software.

[ ] **Minha experiência com Inspeção de Software é básica.**
Eu conheço os seguintes conceitos com relação ao cenário da inspeção de software: controle de qualidade e revisão de software.

[ ] **Minha experiência com Inspeção de Software é moderada.**
Eu conheço os conceitos da opção anterior e já lí sobre algum processo de detecção e remoção de defeitos. Exemplos: Fagan Defect-Free Process™, Inspection / Agile Specification Quality Control (Spec QC) by Tom Gilb, IBM Orthogonal Defect Classification, dentre outros.

[ ] **Minha experiência com Inspeção de Software é avançada.**
Eu conheço os conceitos da opção anterior, além de algumas técnicas de leitura existentes utilizadas em conjunto na inspeção de software: técnicas de leitura de artefato de software, técnicas de leitura *Ad hoc*, técnicas de leitura baseadas em *checklist*, técnicas de leitura baseadas em perspectivas, técnicas de leitura baseadas em defeitos, técnicas de leitura baseada em usabilidade e técnicas de leitura orientada a objetos, dentre outras técnicas.

Assinatura do Participante	Local e Data
_____	Cidade, Estado - País Data

E.3 Questionários Eletrônicos - Estudo Empírico Qualitativo de Viabilidade

A Figura 5.1, a Figura 5.2 e a Figura 5.3 ilustram, a seguir, um exemplo de questionário eletrônico de casos de uso da LPS AGM com o *checklist* da *SMartyCheck*. Tal questionário foi dividido em três partes, a fim de melhorar a visualização com base no formato deste documento.

Já a Figura 5.4, a Figura 5.5 e a Figura 5.6 ilustram, a seguir, um exemplo de questionário eletrônico de classes da LPS AGM com o *checklist* da *SMartyCheck*. Tal questionário foi dividido em três partes, a fim de melhorar a visualização com base no formato deste documento.

Adicionalmente, a Figura 5.7 ilustra dois exemplos de condições (Sim ou Não) de inspeção definidas para os itens do *checklist* da *SMartyCheck* com base na sua taxonomia de tipos de defeitos.

Por fim, a Figura 5.8 ilustra um exemplo do questionário eletrônico sobre a técnica *SMartyCheck* e o estudo empírico qualitativo de viabilidade.

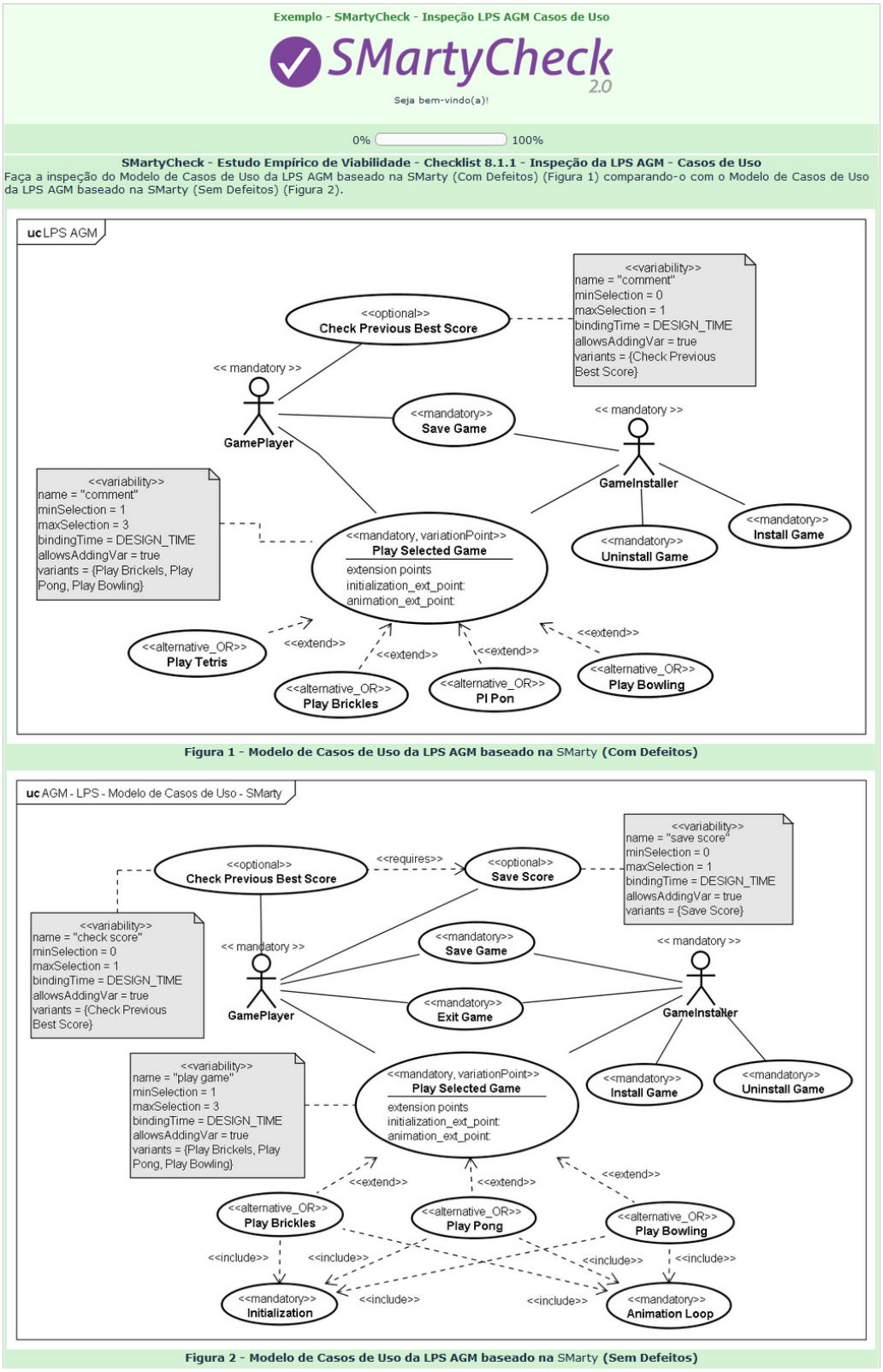


Figura 5.1: 1º Parte - Exemplo de Questionário Eletrônico de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

Figura 5.2: 2º Parte - Exemplo de Questionário Eletrônico de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

Figura 5.3: 3º Parte - Exemplo de Questionário Eletrônico de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

