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

KLEBER LOPES PETRY

***SMartyTesting*: uma abordagem de teste baseado em modelos *SMarty* para linhas de produto de software**

Maringá
2019

KLEBER LOPES PETRY

SMartyTesting: uma abordagem de teste baseado em modelos *SMarty* para linhas de produto de software

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 Junior

Maringá
2019

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

Petry, Kleber Lopes
P498s *SMartyTesting* : uma abordagem de teste baseado em modelos *SMarty* para linhas de produto de software / Kleber Lopes Petry. -- Maringá, 2019.
165 f. : il. color., figs., tabs.

Orientador: Prof. Dr. Edson Alves de Oliveira Junior.
Dissertação (mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2019.

1. Linha de Produto de Software. 2. *SMarty* - Gerenciamento de variabilidade. 3. *SMartyTesting*. 4. Qualidade de Software. I. Oliveira Junior, 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 23.ed. 005.14

Síntique Raquel de C. Eleuterio – CRB 9/1641

FOLHA DE APROVAÇÃO

KLEBER LOPES PETRY

SMartyTesting: uma abordagem de teste baseado em modelos SMarty para linhas de produto de software

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

Prof. Dr. Renato Balancieri
Universidade Estadual do Paraná – CCSA/UNESPAR

Prof. Dr. Avelino Francisco Zorzo
Pontifícia Universidade Católica do Rio Grande do Sul – PPGCC/PUCRS
participação remota

Aprovada em: 18 de novembro de 2019.
Local da defesa: Sala 120, Bloco C56, *campus* da Universidade Estadual de Maringá.

AGRADECIMENTOS

Agradeço a Deus por ter me abençoado e dado forças nesta caminhada, assim como as pessoas que contribuíram de alguma forma na realização deste trabalho. Em especial: À minha esposa e companheira de jornada Cristiane, a qual me apoiou quando eu precisava, incentivando-me a cada desânimo. Minha mãe Ivone que sempre acreditou no meu crescimento pessoal e profissional, com palavras de incentivo e oração.

À minha família pelo carinho, apoio e incentivo.

Ao meu orientador professor Dr. Edson por todo o apoio, paciência com as minhas limitações. Sempre disponível quando precisei, agradeço imensamente pelos comentários, contribuições e sugestões para que este trabalho viesse a se concluir, serei eternamente grato pelos seus ensinamentos.

Gostaria de deixar aqui registrado minha estima e consideração pelos professores da minha graduação e especialização, representados pelo Dr Luiz Fernando. Também aos meus colegas de trabalho do Serviço Nacional de Aprendizagem Industrial (SENAI) aos quais sempre estiveram dispostos a contribuir de alguma forma para que este trabalho fosse possível.

Aos demais professores do curso de Mestrado em Ciência da Computação do Departamento de Informática (DIN) da UEM, agradeço pelos ensinamentos e formação de qualidade que recebi, todos foram fundamentais em minha jornada. Aos colegas das 3 turmas do PCC pela qual passei, onde pude receber um pouco do conhecimento de cada, todos foram fundamentais. Em especial: Viviane, André, Henrique, Eduardo, Renan, Helal, Luciano, Mariane e muitos outros que estimo de coração pela amizade e carinho. Não poderia esquecer o Dr Marcolino pelas contribuições iniciais ao meu trabalho, Dra Aline e o grande Dr Leandro ambos da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) por ter me aturado nos últimos meses de pesquisa e que muito contribuiu para este trabalho.

Agradeço à Inês, por sua simplicidade, paciência, amizade, otimismo, alegria e disponibilidade em sempre poder ajudar de alguma forma.

Agradeço também à empresa *No Magic* pela disponibilização da licença comercial para a ferramenta *Cameo Enterprise Architecture* para o desenvolvimento de parte deste trabalho.

SMartyTesting: uma abordagem de teste baseado em modelos *SMarty* para linhas de produto de software

RESUMO

A utilização de reuso de código e de abordagens de teste no desenvolvimento de software, com a finalidade de garantir e aumentar a produtividade e a qualidade, vem crescendo exponencialmente entre os modelos de processos nas últimas décadas. Linha de Produto de Software (LPS) é um modelo de processo em que o reuso não oportunístico é o cerne do seu desenvolvimento. Levando em consideração a variabilidade inerente aos produtos derivados de uma LPS, uma forma efetiva de garantir a qualidade de tais produtos é a utilização de técnicas de teste. Para gerenciar variabilidades de uma LPS existem diversas abordagens, em especial as baseadas em *Unified Modeling Language* (UML). A abordagem *Stereotype-based Management of Variability (SMarty)* permite realizar tal gerenciamento. *SMarty* guia o usuário na identificação e representação de variabilidades em modelos UML, por meio de estereótipos e meta-atributos. *SMarty* oferece atualmente uma técnica de verificação de seus modelos na forma de inspeção baseada em *checklists*. Porém, *SMarty* não fornece uma forma de validação usando, por exemplo, Teste Baseado em Modelos (TBM). Com base nesse cenário e motivação, a busca por uma abordagem de geração de sequências de teste se faz necessária para a validação dos produtos instanciados. Assim, este trabalho teve como objetivo especificar uma abordagem para auxiliar na geração de sequências de teste, a partir de diagramas de sequência modelados com base em casos de uso e seus fluxos básicos e alternativos. Para avaliar tal abordagem foi realizado um estudo comparativo com outra abordagem existente na literatura, considerando quatro critérios de comparação: complexidade ciclomática, diferenciação das sequências, quantidade de sequências geradas e nível de esforço despendido na utilização da abordagem. Os resultados apontam viabilidade para utilização do modelo de abordagem proposta e, as contribuições são voltadas para a automatização dos processos, diminuição das etapas de tais processos e, suporte à programação concorrente para a ferramenta SPLiT-MBt.

Palavras-chave: Linha de Produto de *Software*. *SMarty*. teste baseado em modelo. variabilidade. qualidade de *software*.

SMartyTesting: a SMarty model-based testing approach for software product lines

ABSTRACT

The use of code reuse and testing approaches in software development to ensure and increase productivity and quality has grown exponentially among process models in recent decades. Software Product Line (SPL) is a process model in which non-opportunistic reuse is the core of its development. Given the inherent variability in products derived from an SPL, an effective way to ensure the quality of such products is to use testing techniques. To manage variability of an SPL there are several approaches, especially those based on UML. The Stereotype-based Management of Variability (SMarty) approach enables such management. SMarty guides the user in identifying and representing variability in UML models through stereotypes and meta-attributes. SMarty currently offers a verification technique for its models in the form of checklist-based inspection. However, SMarty does not provide a form of validation using, for example, Model Based Testing (MBT). Based on this scenario and motivation, the search for a test sequence generation approach is necessary to validate the instantiated products. Thus, this paper aims to specify an approach that assists in the generation of test sequences from sequence diagrams modeled based on use cases and their basic and alternative flows. To evaluate such an approach, a comparative study was performed with another approach in the literature considering four comparison criteria: cyclomatic complexity, sequence differentiation, number of sequences generated and level of effort spent in using the approach. The results indicate the feasibility of using this approach model and the contributions are directed to the automation of processes, reduction of the steps of such processes, concurrent programming support for the SPLiT-MBt tool.

Keywords: Software Product Line. SMarty. Model-Based Testing. Variability. Software Quality.

LISTA DE FIGURAS

Figura - 1.1 Comparação entre desenvolvimento de sistemas únicos e linha de produtos (Weiss e Lai, 1999). 16
Figura - 1.2 Etapas da Metodologia de Desenvolvimento de Pesquisa 17
Figura - 1.3 Desenvolvimento da Abordagem *SMartyTesting* 18

Figura - 2.1 *Framework* de Engenharia de LPS, traduzido de Pohl et al. (2005) 21
Figura - 2.2 Estereótipos e Meta-Atributos do Perfil *SMartyProfile* 5.2 com Suporte a Diagramas de Casos de Uso, Classes, Componentes, Atividades e Sequência (Bera et al., 2015a) 23
Figura - 2.3 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) 24
Figura - 2.4 Diagrama de casos de uso da LPS AGM contendo as notações *SMarty* (Marcolino et al., 2013) 25
Figura - 2.5 Modelo de proposta de implantação do processo de teste de software segundo Crespo et al. (2004) 27
Figura - 2.6 Representação de interesse em testes de SPL: seleção de instâncias do produto para testar. Traduzido de Do Carmo Machado et al. (2014) 27
Figura - 2.7 Esquema simplificado que representa o modelo V de gerenciamento de atividades de engenharia de sistemas. Traduzido de Do Carmo Machado et al. (2014) 28
Figura - 2.8 Ciclo de Processo de TBM, Traduzido de Utting e Legeard (2010) 30
Figura - 2.9 Exemplo de grafo de uma MEF (Costa, 2016) 30
Figura - 2.10 Mapeamento dos trabalhos relacionados ao TBM em LPS 36
Figura - 2.11 A abordagem SPLiT-MBt por Costa (2016) 37
Figura - 2.12 Diagrama de atividades *Game Menu* da LPS AGM (Costa, 2016) 38
Figura - 2.13 Representação da MEF gerada a partir do DA da Figura - 2.12 (Costa, 2016) 39
Figura - 2.14 representação de uma MEF utilizando a ferramenta JPlavisFSM (Pinheiro e Simão, 2012) 43
Figura - 2.15 Geração de sequências de teste pela ferramenta JPlavisFSM (Pinheiro e Simão, 2012) 44
Figura - 2.16 Exemplo de um diagrama de atividades 45
Figura - 2.17 Exemplo de um diagrama de sequência 46

Figura - 3.1 Processo de geração de sequência de teste com *SMartyTesting* (instância da Figura - 2.10) 49
Figura - 3.2 Passos para a geração de sequências de teste usando *SMartyTesting* 50
Figura - 3.3 Control Flow Analysis of UML 2.0 Sequence Diagrams metamodel (Diagramas de Atividades estendido) (Garousi et al., 2005) . . 51
Figura - 3.4 Metamodelo de diagramas de sequência UML (Garousi et al., 2005) 52
Figura - 3.5 Um diagrama de sequência com mensagens assíncronas (Garousi et al., 2005) 52
Figura - 3.6 Resultado do mapeamento do DS da Figura - 3.5. Traduzido de Garousi et al. (2005) 53
Figura - 3.7 Ciclo da etapa 1 55
Figura - 3.8 Diagramas de sequência ilustrando o caso de uso *game menu* da AGM 56
Figura - 3.9 Diagramas de atividades resultante do mapeamento do DS da Figura - 3.8 56
Figura - 3.10 Exemplificação da parametrização dos *controlflows* de um diagrama de atividades usando SPLiT-MBt (Costa, 2016) 57
Figura - 3.11 Ciclo da etapa 2 58
Figura - 3.12 Tela de carregamento de arquivo da ferramenta SPLiT-MBt . . . 59
Figura - 3.13 Validação da estrutura do arquivo carregado pela SPLiT-MBt . . 60
Figura - 3.14 Apresentação da estrutura do arquivo XML 60
Figura - 3.15 Mensagem de sucesso na geração dos casos de teste 61
Figura - 3.16 Arquivo XLSx com os casos de testes gerados 61

Figura - 4.1 Estudo de viabilidade e hipóteses definidas 65
Figura - 4.2 Diagrama de Atividades *Play Selected Game* (Costa, 2016) 68
Figura - 4.3 Diagrama de Atividades *Save Game* (Costa, 2016) 69
Figura - 4.4 Diagrama de Atividades *Pong Moves* (Costa, 2016) 70
Figura - 4.5 Diagrama de Sequência *Play Selected Game* (Marcolino et al., 2017) 73
Figura - 4.6 Diagrama de Atividades resultantes do Diagrama de Sequência da Figura - 4.5 74
Figura - 4.7 Diagrama de Sequência *Save Game* (Marcolino et al., 2017) . . . 76
Figura - 4.8 Diagrama de Atividades resultante dos Diagrama de Sequência da Figura - 4.7 77
Figura - 4.9 Diagrama de Sequência *Pong Moves* 1 (Marcolino et al., 2017) . . 79
Figura - 4.10 Diagramas de Sequência *Pong Moves* 2 (Marcolino et al., 2017) . 80

Figura - 4.11 Diagrama de Atividades resultante do Diagrama de Sequência da Figura - 4.9 e da Figura - 4.10 parte 1 81
Figura - 4.12 Diagrama de Atividades resultante do Diagrama de Sequência da Figura - 4.9 e da Figura - 4.10 parte 2 82

Figura - 5.1 Exemplo de DS com mensagens representando programação concorrente em uma função de *login* 94
Figura - 5.2 DA com mensagens representando programação concorrente em uma função de *login* 95

Figura - 1.1 Visão geral do processo de mapeamento sistemático 116
Figura - 1.2 Processo de pesquisa por MSL 117
Figura - 1.3 Processo Seletivo MSL 123
Figura - 1.4 Processo de extração de MSL 130
Figura - 1.5 Processo de Análise MSL 133
Figura - 1.6 Distribuição de estudos por ano 134
Figura - 1.7 Automação TBM para LPS de abordagens de teste 137
Figura - 1.8 Comparação de níveis e tipos de TBM para LPS 141
Figura - 1.9 Automação TBM para LPS de abordagens de teste 143
Figura - 1.10 Automação de níveis de teste de TBM para LPS 144
Figura - 1.11 Automação TBM para LPS de abordagens de teste 145
Figura - 1.12 Artefatos X Níveis X Modelos 148
Figura - 1.13 Tratamento da variabilidade por tipo de solução 150
Figura - 1.14 Tipos de Avaliação de Propostas e Ambientes 152
Figura - 1.15 TBM4LPS: um roteiro de TBM para LPSs 155
Figura - 1.16 Etapa inicial, mostrando a localização do estudo T31 156
Figura - 1.17 Segunda etapa, mostrando a localização do estudo T31 157
Figura - 1.18 Etapa final mostrando a localização do estudo T31 158

Figura - 2.1 Versão 5.2 do *SMartyProfile*, com suporte a Componentes, Portas, Interfaces e Operações (Bera, 2015). 161
Figura - 2.2 Visão Geral SMarty 5.2 (Bera, 2015). 164
Figura - 2.3 Diagrama de Classes da LPS AGM (OliveiraJr et al., 2010b). . . 165

LISTA DE TABELAS

Tabela - 2.1 Trabalhos selecionados para leitura e extração de dados do MSL . 32
Tabela - 2.2 Sequência de teste gerada a partir da Figura - 2.13 39
Tabela - 2.3 Comparação de requisitos SPLiT-MBt e JPlavisFSM 44

Tabela - 3.1 Regras utilizadas para a conversão de DS para DA (Garousi et al., 2005) 53
Tabela - 3.2 Valores dos atributos de parametrização do DA para utilização em SPLiT-MBt 58

Tabela - 4.1 Diagramas utilizados no processo de geração de sequências de teste. 66
Tabela - 4.2 Sequências de teste do DA *Play Selected Game* da AGM da Figura - 4.2 gerado por SPLiT-MBt 69
Tabela - 4.3 Sequências de teste do DA *Save Game* da Figura - 4.3 geradas por SPLiT-MBt 70
Tabela - 4.4 Sequencia de teste do DA *Pong Moves* da Figura - 4.4 gerada por SPLiT-MBt 71
Tabela - 4.5 Sequências de casos de teste DA da Figura - 4.6 geradas por *SMartyTesting* 75
Tabela - 4.6 Sequências de teste DA da Figura - 4.8 geradas por *SMartyTesting* 78
Tabela - 4.7 Sequências de teste do DA da Figura - 4.11 geradas por *SMarty-Testing* 83
Tabela - 4.7 Sequências de teste do DA da Figura - 4.11 geradas por *SMarty-Testing* 84
Tabela - 4.7 Sequências de teste do DA da Figura - 4.11 geradas por *SMarty-Testing* 85
Tabela - 4.7 Sequências de teste do DA da Figura - 4.11 geradas por *SMarty-Testing* 86
Tabela - 4.8 Quantidade de sequências de teste geradas por abordagem. 87
Tabela - 4.9 Parâmetros aceitáveis para a complexidade ciclomática 88
Tabela - 4.10 complexidade ciclomática apresentada pelas abordagens 89

Tabela - 1.1 String de pesquisa geral do MSL 118
Tabela - 1.2 Fontes de pesquisa eletrônicas definidas 119
Tabela - 1.3 Conferências Definidas e Workshops para Pesquisa Manual 119
Tabela - 1.4 Diários Definidos para Pesquisa Manual 119
Tabela - 1.5 Pesquisadores que avaliaram o protocolo de MSL 121

Tabela - 1.6 Fontes eletrônicas e sequências de pesquisa adaptadas 121
Tabela - 1.7 Número de estudos de fontes eletrônicas 122
Tabela - 1.8 Número de estudos de fontes manuais 122
Tabela - 1.9 Artigos selecionados para leitura completa 124
Tabela - 1.10 Estudos removidos de acordo com os critérios de exclusão 126
Tabela - 1.12 Lista final de estudos 129
Tabela - 1.13 Número de estudos por ano 133
Tabela - 1.14 Avaliação de Qualidade de Estudos 135
Tabela - 1.15 TBM de domínios de aplicativos LPS 136
Tabela - 1.16 Teste de TBM de tipos de solução LPS 139
Tabela - 1.17 Propostas principais de TBM para LPS 140
Tabela - 1.18 Abordagens de TBM para LPS 140
Tabela - 1.19 TBM dos níveis e tipos de testes de LPS 141
Tabela - 1.20 TBM de automação LPS 142
Tabela - 1.21 Artefatos Usados Durante TBM de LPS 144
Tabela - 1.22 Comparação de ferramentas e artefatos primários ou secundários 146
Tabela - 1.23 TBM de LPS Tipos de Modelos 147
Tabela - 1.24 Estudos que tratam da variabilidade durante as atividades da TBM 149
Tabela - 1.25 Tempo de Ligação de Variabilidade em TBM de LPS 150
Tabela - 1.26 Método de Evidenciação da Solução TBM para LPS 151
Tabela - 1.27 Ambientes de evidência de soluções TBM para LPS 152
Tabela - 1.28 Estudos com Suporte de Rastreabilidade para TBM de LPSs . . . 153

LISTA DE SIGLAS E ABREVIATURAS

AGM: *Arcade Game Maker*
ATL: *Atlas Transformation Language*
CAPES: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CCFG: *Constraint Control Flow Graph*
CNPq: Conselho Nacional de Desenvolvimento Científico e Tecnológico
DA: Diagrama de Atividade
DIN: Departamento de Informática
DS: Diagrama de Sequência
FSM: *Finite State Machine*
GQM: *Goal Question Metric*
HSI: *Harmonized State Identifiers*
IDE: *Integrated Development Environment*
LPS: Linha de Produto de Software
MEF: Máquina de Estado Finito
MSL: Mapeamento Sistemático de Literatura
OCL: *ObjectConstraint Langage*
OMG: *Object Management Group*
PLUS: *Product Line UML-based Software Engineering*
PUCRS: Pontifícia Universidade Católica do Rio Grande do Sul
QVT: *Query/View/Transformation*
ROI: *Return on Investment*
RSL: Revisão Sistemática de Literatura
SEI: *Software Engineering Institute*
SENAI: Serviço Nacional de Aprendizagem Industrial
SMarty: *Stereotype-based Management of Variability*
SPLit-MbT: *Software Product Line Testing Method Based on System Models*
SUT: *System Under Test*
TBM: Teste Baseado em Modelo
TT: *Transition Tour*
UEM: Universidade Estadual de Maringá
UML: *Unified Modeling Language*
XML: *eXtensible Markup Language*

SUMÁRIO

1 Introdução **14**

1.1 Contextualização 14

1.2 Motivação e Justificativa 15

1.3 Objetivo 16

1.4 Metodologia de Desenvolvimento 17

1.5 Organização do Texto 19

2 Fundamentação Teórica **20**

2.1 Considerações Iniciais 20

2.2 Linha de Produto de Software e a Abordagem SMarty 20

2.3 Teste de Linha de Produto de Software 25

2.4 Teste Baseado em Modelos para LPS 28

2.5 A Abordagem SPLiT-MBt 37

2.5.1 Métodos de Geração de Casos de Teste 39

2.5.2 Ferramenta de Geração de Sequências de Teste JPlavisFSM 42

2.6 Diagramas de Sequência e de Atividades 44

2.7 Considerações Finais 46

3 A Abordagem SMartyTesting **47**

3.1 Considerações Iniciais 47

3.2 Caracterização da Abordagem *SMartyTesting* 47

3.2.1 Definição do Processo Adotado pela Abordagem *SMartyTesting* . . 48

3.2.2 Modelos Utilizados 48

3.2.3 Conversão de Diagramas de Sequência para Diagramas de Atividades 50

3.2.4 Ferramenta Utilizada 54

3.3 Especificação da Abordagem *SMartyTesting* 54

3.3.1 Etapa 1 - Mapeamento de Diagramas de Sequência para Diagramas de Atividades 55

3.3.2 Etapa 2 - Geração de Sequências de Teste 58

3.3.3 Resolução da Variabilidade 60

3.3.4 Limitações de Uso da SPLiT-MBt 61

3.4 Considerações Finais 62

4 Estudo de Viabilidade **63**

4.1 Considerações Iniciais 63

4.2 Objetivo 63
4.3 Planejamento 64
4.3.1 Critérios de Avaliação 64
4.3.2 Hipóteses do Estudo 64
4.3.3 Instrumentação 66
4.3.4 Procedimentos de Análise 66
4.4 Execução 67
4.4.1 Geração de Sequências de Teste com SPLiT-MBt 67
4.4.2 Geração de Sequências de Teste com *SMartyTesting* 72
4.5 Análise e Interpretação 87
4.5.1 Quantidade de Sequências de Teste Geradas (CT.1) 87
4.5.2 Diferença entre as Sequências Geradas (CT.2) 87
4.5.3 Complexidade na Geração de Sequências de Teste (CT.3) 88
4.5.4 Esforço na Utilização das Abordagens (CT.4) 89
4.6 Ameaças à Validade 89
4.6.1 Validade Interna 90
4.6.2 Validade Externa 90
4.6.3 Validade de *Constructo* 90
4.6.4 Validade de Conclusão 90
4.7 Considerações Finais 91

5 Melhorias Identificadas e Lições Aprendidas 92
5.1 Considerações Iniciais 92
5.2 Melhorias Identificadas 92
5.2.1 Automatização do Processo de Geração de Sequências de Teste . . 92
5.2.2 Utilização da Conversão de DS para MEF em *SMartyTesting* . . . 93
5.2.3 Suporte para Programação Concorrente e Sub-processo em SPLiT-MBt 94
5.2.4 Suporte para Outro Métodos de Geração de Sequências de Teste . . 95
5.3 Lições Aprendidas 96
5.3.1 Compreensão de SPLiT-MBt 96
5.3.2 Conversão Manual de DS para DA 96
5.3.3 ATLs para Conversão 96
5.3.4 Uso de Ferramentas 97
5.3.5 Teste Baseado em Modelos de LPS 98
5.4 Considerações Finais 98

6 Conclusão **99**
6.1 Contribuições 99
6.2 Limitações 100
6.3 Trabalhos Futuros 101

REFERÊNCIAS **102**

A Teste Baseado em Modelos para LPS: um Mapeamento Sistemático da Literatura **115**
A.1 Planejamento do MSL 115
A.1.1 Objetivo da Pesquisa 115
A.2 Metodologia de Pesquisa 115
A.2.1 Objetivo e Questões de Pesquisa 116
A.2.2 Processo de Pesquisa 117
A.2.3 Processo de Seleção 122
A.2.4 Processo de Extração 130
A.2.5 Processo de Análise 133
A.3 Resultados 133
A.3.1 Caracterização dos Estudos 133
A.3.2 Resultados em questões de pesquisa 136
A.3.3 Procedimentos de compartilhamento de dados do MSL 153
A.4 TBM4LPS:um roteiro para testes baseados em modelos de pesquisa de LPSs 154
A.4.1 Rota Primária Baseada no Estudo 156
A.4.2 Rota baseada em critérios 158
A.5 Observações Conclusivas 158

B Anexo: Abordagem *SMarty* **160**

1

Introdução

1.1 Contextualização

O mercado tem se tornado cada vez mais competitivo, isso faz com que situações para diminuição dos custos de operação sejam adotadas, dentre tantas, pode-se citar o reuso de código, assim, técnicas para utilização desse meio são operacionalizadas, ainda mais com a inclusão de indicadores de retorno de investimento, um exemplo é o da Figura - 1.1 que permitem realizar análise mais aprimorada a respeito do sucesso ou insucesso de um produto de software (Weiss e Lai, 1999).

Com base nesse cenário, agilidade no desenvolvimento e qualidade de um produto tornam-se fatores decisivos. A reutilização de código e diminuição de etapas no processo de desenvolvimento de software são técnicas que veem sendo adotadas pela acadêmia e pela indústria ao longos dos anos (Delamaro et al., 2017).

Abordagens veem sendo desenvolvidas com o propósito de aumentar a reusabilidade de software e, consequentemente, o retorno de investimento (ROI) (Delamaro et al., 2017). Entre essas abordagens está Linha de Produto de Software (LPS). LPS é uma alternativa de utilização como processo de desenvolvimento baseado em reuso de software, para se obter maior produtividade, reduzir custo, tempo e risco e proporcionar mais qualidade ao produto (Linden et al., 2007).

Uma LPS é um conjunto de sistemas de software que compartilham características (*features*) comuns e gerenciáveis, que satisfazem as necessidades de um segmento particular ou de uma missão (Clements e Northrop, 2002). Esse conjunto de sistemas é denominado também de família de produtos.

Uma LPS possui artefatos que podem ser reutilizados, assim, tendo em vista esse novo desafio de gerenciamento durante o desenvolvimento de software, os artefatos produzidos nos processos tradicionais de desenvolvimento de software já não são tão eficientes para o contexto de LPS, pois em sua maior parte não possuem suporte à variabilidade (Clements e Northrop, 2002), em especial àquelas baseadas em UML. Assim, visando contornar esse problema, algumas abordagens têm sido propostas.

Para o gerenciamento de variabilidade em LPS, *Product Line UML-based Software Engineering* (PLUS) (Gomaa, 2006), (Ziadi et al., 2003) e *Stereotype-based Management of Variability* (*SMarty*) (OliveiraJr et al., 2010a) são exemplos de tais abordagens. Essas abordagens usam estereótipos UML para representar variabilidade em elementos UML de uma LPS. Assim, dada uma LPS, um de seus maiores desafios é o seu teste, especialmente baseado em modelos que contém variabilidade, pois se torna inviável testar todos os possíveis produtos de uma LPS.

1.2 Motivação e Justificativa

Embora teste de software ser amplamente difundido, assim como os conceitos de LPS, sua utilização de forma errônea, sem o conhecimento prévio ou de forma equivocada, sempre proporcionam pontos em que surgem dúvidas e onde deve-se tomar atenção com relação ao resultado final do desenvolvimento. Isso impacta diretamente na qualidade e custo de um produto de software, principalmente se tratando de uma LPS (Weiss e Lai, 1999).

Na Figura - 1.1 é apresentada uma comparação entre um cenário de desenvolvimento de sistema único e o de linha de produto. Weiss e Lai (1999) afirmam com base em estudos que, o custo no desenvolvimento de sistemas únicos se eleva quando se aumenta a quantidade de produtos diferentes.

Além do processo de execução e desafios da operação de teste em uma LPS, deve-se considerar as particularidades individuais de cada produto gerado e assim, tem-se um ponto que demanda atenção especial, que é a variabilidade (Engström e Runeson, 2011). Testar uma LPS levando em consideração uma ampla cadeia de variações de produtos é um grande desafio, impactando diretamente nas métricas de qualidade seja quais que foram definidas (OliveiraJr et al., 2013).

Existe um desafio muito grande na realização de testes para produtos derivados de LPS, pela quantidade de produtos semelhantes, e por suas particularidades que os diferem uns dos outros. Sobre esse ponto, considera-se neste trabalho que a variabilidade é gerenciada utilizando-se a abordagem *SMarty* (OliveiraJr et al., 2010a), visto que possui um perfil que permite que inspeções baseadas em *checklist* sejam realizadas e que já

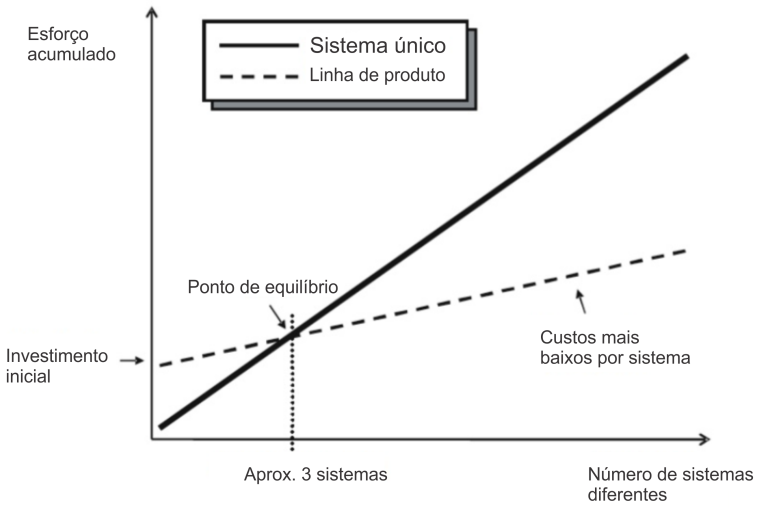


Figura 1.1: Comparação entre desenvolvimento de sistemas únicos e linha de produtos (Weiss e Lai, 1999).

foi amplamente avaliado. Isso motivou, a criação de um perfil voltado para a geração de sequências de testes baseados em modelos *SMarty*, contribuindo, assim, para a antecipação dos casos de teste no ciclo de vida de LPS.

Outro motivo que norteia este trabalho é a suposição de que diagramas de sequência (DS) possam produzir mais sequências de teste que diagramas de atividades (DA). Nesse caso, uma sequência pode conter mais casos de testes que várias sequências com poucos casos de teste. Dessa forma, a questão de pesquisa deste trabalho é **"Diagramas de sequência podem gerar mais sequências de teste do que diagramas de atividades?"**

1.3 Objetivo

Conforme a motivação apresentada, esta dissertação tem como objetivo principal especificar e viabilizar o desenvolvimento da abordagem *SMartyTesting* para teste baseado em modelos *SMarty* para LPS, considerando como modelo de partida diagramas de casos de uso e diagramas de sequência pelo alto grau de detalhes, informações, pelos quais pode ser representado.

Assim, tem-se como objetivos específicos:

- investigar na literatura, soluções que utilizam diagramas de sequência para o teste baseado em modelo de LPS;

- identificar uma forma de utilização de diagramas de sequência para a geração de sequências de testes que possam ser utilizados na criação de casos de teste; e
- analisar a viabilidade da abordagem proposta, com base em resultados de uma comparação entre diagramas de sequência e diagramas de atividades.

1.4 Metodologia de Desenvolvimento

Para atingir o objetivo proposto neste trabalho foi necessário realizar as etapas apresentadas na Figura - 1.2 como descrito a seguir:

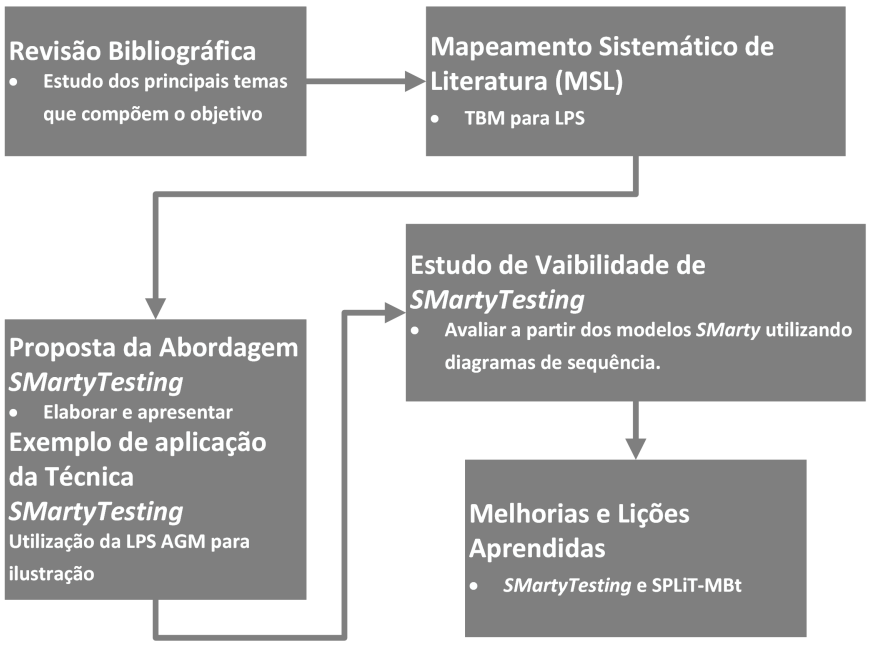


Figura 1.2: Etapas da Metodologia de Desenvolvimento de Pesquisa

- **Revisão Bibliográfica:** foram estudados os conceitos sobre os principais temas que fazem parte do objetivo da pesquisa, tais como: perfis UML, diagramas de sequência, teste de LPS, conceitos de LPS, gerenciamento de variabilidade, a abordagem *SMarty* e a abordagem SPLiT-MBt;
- **Estudo secundário:** foi realizado um Mapeamento Sistemático de Literatura (MSL), em que o tema abordado foi TBM para LPS. Tal estudo encontra-se no Apêndice A. A análise dos estudos do MSL permitiu a identificação de formas diferenciadas de aplicação de TBM em LPS e permitiu embasar este trabalho;

- **Proposta da Abordagem *SMartyTesting*:** foi definido um fluxo de trabalho para concepção e construção da abordagem, cujas etapas que a antecedem foram o embasamento teórico e a realização do mapeamento sistemático. Inicialmente foi realizado um trabalho de Mapeamento Sistemático de Literatura para embasamento da abordagem, após a análise dos dados, foi realizada a concepção da abordagem com os passos necessários para se alcançar o objetivo principal. Na fase de concepção, uma pesquisa sobre ferramentas de apoio foi desenvolvida e, em seguida, as mesmas foram validadas, para se saber qual delas poderia ser utilizada na abordagem *SMartyTesting*. Por fim, foi realizado um estudo de viabilidade comparativo com SPLiT-MBt para a validação, e a proposta final foi concluída;

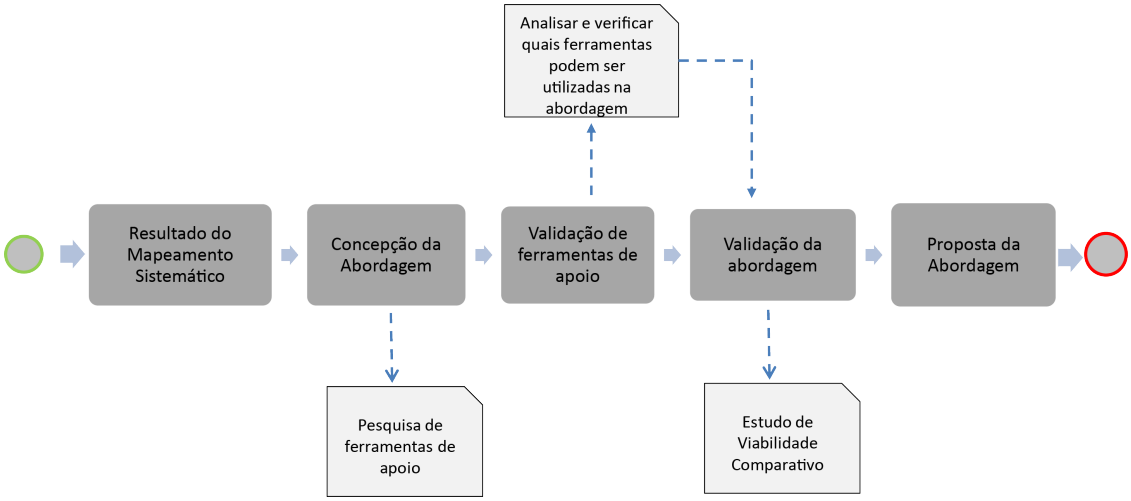


Figura 1.3: Desenvolvimento da Abordagem *SMartyTesting*

- **Estudo de Viabilidade de *SMartyTesting*:** foi realizado um estudo de viabilidade, considerando a abordagem *SMartyTesting*, utilizando diagramas de sequência, com a abordagem SPLiT-MBt que utilizou diagramas de atividades. Com relação aos critérios de análise foram definidos quatro critérios que estão detalhados na Seção 4.3.1; e

- **Melhorias e Lições Aprendidas:** as melhorias identificadas relacionadas às abordagens *SMartyTesting* e SPLiT-MBt são discutidas com base nos resultados do estudo de viabilidade bem como as lições aprendidas com este trabalho (Tópico 5).

1.5 Organização do Texto

O trabalho está organizado como segue. No tópico 2 é apresentada a fundamentação teórica sobre LPS e a abordagem *SMarty*, teste de LPS, Teste Baseado em Modelo (TBM) de LPS e a abordagem SPLiT-MBt que é uma das bases para este trabalho e diagramas de sequência e de atividades. No tópico 3 é descrita a abordagem proposta, assim como processos, modelos, ferramentas e meios para a aplicação e utilização de *SMartyTesting*. O tópico 4 apresenta o estudo de viabilidade realizado. No tópico 5 são discutidas as melhorias identificadas e lições aprendidas para as abordagens *SMartyTesting* e SPLiT-MBt. O tópico 6 apresenta conclusões, limitações e trabalhos futuros desta dissertação.

2

Fundamentação Teórica

2.1 Considerações Iniciais

Este tópico apresenta a fundamentação teórica essencial para a compreensão deste trabalho que engloba: Linha de Produto de Software (LPS) e a abordagem de gerenciamento de variabilidade *SMarty*, teste de LPS, Teste Baseado em Modelo (TBM) de LPS e a abordagem SPLit-MbT.

Neste trabalho é considerada a abordagem *SMarty* por permitir o gerenciamento sistemático de variabilidade em LPS, baseada em UML por meio de um perfil e um processo bem definido e avaliada empiricamente. Além disso, *SMarty* apoia o TBM na abordagem SPLiT-MBt.

A qualidade dos produtos de uma LPS é imprescindível, assim como no método tradicional de desenvolvimento de software. Para tanto, atividades de verificação e validação devem ser realizadas. Dessa forma, este tópico é essencial para o entendimento da abordagem proposta, a *SMartyTesting*.

2.2 Linha de Produto de Software e a Abordagem SMarty

Uma Linha de Produto de Software (LPS) é um conjunto de sistemas que compartilham características comuns e gerenciáveis (Clements e Northrop, 2002), também denominado de família de produtos.

O conceito de LPS tem como objetivo principal o desenvolvimento de produtos de software baseado em reutilização e a migração para uma cultura de desenvolvimento em

que novos sistemas são derivados a partir de um conjunto de componentes e artefatos comuns, os quais constituem o núcleo de artefatos de uma LPS (Linden et al., 2007).

Pohl et al. (2005) desenvolveram um *framework* para engenharia de LPS, cujo objetivo é incorporar os conceitos centrais da engenharia de linha de produto tradicional, proporcionando a reutilização de artefatos e a customização em massa por meio de variabilidades. Tal *framework* é dividido em duas atividades principais, conforme ilustra a Figura - 2.1:

- **Engenharia de Domínio:** atividade em que as similaridades e as variabilidades das LPSs são identificadas e representadas; e
- **Engenharia de Aplicação:** atividade em que os produtos específicos de uma LPS são construídos por meio da reutilização de artefatos de domínio, explorando as variabilidades de uma LPS.

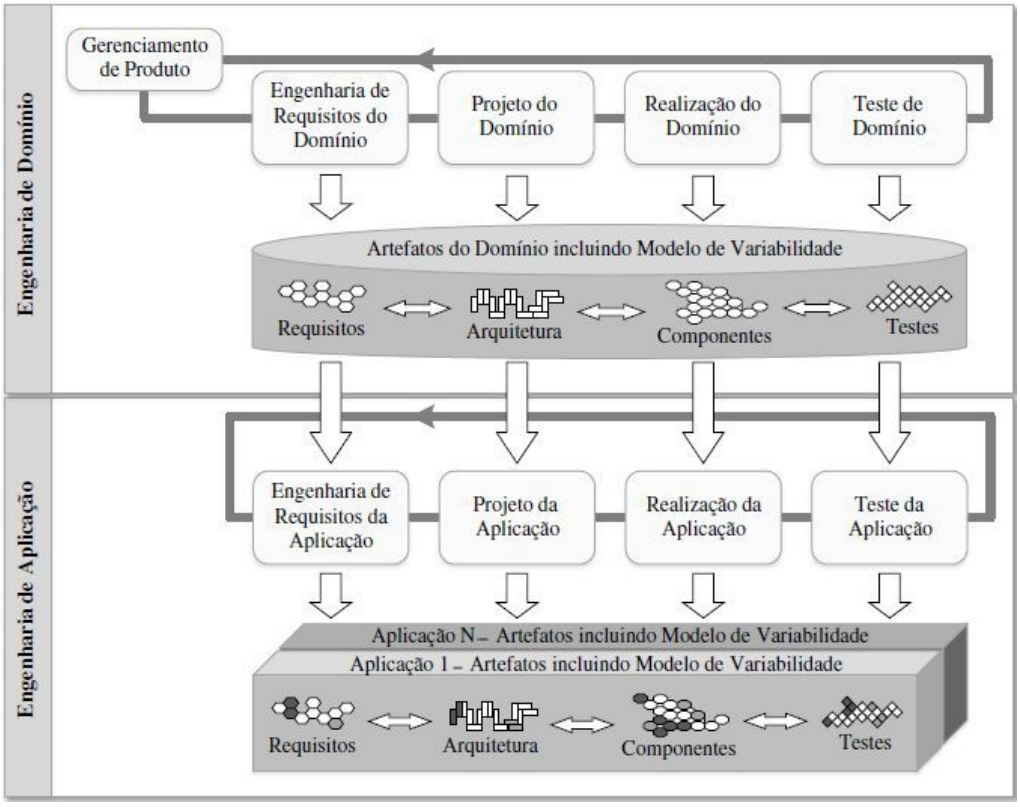


Figura 2.1: *Framework* de Engenharia de LPS, traduzido de Pohl et al. (2005)

Variabilidades são descritas por: Ponto de Variação, que permite a resolução de variabilidades em artefatos genéricos de uma LPS; Variante, que representa os possíveis elementos que podem ser escolhidos para resolver um ponto de variação e; Restrições entre

variantes que 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 (Apel et al., 2013a; Linden et al., 2007; Pohl et al., 2005).

Existem abordagens voltadas para o gerenciamento de variabilidade, especialmente as baseadas em UML. Dentre as quais, pode-se citar a *Product Line UML-based Software Engineering* (PLUS) (Gomaa, 2006) e a *Stereotype-based Management of Variability* (*SMarty*) (OliveiraJr et al., 2010a).

Neste trabalho considera-se *SMarty*, que realiza o gerenciamento de variabilidades de uma LPS fazendo uso de diagramas UML, e é composta de um perfil UML denominado *SMartyProfile* e de um processo denominado *SMartyProcess*.

SMarty guia o usuário por meio do *SMartyProcess* na identificação e representação de variabilidades de uma LPS. O perfil *SMartyProfile* é formado por um conjunto de estereótipos e meta-atributos para representar variabilidades em modelos UML de LPS.

O *SMartyProfile* permite ao usuário a aplicação dos estereótipos de forma clara e objetiva (Fiori et al., 2012; OliveiraJr et al., 2010a, 2013) e se baseia no inter-relacionamento dos principais conceitos de LPS no que tange ao gerenciamento de variabilidade. Tais conceitos são aplicados aos elementos de interesse do metamodelo da UML. Com base no relacionamento entre os conceitos de gerenciamento de variabilidade e os modelos UML, a Figura - 2.2 apresenta o perfil UML *SMartyProfile* 5.1.

O núcleo de artefatos forma a base da LPS e inclui a arquitetura de LPS, componentes reusáveis, modelos de domínios, requisitos da LPS, planos de testes e modelos de características (*features*) e de variabilidades.

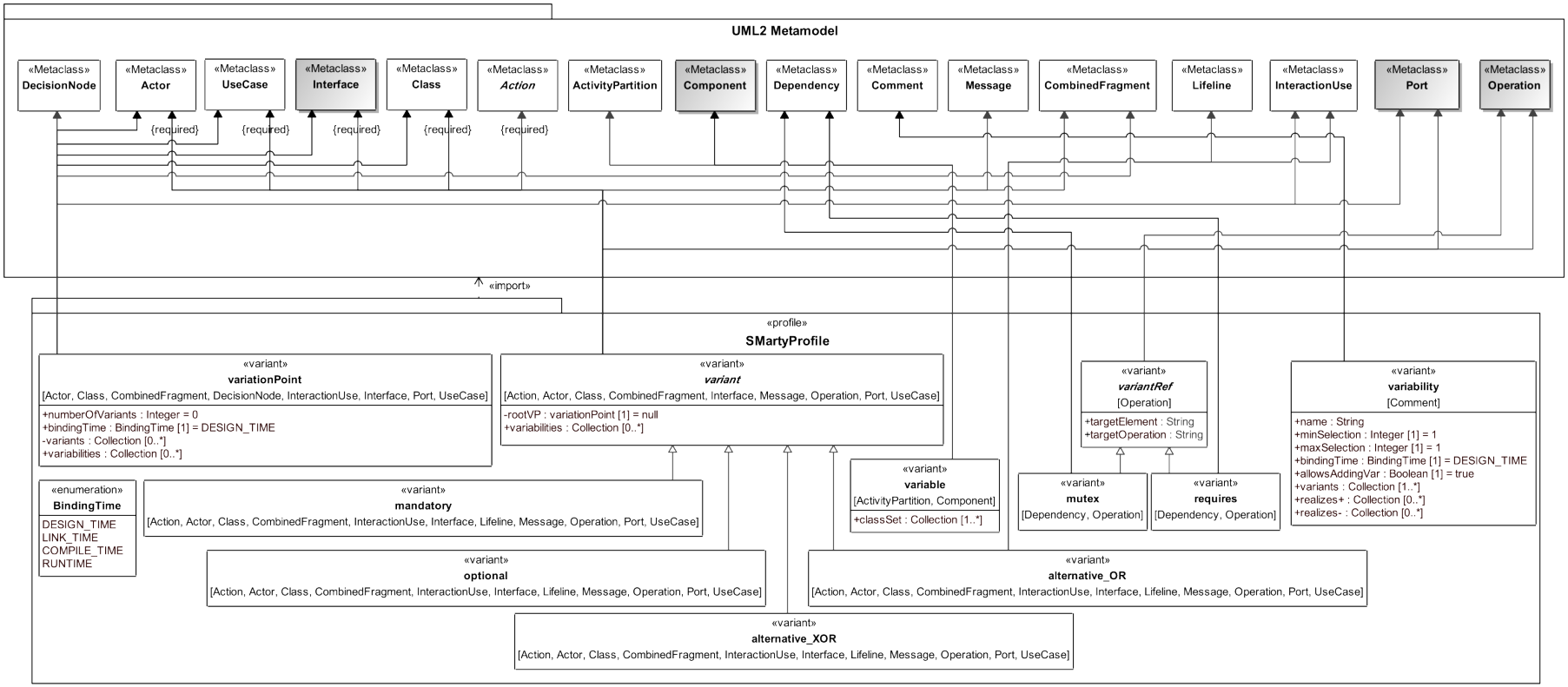


Figura 2.2: Estereótipos e Meta-Atributos do Perfil *SMartyProfile* 5.2 com Suporte a Diagramas de Casos de Uso, Classes, Componentes, Atividades e Sequência (Bera et al., 2015a)

O *SMartyProcess* é um processo sistemático que guia o usuário na identificação, delimitação, representação, rastreamento de variabilidades e análise de configurações de produtos de uma LPS e segue as atividades gerais relacionadas às especificadas no processo de desenvolvimento de LPS (Pohl et al., 2005). Tal relacionamento pode ser observado por meio da Figura - 2.3, em um diagrama de atividades da UML. Nela é possível observar o processo genérico de Desenvolvimento de Linha de Produto, representado pelas atividades alinhadas no lado esquerdo e o *SMartyProcess* representado pelas atividades do retângulo à direita (OliveiraJr et al., 2005). Mais informações relacionadas à *SMarty* podem ser encontradas no Anexo B - Abordagem *SMarty*.

