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

ANA PAULA ALLIAN

***VMTools-RA*: uma arquitetura de referência para ferramentas de variabilidade de software**

Maringá
2016

ANA PAULA ALLIAN

VMTools-RA: uma arquitetura de referência para ferramentas de variabilidade 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
2016

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

Allian, Ana Paula
A436v VMTools-RA: uma arquitetura de referência para ferramentas de variabilidade de software / Ana Paula Allian. - - Maringá, 2016.
188 f. : il. (algumas color.), figs., tabs.

Orientador: Prof. Dr. Edson Alves Oliveira Junior.
Dissertação (mestrado) – Universidade Estadual de Maringá, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação, 2016.

1. Engenharia de software. 2. Arquitetura de referência. 3. Gerenciamento de variabilidade. 4. Linha de produto de software – Modelo de referência. I. Oliveira Junior, Edson Alves, orient. II. Universidade Estadual de Maringá. Centro de Tecnologia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 21.ed. 005.1

MGC-001722

FOLHA DE APROVAÇÃO

ANA PAULA ALLIAN

VMTools-RA: uma arquitetura de referência para ferramentas de variabilidade 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

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

Profa. Dra. Elisa Yumi Nakagawa
Universidade de São Paulo – ICMC/USP

Aprovada em: 29 de junho de 2016.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

DEDICATÓRIA

Aos meus pais, amigos e a toda a minha família,
pelo constante incentivo na busca pela realização
de meus sonhos.

AGRADECIMENTOS

Agradeço a Deus por ter me acompanhado e abençoado nesta caminhada e em todos os momentos da minha vida. Sem Ele, seria impossível chegar até aqui.

Quero agradecer a todas as pessoas que se fizeram presentes, que se preocuparam, que foram solidárias, que torceram por mim. Em especial a minha família por sempre acreditarem em mim. A minha mãe Odete pelos conselhos e dedicação, a minha irmã Juliana pelas orações e incentivo, ao meu irmão Gui por acreditar em mim.

Também quero agradecer aos meus amigos, pelo incentivo dado desde o começo dos meus estudos de mestrado, em especial aos meus queridos amigos Angela por me incentivar quando me encontrava desanimada e me ensinar a ter paciência e ser otimista para alcançar meus objetivos, ao Vlad por me guiar e incentivar desde o início nessa caminhada, ao Xandy por motivar e compartilhar suas experiências de vida acadêmica. Também agradeço ao queridos amigos Alex, Joca, Isaac, Renato, Gil, Ota, Klecius e muitos outros que estimo de coração pela amizade e carinho.

Agradeço, em especial, ao meu orientador professor Dr. Edson pela orientação de qualidade que recebi, por sua disponibilidade, amizade, apoio, paciência, perseverança, comentários e sugestões. Seus ensinamentos foram imprescindíveis para a conclusão desse projeto. Muito obrigada mesmo! À professora Elisa Yumi Nakagawa, pelos ensinamentos sobre arquiteturas de referência, pelo apoio, dedicação, amizade, paciência, respeito e carinho por mim, não possuo palavras para descrever minha gratidão. Também quero deixar registrado meu agradecimento ao professor Dr. Rafael Capilla (Rafa) pelos ensinamentos e contribuições sobre gerenciamento de variabilidade, pela paciência e amizade. ¡Muchas gracias!

Agradeço também aos alunos de doutorado e futuros doutores Marcolino, Lina, Lucas e Milena do ICMC-USP pelas sugestões, avaliações e contribuições para melhoria desse projeto. Também agradeço, em especial, ao Dr. Marcílio Mendonça da *Amazon Web Services (AWS)* e ao Dr. Ivan C. Machado da UFBA pelas contribuições, sugestões e críticas na avaliação desse projeto.

Aos professores do curso de Mestrado em Ciência da Computação do DIN da UEM, agradeço pelos ensinamentos e formação de qualidade que recebi. Agradeço a Inês pela amizade, paciência, otimismo, bom humor e disponibilidade em ajudar sempre. Também agradeço a todos os amigos que fiz durante meus estudos de mestrado, em especial para minha turma de 2014 e também para a turma de 2013.

Agradeço também a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concedido durante o desenvolvimento desta pesquisa. E a todos aqueles que me ajudaram direta ou indiretamente, para que eu chegasse até aqui, MUITO OBRIGADA!

VMTools-RA: uma arquitetura de referência para ferramentas de variabilidade de software

RESUMO

Gerenciamento de variabilidade (GV) possibilita adaptação de sistemas de software para contextos específicos de uma forma pré-planejada, lidando com semelhanças e variabilidade desses sistemas. GV tem sido considerada em vários tópicos de pesquisa e foi consolidada principalmente na área de Linha de Produto de Software (LPS). Várias ferramentas de variabilidade de software têm sido propostas com o objetivo de apoiar empresas na customização de novos produtos de software. Nota-se que a indústria tem adotado diferentes estratégias para gerenciar variabilidades, inclusive produzir suas próprias soluções resultando em sobreposição parcial de funcionalidades dessas ferramentas; além de subutilização das capacidades de tais ferramentas. Por exemplo, as atividades de modelar variabilidades são executadas em ferramentas de variabilidade de software e em editores de texto, tornando o GV inconsistente. Além disso, as soluções proprietárias mais conhecidas alavancam os custos de uso e adoção dessas ferramentas. A padronização de ferramentas de variabilidade de software é necessária, pois permite que empresas desenvolvam suas próprias ferramentas por meio de técnicas documentadas e testadas. Além disso, permite manter a consistência na terminologia usada com base no padrão proposto, aumentando a reusabilidade e convergindo para um conhecimento sólido sobre GV. Iniciativas para padronizar o GV em LPS foram estabelecidas como as normas ISO/IEC 26550 e ISO/IEC 26555, que fornecem modelos de referência e processos que envolvem LPS e GV. No entanto, modelos de referência são altamente abstratos e geralmente não estão diretamente ligado às práticas de implementação. Em outra perspectiva, arquitetura de referência é um tipo especial de arquitetura de software e reúne conhecimento de domínios específicos, facilitando o desenvolvimento, padronização e evolução de sistemas de software. Para mitigar a necessidade de um padrão para ferramentas de variabilidade de software pode-se adotar tal conceito. Portanto, o objetivo deste trabalho é especificar uma arquitetura de referência para ferramentas de variabilidade de software. Assim, o principal resultado é o estabelecimento da VMTools-RA, uma arquitetura de referência que reúne o conhecimento e a experiência de GV fornecendo mais confiança na tecnologia, evolução e reúso. Para tanto, foi utilizado o processo ProSA-RA, que sistematiza o projeto, representação e avaliação de arquiteturas de referência. Tal proposta foi avaliada por meio de um estudo qualitativo utilizando um *checklist*, e por um exemplo de aplicação com a instanciação arquitetural de uma ferramenta de variabilidade de software baseada na VMTools-RA. Os resultados empíricos preliminares fornecem indícios de que a VMTools-RA é uma arquitetura de referência viável para o desenvolvimento de novas ferramentas de variabilidade de software.

Palavras-chave: Gerenciamento de Variabilidades. Arquitetura de Referência. Ferramentas de Variabilidade de Software. Avaliação Empírica. VMTools-RA.

VMTools-RA: A Reference Architecture for Software Variability Tools

ABSTRACT

Variability Management (VM) makes it possible to easily adapt software systems for specific contexts in a preplanned manner, dealing with commonalities and variabilities of these systems. VM has been considered in several research topics and was mainly consolidated by the Software Product Line (SPL) area. A diversity of software variability tools has been proposed to support companies in the customization of new software products. It is observed industry has adopted different strategies to manage variabilities, including produce its own solutions resulting in partially overlapped of functionality of such tools; as well as underutilization of such solutions without using all of the tools capacities. For example, the variability model activities are performed in software variability tools and text editors, resulting in inconsistencies in the VM activity. Furthermore, the most known proprietary solutions leverage the costs to use and adopt such VM tools. The standardization of software variability tools is necessary, because it allows companies to develop their own tools through documented and tested techniques. Besides, it allows to maintain consistency in the terminology used based on the proposed standard, increasing the reuse, and converging to a solid knowledge about VM. Standardization efforts for VM in the SPL context have been established, such as ISO / IEC 26550 and ISO / IEC 26555 standards, which provide reference models and processes involving SPL and VM. However, reference models are highly abstract and they are not usually directly connected to implementation practices. In another perspective, reference architecture is a special type of software architecture and gathers knowledge from specific domains, facilitating the development, standardization, and evolution of software systems. To mitigate the standard necessity for software variability tools the concept of reference architecture can be adopted. Therefore, the purpose of this work is to specify a reference architecture to software variability tools. Hence, the main result is the establishment of VMTools-RA, a reference architecture that combines knowledge and experience in VM providing more confidence in the technology, evolution, and reuse. Therefore, we used the ProSA-RA process, which systematizes the project, representation, and evaluation of reference architectures. Such proposal was evaluated by a qualitative study using a checklist, and with a sample application by instantiating a software variability tool based on the VMTools-RA. The preliminary empirical results provide evidences that VMTools-RA is a viable reference architecture for developing new software variability tools.