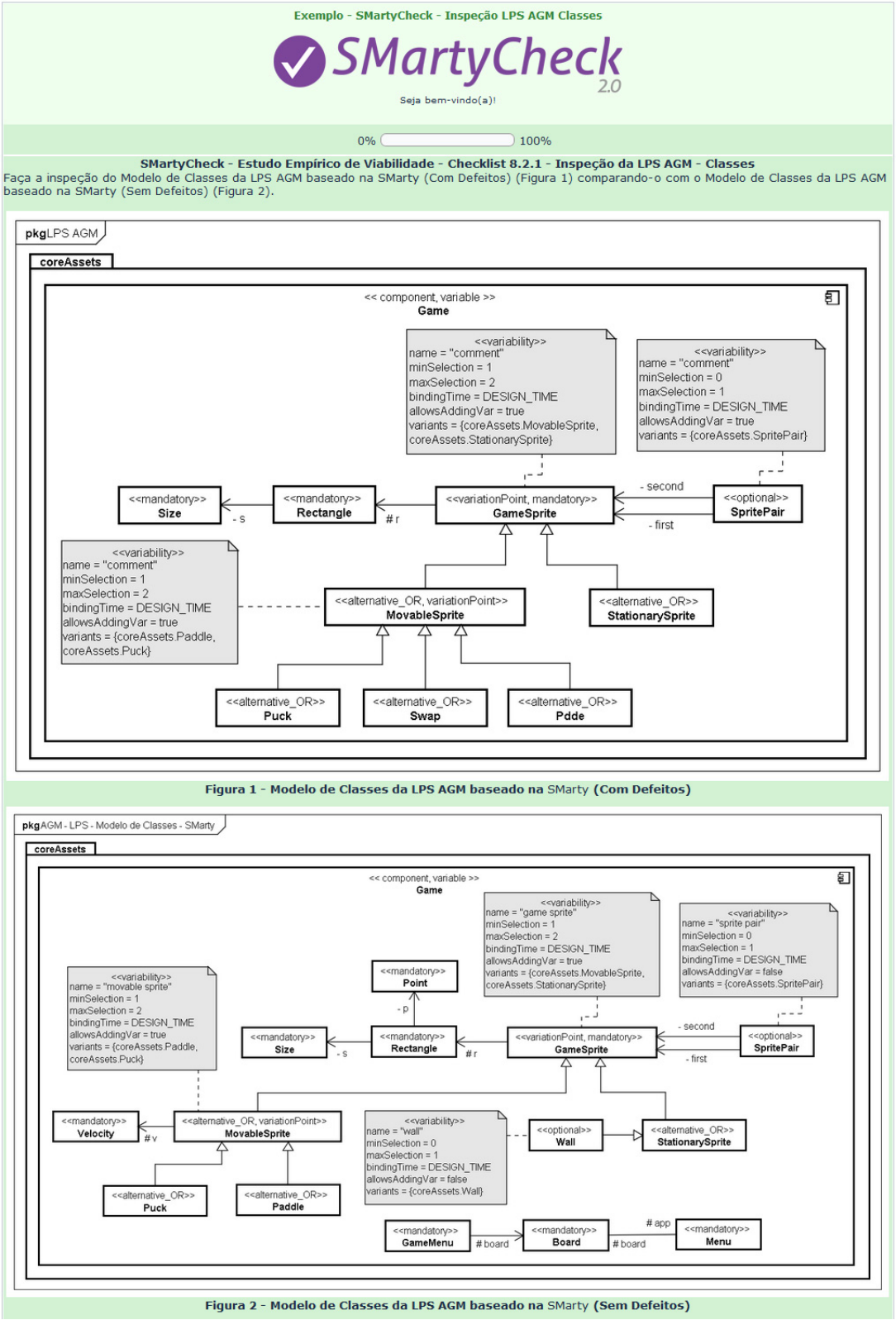


Figura 5.4: 1º Parte - Exemplo de Questionário Eletrônico de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

Para realizar a inspeção utilize o checklist da técnica SMartyCheck apresentado a seguir. A inspeção deve ser realizada da seguinte forma:

- Responda cada item do checklist, selecionando apenas uma opção, ou seja, "Sim" ou "Não";
- Quando a opção "Sim" for selecionada, favor justificar sua resposta no campo que surgirá, denominado "Defeito Identificado";
- Não é necessário justificar sua resposta quando a opção "Não" for selecionada.

01 - Ambiguidade (Am)

* **Am.1 Existe alguma classe no modelo de classes da LPS em que sua descrição não reflete no seu objetivo real?**

○ Sim ○ Não

* **Am.2 Existe alguma classe no modelo de classes da LPS, em que seu nome ou descrição é igual a classe que está relacionada de modo a possuir uma interpretação duplicada?**

○ Sim ○ Não

02 - Anomalia (An)

* **An.1 Existe alguma classe no modelo de classes da LPS especificada por meio do estereótipo <<alternative_OR>> associada com uma classe que não esteja relacionada com o estereótipo <<variationPoint>>?**

○ Sim ○ Não

03 - Incompleto (Incom)

* **Incom.1 Todas as classes com <<mandatory>>, ou seja, obrigatórias não estão especificadas no modelo de classes da LPS?**

○ Sim ○ Não

04 - Inconsistência (Incons)

* **Incons.1 Existe alguma classe no modelo de classes da LPS com o estereótipo <<variationPoint>> cujo número de variantes especificadas é maior que o definido em *maxSelection* da variabilidade associada?**

○ Sim ○ Não

* **Incons.2 Existe alguma classe no modelo de classes da LPS com o estereótipo <<variationPoint>> cujo número de variantes especificadas é maior que o definido em *minSelection* da variabilidade associada?**

○ Sim ○ Não

* **Incons.3 Existe alguma variabilidade (<<variability>>) especificada no modelo de classes da LPS cujo meta-atributo *bindingTime* seja DESIGN_TIME (tempo de projeto)?**

○ Sim ○ Não

05 - Incorreto (Incor)

* **Incor.1 Existe alguma classe no modelo de classes da LPS com o estereótipo <<variationPoint>> que está associada ou relacionada com uma classe na LPS que não seja <<alternative_OR>>?**

○ Sim ○ Não

* **Incor.2 Existe alguma classe no modelo de classes da LPS com o estereótipo <<mandatory>> que está associada ou relacionada com uma classe na LPS marcada com o estereótipo <<mandatory>>?**

○ Sim ○ Não

06 - Desvio Intencional (DI)

* **DI.1 Existe alguma classe no modelo de classes da LPS que exige a seleção de outra (<<requires>>) e essa outra não está especificada na LPS?**

○ Sim ○ Não

* **DI.2 Existe alguma classe no modelo de classes da LPS que exige a não seleção de outra (<<mutex>>) e essa outra está especificada na LPS?**

○ Sim ○ Não

Figura 5.5: 2º Parte - Exemplo de Questionário Eletrônico de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