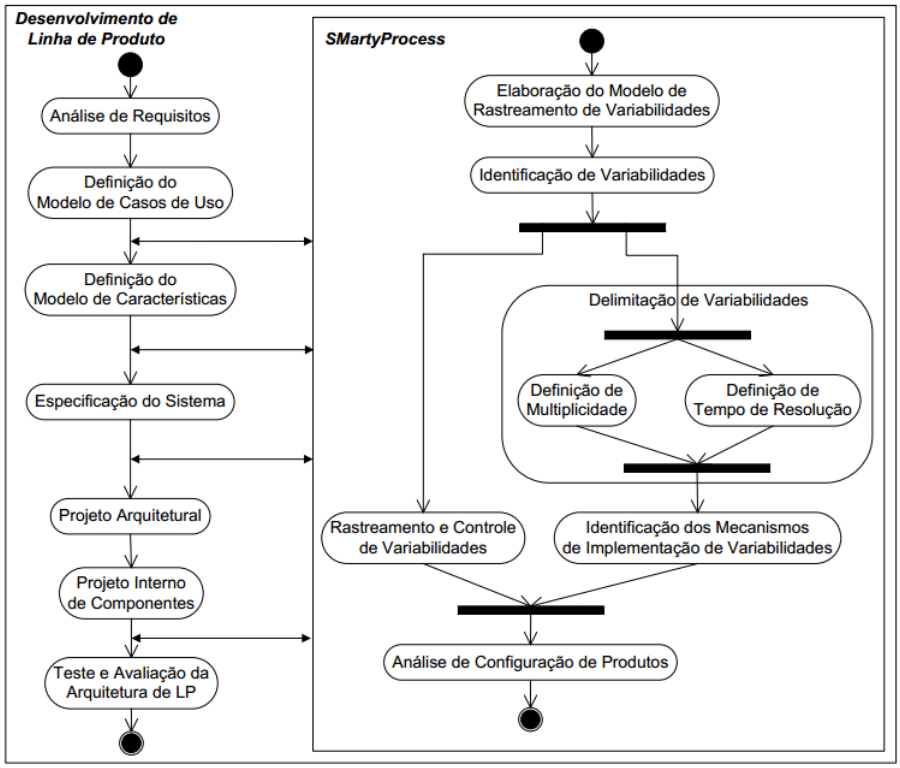


Figura 2.3: O Processo de Gerenciamento de Variabilidades *SMartyProcess* e sua Interação entre as Atividades com o Processo de Desenvolvimento de LPS, traduzido de OliveiraJr et al. (2005)

Na Figura - 2.4 é apresentado um diagrama de casos de uso da LPS *Arcade Game Maker* (AGM) (SEI, 2009) contendo notações de variabilidade modeladas de acordo com *SMarty*.

Na Figura - 2.4 pode-se perceber que existem representações de esteriótipos com itens representando a variabilidade. Para a representação do ponto de variação tem-se

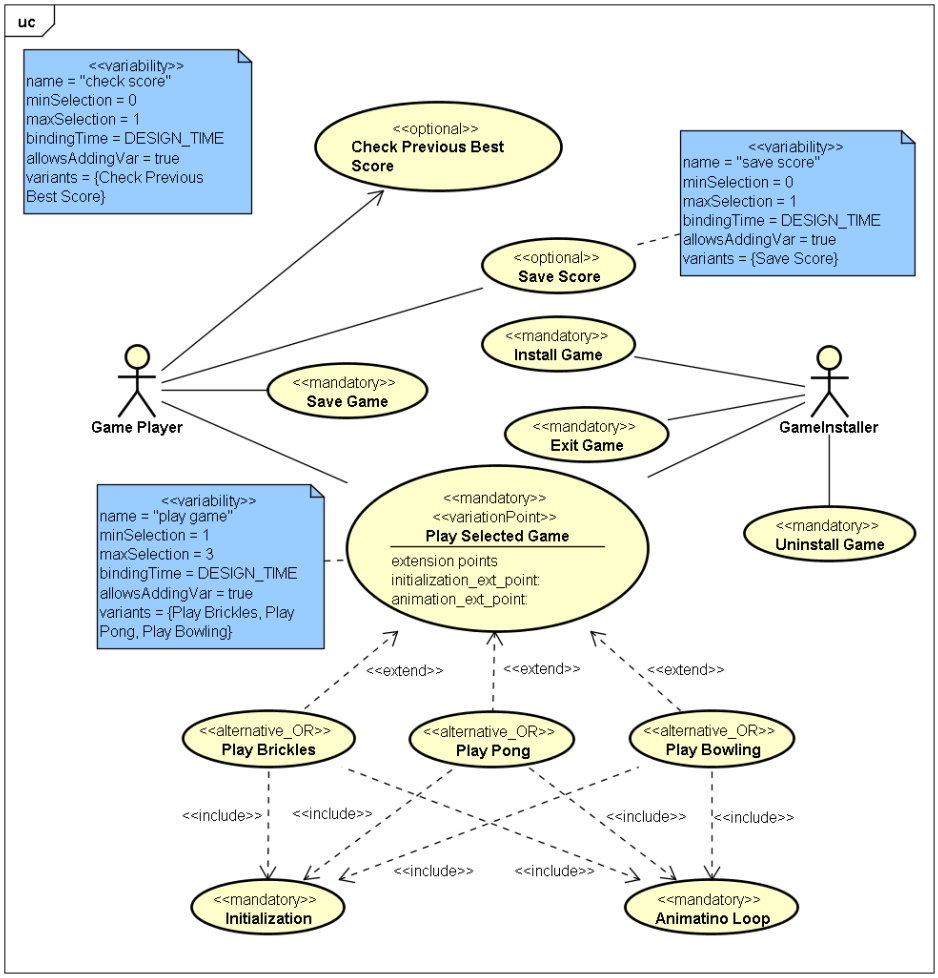


Figura 2.4: Diagrama de casos de uso da LPS AGM contendo as notações *SMarty* (Marcolino et al., 2013)

variationPoint ou ponto de variação e a *variability* ou variabilidade, outros estereótipos são representados, contudo, para entendimento de variabilidade esses dois são essenciais.

O primeiro, ponto de variação, é a resolução de variabilidades em artefatos genéricos de uma LPS, a variabilidade representa os possíveis elementos pelos quais um ponto de variação pode ser resolvido e também pode representar uma maneira de resolver uma variabilidade diretamente.

2.3 Teste de Linha de Produto de Software

O potencial de reuso de artefatos além de aumentar a produtividade torna a LPS atrativa ao mercado. Para alcançar as melhorias pretendidas, a qualidade dos artefatos

reutilizáveis deve ser verificada. Teste de software ainda é a técnica de garantia de qualidade mais comum na indústria, no entanto, a evolução dos sistemas e o aumento da complexidade computacional vêm forçando a criação de processos e atividades mais complexas no que tange ao desenvolvimento de software em larga escala (Apel et al., 2013b).

Teste de software é um processo que faz parte do desenvolvimento de software e tem como principal objetivo revelar defeitos para que sejam corrigidos até que o produto final atinja a qualidade desejada e satisfaça os seus requisitos. Isto porque o custo de reparo de falhas é maior ao final do desenvolvimento do que em estágios iniciais. Assim, antecipando-se os testes, o custo do produto se torna menor e se obtém um ganho em qualidade (Do Carmo Machado et al., 2014).

As terminologias aplicadas ao contexto de teste de software que referem-se aos elementos de investigação podem ser definidas, segundo Delamaro et al. (2017) da seguinte forma:

- **Falha:** É um comportamento inesperado do software. Uma falha pode ter sido causada por diversos erros, mas alguns erros podem nunca causar uma falha.
- **Defeito:** É uma inconsistência no software, algo que foi implementado de maneira incorreta. Ocorre em uma linha de código, como uma instrução errada ou um comando incorreto. O defeito é a causa de um erro, porém, se uma linha de código que contém o defeito nunca for executada, o defeito não irá provocar erro.
- **Erro:** Erro humano produzindo resultado incorreto. O erro evidencia o defeito, ou seja, quando há diferença entre o valor obtido e o valor esperado constitui um erro.

Para que os objetivos da execução dos testes sejam alcançados, existe uma cadeia organizada de atividades, como na engenharia de software, em que um ciclo de vida perfaz todos os pontos necessários para se alcançar o objetivo final (Delamaro et al., 2017). O ciclo de Vida dos Testes é composto de 5 fases segundo Crespo et al. (2004): Planejamento, Preparação, Especificação, Execução e Entrega (Figura - 2.5).

Teste de LPS é um grande desafio por causa do amplo conjunto de produtos que podem ser derivados de um núcleo de artefatos comuns e suas variabilidades (Figura - 2.6). Sendo assim, para uma pesquisa completa a respeito de teste em LPS, primeiro é necessário entender mais sobre o modelo de processo de teste. Um exemplo de ciclo de vida comumente utilizado no apoio ao teste é o modelo V (Figura - 2.7).

O modelo V de desenvolvimento é um modelo conceitual de Engenharia de sistemas/ desenvolvimento de produto, visto como melhoria ao problema de reatividade do modelo

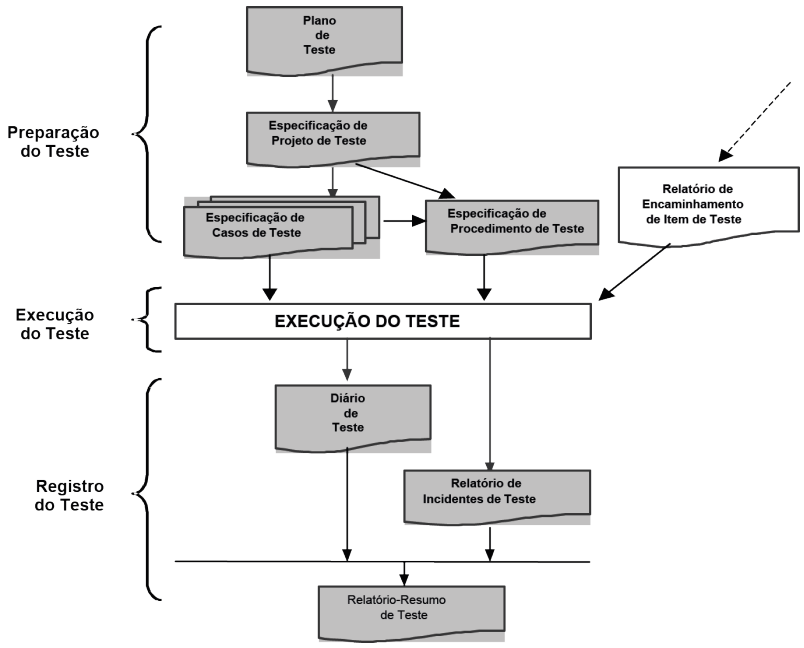


Figura 2.5: Modelo de proposta de implantação do processo de teste de software segundo Crespo et al. (2004)

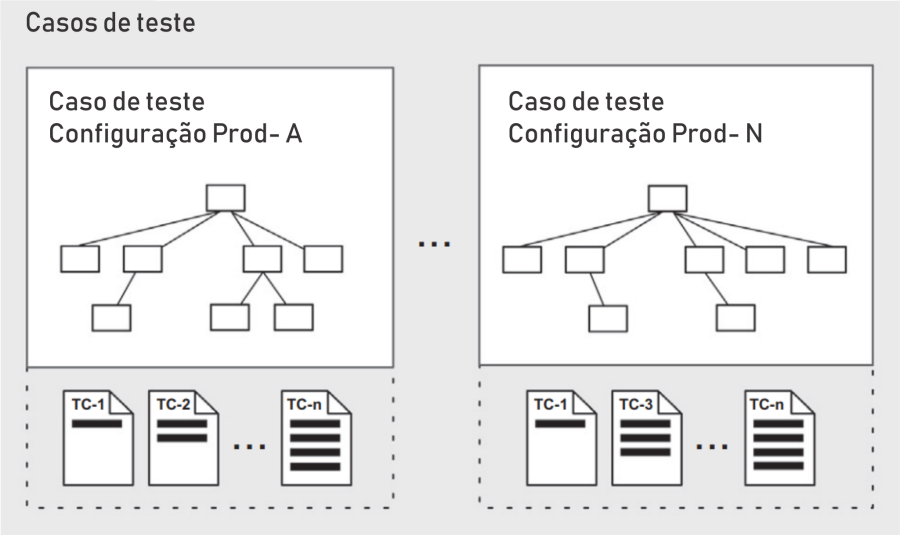


Figura 2.6: Representação de interesse em testes de SPL: seleção de instâncias do produto para testar. Traduzido de Do Carmo Machado et al. (2014)

em cascata. E permite que, durante a integração de um sistema em seus diversos níveis, os testes sejam feitos contra os próprios requisitos do componente/interface que está sendo testado(a), em contraste com modelos anteriores em que o componente era testado contra a especificação do componente/interface.

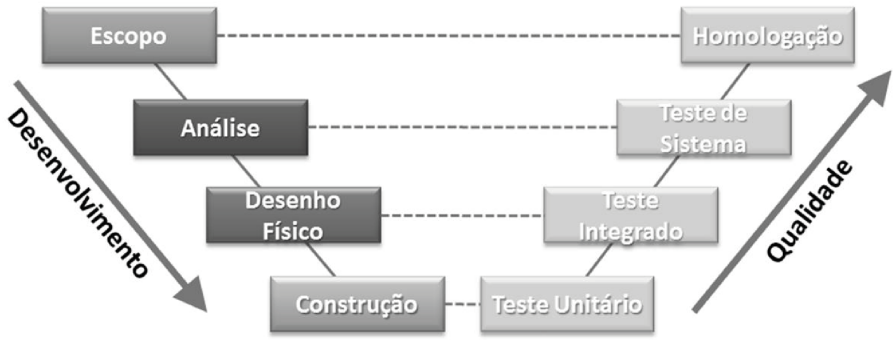


Figura 2.7: Esquema simplificado que representa o modelo V de gerenciamento de atividades de engenharia de sistemas. Traduzido de Do Carmo Machado et al. (2014)

Como mencionado anteriormente, o desafio na geração e execução de teste em LPS, se deve principalmente à geração de produtos derivados do núcleo de artefatos comuns e suas variabilidades (vide Seção 2.2). Isso porque, para cada instanciação de produto na LPS, casos de teste já existentes podem não ser mais aplicados por causa da variação das funcionalidades (Figura - 2.6).

Do Carmo Machado et al. (2014) apresentam algumas abordagens que visam à diminuição dessa lacuna na geração e execução de teste para produtos derivados de LPS. Demonstram também que TBM vem se apresentando como uma alternativa à geração antecipada de casos de teste considerando variações de produtos, isso porque a geração da sequência ou casos de teste se dá durante a criação dos modelos na engenharia de domínio de LPS.

2.4 Teste Baseado em Modelos para LPS

Teste Baseado em Modelo (TBM) tem como característica principal o fato de ser uma abordagem de teste caixa preta, derivada de um modelo que descreve aspectos funcionais do produto testado. Também se refere a um processo de engenharia de software que estuda, constrói, analisa e aplica modelos bem definidos para dar suporte às atividades relacionadas com testes (Shafique e Labiche, 2010).

TBM tem por objetivo verificar se a especificação de um software está de acordo com o seu modelo e foca na geração automática de sequências ou casos de testes. A ideia básica é identificar e construir um modelo abstrato que represente o comportamento do *System Under Test* (SUT). Com esse modelo é possível gerar um grande número de casos de teste ainda na modelagem do produto (Devroey, 2014).

Tais casos de testes, derivados dos modelos, são conhecidos como a suíte abstrata de testes, e seu nível de abstração está intimamente relacionado ao nível de abstração do modelo, segundo Do Carmo Machado et al. (2014). As vantagens da abordagem é que a geração de testes começa mais cedo no ciclo do desenvolvimento e podem ser criados casos de teste automaticamente a partir de um modelo.

Os casos de teste podem ser representados por meio de árvores de decisão, *statecharts*, ontologias de domínio ou diagramas de casos de uso e/ou estados da *Unified Modeling Language* (UML) (Isa et al., 2017).

O processo de TBM (Figura - 2.8) pode ser dividido em cinco passos:

1. Modelar o SUT;
2. Gerar os testes abstratos a partir do modelo;
3. Transformar os testes abstratos em testes executáveis;
4. Executar os testes no SUT;
5. Analisar os resultados.

Do Carmo Machado et al. (2014) apontam que testes devem ser considerados na engenharia de domínio e na de aplicação. Dentro do interesse de teste, dois itens devem ser levados em consideração: o conjunto de requisitos do produto e a qualidade do modelo de variabilidade em teste. Esses autores apresentam também uma grande quantidade de trabalhos que apontam técnicas para lidar com o aspecto de seleção de produto para teste e o teste real dos produtos, utilizando TBM para LPS. No entanto, consideram a falta de relatos reais de experiências industriais que podem limitar algumas conclusões. O estudo ainda apresenta uma série de estratégias que podem apoiar a seleção e a execução dos testes reais em produtos.

Baseado nesse cenário, um dos maiores desafios em teste de LPS se dá em relação às particularidades de cada modelo. Para isso, TBM busca, na criação de modelos da engenharia de domínio, realizar a geração de casos de teste que possam ser reutilizados na engenharia de aplicação. Alguns trabalhos apresentados por Do Carmo Machado et al. (2014) focam na construção da geração antecipada dos testes na modelagem de domínio de uma LPS.

Uma particularidade de TBM é que normalmente faz de uso de Máquinas de Estados Finitos (MEF) (Figura - 2.9), um modelo formal que representa as possíveis configurações de um sistema.

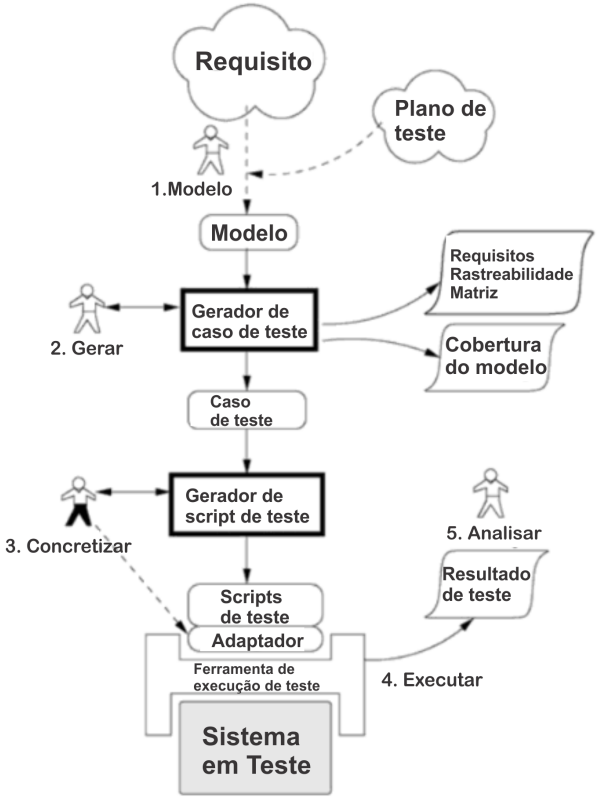


Figura 2.8: Ciclo de Processo de TBM, Traduzido de Utting e Legeard (2010)

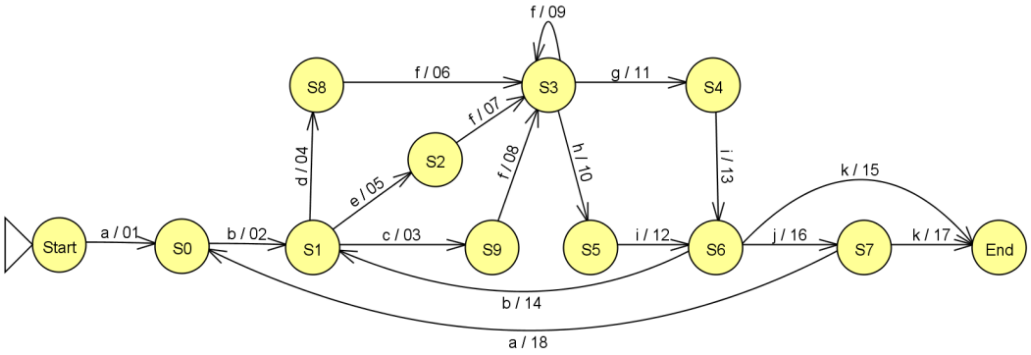


Figura 2.9: Exemplo de grafo de uma MEF (Costa, 2016)

A Figura - 2.9 apresenta um exemplo MEF. Nela observam-se os estados (nós) e as suas transições (arestas). Esse modelo finito e formal permite melhor observação do comportamento das atividades do sistema modelado.

Sabendo que TBM possui particularidades em sua utilização em LPS e que proporciona, de certa forma, suporte à variabilidade, foi realizado um Mapeamento Sistemático

da Literatura (MSL) (Apêncie A), com a finalidade de evidenciar como TBM tem sido utilizado em LPS.

Na Tabela - 2.1 são relacionados os 44 trabalhos analisados no MSL, por meio dos quais pode-se tirar conclusões e suposições para nortear este trabalho com relação:

- aos principais diagramas utilizados;
- à utilização de ferramentas de apoio;
- às abordagens de teste utilizadas; e
- ao gerenciamento de variabilidade.

Em seguida um mapa (Figura - 2.10) foi criado a partir da análise dos trabalhos do MSL. Tal mapa serve como um guia para direcionamento de estudos por temas de interesse. Esse direcionamento pode ser feito por assunto específico ou pelo caminho que um determinado estudo percorreu até o objetivo final, que é a geração de casos de teste.

Tabela 2.1: Trabalhos selecionados para leitura e extração de dados do MSL

ID	Título	Autor(res)	Ano de Publicação	Fonte	Tipo
T1	Delta-Oriented Model-Based SPL Regression Testing	Sascha Lity, Malte Lochau, Ina Schaefer, Ursula Goltz	2012	ACM	Evento
T2	Industrial Evaluation of Pairwise SPL Testing with MoSo-PoLiTe	Michaela Steffens, Sebastian Oster, Malte Lochau, Thomas Fogdal	2012	ACM	Evento
T3	Model-Based Coverage-Driven Test Suíte Generation for Software Product Lines	Harald Cichos, Sebastian Oster, Malte Lochau, Andy Schurr	2011	ACM	Periódico
T4	MoSo-PoLiTe - Tool Support for Pairwise and Model-Based Software Product Line Testing	Sebastian Oster, Ivan Zorcic, Florian Markert, Malte Lochau	2011	ACM	Evento
T5	MPLM - MaTeLo Product Line Manager	Hamza Samih, Ralf Bogusch	2014	ACM	Evento
T6	On the use of test cases in model-based software product line development	Alexander Knapp, Markus Roggenbach, Bernd-Holger Schlingloff	2014	ACM	Evento
T7	Pairwise Feature-Interaction Testing for SPLs: Potentials and Limitations	Sebastian Oster, Malte Lochau, Marius Zink, Mark Grechanik	2011	ACM	Evento
T8	Deriving Usage Model Variants for Model- based Testing: An Industrial Case Study	Hamza Samih, Hélène Le Guen, Ralf Bogusch, Mathieu Acher, Benoit Baudry	2014	IEEE	Evento
T9	Model-based Software Product Line Testing by Coupling Feature Models with Hierarchical Markov Chain Usage Models	Ceren Sahin Gebizli, Hasan Sozer	2016	IEEE	Evento
T10	Model-Based Test Design of Product Lines: Raising Test Design to the Product Line Level	Hartmut Lackner, Martin Thomas, Florian Wartenberg, Stephan Weißleder	2014	IEEE	Periódico
T11	Requirements-Based Delta-Oriented SPL Testing	Michael Dukaczewski, Ina Schaefer, Remo Lachmann, Malte Lochau	2013	IEEE	Evento
T12	Using Feature Model to Support Model- Based Testing of Product Lines: An Industrial Case Study	Shuai Wang, Shaukat Ali, Tao Yue, Marius Liaaen	2013	IEEE	Periódico
T13	An automated Model-based Testing Approach in Software Product Lines Using a Variability Language	Boni García, Rodrigo García-Carmona, Álvaro Navas, Hugo A. Parada-Gélvez, Félix Cuadrado, Juan C. Dueñas	2010	Politécnica Arquivo digital UPM	Evento

ID	Título	Autor(res)	Ano de Publicação	Fonte	Tipo
T14	Automated Product Line Methodologies to Support Model-Based Testing	Shuai Wang, Shaukat Ali, Arnaud Gotlieb	2013	CEUR Event Proceedings	Evento
T15	Behavioural Model Based Testing of Software Product Lines	Xavier Devroey	2014	ACM	Evento
T16	Feature Model-based Software Product Line Testing	Sebastian Oster	2012	TUprints	Periódico
T17	Model-based pairwise testing for feature interaction coverage in software product line engineering	Malte Lochau, Sebastian Oster, Ursula Goltz, Andy Schurr	2011	Springer	Periódico
T18	Model-based Test Generation for Software Product Line	Xinying Cai, Hongwei Zeng	2013	IEEE	Evento
T19	Model-Based Testing for Software Product Lines	Erika Mir Olimpiew	2008	Springer	Evento
T20	PLETS - A Product line of model-based testing tools	Elder de Macedo Rodrigues	2013	PUC-RS	Evento
T21	Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines	Stephan Weißleder, Hartmut Lackner	2013	EPTCS	Evento
T22	A Product Line Modeling and Configuration Methodology to Support Model-Based Testing: An Industrial Case Study	Shaukat Ali, Tao Yue, Lionel Briand, Suneth Walawege	2012	Springer	Periódico
T23	Coverage Criteria for Behavioural Testing of Software Product Lines	Xavier Devroey, Gilles Perrouin, Axel Legay, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans	2014	Springer	Evento
T24	A Model Based Testing Approach for Model-Driven Development and Software Product Lines	Beatriz Pérez Lamancha, Macario Polo Usaola, Mario Piattini Velthius	2010	Springer	Evento
T25	A Vision for Behavioural Model-Driven Validation of Software Product Lines	Xavier Devroey, Maxime Cordy, Gilles Perrouin, Eun-Young Kang, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, Benoit Baudry	2012	Springer	Evento
T26	Abstract Test Case Generation for Behavioural Testing of Software Product Lines	Xavier Devroey, Gilles Perrouin, Pierre-Yves Schobbens	2014	ACM	Evento
T27	Applying Incremental Model Slicing to Product-Line Regression Testing	Sascha Lity, Thomas Morbach, Thomas Th¨ um, Ina Schaefer	2016	Springer	Periódico
T28	Automated Testing of Software-as- a-Service Configurations using a Variability Language	Sachin Patel, Vipul Shah	2015	ACM	Evento

ID	Título	Autor(res)	Ano de Publicação	Fonte	Tipo
T29	Delta-Oriented FSM-Based Testing	Mahsa Varshosaz, Harsh Beohar, Mohammad Reza Mousavi	2015	Springer	Evento
T30	Incremental Model-Based Testing of Delta-oriented Software Product Lines	Malte Lochau, Ina Schaefer, Jochen Kamischke, Sascha Lity	2012	Springer	Periódico
T31	Model Based Testing in Software Product Lines	Pedro Reales, Macario Polo, Danilo Caivano	2011	Springer	Evento
T32	Model-Based Testing	Malte Lochau, Sven Peldszus, Matthias Kowal, Ina Schaefer	2014	Springer	Evento
T33	Parameterized Preorder Relations for Model-Based Testing of Software Product Lines	Malte Lochau, Jochen Kamischke	2012	Springer	Evento
T34	Poster: VIBeS, Transition System Mutation Made Easy	Xavier Devroey, Gilles Perrouin, Pierre-Yves Schobbens, Patrick Heymans	2015	IEEE	Evento
T35	Spinal Test Suites for Software Product Lines	Harsh Beohar, Mohammad Reza Mousavi	2014	EPTCS	Evento
T36	Automated model-based testing using the UML testing profile and QVT	Beatriz Pérez Lamancha, Pedro Reales Mateo, IgnacioRodríguez de Guzmán, Macario Polo Usaola, Mario Piattini Velthius	2009	ACM	Evento
T37	Relating Variability Modeling and Model- Based Testing for Software Product Lines Testing	Hamza Samih	2012	ICTSS	Evento
T38	An Evaluation of Model-Based Testing in Embedded Applications	Stephan Weißleder, Holger Schlingloff	2014	IEEE	Evento
T39	Assessing Software Product Line Testing Via Model-Based Mutation An Application to Similarity Testing	Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Yves Le Traon	2013	IEEE	Evento
T40	Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines	Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, Yves le Traon Lassy	2010	IEEE	Evento
T41	Model-based Testing of System Requirements using UML Use Case Models	Bill Hasling, Helmut Goetz, Klaus Beetz	2008	IEEE	Evento
T42	Successive refinement of models for model-based testing to increase system test effectiveness	Ceren Sahin Gebizli, Hasan Sozer, Ali Ozer Ercan	2016	IEEE	Evento
T43	A Software Product Line for Model-Based Testing Tools	Elder M. Rodrigues, Avelino F. Zorzo, Itana M. Gimenez, Elisa Y. Nakagawa, Flavio M. Oliveira, José C. Maldonado	2012	PUC-RS	Outro

ID	Título	Autor(res)	Ano de Publicação	Fonte	Tipo
T44	Reusing State Machines for Automatic Test Generation in Product Lines	Stephan Weißleder, Dehla Sokenou, Bernd-Holger Schlingloff	2008	MoTip	Evento

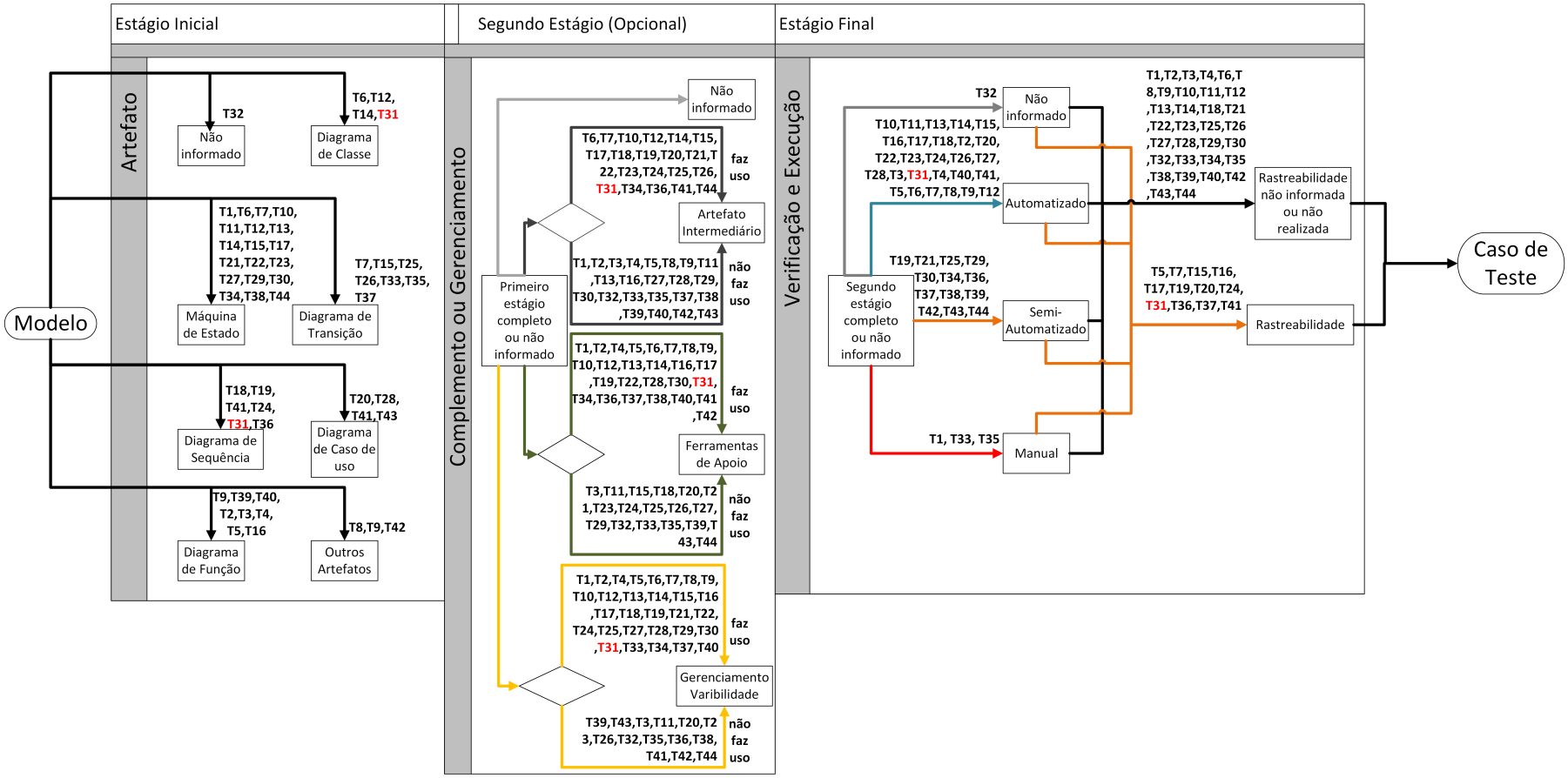


Figura 2.10: Mapeamento dos trabalhos relacionados ao TBM em LPS

2.5 A Abordagem SPLiT-MBt

Costa (2016) desenvolveu uma abordagem denominada *Software Product Line Testing Method Based on System Models* (SPLiT-MBt) em que é possível gerar casos de teste e *scripts* para testar os produtos derivados de uma LPS, com base no reuso inerente à LPS.

O método SPLiT-MBt suporta a geração automática de casos de teste funcionais a partir de diagramas de atividades da UML, mas pode ser expandido para outros modelos. A ideia é gerar artefatos de teste durante a engenharia de domínio e reutilizá-los durante a engenharia de aplicação. A Figura - 2.11 apresenta as etapas da abordagem SPLiT-MBt.

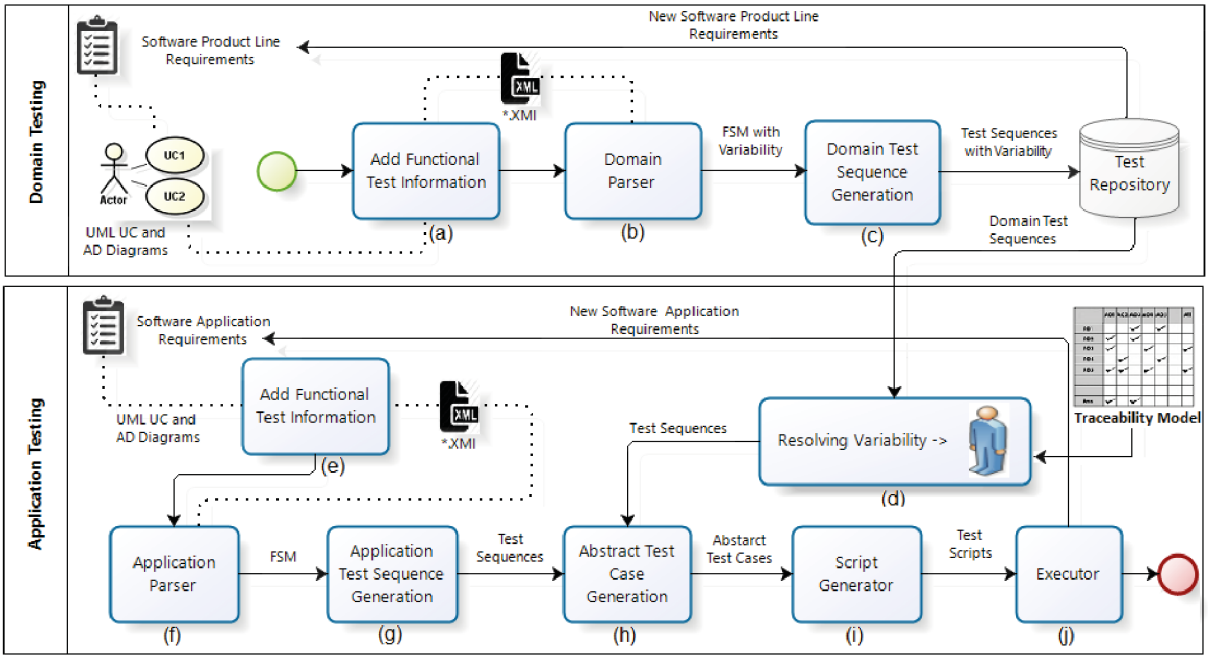


Figura 2.11: A abordagem SPLiT-MBt por Costa (2016)

A abordagem é aplicada em duas etapas: primeiro, adicionam-se informações de teste nos modelos UML que foram projetados anteriormente, usando uma abordagem de gerenciamento de variabilidade para gerar sequências de teste durante a engenharia de domínio, no caso SPLiT-MBt faz uso da abordagem *SMarty*. Segundo, adicionam-se informações de teste em modelos UML (durante a Engenharia de Aplicação) e resolve-se a variabilidade presente nas sequências de teste.

O principal objetivo da abordagem é prover o reuso de artefatos de teste, baseado na adaptação do uso de TBM para se obter a geração automática de casos de teste funcional e *scripts* para modelos, considerando a variabilidade das instâncias da LPS.

Na primeira etapa é realizada a anotação das informações de variabilidade e teste em modelos de sistema, aqui exemplificado por um diagrama de atividades na Figura - 2.12. Em seguida, tais anotações são utilizadas para gerar sequências de teste fazendo uso de métodos de geração, no caso o *Harmonized State Identifiers* (HSI) (Seção 2.5.1). O modelo formal utilizado é a Máquina de Estados Finitos estendida para lidar com as informações sobre variabilidade, pontos de variação e variante.

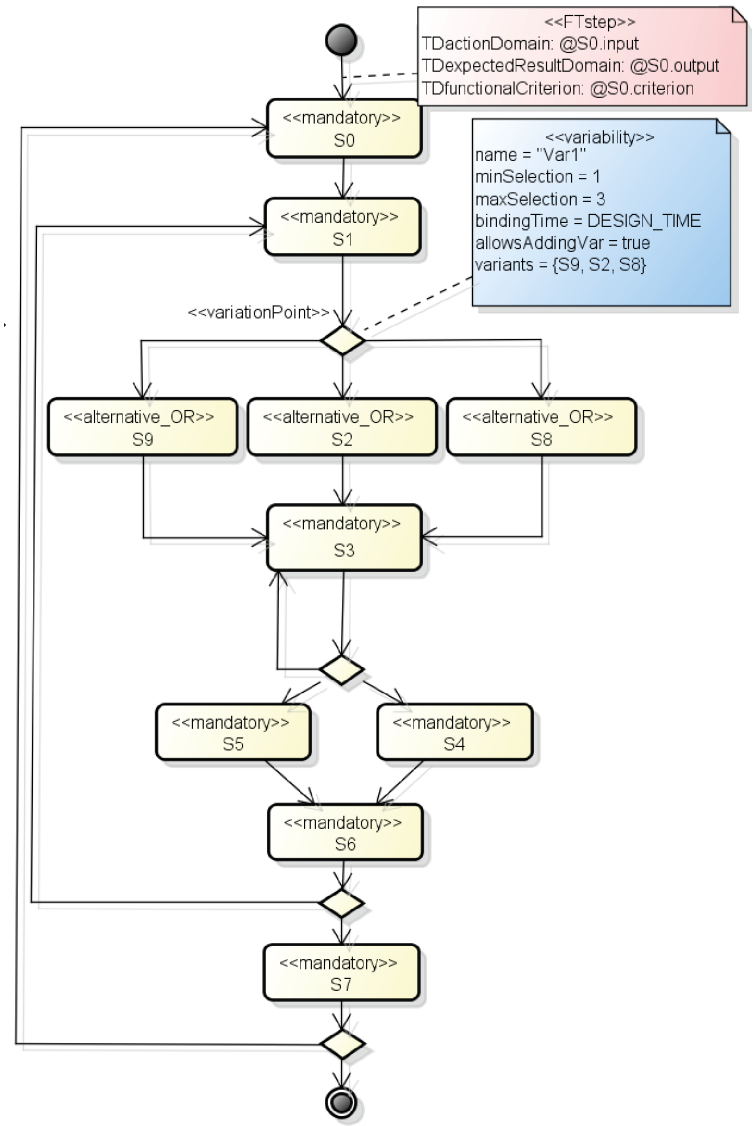


Figura 2.12: Diagrama de atividades *Game Menu* da LPS AGM (Costa, 2016)

Na engenharia de domínio, um DA é convertido em uma MEF que é estendida para lidar com informações sobre variabilidade, exemplificado pela Figura - 2.13. Em seguida, se faz a utilização de um modelo de geração de sequência de teste que foi codificado, denominada *Harmonized State Identifiers* (HSI), a qual realiza a geração das sequências

de teste que são armazenadas em um repositório para utilização/reutilização, porém, ainda sem a resolução das variabilidades.

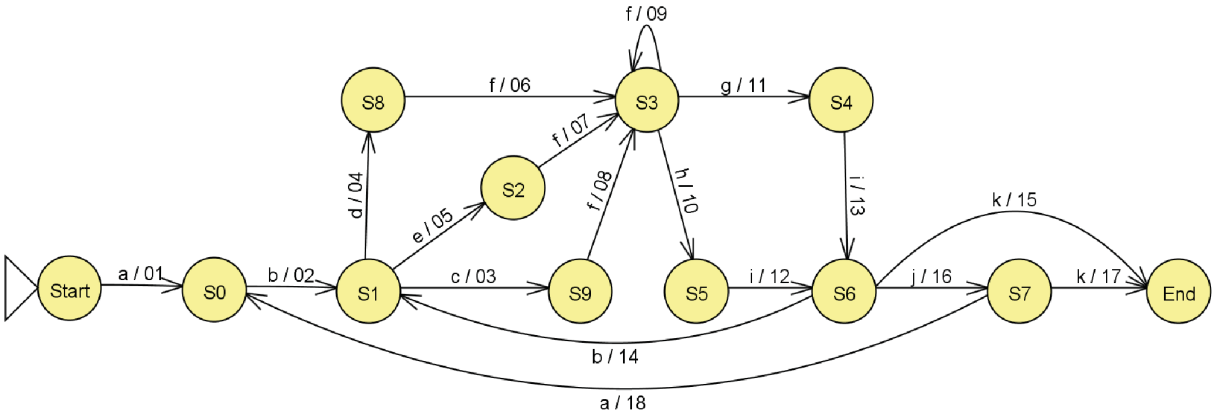


Figura 2.13: Representação da MEF gerada a partir do DA da Figura - 2.12 (Costa, 2016)

Tabela 2.2: Sequência de teste gerada a partir da Figura - 2.13

ID	Conjunto de sequência de teste
Sequência1	ab{d;e;c} VP_or fff,
Sequência2	ab{d;e;c} VP_or fgib,
Sequência3	ab{d;e;c} VP_or fgik,
Sequência4	ab{d;e;c} VP_or fhib,
Sequência5	ab{d;e;c} VP_or fhik,
Sequência6	ab{d;e;c} VP_or fgijk,
Sequência7	ab{d;e;c} VP_or fgijab

2.5.1 Métodos de Geração de Casos de Teste

Métodos de geração de casos de teste têm por objetivo verificar se uma implementação está correta em relação à especificação, por meio da execução de atividades de teste e validação em sistemas descritos por modelos (Fujiwara et al., 1991), para isso, devem ser geradas sequências de testes.

Apesar dos métodos definirem procedimentos para a geração de testes, a principal diferença que os evidenciam é o custo da geração dessas sequências e a capacidade de detecção de defeitos (efetividade). Dessa forma, deve-se levar em conta a relação custo-benefício de cada método (Pinheiro, 2012).

O foco principal consiste em promover a detecção do maior número possível de defeitos existentes em uma implementação, levando em conta o tamanho do conjunto gerado, para que não inviabilize a sua aplicação prática.

Os métodos de geração são fortemente baseados em sequências básicas, assim, são apresentados de forma simplificada os métodos de geração de sequências de teste mais utilizados na academia:

- *State Cover* (Q);
- *Transition Cover* (P);
- Sequência de Separação (SS);
- Sequência de Distinção (DS);
- Sequência Única de Entrada e Saída (UIO);
- Conjunto de Caracterização (W);
- Conjunto de Identificação (Wp); e
- Conjunto de Identificadores Harmonizados (Hi).

Considerando essas sequências básicas, os métodos de geração utilizam em sua base uma ou várias dessas sequências. A seguir são discutidos, de forma objetiva, os métodos utilizados para a geração dos casos de teste.

Método TT

O método *Transition Tour* (TT), proposto por Naito e Tsunoyama (1981), consiste em construir uma sequência que percorra pelo menos uma vez todas as transições de uma MEF. Sidhu e Leung (1989) definem uma *Transition Tour* como sendo uma sequência de teste que pode ser gerada simplesmente aplicando entradas aleatórias em uma MEF, a partir do estado inicial, até que todas as transições tenham sido cobertas, finalizando no estado inicial. Porém, como a geração é aleatória, muitas sequências redundantes podem ser geradas. Assim, métodos de redução devem ser aplicados para eliminar as redundâncias presentes na sequência final.

Método W

Um dos métodos mais difundidos para geração de sequências de testes, o método *Auto-mata Theoretic* ou, como ficou mais conhecido, Método W foi proposto por Chow (1978). O nome atribuído ao método originou-se por causa da referência feita por Chow (1978) ao conjunto de caracterização, utilizado pelo método, como conjunto W. O método W pode ser considerado um método clássico e precursor da área, uma vez que a maioria dos trabalhos seguintes foram baseados no método W. Uma restrição quanto ao método é em relação à sua aplicabilidade, que exige que as MEFs sejam: determinísticas, completas, inicialmente conexas e minimais.

Método WP

O método W parcial (Wp), do inglês *partial W*, foi proposto por Fujiwara et al. (1991) como uma melhoria do método W. A partir do conjunto W é criado o conjunto de identificação Wi, que extrai um subconjunto do conjunto W capaz de identificar cada estado da MEF, dependendo do estado final *si* que foi alcançado pela sequência.

Método DS

O método DS é baseado na sequência de distinção DS e foi proposto inicialmente por Hennine (1964) com a utilização de sequências SS para gerar uma sequência de verificação em conjunto com a DS. Gonenc (1970) apresenta a opção de se utilizar grafos para a geração das sequências.

Os demais trabalhos foram desenvolvidos com o intuito de diminuir o tamanho da sequência de verificação, utilizando a modelagem baseada em grafos proposta por Gonenc (1970). A aplicabilidade do método é condicionada a existência da sequência DS, uma vez que nem todas as MEFs a possuem. Além disso, o método DS só é aplicável em MEFs: determinísticas, completas, fortemente conexas e minimais.

Método UIO

O método UIO foi originalmente proposto por Sabnani e Dahbura (1988) como um método completo. Porém, Vuong (1989) apresentou um contra-exemplo que demostrou que o método UIO não garante a cobertura completa de defeitos para todas as MEFs. O método é baseado em sequências UIO e é realizado em apenas uma fase, correspondente à fase dois do método W. Da mesma forma que o método W, a aplicabilidade do método UIO está

restrita à existência das sequências UIO e à MEFs que sejam: determinísticas, inicialmente conexas, completas e minimais.

Método UIOv

Como uma variação do método UIO, o método UIOv (*UIO variation*) foi proposto por Vuong (1989). A solução apresentada inclui uma fase a mais no processo de geração de testes, em que seriam aplicadas todas as sequências UIO a todos os estados pelo menos uma vez, ou seja, a sequência UIOi seria aplicada em cada estado *sj* atingido a partir do conjunto *Q*, tal que *i=0..n* e *j=0..n*.

Método HSI

O método HSI (Petrenko, 1994), assim como a maioria dos métodos propostos, foi proposto como uma melhoria para o método *W*. Além de gerar conjuntos de testes completos, o método apresenta um grau de aplicabilidade maior que o das demais métodos, pois pode ser aplicado em MEFs completas e em MEFs parciais. Dessa forma, o método consegue cobrir um número maior de especificações¸ comparado com o método *W*.

Fazendo um paralelo entre os métodos *Wp* e HSI, ambos têm sequências de separação por estado que possuem o objetivo de distinguir o estado *si* dos demais. Dessa forma, a diferença entre os conjuntos *Wi* e *Hi* está na maneira como são construídos. Enquanto *Wi* é obtido a partir do conjunto *W*, o *Hi* é construído a partir de sequências de separação que distinguem cada par de estados da MEF.

Durante a pesquisa sobre as ferramentas de geração de casos de teste mencionadas na Seção 3.2.1 foram encontrados dois trabalhos que possuíam características compatíveis com os requisitos necessários para a utilização na abordagem proposta nesta dissertação, a ferramenta JPlavisFSM (Pinheiro e Simão, 2012) e a SPLiT-MBt (Costa, 2016), ambas realizam o processo de geração de casos de teste considerando MEF. A partir desse cenário, procurou-se avaliar tais ferramentas para verificar a viabilidade de utilização.

2.5.2 Ferramenta de Geração de Sequências de Teste JPlavisFSM

A ferramenta JPlavisFSM (Pinheiro e Simão, 2012) é uma versão atualizada da antiga plataforma PLAVIS, resultado do projeto "PLAVIS - Plataforma para Validação e Integração de Software em Sistemas Espaciais"(Da Silva Simao et al., 2008), financiado pelo Conselho Nacional de Pesquisa (CNPq) e desenvolvido entre os anos de 2002 e 2004. A primeira versão da plataforma integrava outras ferramentas de geração e uma

implementação do algoritmo UIO, responsáveis pela geração de casos de teste a partir de MFEs. A referida ferramenta foi integrada a ferramenta ProteumFSM que possuía a capacidade de criar mutantes de especificação em MEFs. A JPlavisFSM faz algo similar que a SPLiT-MBt, porém conforme será apresentado, não atende a todos os requisitos essenciais da abordagem *SMartyTesting*.

A ferramenta permite a criação de grafos que representam MEFs (Figura - 2.14), que posteriormente possam ser utilizados para a criação de sessões de teste. Na geração de sessão de teste a ferramenta dispõe de alguns métodos de geração de sequência (Figura - 2.15) o que a torna uma ferramenta com grande potencial de recursos.