Keywords: Variability Management. Reference Architecture. Software Variability Tools. Empirical Evaluation. VMTools-RA.

LISTA DE FIGURAS

Figura 2.1 Modelo de referência para LPS. Traduzido de (ISO/IEC, 2013a) . 20
Figura 2.2 Modelo de referência para o gerenciamento técnico de LPS. Traduzido de (ISO/IEC, 2013b) 21
Figura 2.3 Fases do processo do MS. Adaptação de (Kitchenham e Charters, 2007; Petersen et al., 2015) 22
Figura 2.4 Processo e resultados do MS 23
Figura 2.5 Modelo de arquitetura de referência. Traduzido de (Cloutier et al., 2010) 27
Figura 2.6 Diferenças entre arquitetura de referência e arquitetura de LPS. Traduzido de (Martínez-Fernández et al., 2013b) 29
Figura 2.7 Etapas do processo ProSA-RA. Traduzido de (Nakagawa et al., 2014) 32

Figura 3.1 Relação entre ferramentas de variabilidade de software e arquitetura de referência. Traduzido de (OliveiraJr e Allian, 2015) 36
Figura 3.2 Visão geral da VMTools-RA 49
Figura 3.3 Visão conceitual da VMTools-RA 55
Figura 3.4 Visão de variabilidade da VMTools-RA utilizando modelo de *feature* 61
Figura 3.5 Visão de decisão arquitetural da VMTools-RA 63
Figura 3.6 Visão de processos da VMTools-RA 65
Figura 3.7 Visão em módulo da VMTools-RA 72
Figura 3.8 Visão de implantação da VMTools-RA 74

Figura 5.1 Instância da arquitetura de referência 90
Figura 5.2 Relação entre conceitos da ferramenta de variabilidade de software 94
Figura 5.3 Visão de variabilidades da VMTools-RA destacando a seleção das *features* correspondentes à instância arquitetural 95
Figura 5.4 Visão de implantação da ferramenta de variabilidade de software . 96
Figura 5.5 Visão em módulo da ferramenta de variabilidade de software . . . 97
Figura 5.6 Visão de processos da ferramenta de variabilidade de software . . 98

Figura 1.1 Query geral 122
Figura 1.2 Estudos obtidos por motores de busca de dados eletrônicos 124
Figura 1.3 Distribuição dos estudos em diferentes locais de publicação 124
Figura 1.4 Estágios e resultados do processo de seleção de estudos. 125
Figura 1.5 Distribuição das ferramentas por ano relacionado aos tipos 131
Figura 1.6 Cronologia da notação FODA e suas extensões 135
Figura 1.7 Ferramentas de variabilidades de software open source e comerciais 137

Figura 2.1 Verificação de consistência da VMTools-RA. 149

LISTA DE TABELAS

Tabela 2.1 Ferramentas classificadas por ano e tipo 23

Tabela 3.1 Características das ferramentas identificadas no MS 37
Tabela 3.2 Funcionalidades das ferramentas de variabilidade de software . . . 40
Tabela 3.3 Matriz de rastreamento entre requisitos e fontes de informação . . 45
Tabela 3.4 Matriz de rastreabilidade entre requisitos arquiteturais 46
Tabela 3.5 Mapeamento entre requisitos e elementos da VMTools-RA 48
Tabela 3.6 Relação entre *stakeholders*, requisitos, riscos e interesses 54
Tabela 3.7 Instâncias arquiteturais de ferramentas de variabilidade de software 59
Tabela 3.8 Relação entre ferramentas e elementos da VMTools-RA 60
Tabela 3.9 Documentação de decisão arquitetural - Integração (Q1) 62
Tabela 3.10 Documentação de decisão arquitetural - Integração (Q2) 63
Tabela 3.11 Histórico de mudança para o ponto de vista transversal 64
Tabela 3.12 Histórico de mudança para o ponto de vista de tempo de execução 66
Tabela 3.13 Guia para criação da base de ativos 67
Tabela 3.14 Guia para integração 68
Tabela 3.15 Guia para modelo de variabilidade e verificação de consistência . . 69
Tabela 3.16 Guia para realização das variabilidades 70
Tabela 3.17 Guia para evolução das variabilidades 71
Tabela 3.18 Histórico de mudança para o ponto de vista código fonte 72
Tabela 3.19 Histórico de mudança para o ponto de vista implantação 74

Tabela 4.1 Especialistas em arquitetura de referência 77
Tabela 4.2 Especialistas em ferramentas de variabilidade de software 78
Tabela 4.3 Resultados do *checklist* obtidos por especialistas 79
Tabela 4.4 Relação de códigos, categorias e especialistas 83
Tabela 4.5 Solicitações de melhorias e refatoração da VMTools-RA 84

Tabela 1.1 Fontes de literatura e respectivas queries 122
Tabela 1.2 Locais de publicação 123
Tabela 1.3 Possíveis ferramentas de variabilidades de software 127
Tabela 1.4 Ferramentas de variabilidades de software selecionadas 129
Tabela 1.5 Tipo da ferramentas de variabilidades de software 130
Tabela 1.6 Ferramentas disponíveis para download 132
Tabela 1.7 Ferramentas avaliadas pela indústria 133
Tabela 1.8 Ferramentas classificadas por notações do modelo de *feature* . . . 136
Tabela 1.9 Linguagem de programação 138
Tabela 1.10 Visualização das variabilidades 139
Tabela 1.11 Serviços de suporte 140

LISTA DE SIGLAS E ABREVIATURAS

AD: *Architecture Description*
ADL: *Architecture Description Language*
API: *Application Programming Interface*
AR: Arquitetura de Referência
CBFM: *Cardinality Based Feature Model*
FODA: *Feature Orientaded Domain Analysis*
GNU: *General Public Licence*
GV: Gerenciamento de Variabilidade
HTML: *Hyper Text Markup Language*
LPS: Linha de Produto de Software
MS: Mapeamento Sistemático
OMG: *Object Management Group*
PLA: *Product Line Architecture*
REST: *REpresentational State Transfer*
RMI: *Remote Method Invocation*
ROI: *Return On Investment*
RSL: Revisão Sistemática da Literatura
SOAP: *Simple Object Access Protocol*
TCP: *Transmission Control Protocol*
UML: *Unified Modeling Language*
VM: *Variability Management*
VMTools-RA: *Variability Management Tools - Reference Architecture*
XML: *Extensible Markup Language*
XSL: *Extensible Stylesheet Language*

SUMÁRIO

1 Introdução **14**

1.1 Contextualização 14
1.2 Motivação 15
1.3 Objetivos 16
1.4 Metodologia de Desenvolvimento 17
1.5 Organização 17

2 Fundamentação Teórica **18**

2.1 Considerações Iniciais 18
2.2 Variabilidade de Software 18
2.3 Arquitetura de Software, Terminologias e Conceitos 25
2.4 Arquitetura de Referência 26
2.4.1 Arquitetura de Referência x Arquitetura de LPS 27
2.4.2 Benefícios e Limitações das Arquiteturas de Referência 28
2.4.3 Representação Arquitetural 30
2.4.4 Processo para a Construção de Arquitetura de Referência 31
2.5 Considerações Finais 33

3 VMTools-RA **34**

3.1 Considerações Iniciais 34
3.2 Etapa 1 - Investigação das Fontes de Informação 34
3.2.1 Grupo 1: Padronizações Aplicáveis à LPS e ao GV 35
3.2.2 Grupo 2: Ferramentas de Variabilidade de Software 36
3.2.3 Grupo 3: Estudos Secundários sobre Ferramentas de Variabilidade de Software 37
3.3 Etapa 2 - Estabelecimento dos Requisitos Arquiteturais 41
3.3.1 Requisitos Arquiteturais para GV 41
3.3.2 Requisitos Arquiteturais para Evolução do GV 42
3.3.3 Rastreamento dos Requisitos 44
3.4 Etapa 3 - Projeto da Arquitetura de Referência 47
3.4.1 Visão Geral da VMTools-RA 47
3.4.2 Escopo 50
3.4.3 Objetivos, *Stakeholders*, Interesses e Riscos 50
3.4.4 V.1 - Ponto de Vista Transversal 54
3.4.5 V.2 - Ponto de Vista de Tempo de Execução 64
3.4.6 V.3 - Ponto de Vista de Código Fonte 66
3.4.7 V.4 - Ponto de Vista de Implantação 73
3.5 Etapa 4 - Avaliação da Arquitetura de Referência 74