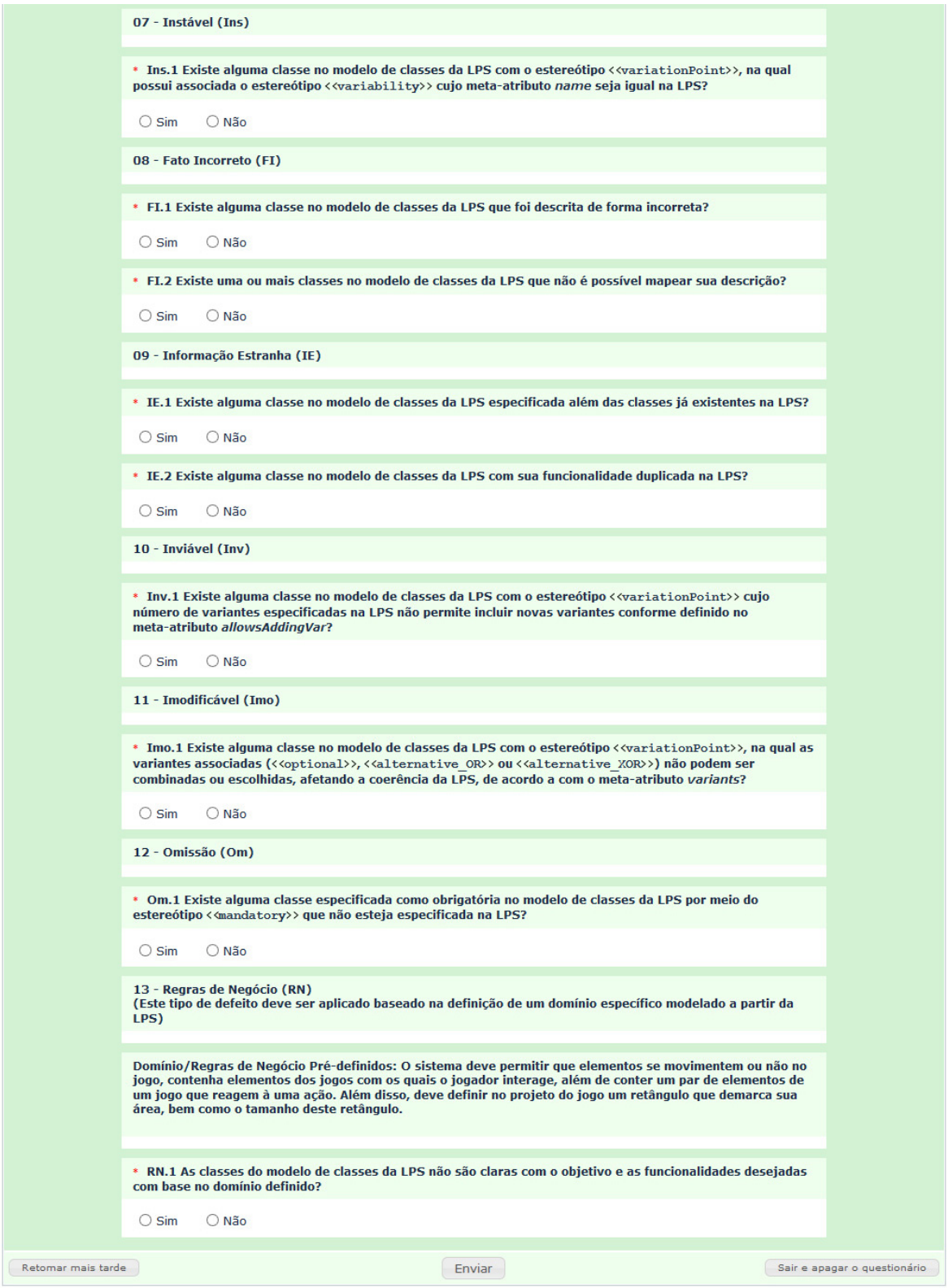


Figura 5.6: 3º Parte - Exemplo de Questionário Eletrônico de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

A Figura 5.7 ilustra dois exemplos de condições (Sim ou Não) de Inspeção definidas para os itens do *checklist* da *SMartyCheck* com base na sua taxonomia de tipos de defeitos. Portanto, a inspeção deve ser realizada da seguinte forma:

- Responda cada item do *checklist*, selecionando apenas uma opção, ou seja, "Sim"ou "Não";
- Quando a opção "Sim"for selecionada, favor justificar sua resposta no campo que surgirá, denominado "Defeito Identificado";
- Não é necessário justificar sua resposta quando a opção "Não"for selecionada.

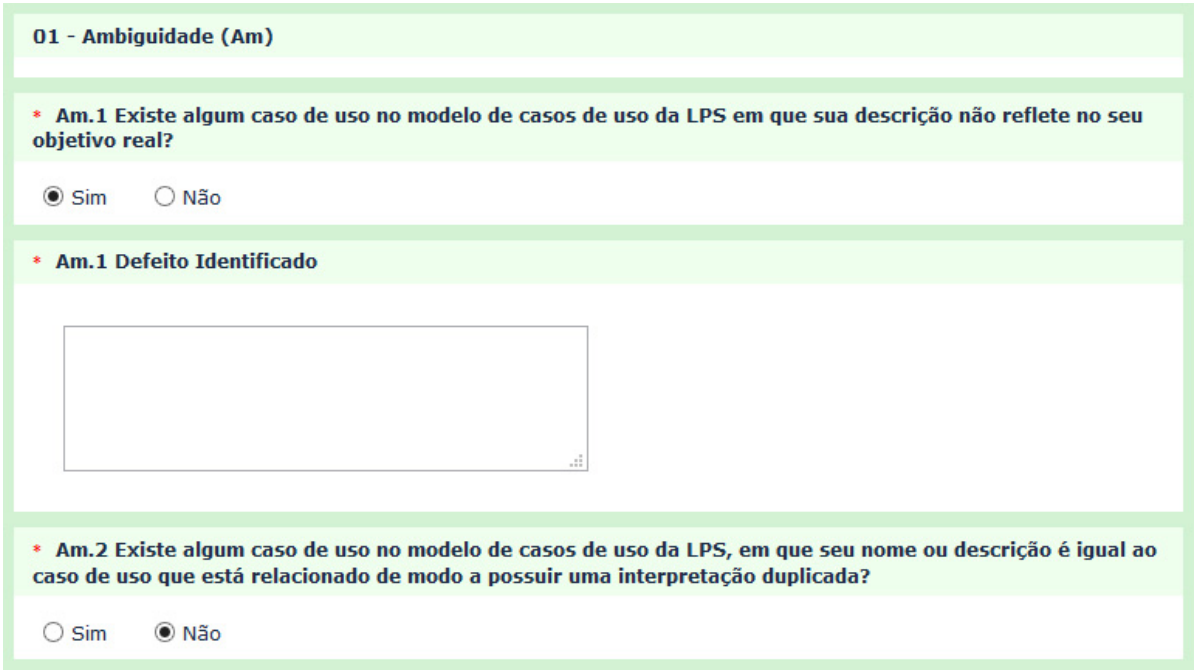


Figura 5.7: Exemplos de Condições (Sim ou Não) de Inspeção definidas para os itens do *Checklist* da *SMartyCheck*.