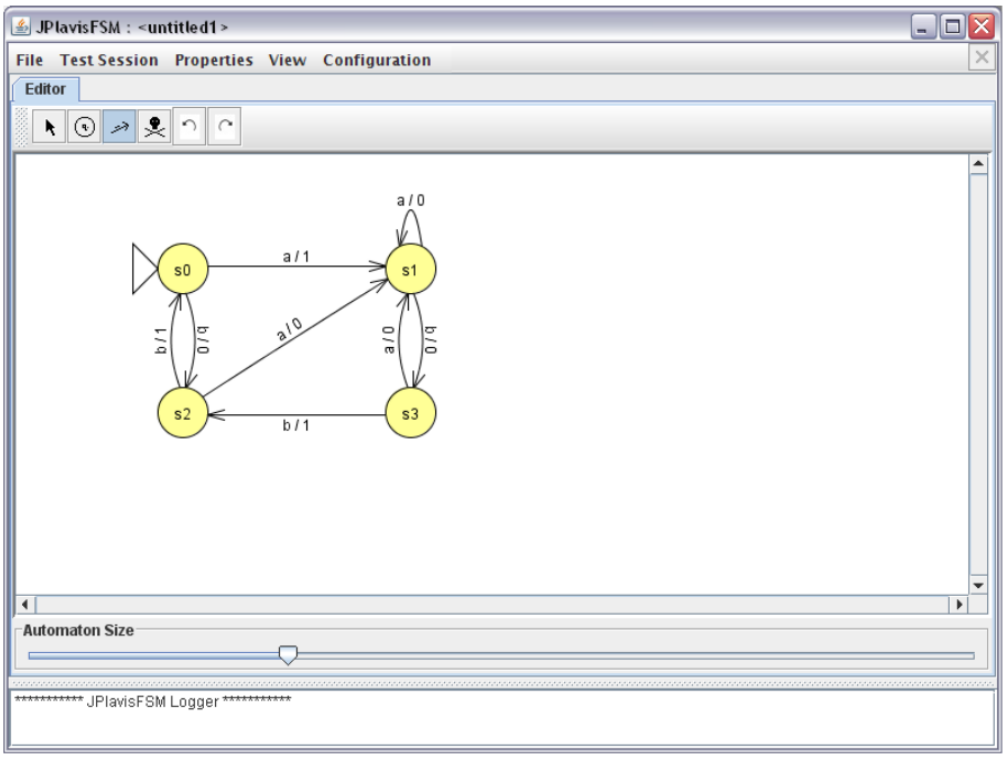


Figura 2.14: representação de uma MEF utilizando a ferramenta JPlavisFSM (Pinheiro e Simão, 2012)

Após comparações entre as duas ferramentas (Tabela - 2.3) optou-se por utilizar a SPLiT-MBt como ferramenta de apoio na segunda etapa da abordagem *SMartyTesting*, para apoiar a geração de casos de teste com base nos artefatos de entrada provenientes da primeira etapa.

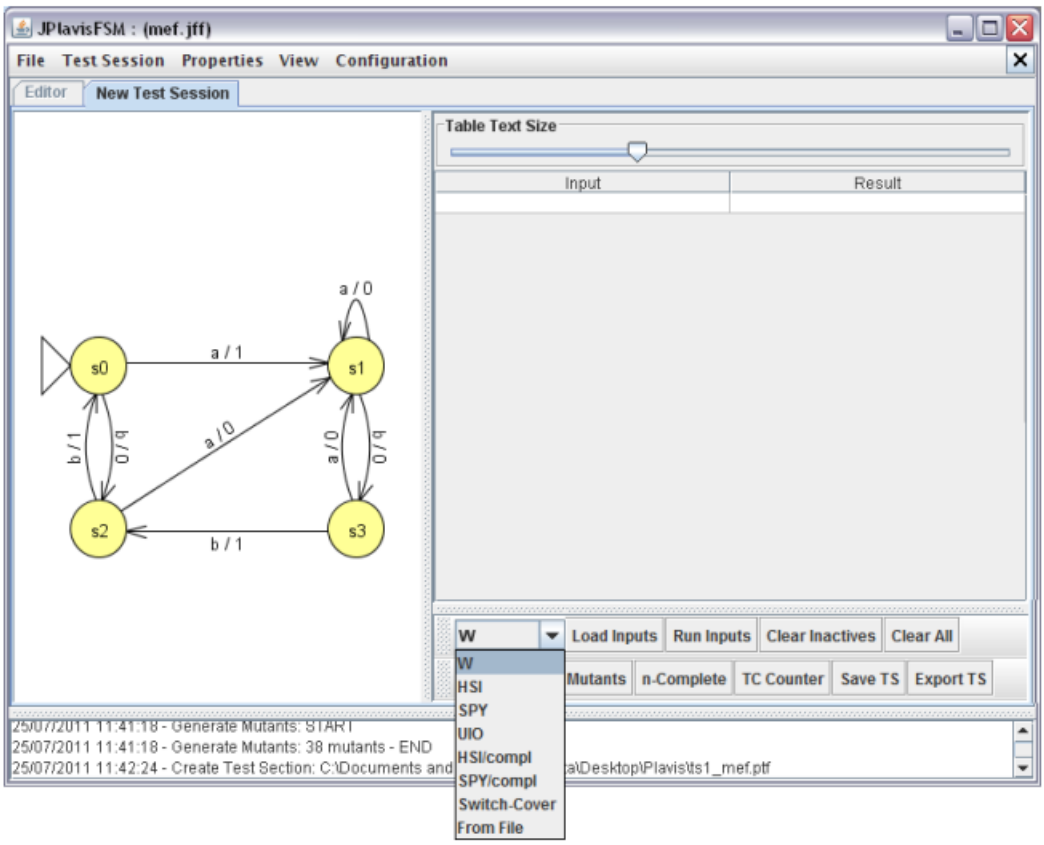


Figura 2.15: Geração de sequências de teste pela ferramenta JPlavisFSM (Pinheiro e Simão, 2012)

Tabela 2.3: Comparação de requisitos SPLiT-MBt e JPlavisFSM

Requisito	SPLiT-MBt	JPlavisFSM
Suporte à variabilidade	Sim	Não
Compatível com LPS	Sim	Sim
Formato de artefato de entrada compatível com o requisito	Sim	Não
Suporte a método de geração de sequência de teste	Sim	Sim
Compatível com SMarty	Sim	Não

2.6 Diagramas de Sequência e de Atividades

A UML (*Unified Modeling Language*) é uma linguagem-padrão para a elaboração da estrutura de projetos de software. Pode ser empregada para a visualização, a especificação,

a construção e a documentação de artefatos que façam uso de sistemas complexos de software (Booch et al., 2006).

Na linguagem UML faz-se uso de diagramas para a modelagem de sistemas, onde, para cada nível de interpretação e criação dispõe um diagrama com finalidade diferente. Quando se fala em modelagem de aspectos dinâmicos de sistemas, é possível mencionar os diagramas de atividades e sequência, onde a atividade mostrando o fluxo de controle de uma atividade para outra, apresentando a concorrência bem como, as ramificações de controle, Figura - 2.16.

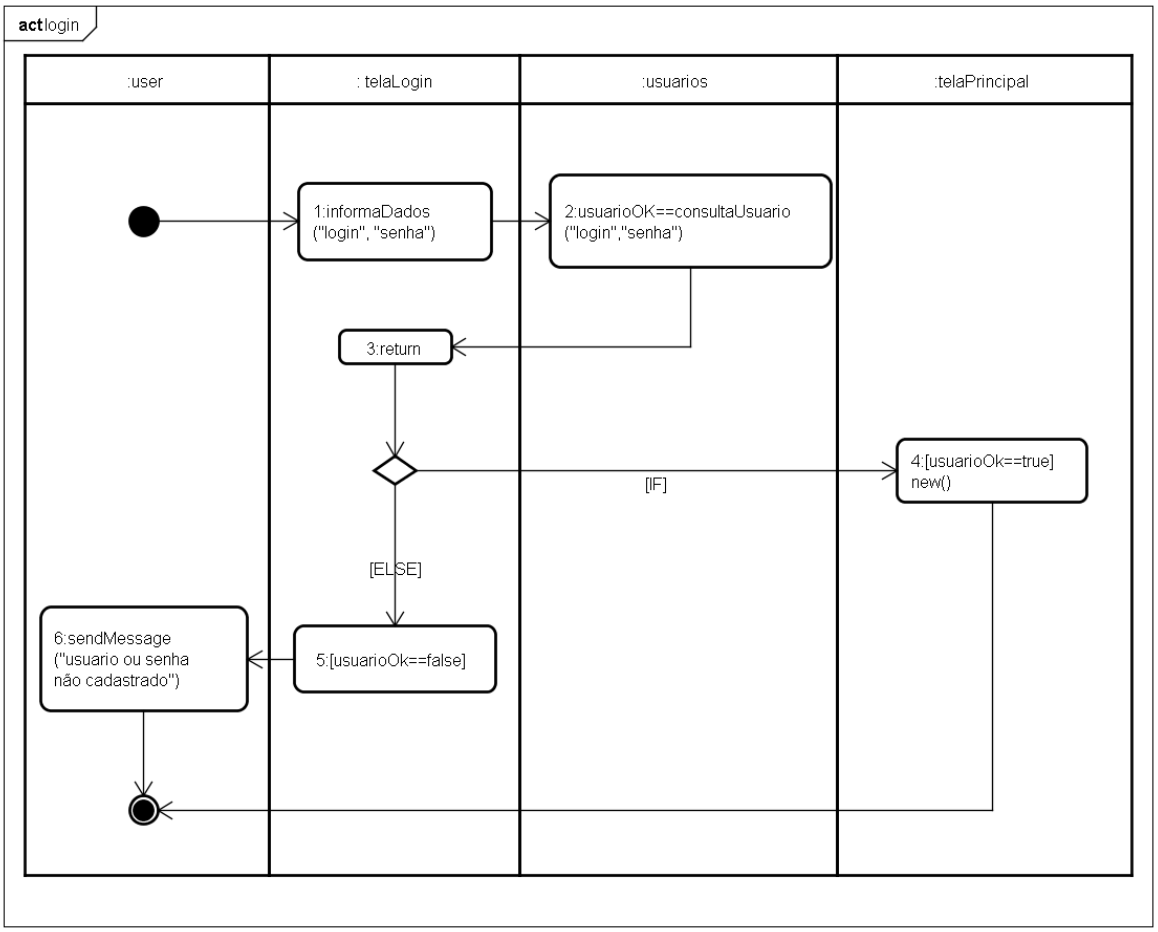


Figura 2.16: Exemplo de um diagrama de atividades

O diagrama de sequências é um diagrama de interação, formado por um conjunto de objetos e seus relacionamentos, incluindo mensagens que poderão ser enviadas entre eles. O DS dá ênfase à ordenação temporal das mensagens e a estrutura dos objetos. (Figura - 2.17)

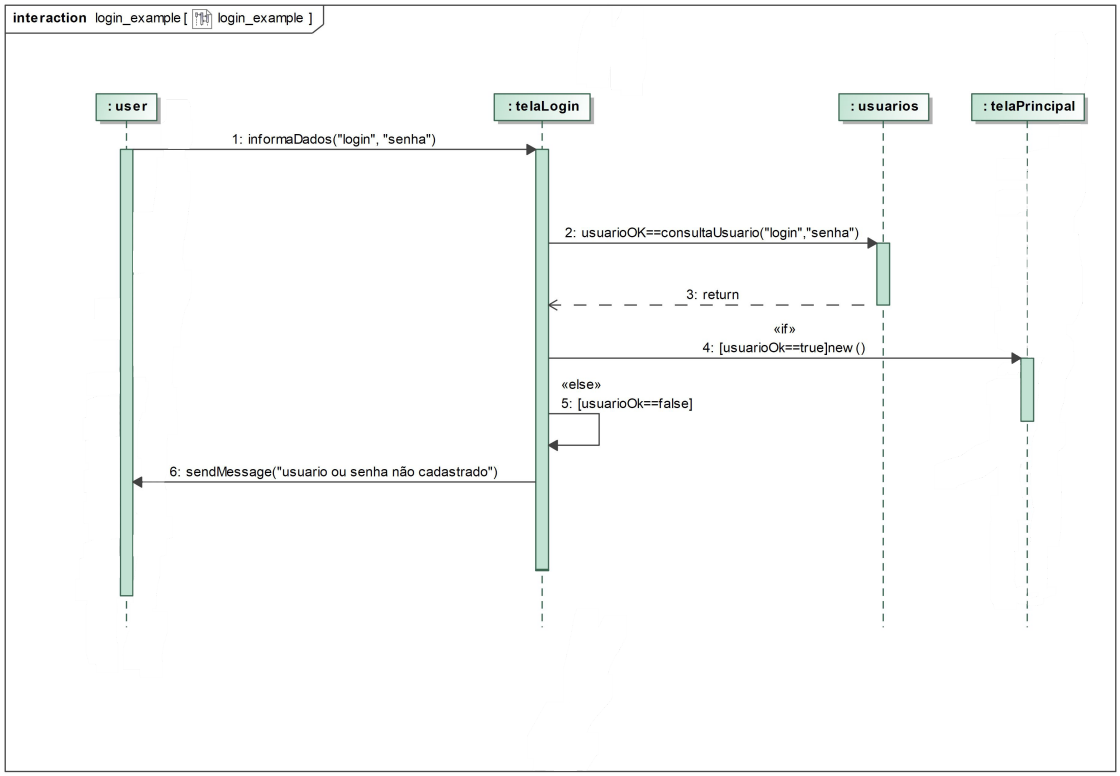


Figura 2.17: Exemplo de um diagrama de sequência

Por isso, pode se concluir que diagramas de atividades possuem um nível maior de abstração se comparados ao diagrama de sequência, que fica mais próximo à classe do sistema, demonstrando comportamentos de objetos, enquanto que o de atividades, demonstra os fluxos e direções que cada atividade do sistema seguirá.

Essa seção apresenta a aplicação para cada diagrama aqui mencionado, dado que ambos fazem parte deste trabalho de dissertação, *SMartyTesting* fazendo uso de diagramas de sequência em comparação à SPLiT-MBt, que faz uso de diagramas de atividades.

2.7 Considerações Finais

Neste tópico foram apresentados todos os elementos que são considerados importantes para fundamentar esta dissertação, como os conceitos de LPS e a abordagem *SMarty*, assim como, os conceitos e aplicação de TBM para LPS e, por último, a abordagem SPLiT-MBt.

3

A Abordagem SMartyTesting

3.1 Considerações Iniciais

Neste tópico é apresentada uma abordagem que auxilia na geração de sequências de teste na Engenharia de Domínio de LPS a partir de modelos *SMarty*. Tais sequências de teste podem ser utilizadas para a geração de casos de teste que, em um segundo momento, podem ser usados na Engenharia de Aplicação. Denominada *SMartyTesting*, a abordagem auxilia na geração de sequências de teste a partir de diagramas de sequência modelados com base em casos de uso e seus fluxos básicos e alternativos, em que o diagrama é convertido para um segundo artefato, diagrama de atividades, que é o requisito de entrada da abordagem SPLiT-MBt. Assim, é realizada a geração de sequências de teste contendo variabilidade. Dessa forma, as sequências de teste podem ser reutilizadas para cada produto a ser testado, aproveitando os benefícios de SPLiT-MBt.

3.2 Caracterização da Abordagem *SMartyTesting*

Nesta seção são abordados os elementos que caracterizam a abordagem *SMartyTesting* como modelos, processos e ferramentas.

3.2.1 Definição do Processo Adotado pela Abordagem *SMartyTesting*

Na Figura - 2.10 (Tópico 2) pode-se observar a quantidade de possibilidades e caminhos que podem ser direcionados aos objetivos desejados por este trabalho, partindo de um modelo inicial para sequência de teste/caso de teste.

Para a abordagem *SMartyTesting*, com base na hipótese estabelecida no tópico 1, foi definido o seguinte processo de teste, de acordo com a Figura - 3.1.

Em um estágio inicial partindo de um diagrama de casos de uso e um diagrama de sequência já modelados por *SMarty*, segue para um segundo estágio, que no caso, para *SMartyTesting* é um estágio obrigatório. No segundo estágio é realizada uma conversão para um artefato intermediário, nesse caso, um diagrama de atividades.

O gerenciamento da variabilidade fica a cargo de *SMarty*, e a ferramenta de apoio para a geração de sequências é a SPLiT-MBt, assim, indo para o terceiro estágio que é o processo de automatização e geração da sequência, onde é utilizada a SPLiT-MBt.

3.2.2 Modelos Utilizados

A abordagem *SMartyTesting* utiliza os seguintes modelos no apoio ao teste de LPS:

- **Diagramas de Casos de Uso:** representam a modelagem dos requisitos de uma LPS contendo variabilidade na representação gráfica e consideram os fluxos básicos e alternativos como fonte de informação para a modelagem de diagramas de sequência;
- **Diagramas de Sequência:** são usados pela abordagem por permitir a representação de fluxos e procedimentos em um baixo nível de abstração, muito próximo ao código-fonte e com mais detalhes sobre as variabilidades;
- **Diagramas de Atividades:** usados como artefatos de entrada para a conversão em MEF, porém com maior nível de abstração que um diagrama de sequência.

A Figura - 3.2 ilustra o processo com destaque aos diagramas usados pela abordagem.

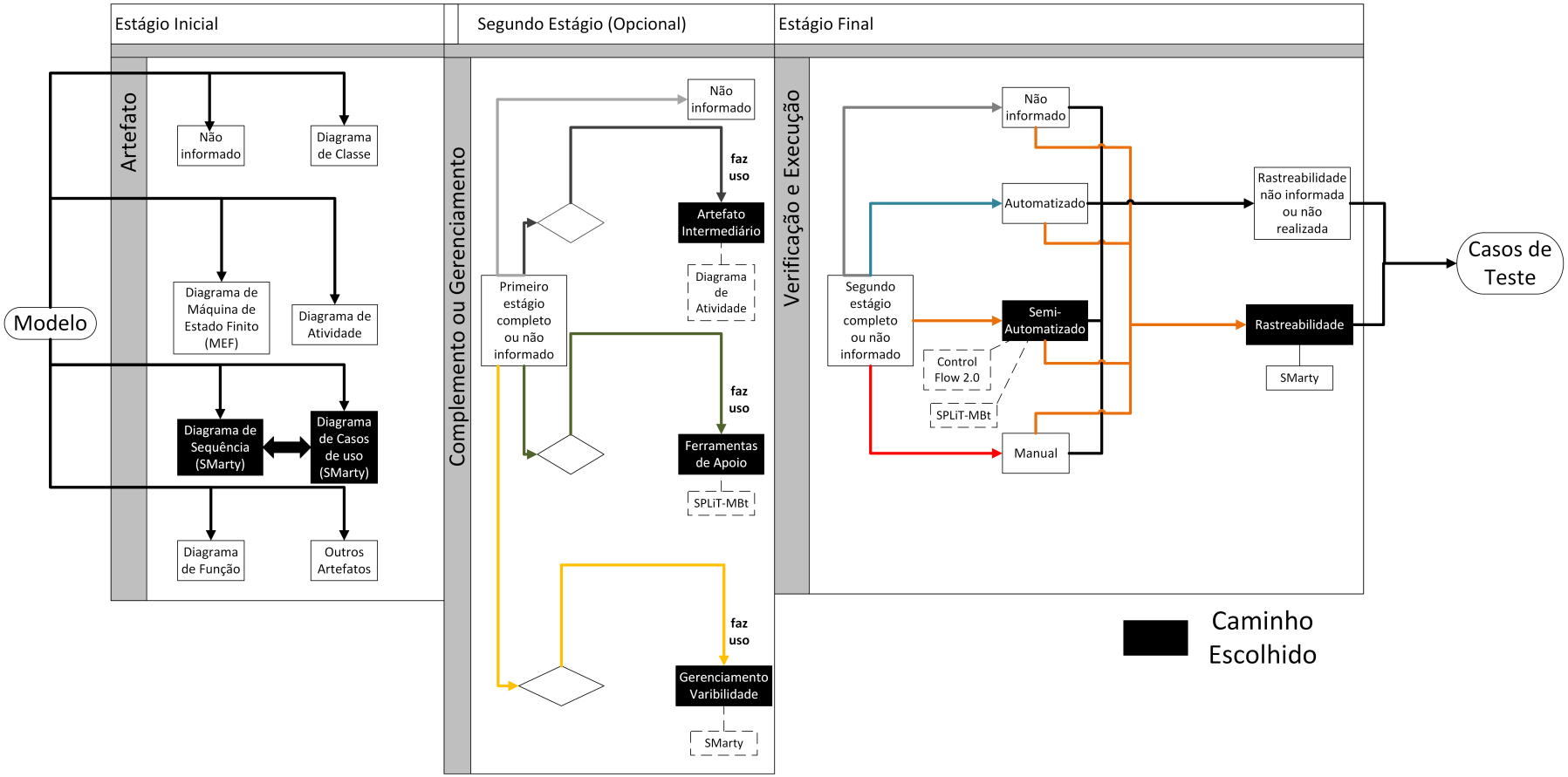


Figura 3.1: Processo de geração de sequência de teste com *SMartyTesting* (instância da Figura - 2.10)

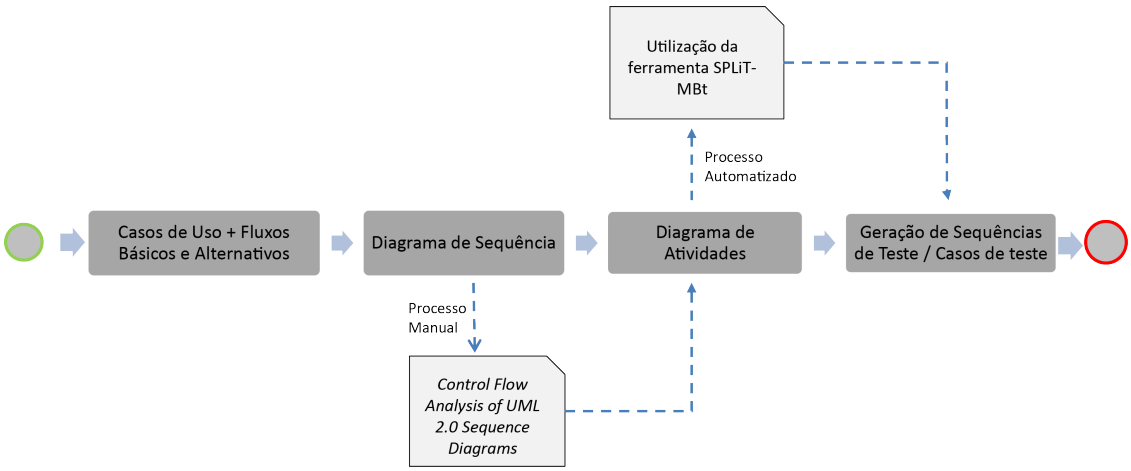


Figura 3.2: Passos para a geração de sequências de teste usando *SMartyTesting*

3.2.3 Conversão de Diagramas de Sequência para Diagramas de Atividades

A conversão de diagramas de sequência para diagramas de atividades foi uma opção restritiva pelo fato de a ferramenta SPLiT-MBt ter como entrada diagramas de atividades. Embora também seja um diagrama de comunicação, o diagrama de atividades demonstra mais o fluxo de controle de uma atividade para outra, assim como a concorrência das atividades. Pode-se dizer que, enquanto o diagrama de sequência está mais próximo de métodos e do código-fonte, o diagrama de atividades está mais próximo dos casos de uso em termos de abstração.

Os dois diagramas são equivalentes quando se trata de elementos de representação, um exemplo é que em um diagrama de sequência, uma condicional de decisão pode ser representada por um diagrama de atividades, assim como quando um método, em um diagrama de sequência envia uma mensagem concorrente, criando uma bifurcação (*fork*).

Baseado no estudo de Garousi et al. (2005), outro fator para a utilização de conversão de sequência para atividade é que não se corre o risco de perder propriedades no mapeamento, mantendo as características de variabilidades que foram estendidas no diagrama de sequência. Porém, é necessária uma validação desse mapeamento, o que foi feito por Swain et al. (2010), sendo portanto, uma oportunidade para validação também com *SMartyTesting*.

Por esses fatores é que se optou pela conversão de diagramas de sequência para atividades, aproveitando a validação da abordagem em uma ferramenta já desenvolvida SPLiT-MBt, que tem suporte à variabilidade, enquadrando-se no escopo deste trabalho.

A proposta de Garousi et al. (2005) consiste de uma metodologia de análise de fluxo de controle baseada em diagramas de sequência (DS) da UML 2.0. Essa técnica pode ser usada durante o ciclo de desenvolvimento e em outras abordagens de teste que façam compreensão e execução de modelos. Entre muitas aplicações, essa técnica pode ser usada em sistemas baseados em DS.

Com base nos diagramas de atividades bem definidos, a proposta *Control Flow Analysis of UML 2.0 Sequence Diagrams* (CCFG) (Garousi et al., 2005) apresenta um metamodelo do diagrama de atividades estendido (Figura - 3.3), para apoiar a análise de fluxo de controle dos diagramas de sequência. Assim, pode-se definir um mapeamento baseado em *Object Constraint Language* (OCL) (Warmer e Kleppe, 2003) que é uma linguagem declarativa para descrever as regras que se aplicam aos modelos UML.

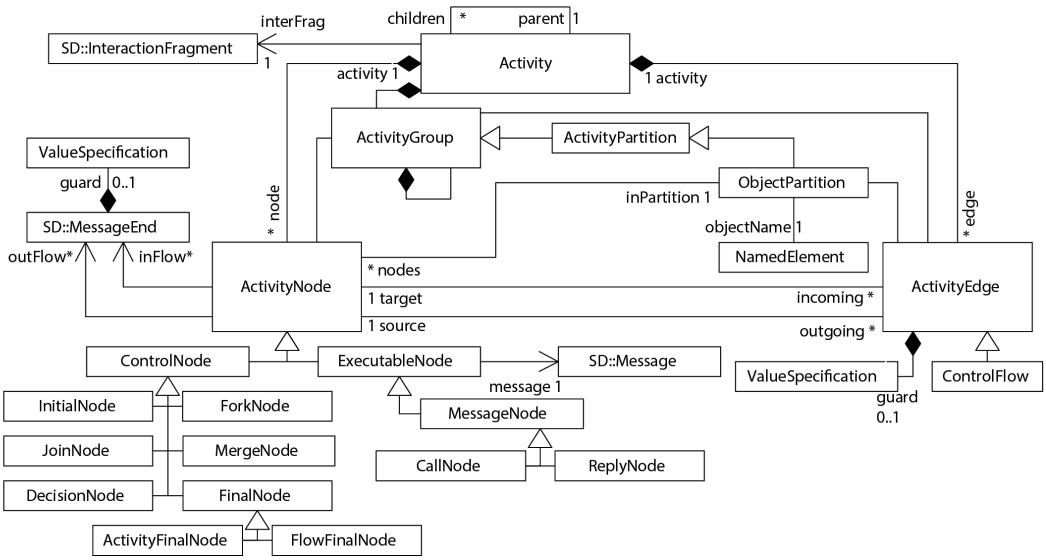


Figura 3.3: Control Flow Analysis of UML 2.0 Sequence Diagrams metamodel (Diagramas de Atividades estendido) (Garousi et al., 2005)

A aplicação de OCL é realizada formalmente e verificável com regras de consistência entre um DS e um CCFG (diagramas de atividades estendido), em que CCFG possui todas as classes e associações necessárias, assim como suporte aos caminhos de fluxo de controle simultâneos (concorrência), que são uma generalização do conceito convencional de caminho de fluxo de controle (Garousi et al., 2005).

O mapeamento consiste da utilização de um metamodelo DS (Figura - 3.4) e um conjunto de regras que deve ser utilizado na conversão, em que o metamodelo CCFG (Figura - 3.3) é considerado como validador.

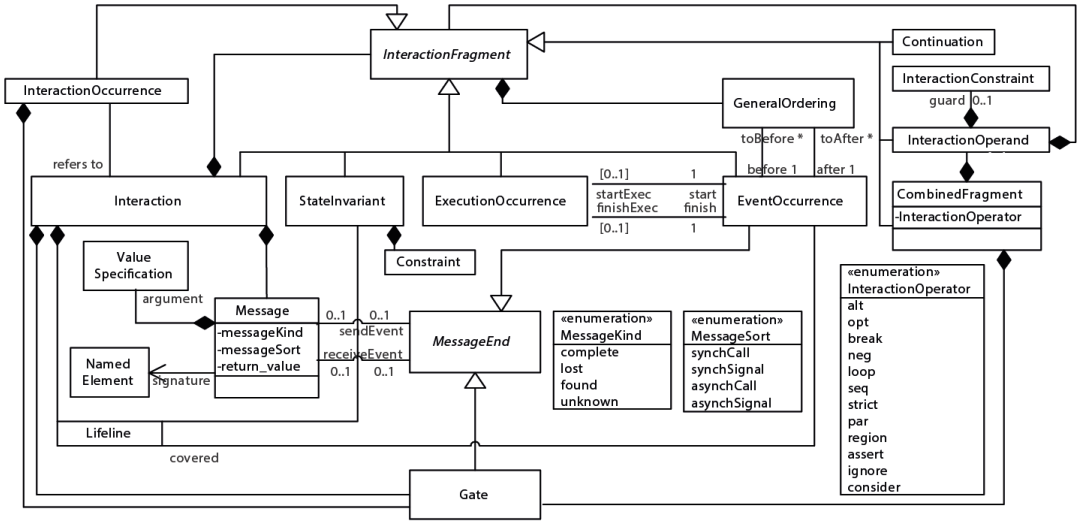


Figura 3.4: Metamodelo de diagramas de sequência UML (Garousi et al., 2005)

Como exemplo, tem-se diagramas de sequência (Figura - 3.5) que aparenta mensagens assíncronas. Para realizar o mapeamento para atividades (CCFG) foi utilizado um conjunto de regras criadas a partir dos metamodelos (Garousi et al., 2005). As regras são apresentadas na Tabela - 3.1. Com isso, o processo de mapeamento deu origem ao diagrama de atividades apresentado na Figura - 3.6.

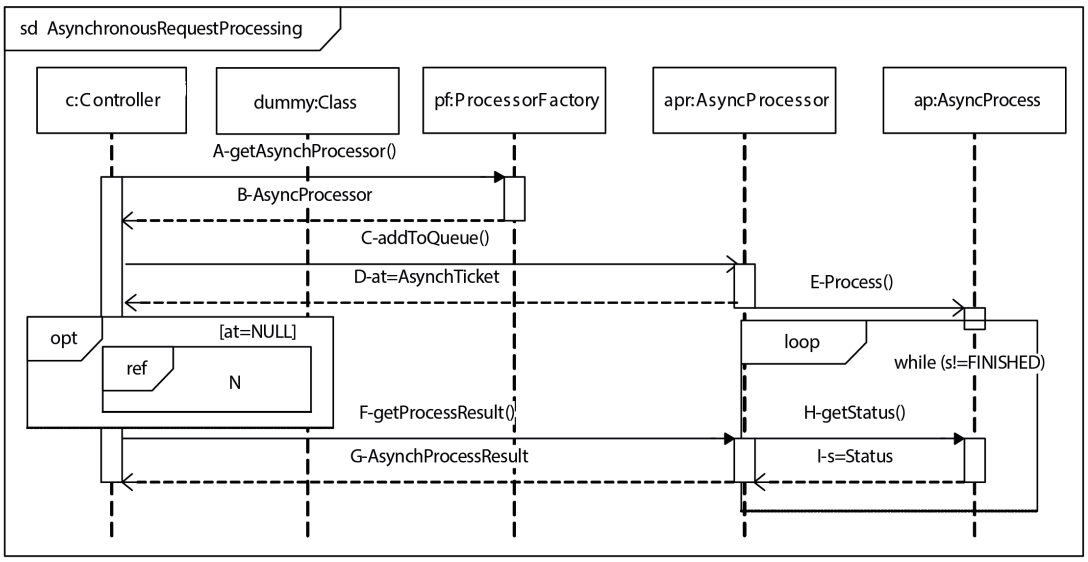


Figura 3.5: Um diagrama de sequência com mensagens assíncronas (Garousi et al., 2005)

Tabela 3.1: Regras utilizadas para a conversão de DS para DA (Garousi et al., 2005)

N°	Recurso do DS	Recurso do CCFG (Atividades)
1	Interaction	Activity
2	First message end	Flow between InitialNode and first control node
3	SynchCall/SynchSignal	CallNode
4	AsynchCall or AsynchSignal	(CallNode+ForkNode) or ReplyNode
5	Message SendEvent and ReceiveEvent	ControlFlow
6	Lifeline	ObjectPartition
7	par CombinedFragment	ForkNode
8	loop CombinedFragment	DecisionNode
9	alt/opt CombinedFragment	DecisionNode
10	break CombinedFragment	ActivityEdge
11	Last message ends	Flow between end control nodes and FinalNode
12	InteractionOccurrence	Control Flow across CCFGs
13	Polymorphic message	DecisionNode
14	Nested InteractionFragmen	Nested CCFGs

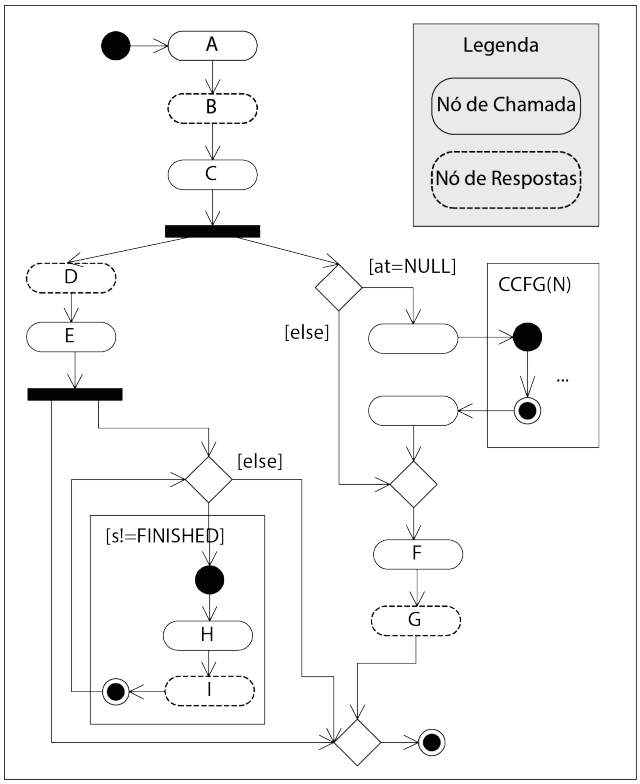


Figura 3.6: Resultado do mapeamento do DS da Figura - 3.5. Traduzido de Garousi et al. (2005)

Para a validação da abordagem *SMartyTesting* foi realizada a conversão manual dos DSs para DAs utilizando a ferramenta *Astah professional 6*[1]. Ao final da criação dos diagramas, os mesmos foram exportados com extensão XML, para serem utilizados na segunda etapa por SPLiT-MBt.

3.2.4 Ferramenta Utilizada

SPLiT-MBt faz o uso do método de geração HSI. A razão para a escolha desse método se deve ao fato de ser um dos métodos menos restritivos em relação às propriedades que as MEFs devem ter. Por exemplo, o HSI é capaz de interpretar MEFs completas e parciais (Petrenko, 1994).

Além disso, o método HSI permite cobertura total de falhas existentes e gera sequências de teste menores que outros métodos, o que contribui para a otimização do processo de teste. Esses fatores são muito relevantes no contexto da LPS, porque quanto maior o número de *features* de uma LPS, maior é o número de casos de teste necessários para testar produtos da LPS (Engström e Runeson, 2011).

Como houve necessidade de se estender a abordagem por causa da variabilidade, para gerar sequências de teste a partir de uma versão estendida do HSI, foi necessário aplicar o método considerando as informações de variabilidade presentes no MEF.

A SPLiT-MBt aceita, inicialmente, um arquivo de entrada em formato *eXtensible Markup Language* (XML), faz a leitura do arquivo e valida todos os requisitos de entrada do artefato. Caso esteja correto, monta-se a estrutura e, assim que o usuário clica na opção de geração, SPLiT-MBt converte o DA em MEF em tempo de execução e realiza o processo de geração de sequências de teste.

3.3 Especificação da Abordagem *SMartyTesting*

Nesta seção é apresentado cada passo da abordagem *SMartyTesting*. Neste trabalho, parte-se do princípio de que a modelagem de casos de uso tenha dado origem aos diagramas de sequência.

[1]Disponível em: http://astah.net/download - Acessado em 20/10/2019

3.3.1 Etapa 1 - Mapeamento de Diagramas de Sequência para Diagramas de Atividades

Nesta primeira etapa foi realizado o mapeamento dos diagramas de sequência (DS) para o de atividades estendido (DA). Para tanto, foi usada a LPS AGM. A etapa 1 é ilustrada na Figura - 3.7.

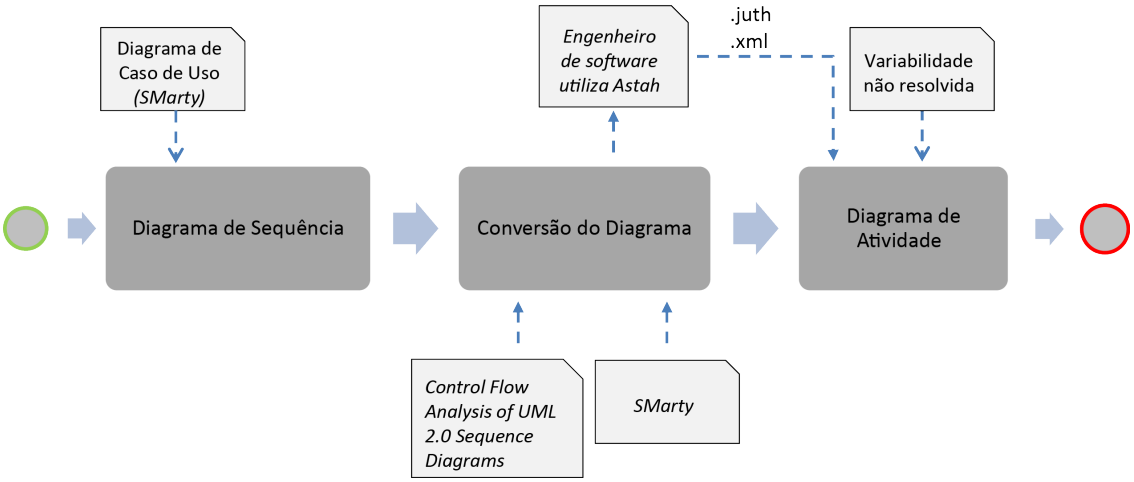


Figura 3.7: Ciclo da etapa 1

O DS foi modelado considerando a abordagem *SMarty* utilizando a ferramenta *Cameo Enterprise Architecture* versão 19 [1]. Depois de modelado o diagrama de sequência e todas as suas propriedades (Figura - 3.8), o engenheiro de software deverá construir um novo diagrama de atividades, porém levando em consideração as regras da Tabela - 3.1 e as propriedades do diagrama de sequência criado.

Neste trabalho utilizou-se outra ferramenta (IDE) de modelagem para a criação do modelo convertido de DS para DA, a ferramenta utilizada foi a *Astah* versão 6. Optou-se por essa versão a fim de demonstrar que mesmo com IDEs diferentes para modelagem, pode-se chegar ao mesmo resultado esperado.

A criação do DA deve levar em consideração o uso das regras de mapeamento contidas na Tabela - 3.1, as propriedades de variabilidade são mantidas sem a resolução, a fim de serem transportadas às sequências de teste para serem reutilizadas. A Figura - 3.9 apresenta o resultado da conversão do DS da Figura - 3.8.

Na execução da construção do DA deve-se levar em consideração os requisitos necessários que serão utilizados na Etapa 2. O arquivo do artefato modelado deve ser exportado da ferramenta em formato XML, que é o formato de entrada suportado por SPLiT-MBt.

[1] `https://www.nomagic.com/products/cameo-enterprise-architecture`

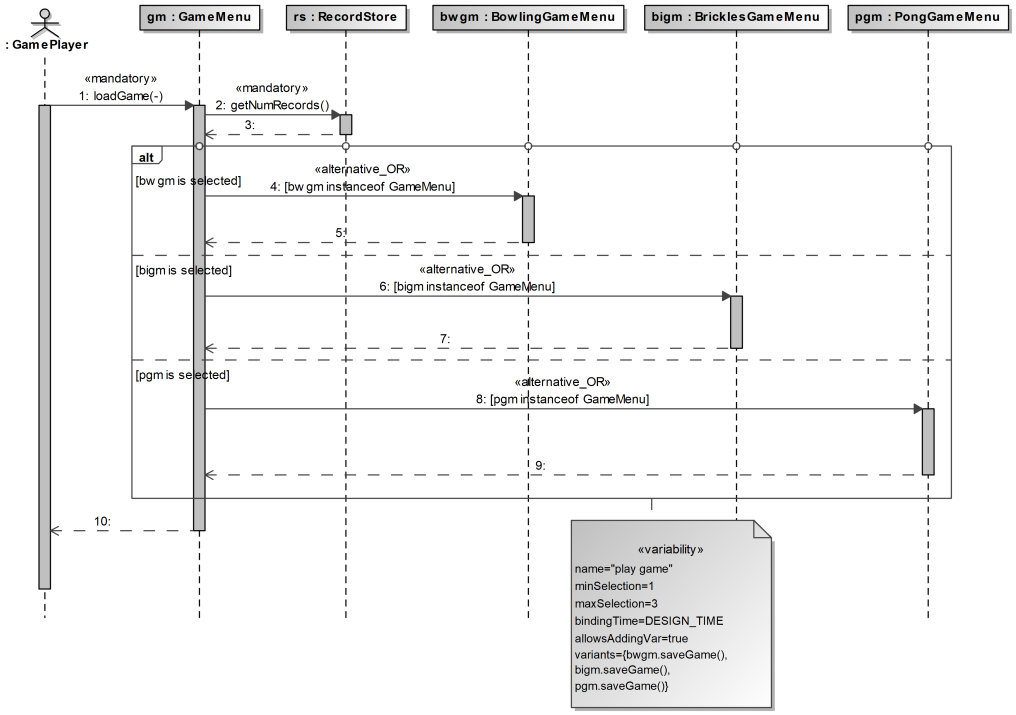


Figura 3.8: Diagramas de sequência ilustrando o caso de uso *game menu* da AGM

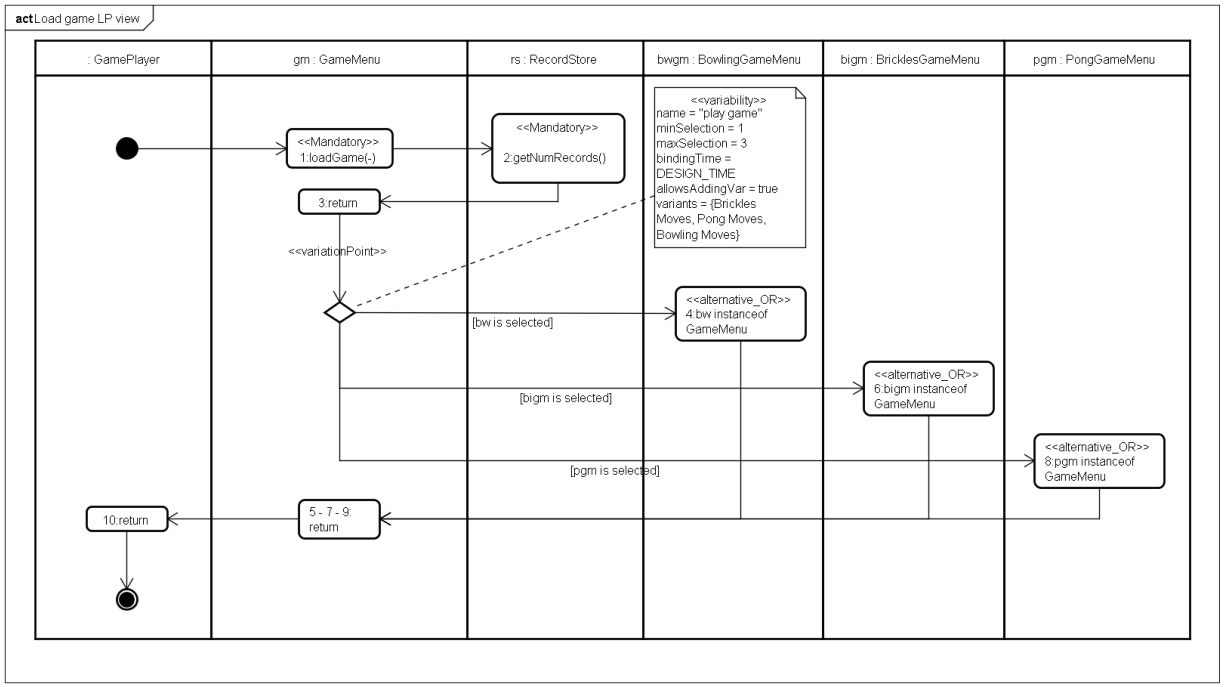


Figura 3.9: Diagramas de atividades resultante do mapeamento do DS da Figura - 3.8

Para a utilização do arquivo do artefato de DA, antes da exportação pela IDE, devem ser levadas em consideração algumas características. Uma delas é a parametrização das transições do DA, necessárias para utilização na ferramenta SPLiT-MBt. Deve ser levado em consideração também o estereótipo da atividade que está sendo modelada fazendo uso de *SMarty*.

Esses parâmetros são informações referentes às transições no DA. Assim, tem-se a transição da atividade, na que está contida a *tag* identificando qual é a ação (*TDaction*) e o resultado esperado (*TDexpectedResult*) e, por fim, os valores (*Values*) que são adicionados conforme as especificações do DA. A Figura - 3.10 apresenta uma visão dessa configuração necessária.

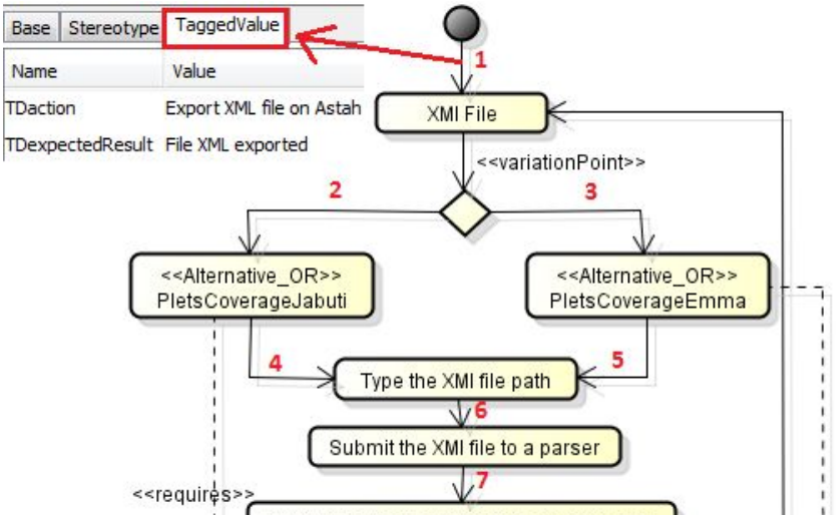


Figura 3.10: Exemplificação da parametrização dos *controlflows* de um diagrama de atividades usando SPLiT-MBt (Costa, 2016)

Seguindo essa parametrização necessária, a Tabela - 3.2 apresenta todos os parâmetros das transições do DA resultante do mapeamento.

Tabela 3.2: Valores dos atributos de parametrização do DA para utilização em SPLiT-MBt

Transição das Atividades	Tags	Valores
1- Estágio Inicial para LoadGame	TDaction TDexpectedResult	- Game Player faz o carregamento do jogo - Jogo carrega
2- LoadGame para getNumRecords	TDaction TDexpectedResult	- Game menu faz acesso ao dados de pontuação - Consegue acesso ao dados de pontuação
3- getNumRecords para return	TDaction TDexpectedResult	- recordStore retorna os dados - Dados de pontuação são retornados
4- Decision node para bw instanceof GameMenu	TDaction TDexpectedResult	- Opção bw é selecionada - Acesso aos recurso da opção bw
5- return	TDaction TDexpectedResult	- Retorno da opção bw - Retorne ao menu de opções
6- Decision node para bigm instanceof GameMenu	TDaction TDexpectedResult	- Opção bigm é selecionada - Acesso aos recurso da opção bigm
7- return	TDaction TDexpectedResult	- Retorno da opção bigm - Retorne ao menu de opções
8- Decision node para pgm instanceof GameMenu	TDaction TDexpectedResult	- Opção pgm é selecionada - Acesso aos recurso da opção pgm
9- return	TDaction TDexpectedResult	- Retorno da opção pgm - Retorne ao menu de opções
10- return	TDaction TDexpectedResult	- Retorno da informação ao Game Player - Player recebe retorno da ação escolhida

3.3.2 Etapa 2 - Geração de Sequências de Teste

Na segunda etapa considera-se que o DS já tenha sido mapeado e convertido para DA, conforme o fluxo apresentado na Figura - 3.11. Dado o mapeamento de conversão realizado faz-se a geração de sequências de casos de teste.