3.6 Considerações Finais 75

4 Estudo Empírico Qualitativo da VMTools-RA 76
4.1 Considerações Iniciais 76
4.2 Planejamento da Avaliação 76
4.3 Execução da Avaliação 77
4.4 Definição do Estudo Qualitativo 80
4.5 Análise e Interpretação dos Resultados 81
4.6 Refatoração e Melhorias para VMTools-RA 83
4.7 Ameaças à Validade do Estudo 86
4.7.1 Ameaças à Validade de Conclusão 86
4.7.2 Ameaças à Validade de ***Constructo*** 86
4.7.3 Ameaças à Validade Interna 87
4.7.4 Ameaças à Validade Externa 88
4.8 Considerações Finais 88

5 Exemplo de Aplicação: Instanciação da VMTools-RA 89
5.1 Considerações Iniciais 89
5.2 Descrição Geral da Ferramenta 89
5.3 Instanciação da VMTools-RA 91
5.3.1 Requisitos Arquiteturais, *Stakeholders* e Interesses 91
5.3.2 Ponto de Vista Transversal 93
5.3.3 Ponto de Vista de Implantação 95
5.3.4 Ponto de Vista de Código Fonte 96
5.3.5 Ponto de Vista de Tempo de Execução 97
5.4 Discussão sobre a Instanciação 99
5.5 Considerações Finais 99

6 Conclusão 100
6.1 Contribuições 100
6.2 Dificuldades e Limitações 102
6.3 Trabalhos Futuros 103

REFERÊNCIAS 104

A Apêndice A - Mapeamento Sistemático da Literatura de Ferramentas de Variabilidade de Software 120
A.1 Método de Pesquisa 120
A.1.1 Necessidade do Estudo 120
A.1.2 Questões de Pesquisa 121
A.1.3 Estratégia de Pesquisa 121

A.1.4 Seleção Inicial . . . 123
A.1.5 Critérios de Inclusão e Exclusão . . . 125
A.1.6 Estudos Selecionados . . . 128
A.2 Análise dos Resultados . . . 128
A.2.1 Classificação dos Tipos de Ferramentas de Variabilidades de Software 128
A.2.2 Notações Utilizadas nas Ferramentas de Variabilidades de Software 134
A.2.3 Tecnologias Utilizadas nas Ferramentas de Variabilidades de Software 137
A.3 Estudos Relacionados . . . 143
A.4 Ameaças à Validade . . . 144
A.5 Tendências Futuras . . . 146
A.6 Conclusão . . . 147

B Apêndice B - Verificação de Consistência da VMTools-RA 148
B.1 Verificação de Consistência da VMTools-RA . . . 149
B.2 Instância Arquitetural de Ferramenta de Modelagem de Variabilidades . . . 150
B.3 Instância Arquitetural de Ferramenta de Modelagem e Configuração das Variabilidades com Verificação de Consistência . . . 151
B.4 Instância Arquitetural de Ferramenta de Gerenciamento de Variabilidade . . . 152
B.5 Verificação da Consistência da Instância Arquitetural de Ferramenta de Variabilidade . . . 153

C Apêndice C - Questionário de Caracterização dos Especialistas 154
C.1 Questionário de Caracterização dos Especialistas em Arquitetura de Referência . . . 155
C.2 Questionário de Caracterização dos Especialistas em Gerenciamento de Variabilidade . . . 156

D Apêndice D - Respostas dos Especialistas no Estudo Empírico Qualitativo da VMTools-RA 157

E Apêndice E - Documentação da Primeira Versão da *VMTools-RA* 169

F Anexos 182
F.1 *Checklist* de Avaliação do FERA . . . 182

1

Introdução

1.1 Contextualização

Variabilidade de software possibilita antecipar decisões de projetos com relação a personalização, extensão e adaptação de sistemas de software para contextos distintos (Bachmann e Clements, 2005). Nas últimas décadas, LPS consolidou o gerenciamento de variabilidade (GV) como uma de suas atividades essenciais para atingir a reutilização em larga escala (Bosch et al., 2015; Capilla et al., 2013; Galster et al., 2014; Linden et al., 2007). O GV tem início na análise de domínio e especificação de requisitos, que tratam da identificação dos artefatos comuns e variáveis resultando em um modelo de variabilidade, e prossegue com a especificação arquitetural, implementação, análise e refatoração do código para a configuração e criação do produto final (Gurp et al., 2001; Pohl et al., 2005). Para realizar o ciclo completo de GV, uma infraestrutura computacional é necessária. Por isso, várias abordagens e ferramentas de variabilidade de software têm sido propostas e utilizadas pelas indústrias para possibilitar a geração de produtos de software.

Iniciativas para padronizar o GV no desenvolvimento de LPS foram estabelecidas, tais como, normas internacionais ISO/IEC 25550 (2013a) e ISO/IEC 25555 (2013b). Essas normas contribuem com modelos de referência e detalhes dos processos para a construção de LPS. Além disso, fornecem modelos de referência e informações do grupo de processos que envolvem o GV. Tais informações são necessárias para estabelecer uma padronização de ferramentas de variabilidade de software.

Ainda no contexto de padronizações, pode-se citar arquitetura de referência, que é um tipo especial de arquitetura de software (Nakagawa et al., 2013) com artefatos reutilizáveis para um domínio particular (Buchgeher e Weinreich, 2013). Uma arquitetura de referência abrange o conhecimento e a experiência sobre como estruturar arquiteturas de software de um domínio de sistemas, além de ser um guia para o desenvolvimento, padronização

e evolução dos sistemas desses domínios (Angelov et al., 2009; Cloutier et al., 2010; Martínez-Fernández, 2013; Nakagawa et al., 2013). Bons exemplos de arquitetura de referência são *AUTomotive Open System ARchitecture* (AUTOSAR) (AUTOSAR, 2016) para o domínio automotivo, *Continua* (Continua, 2016) e UniversAAL (UniversAAL, 2016) para *Ambient Assisted Living* (AAL) (AALIANCE, 2010).

Para apoiar o projeto de arquitetura de referência, pode-se utilizar um processo que organiza a sua construção como, por exemplo, o *Process based on Software Architecture - Reference Architecture* (ProSA-RA) (Nakagawa et al., 2014). ProSA-RA é um processo iterativo para a especificação, projeto e avaliação de arquitetura de referência. Esse processo apoia a criação de arquiteturas de referência para um determinado domínio e tem sido utilizado na construção de arquiteturas de referência de diversos domínios, tais como sistemas marítimos (Borg, 2011), sistemas robóticos (Feitosa, 2013), aplicação para TV digital (Duarte, 2012), ambientes educacionais (Fioravanti et al., 2010), mineração visual (Nakagawa et al., 2009) e teste de software (Nakagawa et al., 2007; Oliveira e Nakagawa, 2011).

1.2 Motivação

Devido à complexidade para gerenciar variabilidades em softwares, diversos tipos de notações e ferramentas de apoio têm sido construídas para possibilitar a implementação do GV (Berger et al., 2013). A heterogeneidade de notações e ferramentas mostra que a indústria ainda não resolveu o problema do GV e, por isso, continua a experimentar diferentes soluções para atender necessidades de domínios específicos (Bosch et al., 2015). Dentre os problemas enfrentados por empresas que adotam o GV pode-se destacar a falta de ferramentas robustas para gerenciar variabilidade dinamicamente, além de soluções capazes de gerenciar grandes modelos de variabilidade (Bosch et al., 2015). Também, destaca-se o custo elevado para a contratação de ferramentas comerciais consolidadas, tais como a Gears da empresa Biglever[1] e pure::variants da empresa pure::systems[2]. Dessa forma, a indústria tem criado suas próprias soluções (Berger et al., 2013) e utilizado uma variedade de ferramentas com diferentes finalidades (por exemplo, gerenciar variabilidade, configurar produtos, especificar requisitos, derivar produtos, modelar variabilidade e documentação da variabilidade)(Berger et al., 2013). Como consequência, é comum a mistura de técnicas para gerenciar variabilidade, tais como o uso de soluções baseadas em planilhas eletrônicas para gerenciar os ativos de domínio (por exemplo, *features*, requisitos, componentes, casos de testes e arquitetura), notações baseadas em UML para representar os modelos de variabilidade e a subutilização de ferramentas de variabilidade de software que pode envolver um investimento financeiro sem a utilização de todas as capacidades

[1] www.biglever.com
[2] www.pure-systems.com

da ferramenta (Berger et al., 2014, 2013). Em empresas que desejam apenas modelar a variabilidade, as notações UML e as planilhas eletrônicas parecem atender às necessidades iniciais, mas à medida que a atividade de GV evolui e outras variantes e pontos de variação são identificados, problemas de consistência entre os modelos e os ativos de domínio se tornam mais expressivos e, muitas vezes, empresas precisam refazer os modelos de variabilidade. A falta de uma completa integração entre a modelagem de variabilidade e o gerenciamento de ativos de domínio dificulta o gerenciamento eficiente da variabilidade nos produtos de software (Bosch et al., 2015). Portanto, uma boa prática é a definição de uma estrutura de apoio que possa ser utilizada na atividade de GV e assim servir de guia e padronização para as empresas desenvolverem suas próprias ferramentas.