Figura 5.8: Exemplo do Questionário Eletrônico criado no LimeSurvey sobre a técnica *SMartyCheck* e o Estudo Empírico Qualitativo.

E.4 Questionários Eletrônicos - Estudo Empírico Quantitativo de Efetividade

Após a análise e interpretação dos resultados obtidos em ambos os estudos empíricos, o *checklist* da *SMartyCheck* foi refinado e aprimorado. Com isso, esta seção apresenta os questionários eletrônicos, os quais foram utilizados pelos participantes como um instrumento para a coleta de dados com relação a inspeção de diagramas UML *SMarty* de casos de uso e de classes por meio da técnica *SMartyCheck* e/ou utilizando a técnica *Ad hoc* em comparação.

A Figura 5.9, a Figura 5.10, a Figura 5.11 e a Figura 5.12 ilustram, a seguir, um exemplo de questionário eletrônico de casos de uso da LPS AGM com o *checklist* da *SMartyCheck*. Tal questionário foi dividido em quatro partes, a fim de melhorar a visualização com base no formato deste documento.

Já a Figura 5.13, a Figura 5.14, a Figura 5.15 e a Figura 5.16 ilustram, a seguir, um exemplo de questionário eletrônico de classes da LPS AGM com o *checklist* da *SMartyCheck*. Tal questionário foi dividido em quatro partes, a fim de melhorar a visualização com base no formato deste documento.

A Figura 5.17 e a Figura 5.18 ilustram, a seguir, um exemplo de questionário eletrônico de casos de uso da LPS AGM com a técnica *Ad hoc*. Tal questionário foi dividido em duas partes, a fim de melhorar a visualização com base no formato deste documento.

Já a Figura 5.19 e a Figura 5.20 ilustram, a seguir, um exemplo de questionário eletrônico de classes da LPS AGM com a técnica *Ad hoc*. Tal questionário foi dividido em duas partes, a fim de melhorar a visualização com base no formato deste documento.