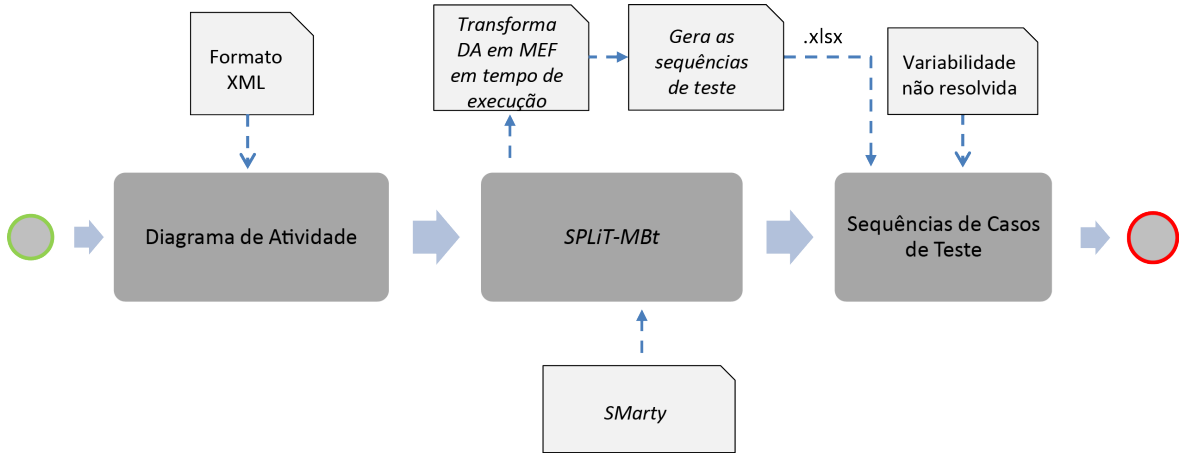


Figura 3.11: Ciclo da etapa 2

De posse do arquivo XML gerado, inicia-se o carregamento do arquivo na ferramenta (Figura - 3.12). Após o arquivo ser carregado pela aplicação, faz-se a verificação da estrutura do arquivo XML (Figura - 3.13).

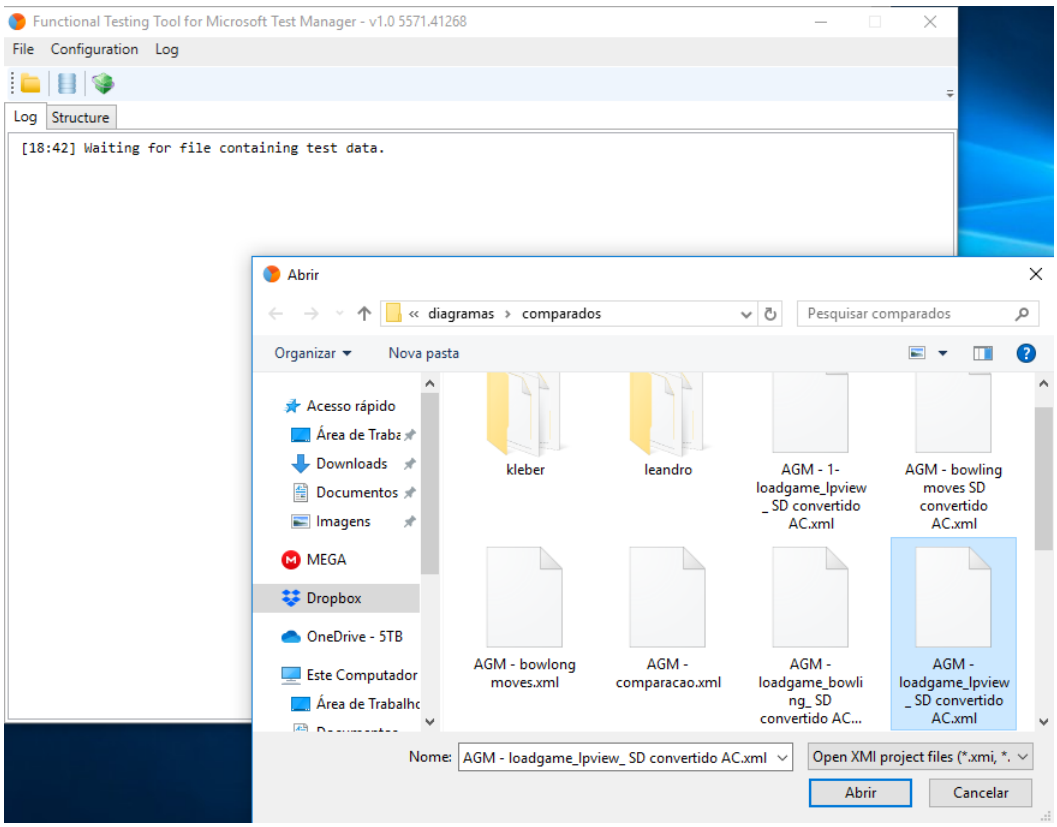


Figura 3.12: Tela de carregamento de arquivo da ferramenta SPLiT-MBt

Após a validação do arquivo que é carregado, pode ser observada a sua estrutura usando a própria ferramenta (Figura - 3.14), com isso pode-se realizar a geração de casos de teste e, quando não houver erros na geração, a mensagem de sucesso é apresentada (Figura - 3.15).

No momento da geração a ferramenta faz a conversão do DA em MEFs estendidas para suportar variabilidade. Essa conversão é realizada em tempo de execução, assim, não há acesso a nenhum arquivo ou processo, tornando a execução um processo interno da ferramenta.

Após a geração, é possível exportar um arquivo XLSx, que é umdos formatos de extensão aceitados pelo Excel, (Figura - 3.16) que pode ser utilizado em ferramentas de teste que fazem uso de *scripts*.

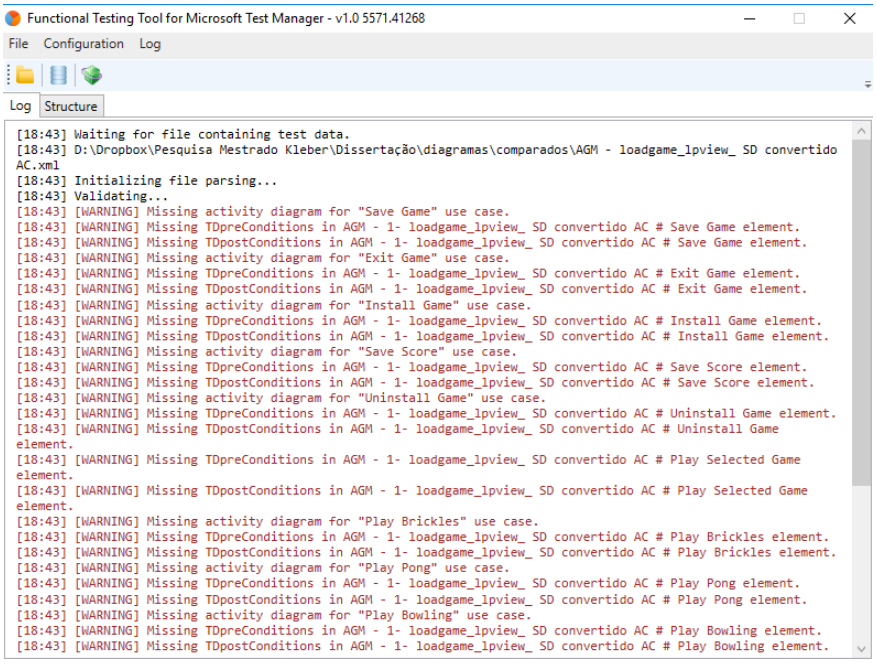


Figura 3.13: Validação da estrutura do arquivo carregado pela SPLiT-MBt

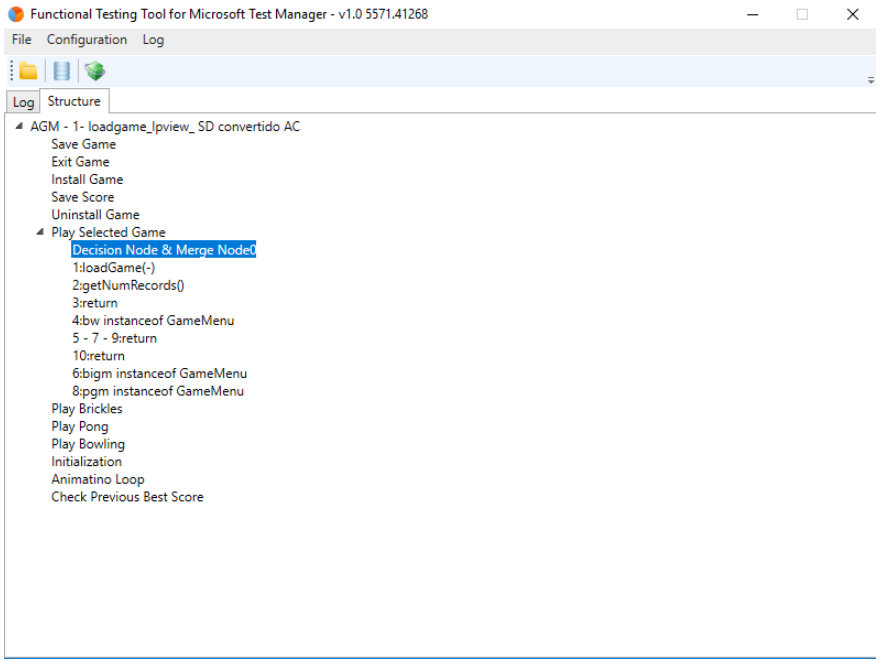


Figura 3.14: Apresentação da estrutura do arquivo XML

3.3.3 Resolução da Variabilidade

SMartyTesting trata inicialmente a resolução da variabilidade no mesmo formato que SPLiT-MBt trata, onde as sequências de casos de teste são geradas contendo a variabi-

```
[18:43] [WARNING] Missing TDpostConditions in AGM - 1- loadgame_lpview_ SD convertid
Score element.
[18:43] [WARNING] Missing TDpostConditions in AGM - 1- loadgame_lpview_ SD convertid
Score element.
[18:43] The next steps may not generate the expected results.
[18:45] Your test cases were generated
```

Figura 3.15: Mensagem de sucesso na geração dos casos de teste

	A	B	C	D	E	F	G
1	Test Case #	Work Item ID	Test Title	Summary	Test Step	Action/Description	Expected Results
2	TC000	Test Case 1000	Play Selected Game_1000		1	1:loadGame(-) - Game Player faz o carregamento do jogo{Mandatory};	Jogo carrega.
3					2	2:getNumRecords() - Game menu faz acesso ao dados de pontuacao{Mandatory};	Consegue acesso ao dados de pontuacao.
4					3	3:return - recordStore retorna os dados;	Dados de pontuacao sao retornados.
5					4	VP_3:return - {; - Opcao bw e selecionada{alternative_OR}; - Opcao bigm e selecionada{alternative_OR}; - Opcao pgm e selecionada{alternative_OR}};	{. Acesso aos recurso da opcao bw. Acesso aos recurso da opcao bigm. Acesso aos recurso da opcao pgm}.
6					5	5 - 7 - 9:return - {; - Retorno da opcao bw; - Retorno da opcao bigm; - Retorno da opcao pgm};	{. Retorne ao menu de opcaes. Retorne ao menu de opcoes. Retorne ao menu de opcoes}.
7					6	10:return - Retorno da informacao ao Game Player;	Player recebe retorno da acao escolhida.
8							

Figura 3.16: Arquivo XLSx com os casos de testes gerados

lidade sem a resolução. Isso é importante quando se aborda a questão da reutilização, uma vez que quando novos produtos são gerados, a variabilidade pode sofrer alterações, ou haver mais pontos de variação.

A viabilidade se torna desconhecida justamente por essa diferenciação que pode ser extensa, muitas instâncias e, quando se fala em reaproveitamento, a diferença pode sofrer um aumento e diferenciação, o que torna inviável a solução durante a modelagem.

Nesse caso, *SMartyTesting* gera as sequências de teste com a variabilidade sem resolução, para que o engenheiro de software faça a tomada de decisão na engenharia de aplicação, analisando a melhor solução para a resolução da variabilidade contida.

3.3.4 Limitações de Uso da SPLiT-MBt

Por se estar utilizando ferramentas de apoio de terceiros, era previsto que limitações poderiam ser encontradas. Nesse caso, foram encontradas limitações em ambas as etapas.

Na primeira, com relação à utilização da notação *SMarty* no mapeamento de conversão, pois *Control Flow Analysis of UML 2.0 Sequence Diagrams* não cita soluções sobre variabilidade, embora, não sejam encontrados obstáculos sobre notações na utilização, a limitação ocorre somente por não ter uma regra explícita sobre notação de variabilidade.

Já na segunda etapa, envolvendo a aplicação de SPLiT-MBt, as limitações são mais significativas, embora não tenham influenciado nas validações. As limitações estão ligadas diretamente à estrutura do metamodelo do diagrama de atividades (Figura - 3.3).

SPLiT-MBt não fornece suporte a dois itens de *ControlNode*, *ForkNode* e *JoinNode*, em testes com programação concorrente (atividades simultâneas).

Outro ponto de limitação está ligado à continuidade de uma atividade após um *DecisionNode* em que não são aceitas sub atividades a não ser que seja um estado de *Merge*. Assim, considera-se isso uma não conformidade com sub-sistemas representados no artefato de entrada, um exemplo seria um *InitialNode* vide (Figura - 3.6).

Em um processo que não seja automatizado pode haver ameaças ao funcionamento, nesse caso a primeira etapa. Por esse motivo, é recomendado em trabalhos futuros a implementação e automatização de todos os processos do início ao fim do ciclo.

Pode-se considerar outra ameaça ao funcionamento da *Control Flow Analysis of UML 2.0 Sequence Diagrams*, a metodologia de mapeamento. Por não conter uma regra explícita sobre variabilidade, pode-se entender que a interpretação errada de uma das regras ocasione um erro de conversão.

3.4 Considerações Finais

Neste tópico foi apresentada a estrutura de utilização da abordagem *SMartyTesting*, partindo da caracterização da abordagem, a definição dos processos e o porquê das escolhas dos artefatos utilizados. Foi apresentado o modelo de conversão de DS para DA e as ferramentas utilizadas nas etapas posteriores. Na especificação detalhada, as duas etapas, em que se conta com o artefato de entrada DS, faz-se conversão para DA que é o artefato de entrada para a etapa seguinte.

4

Estudo de Viabilidade

4.1 Considerações Iniciais

Neste tópico é apresentado um estudo de viabilidade da abordagem *SMartyTesting*.

Um estudo de viabilidade é essencial para verificar se existem evidências iniciais para o desenvolvimento aprofundado de determinada teoria ou tecnologia, neste caso, da abordagem *SMartyTesting*. Para isso, hipóteses foram estabelecidas com base em critérios de avaliação com relação ao uso de DS e DA para a geração de sequências de teste.

4.2 Objetivo

Com base nas considerações iniciais, o propósito principal deste estudo é analisar se ***SMartyTesting* possui condições e meios para a geração de sequências de casos de teste utilizando diagramas de sequência**.

Baseado no modelo *Goal Question Metric* (GQM) (Basili, 1992), este estudo visa: **Caracterizar a abordagem *SMartyTesting***, **com o propósito de** identificar a sua viabilidade **com relação à** capacidade de geração de sequências de teste a partir de diagramas de sequência do ponto de vista de pesquisadores de LPS, **no contexto de** docentes e aluno de pós-graduação em Engenharia de Software da Universidade Estadual de Maringá (UEM).

4.3 Planejamento

4.3.1 Critérios de Avaliação

Para alcançar o objetivo deste estudo, foram definidos os seguintes critérios:

- **CT.1:** quantidade de sequências de teste geradas. Esse critério tem como base as sequências de teste geradas por cada abordagem, em que uma sequência de teste pode apresentar um número X de casos de teste;
- **CT.2:** diferenciação entre sequências geradas. Quando é feita a geração das sequências, os artefatos de entrada de cada abordagem diferem uns dos outros por causa das particularidades do modelo inicial. Sendo assim, pode-se obter caminhos de sequências diferentes uns dos outros, demonstrando que foram tomados caminhos diferentes, diferenciando-se um dos outros;
- **CT.3:** complexidade das sequências de teste geradas. Para verificação da complexidade de um programa de computador, McCabe (1976) desenvolveu uma métrica para medição chamada de complexidade ciclomática, que mede a quantidade de caminhos em execução em uma aplicação. Assim, será usada essa métrica para verificar a complexidade das sequências de teste geradas; e
- **CT.4:** esforço no uso da abordagem para geração de sequência de teste. Quanto à medição do esforço, por meio de observação, foi verificado o tempo gasto para a utilização da abordagem desde a entrada do artefato inicial até as sequências de teste geradas. Nesse caso, verifica-se o esforço de utilização, para isso realizou-se uma observação em relação ao tempo de utilização de cada abordagem, do inicio até a geração das sequências de teste.

4.3.2 Hipóteses do Estudo

As seguintes hipóteses foram estabelecidas de acordo com o objetivo principal deste estudo e os critérios definidos:

- **Hipótese** $H_{CT.1}$**:** a geração de sequências fazendo uso de Diagramas de Sequência por meio da abordagem *SMartyTesting* gera mais sequências de teste do que o uso de Diagramas de Atividades utilizando SPLiT-MBt;

- **Hipótese $H_{CT.2}$:** a utilização de Diagramas de Sequência por meio da abordagem *SMartyTesting* cria diferenciação entre sequências de teste, em comparação aos Diagramas de Atividades com SPLiT-MBt;
- **Hipótese $H_{CT.3}$:** a utilização de Diagramas de Sequência com a abordagem *SMartyTesting* gera menor complexidade das sequências de teste geradas, em comparação com os Diagramas de Atividades com SPLiT-MBt; e
- **Hipótese $H_{CT.4}$:** a utilização de Diagramas de Sequência com a abordagem *SMartyTesting* possui menor esforço na geração de sequências de teste, comparado com o uso de Diagramas de Atividades com SPLiT-MBt.

A Figura - 4.1 ilustra as abordagens e as hipóteses definidas.

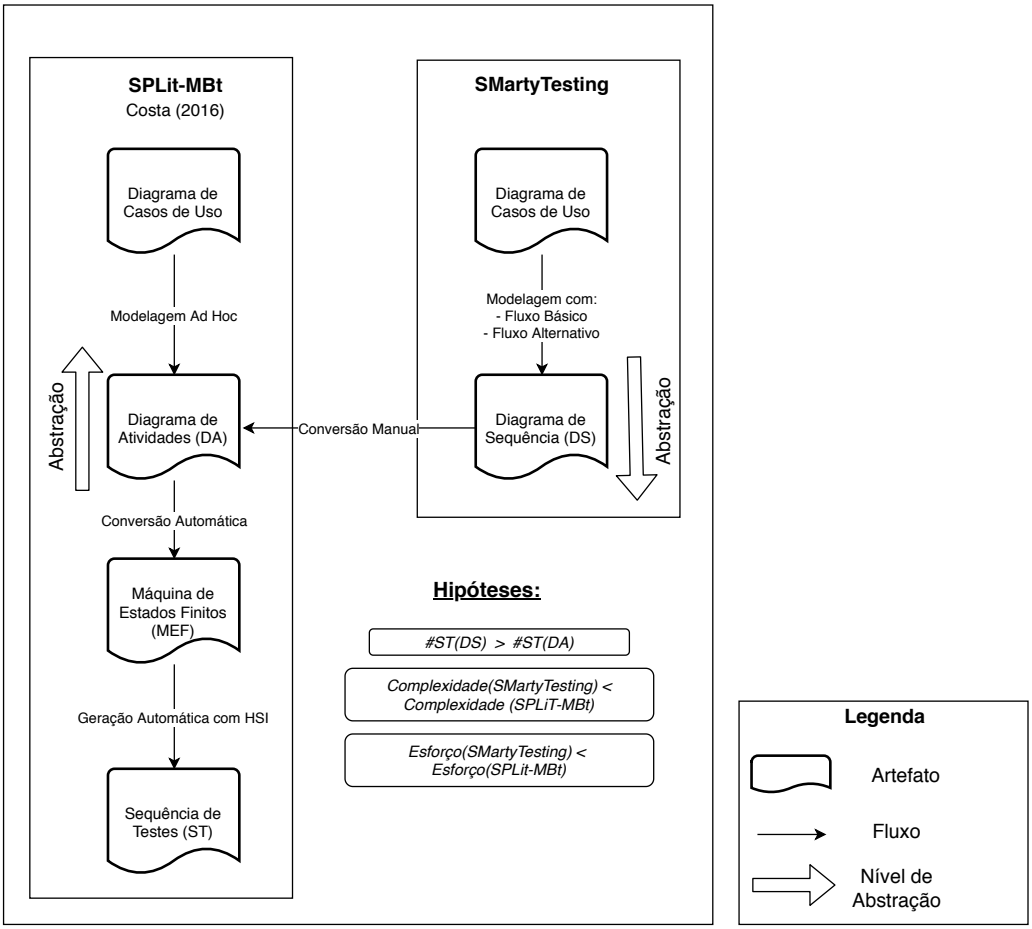


Figura 4.1: Estudo de viabilidade e hipóteses definidas

4.3.3 Instrumentação

Para a geração das sequências de teste foram selecionados três diagramas que contivessem elementos necessários para a validação. A Tabela - 4.1 contém as características assim como as variabilidades que cada diagrama possui.

Tabela 4.1: Diagramas utilizados no processo de geração de sequências de teste.

Modelos	Característica	Variabilidade
Play Selected Game Figura - 4.2 (DA) Figura - 4.5 (DS)	*Play Selected Game* é a representação do menu do jogo. Por meio dela é feita a seleção de qual jogo será jogado.	-ponto de variação -variabilidade *-alternative_OR*
Save Game Figura - 4.3 (DA) Figura - 4.7 (DS)	*Save Game* é a ação de salvar o jogo.	*-mandatory*
Pong Moves Figura - 4.4 (DA) Figura - 4.9 (DS) Figura - 4.10 (DS)	*Pong Moves* são as ações e movimentos do jogo *Pong*.	- não possui

4.3.4 Procedimentos de Análise

O procedimento de análise segue conforme os critérios definidos na Seção 4.3.1. Porém, para cada critério foi utilizada uma forma de análise diferente, como segue:

- **CT.1:** utilizando comparação de resultados. Quantidade de casos de teste apresentados nas sequências de teste geradas por cada abordagem;
- **CT.2:** utilização de análise de quantidade de caminhos criados por cada geração de sequência. Se uma abordagem gera mais sequências, isso significa que mais caminhos podem ser visitados;
- **CT.3:** para o cálculo de complexidade, foi utilizada a complexidade ciclomática 4.5.3, métrica bastante utilizada para a determinação da quantidade de caminhos seguidos por uma aplicação computacional; e
- **CT.4:** esforço é baseado no tempo de utilização de cada abordagem, em que a abordagem que levar maior tempo para a construção do objetivo final é considerada a de maior esforço de utilização.

4.4 Execução

4.4.1 Geração de Sequências de Teste com SPLiT-MBt

Como mencionado anteriormente, foram utilizados três diagramas de atividades que fazem parte da LPS AGM e que foram modelados por Costa (2016), são eles: Figura - 4.2, Figura - 4.3 e Figura - 4.4.

Neste trabalho, cada um deles é acompanhado de uma tabela contendo a sequência de teste gerada por SPLiT-MBt, são elas: Tabela - 4.2, Tabela - 4.3 e Tabela - 4.4. Esses dados serão utilizados na Seção 4.5 para comparação com diagramas equivalentes aos apresentados por Costa (2016). A Figura - 4.2 apresenta um diagrama de atividades da LPS AGM que foi utilizado pela SPLiT-MBt para a comparação.

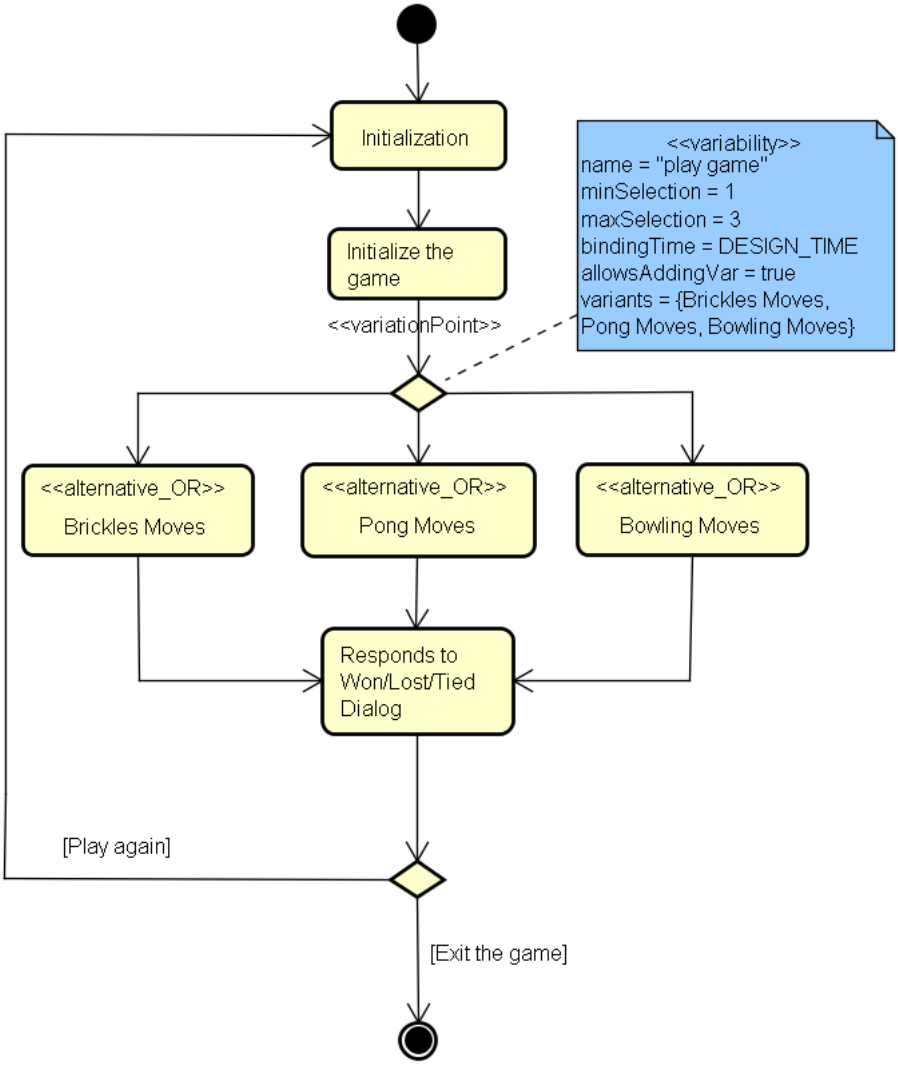


Figura 4.2: Diagrama de Atividades *Play Selected Game* (Costa, 2016)

A Tabela - 4.2 apresenta os resultados da sequência de teste gerada da Figura - 4.2.

Tabela 4.2: Sequências de teste do DA *Play Selected Game* da AGM da Figura - 4.2 gerado por SPLiT-MBt

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	1	*Initialization* - *Select Play from menu*;	Cria as instâncias padrão das classes necessárias.
Test Case 1	2	*Initialize the game* - *Left-click Button to begin play;*	Inicie a ação do jogo e a animação começa.
Test Case 1	3	*VP_Initialize the game* - {; - b{*alternative_OR*}; - c{*alternative_OR*}; - a{*alternative_OR*}};	{. As pás e o disco começam a se mover. A bola começa a se mover. Move a raquete horizontalmente para seguir a trilha do mouse }.
Test Case 1	4	*Responds to Won/Lost/Tied Dialog* - {; - *Responds to Won/Lost/Tied dialog*; - *Responds to Won/Lost/Tied dialog*; - *Responds to Won/Lost/Tied dialog* };	{. Retorne ao estado inicial do tabuleiro. Retorne ao estado inicial do tabuleiro. A caixa de diálogo para reproduzir novamente é apresentada}.
Test Case 1	5	*Initialization* - *Respond"yes"in the dialog to play again*;	Retorna o tabuleiro de jogo ao seu estado inicializado, pronto para jogar.

A Figura - 4.3 apresenta um diagrama de atividades da LPS AGM que foi utilizado pela SPLiT-MBt para a comparação.

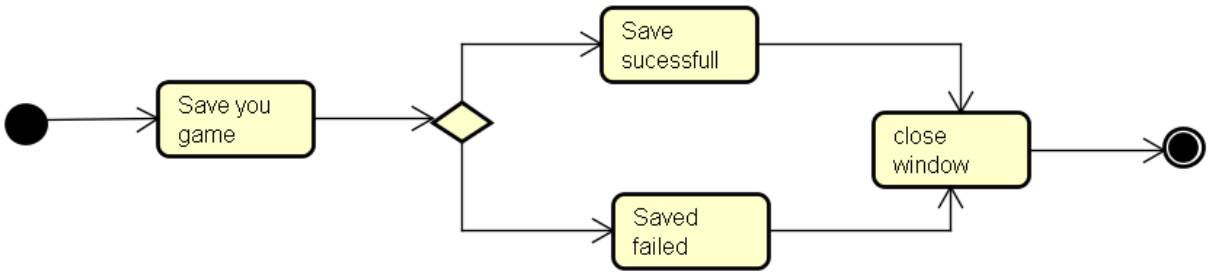


Figura 4.3: Diagrama de Atividades *Save Game* (Costa, 2016)

A Tabela - 4.3 apresenta os resultados da sequência de teste gerada da Figura - 4.3.

Tabela 4.3: Sequências de teste do DA *Save Game* da Figura - 4.3 geradas por SPLiT-MBt

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	1	*Save your game* - *save GAME window are showed*;	Termine o jogo.
Test Case 1	2	*Saved failed* - *click SAVE GAME button*;	mensagem SALVAR JOGO com falha é mostrada.
Test Case 1	3	*close window* - *Click close SAVE THE GAME*;	A janela SALVAR JOGO é fechada.
Test Case 2	1	*Save your game* - *save GAME window are showed*;	Termine o jogo.
Test Case 2	2	*Save sucessfull* - *click SAVE GAME button*;	messagem SALVAR JOGO é mostrada.
Test Case 2	3	*close window* - *click close SAVE GAME button*;	A janela SALVAR JOGO é fechada.

A Figura - 4.4 apresenta um diagrama de atividades da LPS AGM que foi utilizado pela SPLiT-MBt para a comparação.

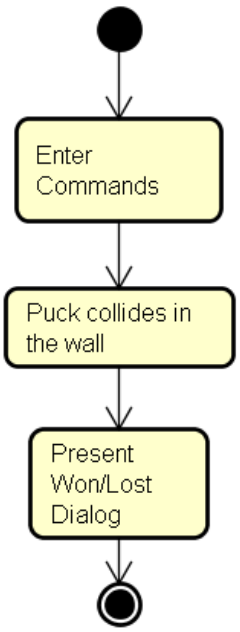


Figura 4.4: Diagrama de Atividades *Pong Moves* (Costa, 2016)

A Tabela - 4.4 apresenta os resultados da sequência de teste gerada da Figura - 4.4.

Tabela 4.4: Sequencia de teste do DA *Pong Moves* da Figura - 4.4 gerada por SPLiT-MBt

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	1	*Enter Commands* - *Pong*;	Disco começa a se mover.
Test Case 1	2	*Puck collides in the wall* - *Let the puck collide into the walls*;	Com base nas regras, o disco é absorvido ou muda de direção de acordo com as leis da física.
Test Case 1	3	*Present Won/Lost Dialog* - *The puck is absorbed by the left or right side of the playing field*;	A caixa de diálogo Ganhos / Perdas é apresentada.

4.4.2 Geração de Sequências de Teste com *SMartyTesting*

Após a geração da sequência de teste com DA como artefatos de entrada da SPLiT-MBt, são utilizados DS criados por Marcolino et al. (2017), equivalentes aos DA criados por Costa (2016) que são os da Figura - 4.5, Figura - 4.7 e Figura - 4.9.

Essa equivalência se deve ao fato do nível de abstração utilizado, um exemplo é a Figura - 4.3 que representa o *Save Game*, nela são observadas duas condições, onde houve sucesso e onde houve falha no salvamento. Nesse caso, a representação em DS por ser mais próxima ao código dar-se-ia em dois diagramas: um para sucesso e outro para falha. A Figura - 4.7 representa somente a condição de sucesso.

Após essa criação, os DS foram utilizados em *SMartyTesting* para a geração de sequências de teste para comparação. Na etapa inicial foi realizada a conversão para DA conforme pode ser visto nas figuras: Figura - 4.6, Figura - 4.8 e Figura - 4.11, que na etapa seguinte foram utilizados para a geração das sequências de teste: Tabela - 4.5, Tabela - 4.6 e Tabela - 4.7.

A Figura - 4.5 apresenta um diagrama de sequência da LPS AGM que foi utilizado por *SMartyTesting* para a conversão para DA.

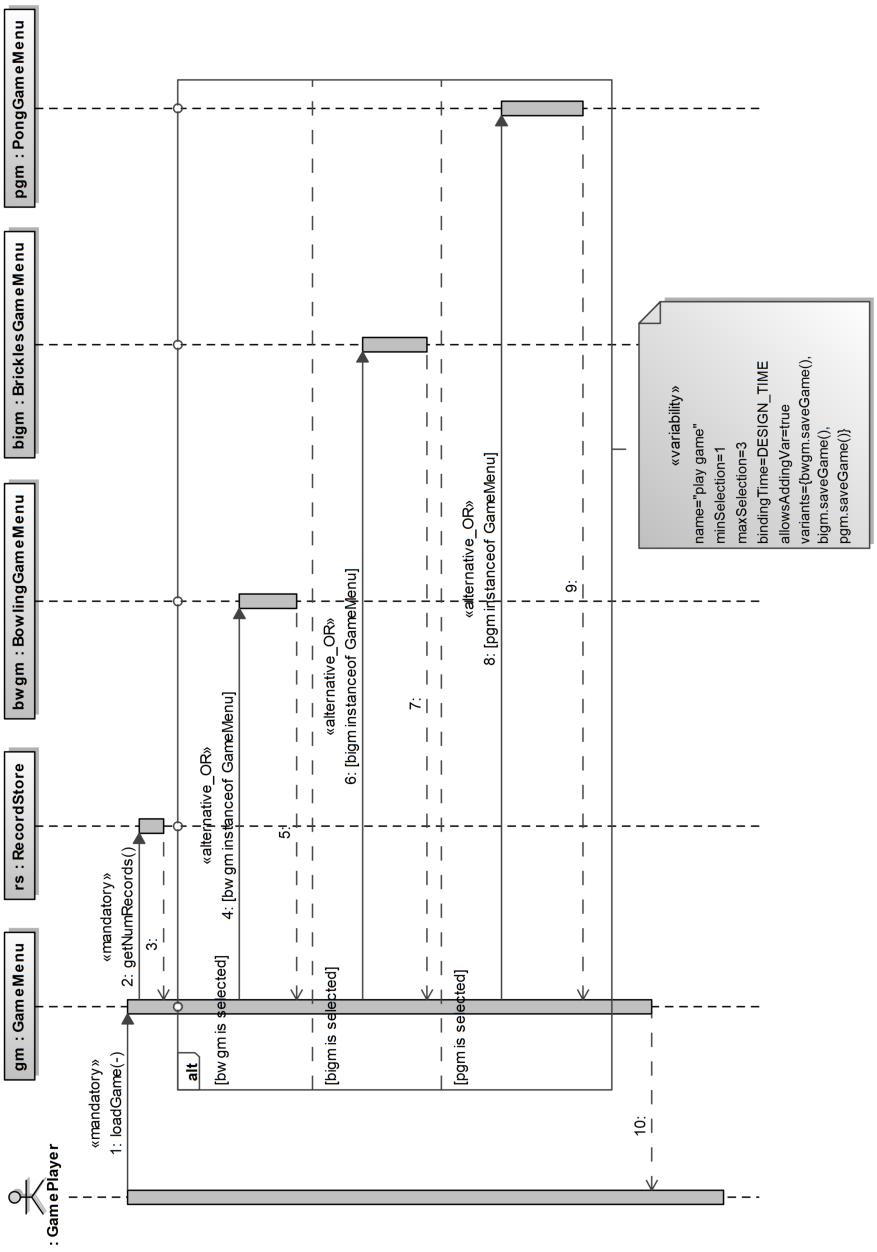


Figura 4.5: Diagrama de Sequência *Play Selected Game* (Marcolino et al., 2017)

A Figura - 4.6 apresenta um diagrama de atividades da LPS AGM que foi convertido por *SMartyTesting* para a geração de sequências de teste.

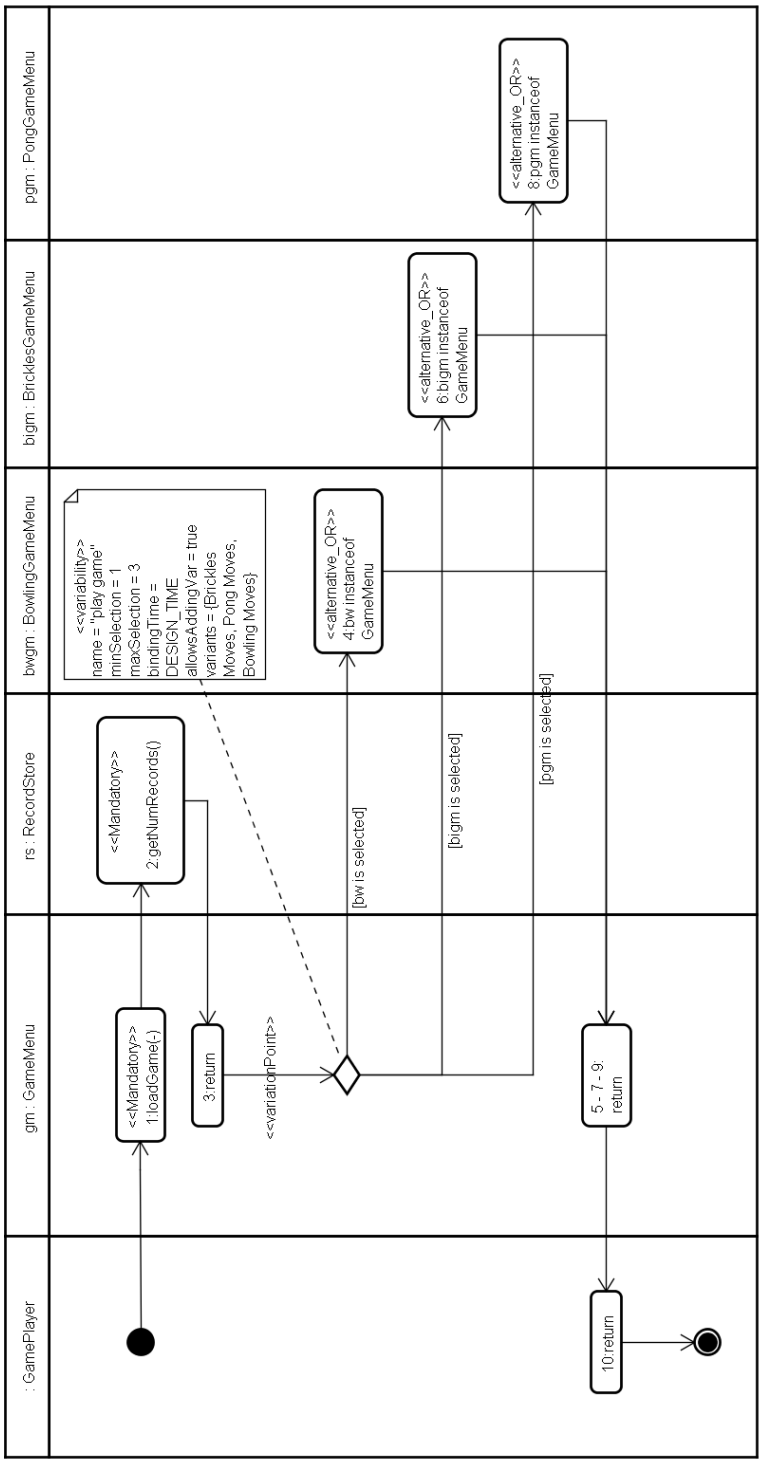


Figura 4.6: Diagrama de Atividades resultantes do Diagrama de Sequência da Figura - 4.5

A Tabela - 4.5 apresenta os resultados da sequência de teste gerada da Figura - 4.6.

Tabela 4.5: Sequências de casos de teste DA da Figura - 4.6 geradas por *SMartyTesting*

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	1	1:*loadGame*(-) - *Game Player* dispara método *loadGame*{*Mandatory*};	*loadGame* é carregado.
Test Case 1	2	2:*getNumRecords*() - *Game menu* após carregado faz uso do método *getNumRecords*{*Mandatory*};	acessa dados de *recordStore*.
Test Case 1	3	3:*return* - *recordStore* envia mensagens de retorno;	Dados de pontuação são retornados por *getNumrecords* para *GameMenu*.
Test Case 1	4	VP_3:*return* - {; - Opção bw é selecionada{*alternative_OR}*; - Opção bigm é selecionada{*alternative_OR}*; - Opção pgm é selecionada{*alternative_OR}*};	{. Instancia recurso da opção *bowling*. instancia recurso da opção bigm *brickles*. instancia recurso da opção *pong*}.
Test Case 1	5	5 - 7 - 9:*return* - {; - Retorno da opção bw; - Retorno da opção bigm; - Retorno da opção pgm};	{. Retorna após bw *instanceof GameMenu* for executada. Retorna após bigm *instanceof* GameMenu for executada. Retorna após pgm *instanceof GameMenu* for executada}.
Test Case 1	6	10:*return* - Retorno da informação ao *Game Player*;	*Player* recebe retorno da ação escolhida.

A Figura - 4.7 apresenta um diagrama de sequência da LPS AGM que foi utilizado por *SMartyTesting* para a conversão para DA.

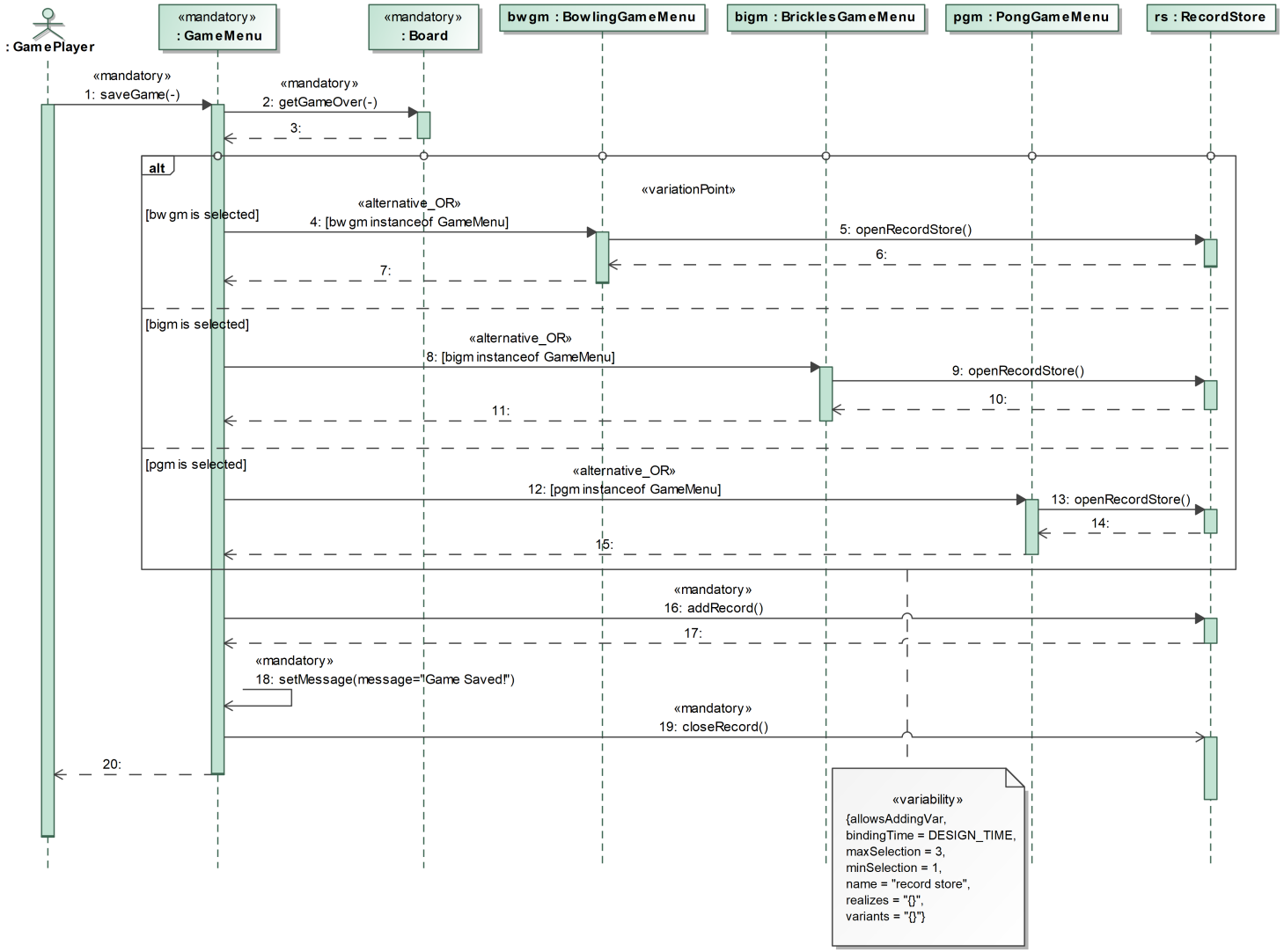


Figura 4.7: Diagrama de Sequência *Save Game* (Marcolino et al., 2017)

A Figura - 4.8 apresenta um diagrama de atividades da LPS AGM que foi convertido por *SMartyTesting* para a geração de sequências de teste.

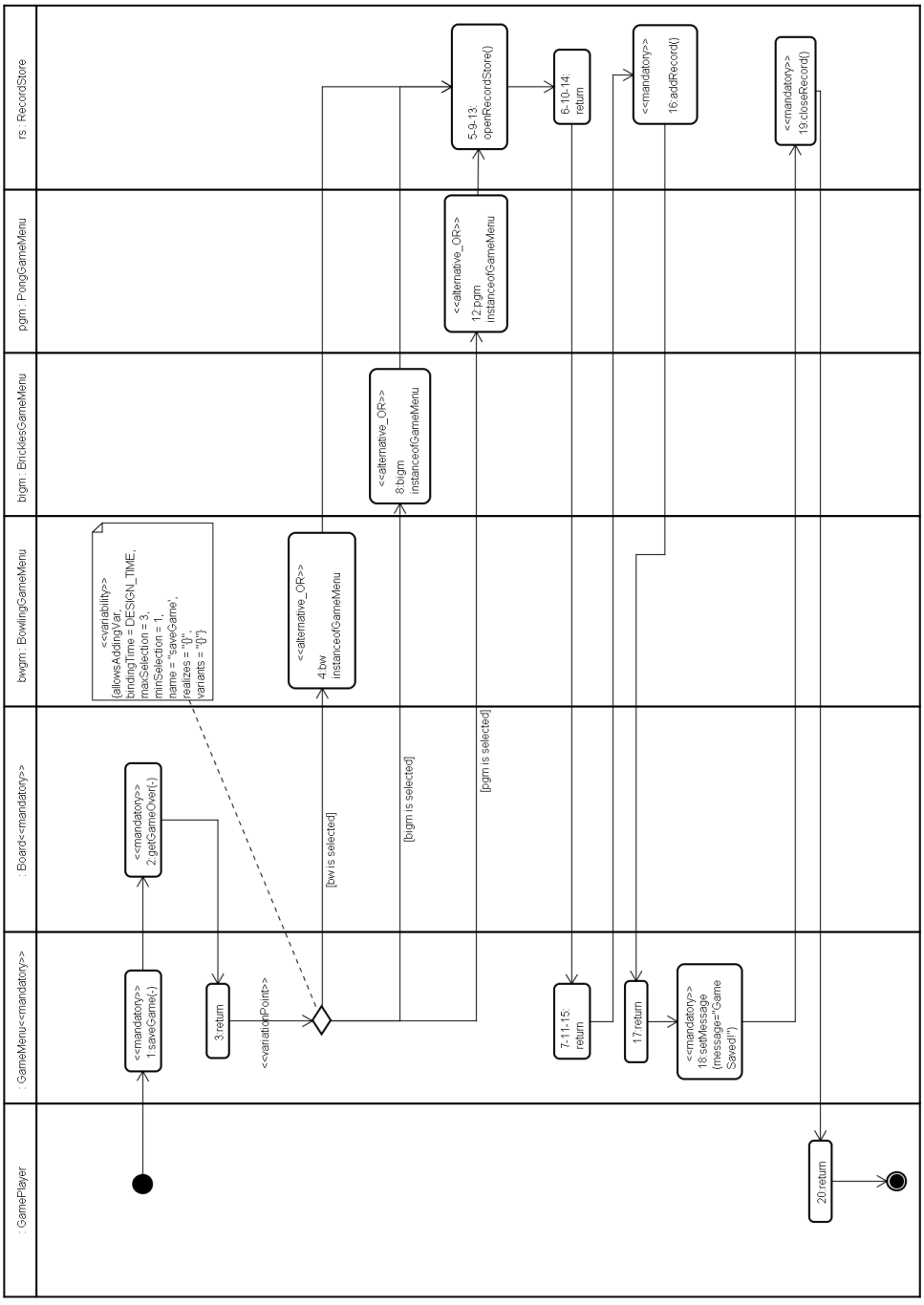


Figura 4.8: Diagrama de Atividades resultante dos Diagrama de Sequência da Figura - 4.7

A Tabela - 4.6 apresenta os resultados da sequência de teste gerada da Figura - 4.8.