Na engenharia de software, iniciativas de padronizações são importantes, pois possibilitam a adoção de métodos, técnicas e ferramentas estabelecidas para o desenvolvimento de produtos de software (Käkölä, 2010). A falta de padronização possibilita divergências e inconsistências de métodos, técnicas e ferramentas, aumentando o risco de comprometer os atributos de qualidade do sistema e resultar em retrabalho (Käkölä, 2010) . ISO/IEC 26550 e ISO/IEC 26555 são bons exemplos de normas internacionais estabelecidas para LPS e GV. Elas compreendem modelos de referência e processos guiando especialistas nos principais conceitos para adoção de LPS e GV. No entanto, modelos de referência apresentam representações mais abstratas do domínio, não dando suporte adequado ao desenvolvimento sistemático de sistemas. Por isso, o conceito de arquitetura de referência pode colaborar na construção de sistemas de software, pois abrange elementos técnicos, padrões arquiteturais, modelos de negócio e necessidades dos clientes, sendo assim um ponto de partida para especificação de um sistema.

Portanto, acredita-se que uma arquitetura de referência possa facilitar o desenvolvimento de ferramentas de variabilidade de software, permitindo guiar-se por soluções conhecidas e experimentadas pela academia e indústria. Além disso, há possíveis contribuição com reúso e simplificação na tomada de decisão no desenvolvimento de novas ferramentas, o que se torna possível frente ao conhecimento prático e teórico estabelecido e documentado sobre GV.

1.3 Objetivos

Conforme a motivação apresentada, esta dissertação tem como objetivo especificar uma arquitetura de referência para ferramentas de variabilidade de software, que facilite o desenvolvimento de soluções desse domínio e permita a evolução das ferramentas de variabilidade de software existentes. Consequentemente, o principal resultado é o estabelecimento da *Variability Management Tools - Reference Architecture* (VMTools-RA), uma arquitetura de referência que reúne o conhecimento e a experiência de GV. Para tanto, os seguintes objetivos específicos são propostos:

- identificar padrões/normas internacionais e modelos de referência no contexto de GV;
- identificar as principais ferramentas de variabilidade de software existentes na academia e na indústria; e
- identificar os elementos arquiteturais em diferentes níveis de abstração das ferramentas existentes;
- projetar e avaliar a VMTools-RA.

1.4 Metodologia de Desenvolvimento

Para o desenvolvimento deste trabalho, foram utilizadas três abordagens: teórica, empírica e exemplo de aplicação. A abordagem teórica envolveu a revisão bibliográfica apoiada por um estudo secundário de literatura sobre ferramentas de variabilidade de software na forma de um mapeamento sistemático. A especificação da VMTools-RA foi realizada com base nas quatro etapas do processo ProSA-RA utilizando os principais elementos arquiteturais, incluindo tecnologias de implementação, intercâmbio de dados e as etapas do ciclo de vida de suporte dessas ferramentas. A avaliação levou em consideração uma análise empírica qualitativa por meio de inspeção por *checklist* e procedimentos de *Grounded Theory* com base no conhecimento de especialistas em GV e em arquitetura de referência. O exemplo de aplicação envolveu a instanciação, por diagramas UML, de uma ferramenta de variabilidade de software baseada na VMTools-RA.

1.5 Organização

Este capítulo apresentou a contextualização desta dissertação, a motivação, objetivos e metodologia. O restante desse documento está estruturado da seguinte forma: o Capítulo 2 apresenta conceitos sobre variabilidade de software e fundamentação teórica sobre GV; além disso, apresenta definições e conceitos sobre arquiteturas de software e arquiteturas de referência, bem como o processo ProSA-RA. No Capítulo 3 é apresentada a especificação da arquitetura de referência denominada VMTools-RA. A avaliação empírica da VMTools-RA por especialistas é apresentada no Capítulo 4. O exemplo de aplicação com a instanciação de uma ferramenta de variabilidade de software baseada na VMTools-RA é apresentado no Capítulo 5. No Capítulo 6 é apresentada a conclusão acerca desta dissertação, bem como as suas contribuições, limitações e trabalhos futuros.

2

Fundamentação Teórica

2.1 Considerações Iniciais

A demanda por sistemas de software de qualidade e com baixo custo tem levado especialistas a explorarem novas abordagens que promovam reúso e conhecimento do domínio. Dentre essas soluções, pode-se citar arquiteturas de referência e GV que possibilitam às empresas construírem uma variedade de sistemas com diversidade técnica reduzida.

Dessa forma, este capítulo revisita importantes conceitos sobre o referencial teórico necessário para a compreensão desta pesquisa. Assim, os principais conceitos relacionados à variabilidade de software, GV, modelos de referência e ferramentas de apoio são apresentados na Seção 2.2; conceitos e terminologias sobre arquitetura de software são apresentados na Seção 2.3; e os principais conceitos, benefícios e limitações sobre arquitetura de referência são apresentados na Seção 2.4; e, por fim, na Seção 2.5 são discutidas as considerações finais deste capítulo.

2.2 Variabilidade de Software

Variabilidade de software é a capacidade de um sistema, ativo de software, ou ambiente de desenvolvimento ser configurado, customizado, ou alterado para uso em um domínio específico de uma forma pré-planejada (Bachmann e Clements, 2005; Capilla et al., 2013; Chen et al., 2009). As variabilidades surgem do adiamento de certas decisões do projeto de produtos de software. Quanto maior o número de decisões adiadas, maior será o número de variabilidades de um produto de software (Halmans e Pohl, 2003). As variabilidades são descritas por: `Ponto de variação` que permite a resolução de suas variabilidades em um ou diversos locais por meio das suas variantes associadas; `Variantes` representam artefatos de software ou possíveis elementos, os quais podem ser escolhidos para resolver

um ponto de variação; `Restrições entre Variantes` estabelecem os relacionamentos entre uma ou mais variantes com o objetivo de resolver seus respectivos pontos de variação (Halmans e Pohl, 2003; Linden et al., 2007; Pohl et al., 2005).

Variabilidade de software pode estar associada a diferentes níveis de abstração, sendo eles a descrição da arquitetura, a documentação de projeto, o código fonte, o código compilado, o código ligado e o código executável. Esses diferentes níveis de abstração podem estar associados a diferentes estágios do desenvolvimento de software (Linden et al., 2007; OliveiraJr et al., 2013). As variabilidades podem ser inicialmente identificadas por meio de *feature* que pode ser definida como uma característica de um sistema que é relevante e visível para o usuário final (Kang et al., 1990).

As *features* podem ser classificadas em: Obrigatórias *(Mandatory)*, estão sempre presentes e identificam um produto; Opcionais *(Optional)*, podem ou não estar presentes em um produto; Inclusivas *(OR)*, possuem um conjunto de *features* relacionadas em que zero ou mais dessas *features* podem ser selecionadas para estarem presentes em um produto; Exclusivas *(XOR)*, possuem um conjunto de *features* relacionadas em que apenas uma dessas *features* pode ser selecionada para estar presente em um produto (Czarnecki e Eisenecker, 2000);

Gerenciar variabilidade é uma das atividades essenciais para produzir uma família de produtos de software; por isso, o GV é considerado uma das atividades mais importantes na construção de LPS, e requer mudanças abrangentes para o processo de desenvolvimento de software. Van Gurp et al. (2001) sugerem um conjunto de atividades para gerenciar variabilidade em LPS:

1. **Identificação das variabilidades:** identifica onde as variabilidades são necessárias utilizando modelos de variabilidade, tais como: modelos de *features*, modelo de decisão, modelos ortogonais, etc.; ou especificação de requisitos;

2. **Restrições das Variabilidades:** delimita o ponto de variação com diferentes atividades, tais como: escolher o tempo de resolução *(binding time)* para cada ponto de variação, decidir quando e como as variantes serão adicionadas ao sistema, escolher padrões de variabilidades para cada ponto de variação e escolher a representação da realização das variabilidades para os pontos de variação;

3. **Implementação das Variabilidades:** uma vez que as variabilidades foram identificadas, o sistema deve ser projetado e implementado de uma maneira que mecanismos ou técnicas para resolver as variabilidades sejam selecionados; e

4. **Gerenciamento de Variantes:** inclui o controle para adicionar ou remover variantes de acordo com as mudanças dos requisitos.