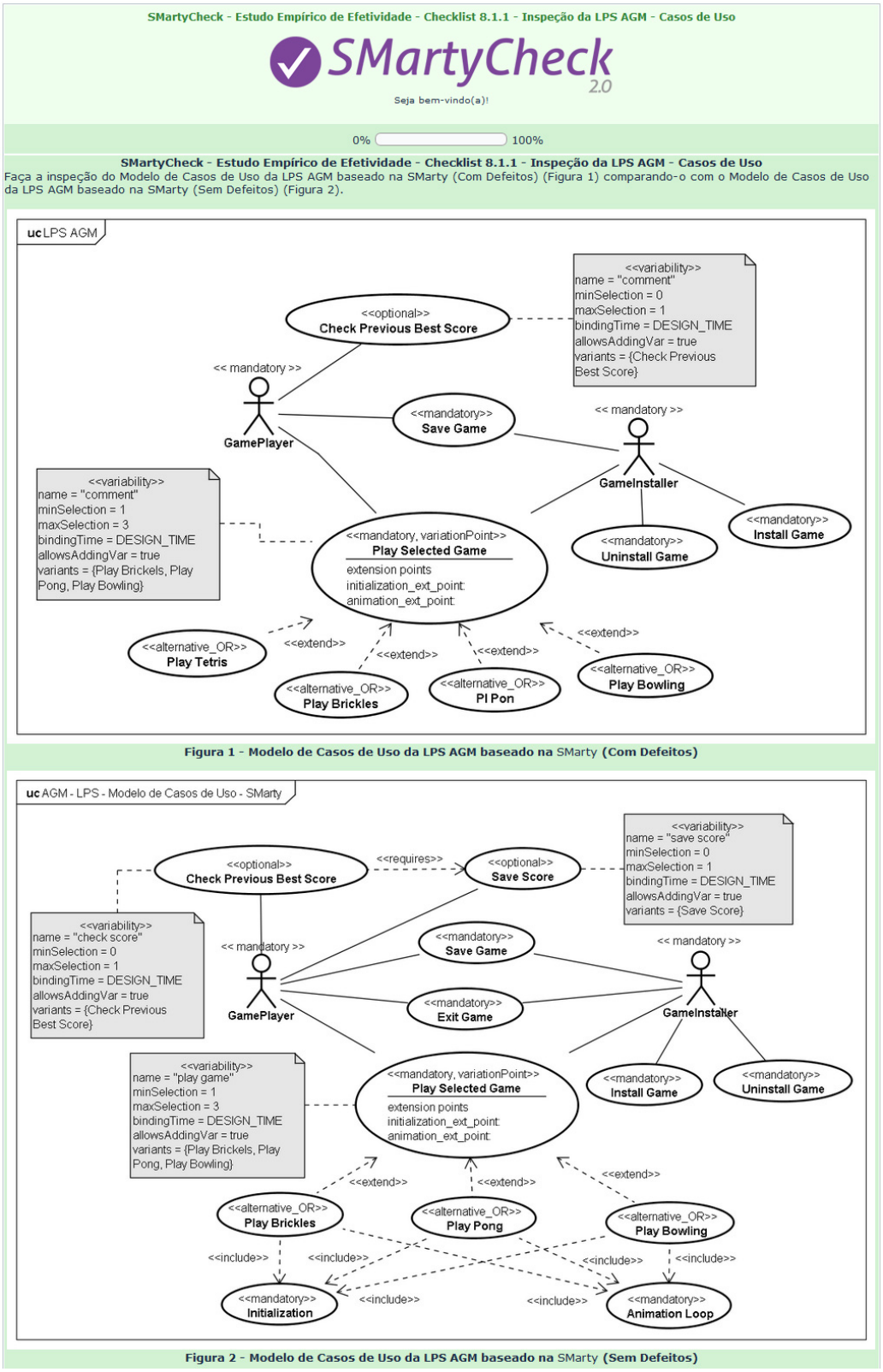


Figura 5.9: 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

Figura 5.10: 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

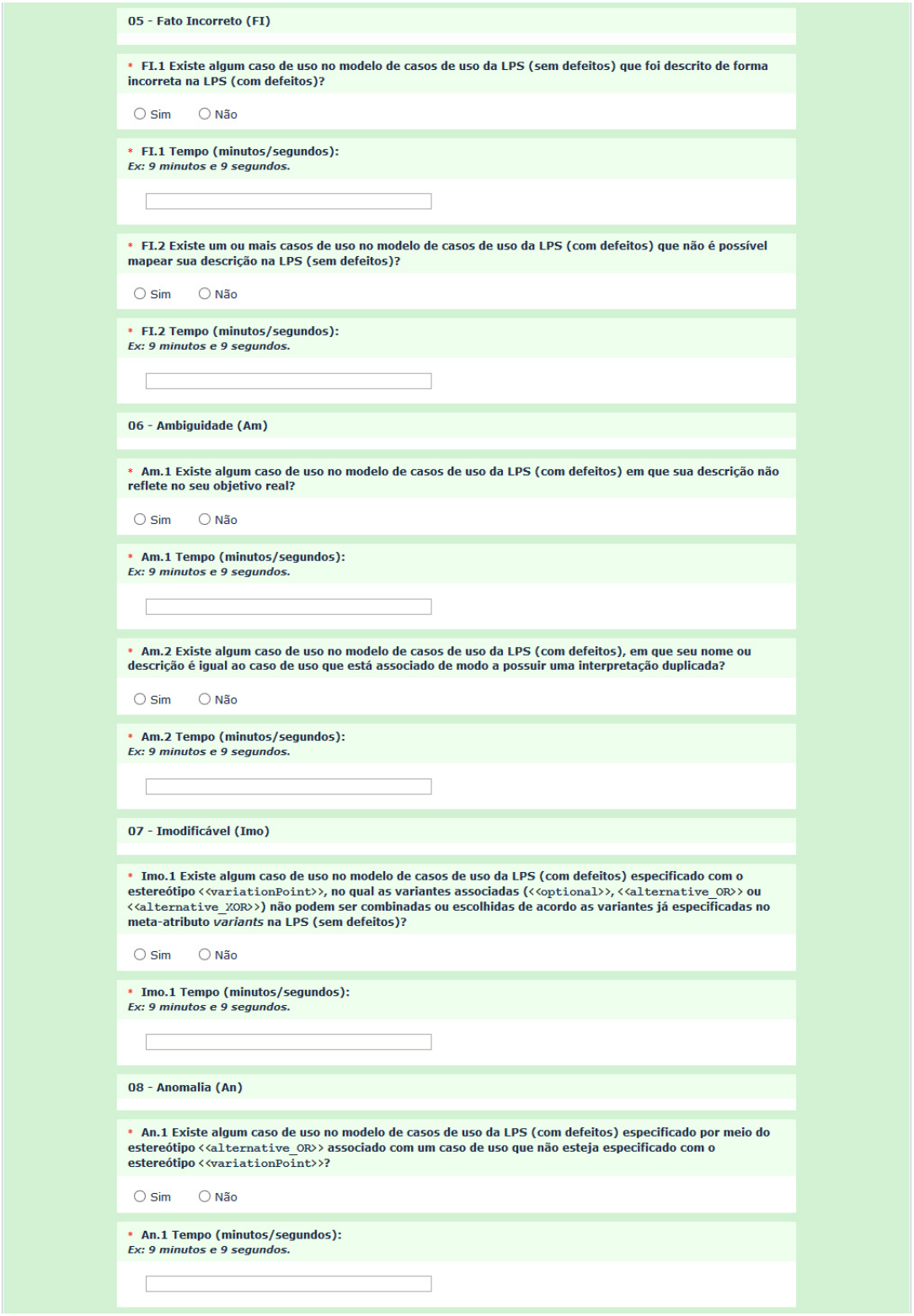


Figura 5.11: 3º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

Figura 5.12: 4º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