Tabela 4.6: Sequências de teste DA da Figura - 4.8 geradas por *SMartyTesting*

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	1	1:*saveGame*(-) - *GamePlayer* dispara método *saveGame*{*mandatory*};	*GameMenu* é carregado.
Test Case 1	2	2:*getGameOver*(-) - *GameMenu* dispara método *getGameOver*{*mandatory*};	*Board* verifica ação.
Test Case 1	3	3:*return* - *Board* retorna solicitação;	Valor retorna para jogo selecionado.
Test Case 1	4	VP_3:*return* - {; bw é selecionado{*alternative_OR*}; - bigm é selecionado{*alternative_OR*}; - pgm é selecionado{*alternative_OR*}};	{. dispara método *instanceofGameMenu*. dispara método *instanceofGameMenu*. dispara método *instanceofGameMenu*}.
Test Case 1	5	5-9-13:*openRecordStore*() - {; bw envia dados para método *openRecordStore*; - bigm envia dados para método *openRecordStore*; - pgm envia dados para método *openRecordStore*};	{. dados são utilizados por *openRecordStore*. dados são utilizados por *openRecordStore*. dados são utilizados por *openRecordStore*}.
Test Case 1	6	6-10-14:*return* - *operRecordStore* retorna para bw - bigm - pgm;	confirma se possui dados disponíveis.
Test Case 1	7	7-11-15:*return* - Return dos dados para *GameMenu*;	dados disponíveis são retornados para serem adicionados.
Test Case 1	8	16:*addRecord*() - acionado método *addRecord*{*mandatory*};	Dados são salvos.
Test Case 1	9	17:*return* - retorna ação do *addRecord*;	Confirma persistência dos dados.
Test Case 1	10	18:*setMessage(message="Game Saved!")* - dispara mensagem de confirmação{*mandatory*};	Mensagem de confirmação apresentada.
Test Case 1	11	19:*closeRecord*() - Dispara método *closeRecord*{*mandatory*};	Método encerra operação.
Test Case 1	12	20:*return* - retornar confirmação da operação;	Concluída com sucesso devolver Confirmação ao usuário

A Figura - 4.9 e a Figura - 4.10 apresentam um diagrama de sequência da LPS AGM que foi utilizado por *SMartyTesting* para a conversão para DA.

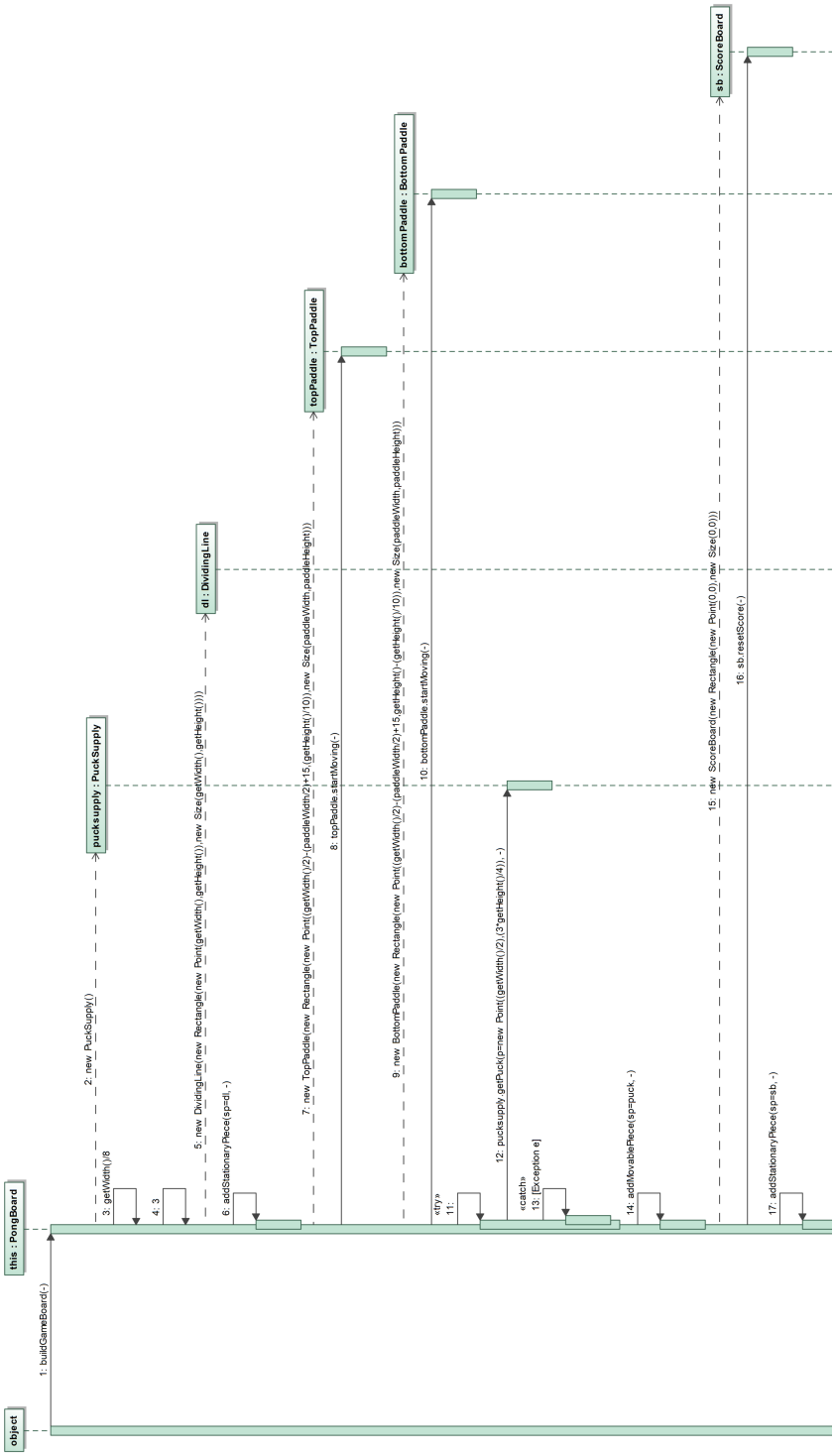


Figura 4.9: Diagrama de Sequência *Pong Moves* 1 (Marcolino et al., 2017)

Figura 4.10: Diagramas de Sequência *Pong Moves* 2 (Marcolino et al., 2017)

As Figura - 4.11 e Figura - 4.12 apresentam um diagrama de atividades da LPS AGM que foi convertido por *SMartyTesting* para a geração de sequências de teste.

A Tabela - 4.7 apresenta os resultados da sequência de teste gerada das Figura - 4.11 e Figura - 4.12

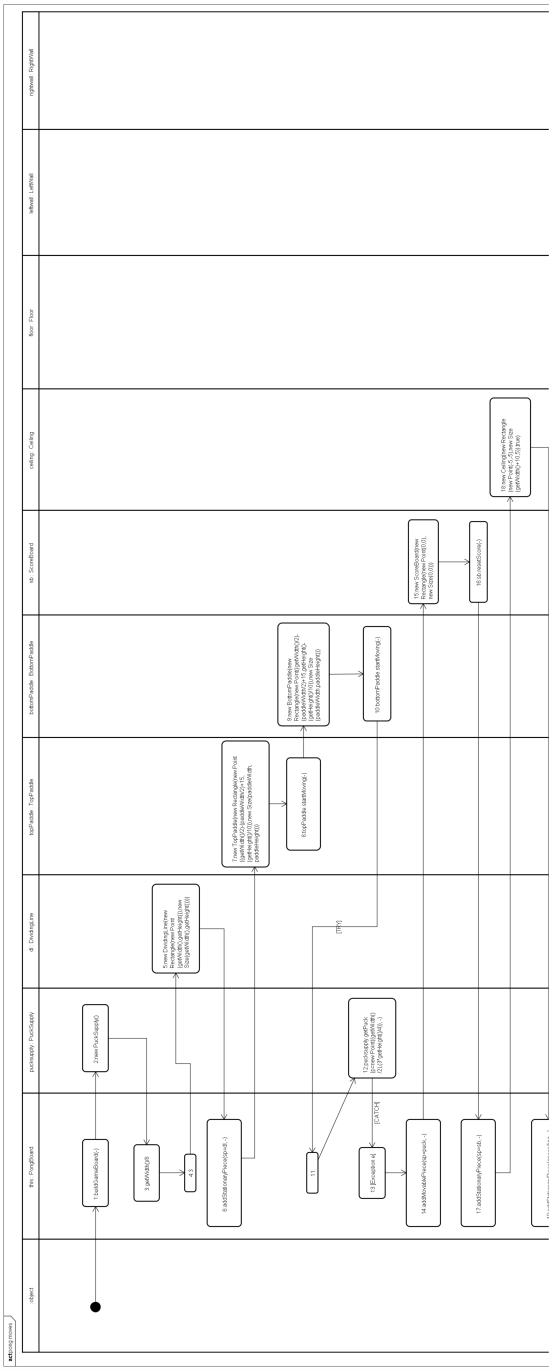


Figura 4.11: Diagrama de Atividades resultante do Diagrama de Sequência da Figura - 4.9 e da Figura - 4.10 parte 1

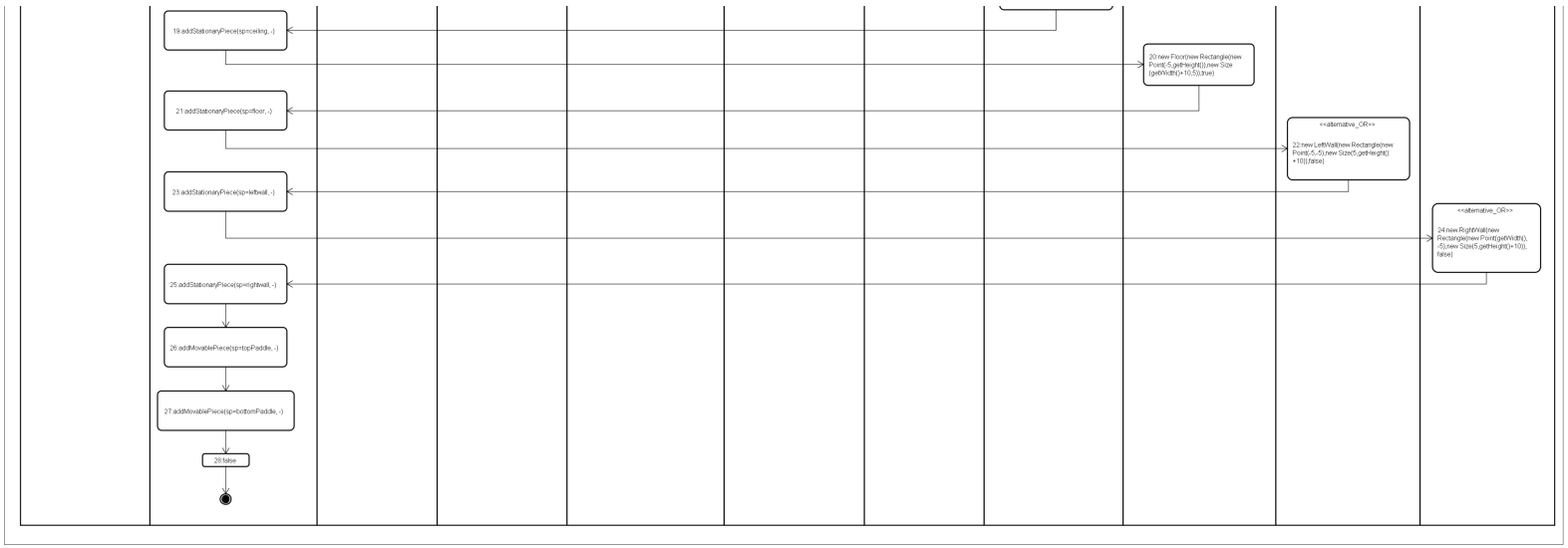


Figura 4.12: Diagrama de Atividades resultante do Diagrama de Sequência da Figura - 4.9 e da Figura - 4.10 parte 2

Tabela 4.7: Sequências de teste do DA da Figura - 4.11
geradas por *SMartyTesting*

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	1	*buildGameBoard*(-) - método *buildGameBoard* chamado;	recebe parâmetro de inicialização.
Test Case 1	2	*new PuckSupply*() - instancia do método *newPuckSupply*;	método iniciado.
Test Case 1	3	*getWidth*()/8 - *getWidth*()/;	captura do tamanho.
Test Case 1	4	3 - *return*;	retornar valor.
Test Case 1	5	- *new DividingLine(new Rectangle(new Point(getWidth()* *,getHeight()),new Size(getWidth(),getHeight())))*	tamanho dos objetos definidos.
Test Case 1	6	- *addStationaryPiece*(sp=dl, -)	complementos adicionados.
Test Case 1	7	- *new TopPaddle(new Rectangle(new Point((getWidth()/2)* -(*paddleWidth*/2) 15,(*getHeight*()/10)),*new* *Size(paddleWidth,paddleHeight))*)	posição dos objetos definidos.
Test Case 1	8	- *topPaddle.startMoving*(-)	*paddle* permite movimentos.
Test Case 1	9	- *new BottomPaddle(new Rectangle(new Point((getWidth()/2)* -(*paddleWidth*/2) 15,*getHeight*()-(*getHeight*()/10)), *new Size(paddleWidth,paddleHeight)*));	posição dos objetos definidos.

Tabela 4.7: Sequências de teste do DA da Figura - 4.11
geradas por *SMartyTesting*

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	10	- *topPaddle.startMoving*(-);	*paddle* permite movimentos.
Test Case 1	11	11: - *return*;	verifica posição e movimento.
Test Case 1	12	- *pucksupply.getPuck(p=new Point((getWidth()/2)*, (3**getHeight*()/4)), -);	se objeto cruzou posição realiza a busca.
Test Case 1	13	- [*Exception* e];	situação tratada dispara próximo método.
Test Case 1	14	- *addMovablePiece(sp=puck, -)*;	adiciona novo objeto móvel.
Test Case 1	15	- *new ScoreBoard(new Rectangle(new Point(0,0)*, *new Size*(0,0)));	objeto criado na posição determinada.
Test Case 1	16	- sb.*resetScore*(-);	valor da variável resetada.
Test Case 1	17	- *addStationaryPiece*(sp=sb, -);	adiciona novo objeto estacionário.
Test Case 1	18	- *new Ceiling(new Rectangle(new Point(-5,-5)*, *new Size(getWidth() 10,5)),true)*;	objeto criado na posição determinada.
Test Case 1	19	- *new Floor(new Rectangle(new Point(-5,getHeight())*, *new Size(getWidth() 10,5)),true)*;	adiciona novo objeto estacionário no teto.
Test Case 1	20	- *addStationaryPiece(sp=floor, -)*;	objeto criado na posição determinada.

Tabela 4.7: Sequências de teste do DA da Figura - 4.11
geradas por *SMartyTesting*

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	21	VP_21:*addStationaryPiece*(sp=*floor*, -) - {; - *new LeftWall*(*new Rectangle*(*new Point*(-5,-5), *new Size*(5,*getHeight*() 10)),*false*){*alternative_OR*}};	adiciona novo objeto estacionário na base.
Test Case 1	22	- {; - *addStationaryPiece(sp=leftwall, -)*};	{. objeto criado na posição determinada}.
Test Case 1	23	VP_23:*addStationaryPiece(sp=leftwall, -)* - {; - *new RightWall(new Rectangle(new Point(getWidth(),-5),* *new Size(5,getHeight()* 10)),*false*){*alternative_OR*}};	{. adiciona novo objeto estacionário parede esquerda}.
Test Case 1	24	VP_23:*addStationaryPiece*(sp=*leftwall*, -) - {; - *new RightWall(new Rectangle(new Point(getWidth(),-5),* *new Size(5,getHeight() 10)),false){alternative_OR}*};	{. objeto criado na posição determinada}.
Test Case 1	25	- {; - *addStationaryPiece(sp=rightwall, -)*};	{. adiciona novo objeto estacionário parede direita}.
Test Case 1	26	- *addMovablePiece(sp=topPaddle, -)*;	adiciona novo objeto móvel ao *paddle* posição *top*.

Tabela 4.7: Sequências de teste do DA da Figura - 4.11
geradas por *SMartyTesting*

Sequência de Teste	**Passo**	**Ação/Descrição**	**Resultado Esperado**
Test Case 1	27	- *addMovablePiece(sp=bottomPaddle, -)*;	adiciona novo objeto móvel ao *paddle* posição *bottom*.
Test Case 1	28	*false* - *return false*;	caso retorne *false* para ações encerra sessão.

4.5 Análise e Interpretação

4.5.1 Quantidade de Sequências de Teste Geradas (CT.1)

Na Tabela - 4.8 são apresentados os números da geração de sequências de teste de abordagem e também a soma de todas as sequências de teste geradas conforme o diagrama usado.

Tabela 4.8: Quantidade de sequências de teste geradas por abordagem.

Diagrama	SMartyTesting (DS)	SPLiT-MBt (DA)
Play Selected Game	6	5
Save Game	12	6
Pong Moves	28	3
Total de sequências de teste	**46**	**14**

Com base nesses dados, pode-se identificar que quando utilizados DS a quantidade de sequências de teste tende a ser consideravelmente maior. Entende-se, dessa forma, que isso possa ser determinado pelo nível de abstração do diagrama: quanto mais abstrato, menor a quantidade de casos de teste.

Se for considerado o nível de abstração de cada diagrama, com esses dados é possível fornecer evidências iniciais de que diagramas de sequência possuem potencial para a geração de maior quantidade de sequências de teste. Portanto, pode-se aceitar a hipótese $H_{CT.1}$, o que significa que há evidência de que *SMartyTesting* é viável para geração de um número maior de sequências de teste utilizando DSs.

4.5.2 Diferença entre as Sequências Geradas (CT.2)

Ao se observar os artefatos de entrada, em que suas semelhanças se fazem por equivalência, identifica-se que existe diferença entre as sequências geradas, isso já mencionado anteriormente, e se deve ao fato de nível de abstração.

Embora DS e DA sejam diagramas de comunicação, existe aplicabilidade diferenciada para cada um deles e, por DS estar mais próximo do código fonte, é de se esperar que as sequências geradas diferem de um modelo alto nível, porém, em determinados modelos as sequências quase se equivalem. Entende-se, portanto, que isso depende do nível de detalhamento que o engenheiro de software aplica ao modelo DS.

4.5.3 Complexidade na Geração de Sequências de Teste (CT.3)

Para determinar a complexidade da geração foi utilizada a métrica de complexidade ciclomática de McCabe (1976).

Para essa viabilidade foram encontrados mais estudos que utilizam e fazem análise de complexidade ciclomática, (Gill e Kemerer, 1991; Jay et al., 2009; Shepperd, 1988), porém, todos fazem utilização do mesmo conceito primário abordado por McCabe (1976). Por isso optou-se por seguir os conceitos primários para a utilização da complexidade ciclomática.

Tal métrica serve para mensurar a complexidade de um determinado módulo (uma classe, um método, uma função etc.), a partir da contagem do número de caminhos independentes que o modelo pode executar até o seu fim. Um caminho independente é aquele que apresenta pelo menos uma nova condição (possibilidade de desvio de fluxo) ou um novo conjunto de comandos a serem executados.

Tendo um grafo de fluxo, tem-se três fórmulas equivalentes para se mensurar a complexidade ciclomática McCabe (1976):

1. V(G) = R, em que R é o número de regiões do grafo de fluxo;
2. V(G) = E – N + 2, em que E é o número de arestas (setas) e N é o número de nós do grafo G;
3. V(G) = P + 1, em que P é o número de nós-predicados contidos no grafo G (só funciona se os nós-predicado tiverem no máximo duas arestas saindo.)

A Tabela - 4.10 apresenta os dados utilizando a fórmula número 2.

Tabela 4.9: Parâmetros aceitáveis para a complexidade ciclomática

Complexidade	Avaliação
1-10	Método simples. **Baixo risco**
11-20	Método razoavelmente complexo. **Moderado risco**
21-50	Método muito complexo. **Elevado risco**
51-N	Método de **altíssimo risco** e bastante instável

Após a análise dos grafos gerados por ambos os DA utilizados na geração de sequências de teste obteve-se a Tabela - 4.10. Nela pode-se perceber que a complexidade diferentemente da quantidade de sequências geradas, quase se equivale baseado na Tabela - 4.9. Outro fator é a complexidade dos artefatos, que se manteve como simples e de baixo risco, o que é muito bom.

Tabela 4.10: complexidade ciclomática apresentada pelas abordagens

Diagrama	***SMartyTesting* (DS)**	**SPLiT-MBt (DA)**
Play Selected Game	3	4
Save Game	3	2
Pong Moves	1	1
Total	**7**	**7**

Pode-se concluir que, embora diagramas DS possuam maior nível de detalhes, se usados por *SMartyTesting* possuem complexidade menor se comparados com DA utilizado por SPLiT-MBt, que possui nível maior de abstração.

4.5.4 Esforço na Utilização das Abordagens (CT.4)

Outro critério de análise é o de esforço na utilização das abordagens. Para isso, levou-se em consideração o tempo utilizado para a realização do ciclo de geração completo, da entrada do artefato inicial DS para *SMartyTesting* e DA para SPLiT-MBt até a geração de sequência de teste.

Considera-se que para ser observado o esforço, parte-se de que os diagramas DS e DA, ambos artefatos de entrada para cada uma das abordagens observadas, já estejam modelados.

Então, é dado início ao processo de geração de sequência para cada abordagem, *SMartyTesting* com diagramas de sequência e SPLiT-MBt com o diagrama de atividades. Por não possuir ainda todo seu processo automatizado, *SMartyTesting* se torna uma abordagem que demanda esforço maior em sua utilização e compreensão, vide o mapeamento manual que deve ser feito de DS para DA.

Embora *SMartyTesting* utilize SPLiT-MBt, *SMartyTesting* demanda mais esforço em sua primeira etapa, para a realização da conversão manual de DS para DA, o que pode ser amenizado com implementações futuras.

Em trabalhos futuros a recomendação é que seja feita conversão automática direta de DS para MEF, garantindo assim maior performance e menor redundância de dados.

4.6 Ameaças à Validade

Nesta seção são tratadas as ameaças e ações tomadas para amenizar alguns dos problemas encontrados durante a execução deste estudo, incluindo ameaças à validação interna, externa, de *constructo* e de conclusão.

4.6.1 Validade Interna

Tem a intenção de verificar se a comparação das gerações de sequências de teste realmente obtiveram os resultados esperados, dado que a seguinte ameaça foi identificada:

Método de conversão manual de DS para DA: por ser o principal item de diferenciação entre *SMartyTesting* e SPLiT-MBt, a conversão do artefato DS para DA feita de forma manual, pode gerar um viés em relação à aplicação das regras de mapeamento, devido ao processo de conversão não ser automatizado. Podendo induzir a erros de modelagem durante a criação do diagrama de atividades. Para contornar esse viés, durante a comparação de viabilidade, foram seguidos sistematicamente todos os itens da Tabela - 3.1 do mapeamento proposto por Garousi et al. (2005).

4.6.2 Validade Externa

Autores especialistas: a interpretação das funcionalidades no momento da modelagem pode influenciar no nível de complexidade do diagrama de DS e DA. Para poder realizar o estudo de viabilidade amenizando esse viés foram selecionados dois autores que são doutores além de já dominarem o assunto abordado neste trabalho.

4.6.3 Validade de *Constructo*

Pelo fato de a conversão ser feita de forma manual, mesmo sendo realizada com base em um conjunto de regras de mapeamento (Garousi et al., 2005) pode gerar questionamentos sobre as funcionalidades representadas.

Para amenizar esse viés, foi analisado se o mapeamento já havia sido validado, nesse caso por Shafique e Labiche (2010), que apresentam uma abordagem de geração de casos de teste a partir de diagramas de sequência.

4.6.4 Validade de Conclusão

Este trabalho é um estudo inicial de viabilidade, não sendo possível generalizar os resultados.

4.7 Considerações Finais

Os resultados obtidos por comparações e observações de utilização das duas abordagens, permitiram avaliar a viabilidade da abordagem *SMartyTesting* além de propor possíveis melhorias para a abordagem SPLiT-MBt.

Além de viabilizar estudos futuros para implementação de ferramentas de apoio, que façam utilização dos conceitos da abordagem *SMartyTesting*, no próximo tópico são apresentadas todas as melhorias identificadas e lições aprendidas com a pesquisa.

5

Melhorias Identificadas e Lições Aprendidas

5.1 Considerações Iniciais

Com base no estudo realizado no tópico 4 e os seus resultados, foram identificadas algumas melhorias e lições aprendidas sobre ambas as abordagens *SMartyTesting* e SPLiT-MBt.

As melhorias são fundamentais para guiar trabalhos futuros relacionados à esta dissertação. Lições aprendidas são discutidas com o intuito de transferir o máximo possível de conhecimento adquirido com este trabalho.

5.2 Melhorias Identificadas

As melhorias identificadas estão voltadas tanto para *SMartyTesting* como para SPLiT-MBt. Tais sugestões foram analisadas e são discutidas a seguir.

5.2.1 Automatização do Processo de Geração de Sequências de Teste

Na Seção 4.5 foi possível a análise da viabilidade de utilização de *SMartyTesting*. Embora evidenciado que exista viabilidade na sua utilização, um fator observado é o nível de esforço que a abordagem necessita para a geração de sequências de teste, o que não ocorre em SPLiT-MBt uma vez que todo o processo é automatizado.

Nesse caso a melhoria pontual seria a implementação completa do processo de teste usando *SMartyTesting*, partindo do modelo de entrada (DS), convertendo o DS para MEF e, então, gerando as sequências de teste, utilizando SPLiT-MBt como base ou uma nova implementação com uma estrutura própria.

Partindo de uma implementação nova, pode-se garantir um código mais ajustado ao propósito da abordagem, sem dependências de códigos não utilizados ou sistemas legados sem suporte. Aqui três pontos devem ser tratados:

- a conversão de DS para MEF;
- a implementação ou adaptação de um dos algoritmos apresentados na Seção 2.5.1; e
- a geração das sequências de teste.

Caso seja considerada a adaptação da SPLiT-MBt nessa nova implementação, dois itens devem ser tratados:

- suporte a *fork node* e *join node*, e
- suporte a subsistema com mais de um início ou fim.

5.2.2 Utilização da Conversão de DS para MEF em *SMartyTesting*

Para a realização da automatização de todo o ciclo de *SMartyTesting*, o ideal é que algumas mudanças com relação ao seu processo de conversão sejam realizadas. Embora a utilização de SPLiT-MBt por *SMartyTesting* seja viável, estudos preliminares contidos no MSL (Apêndice A) demonstram que a diminuição de etapas torna o processo de teste mais objetivo e confiável em nível de falhas ocasionadas por erros básicos.

Assim como SPLiT-MBt faz o uso de MEFs, a recomendação seria que *SMartyTesting* também fizesse uso do mesmo artefato, pois MEF está entre os modelos formais mais utilizados em TBM de LPS. Além disso, MEF é uma boa alternativa para projetar componentes de teste de software, pois pode ser aplicada a qualquer modelo de especificação que descreva um número finito de etapas (Costa, 2016).

Outro ponto de análise é que para gerar uma quantidade considerável de sequências de teste sobre um modelo, o ideal é a utilização de métodos de geração de sequências de teste. Para que isso seja possível, a utilização de MEF é a solução mais adequada. Porém, assim como Costa (2016), deve-se também aprimorar uma extensão para que MEF possa representar informações de variabilidade, visto que MEF foi projetada essencialmente para testar softwares com base no paradigma de desenvolvimento de sistema único.

5.2.3 Suporte para Programação Concorrente e Sub-processo em SPLiT-MBt

Foram encontradas duas situações em que SPLiT-MBt se torna limitante, a primeira delas é em relação às chamadas de fluxo concorrente, que utilizam um *Fork Node* diferente de um *Node Decision* que só possui um caminho. Representando *if* ou *else*, o *Fork Node* representa uma ação disparada que pode ser direcionada a dois estados ao mesmo tempo, como exemplificado na Figura - 3.6.

Um exemplo seria a conversão do diagrama da Figura - 5.1, em que as mensagens seis e sete são assíncronas, disparando o processo e dando continuidade ao fluxo principal.

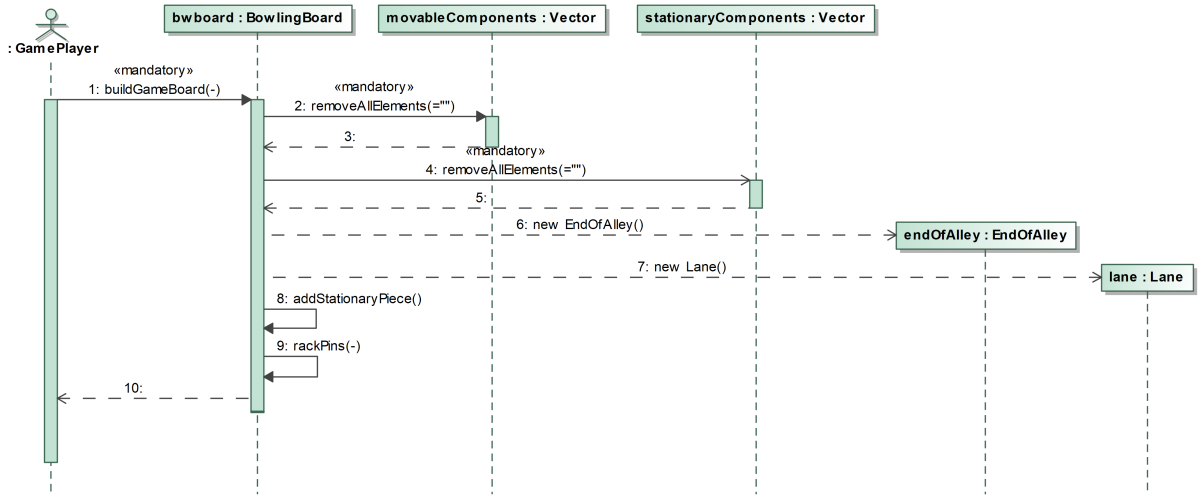


Figura 5.1: Exemplo de DS com mensagens representando programação concorrente em uma função de *login*

No DA convertido, essas mensagens seis e sete, iriam partir de um *Fork Node* e, após isso, poderiam ser representadas como um subprocesso ou um *End Node* (Figura - 5.2).

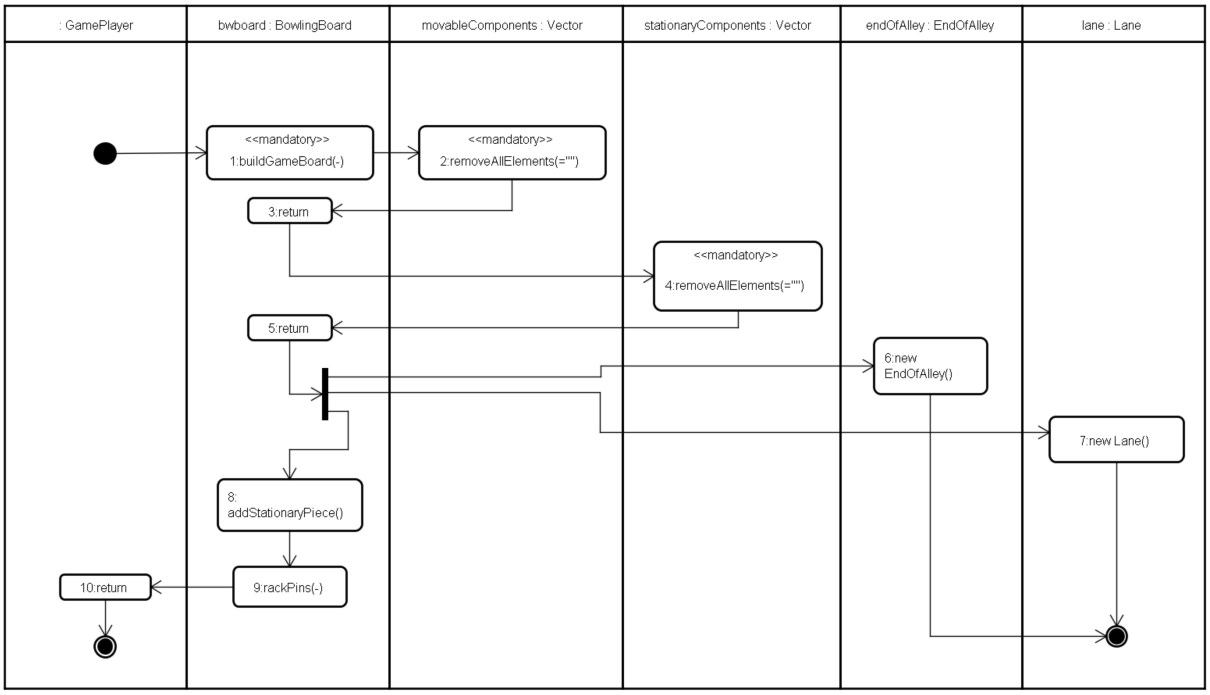


Figura 5.2: DA com mensagens representando programação concorrente em uma função de *login*

Como o DA não possui suporte ao *Fork Node*, consequentemente, não possui suporte também ao *Join Node*, que faz a junção de fluxo vindo de objetos diferentes. Além disso, fica visível a necessidade de suporte também a subsistemas em que um modelo possui mais de um elemento de *Initial Node* ou *End Node*. Quando se trabalha com subsistemas pode haver a finalização de um subprocesso para dar continuidade no processo principal, exemplificado na Figura - 3.6.

5.2.4 Suporte para Outro Métodos de Geração de Sequências de Teste

Na Seção 2.5.1 foram abordados os métodos de geração de sequências de teste. SPLiT-MBt faz utilização do método HSI, em que Costa (2016) justifica que tal método é o menos restritivo com relação às propriedades na utilização em LPS.

Pinheiro e Simão (2012), no entanto citam que se deve levar em conta a relação custo-benefício de cada método, pois o foco principal consiste em promover a detecção do maior número possível de defeitos existentes em uma implementação, levando em conta o tamanho do conjunto gerado, para que esse fato não inviabilize a sua aplicação prática.

Pinheiro e Simão (2012) realizaram um trabalho de geração de sequências com os métodos de geração mais utilizados, como visto na Seção 2.5.2.

Embora Costa (2016) tenha feito ensaios demonstrando que o HSI produz melhores resultados, a sugestão seria a implementação dos demais métodos como opção de geração de sequências de teste para SPLiT-MBt, deixando a cargo do engenheiro de software selecionar qual o método de geração é mais efetivo ao objetivo almejado.

5.3 Lições Aprendidas

5.3.1 Compreensão de SPLiT-MBt

A abordagem SPLiT-MBt contribuiu muito para o entendimento da geração de sequências de teste e o entendimento sobre a estrutura necessária para a geração das sequências.

Adaptação de métodos de geração de sequência a partir de modelos formais como MEF foram itens fundamentais para a visualização de trabalhos futuros relacionados à geração de sequências de teste fazendo uso de outros métodos de geração.

Embora a ferramenta seja caixa preta, uma boa documentação com todos os parâmetros bem explicados se faz necessária, ficando o aprendizado de todo o processo, desde o artefato de entrada até as sequências de teste geradas, explicado na Seção 2.5. Dessa forma, a lição aprendida é a necessidade de disponibilidade das implementações das abordagens e dos dados obtidos para que a pesquisa possa ser reproduzida.

5.3.2 Conversão Manual de DS para DA

A conversão manual proporcionou aprendizado sobre os metamodelos de DS e DA *Control Flow Analysis of UML 2.0 Sequence Diagrams* possui regras claras, definidas e baseadas no metamodelo de cada diagrama. Isso contribuiu diretamente para o entendimento das conversões dos elementos, embora precise de aprimoramento (extensão) com relação a elementos de variabilidade.

Por mais que não influencie diretamente neste trabalho, o processo de automatização da abordagem *SMartyTesting* contribui para maior confiabilidade e consistência das gerações de sequências de teste.

5.3.3 ATLs para Conversão

Por causa do grande aumento da complexidade no desenvolvimento de software nos últimos anos, academia e indústria criaram uma solução racional na engenharia de

software, chamada de Engenharia Dirigida por Modelos, que busca apoiar o gerenciamento de tal complexidade. Uma operação importante na Engenharia Dirigida por Modelos é a transformação de modelos, que consiste em um processo automatizado de conversão de um modelo origem para um modelo de destino (Allilaire et al., 2006).

Durante o processo de pesquisa foram encontradas duas ATLs de conversão de DS para *Statechart* e outra de DS para MEF, em que a primeira *UML Sequence Diagrams to Statechart Diagram* [1] possui documentação de utilização extremamente confusa e de difícil configuração e, para tanto, foram realizados testes de conversão de diagramas de exemplos que estavam disponíveis no pacote, mas não foi obtido o resultado esperado. A segunda, também *open-source*, é chamada de *Convert UML Sequence Diagram to UML State Machine* [2] que possui documentação um pouco mais detalhada em comparação a anterior. Foram realizados testes iniciais com os exemplos contidos no pacote onde houve êxito de conversão com alguns diagramas (Hennicker e Knapp, 2007).

Ambas as ATLs dispõem de pacotes que utilizam uma linguagem de conversão *Query/View/Transformation* (QVT). QVT é uma linguagem utilizada para transformar (meta) modelos e usa OCL (*Object constraint language*) estendida em combinação com uma linguagem específica de domínio: Relações, Core ou Operacional. Este último é uma linguagem imperativa, enquanto as outras duas são declarativas e ambas dependem do IDE Eclipse.

Convert UML Sequence Diagram to UML State Machine possui suporte a arquivos com extensão ".UML", sendo assim, a quantidade de ferramentas que poderiam auxiliar na concepção de diagramas que pudessem ser convertidos ficou limitada, por causa de outras existentes utilizarem formatos próprios de saída. Em princípio, não foi realizada busca por conversões de formato para obter abrangência maior, em vez disso, procurou-se uma ferramenta que fosse robusta e pudesse entregar o proposto. Sendo assim, optou-se por realizar testes como a *UML Designer*[3] embora, sem sucesso também.

Por fim, a experiência com ATLs de conversão de DS para MEF não foi satisfatória por causa das complexidades de utilização, curva de aprendizado e pouca documentação, o que reforça a lição aprendida anteriormente.

5.3.4 Uso de Ferramentas

O uso de ferramentas se faz necessário para a automatização dos processos ou até mesmo para apoio à criação de diagramas. Houve dificuldades em relação à curva

[1]Disponível em:http://www.st.ewi.tudelft.nl/basgraaf/ - Acessado em 20/03/2019
[2]Disponível em: https://github.com/slashburn/mdd-project - Acessado em 10/04/2019
[3]Disponível em: http://www.umldesigner.org/download/ - Acessado em 10/04/2019

de aprendizagem e documentação das ferramentas pesquisadas para validação, como, por exemplo, JPlavisFSM, *Control Flow Analysis of UML 2.0 Sequence Diagrams* e SPLiT-MBt.

Outro fator de aprendizado são os níveis de configuração e disponibilização das ferramentas, *links* quebrados, informações falhas sobre locais para *download*, principalmente qual a versão da ferramenta utilizada. Esses são os exemplos de aprendizado obtido com relação às ferramentas utilizadas.

5.3.5 Teste Baseado em Modelos de LPS

O processo TBM para LPS que foi vivenciado demonstra de fato o que é apresentado no Apêndice A, processos similares, assim como as dificuldades e desafios.

TBM se apresenta como uma forma bem estruturada de técnica de teste voltada para utilização em LPS, por ser adaptável e flexível em seus processos.

5.4 Considerações Finais

O trabalho de viabilidade de *SMartyTesting* permitiu a validação de SPLiT-MBt em pontos importantes, essa validação contribuiu direta e indiretamente para a incorporação do conceito de SPLiT-MBt em *SMartyTesting*.

Nas melhorias identificadas, além do processo de automatização de *SMartyTesting* e a ampliação do suporte de SPLiT-MBt a novos processos que fazem parte do metamodelo de atividades, permitiram muitas lições aprendidas aos processos apresentados nesse trabalho.

6

Conclusão

6.1 Contribuições

O resultado principal desta pesquisa foi a especificação e a evidência preliminar de viabilidade da abordagem *SMartyTesting* que permite a geração de sequências de teste para LPS considerando variabilidade modelada com suporte da abordagem *SMarty*. *SMartyTesting* tem como artefato de entrada um diagrama de sequência (DS) que possui estrutura definida. Com o objetivo principal de apoiar a cultura de TBM em fases iniciais de LPS, entende-se como uma contribuição a possibilidade de melhorar a qualidade de modelos de LPS.

Sendo assim, após a especificação da abordagem, foi realizado um estudo de viabilidade com alguns critérios de comparação para que fosse possível responder a seguinte questão de pesquisa: **"Diagramas de sequência podem gerar mais sequências de teste do que diagramas de atividades?"**

A geração de sequências de teste a partir de DS se mostrou viável por apresentar baixa complexidade ciclomática (CT.3) em suas conversões, demonstrando potencial para a sua utilização, o que se alinha aos resultados do MSL realizado.

A quantidade de sequências de teste geradas é maior, conforme esperado, devido a quantidades de atividades geradas a partir do diagrama de sequência (CT.1). Isso impacta diretamente na diferenciação das sequências de teste geradas (CT.2).

Devido a conversão manual de DS para DA, o esforço de utilização (CT.4) se torna maior com *SMartyTesting*, mas isso já se era previsto por ser uma abordagem semi-automatizada.

Além da contribuição principal, considera-se muito importante também a análise da abordagem SPLiT-MBt e de sua ferramenta de apoio. Assim, a Seção 5.2.3 mostra como melhorias podem ser realizadas para as abordagens *SMartyTesting* e SPLiT-MBt contribuindo para a evolução dessas.

O MSL conduzido para a fundamentação desta pesquisa, além de contribuir com dados que apresentam o cenário de TBM para LPS, contribuiu também para a criação de um mapa de direcionamento de estudos relacionados por temas. Tal mapa permite entender como estudos de TBM para LPS têm sido realizados e como pesquisadores e profissionais podem utilizá-los ao tratar deste assunto em pesquisas e práticas na indústria.

Em síntese, buscou-se com o desenvolvimento deste projeto contribuir para a comunidade acadêmica e industrial de LPS com uma abordagem de geração de sequências de teste, que considerem variabilidade na engenharia de domínio e para a engenharia de aplicação de LPS.

6.2 Limitações

Foram encontradas duas situações em que SPLiT-MBt se torna limitante, a primeira delas é em relação a chamadas de fluxo concorrente, comentado na Seção 5.2.3, como não possui esse suporte ao *Fork Node*, consequentemente, não possui suporte também ao *Join Node*. Além disso, é visível a necessidade de suporte também a subsistemas em que um modelo possui mais de um elemento de *Initial Node* ou *End Node* estágios de início e fim, dado que quando se trabalha com subsistemas pode haver a finalização de um subprocesso para dar continuidade no processo principal exemplificado na Seção 5.2.3.

Baseado nesses fatores, pode-se considerar que SPLiT-MBt não tem suporte à mensagens assíncronas, limitando-se apenas a mensagens síncronas em que toda mensagem deve possuir um retorno, não conformante com 100% do metamodelo da UML.

Quanto às limitações de *SMartyTesting*, é uma abordagem que não converte DS para DA ou para MEF automaticamente, isso é um dos itens que estão considerados como trabalho futuro para a abordagem. Outro item está relacionado ao estudo de viabilidade conduzido com uma amostra pequena de LPS, DS e DA, o que não permite generalizar os resultados obtidos.

6.3 Trabalhos Futuros

Com base na experiência de criação e especificação da abordagem *SMartyTesting*, no processo de comparação e nas observações sobre sua utilização e, visando a continuidade do trabalho que possui um grande potencial, são apresentadas a seguir as principais perspectivas em relação a trabalhos futuros que possam decorrer deste estudo preliminar.

Implementação Total para *SMartyTesting*: após a análise dos dados comparativos e observações realizadas com a abordagem para se ter a certeza que era possível a geração de sequências de teste a partir de DSs, demonstra ser viável a implementação total da abordagem *SMartyTesting*, isso não impede a continuidade da utilização de SPLiT-MBt, que também pode ser incorporada ao processo de *SMartyTesting*.

Diminuição das Etapas do Processo: no MSL foi apontado que diversos trabalhos fazem uso de conversão de DS para MEF, e Pinheiro e Simão (2012) apresentam na Seção 2.5.2 motivos para a utilização de MEF. Sendo assim, seria interessante em um trabalho futuro analisar a viabilidade do processo de geração de sequências de teste a partir de um DS com conversão direta para MEF e, após essa transformação, utilizar outros métodos de geração de sequências de teste.

Incorporar *SMartyTesting* à ferramenta *SMartyModeling*: em paralelo a este trabalho está sendo desenvolvida outra pesquisa sobre uma ferramenta para modelagem de LPS considerando *SMarty*, em que os diagramas suportados pela versão 5.1 são: sequência, casos de uso, atividades, componentes e classes. Com isso, quando o engenheiro de software fizer a modelagem de uma LPS já tem a possibilidade de gerar das sequências de teste.

Aplicar métricas de teste aos modelos *SMarty*: considerar as métricas de teste no processo de teste e comparar as sequências de teste ou casos de teste em modelos *SMarty*.

Estender o perfil de teste da OMG: estender o perfil de teste da OMG [1] (Bagnato et al., 2013) para TBM de LPS, considerando o processo da *SMartyTesting*.

[1] Disponível em: `https://www.omg.org/spec/UTP` - Acessado em 20/10/2019

REFERÊNCIAS

Al-Hajjaji, M.; Krieter, S.; Thüm, T.; Lochau, M.; Saake, G. Incling: efficient product-line testing using incremental pairwise sampling. In: *ACM SIGPLAN Notices*, ACM, 2016, p. 144–155.

Ali, S.; Yue, T.; Briand, L.; Walawege, S. A product line modeling and configuration methodology to support model-based testing: an industrial case study. In: *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2012, p. 726–742.

Allilaire, F.; Bézivin, J.; Jouault, F.; Kurtev, I. Atl-eclipse support for model transformation. In: *Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France*, Citeseer, 2006.

Alves, V.; Schneider, D.; Becker, M.; Iese, F.; Platz, F.; Bencomo, N.; Grace, P. Comparitive study of variability management in software product lines and runtime adaptable systems. In: *International Workshop on Variability Modelling of Software-Intensive Systems*, 2009, p. 9–17.

Apel, S.; Batory, D.; Kästner, C.; Saake, G. *Feature-oriented software product lines*. Springer, 2013a.

Apel, S.; Batory, D.; Kstner, C.; Saake, G. *Feature-oriented software product lines: Concepts and implementation*. Springer Publishing Company, Incorporated, 2013b.

Bagnato, A.; Sadovykh, A.; Brosse, E.; Vos, T. E. J. The omg uml testing profile in use–an industrial case study for the future internet testing. In: *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, p. 457–460.

Basili, V. R. *Software modeling and measurement: the goal/question/metric paradigm*. Relatório Técnico, 1992.

Beohar, H.; Mousavi, M. R. Input-output conformance testing based on featured transition systems. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ACM, 2014a, p. 1272–1278.

Beohar, H.; Mousavi, M. R. Spinal test suites for software product lines. *arXiv preprint arXiv:1403.7260*, 2014b.

Beohar, H.; Varshosaz, M.; Mousavi, M. R. Basic behavioral models for software product lines: Expressiveness and testing pre-orders. *Science of Computer Programming*, v. 123, p. 42–60, 2016.

Bera, M. H.; OliveiraJr, E.; Colanzi, T. E. Evidence-based smarty support for variability identification and representation in component models. In: *Proceedings of the 17th International Conference on Enterprise Information Systems-Volume 2*, SCITEPRESS-Science and Technology Publications, Lda, 2015a, p. 295–302.

Bera, M. H. G. *Smartycomponents: um processo para especificação de arquiteturas de linha de produto de software baseadas em uml.* Dissertação de Mestrado, Universidade Estadual de Maringá, Departamento de Informática, Programa de Pós Graduação em Ciência da Computação, 2015.

Bera, M. H. G.; OliveiraJr, E.; Colanzi, T. E. Evidence-based smarty support for variability identification and representation in component models. In: *Proceedings of the 17th International Conference on Enterprise Information Systems - Volume 2: ICEIS,*, 2015b, p. 295–302.

Bernardino, M.; Rodrigues, E. M.; Zorzo, A. F.; Marchezan, L. Systematic mapping study on mbt: tools and models. *IET Software*, v. 11, n. 4, p. 141–155, 2017.

Booch, G.; Rumbaugh, J.; Jacobson, I. *Uml: guia do usuário.* Elsevier Brasil, 2006.

Bringmann, E.; Krämer, A. Model-based testing of automotive systems. In: *2008 1st international conference on software testing, verification, and validation*, IEEE, 2008, p. 485–493.

Cai, X.; Zeng, H. Model-based test generation for software product line. In: *Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on*, IEEE, 2013, p. 347–351.

Chen, L.; Ali Babar, M.; Ali, N. Variability management in software product lines: A systematic review. In: *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, Pittsburgh, PA, USA: Carnegie Mellon University, 2009, p. 81–90 (*SPLC '09*,).
Disponível em `http://dl.acm.org/citation.cfm?id=1753235.1753247`

Chow, T. S. Testing software design modeled by finite-state machines. *IEEE transactions on software engineering*, , n. 3, p. 178–187, 1978.

Cichos, H.; Oster, S.; Lochau, M.; Schürr, A. Model-based coverage-driven test suite generation for software product lines. In: *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2011, p. 425–439.

Clements, P.; Northrop, L. *Software product lines*. Addison-Wesley„ 2002.

Costa, L. T. *Split-mbt: A model-based testing method for software product lines*. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio Grande do Sul, 2016.

Crespo, A. N.; Silva, O. J.; Borges, C. A.; Salviano, C. F.; Argollo, M.; Jino, M. Uma metodologia para teste de software no contexto da melhoria de processo. *Simpósio Brasileiro de Qualidade de Software*, p. 271–285, 2004.