Atualmente mais organizações estão adotando o GV e LPS como solução para aumento do retorno de investimento (ROI), produtividade, flexibilidade e necessidades de customização em massa (SEI, 2016). O Instituto de Engenharia de Software (SEI)

mantém uma página *online* conhecida por *Hall-of-Fame*[1] das organizações que tem adotado LPS, tais como, Hewlett-Packard, Cummins, Inc., General Motors, CelsiusTech, Rockwell-Collins, Motorola, Philips, Nokia, Boeing, Raytheon, and Salion, Inc. Essas organizações tipicamente precisam superar diversos desafios para se beneficiar da LPS, tais como, implementar mudanças coordenadas em metodologias de desenvolvimento, ferramentas, arquiteturas, projetos organizacionais e modelos de negócio (Käkölä, 2010). Por isso, modelos de referência e normas internacionais no contexto de LPS e GV foram estabelecidos para apoiar empresas a compreender os processos que envolvem LPS e GV. Dentre essas normas pode-se citar ISO/IEC 26550 (2013a) e ISO/IEC 26555 (2013b). Mais detalhes sobre tais normas e modelos de referência são apresentados a seguir.

Modelos de Referência para LPS

O desenvolvimento de LPS não leva em conta apenas a codificação da lógica da aplicação, mas também fatores externos, a saber, colaboração de *stakeholders*, *feedbacks* para evolução da LPS, análise de mercado, entre outros. Dessa forma, modelos de referência facilitam a obtenção de uma visão geral de todo o processo de desenvolvimento de LPS e, consequentemente, de ferramentas de variabilidade de software.

O modelo de referência para LPS estabelecido na norma ISO/IEC 26550 (2013a) e representado na Figura 2.1, tem o objetivo de incorporar os conceitos centrais da engenharia de LPS tradicional, apoiar o desenvolvimento de LPS e facilitar a reutilização de artefatos, além de permitir a customização em massa por meio de variabilidades.

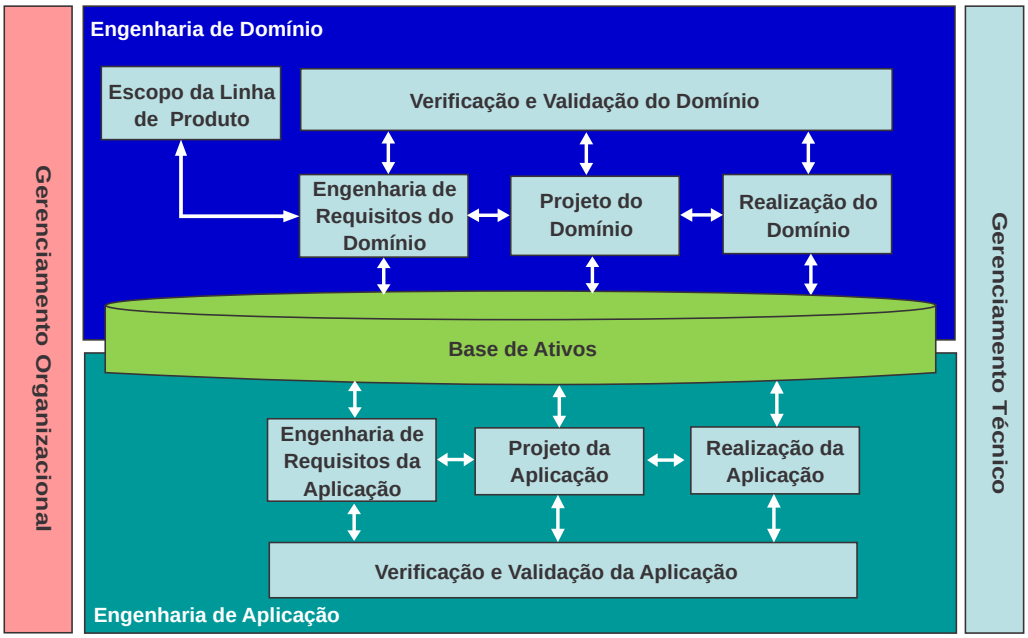


Figura 2.1: Modelo de referência para LPS. Traduzido de (ISO/IEC, 2013a)

[1] http://splc.net/fame.html

Tal modelo é constituído por dois ciclos de vida de desenvolvimento denominados `Engenharia de Domínio` e `Engenharia de Aplicação`, e por dois grupos de processos denominados `Gerenciamento Organizacional` e `Gerenciamento Técnico` (ISO/IEC, 2013a; Linden et al., 2007; Pohl et al., 2005). `Engenharia de Domínio` é responsável por modelar as semelhanças e variabilidades dos principais ativos da LPS. `Engenharia de Aplicação` é responsável por derivar os produtos de uma LPS selecionando, configurando e integrando os principais ativos. A `Base de Ativos` armazena ativos tanto do domínio como da aplicação, além de separar, sincronizar e coordenar os dois ciclos de vida de desenvolvimento. O grupo de processo do `Gerenciamento Organizacional` é responsável por auxiliar organizações a estabelecer e melhorar o desenvolvimento de LPS, além de gerenciar os relacionamentos entre clientes, fornecedores e *stakeholders* (ISO/IEC, 2013a). O grupo de processo do `Gerenciamento Técnico` fornece processos para abordar as questões técnicas que surgem durante a implementação de uma LPS e é também representado por um modelo de referência estabelecido na norma ISO/IEC 26555 (2013b), conforme apresentado na Figura 2.2.

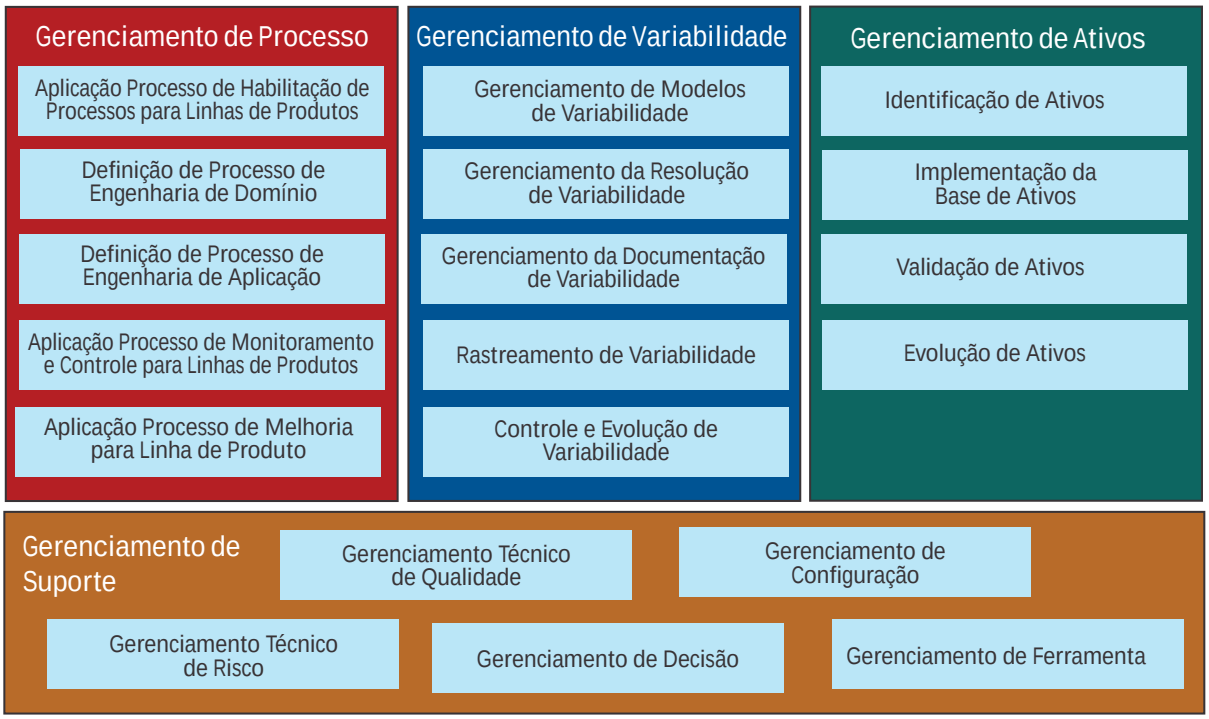


Figura 2.2: Modelo de referência para o gerenciamento técnico de LPS. Traduzido de (ISO/IEC, 2013b)

O modelo de referência para o Gerenciamento Técnico é composto por quatro grupo de processos: (i) `Gerenciamento de Processo`, responsável pelo suporte organizacional e cooperação entre *stakeholders*; (ii) `Gerenciamento de Variabilidade`, define como os membros de uma LPS serão diferenciados; (iii) `Gerenciamento de Ativos`,

responsável pelo armazenamento dos ativos; e (iv) `Gerenciamento de Suporte`, apoia implementação, capacitação e automação dos outros processos (ISO/IEC, 2013b). O grupo de processos do `Gerenciamento de Variabilidade`, que é o foco desta dissertação, é considerado um dos mais desafiadores e importantes para o sucesso na adoção de LPS (Bosch et al., 2015; Capilla et al., 2013; ISO/IEC, 2013a; Linden et al., 2007).