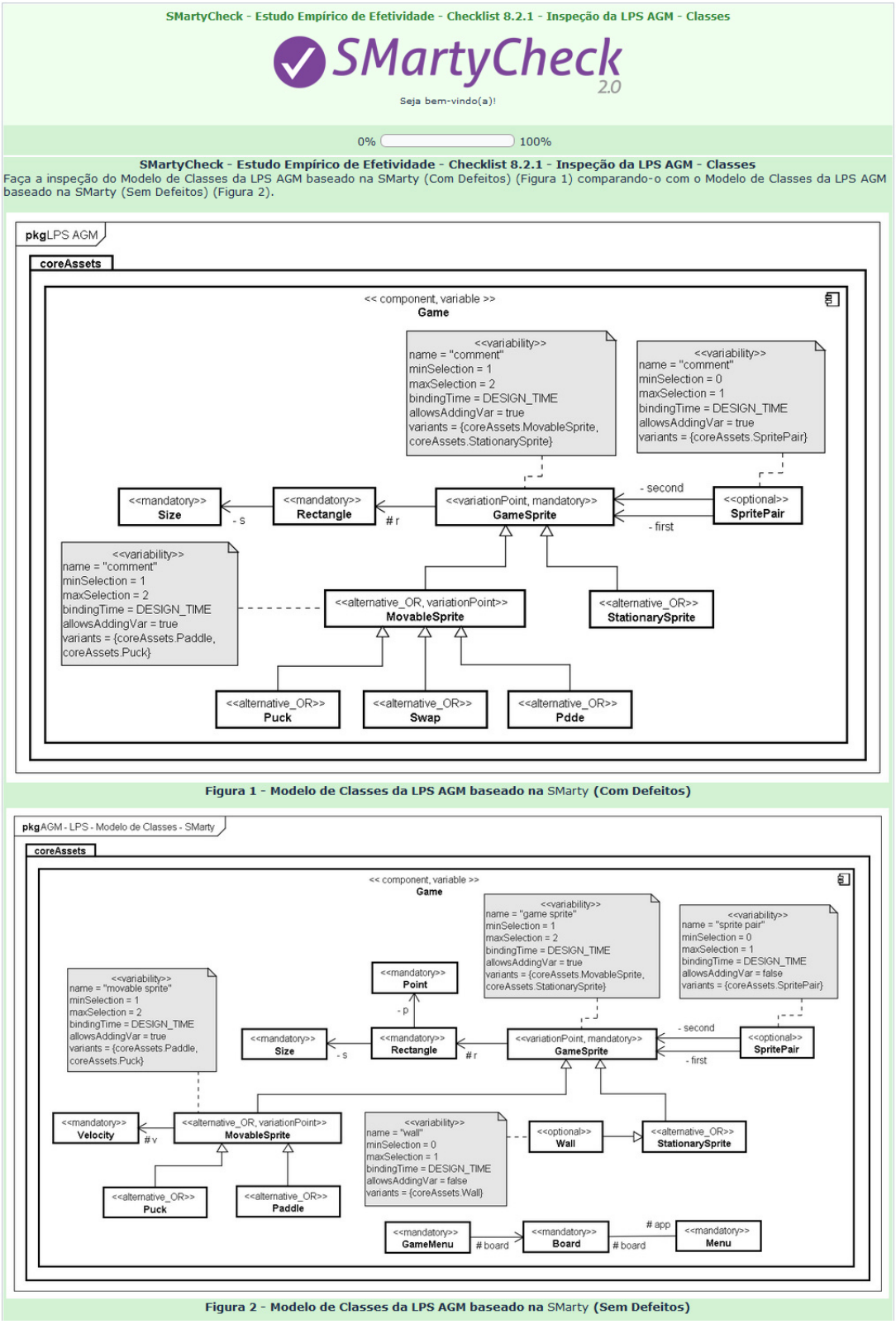


Figura 5.13: 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

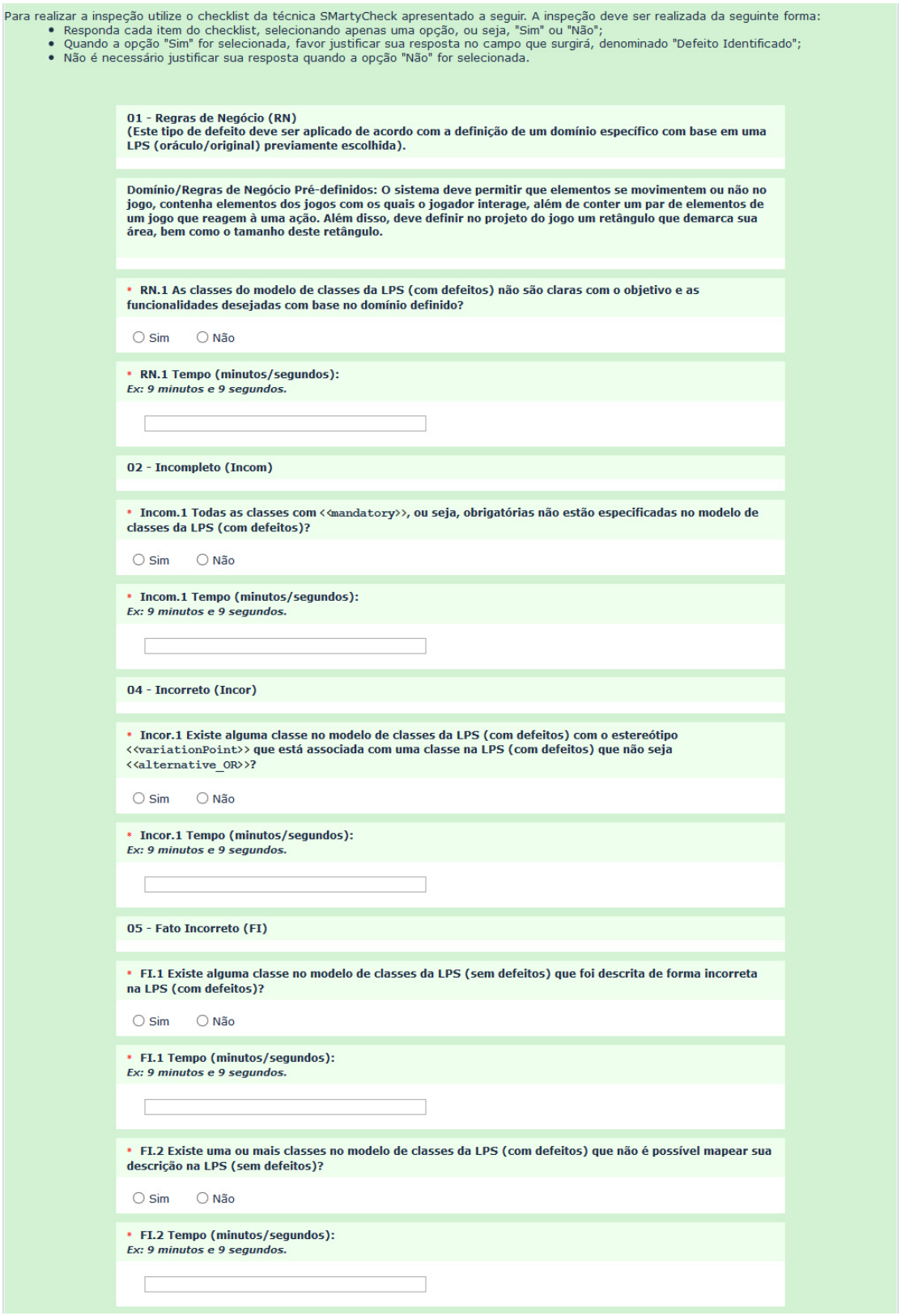


Figura 5.14: 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

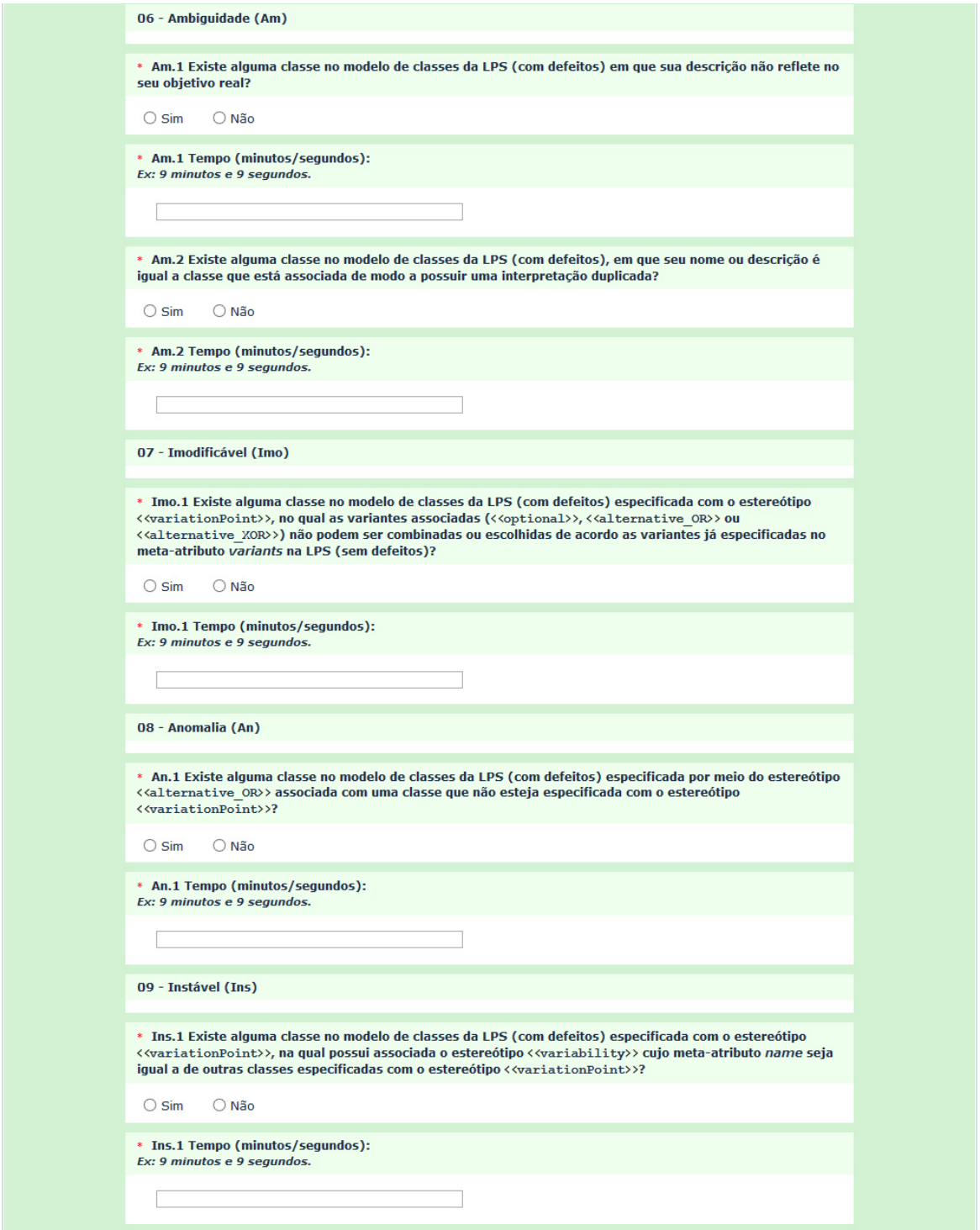


Figura 5.15: 3º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