Da Silva Simao, A.; Ambrósio, A. M.; Fabbri, S.; do Amaral, A.; Martins, E.; Maldonado, J. C. Plavis/fsm: an environment to integrate fsm-based testing tools. In: *Tool Session of XIX Brazilian Symposium on Software Engineering*, Citeseer, 2008, p. 1–6.

Damiani, F.; Faitelson, D.; Gladisch, C.; Tyszberowicz, S. A novel model-based testing approach for software product lines. *Software & Systems Modeling*, v. 16, n. 4, p. 1223–1251, 2017.

Damiani, F.; Gladisch, C.; Tyszberowicz, S. Refinement-based testing of delta-oriented product lines. In: *Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*, ACM, 2013, p. 135–140.

Delamaro, M.; Jino, M.; Maldonado, J. *Introdução ao teste de software*. Elsevier Brasil, 2017.

Devroey, X. Behavioural model based testing of software product lines: Research abstract. In: *18th International Software Product Lines Conference (SPLC'14)*, ACM Press, 2014.

Devroey, X.; Cordy, M.; Perrouin, G.; Kang, E.-Y.; Schobbens, P.-Y.; Heymans, P.; Legay, A.; Baudry, B. A vision for behavioural model-driven validation of software product lines. In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Springer, 2012, p. 208–222.

Devroey, X.; Perrouin, G.; Legay, A.; Cordy, M.; Schobbens, P.-Y.; Heymans, P. Coverage criteria for behavioural testing of software product lines. In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Springer, 2014a, p. 336–350.

Devroey, X.; Perrouin, G.; Schobbens, P.-Y. Abstract test case generation for behavioural testing of software product lines. In: *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*, ACM, 2014b, p. 86–93.

Devroey, X.; Perrouin, G.; Schobbens, P.-Y.; Heymans, P. Vibes, transition system mutation made easy. In: *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, IEEE Press, 2015, p. 817–818.

Do Carmo Machado, I.; Mcgregor, J. D.; Cavalcanti, Y. C.; De Almeida, E. S. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, v. 56, n. 10, p. 1183–1199, 2014.

Dukaczewski, M.; Schaefer, I.; Lachmann, R.; Lochau, M. Requirements-based delta-oriented spl testing. In: *Product Line Approaches in Software Engineering (PLEASE), 2013 4th International Workshop on*, IEEE, 2013, p. 49–52.

Engström, E.; Runeson, P. Software product line testing–a systematic mapping study. *Information and Software Technology*, v. 53, n. 1, p. 2–13, 2011.

Escalona, M.; Gutierrez, J. J.; Mejías, M.; Aragón, G.; Ramos, I.; Torres, J.; Domínguez, F. An overview on test generation from functional requirements. *Journal of Systems and Software*, v. 84, n. 8, p. 1379–1393, 2011.

Farrag, M. *Colored model based testing for software product lines (cmbt-swpl).* Tese de Doutoramento, Technical University of Ilmenau, 2010.

Feldt, R.; Zimmermann, T.; Bergersen, G. R.; Falessi, D.; Jedlitschka, A.; Juristo, N.; Münch, J.; Oivo, M.; Runeson, P.; Shepperd, M.; Sjøberg, D. I. K.; Turhan, B. Four commentaries on the use of students and professionals in empirical software engineering experiments. *Empirical Software Engineering*, v. 23, n. 6, p. 3801–3820, 2018.
Disponível em `https://doi.org/10.1007/s10664-018-9655-0`

Feng, Y.; Liu, X.; Kerridge, J. A product line based aspect-oriented generative unit testing approach to building quality components. In: *Annual International Computer Software and Applications Conference*, 2007, p. 403–408.

Fiori, D. R.; Gimenes, I. M.; Maldonado, J. C.; OliveiraJr, E. Variability management in software product line activity diagrams. In: *DMS*, 2012, p. 89–94.

Fragal, V. H.; Simao, A.; Endo, A. T.; Mousavi, M. R. Reducing the concretization effort in fsm-based testing of software product lines. In: *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE, 2017, p. 329–336.

Fujiwara, S.; Bochmann, G. v.; Khendek, F.; Amalou, M.; Ghedamsi, A. Test selection based on finite state models. *IEEE Transactions on software engineering*, v. 17, n. 6, p. 591–603, 1991.

García Gutiérrez, B.; Garcia Carmona, R.; Navas Baltasar, A.; Parada Gélvez, H. A.; Cuadrado Latasa, F.; Dueñas López, J. C. An automated model-based testing approach in software product lines using a variability language. In: *Workshop on Model-Driven Tool and Process Integration*, 2010, p. 1–10.

Garousi, V.; Briand, L. C.; Labiche, Y. Control flow analysis of uml 2.0 sequence diagrams. In: *European Conference on Model Driven Architecture-Foundations and Applications*, Springer, 2005, p. 160–174.

Gebizli, C. S.; Sözer, H. Model-based software product line testing by coupling feature models with hierarchical markov chain usage models. In: *Software Quality, Reliability and Security Companion (QRS-C), 2016 IEEE International Conference on*, IEEE, 2016, p. 278–283.

Gebizli, C. S.; Sözer, H.; Ercan, A. Ö. Successive refinement of models for model-based testing to increase system test effectiveness. In: *Software Testing, Verifi-*

cation and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on, IEEE, 2016, p. 263–268.

Geraldi, R. T.; OliveiraJr, E.; Conte, T.; Steinmacher, I. Checklist-based inspection of smarty variability models - proposal and empirical feasibility study. In: *Proceedings of the 17th International Conference on Enterprise Information Systems - Volume 2: ICEIS,*, 2015, p. 268–276.

Gill, G. K.; Kemerer, C. F. Cyclomatic complexity density and software maintenance productivity. *IEEE transactions on software engineering*, v. 17, n. 12, p. 1284, 1991.

Gomaa, H. Designing software product lines with uml 2.0: From use cases to pattern-based software architectures. In: *Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components*, Springer-Verlag, 2006, p. 440–440.

Gonenc, G. A method for the design of fault detection experiments. *IEEE transactions on Computers*, v. 100, n. 6, p. 551–558, 1970.

Hasling, B.; Goetz, H.; Beetz, K. Model based testing of system requirements using uml use case models. In: *Software Testing, Verification, and Validation, 2008 1st international conference on*, IEEE, 2008, p. 367–376.

Henard, C.; Papadakis, M.; Perrouin, G.; Klein, J.; Le Traon, Y. Assessing software product line testing via model-based mutation: An application to similarity testing. In: *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, IEEE, 2013, p. 188–197.

Hennicker, R.; Knapp, A. Activity-driven synthesis of state machines. In: *International Conference on Fundamental Approaches to Software Engineering*, Springer, 2007, p. 87–101.

Hennine, F. Fault detecting experiments for sequential circuits. In: *1964 Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, IEEE, 1964, p. 95–110.

Isa, M. A. B.; Razak, S. B. A.; Jawawi, D. N. B. A.; Fuh, O. L. Model-based testing for software product line: A systematic literature review. *International Journal of Software Engineering and Technology*, v. 2, n. 2, 2017.

Jay, G.; Hale, J. E.; Smith, R. K.; Hale, D. P.; Kraft, N. A.; Ward, C. Cyclomatic complexity and lines of code: Empirical evidence of a stable linear relationship. *JSEA*, v. 2, n. 3, p. 137–143, 2009.

Kakarontzas, G.; Stamelos, I.; Katsaros, P. Product line variability with elastic components and test-driven development. In: *International Conference on Computational Intelligence for Modelling Control Automation*, 2008, p. 146–151.

Keele, S.; et al. Guidelines for performing systematic literature reviews in software engineering. In: *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*, sn, 2007.

Kitchenham, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

Kitchenham, B. A.; Budgen, D.; Brereton, P. *Evidence-based software engineering and systematic reviews*, v. 4. CRC press, 2015.

Knapp, A.; Roggenbach, M.; Schlingloff, B.-H. On the use of test cases in model-based software product line development. In: *Proceedings of the 18th International Software Product Line Conference-Volume 1*, ACM, 2014, p. 247–251.

Lachmann, R.; Beddig, S.; Lity, S.; Schulze, S.; Schaefer, I. Risk-based integration testing of software product lines. In: *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*, ACM, 2017, p. 52–59.

Lackner, H.; Thomas, M.; Wartenberg, F.; Weissleder, S. Model-based test design of product lines: Raising test design to the product line level. In: *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, IEEE, 2014, p. 51–60.

Lamancha, B. P.; Mateo, P. R.; de Guzmán, I. R.; Usaola, M. P.; Velthius, M. P. Automated model-based testing using the uml testing profile and qvt. In: *International Workshop on Model-Driven Engineering, Verification and Validation*, ACM, 2009, p. 6.

Lamancha, B. P.; Usaola, M. P.; Velthius, M. P. A model based testing approach for model-driven development and software product lines. In: *International Conference on Evaluation of Novel Approaches to Software Engineering*, Springer, 2010, p. 193–208.

Lee, J.; Kang, S.; Lee, D. A survey on software product line testing. In: *International Software Product Line Conference*, SPLC '12, New York, NY, USA: ACM, 2012, p. 31–40 (*SPLC '12*,).

Li, Q. A novel likert scale based on fuzzy sets theory. *Expert Systems with Applications*, v. 40, n. 5, p. 1609–1618, 2013.

Linden, F.; Schmid, K.; Rommes, E. The product line engineering approach. *Software Product Lines in Action*, p. 3–20, 2007.

Lity, S.; Lochau, M.; Schaefer, I.; Goltz, U. Delta-oriented model-based spl regression testing. In: *International Workshop on Product LinE Approaches in Software Engineering*, IEEE Press, 2012, p. 53–56.

Lity, S.; Morbach, T.; Thüm, T.; Schaefer, I. Applying incremental model slicing to product-line regression testing. In: *International Conference on Software Reuse*, Springer, 2016, p. 3–19.

Lochau, M.; Kamischke, J. Parameterized preorder relations for model-based testing of software product lines. In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Springer, 2012, p. 223–237.

Lochau, M.; Oster, S.; Goltz, U.; Schürr, A. Model-based pairwise testing for feature interaction coverage in software product line engineering. *Software Quality Journal*, v. 20, n. 3-4, p. 567–604, 2012a.

Lochau, M.; Peldszus, S.; Kowal, M.; Schaefer, I. Model-based testing. In: *Advanced Lectures of the 14th International School on Formal Methods for Executable Software Models - Volume 8483*, New York, NY, USA: Springer-Verlag New York, Inc., 2014, p. 310–342.
Disponível em `http://dx.doi.org/10.1007/978-3-319-07317-0_8`

Lochau, M.; Schaefer, I.; Kamischke, J.; Lity, S. Incremental model-based testing of delta-oriented software product lines. In: *International Conference on Tests and Proofs*, Springer, 2012b, p. 67–82.

Marcolino, A.; OliveiraJr, E.; Gimenes, I. Towards the effectiveness of the smarty approach for variability management at sequence diagram level. In: *Proceedings of the 16th International Conference on Enterprise Information Systems - Volume 2: ICEIS,*, 2014a, p. 249–256.

Marcolino, A.; OliveiraJr, E.; Gimenes, I.; Barbosa, E. F. Empirically based evolution of a variability management approach at uml class level. In: *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014b, p. 354–363.

Marcolino, A.; OliveiraJr, E.; Gimenes, I. M. S.; Maldonado, J. C. Towards the effectiveness of a variability management approach at use case level. In: *The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27-29, 2013.*, 2013, p. 214–219.

Marcolino, A. S.; OliveiraJr, E.; Gimenes, I. M.; Barbosa, E. F. Variability resolution and product configuration with smarty: An experimental study on uml class diagrams. *JCS*, v. 13, n. 8, p. 307–319, 2017.

McCabe, T. J. A complexity measure. *IEEE Transactions on software Engineering*, , n. 4, p. 308–320, 1976.

de Morais, S. R.; Aussem, A. A novel markov boundary based feature subset selection algorithm. *Neurocomputing*, v. 73, n. 4-6, p. 578–584, 2010.

Naito, S.; Tsunoyama, M. Fault detection for sequential machines by transitions tours. ieee fault tolerant comput. symp. *IEEE Computer Soc. Press, pages 238w243*, 1981.

Olimpiew, E. M. *Model-based testing for software product lines.* Tese de Doutoramento, George Mason University, Fairfax, VA, USA, aAI3310145, 2008.

Olimpiew, E. M.; Gomaa, H. Model-based testing for applications derived from software product lines. In: *ACM SIGSOFT Software Engineering Notes*, ACM, 2005, p. 1–7.

OliveiraJr, E.; Gimenes, I.; Huzita, E. H. M.; Maldonado, J. C. A variability management process for software product lines. In: *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, 2005, p. 225–241.

OliveiraJr, E.; Gimenes, I. M.; Maldonado, J. C. Systematic management of variability in uml-based software product lines. *Journal of Universal Computer Science*, v. 16, n. 17, p. 2374–2393, 2010a.

OliveiraJr, E.; Gimenes, I. M.; Maldonado, J. C.; Masiero, P. C.; Barroca, L. Systematic evaluation of software product line architectures. *Journal of Universal Computer Science*, v. 19, n. 1, p. 25–52, 2013.

OliveiraJr, E.; Gimenes, I. M. S.; Maldonado, J. C. Systematic management of variability in uml-based software product lines. *Journal of Universal Computer Science (JUCS)*, v. 16, n. 17, p. 2374–2393, 2010b.

Oster, S. *Feature model-based software product line testing.* Tese de Doutoramento, Technische Universität, 2012.

Oster, S.; Zink, M.; Lochau, M.; Grechanik, M. Pairwise feature-interaction testing for spls: potentials and limitations. In: *Proceedings of the 15th International Software Product Line Conference, Volume 2*, ACM, 2011a, p. 6.

Oster, S.; Zorcic, I.; Markert, F.; Lochau, M. Moso-polite: tool support for pairwise and model-based software product line testing. In: *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, ACM, 2011b, p. 79–82.

Patel, S.; Shah, V. Automated testing of software-as-a-service configurations using a variability language. In: *Proceedings of the 19th International Conference on Software Product Line*, ACM, 2015, p. 253–262.

Pérez, A. M.; Kaiser, S. Bottom-up reuse for multi-level testing. *Journal of Systems and Software*, v. 83, n. 12, p. 2392–2415, 2010.

Perrouin, G.; Sen, S.; Klein, J.; Baudry, B.; Le Traon, Y. Automated and scalable t-wise test case generation strategies for software product lines. In: *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, IEEE, 2010, p. 459–468.

Petersen, K.; Vakkalanka, S.; Kuzniarz, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, v. 64, p. 1–18, 2015.

Petrenko, A. Nondeterministic state machine in protocol conformance testing. *Protocol Test Systems*, p. 363–378, 1994.

Pinheiro, A. C. *Subsídios para a aplicação de métodos de geração de casos de testes baseados em máquinas de estados.* Tese de Doutoramento, Universidade de São Paulo, 2012.

Pinheiro, A. C.; Simão, A. d. S. Jplavisfsm: Manual de instruções. 2012.

Pohl, K.; Böckle, G.; van Der Linden, F. J. *Software product line engineering: foundations, principles and techniques.* Springer Science & Business Media, 2005.

Reales, P.; Polo, M.; Caivano, D. Model based testing in software product lines. In: *International Conference on Enterprise Information Systems*, Springer, 2011, p. 270–283.

Reis, S.; Metzger, A.; Pohl, K. Integration testing in software product line engineering: A model-based technique. In: Dwyer, M. B.; Lopes, A., eds. *Fundamental Approaches to Software Engineering*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 321–335.

Reuys, A.; Kamsties, E.; Pohl, K.; Reis, S. Model-based system testing of software product families. In: *International Conference on Advanced Information Systems Engineering*, Springer, 2005, p. 519–534.

Rodrigues, E. d. M. Plets: a product line of model-based testing tools. 2013.

Rodrigues, E. d. M.; Zorzo, A. F.; Nakagawa, E.; Gimenez, I.; Maldonado, J.; de Oliveira, F. A software product line for model-based testing tools. *Journal of Systems and Software*, p. 1–26, 2012.

Sabnani, K.; Dahbura, A. A protocol test generation procedure. *Computer Networks and ISDN systems*, v. 15, n. 4, p. 285–297, 1988.

Samih, H.; Baudry, B. Relating variability modeling and model-based testing for software product lines testing. *Proceedings of the ICTSS*, p. 18–22, 2012.

Samih, H.; Bogusch, R. Mplm-matelo product line manager:[relating variability modelling and model-based testing]. In: *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*, ACM, 2014, p. 138–142.

Samih, H.; Le Guen, H.; Bogusch, R.; Acher, M.; Baudry, B. An approach to derive usage models variants for model-based testing. In: Merayo, M. G.; de Oca, E. M., eds. *Testing Software and Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014a, p. 80–96.

Samih, H.; Le Guen, H.; Bogusch, R.; Acher, M.; Baudry, B. Deriving usage model variants for model-based testing: an industrial case study. In: *Engineering of*

Complex Computer Systems (ICECCS), 2014 19th International Conference on, IEEE, 2014b, p. 77–80.

SEI The arcade game maker pedagogical product line. *https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=485941*, 2009.

Shafique, M.; Labiche, Y. A systematic review of model based testing tool support. *Carleton University, Canada, Tech. Rep. Technical Report SCE-10-04*, p. 01–21, 2010.

Shepperd, M. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, v. 3, n. 2, p. 30–36, 1988.

Sidhu, D. P.; Leung, T.-K. Formal methods for protocol testing: A detailed study. *IEEE transactions on software engineering*, v. 15, n. 4, p. 413–426, 1989.

Steffens, M.; Oster, S.; Lochau, M.; Fogdal, T. Industrial evaluation of pairwise spl testing with moso-polite. In: *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, ACM, 2012, p. 55–62.

Swain, S. K.; Mohapatra, D. P.; Mall, R. Test case generation based on use case and sequence diagram. *International Journal of Software Engineering*, v. 3, n. 2, p. 21–52, 2010.

Utting, M.; Legeard, B. *Practical model-based testing: a tools approach*. Elsevier, 2010.

Varshosaz, M.; Beohar, H.; Mousavi, M. R. Delta-oriented fsm-based testing. In: *International Conference on Formal Engineering Methods*, Springer, 2015, p. 366–381.

Vuong, S. T. The uiov-method for protocol test sequence generation. In: *Proc. 2nd IFIP Int. Workshop on Protocol Test Systems (IWPTS'89)*, 1989, p. 161–175.

Wang, S.; Ali, S.; Gotlieb, A. Automated product line methodologies to support model-based testing. In: *Demos/Posters/StudentResearch@ MoDELS*, 2013a, p. 56–60.

Wang, S.; Ali, S.; Yue, T.; Liaaen, M. Using feature model to support model-based testing of product lines: an industrial case study. In: *Quality Software (QSIC), 2013 13th International Conference on*, IEEE, 2013b, p. 75–84.

Warmer, J. B.; Kleppe, A. G. *The object constraint language: getting your models ready for mda*. Addison-Wesley Professional, 2003.

Weiss, D. M.; Lai, C. T. R. *Software product-line engineering: A family-based software development process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

Weissleder, S.; Lackner, H. Top-down and bottom-up approach for model-based testing of product lines. *arXiv preprint arXiv:1303.1011*, 2013.

Weissleder, S.; Schlingloff, H. An evaluation of model-based testing in embedded applications. In: *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, IEEE, 2014, p. 223–232.

Weissleder, S.; Sokenou, D.; Schlingloff, B.-H. Reusing state machines for automatic test generation in product lines. In: *1st workshop on model-based testing in practice (MoTiP)*, 2008, p. 19.

Zhang, M.; Ali, S.; Yue, T.; Norgre, R. Uncertainty-wise evolution of test ready models. *Information and Software Technology*, v. 87, p. 140–159, 2017.

Ziadi, T.; Hélouët, L.; Jézéquel, J.-M. Towards a uml profile for software product lines. In: *International Workshop on Software Product-Family Engineering*, Springer, 2003, p. 129–139.

A

Teste Baseado em Modelos para LPS: um Mapeamento Sistemático da Literatura

A.1 Planejamento do MSL

A.1.1 Objetivo da Pesquisa

O objetivo deste mapeamento sistemático da literatura é identificar estudos sobre Teste Baseado em Modelo (TBM) para Linha de Produto de Software (LPS) com foco em variabilidade, domínio de aplicação, ferramenta, modelos de LPS, entre outros.

A MSL apresentada aqui segue padrões definidos como principais orientações para a execução de um mapeamento sistemático da literatura por Kitchenham (2004) e Keele et al. (2007).

A.2 Metodologia de Pesquisa

Elaboramos a metodologia de pesquisa seguindo as diretrizes de Petersen et al. (2015) e Kitchenham et al. (2015).

A Figura - 1.1 mostra a visão geral do nosso processo de mapeamento sistemático.

Começamos nosso estudo definindo questões de pesquisa (Seção A.2.1). Em seguida, realizamos o processo de pesquisa (Seção A.2.2) para reunir um conjunto inicial de estudos

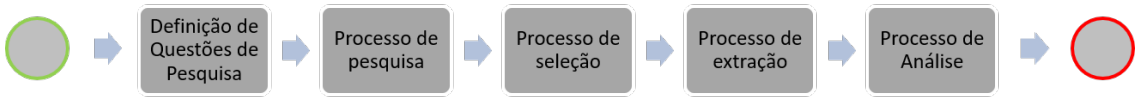


Figura 1.1: Visão geral do processo de mapeamento sistemático

primários de várias fontes diferentes. Com este conjunto de estudos, realizamos o processo de seleção (Seção A.2.3), filtrando estudos de acordo com diferentes critérios. O processo de extração (Seção A.2.4) foi realizado nos estudos filtrados para suportar o processo de análise. Não processo de análise (Seção A.2.5) respondemos as questões de pesquisa definidas para este estudo.

A.2.1 Objetivo e Questões de Pesquisa

Este MSL **tem como objetivo** coletar evidências da literatura, **com o propósito de** caracterizar Testes Baseados em Modelo, **com relação a** sua aplicação em Linhas de Produto de Software, **do ponto de vista de** Pesquisadores de LPS, **no contexto de** diferentes fontes digitais de estudo primário.

Nós formulamos seis questões de pesquisa com base no objetivo principal deste MSL, como segue:

- **RQ.1: Para quais domínios do aplicativo LPS, tipos de solução e propostas o TBM é usado?**Estamos interessados em reunir evidências de quais domínios de LPS são mais frequentes, como, por exemplo, software, aeroespacial e automotivo, bem como que tipo de soluções foram propostas para TBM de LPSs;

- **RQ.2: Quais abordagens de TBM, níveis de teste e artefatos foram usados para testar LPSs?** Nesta questão, o foco principal é identificar técnicas de TBM, níveis de teste e automação de teste;

- **RQ.3: Como a variabilidade e o tempo de ligação são tratados durante o TBM das LPSs?**Nesta questão, gostaríamos de compreender quais estudos primários levam em conta a variabilidade e como eles realizam o gerenciamento da variabilidade nas atividades de TBM da LPS, bem como se o tempo de ligação afeta as atividades do TBM nas LPSs;

- **RQ.4: O TBM suporta testes de requisitos não-funcionais de LPS?** Estamos interessados em coletar evidências se os estudos primários dependem do teste de requisitos não-funcionais de LPS;
- **RQ.5: Como as propostas de TBM de LPS são avaliadas?** As soluções de TBMs de LPSs devem ser avaliadas como um meio de fornecer evidência de sua viabilidade. Portanto, gostaríamos de entender que tipo de avaliação é realizada, como experimentos controlados e estudos de caso;
- **RQ.6: Como a rastreabilidade é considerada durante atividades de TBM para LPSs?** Rastreabilidade é um conceito fundamental de TBM e LPS. Nesta questão, reunimos evidências de como as soluções consideram tal conceito para atividades de TBM em LPS.

A.2.2 Processo de Pesquisa

A Figura - 1.2 mostra o processo de busca do MSL.

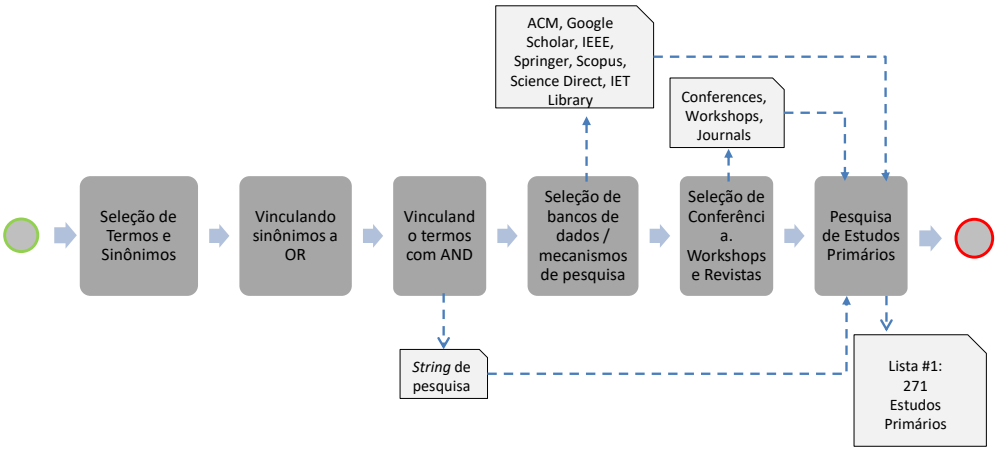


Figura 1.2: Processo de pesquisa por MSL

A estratégia de busca para encontrar estudos primários relevantes definidos para este MSL é baseada principalmente na seleção de termos e sinônimos relacionados ao TBM aplicado ao LPS. Esses termos e sinônimos levam em consideração algumas das palavras-chave de trabalhos relacionados e são apresentadas da seguinte forma:

- ***software***;

- ***product line***:*product-line*, *product family*, *product-family*, *product-families*, *family of products*;
- ***model-based testing***:*model based testing*, *TBM*.

Uma vez que tivemos esses termos e seus sinônimos, associamos esses sinônimos ao operador lógico " OR " e termos com " AND ". Assim, criamos nossa *string* de pesquisa geral, conforme apresentado na Tabela - 1.1.

Tabela 1.1: String de pesquisa geral do MSL

software **AND** ("product line" **OR** "product-line" **OR** "product family" **OR** "product-family" **OR** "product-families" **OR** "family of products" **OR** variability) **AND** ("model-based testing" **OR** "model based testing" **OR** TBM)

Além disso, selecionamos fontes de pesquisa (eletrônica e manual), idioma dos estudos primários e tipo de publicação da seguinte forma:

- **Fontes de Pesquisa:** bases de dados eletrônicas indexadas (IEEE, ACM, Science-Direct, Scopus, Biblioteca TET IET, Google Scholar, Springer) listadas na Tabela - 1.2 e pesquisa manual em revistas especializadas, conferências e workshops como na Tabela - 1.3 e Tabela - 1.4;
- **Linguagem de Estudos:** Inglês, devido à sua abrangência na área de Ciência da Computação;
- **Tipo de Publicação:** estudos primários revisados por pares publicados em periódicos, conferências / workshops ou capítulos de livros.

As razões para escolher essas fontes de pesquisa estão listadas:

- fonte de ciência da computação mundialmente conhecida;
- fornece um mecanismo de pesquisa avançado capaz de lidar com consultas avançadas;
- indexado internacionalmente;
- índices de origem conferências e periódicos mais relevantes em Engenharia de Software;
- índices de fontes de artigos com fator de alto impacto (JCR e / ou índice H);

Tabela 1.2: Fontes de pesquisa eletrônicas definidas

Fonte Eletrônica	**URL**
ACM Digital Library	`http://dl.acm.org`
IEEE Xplore	`http://ieeexplore.ieee.org`
ScienceDirect	`http://www.sciencedirect.com`
Scopus	`http://www.info.sciverse.com/scopus`
Springer	`http://www.springer.com`
Google Scholar	`https://scholar.google.com`
IET Digital Library	`http://digital-library.theiet.org/content/journals/iet-sen`

Tabela 1.3: Conferências Definidas e Workshops para Pesquisa Manual

Acrônimo	**Conferência/Workshop**
ICECCS	Engineering of Complex Computer Systems
AISE	Advanced Information Systems Engineering
ICST	International Conference on Software Testing
ICIS	Computer and Information Science
ACM SIGSOFT	Software Engineering Nãotes
MSIS	Workshop on Variability Modeling of Software-Intensive Systems
ILPS	International Software Product Line Conference
ICSTSS	International Conference on Testing Software and Systems
QA&TEST	International Conference on Software QA and Testing
IFIP	International Conference on Testing Software and Systems

Tabela 1.4: Diários Definidos para Pesquisa Manual

Acrônimo	***Journal***
STVR	Software Testing, Verification & Reliability
ESE	Empirical Software Engineering
JSS	Journal of Systems and Software
IST	Information and Software Technology
ASC	Applied Soft Computing
SQJ	Software Quality Journal
SCP	Science of Computer Programming

- Fonte não cessou a publicação.

Como última atividade do processo de busca, realizamos a busca por estudos primários aplicando a string de busca a fontes eletrônicas, e buscando estudos primários manualmente em congressos, workshops e periódicos, conforme apresentado nas próximas seções. Um total de 271 estudos (Lista # 1) foram recuperados no final deste processo.

Avaliação de protocolo

Elaboramos um questionário [1] para avaliar nosso protocolo MSL para coletar a opinião de pesquisadores que trabalharam com TBM e / ou LPS, para facilitar a construção da pesquisa. O questionário é composto por oito itens, sete questões em escala de Likert (Li, 2013) e uma questão aberta, como segue:

[1] `https://drive.google.com/open?id=1UOzfLS6JLfO6krrFkbpwLOmflCtrNkVBJ5cGKDjWfJw`

1. Este estudo baseia-se em uma importante questão de pesquisa em *Model-Based Testing* (TBM) de linhas de produto de software (LPS).
2. A cadeia de pesquisa é adequadamente derivada das questões de pesquisa.
3. As fontes de pesquisa definidas são suficientes para cobrir estudos relevantes.
4. Os critérios de inclusão e exclusão são adequados e adequadamente descritos.
5. A pesquisa pode avaliar a qualidade / validade dos estudos selecionados.
6. O processo de extração aborda adequadamente as questões de pesquisa.
7. O processo de análise é apropriado para responder às questões de pesquisa.

Cada pergunta do tipo likert tem as seguintes opções como respostas:

- **Eu discordo totalmente:** quando o protocolo não atende aos critérios da pergunta de forma alguma;
- **Discordo Parcialmente** quando o protocolo não atende a alguns critérios da questão;
- **Neutro:** quando o protocolo não deixa claro se ele atende ou não aos critérios da questão;
- **Eu Concordo Parcialmente** quando o protocolo atende a alguns critérios da questão; e
- **Eu concordo plenamente:** quando o protocolo atende totalmente aos critérios da pergunta.

para os pesquisadores anotarem onde quer que julguem importante sobre o protocolo

Na parte inferior do questionário, uma opção foi disponibilizada para sugestões gratuitas sobre melhorias do protocolo. O objetivo principal desta avaliação foi refinar o protocolo com base nas respostas e sugestões dos pesquisadores.

Enviamos o questionário de avaliação para sete pesquisadores na área de Sistemas de Informação, Engenharia de Software e Engenharia Elétrica (ver Tabela - 1.5). Disponibilizamos o questionário por 20 dias. Nós tivemos cinco respostas.

Os pesquisadores convidados fizeram uma avaliação muito produtiva. Todos consideram que a pesquisa pode gerar dados satisfatórios com base no contexto apresentado

Tabela 1.5: Pesquisadores que avaliaram o protocolo de MSL

ID	Nível de educação	Instituição	Área Especialista
R.1	Ph.D.	Universidade Federal de Tecnologia do Paraná	Sistemas de informação
R.2	Ph.D.	Universidade Federal do Pampa	Engenharia de software
R.3	Ph.D.	Universidad Politècnica de València Espanha	Engenharia elétrica
R.4	Ph.D.	Universidade Federal do Paraná	Engenharia de software
R.5	M.Sc.	Instituto SENAI de Tecnologia em Metalmecânica	Engenharia elétrica

sobre o tema. A maioria também concorda que o Search String é adequado com base nas questões de pesquisa.

Outro ponto de concordância é que as fontes e os tipos de pesquisa podem abranger estudos relevantes, mas com relação aos critérios de inclusão e exclusão, alguns discordaram, por exemplo, que a seleção de artigos está limitada a apenas os últimos 10 anos de pesquisa. Nesse caso, a justificativa seria evitar o estudo duplicado que foi atualizado ou continuado.

Uma sugestão foi realizar uma avaliação baseada no número de citações de um determinado estudo. Essa sugestão foi aplicada em conjunto com o questionário de critérios.

Busca Eletrônica

Realizamos a busca eletrônica de agosto / 2017 a outubro / 2017. Nós derivamos nossa *string* de busca geral para cada uma das fontes eletrônicas, como apresentado em Tabela - 1.6.

Tabela 1.6: Fontes eletrônicas e sequências de pesquisa adaptadas

Fonte Eletrônica	String de pesquisa adaptada
ACM	(*"software"+"product line" "product-line" "product family" "product-family" "product-families" "family of products" "variability"+"model-based testing" "model based testing" "TBM"*)
Google Scholar	(*"Software") AND ("product line" or "product-line" or "product family" or "product-family" or "product-families" or "family of products" or "variability") AND ("model-based testing" or "model based testing" or "TBM"*)
IEEE	((*"software") AND ("product line" OR "product-line" OR "product family" OR "product-family" OR "product-families" OR "family of products" OR "variability") AND ("model-based testing" OR "model based testing" OR "TBM")*)
Springer	(*(software) AND ("product line" or "product-line" or "product family" or "product-family" or "product-families" or "family of products" or "variability") AND ("model-based testing" or "model based testing" or TBM)*
Scopus	*(TITLE-ABS-KEY (software) AND TITLE-ABS-KEY (product line OR product-line OR product family OR product-family OR product-families OR family of products OR variability) AND TITLE-ABS-KEY (model-based testing OR model based testing OR TBM)) AND (PUBYEAR > 2005)*
Science Direct	*("software") AND ("product line" OR "product-line" OR "product family" OR "product-family" OR "product-families" OR "family of products" OR "variability") AND ("model-based testing" OR "model based testing" OR "TBM")*
IET Digital Library	*(("software") AND ("product line" OR "product-line" OR "product family" OR "product-family" OR "product-families" OR "family of products" OR "variability") AND ("model-based testing" OR "model based testing" OR "TBM"))*

A busca eletrônica retornou 251 estudos, como podemos ver na Tabela - 1.7, coluna " Retornado ". A Science Direct foi a fonte mais voltada, com 125 estudos, seguida pela Scopus, com 62 estudos.

Tabela 1.7: Número de estudos de fontes eletrônicas

Fonte Eletrônica	Retornou	Aceitos	Duplicados	Rejeitados
Science Direct	125	07	00	118
Scopus	62	10	21	31
IEEE Xplore	25	00	00	25
ACM	20	16	01	03
IET Digital Library	09	00	00	09
Springer	07	05	01	01
Google Scholar	03	02	00	01
Total	**251**	**40**	**23**	**188**

Busca Manual

A busca manual retornou 20 estudos de conferências, oficinas e periódicos, conforme apresentado na Tabela - 1.8, coluna " Retornado ".

Tabela 1.8: Número de estudos de fontes manuais

Fonte	Retornados	Aceitos	Rejeitados
Software & Systems Modeling	01	01	00
Model Driven Engineering Languages and Systems	01	00	01
Software Quality Journal	01	01	00
Software Testing, Verification and Validation (ICST)	01	01	00
Computer and Information Science (ICIS)	01	01	00
ACM SIGSOFT Software Engineering Nãotes	01	01	00
Workshop on Variability Modeling of Software-Intensive Systems	01	00	01
International Software Product Line Conference	01	00	01
IEEE Software	01	00	01
Google Scholar	11	06	05
Total	**20**	**11**	**09**

A.2.3 Processo de Seleção

A Figura - 1.3 mostra o processo de seleção do MSL.

As próximas seções apresentam como realizamos o processo de seleção.

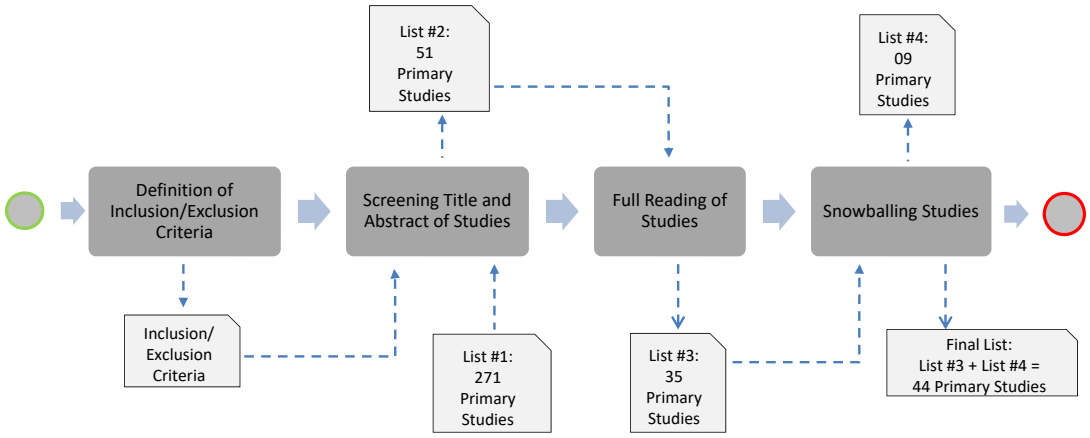


Figura 1.3: Processo Seletivo MSL

Definição de Critérios de Inclusão e Exclusão

Para selecionar estudos relevantes e contribuir para responder às questões de pesquisa deste estudo, definimos os seguintes critérios de inclusão e exclusão:

- **Critério de Inclusão:**
  - IC.1 - estudos que discutem atividades de TBM realizadas exclusivamente no domínio LPS.
- **Critérios de exclusão:**
  - EC.1 - estudos não abordam atividades de TBM realizadas exclusivamente no domínio LPS;
  - EC.2 - estudos com texto incompleto;
  - EC.3 - estudos em um idioma diferente do inglês para promover a disseminação e reprodutibilidade internacional;
  - EC.4 - estudos com menos de quatro páginas, pois acreditamos que eles não têm espaço para contribuições relevantes;
  - EC.5 - estudos duplicados;
  - EC.6 - estudos indisponíveis, mesmo em contato com os autores.

Títulos de triagem e resumos de estudos

Depois de definir os critérios de inclusão e exclusão, aplicamos os mesmos aos estudos lendo títulos e resumos de Lista # 1 (271 estudos). Tal leitura filtrou Lista # 1 para

aceitar 51 estudos (Lista # 2), onde 40 estudos provêm da busca eletrônica (Tabela - 1.7, coluna “ Aceito ”) e 11 da busca manual (Tabela - 1.8, coluna “ Aceito ”).

Tabela 1.9: Artigos selecionados para leitura completa

Título	**Ano**	**Local**	**Tipo Pub.**
Delta-Oriented Model-Based LPS Regression Testing (Lity et al., 2012)	2012	ACM	Evento
Industrial Evaluation of Pairwise LPS Testing with MoSo-PoLiTe (Steffens et al., 2012)	2012	ACM	Evento
Model-Based Coverage-Driven Test Suite Generation for Software Product Lines (Cichos et al., 2011)	2011	ACM	Journal
MoSo-PoLiTe - Tool Support for Pairwise and Model-Based Software Product Line Testing (Oster et al., 2011b)	2011	ACM	Evento
MPLM - MaTeLo Product Line Manager (Samih et al., 2014a)	2014	ACM	Evento
On the use of test cases in model-based software product line development (Knapp et al., 2014)	2014	ACM	Evento
Pairwise Feature-Interaction Testing for LPSs:Potentials and Limitations (Oster et al., 2011a)	2011	ACM	Evento
Deriving Usage Model Variants for Model- based Testing: An Industrial Case Study (Samih et al., 2014b)	2014	IEEE	Evento
Model-based Software Product Line Testing by Coupling Feature Models with Hierarchical Markov Chain Usage Models (Gebizli e Sözer, 2016)	2016	IEEE	Evento
Model-Based Test Design of Product Lines: Raising Test Design to the Product Line Level (Lackner et al., 2014)	2014	IEEE	Journal
Requirements-Based Delta-Oriented LPS Testing (Dukaczewski et al., 2013)	2013	IEEE	Evento
Using Feature Model to Support Model-Based Testing of Product Lines: (Wang et al., 2013b) An Industrial Case Study	2013	IEEE	Journal
An automated Model-based Testing Approach in Software Product Lines (García Gutiérrez et al., 2010) Using a Variability Language	2010	Politécnica Arquivo digital UPM	Evento
Automated Product Line Methodologies to Support Model-Based Testing (Wang et al., 2013a)	2013	CEUR Proceedings	Evento
Behavioural Model Based Testing of Software Product Lines (Devroey, 2014)	2014	ACM	Evento
Feature Model-based Software Product Line Testing (Oster, 2012)	2012	TUprints Darmstadt publication service	Journal
Model-based pairwise testing for feature interaction coverage (Lochau et al., 2012a) in software product line engineering	2011	Springer	Journal
Model-based Test Generation for Software Product Line (Cai e Zeng, 2013)	2013	IEEE	Evento
Model-Based Testing for Software Product Lines (Olimpiew, 2008)	2008	Springer	Evento
PLETS - A Product Line of Model- Based Testing Tools (Rodrigues, 2013)	2013	PUC-RS	Evento
Top-Down and Bottom-Up Approach for Model-Based (Weißleder e Lackner, 2013) Testing of Product Lines	2013	EPTCS	Evento
A Product Line Modeling and Configuration Methodology to Support Model-Based Testing: (Ali et al., 2012) An Industrial Case Study	2012	Springer	Journal
Coverage Criteria for Behavioural Testing of Software Product Lines (Devroey et al., 2014a)	2014	Springer	Evento
A Model Based Testing Approach for Model-Driven Development (Lamancha et al., 2010) and Software Product Lines	2010	Springer	Evento
A Vision for Behavioural Model-Driven Validation of Software Product Lines (Devroey et al., 2012)	2012	Springer	Evento
Abstract Test Case Generation for Behavioural Testing of Software Product Lines (Devroey et al., 2014b)	2014	ACM	Evento

Tabela - 1.9 Continuação da página anterior

Título	Ano	Local	Tipo Pub.
Applying Incremental Model Slicing to Product-Line Regression Testing (Lity et al., 2016)	2016	Springer	Journal
Automated Testing of Software-as-a-Service Configurations using a Variability Language (Patel e Shah, 2015)	2015	ACM	Evento
Delta-Oriented FSM-Based Testing (Varshosaz et al., 2015)	2015	Springer	Evento
Incremental Model-Based Testing of Delta-oriented Software Product Lines (Lochau et al., 2012b)	2012	Springer	Journal
Model Based Testing in Software Product Lines (Reales et al., 2011)	2011	Springer	Evento
Model-Based Testing (Lochau et al., 2014)	2014	Springer	Evento
Parameterized Preorder Relations for Model-Based Testing of Software Product Lines (Lochau e Kamischke, 2012)	2012	Springer	Evento
Poster:VIBeS, Transition System Mutation Made Easy (Devroey et al., 2015)	2015	IEEE	Evento
Spinal Test Suites for Software Product Lines (Beohar e Mousavi, 2014b)	2014	EPTCS	Evento
Automated model-based testing using the UML testing profile and QVT (Lamancha et al., 2009)	2009	ACM	Evento
Relating Variability Modeling and Model-Based Testing for Software Product Lines Testing (Samih e Baudry, 2012)	2012	ICTSS	Evento
An Evaluation of Model-Based Testing in Embedded Applications (Weißleder e Schlingloff, 2014)	2014	IEEE	Evento
Assessing Software Product Line Testing Via Model-Based Mutation An Application to Similarity Testing (Henard et al., 2013)	2013	IEEE	Evento
Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines (Perrouin et al., 2010)	2010	IEEE	Evento
Model-based Testing of System Requirements using UML Use Case Models (Hasling et al., 2008)	2008	IEEE	Evento
Successive refinement of models for model-based testing to increase system test effectiveness (Gebizli et al., 2016)	2016	IEEE	Evento
A Software Product Line for Model-Based Testing Tools (Rodrigues et al., 2012)	2012	PUC-RS	Evento
Reusing State Machines for Automatic Test Generation in Product Lines (Weißleder et al., 2008)	2008	MoTip	Evento
A Nãovel Markov Boundary Based Feature Subset Selection Algorithm (de Morais e Aussem, 2010)	2010	Elsevier	Journal
A Nãovel Model-Based Testing Approach for Software Product Lines (Damiani et al., 2017)	2016	Springer	Evento
Abstract Test Case Generation for Behavioural Testing of Software Product Lines (Devroey et al., 2014b)	2014	ACM	Evento
An Overview on Test Generation from Functional Requirements (Escalona et al., 2011)	2011	Elsevier	Journal
Basic Behavioral Models for Software Product Lines:Expressiveness and Testing Pre-Orders (Beohar et al., 2016)	2016	Elsevier	Journal
Bottom-up reuse for multi-level testing (Pérez e Kaiser, 2010)	2010	Elsevier	Journal
Colored Model Based Testing for Software Product Lines (Farrag, 2010)	2010	semanticscholar	Gray Literature
IncLing:Efficient Product-Line Testing Using Incremental Pairwise Sampling (Al-Hajjaji et al., 2016)	2016	ACM	Evento
Input-output conformance testing based on featured transition systems (Beohar e Mousavi, 2014a)	2014	ACM	Evento
Model-based testing of automotive systems (Bringmann e Krämer, 2008)	2008	IEEE	Evento
Model-based system testing of software product families (Reuys et al., 2005)	2005	Springer	Evento
Model-based testing for applications derived from software product lines (Olimpiew e Gomaa, 2005)	2005	ACM	Evento
Reducing the Concretization Effort in FSM-Based Testing of Software Product Lines (Fragal et al., 2017)	2016	IEEE	Evento
Refinement-based testing of delta-oriented product lines (Damiani et al., 2013)	2013	ACM	Evento
Risk-based integration testing of software product lines (Lachmann et al., 2017)	2016	ACM	Evento
Uncertainty-wise evolution of test ready models (Zhang et al., 2017)	2016	Elsevier	Journal

Leitura Completa de Estudos

Dada Lista # 2, lemos cada um dos 51 estudos. Rejeitamos 16 deles (veja Tabela - 1.10) com base nos critérios de exclusão da Seção A.2.3, fornecendo assim Lista # 3 com 35 estudos.

Tabela 1.10: Estudos removidos de acordo com os critérios de exclusão

Título	**EC.1**	**EC.2**	**EC.3**	**EC.4**	**EC.5**	**EC.6**
A Novel Markov Boundary Based Feature Subset Selection Algorithm	X	-	-	-	-	-
A Nãovel Model-Based Testing Approach for Software Product Lines	-	-	-	-	X	-
Abstract Test Case Generation for Behavioural Testing of Software Product Lines	X	-	-	-	-	-
An Overview on Test Generation from Functional Requirements	X	-	-	-	-	-
Basic Behavioral Models for Software Product Lines:Expressiveness and Testing Pre-Orders	-	-	-	X	-	-
Bottom-up reuse for multi-level testing	X	-	-	-	-	-
Colored Model Based Testing for Software Product Lines	X	-	-	-	X	-
IncLing:Efficient Product-Line Testing Using Incremental Pairwise Sampling	X	-	-	-	-	-
Input-output conformance testing based on featured transition systems	X	-	-	-	-	-
Model-based testing of automotive systems	-	-	-	X	-	-
Model-based system testing of software product families	-	-	-	-	X	-
Model-based testing for applications derived from software product lines	-	-	-	-	X	-
Reducing the Concretization Effort in FSM-Based Testing of Software Product Lines	X	-	-	-	-	-
Refinement-based testing of delta-oriented product lines	X	-	-	-	-	-

Risk-based integration testing of software product lines	X	-	-	-	-	-
Uncertainty-wise evolution of test ready models	X	-	-	-	-	-

Estudos derivados do Snowballing

Realizamos bolas de neve invertidas em Lista # 3, onde avaliamos as referências dos estudos. Dez estudos retornaram de bola de neve, e os critérios de inclusão e exclusão foram aplicados a eles. Portanto, um deles foi duplicado conforme apresentado em Tabela - 1.11. Os nove estudos restantes compuseram Lista # 4.

Estudos do Snowballing

ID	Fonte do estudo	ID	Estudo do Snowballing	Estatus
S24	A Model Based Testing Approach for Model-Driven Development and Software Product Lines	S36	Automated model-based testing using the UML testing profile and QVT	Aceito
S8	Deriving Usage Model Variants for Model-based Testing:An Industrial Case Study	S37	Relating Variability Modeling and Model-Based Testing for Software Product Lines Testing	Aceito
S10	Model-Based Test Design of Product Lines:Raising Test Design to the Product Line Level	S38	An Evaluation of Model-Based Testing in Embedded Applications	Aceito
S10	Model-Based Test Design of Product Lines:Raising Test Design to the Product Line Level	S39	Assessing Software Product Line Testing Via Model-Based Mutation An Application to Similarity Testing	Aceito
S5	MPLM-MaTeLo Product Line Manager	S40	Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines	Aceito
S20	PLETS - a Product Line of Model-Based Testing Tools	S41	Model based testing of system requirements using UML use case models	Aceito
S9	Model-based Software Product Line Testing by Coupling Feature Models with Hierarchical Markov Chain Usage Models	S42	Successive refinement of models for model-based testing to increase system test effectiveness	Aceito
S20	PLETS - a Product Line of Model-Based Testing Tools	S43	A Software Product Line for Model-Based Testing Tools	Aceito
S21	Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines	S44	Reusing State Machines for Automatic Test Generation in Product Lines	Aceito
S5	MPLM-MaTeLo Product Line Manager	–	An approach to derive usage models variants for model-based	Duplicado

A Tabela - 1.12 apresenta o conjunto final de estudos selecionados para este MSL, composto por 44 estudos (Lista Final), dos quais:os primeiros 35 são de pesquisas eletrônicas e manuais e os restantes nove são do *Snowballing*.