Além dos modelos de referência que fornecem importantes informações sobre o GV, diversas ferramentas foram desenvolvidas com o objetivo de apoiar os processos que envolvem o GV. Para a aquisição desse conhecimento, um mapeamento sistemático da literatura foi realizado (Apêndice A).

Ferramentas de Apoio ao GV

Um mapeamento sistemático (MS)[2] (Apêndice A) foi conduzido seguindo as diretrizes propostas em Petersen (2015) e Kitchenham et al., (2009) com objetivo de identificar as ferramentas de variabilidade de software disponíveis na academia e indústria. As fases do processo deste MS são apresentados na Figura 2.3.

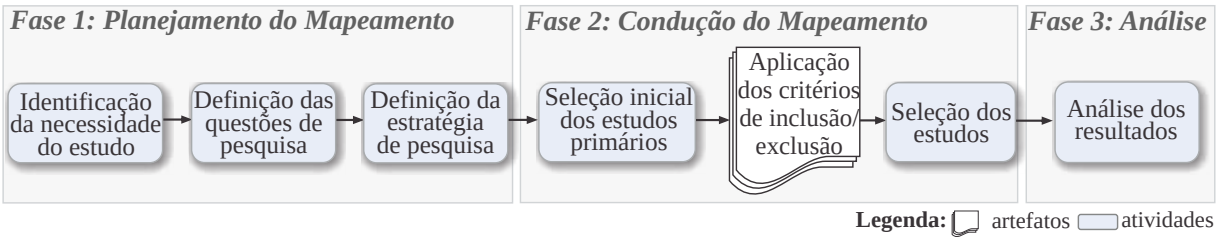


Figura 2.3: Fases do processo do MS. Adaptação de (Kitchenham e Charters, 2007; Petersen et al., 2015)

Em síntese, o objetivo do MS inclui a identificação dos tipos de ferramentas de variabilidade de software disponíveis, se foram utilizadas pela indústria e evidências de características técnicas utilizadas para implementá-las. Portanto, as seguintes questões de pesquisa foram estabelecidas: (RQ1) Quais tipos de ferramentas de variabilidade de software têm sido desenvolvidas e utilizadas?; e (RQ2) Quais são as tecnologias fundamentais que apoiam as ferramentas de variabilidade de software?.

Com a execução do MS, 879 estudos primários foram recuperados. Dentre esse total, 302 estudos foram imediatamente excluídos por se tratarem de estudos duplicados e informações de capas de apresentação de eventos. Finalmente, 577 estudos foram selecionados para a aplicação dos critérios de inclusão e exclusão. Esses critérios correspondem a leitura do título e resumo de cada documento resultando em 512 estudos removidos. Após essa etapa, 65 trabalhos foram selecionados para leitura na integra, possibilitando a identificação de 30 novos estudos para leitura por meio da aplicação da

[2]Um artigo sobre o MS foi submetido ao periódico *ACM Computing Surveys*.

técnica de *snowballing*, que consistiu da utilização dos métodos *backward* (busca novos estudos utilizando as referências dos estudos selecionados) e o método *forward* (busca novos estudos por meio das citações dos estudos selecionados) (Wohlin, 2014; Wohlin et al., 2000). Em ambos os casos, a técnica *snowballing* termina quando a convergência é alcançada e nenhum novo estudo pode ser encontrado. A Figura 2.4 apresenta o processo e resultados obtidos na condução do MS.

Figura 2.4: Processo e resultados do MS

Finalmente, foi possível identificar 95 estudos relacionados a diferentes tipos de ferramentas de suporte à LPS. Apesar do grande número de soluções, apenas 29 foram consideradas ferramentas de variabilidade de software de acordo com as definições de Pohl et al., (2005) e de Van Gurp et al., (2001). A Tabela 2.1 apresenta as ferramentas de variabilidade de software identificadas e classificadas conforme seu objetivo. A maioria das ferramentas está focada em gerenciar variabilidades (48,3%), seguida de ferramentas de verificação de restrição de variabilidade (24,1%), ferramentas de derivação de produto (17,2%), configuração de variabilidade (6,9%) e, finalmente, modelagem de variabilidade (3,5%). Mais detalhes dessa classificação podem ser visto no Apêndice A.

Tabela 2.1: Ferramentas classificadas por ano e tipo

Tipo da Ferramenta	**Ferramentas**
Gerenciamento de Variabilidade	CaptainFeature (2002); CVL (2009); CVM(2009); FeatureIDE (2005); FMT (2009); Gears (2002); Kumbang Tools (2007); Lisa (2012); PLUM (2008); pure::variants (2004); SOASPL(2013); VManage(2006); VMWT (2007); XFeature (2005)
Verificação de Restrição da Variabilidade	Clafer (2013), Fama (2007); FMP (2004); Hydra (2009); s2t2 (2009); SPLOT (2009); Variamos (2012);
Derivação de Produto	Covamof-VS (2004); DecisionKing(2007); Dopler (2007); GenArch (2007); Hephaestus (2009);
Modelagem de Variabilidade	Metadoc (1998);
Configuração de Variabilidade	Visit-FC (2008); WeConTin(2000);

Quanto à avaliação dessas ferramentas pode-se extrair que aproximadamente 50% foram avaliadas pela indústria. Também, foram extraídas as abordagens utilizadas para representar o modelo de variabilidade. O Modelo de *Feature* Baseado em Cardinalidade *Cardinality Based Feature Model* (CBFM)(Czarnecki et al., 2005b) é a notação mais utilizada pelas ferramentas identificadas. Tal notação é uma extensão do *Feature Oriented Domain Analys* (FODA)(Kang et al., 1990) que foi identificada na avaliação como a segunda mais utilizada entre as ferramentas. O tipo de notação selecionada pode interferir na representação gráfica das *features*.

Com relação às tecnologias foi observado que 19 ferramentas, isto é, 65,5% foram construídas como plugins. Plugins permitem maior comunicação e integração entre ferramentas externas de suporte. Além disso, pode-se observar que 23 ferramentas (79,3%) foram desenvolvidas utilizando a linguagem de programação Java; duas ferramentas (6,9%) foram implementadas em CSharp, uma ferramenta em Haskel (3,4%) e uma ferramenta em PHP (3,4%).

Considerando a existência de mecanismo de integração entre as ferramentas pode-se observar que a maior parte, 22 ferramentas (75,8%), tem algum tipo de mecanismo de integração com ferramentas de suporte à LPS. Além disso, 25 ferramentas (86,2%) possuem suporte a arquivos XML/XMI permitindo a importação e exportação de informação e modelos. Porém, quanto ao suporte a banco de dados, pode-se identificar que apenas três ferramentas (10,3%) possuem essa estrutura. É importante destacar que as informações foram extraídas de artigos e relatórios disponíveis na web, e que nem todos os detalhes técnicos estavam disponíveis nesses documentos dificultando a exatidão das informações extraídas.

Estudos Secundários sobre Ferramentas de Variabilidade de Software

Ainda no contexto de ferramentas de apoio ao GV, pode-se destacar a RSL realizada por Lisboa et al., (2010) que abrange as principais funcionalidades das ferramentas de análise de domínio e ferramentas de variabilidade de software. Tal estudo identificou 19 soluções com funcionalidades e objetivos análogos, que são: (i) `Planejamento`, responsável por recolher informações baseada em documentação de pré-análise, matriz de domínio, funções de avaliação e definição do escopo do domínio; (ii) `Modelagem`, representa o escopo de domínio baseado em diagramas e tabelas, variabilidade, *features* obrigatórias, regras de composição, identificação de grupo de *features*, tipos de relacionamentos entre as *features* e atributos de *features*; e (iii) `Validação`, funcionalidades baseada em documentações, relatórios e verificação de consistência. Configuração e documentação do produto são funcionalidades extras que também foram extraídas.

Outro estudo importante é a RSL de Pereira et al., (2015) que identificou 41 ferramentas de suporte ao GV incluindo ferramentas de variabilidade de software, análise de domínio e especificação de requisitos. O estudo de Pereira et al., (2015) utilizou a mesma

classificação de funcionalidades de Lisboa et al., (2010), sendo que 73% necessitam de suporte às atividades de planejamento; 81% dão suporte à derivação de produtos; e 71% dão suporte à importação/exportação. Mais detalhes sobre esses estudos secundários podem ser vistos no Apêndice A.

2.3 Arquitetura de Software, Terminologias e Conceitos

Arquitetura de software tem sido uma área de pesquisa fundamental em engenharia de software, devido ao seu profundo impacto sobre a produtividade e qualidade de software (SEI, 2016). De acordo com Bass et al. (2003) uma arquitetura de software é a estrutura do sistema que compreende elementos de software, propriedades externamente visíveis desses elementos e relacionamentos entre eles. O padrão IEEE 1471 define uma arquitetura de software como uma organização fundamental de um sistema integrado com seus componentes, relacionamentos, contexto e princípios que guiam seu projeto e evolução (ANSI/IEEE, 2000; Eeles e Cripps, 2010). Arquitetura de software envolve a descrição de elementos, suas interações, restrições e padrões que orientam a composição dos quais sistemas são construídos (Shaw e Garlan, 1996).