Figura 5.16: 4º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com o *Checklist* da *SMartyCheck*.

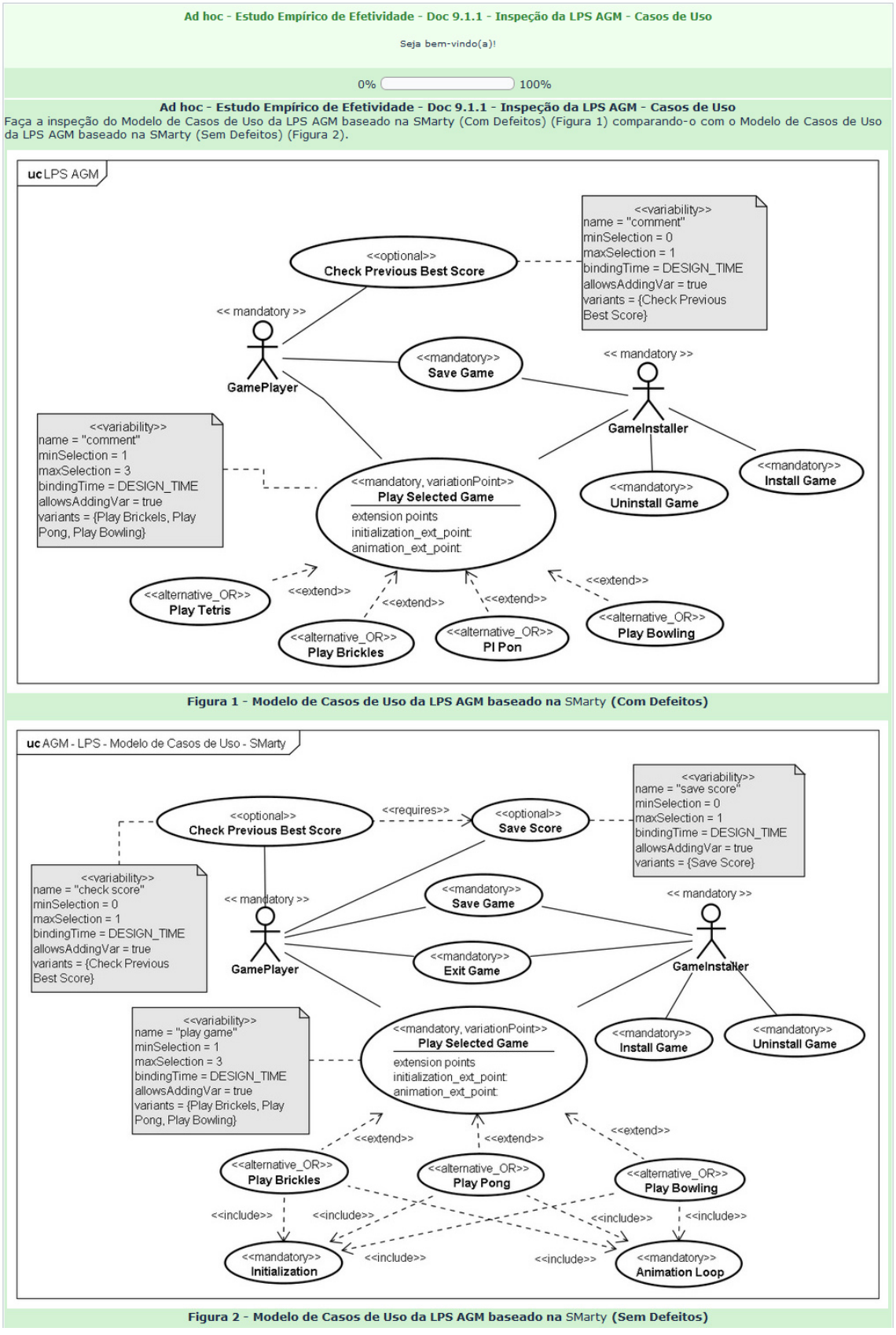


Figura 5.17: 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*.

Figura 5.18: 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Casos de Uso da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*.