Tabela 1.12: Lista final de estudos

ID	Título	Ano	Local	Tipo de Pub.
S1	Delta-Oriented Model-Based LPS Regression Testing (Lity et al., 2012)	2012	ACM	Evento
S2	Industrial Evaluation of Pairwise LPS Testing with MoSo-PoLiTe (Steffens et al., 2012)	2012	ACM	Evento
S3	Model-Based Coverage-Driven Test Suíte Generation for Software Product Lines (Cichos et al., 2011)	2011	ACM	Journal
S4	MoSo-PoLiTe - Tool Support for Pairwise and Model-Based Software Product Line Testing (Oster et al., 2011b)	2011	ACM	Evento
S5	MPLM - MaTeLo Product Line Manager (Samih e Bogusch, 2014)	2014	ACM	Evento
S6	On the use of test cases in model-based software product line development (Knapp et al., 2014)	2014	ACM	Evento
S7	Pairwise Feature-Interaction Testing for LPSs:Potentials and Limitations (Oster et al., 2011a)	2011	ACM	Evento
S8	Deriving Usage Model Variants for Model- based Testing: An Industrial Case Study (Samih et al., 2014b)	2014	IEEE	Evento
S9	Model-based Software Product Line Testing by Coupling Feature Models with Hierarchical Markov Chain Usage Models (Gebizli e Sözer, 2016)	2016	IEEE	Evento
S10	Model-Based Test Design of Product Lines: Raising Test Design to the Product Line Level (Lackner et al., 2014)	2014	IEEE	Journal
S11	Requirements-Based Delta-Oriented LPS Testing (Dukaczewski et al., 2013)	2013	IEEE	Evento
S12	Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study (Wang et al., 2013b)	2013	IEEE	Journal
S13	An automated Model-based Testing Approach in Software Product Lines Using a Variability Language (García Gutiérrez et al., 2010)	2010	Politécnica Arquivo digital UPM	Evento
S14	Automated Product Line Methodologies to Support Model-Based Testing (Wang et al., 2013a)	2013	CEUR Proceedings	Evento
S15	Behavioural Model Based Testing of Software Product Lines (Devroey, 2014)	2014	ACM	Evento
S16	Feature Model-based Software Product Line Testing (Oster, 2012)	2012	TUprints Darmstadt publication service	Journal
S17	Model-based pairwise testing for feature interaction coverage in software product line engineering (Lochau et al., 2012a)	2011	Springer	Journal
S18	Model-based Test Generation for Software Product Line (Cai e Zeng, 2013)	2013	IEEE	Evento
S19	Model-Based Testing for Software Product Lines (Olimpiew, 2008)	2008	Springer	Evento
S20	PLETS - A Product Line of Model- Based Testing Tools (Rodrigues, 2013)	2013	PUC-RS	Evento
S21	Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines (Weißleder e Lackner, 2013)	2013	EPTCS	Evento
S22	A Product Line Modeling and Configuration Methodology to Support Model-Based Testing: An Industrial Case Study (Ali et al., 2012)	2012	Springer	Journal
S23	Coverage Criteria for Behavioural Testing of Software Product Lines (Devroey et al., 2014a)	2014	Springer	Evento
S24	A Model Based Testing Approach for Model-Driven Development and Software Product Lines (Lamancha et al., 2010)	2010	Springer	Evento
S25	A Vision for Behavioural Model-Driven Validation of Software Product Lines (Devroey et al., 2012)	2012	Springer	Evento

Tabela - 1.12 Continuação da página anterior

ID	Título	Ano	Local	Tipo de Pub.
S26	Abstract Test Case Generation for Behavioural Testing of Software Product Lines (Devroey et al., 2014b)	2014	ACM	Evento
S27	Applying Incremental Model Slicing to Product-Line Regression Testing (Lity et al., 2016)	2016	Springer	Journal
S28	Automated Testing of Software-as-a-Service Configurations using a Variability Language (Patel e Shah, 2015)	2015	ACM	Evento
S29	Delta-Oriented FSM-Based Testing (Varshosaz et al., 2015)	2015	Springer	Evento
S30	Incremental Model-Based Testing of Delta-oriented Software Product Lines (Lochau et al., 2012b)	2012	Springer	Journal
S31	Model Based Testing in Software Product Lines (Reales et al., 2011)	2011	Springer	Evento
S32	Model-Based Testing (Lochau et al., 2014)	2014	Springer	Evento
S33	Parameterized Preorder Relations for Model-Based Testing of Software Product Lines (Lochau e Kamischke, 2012)	2012	Springer	Evento
S34	Poster:VIBeS, Transition System Mutation Made Easy (Devroey et al., 2015)	2015	IEEE	Evento
S35	Spinal Test Suites for Software Product Lines (Beohar e Mousavi, 2014b)	2014	EPTCS	Evento
S36	Automated model-based testing using the UML testing profile and QVT (Lamancha et al., 2009)	2009	ACM	Evento
S37	Relating Variability Modeling and Model-Based Testing for Software Product Lines Testing (Samih e Baudry, 2012)	2012	ICTSS	Evento
S38	An Evaluation of Model-Based Testing in Embedded Applications (Weißleder e Schlingloff, 2014)	2014	IEEE	Evento
S39	Assessing Software Product Line Testing Via Model-Based Mutation An Application to Similarity Testing (Henard et al., 2013)	2013	IEEE	Evento
S40	Automated and Scalable T-wise Test Case Generation Strategies (Perrouin et al., 2010) for Software Product Lines	2010	IEEE	Evento
S41	Model-based Testing of System Requirements using UML Use Case Models (Hasling et al., 2008)	2008	IEEE	Evento
S42	Successive refinement of models for model-based testing to increase system test effectiveness (Gebizli et al., 2016)	2016	IEEE	Evento
S43	A Software Product Line for Model-Based Testing Tools (Rodrigues et al., 2012)	2012	PUC-RS	Evento
S44	Reusing State Machines for Automatic Test Generation in Product Lines (Weißleder et al., 2008)	2008	MoTip	Evento

A.2.4 Processo de Extração

A Figura - 1.4 mostra o processo de extração do MSL.

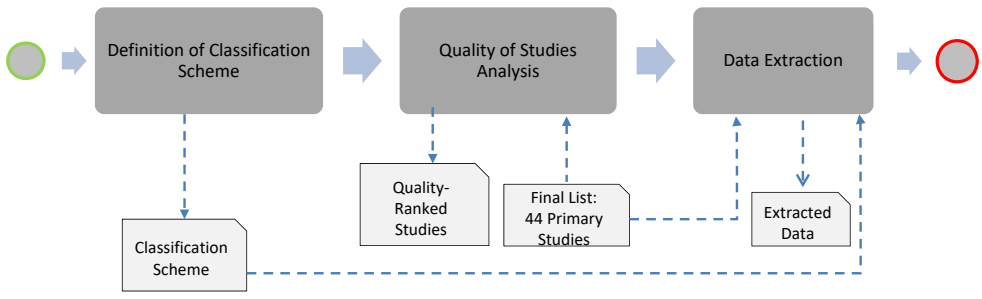


Figura 1.4: Processo de extração de MSL

Definição do Esquema de Classificação

Análise de Qualidade de Estudos

Para avaliação da qualidade dos estudos, definimos um questionário e aplicamos em cada estudo. Portanto, elaboramos as três perguntas a seguir:

- O par de estudos é revisado?
- O objetivo do estudo é claro?
- A proposta do estudo foi avaliada / validada?

Para cada questão, havia três alternativas, nas quais apenas uma delas poderia ser escolhida:“ Não ”, “ Sim ” e “ Parcialmente ”. Estudos com “ Não ” para quaisquer questões são automaticamente descartados. Nós cuidadosamente lemos novamente os estudos com ”Parcialidade”como uma resposta para qualquer pergunta.

Adotamos este procedimento, assim como a aplicação dos critérios de inclusão e exclusão para garantir contribuições mínimas dos estudos selecionados para o processo de extração.

Realizamos a extração de dados para coletar informações necessárias para responder às questões de pesquisa, bem como analisar os estudos dos critérios de seleção. Portanto, definimos os seguintes critérios de qualidade (QC) para garantir a qualidade mínima dos estudos:

- QC.1 - O estudo descreve claramente o propósito da pesquisa?
- QC.2 - O campo de ação é compatível com a área de pesquisa?
- QC.3 - Como o estudo é avaliado?
- QC.4 - Houve uma coleta de dados apropriada?
- QC.5 - Houve uma análise de dados apropriada?
- QC.6 - O estudo apresenta resultados consistentes com seus objetivos?
- QC.7 - Os resultados contribuem para o processo realizado para a pesquisa?

Extração de dados

Os próximos itens apresentam a lista de dados que definimos como extraídos de cada estudo primário:

- Dados bibliométricos:Título, Autor(es), Ano de Publicação, Fonte de Publicação, Tipo de Publicação, Tipo de Documento;
- Domínio da Solução;
- Inclui variabilidade ?;
- *Feature Interaction*?;
- Método de Execução;
- Item de teste usado;
- Nível de teste aplicado;
- Abordagem de Teste Usada;
- Nível de cobertura;
- Trate a rastreabilidade ?;
- Finalidade do TBM;
- Teste de Requisito Não-Funcional é Suportado ?;
- Artefato de Origem;
- Artefato Intermediário;
- Uso de Ferramentas;
- Tempo de Ligação;
- Atividades do TBM avaliadas;
- Resultados da pesquisa;
- Método de Coleta de Evidências;
- Local de validação;
- Tipo de contribuição.

A.2.5 Processo de Análise

A Figura - 1.5 mostra o processo de análise do MSL.

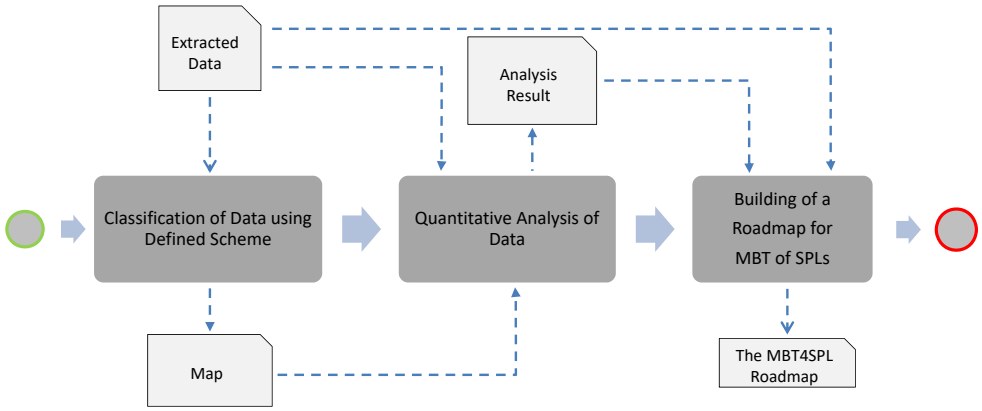


Figura 1.5: Processo de Análise MSL

Classificação de dados usando o esquema definido
Análise Quantitativa de Dados
Criação de um roteiro para TBM de LPSs

A.3 Resultados

A.3.1 Caracterização dos Estudos

Estudos por ano

É importante entender como o tema de pesquisa do TBM aplicado ao LPS se comportou ao longo dos últimos anos. Portanto, Tabela - 1.13 apresenta o número de estudos por ano e a Figura - 1.6 mostra a distribuição desses estudos ao longo dos anos.

Tabela 1.13: Número de estudos por ano

Tipo de local	2008	2009	2010	2011	2012	2013	2014	2015	2016	Count
Conferência	1	1	1	2	-	3	6	3	2	19
Workshop	2	1	1	1	4	3	1	-	-	13
Journal	-	-	-	2	3	1	1	-	1	8
Simpósio	-	-	-	-	2	-	2	-	-	4
Total	3	2	2	5	9	7	10	3	3	44

Com base na Figura - 1.6, observamos um número crescente de estudos ao longo dos anos. Especial atenção é dada a 2014 com 10 estudos. Além disso, com exceção de 2006 e 2007, sem ocorrência, os estudos são bem distribuídos na maioria dos anos, como em 2008, 2009, 2010, 2011, 2015 e 2016.

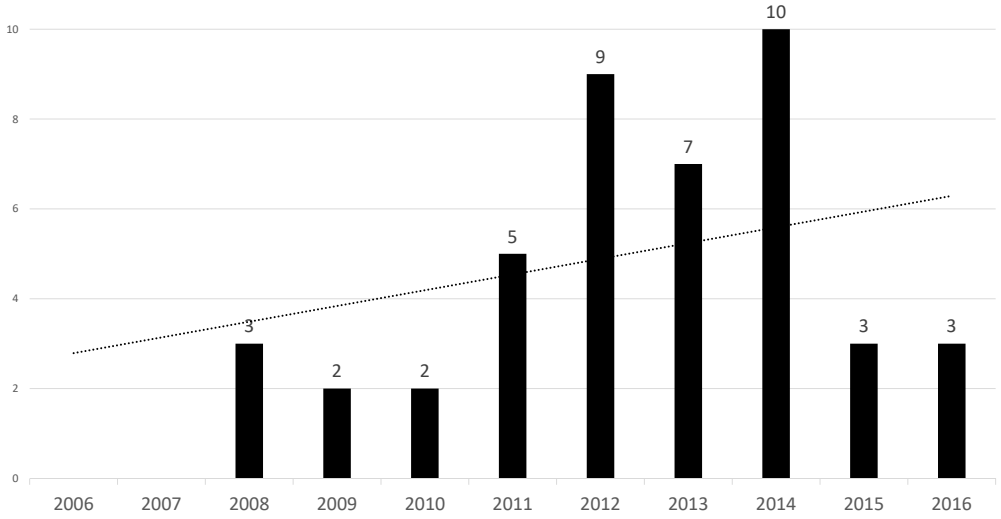


Figura 1.6: Distribuição de estudos por ano

Qualidade de Estudos

Para garantir a qualidade mínima dos estudos, realizamos uma avaliação de qualidade conforme planejado na seção A.2.4, levando em consideração a aplicação dos critérios de qualidade da Seção A.2.4.

Todos os estudos da Lista Final têm qualidade mínima com base em tais critérios, como podemos ver na Tabela - 1.14 com a maioria das respostas “ Sim ”.

Tabela 1.14: Avaliação de Qualidade de Estudos

ID do Estudo	QC.1-Propósito	QC.2-Compatibilidade	QC.3-Avaliação	QC.4-Coleção de dados	QC.5-Análise de dados	QC.6-Consistência dos Resultados	QC.7-Contribuição
S1	Sim	Não	Sim	Sim	Sim	Sim	Sim
S2	Sim	Não	Sim	Sim	Sim	Sim	Sim
S3	Sim	Não	Sim	Sim	Sim	Sim	Sim
S4	Sim	Não	Não Informado	Sim	Sim	Sim	Sim
S5	Sim	Não	Não Informado	Não Informado	Não Informado	Sim	Sim
S6	Sim	Não	Não Informado	Sim	Sim	Sim	Sim
S7	Sim	Não	Sim	Sim	Sim	Sim	Sim
S8	Sim	Não	Sim	Sim	Sim	Sim	Sim
S9	Sim	Não	Sim	Sim	Sim	Sim	Sim
S10	Sim	Não	Sim	Sim	Sim	Sim	Sim
S11	Sim	Não	Não	Sim	Sim	Sim	Sim
S12	Sim	Não	Sim	Sim	Sim	Sim	Sim
S13	Sim	Sim	Não Informado	Sim	Sim	Sim	Sim
S14	Sim	Não	Sim	Sim	Sim	Sim	Sim
S15	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S16	Sim	Não	Não Informado	Sim	Sim	Sim	Sim
S17	Sim	Não	Sim	Sim	Sim	Sim	Sim
S18	Sim	Sim	Não Informado	Sim	Sim	Sim	Sim
S19	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S20	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S21	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S22	Sim	Não	Sim	Sim	Sim	Sim	Sim
S23	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S24	Sim	Sim	Não Informado	Sim	Sim	Sim	Sim
S25	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S26	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S27	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S28	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S29	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S30	Sim	Não	Sim	Sim	Sim	Sim	Sim
S31	Sim	Sim	Sim	Não Informado	Não Informado	Sim	Sim
S32	Sim	Sim	Não Informado	Não Informado	Não Informado	Sim	Sim
S33	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S34	Sim	Sim	Não Informado	Não Informado	Não Informado	Sim	Sim
S35	Sim	Sim	Sim	Não Informado	Não Informado	Sim	Sim
S36	Sim	Sim	Sim	Não Informado	Não Informado	Sim	Sim
S37	Sim	Sim	Não	Não Informado	Não Informado	Sim	Sim
S38	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S39	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S40	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S41	Sim	Não	Não Informado	Não Informado	Não Informado	Sim	Sim
S42	Sim	Não	Sim	Sim	Sim	Sim	Sim
S43	Sim	Sim	Sim	Sim	Sim	Sim	Sim
S44	Sim	Não	Sim	Sim	Sim	Sim	Sim

A.3.2 Resultados em questões de pesquisa

Nesta seção apresentamos os resultados com base em cada questão de pesquisa pré-definida.

RQ.1:Para quais domínios do aplicativo LPS, tipos de solução e propostas TBM são usados?

Para responder a essa pergunta, levamos em consideração o TBM aplicado a domínios LPS (por exemplo, software, automotivo), tipos de solução de área de pesquisa LPS (por exemplo, modelagem de recursos) e o tipo de proposta de estudos (por exemplo, abordagem, técnica, ferramenta).

Identificamos sete domínios de aplicação distintos nos quais o TBM for LPS é aplicado. Tabela - 1.15 lista cada domínio e respectivos estudos e a Figura - 1.7 mostra o número de estudos por domínio.

Tabela 1.15: TBM de domínios de aplicativos LPS

Domínio de Aplicação	ID do Estudo	Count
Aeroespacial	S5, S8	02
Automotivo	S1, S3, S4, S7, S10, S11, S16, S17, S30	09
Eletrônico	S2, S9, S42, S44	04
Fabricação	S6	01
Medicina	S41	01
Software	T13, S15, S18, S19, S20, S21, S23, S24, S25, S26, S27, S28, S29, S31, S32, S33, S34, S35, T36, S37, S38, S39, S40, S43	24
Telecomunicação	S12, S14, S22	03
TOTAL		**44**

O domínio de aplicação *Software* tem a maioria das ocorrências com 24 estudos, seguido por *Automotive* com nove, que utiliza software embutido em seus produtos.

Aerospacial MPLM é uma das poucas ferramentas que concentram o maior esforço na questão da solução de variabilidade na geração de casos de teste, Samih e Bogusch (2014) em seu recurso de estudo a ferramenta Matelo, juntamente com outras ferramentas permite derivar a variante do modelo a partir de um o conjunto desejado e, assim, gerar casos de teste para um projeto experimental na Airbus defense & Space. O TBM auxilia na reutilização para geração dos casos de teste, o outro estudo que trabalha no mesmo domínio é uma extensão deste citado.

Automotivo Lity et al. (2012) apresenta a abordagem delta, que trabalha com elementos cruzados de um elemento chamado G, onde os casos de teste são definidos com uma cobertura satisfatória. Os artefatos de teste são desenvolvidos de forma incremental à

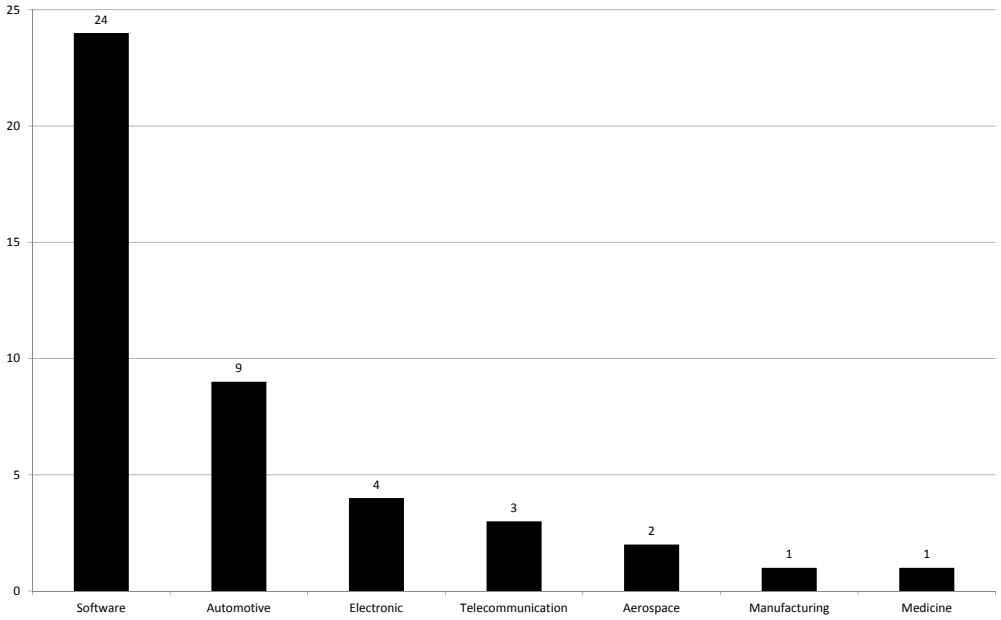


Figura 1.7: Automação TBM para LPS de abordagens de teste

medida que surgem novas variantes de produtos. Consideramos que o fator de regressão e a adaptabilidade dos artefatos de teste podem ter uma contribuição significativa ao serem trabalhados em um LPS, embora não tenham enfoque no TBM. O uso de software embarcado em produtos automotivos cresce exponencialmente, o TBM tem ajudado a gerar os testes desses produtos que sofrem variações, onde existe um núcleo central, e cada variação pode ou não conter uma especificidade única das demais.

Eletrônico Seguindo esse conceito Steffens et al. (2012) usa modelos para gerar subgrupos ou casos de teste desses modelos, aborda a questão da interação de recursos com a cobertura de teste e gerencia as variabilidades como itens cruzados. MoSo-PoLiTe, uma ferramenta que propõe fazer a derivação dos produtos a partir do modelo principal, utilizando o teste combinatório automatizado. O TBM atua na geração de casos de teste para subgrupos de produtos, além de buscar garantir uma maior cobertura de erros de funcionalidade, já que se trata de um domínio eletrônico, a garantia de qualidade deve ser com uma margem significativamente maior.

Fabricação Knapp et al. (2014) apresenta um procedimento associado a um conjunto de ferramentas para atribuir o resultado de um caso de teste a um membro arbitrário de um LPS, usando modelos de variabilidade baseados em UML e CVL, o modelo é usado para a criação de máquinas de estado são então convertidos em diagrama de classes.

Medicina Hasling et al. (2008) realiza a conversão dos modelos em diagramas de atividades e caso de uso, após esta conversão é elaborada a criação de casos de teste, onde as variantes dos modelos podem ser consideradas como variáveis. Ao usar o TBM, ele não menciona se a variabilidade é manipulada, simplesmente usada para criar casos de teste a partir de diagramas de atividades.

Software O software é um dos domínios mais relatados, inicialmente porque está diretamente ligado ao tópico de pesquisa, embora os demais domínios também utilizem o TBM no processo de desenvolvimento de algum produto de software específico para um determinado domínio.

Olimpiew (2008) também fornece muitos dados e informações sobre TBM em LPS, embora o foco seja em diagramas de atividades customizados, mas com foco na reutilização do teste em produtos de variante LPS. A abordagem proposta do CADeT transforma os casos de uso em diagramas de atividades, que por sua vez são transformados em casos de teste. As especificações de teste são rastreadas a partir do diagrama de atividades e do caso de uso.

A grande maioria dos estudos relata a criação de alguma abordagem ou ferramenta em TBM para auxiliar no desenvolvimento de LPS, no entanto, muitos não relatam com mais detalhes o uso de TBM em seu estudo.

Telecomunicação O TBM atua no auxílio da derivação de casos de teste de máquinas de estado, o desempenho neste domínio é devido ao uso de mídia embutida nos dispositivos de comunicação. Em um dado momento é possível interpretar que o estudo referente a esse domínio seria manufatura, mas tem uma área de atuação bem definida que seria a divisão de pesquisa de uma grande empresa de telecomunicações.

Ainda sobre o uso de ferramentas Wang et al. (2013a) retrata o uso de um processo automatizado de seleção de teste de forma mínima e sistematizado. Processos automatizados contribuem para algoritmos de seleção, por mais que estejam em uma manufatura, para criar uma visão sistêmica da seleção de variabilidade.

A Tabela - 1.16 lista os estudos realizados para testar diferentes tipos de solução de LPS, como: *Teste de recurso*, *Teste de modelo LPS* e *Teste PLA* (arquitetura de linha de produto - PLA). Para essa classificação, separamos o PLA dos artefatos gerais do LPS, devido à importância direta do PLA para o sucesso dos LPSs.

Podemos observar na Tabela - 1.16 a maioria dos estudos voltados para o teste de PLA. Isso ocorre principalmente devido ao papel central do PLA para o desenvolvimento de LPSs, uma vez que é uma abstração de todos os potenciais produtos únicos a serem

Tabela 1.16: Teste de TBM de tipos de solução LPS

Tipo de Solução	ID do estudo	Estudos
Feature Testing	S1, S8, S11, S14, S15, S17, S18, S26, S27, S29, S32	11
LPS Model Testing	T2, S3, S4, S5, S9, S12, S13, S19, S20, S21, S24, S30, S36, S37, S39	15
PLA Testing	T6, S7, S10, S16, S22, S23, S25, S28, S31, S33, S34, S35, S38, S40, S41, S42, S43, S44	18
TOTAL		**44**

desenvolvidos por um LPS. Portanto, 18 estudos (40,9 %) testaram o PLA e seus modelos. Os testes de PLA estão no nível de dados de entrada e saída, por meio de testes de caixa preta e testes de funcionalidade. Ao buscar a reutilização, há uma preocupação com a confiabilidade da arquitetura principal e com a derivação de produtos LPS.

Com relação ao teste de modelos de LPS, encontramos 34 % dos estudos levando em consideração qualquer tipo de nível de teste ou tipo para LPSs para engenharia de domínio e engenharia de aplicação. Testar modelos de LPS é um meio de fornecer reutilização de casos de teste para produtos específicos derivados de um LPS. Como apontado na literatura geral de testes, a detecção de defeitos nos modelos pode ser várias vezes menos custosa do que nas atividades posteriores do NPS. O teste nos modelos de LPS depende muito de qual nível será testado. Na maioria desses estudos, os testes dependem de funcionalidades, portanto, os modelos estão sendo testados de acordo com o resultado esperado dos requisitos especificados.

O teste de recursos é composto por 25 % dos estudos. Embora os recursos estejam no nível do espaço do problema, seus testes são essenciais para o sucesso de um LPS, pois os recursos estão diretamente relacionados aos modelos de LPS em todos os níveis, incluindo o código-fonte. O teste de recurso funcional está de acordo com a cobertura das ações esperadas. Assim, este é um grande desafio, pois com a derivação de novos produtos suas funcionalidades podem sofrer variações, o que pode causar uma combinação exponencial de sequências de teste.

Por uma questão de contribuições de estudos, analisamos e apresentamos na Tabela - 1.17 em termos de seus tipos de propostas.

Podemos observar que a maioria dos estudos (33) apresenta como uma contribuição de proposta uma abordagem. Os estudos restantes relatam de uma a três propostas cada.

RQ.2:Quais abordagens de TBM, níveis de teste, artefatos, ferramentas e modelos foram usados para testar LPSs?

Nesta seção, fornecemos resultados sobre abordagens, níveis de teste, artefatos, ferramentas e modelos usados para aplicar o TBM às LPSs.

Tabela 1.17: Propostas principais de TBM para LPS

Tipo de proposta	ID do estudo	Estudos
Abordagem	T1, S5, S6, S7, S8, S9, S10, S11, S13, S14, S15, S16, S17, S18, S20, S21, S22, S23, S24, S27, S28, S29, S30, S31, S35, S36, S37, S38, S39, S41, S42, S43, S44	33
Algoritmo	S3, S25, S26	03
Kit de ferramentas	S40	01
Estratégia	S33	01
Abordagem de Extensão	S12	01
Ferramenta	S2, S4, S34	03
Metodologia	S19	01
Tutorial	S32	01
TOTAL		**44**

Os estudos analisados realizam principalmente testes de tempo de projeto. Além disso, para esta questão de pesquisa, analisamos os testes:abordagens, níveis e tipos e automação.

Abordagens Usadas A partir de estudos selecionados, criamos duas abordagens baseadas em ”caixa”:caixa branca e caixa preta. O primeiro é quando alguém tem acesso ao código-fonte e pode fornecer análises com maior profundidade. Este último é realizado quando não se tem acesso ao código, mas apenas para inserir dados e parâmetros, assim como os resultados produzidos.

Tabela - 1.18 apresenta os estudos classificados em cada uma das abordagens.

Tabela 1.18: Abordagens de TBM para LPS

Abordagem de Teste	ID do estudo	Estudos
Caixa-Preta	S2, S3, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S32, S33, S34, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44	39
Caixa-Branca	S1, S4, S20	03
Não Informado	S5, S6	02
TOTAL		**44**

Podemos observar que a maioria dos estudos (39/44) executa testes de caixa preta, principalmente porque:os testes são realizados em uma abstração de nível superior, como, por exemplo, testes funcionais; e LPS na atividade de engenharia de domínio.

A grande maioria dos estudos procura gerar casos de teste por meio de modelos, portanto, esse é um dos fatores que levam ao uso da abordagem da caixa preta. Steffens et al. (2012) use modelos para gerar casos de teste. Esta prática é feita por quase todos os outros estudos (Cichos et al., 2011; García Gutiérrez et al., 2010; Lackner et al., 2014; Oster et al., 2011a; Samih et al., 2014b).

Níveis de teste Encontramos diferentes níveis e tipos de testes TBM. Da Lista Final, quatro estudos se destacam mais (Tabela - 1.19 [2]), Que são:*Integração*, *Sistema* , *Regressão* e *Funcional*. A Figura - 1.8 descreve os níveis e tipos de TBM de LPSs.

Tabela 1.19: TBM dos níveis e tipos de testes de LPS

Nível / Tipo de Teste	ID do estudo	Estudos
Combinação	S4	01
Estrutural	S20	01
Funcional	S6, S9, S10, S22, S24, S28, S31	07
Performance	S20	01
Regressão	S1, S11, S13, S14, S15, S17, S18, S27, S30,	09
Sistema	S25, S26, S33, S34, S35, S36, S38, S39, S40, S41, S42, S43, S44	13
Integração	S3, S5, S7, S8, S11, S13, S14, S15, S16, S17, S18, S19, S21, S23, S29, S37	16
Unitário	S2	01
Não Informado	S12, S32	02
TOTAL		**51**

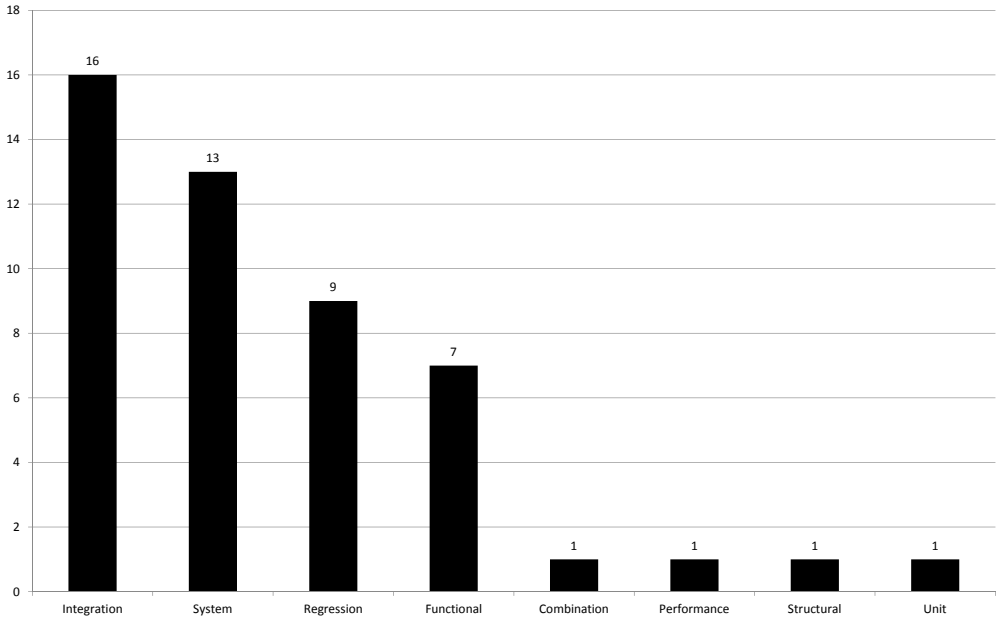


Figura 1.8: Comparação de níveis e tipos de TBM para LPS

Podemos observar que o nível de teste de integração é um dos mais considerados, pois a maioria dos testes é realizada na engenharia de domínio LPS. Além disso, para a reutilização de muitos casos de teste, a derivação de novos produtos requer uma verificação de integração entre modelos (Cai e Zeng, 2013; Cichos et al., 2011; Devroey, 2014; Devroey et al., 2014a; Dukaczewski et al., 2013; García Gutiérrez et al., 2010; Olimpiew e Gomaa,

[2]Um estudo pode suportar mais de um nível ou tipo de teste.

2005; Oster, 2012; Oster et al., 2011a; Samih e Baudry, 2012; Samih e Bogusch, 2014; Samih et al., 2014b; Varshosaz et al., 2015; Wang et al., 2013a) .

Automação de Testes Ao analisar os estudos selecionados, identificamos o uso de ferramentas totalmente automatizadas e semi-automatizadas para auxiliar no teste de TBMs de LPSs. Também identificamos estudos sem auxílio de ferramentas, desenvolvendo manualmente atividades de testes, como podemos observar na Tabela - 1.20.

Tabela 1.20: TBM de automação LPS

Abordagem de automação	**ID do estudo**	**Estudos**
Totalmente automatizado	S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S20, S22, S23, S24, S26, S27, S28, S31, S40, S41	27
Semi-automatizado	S19, S21, S25, S29, S30, S34, S36, S37, S38, S39, S42, S43, S44	13
Manual	S1, S33, S35	03
Não Informado	S32	01
TOTAL		**44**

Nãos estudos selecionados, verificamos o uso de ferramentas *fully-automated* em atividades de teste. Segundo estudos selecionados, a maioria deles utiliza algum tipo de automação para realizar tais atividades. As ferramentas usadas são amplamente implementadas pelos autores dos estudos (Devroey et al., 2015; Oster et al., 2011b; Perrouin et al., 2010; Steffens et al., 2012), mas esses autores também fornecem comparações entre ferramentas de abordagens semelhantes (Devroey et al., 2015; Oster et al., 2011b; Perrouin et al., 2010; Rodrigues, 2013; Steffens et al., 2012).

Outro conjunto de estudos selecionados usa ferramentas *semi-automated*, nas quais apenas um trecho do processo de teste é automatizado, com ou sem atividades manuais. O Olimpiew (Olimpiew e Gomaa, 2005) cria casos de uso para permitir especificações de casos de teste reutilizáveis pelo arquiteto de software. De acordo com Weissleder e Lackner (Weißleder e Lackner, 2013), os modelos são usados em conjunto para modelar recursos e comportamento para a geração de diagramas de máquinas de estado e comportamento por engenheiros de software usando abordagens semi-automatizadas.

Estudos puramente manuais não adotam nenhum tipo de automação das atividades de teste. Tais estudos discutem abordagens teóricas relacionadas com TBM de LPS e, portanto, foram classificados como *Manual*. Em geral, as atividades manuais seriam a conversão de um modelo inicial em um modelo secundário com maior detalhe, depois para casos de teste (Beohar e Mousavi, 2014b; Lity et al., 2016; Lochau e Kamischke, 2012). As abordagens manuais tendem a apresentar apenas conceitos teóricos do modelo de processo desenvolvido, ou buscam apresentar a essência referente a um contexto específico.

O gráfico de bolha da Figura - 1.9 mostra a automação de abordagens. Como podemos observar, o teste caixa-preta tem maior automação, tanto por ferramentas totalmente automatizadas quanto semi-automatizadas, totalizando 39 estudos. As abordagens de caixa branca são menos executadas com o auxílio de ferramentas (semi) automatizadas, apenas três estudos.

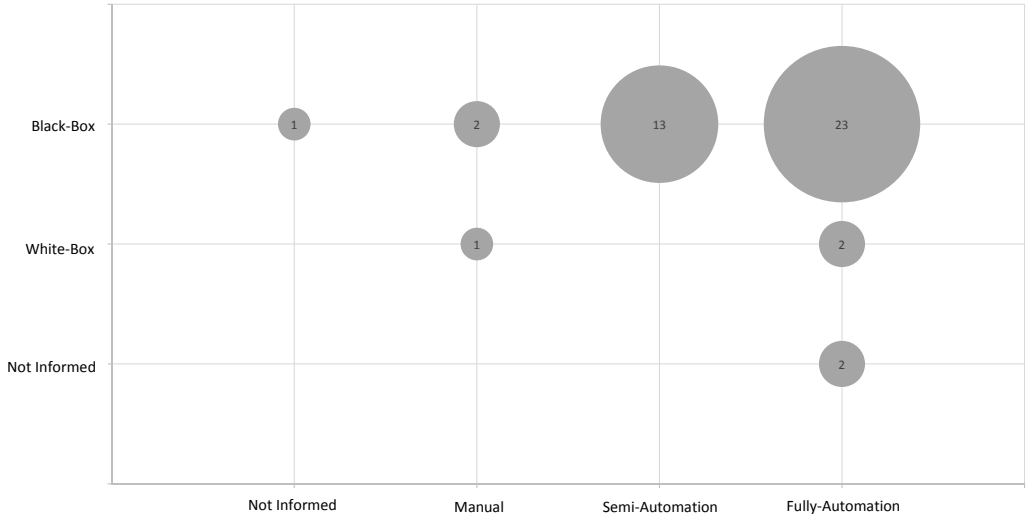


Figura 1.9: Automação TBM para LPS de abordagens de teste

O gráfico de bolha da Figura - 1.10 mostra a automação dos níveis e tipos de teste.

Treze estudos são menos realizados com o auxílio de ferramentas (semi) automatizadas. Apenas três estudos não apresentam nenhum tipo de automação, enquanto um estudo não relata.

Artefatos Usados Para gerar casos de teste, os estudos de TBM usam vários artefatos diferentes, como Máquina de estados, Diagrama de classes, Diagrama de sequência e Modelo de recursos, como podemos observar na Tabela - 1.21. Uma comparação de tais artefatos é fornecida na Figura - 1.11.

Alguns desses artefatos desempenham o papel do artefato original (inicial) em teste, que são os artefatos dos quais os casos de teste devem ser gerados. Outros artefatos são usados como elementos intermediários para fornecer ao TBM um meio de testar um determinado modelo original.

Ferramentas usadas Figura - 1.12 apresenta ferramentas usadas para executar TBMs de LPSs relacionadas a artefatos. Algumas dessas ferramentas são mais frequentes, pois são usadas em mais de um estudo.

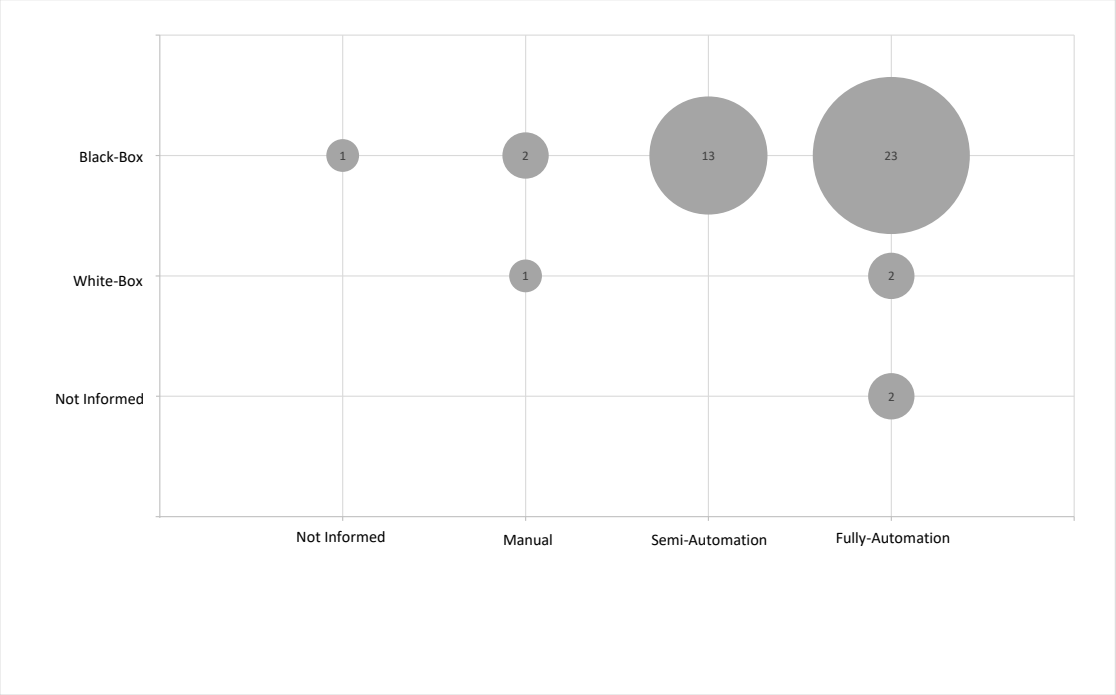


Figura 1.10: Automação de níveis de teste de TBM para LPS

Tabela 1.21: Artefatos Usados Durante TBM de LPS

Artefato	**ID do estudo**	**Estudos**
Máquina de Estado	S1, S6, S7, S10, S11, S12, S13, S14, S15, S17, S21, T22, S23, S27, S29, S30, S34, S38, S44	19
Modelo original	S8, S9, S42	03
OVM Variability Model	S8	01
Diagrama de Classes	S6, S12, S14, S31	04
Diagrama de atividades	S18, S19, S41	03
Diagrama de casos de uso	S20, S28, S41, S43	04
Diagrama de sequência	S24, S31, S36	03
Diagrama de Transição	S7, S15, S25, S26, S33, S35, S37	07
Diagrama de Funções	S2, S3, S4, S5, S9, S16, S39, S40	08
Não Informado	S32	01
TOTAL		**53**

Podemos citar algumas ferramentas com um número maior de citações, como a série Rational IBM, o gerenciador de linha de produtos MaTeLo, variantes PURE, ferramentas CADeT e Eclipse (Beohar e Mousavi, 2014b; Costa, 2016; Lity et al., 2012; Olimpiew e Gomaa, 2005; Samih e Bogusch, 2014).

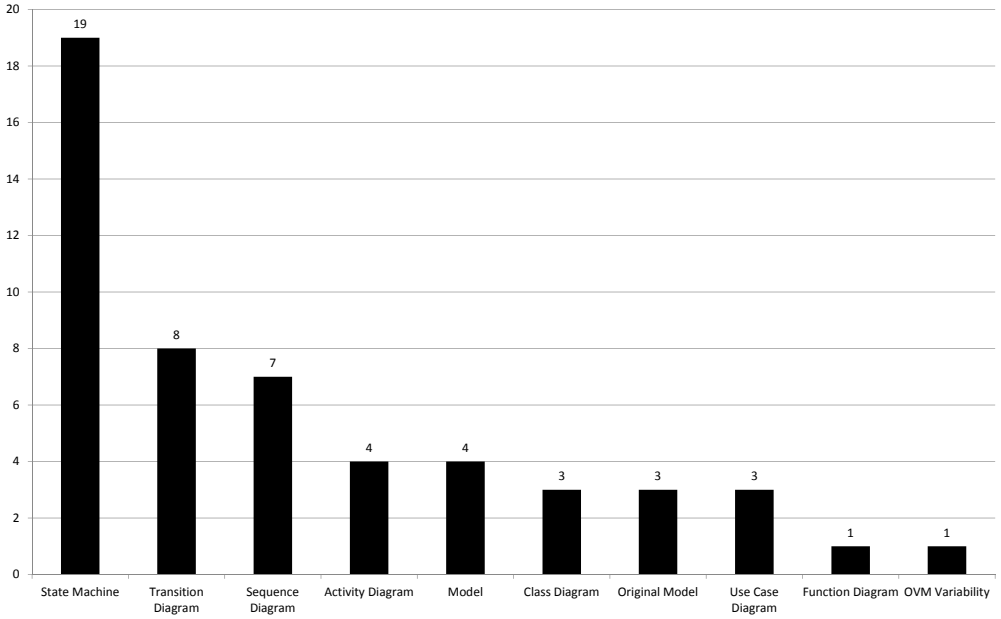


Figura 1.11: Automação TBM para LPS de abordagens de teste

As ferramentas CADeT (Olimpiew e Gomaa, 2005) ajudam na geração de casos de teste a partir de modelos, bem como na especificação de casos de teste reutilizáveis. Fazendo uso do Delta para extrair os casos de teste com foco em regressão, esta ferramenta, bem como as ferramentas IBM Rational, Eclipse (Lochau et al., 2012b) e MoSo-PoLiTe foram consideradas para resolver o mesmo problema.

Tabela 1.22: Comparação de ferramentas e artefatos primários ou secundários

Ferramenta	Máquinas de Estado	Direto do Modelo	Diagrama de Classe	Statechart	Diagrama de Atividade	Diagrama de Sequência	Diagrama de Transição	Diagrama de Funcionalidade	Caso de Uso
IBM Rational Rhapsody	✓								
MoSo-PoLiTe		✓		✓					
MaTeLo Product Line Manager		✓							
LPSOT tool		✓						✓	
LPSTestbench	✓								
PURE variantes	✓		✓	✓					
UML2 Eclipse	✓								
CVL-Too	✓								
Rational Software Architect	✓		✓						
MOFLON - SDM		✓							
Rhapsody				✓					
CADeT Tools					✓				
TRUST tool	✓								
Selenium									✓
AspectOPTIMA								✓	
Quick Test Pro									✓
Rational Functional Tester									✓
Pralíntool			✓			✓			
VIBeS	✓								
Eclipse	✓					✓			
UML2 Tools						✓			
ViTAL							✓		
visio	✓								
ATG	✓								
Excel	✓								

Modelos LPS Quanto ao tipo de modelo de LPS considerado entre os trabalhos analisados, a maioria foi feita usando *Behavioral-based*, que se caracteriza por ser comportamental, talvez a razão para isso, seja porque os modelos são ricos em detalhes, facilitando interpretação ou configuração, mesmo se o modelo é posteriormente convertido em um artefato, primário ou intermediário. Na Tabela - 1.23 pode-se observar que vários trabalhos utilizam o modelo de modelos baseados em Cenário, a relação se deve ao fato de estes trabalhos fazerem uso de artefatos intermediários, trabalhando com cenários de uso ao invés de comportamento direto.

Tabela 1.23: TBM de LPS Tipos de Modelos

Modelo LPS Considerado	**Referência**	**Estudos**
Scenario-based	S3, S11, S19, S20, S24, S31, S38, S40, S41, S42, S43, S44	12
Class-oriented	S6	01
Behavioral-based	S1, S2, S4, S7, S8, S9, S10, S12, S13, S14, S15, S16, S17, S18, S21, S22, S23, S25, S26, S27, S28, S29, S30, S33, S34, S35, S36, S37, S39	29
Não Identificado	S5, S32	02
TOTAL		**44**

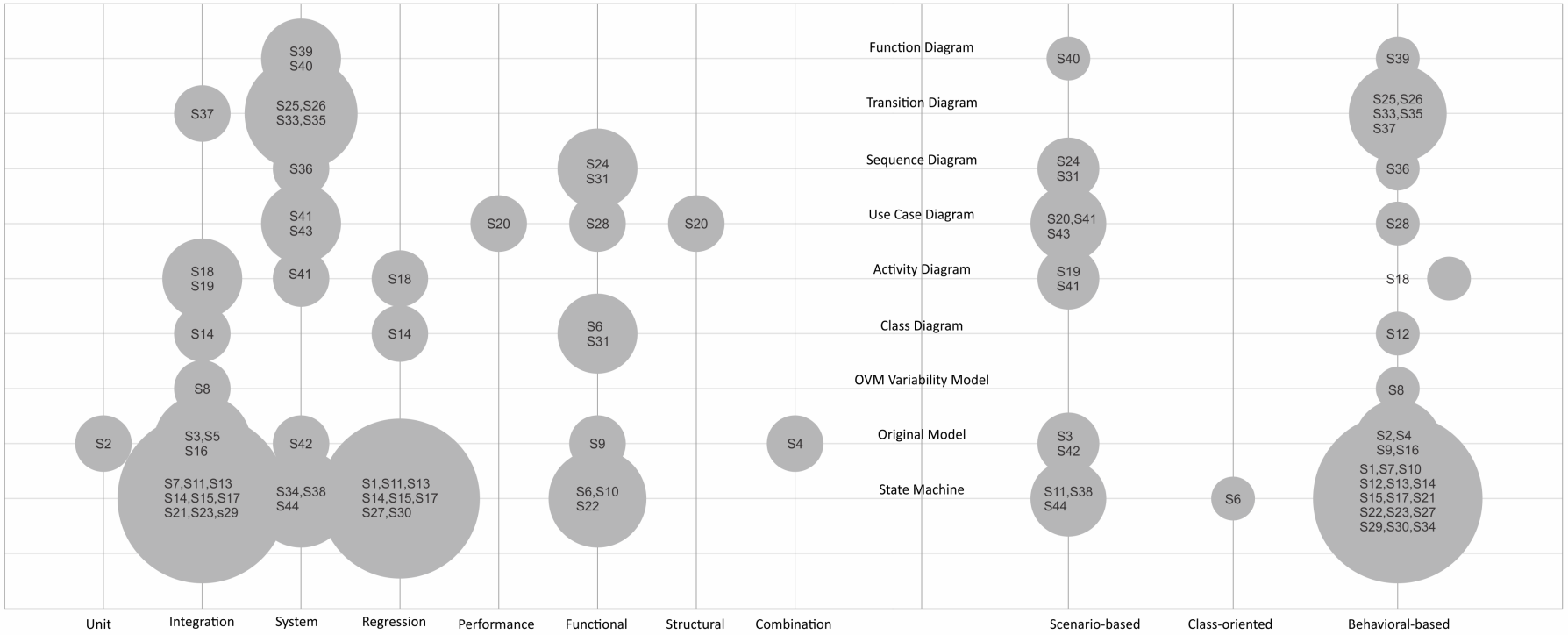


Figura 1.12: Artefatos X Níveis X Modelos

RQ.3:Como a variabilidade e o tempo de ligação são tratados durante o TBM de LPSs?