Em arquitetura de software, diversos conceitos e terminologias são utilizados com o objetivo de estabelecer um glossário comum para a comunidade acadêmica e empresarial. Dentre tais conceitos estão: arquitetura concreta, instância arquitetural, estilo arquitetural, padrão arquitetural, modelo de referência e arquitetura de referência.

`Arquitetura Concreta` é uma arquitetura de software de um sistema ou coleção de sistemas. Ela pode estar em conformidade com um estilo arquitetural, que pode ser influenciado pelos *stakeholders* e pelo domínio da aplicação. Esse tipo de arquitetura está ligada diretamente à `Instância Arquitetural`, a qual é uma arquitetura concreta específica aplicada em um determinado domínio.

`Estilo Arquitetural` e `Padrão Arquitetural` fornecem um conjunto de tipos de elementos pré-definidos, especificam as responsabilidades, e incluem regras e orientações para organizar as relações entre eles (Bass et al., 2003; Rozanski e Woods, 2005). Também, define um vocabulário de componentes e tipos de conectores, e um conjunto de restrições sobre como eles podem ser combinados (Shaw e Garlan, 1996).

Alguns estilos arquiteturais representam soluções conhecidas para os problemas de desempenho, outros para sistemas de alta segurança, outros têm sido utilizados com sucesso em sistemas de alta disponibilidade (Bass et al., 2003). Exemplos de estilos arquiteturais são: (i) Estilo cliente-servidor, separa o sistema em duas aplicações em que o cliente faz solicitações ao servidor; (ii) Estilo arquitetura em camadas, distribui os interesses da aplicação em grupos empilhados que são as camadas; e (iii) Estilo de arquitetura baseada em componente, o projeto da aplicação é decomposto em componentes reutilizáveis, funcionais ou lógicos, que expõem interfaces de comunicação bem definidas.

`Modelo de Referência` é uma divisão do sistema em funcionalidades que juntas cooperam para resolver um problema (Bass et al., 2003). Tais funcionalidades são produtos de experiência da equipe técnica e de negócio. Além disso, modelo de referência pode ser considerado uma representação abstrata de entidades, seus relacionamentos e comportamentos em um dado domínio de interesse, que tipicamente formam a base conceitual para o desenvolvimento de elementos concretos. Exemplos são modelos de negócio, modelos de informação e termos de glossários (Eeles e Cripps, 2010). De acordo com OASIS[3], um modelo de referência não está diretamente ligado às normas, tecnologias ou outros detalhes de práticas de implementação, mas procura oferecer uma semântica comum que pode ser usada de forma não ambígua por diferentes implementações.

`Arquitetura de Referência` fornece um ponto inicial para a construção de um novo sistema. Ela está associada com um domínio particular e tipicamente inclui diferentes padrões arquiteturais (Eeles e Cripps, 2010). Tal arquitetura será apresentada com mais detalhes na Seção 2.4.

2.4 Arquitetura de Referência

As exigências em aumentar a funcionalidade, qualidade e manter um padrão de software, além de reduzir custos levou a comunidade de engenharia de software a investigar e propor uma infraestrutura central, conhecida como Arquitetura de Referência (Eklund et al., 2005).

Arquitetura de referência é considerado um padrão pré-definido projetado para um determinado contexto de negócio (Nakagawa et al., 2011). Tal conceito faz uso de artefatos muitas vezes concebidos em projetos anteriores. Para isso, uma arquitetura de referência deve possuir conhecimento, experiência e informação adquirida (Angelov et al., 2009; Martínez-Fernández, 2013; Nakagawa et al., 2013).

Em síntese uma arquitetura de referência deve abranger três conceitos: `Elementos técnicos`, fornecem soluções em tecnologia abrangendo os padrões de projeto e padrões arquiteturais; `Modelo de negócio`, abrange os modelos e ciclo de vida do projeto, além de guiar as decisões no domínio técnico; e `Necessidade dos clientes`, as orientações são fornecidas por processos e considerações do domínio empresarial, cliente e usuário (Cloutier et al., 2010). A Figura 2.5 mostra a união da arquitetura de negócio, arquitetura técnica e de contexto do cliente representando uma arquitetura de referência. O único denominador comum são os requisitos onde as *features* e funções são modeladas de forma independente no produto.

Portanto, uma arquitetura de referência é uma arquitetura generalizada, com base nas melhores práticas e pode ser aplicada a várias soluções como no meio automotivo, aviação e robótica (Greefhorst e Proper, 2011; Nakagawa et al., 2011). O compilador é

[3]OASIS - https://www.oasis-open.org/committees/soa-rm/faq.php

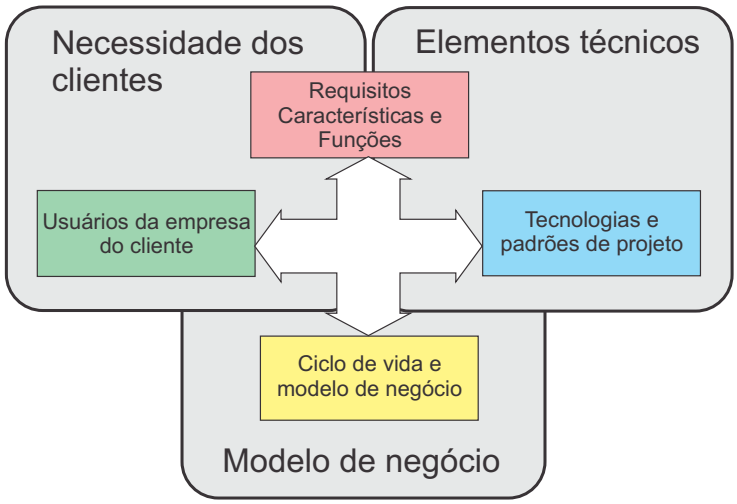


Figura 2.5: Modelo de arquitetura de referência. Traduzido de (Cloutier et al., 2010)

um exemplo de arquitetura de referência. Há uma noção geral dos elementos básicos de (por exemplo, *front-end*, *back-end*, tabela de símbolos), o que devem fazer, e como eles estão interligados. Se algum arquiteto de software for idealizar um novo compilador não irá começar do zero, mas sim começará com uma arquitetura de referência em mente (Hofmeister et al., 2000).

Um dos objetivos da arquitetura de referência é capturar o conhecimento implícito para simplificar o desenvolvimento de novos produtos (Cloutier et al., 2010). Uma das vantagens é a de facilitar a reutilização e, assim, obter maior economia por meio da redução de ciclos de tempo, custo e risco, além de um aumento da qualidade (Angelov et al., 2009; Martínez-Fernández, 2013). O controle de qualidade é realizado por meio da verificação de conformidade dos sistemas em desenvolvimento para arquitetura de referência (Buchgeher e Weinreich, 2013; Nakagawa et al., 2014). Essa verificação de conformidade pode ser realizada semiautomática e continuamente, automatizando passos importantes, como a extração da arquitetura do aplicativo real, a ligação de funções da arquitetura de referência para os elementos de uma arquitetura de aplicação específica, bem como a avaliação das regras de arquitetura de referência para uma arquitetura de aplicativo (Buchgeher e Weinreich, 2013). Além disso, outra vantagem está no aumento da interoperabilidade entre empresas (Cloutier et al., 2010).

2.4.1 Arquitetura de Referência x Arquitetura de LPS

Arquitetura de Referência e Arquitetura de LPS (*Product Line Architecture - PLA*) são conceitos frequentemente considerados equivalentes (Linden et al., 2007; Pohl et al., 2005), pois ambos têm como objetivo melhorar o desenvolvimento de sistemas de software por meio da padronização das arquiteturas. Apesar das semelhanças entre arquitetura de

referência e PLA existem diferenças importantes entre esses dois conceitos que devem ser mencionadas (Angelov et al., 2012; Eklund et al., 2012; Galster et al., 2013; Hofmeister et al., 2000; Martínez-Fernández et al., 2013b; Nakagawa et al., 2011). Os itens a seguir apresentam as diferenças entre arquitetura de referência e PLA:

- PLA geralmente aplica se a um conjunto de produtos dentro de uma organização ou empresa e pode ser baseada em uma arquitetura de referência (Hofmeister et al., 2000);
- PLAs são arquiteturas de referência, mas nem todas as arquiteturas de referência são PLAs;
- PLAs pertence ao núcleo de artefatos de uma LPS e por ser um artefato que indica `"o que o projeto é"`, enquanto que arquiteturas de referência ditam os padrões e princípios para a implementação, ou seja, `"o que o projeto deve ser"` (Eklund et al., 2012);
- PLAs são mais especializadas, com foco, por vezes, em um subconjunto específico de sistemas de um domínio de software. Além disso, fornecem soluções padronizadas para uma família menor de sistemas, e preocupados com as variabilidade entre os produtos (Hofmeister et al., 2000; Nakagawa et al., 2011);
- arquiteturas de referência são, em geral, um nível mais alto de abstração em comparação com PLA (Nakagawa et al., 2011).