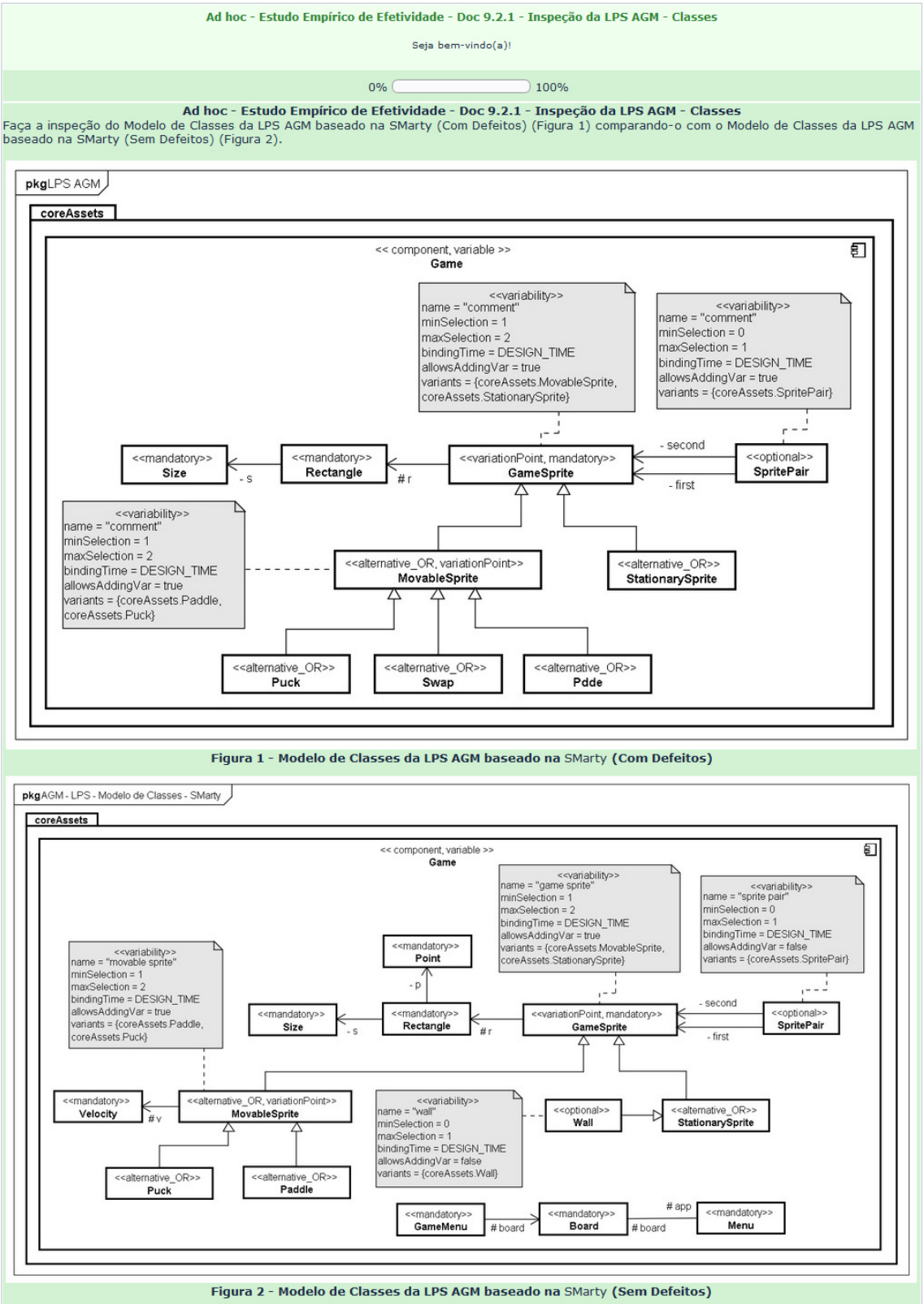


Figura 5.19: 1º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*.

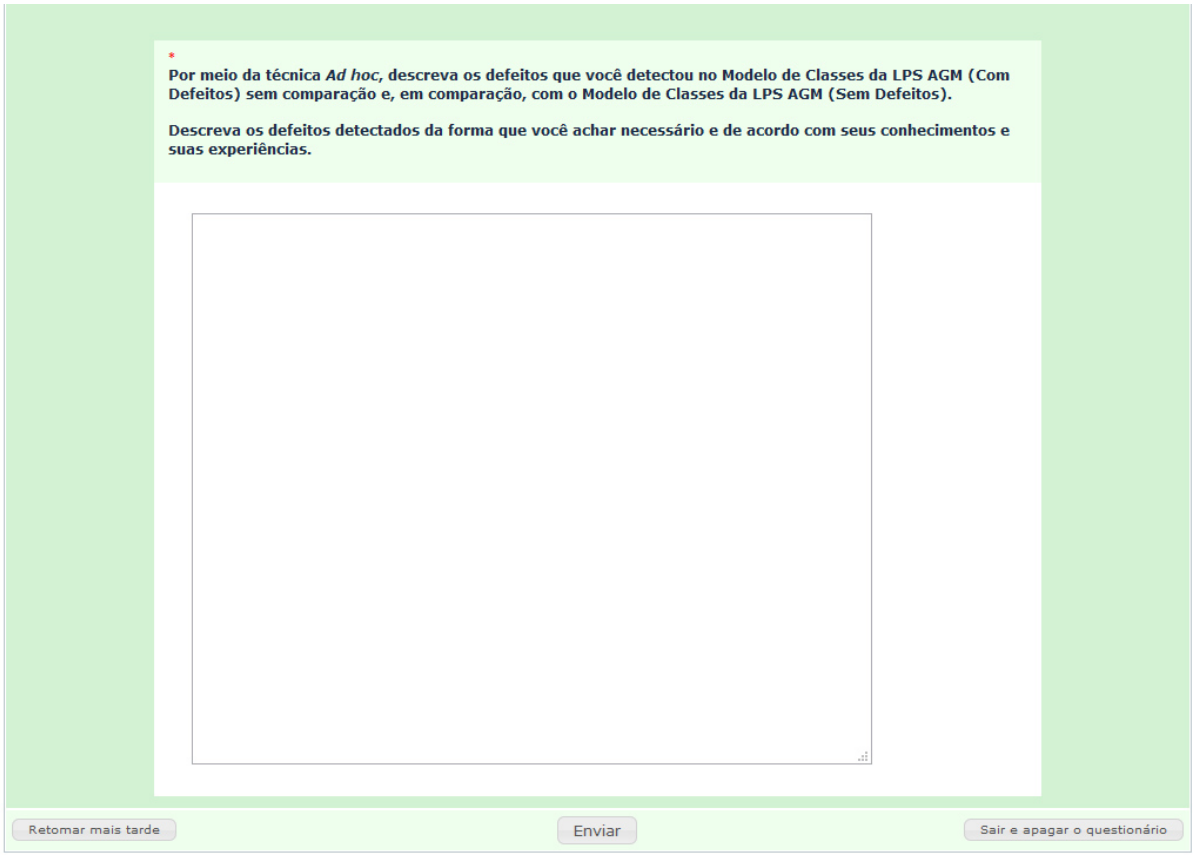


Figura 5.20: 2º Parte - Exemplo de Questionário Eletrônico (Estudo Quantitativo) de Classes da LPS AGM criado no LimeSurvey com a técnica *Ad hoc*.