Como a variabilidade é a essência da LPS, assim como é diretamente relacionada ao tempo de vinculação, nesta seção apresentamos resultados nos quais os estudos levam em conta tais conceitos e como eles os tratam.

Tabela - 1.24 classifica os estudos que:consideram a variabilidade (linha *" Sim "*) durante o TBM do LPS e não consideram a variabilidade (linha *" Não "*). Em vários estudos, não conseguimos identificar se as variabilidades são tratadas.

Tabela 1.24: Estudos que tratam da variabilidade durante as atividades da TBM

Trata a variabilidade?	ID do Estudo	Estudos
Sim	S1, S2, S4, S5, S6, S7, S8, S9, S10, S12, S13, S14, S15, S16, S17, S18, S19, S21, S22, S24, S25, S27, S28, S29, S30, S31, S33, S34, S37, S40	30
Não	S39, S43	02
Não Identificado	S3, S11, S20, S23, S26, S32, S35, S36, S38, S41, S42, S44	12
TOTAL		**44**

Apenas dois estudos (S39, S43) não tratam a variabilidade durante atividades de TBM. Isso ocorre porque eles aplicam testes em produtos específicos de LPS na engenharia de aplicativos.

Como podemos observar na Tabela - 1.24, 68,1 % (30/44) dos estudos tratam a variabilidade durante qualquer atividade de TBM de LPSs.

Muitas estratégias diferentes são usadas para lidar com a variabilidade durante o TBM. A variabilidade pode ser modelada e resolvida em diagramas UML, com a geração de diagramas de classes e máquinas de estados (Wang et al., 2013a) a serem testadas.

Idiomas específicos também podem ser usados para especificar a variabilidade dos casos de teste (Devroey, 2014). A Figura - 1.13 resume quais estudos tratam ou não a variabilidade por tipo de solução.

O tempo de ligação é a capacidade de resolver a variabilidade em qualquer ponto do ciclo de vida do LPS. É obrigatório o aumento significativo dos níveis de variabilidade (Chen et al., 2009).

O tempo de vinculação no LPS é tradicionalmente em tempo de design, pré-compilação, compilação ou tempo de vinculação, lidando com a variabilidade estática. Já nos sistemas que lidam com a variabilidade dinâmica, pode ser em tempo de carregamento quando um sistema é implementado e carregado na memória e, em tempo de execução, após o sistema ter iniciado a execução de (Alves et al., 2009).

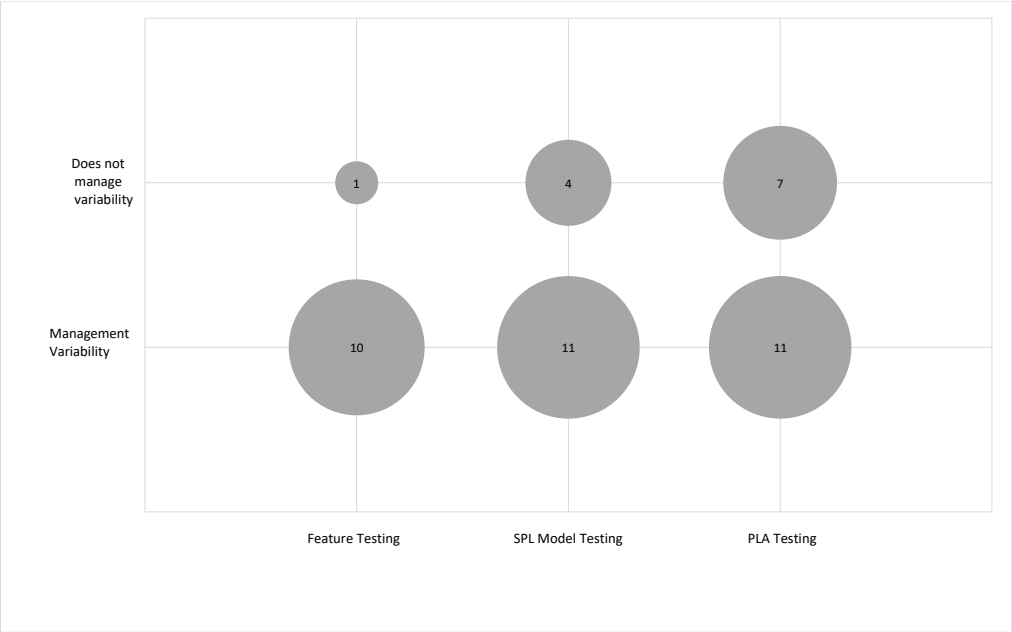


Figura 1.13: Tratamento da variabilidade por tipo de solução

Tabela - 1.25 lista estudos dos quais 86.3 % (38/44) lidam com a variabilidade de vinculação em tempo de design. Os estudos restantes não informam seus tempos de ligação.

Tabela 1.25: Tempo de Ligação de Variabilidade em TBM de LPS

Binding Time	Id do Estudo	Estudos
Design Time	S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S32, S33, S34, S35, S36, S37, S38, S39, S40, S41, S42, S43, S44	38
Não Identificado	S1, S2, S3, S4, S5, S6	06
TOTAL		**44**

RQ.4:O TBM suporta testes de requisitos não funcionais do LPS?

Nenhum dos 44 estudos selecionados deste MSL apresenta ou menciona qualquer tipo de teste de requisitos de LPS não funcional. Embora o LPS e os produtos testados estejam geralmente em tempo de design, nada impede a criação dos requisitos não funcionais do plano de teste (Lee et al., 2012).

Feng et al. (2007) e Reis et al. (2007), por exemplo, propõe a criação de casos de teste para a variabilidade considerando requisitos não-funcionais. (Kakarontzas et al., 2008) propõe casos de teste hierárquicos para requisitos não funcionais específicos.

RQ.5:Como as propostas de TBM de LPS são avaliadas?

Analisamos a avaliação da proposta de estudos de TBM de LPS com base no tipo de avaliação e no ambiente em que as avaliações são realizadas.

Tabela - 1.26 lista estudos de acordo com seus tipos de avaliação, que são:*Experiment* e *Case Study*. Dos estudos selecionados, 79,5 % (35/44) deles realizaram um desses tipos de avaliação. Em nove estudos, nem a avaliação foi realizada nem eles puderam ser identificados.

Tabela 1.26: Método de Evidenciação da Solução TBM para LPS

Método de avaliação	ID do Estudo	Estudos
Experimento	S1, S3, S4, S16, S20, S21, S23, S24, S25, S28, S39, S40, S42	13
Estudo de Caso	S2, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S17, S19, S22, S26, S27, S29, S30, S33, S38, S43, S44	22
Não Informado	S5, S18, S31, S32, S34, S35, S36, S37, S41	09
TOTAL		**44**

Os experimentos são responsáveis por 29,5 % (13/44) das avaliações realizadas. Nós não percebemos qualquer replicação experimental como um meio de impor resultados originais em uma determinada proposta. Além disso, poucos trabalhos apresentam elementos experimentais essenciais, como validade do estudo, perfil dos participantes, formulação e teste de hipóteses. Embora a maioria dos estudos relacionados à indústria não use participantes humanos, o tamanho da amostra não é relatado na maioria dos estudos. Cerca de 60 % dos estudos disponibilizam seu pacote experimental ou parte dele, incluindo, por exemplo, diagramas e algoritmos, por meio de uma URL pública. Não entanto, a maioria das URLs não está mais disponível . Acreditamos que isso ocorre porque a maioria das avaliações é realizada em conjuntos de setores (consulte Tabela - 1.27).

Os estudos de caso são o tipo de avaliação mais realizado das propostas de TBM para LPS, com 50 % (22/44) dos estudos e podem ter questões confidenciais.

Um aspecto importante relacionado à avaliação de propostas é o ambiente em que tais avaliações ocorrem. Portanto, analisamos quais estudos são realizados na academia e quais são realizados na indústria. Tabela - 1.27 lista estudos por ambiente de avaliação.

A maioria dos estudos ocorre na indústria, 59 % (26/44), o que é de grande importância devido ao seu realismo e participação de profissionais de testes reais. Além disso, o

Tabela 1.27: Ambientes de evidência de soluções TBM para LPS

Environment	Study ID	Count
Industria	S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S20, S22, S29, S30, S38, S41, S42, S43, S44	26
Academia	S19, S23, S24, S25, S26, S27, S28, S31, S32, S33, S34, S35, S36, S37, S39, S40	16
Não Informado	S18, S21	02
TOTAL		**44**

conjunto da indústria fornece dados reais de projetos de LPS. Por outro lado, em estudos realizados em ambiente acadêmico, geralmente recruta os estudantes como participantes e usam NPSs pedagógicos ou não comerciais. Nãote que o uso de estudantes na engenharia de software baseada em evidências não é uma ameaça tão amplamente discutida (Feldt et al., 2018).

A Figura - 1.14 mostra um gráfico de bolhas dos tipos de avaliação e seus ambientes.

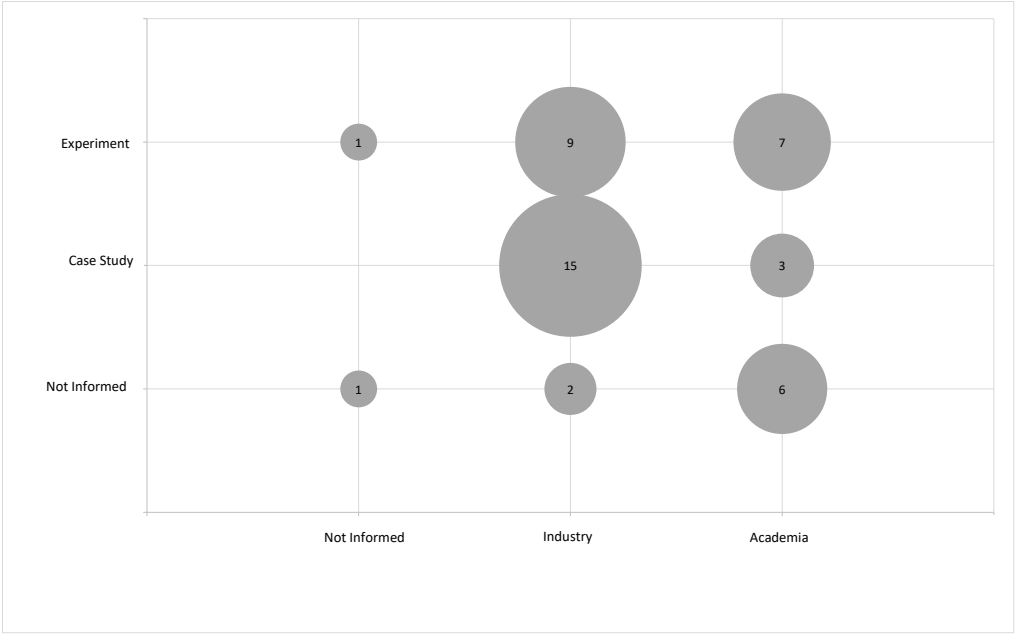


Figura 1.14: Tipos de Avaliação de Propostas e Ambientes

Como podemos observar na Figura - 1.14, a maioria dos estudos, nove experimentos e 15 estudos de caso, foram avaliados no conjunto da indústria, o que contribui para aumentar a confiabilidade das evidências das propostas fornecidas.

RQ.6:Como a rastreabilidade é considerada durante atividades de TBM para LPSs?

A rastreabilidade ainda é uma das muitas lacunas de pesquisa no desenvolvimento de software, especialmente para LPSs. Como a rastreabilidade é responsável por identificar elementos importantes para derivar produtos específicos de LPS e para a evolução do LPS, é diretamente necessário durante as atividades do TBM (Bernardino et al., 2017; Costa, 2016). Além disso, a rastreabilidade fornece suporte essencial para garantia de qualidade de LPSs.

Nesta seção, analisamos quais estudos levam em consideração qualquer tipo de rastreabilidade durante as atividades de TBM. Assim, Tabela - 1.28 lista os estudos e se fornecem algum suporte para a rastreabilidade.

Tabela 1.28: Estudos com Suporte de Rastreabilidade para TBM de LPSs

Suporte a Rastreabilidade?	**ID do estudo**	**Estudos**
Sim	S5, S7, S15, S16, S17, S19, S20, S24, S31, S36, S37, S41	12
Não	S1, S2, S3, S4, S6, S8, S9, S10, S11, S12, S13, S14, S18, S21, S22, S23, S25, S26, S27, S28, S29, S30, S32, S33, S34, S35, S38, S39, S40, S42, S43, S44	32
TOTAL		**44**

Poucos estudos (12/44) relatam ou mencionam a rastreabilidade como suporte. Além disso, nenhum deles declara como eles usam ou tratam a rastreabilidade. Eles só alegam que a rastreabilidade é fundamental para a geração de casos de teste, que devem ser o mais automatizados possível.

A.3.3 Procedimentos de compartilhamento de dados do MSL

Realizamos procedimentos para promover a abertura e a reprodutibilidade deste estudo. Portanto, nós:

- fornece um formato de arquivo CSV com dados de todos os estudos deste MSL;
- fornece um formato de arquivo de metadados CSV explicando o arquivo CSV dos estudos;
- fornece arquivos PDF para todas as figuras e gráficos deste documento;
- fornece o formato de arquivo TEX para todas as tabelas deste documento;
- fornece formato de arquivo BIBTEX e CSV com referências dos estudos da Lista Final para uso em diferentes ferramentas bibliográficas;

- Fornece item de dados (CSV, PDF, TEX, arquivos BibTeX) em um repositório confiável compartilhamento de dados público aberto e permanente, chamado Zenodo [3], sob DOI `xxxx/aaaa/zzzz`.

A.4 TBM4LPS:um roteiro para testes baseados em modelos de pesquisa de LPSs

Esta seção apresenta um roteiro, denominado TBM4LPS, como resultado do mapeamento da Lista Final de estudos com relação às fases do processo de TBM para LPSs. Três fases são consideradas:Baseado em artefato, Gerenciamento e Verificação de Variabilidade e Execução de TBM.

A Figura - 1.15 mostra o TBM4LPS sobre como os estudos são divididos de acordo com as fases e atividades do TBM. Para ilustrar o readomap, duas rotas serão apresentadas:a primeira baseada principalmente em um estudo primário específico, guia os usuários do modelo de estudo até a geração de casos de teste; e o segundo, considerando as três fases e atividades do TBM para LPS, dando aos usuários estudos que suportam tais fases.

[3] `zenodo.org`

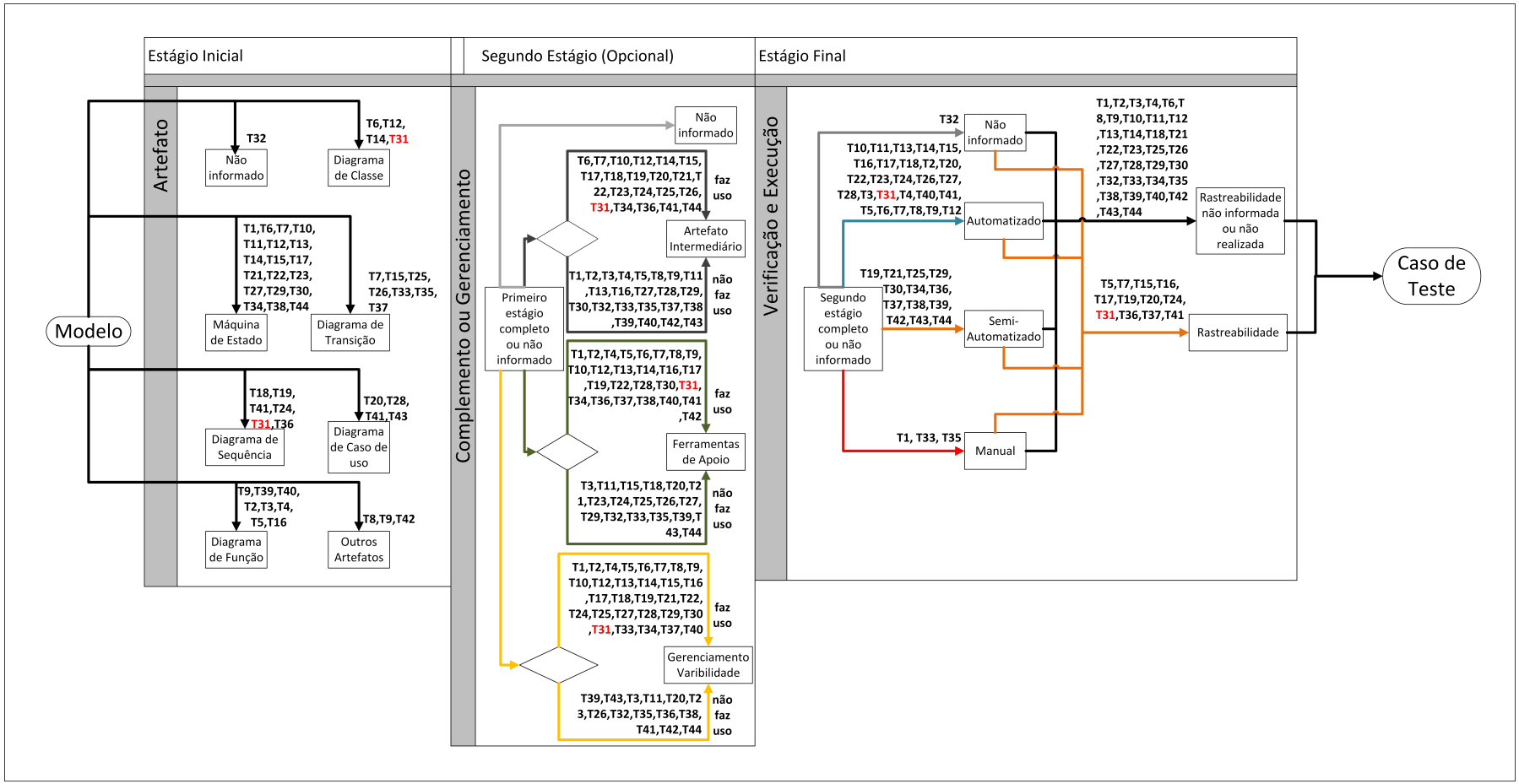


Figura 1.15: TBM4LPS: um roteiro de TBM para LPSs

As seções a seguir apresentam exemplos de aplicativos sobre como usar o TBM4LPS.

A.4.1 Rota Primária Baseada no Estudo

Digamos que, ao selecionar um estudo de Tabela - 1.12 do seu ID, você queira verificar o Figura - 1.15 qual caminho percorre, identificando seus estágios e atividades. Assim, inicialmente analisamos o primeiro estágio representado em Figura - 1.16, que é um excerto de Figura - 1.15.

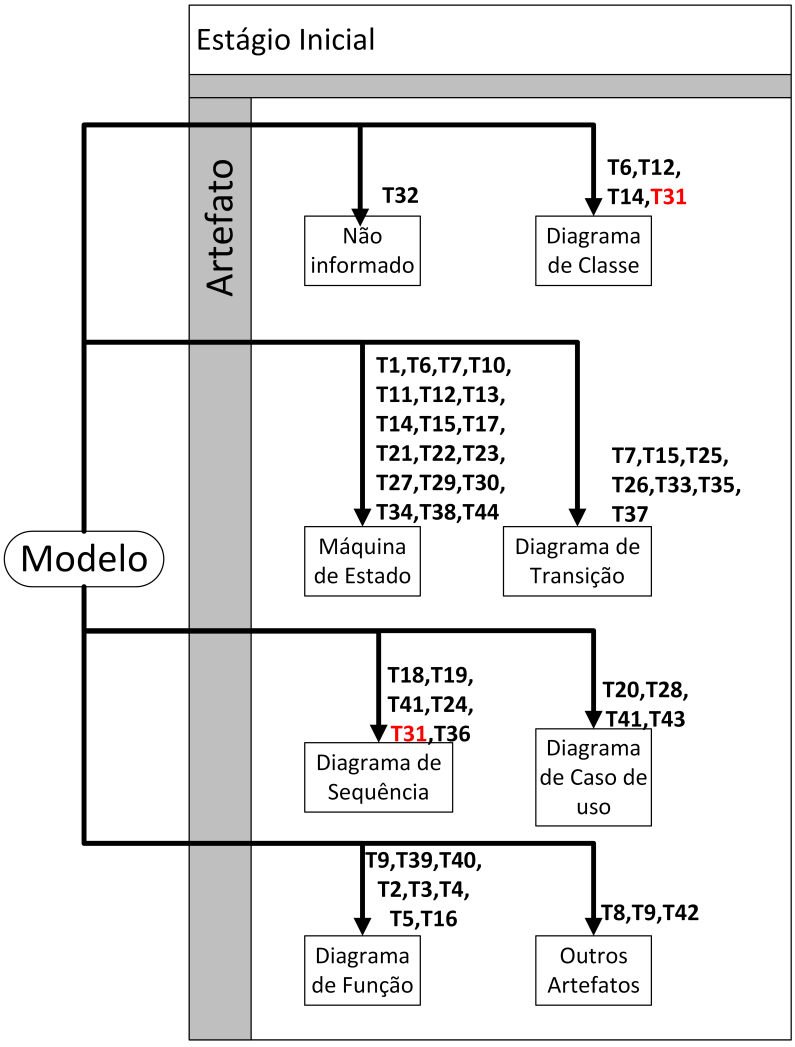


Figura 1.16: Etapa inicial, mostrando a localização do estudo T31

Tomemos, por exemplo, o estudo T31. A partir de um modelo, o T31 faz uma conversão para o artefato do diagrama de sequência, então podemos considerar o T31. Com o estágio inicial concluído, o segundo estágio (opcional) é iniciado e o T31 considera o gerenciamento de variabilidade, suporte de ferramentas e geração de artefatos

intermediários, pois nesse estágio abordamos o gerenciamento de variabilidade, uso de ferramentas, geração de artefatos intermediários e se não executar qualquer uma dessas atividades automaticamente, pode ser na geração do caso de teste.

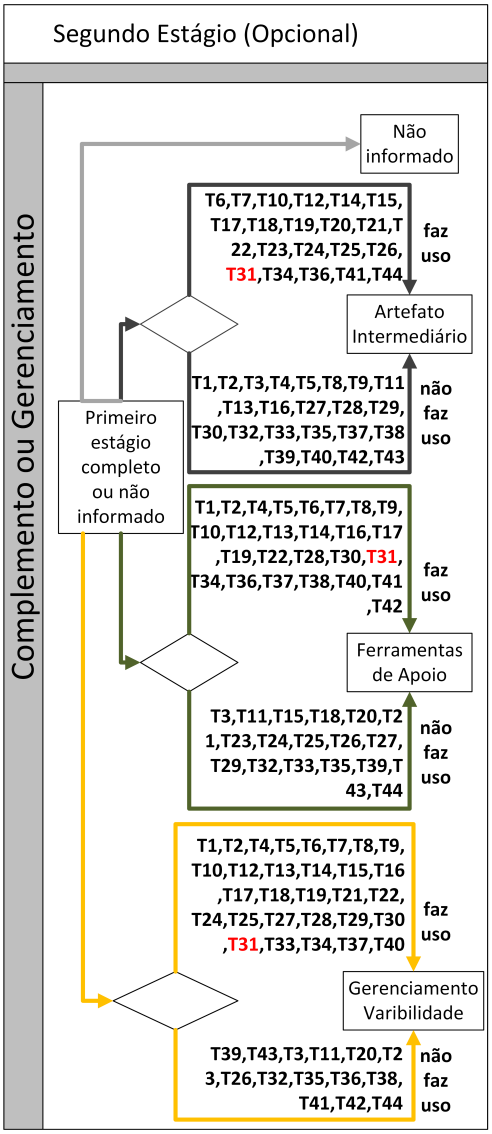


Figura 1.17: Segunda etapa, mostrando a localização do estudo T31

Ao apresentar o estágio final em Figura - 1.18 pode-se analisar que o estudo T31 faz uso de processo automatizado e menciona a rastreabilidade. Isso conclui o curso do guia para o estudo T31, identificando, assim, quais etapas o estudo passou.

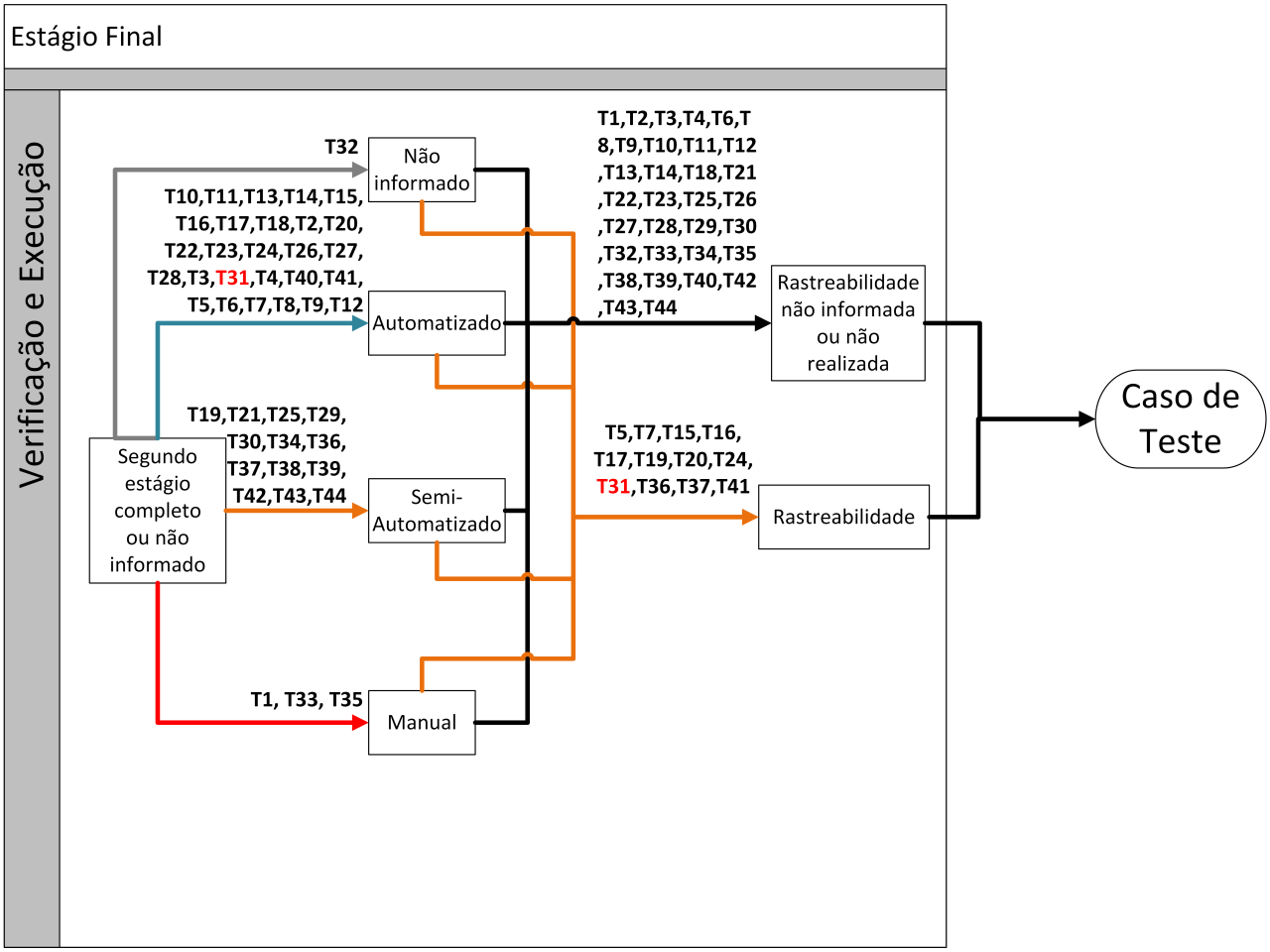


Figura 1.18: Etapa final mostrando a localização do estudo T31

A.4.2 Rota baseada em critérios

O segundo cenário executa o tema TBM no LPS por assunto. Ao analisar o Figura - 1.18, por exemplo, pode-se verificar quais trabalhos fazem uso da rastreabilidade. Portanto, se o interesse do leitor é por obras que fazem uso da rastreabilidade, basta consultar essa atividade para obter a lista de tais obras.

A.5 Observações Conclusivas

Este mapeamento sistemático foi realizado para obter uma visão geral sobre testes baseados em modelo para linhas de software, e identificamos quarenta e quatro trabalhos publicados entre 2006 e 2016. Há um desafio muito grande ao abordar o teste em LPS,

onde os principais pontos levantados são o gerenciamento de variabilidade, a cobertura de erros, a automação de processos e a rastreabilidade.

O objetivo principal da pesquisa foi verificar se o TBM é aplicado no NPS e, quando aplicado, se o gerenciamento de variabilidade foi utilizado. Após a análise dos dados extraídos do MSL, foi possível verificar e identificar as técnicas de teste de TBM utilizadas, que utilizam a notação UML, os trabalhos que realizam um controle de variabilidade, bem como a menção de rastreabilidade. Além disso, como os casos de teste são gerados, manualmente ou automatizados.

As contribuições para o TBM em LPS não são tão detalhadas quanto esperávamos, o que exigiu um trabalho extra na interpretação dos dados extraídos. Os principais dados que o MSL apresenta como contribuição são os artefatos utilizados, as técnicas relatadas, o uso de ferramentas e o uso do gerenciamento de variabilidade.

Com esse MSL, podemos ver que o domínio de maior desempenho é o software, a grande maioria considera o uso da variabilidade e a maior parte das validações estão sendo realizadas no setor, isso mostra que há um crescente interesse no assunto, embora não podemos identificar por que a rastreabilidade e os testes de requisitos não funcionais não são explicitamente apresentados, estudos adicionais sobre esses dois temas podem nos dizer mais sobre isso.

Um dos principais pontos que é a gestão da variabilidade atrelada ao modelo em LPS foi apresentado por uma maioria significativa, isso demonstra que há uma crescente conscientização da qualidade dos produtos variantes oriundos de um produto principal.

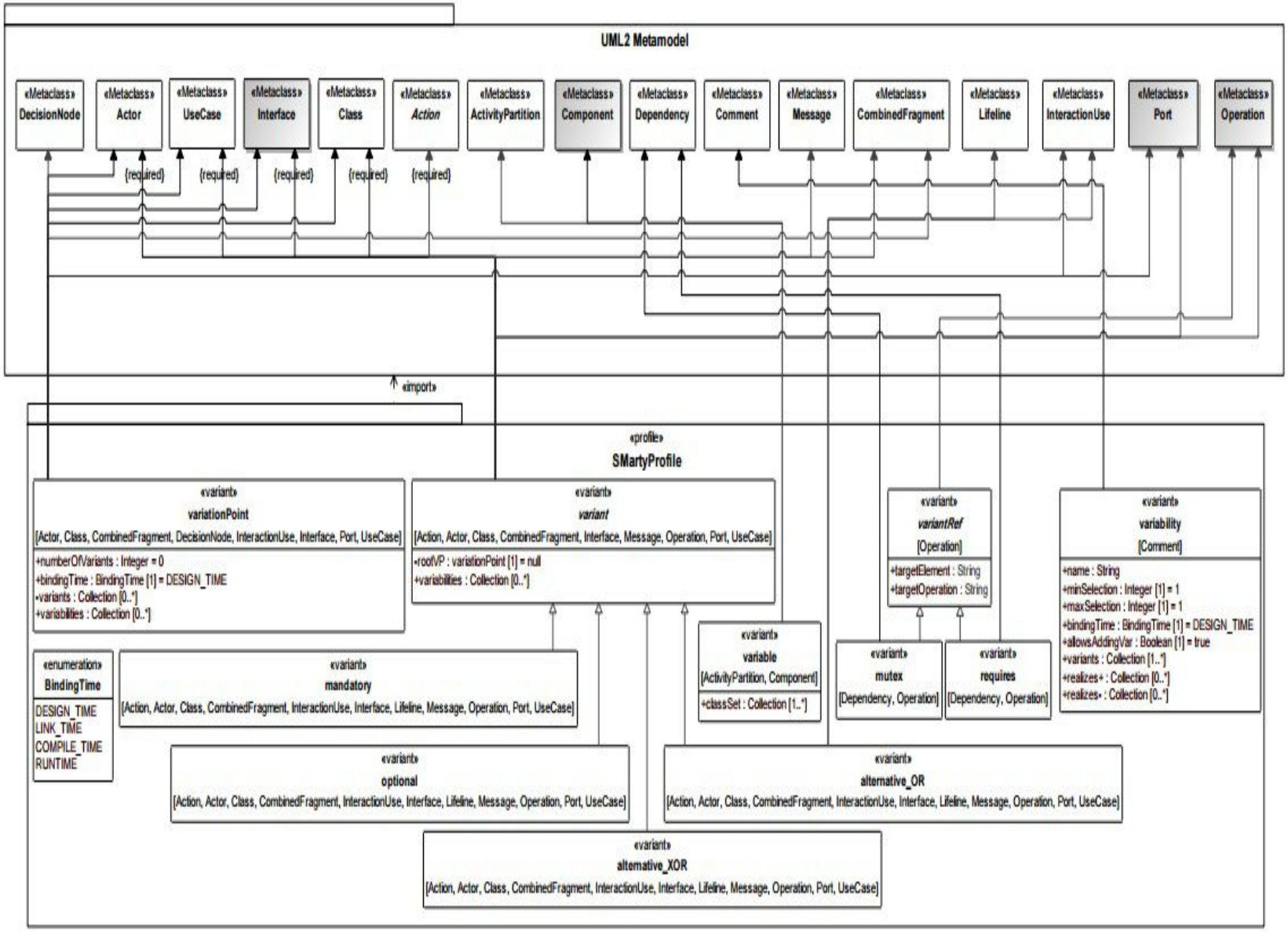
Devido a esse crescente interesse, incentivamos a comunidade a continuar com novas pesquisas para testes baseados em modelo, a fim de apresentar dados sobre rastreabilidade, automação de processos, cobertura de erros e gerenciamento de variabilidade. Para estes parecem precisar de mais compreensão e avaliação.

Anexo: Abordagem *SMarty*

A abordagem *SMarty* (Bera et al., 2015b); (Geraldi et al., 2015); (Marcolino et al., 2014b); (Marcolino et al., 2014a); (Marcolino et al., 2013); (OliveiraJr et al., 2010b) possibilita o gerenciamento de variabilidades em LPSs modeladas em *Unified Modeling Language* (UML), a partir de um conjunto de estereótipos e diretrizes, que aplicadas aos elementos dos modelos, permitem a representação das variabilidades.

Os estereótipos fornecidos por *SMarty* estão organizados em um perfil UML denominado *SMartyProfile* (apresentado abaixo) e em um conjunto de diretrizes para auxiliar na identificação, delimitação e representação das variabilidades, com base em tais estereótipos, denominado *SMartyProcess*.

Figura 2.1: Versão 5.2 do *SMartyProfile*, com suporte a Componentes, Portas, Interfaces e Operações (Bera, 2015).



O *SMartyProfile* é uma extensão dos metamodelos da UML, versão 2.5. É possível perceber todos os estereótipos suportados pelo perfil na região inferior da Figura - 2.1. Os estereótipos representam conceitos característicos de LPS, como variabilidades, pontos de variação e variantes nos modelos UML. A seguir, são apresentados os estereótipos do *SMartyProfile*:

<< ***variability*** >> estereótipo de variabilidade. Esse estereótipo é uma extensão da metaclasse UML *Comment*, para representar explicitamente as variabilidades nos modelos UML. Tal estereótipo apresenta os seguintes meta-atributos;

- ***name*:** nome utilizado para referenciar uma variabilidade;
- ***minSelection*:** corresponde ao número mínimo de variantes selecionadas para resolver um ponto de variação e/ou uma variabilidade;
- ***maxSelection*:** corresponde ao número máximo de variantes selecionadas para resolver um ponto de variação e/ou uma variabilidade;
- ***bindingTime*:** corresponde ao momento de resolução da variabilidade. Os possíveis momentos de resolução são representados pela classe de enumeração Binding-Time;
- ***allowsAddingVar*:** indica se novas variantes podem ser incluídas após a resolução de uma variabilidade;
- ***variants*:** coleção de instâncias, associadas à variabilidade; e
- ***realizes*:** coleção de variabilidades de modelos de menor nível que realiza a variabilidade.

<< ***variationPoint*** >> estereótipo de ponto de variação. Esse estereótipo estende as metaclasses *Actor*, *Class*, *CombinedFragment*, *DecisionNode*, *InteractionUse*, *Interface*, *Port*, *UseCase* e possui os seguintes meta-atributos:

- ***numberofVariants:*** número de variantes associadas ao ponto de variação;
- ***bindingTime:*** estabelece o tempo de resolução do ponto de variação. Os possíveis tempos de resolução são determinados pela classe de enumeração *BindingTime* do Figura - 2.1;
- ***variants*:** coleção de instâncias de variantes associadas ao ponto de variação; e
- ***variabilities*:** coleção de variabilidades associadas com este ponto de variação.

<< ***variant*** >> estereótipo de variante. Tal estereótipo é especializado em outros quatros estereótipos (<< ***mandatory*** >>, << ***optional*** >>, << ***alternative_XOR*** >> e << ***alternative_OR*** >>), estende as metaclasses *Action*, *Actor*, *Class*, *CombinedFragment*, *Interface*, *Message*, *Operation*, *Port*, *UseCase* e possui os seguintes meta-atributos:

- ***rootVP*:** representa o ponto de variação ao qual a variante está associada; e
- ***variabilities*:** coleção de variabilidades ao qual a variante está associada.

<< ***mandatory*** >> estereótipo de variante obrigatória. Isso indica que essa variante sempre deve estar na resolução de um ponto de variação e/ou variabilidade associado(a). As seguintes metaclasses são estendidas por esse estereótipo: *Action*, *Actor*, *Class*, *CombinedFragment*, *InteractionUse*, *Interface*, *Lifeline*, *Message*, *Operation*, *Port*, *UseCase*;

<< ***optional*** >> estereótipo opcional, na resolução de um ponto de variação e/ou variabilidade associado(a). Tal estereótipo é uma extensão das seguintes metaclasses: *Action*, *Actor*, *Class*, *CombinedFragment*, *InteractionUse*, *Interface*, *Lifeline*, *Message*, *Operation*, *Port* e *UseCase*;

<< ***alternative_XOR*** >> estereótipo que indica a existência de um grupo de variantes exclusivas, do qual a variante marcada com tal estereótipo faz parte. Isso significa que apenas uma variante desse grupo pode ser selecionada para a resolução de um ponto de variação e/ou variabilidade associado(a). Esse estereótipo é uma extensão das metaclasses *Action*, *Actor*, *Class*, *CombinedFragment*, *InteractionUse*, *Interface*, *Lifeline*, *Message*, *Operation*, *Port* e *UseCase*;

<< ***alternative_OR*** >> estereótipo que indica que a variante pertence a um grupo de variantes inclusivas. Isso significa que diferentes combinações de variantes inclusivas podem ser selecionadas para a resolução de um ponto de variação e/ou variabilidade associado(a). Esse estereótipo é uma extensão das metaclasses *Action*, *Actor*, *Class*, *CombinedFragment*, *InteractionUse*, *Interface*, *Lifeline*, *Message*, *Operation*, *Port* e *UseCase*;

<< ***mutex*** >> estereótipo que representa o relacionamento mutuamente exclusivo entre variantes. Isso significa que a escolha de uma variante desse relacionamento exige a não-escolha da outra variante do relacionamento. Tal estereótipo é uma extensão das metaclasses *Dependency* e *Operation*;

<< ***requires*** >> estereótipo que representa um relacionamento de complemento entre duas variantes. Isso significa que a escolha de uma variante desse relacionamento requer

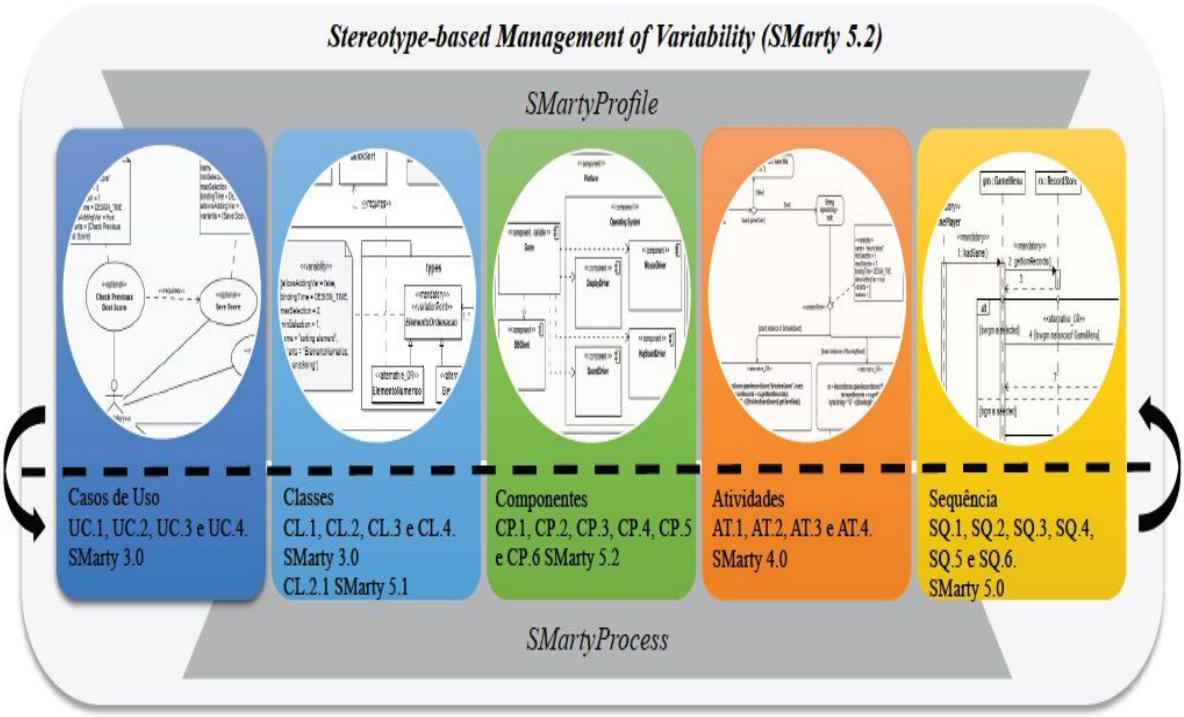
a seleção da outra variante relacionada. As metaclasses *Dependency* e *Operation* são estendidas por << *requires* >>;

<< ***variable*** >> estereótipo extensão das metaclasses *ActivityPartition* e *Component*, que indica a existência de classes com variabilidades explícitas em um componente. O atributo *classSet* representa a coleção de instâncias das classes variáveis existentes no componente.

Os estereótipos do *SMartyProfile* permitem a representação explícita dos conceitos de LPS e possibilitam que o processamento automatizado de modelos UML, realizado por ferramentas de modelagem UML, também considere as características de LPS.

A Figura - 2.2 exibe uma visão geral da abordagem SMarty.

Figura 2.2: Visão Geral SMarty 5.2 (Bera, 2015).

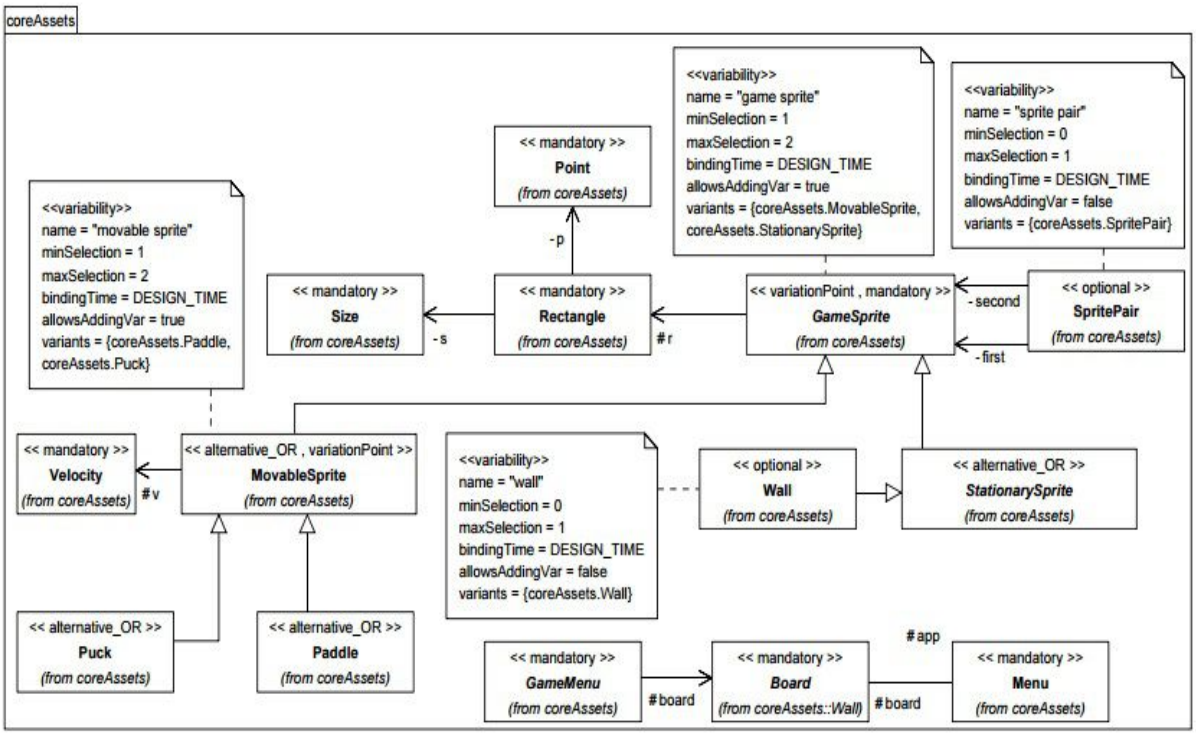


A Figura - 2.2 possibilita visualizar os modelos UML para o qual a abordagem *SMarty* oferece suporte. Para cada modelo suportado, tem-se os estereótipos, definidos pelo *SMartyProfile* e um conjunto de diretrizes específicas para cada diagrama. As diretrizes são apresentadas por diferentes siglas, representando seus respectivos diagramas. UC representa as diretrizes para o diagrama de casos de uso, CL representa as diretrizes para o diagrama de classes, CP representa as diretrizes para o diagrama de componentes, AT representa as diretrizes para o diagrama de atividades e SQ representa as diretrizes para

o diagrama de sequência. Cada conjunto de diretrizes foi especificado em uma versão do *SMarty*. Atualmente, o *SMarty* está na versão 5.2.

A Figura - 2.3 ilustra a aplicação dos estereótipos do *SMartyProfile* em um diagrama de classes da LPS *Arcade Game Maker* (AGM) .

Figura 2.3: Diagrama de Classes da LPS AGM (OliveiraJr et al., 2010b).



A relação entre variabilidades, pontos de variação e variantes pode ser percebida claramente na Figura - 2.3. Considerando a classe *GameSprite* presente na figura, percebe-se os estereótipos << *variationPoint* >> e << *mandatory* >>, indicando que a classe é um ponto de variação e simultaneamente uma variante obrigatória. Por ser um ponto de variação, tal classe está associada com uma variabilidade, no caso o comentário *game sprite*, estereotipado com << *variability* >>. Duas variantes estão associadas ao ponto de variação, *StationarySprite* e *MovableSprite*, ambas estereotipadas com << *alternative_OR* >>. Outros estereótipos também são observados, tais como o << *optional* >>, na classe *SpritePair* e o << *alternative_OR* >>, nas classes *Puck* e *Paddle*. Na geração de um produto por exemplo, o arquiteto de LPS pode escolher se o produto conterá elementos estáticos (*StationarySprite*), móveis (*MovableSprite*) ou mesmo ambos no jogo gerado.