Pode-se observar na Figura 2.6 que a PLA é produzida no núcleo de artefatos na engenharia de domínio e também envolve as atividades de gerenciamento que engloba as atividades que representam variabilidade em artefatos de software. Entretanto, uma arquitetura de referência possui um nível de abstração maior, envolvendo conceitos, terminologias, conhecimento relacionado ao projeto arquitetural e melhores práticas. A relação entre esses dois conceitos é que há uma arquitetura de referência e uma possível instância dessa arquitetura de referência pode ser uma PLA como também pode ser uma arquitetura de software tradicional. Consequentemente, uma PLA pode ser uma instância da arquitetura de referência. Quando se instancia uma PLA as variabilidades são resolvidas e como resultado obtém-se uma arquitetura de um produto.

2.4.2 Benefícios e Limitações das Arquiteturas de Referência

Apesar de muitas vantagens em se utilizar uma arquitetura de referência como, por exemplo, facilitar o reúso em projetos de software, há também algumas desvantagens que devem ser consideradas antes de iniciar uma atividade tão abrangente como projetar uma arquitetura de referência. Neste capítulo será citado alguns benefícios e limitações existentes em arquitetura de referência.

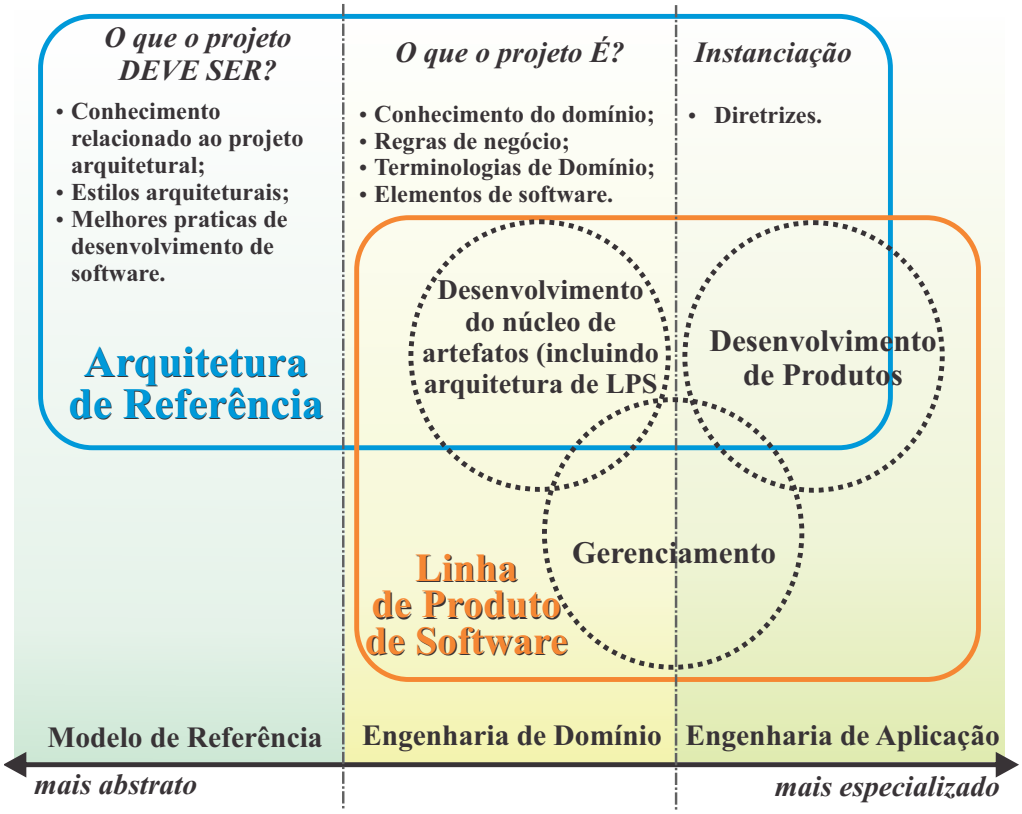


Figura 2.6: Diferenças entre arquitetura de referência e arquitetura de LPS. Traduzido de (Martínez-Fernández et al., 2013b)

Benefícios

A arquitetura de referência ajuda a garantir que os usuários finais e desenvolvedores possuam mais confiança na tecnologia desenvolvida por meio da reutilização de experiência em projetos, esse e outros benefícios da arquitetura de referência podem ser vistos a seguir:

- reduz o risco de implantação, pois conta com soluções conhecidas e testadas;
- simplifica a tomada de decisão;
- fornece modelos mais consistentes;
- aumenta a confiança da equipe de desenvolvimento por contar com soluções comprovadas;
- reduz as lacunas culturais entre as organizações, pois une a experiência e conhecimento de diferentes seguimentos em um modelo comum;
- fornece orientação para várias organizações que evoluem ou criam novas arquiteturas (Cloutier et al., 2010);

Somados aos benefícios citados anteriormente, Martinez-Fernandez et al., (2013a) uniram seu grupo de pesquisa GESSI com a empresa de TI Everis e desenvolveram um *framework* objetivando avaliar o uso de arquitetura de referência em projetos de TI. Desse estudo foi possível evidenciar mais motivos para se utilizar arquiteturas de referência. Dentre eles pode-se citar:

- fazendo uso de arquitetura de referência é possível iniciar o desenvolvimento de aplicações desde o primeiro dia seguindo decisões de arquitetura já tomadas;
- uma arquitetura de referência indica as diretrizes a serem seguidas pelos desenvolvedores de aplicativos;
- uma arquitetura de referência reduz a complexidade no desenvolvimento de aplicações porque parte das funcionalidades já estão resolvidas;
- uma arquitetura de referência aumenta o controle sobre aplicações por meio da sua homogeneidade. Isso gera uma redução do custo de manutenção das aplicações;
- uma arquitetura de referência ajuda a melhorar os processos de negócios de uma organização.

Limitações

Apesar das vantagens citadas acima deve-se considerar alguns aspectos que podem se tornar um risco em projetos de arquitetura de referência. Abaixo alguns exemplos:

- uma arquitetura de referência implica em um treinamento adicional para a equipe envolvida no projeto aumentado a curva de aprendizado;
- aplicação depende dos módulos reutilizáveis da arquitetura de referência. Se for necessário fazer mudanças em tais módulos, desenvolvedores terão de aguardar até que a alteração seja concluída;
- o uso de uma arquitetura de referência implica seguir suas orientações durante o desenvolvimento de aplicativos e adotar seu projeto arquitetural. Se as necessidades de negócios exigirem um tipo diferente de aplicação a arquitetura de referência limitaria essa flexibilidade.

2.4.3 Representação Arquitetural

A representação da arquitetura de software ou descrição arquitetural *Architectural Description* (AD) é essencial para garantir que os *stakeholders* possam compreender a arquitetura e se comuniquem por meio de todo ciclo de vida de projeto (Rozanski

e Woods, 2005). Apesar da maioria dessas arquiteturas ser representada de maneira informal (usando caixas e linhas, por exemplo), existem iniciativas que propõem meios para documentar tais arquiteturas de forma mais adequada (Nakagawa et al., 2014).

Para dar suporte ao desenvolvimento da base arquitetural algumas notações formais de modelagem podem ser utilizadas, entre elas estão *Unified Modeling Language* (UML) e *Architecture Description Language* (ADL). ADL é uma linguagem de finalidade específica para o tipo de modelos utilizados para definir a arquitetura de um sistema. A vantagem de usar uma ADL é que ela foi projetada especificamente para a criação de modelos em nível de detalhe arquitetural (Rozanski e Woods, 2005). A UML se tornou o padrão OMG[4] e oferece uma grande variedade de conceitos para a definição de estruturas e comportamentos de um sistema de software, além de ser um padrão industrial (Hilliard, 1999). A UML é uma linguagem gráfica para visualizar, especificar, construir e documentar os artefatos de um sistema complexo de software. O objetivo da UML é encapsular as melhores práticas em design de software em uma notação padrão e extensível (Rozanski e Woods, 2005).

Além das notações, a representação arquitetural também faz uso de outros mecanismos para representar a arquitetura de software. Entre eles estão os conceitos de `visão arquitetural` e `ponto de vista arquitetural`. Uma visão arquitetural é uma forma de retratar os aspectos ou elementos da arquitetura que são relevantes aos *stakeholders* (Rozanski e Woods, 2005). A visão arquitetural é uma representação de um ou mais aspectos particulares da arquitetura, que ilustra como a arquitetura aborda um ou mais interesses dos *stakeholders* (ANSI/IEEE, 2000). Pontos de vista arquitetural são um conjunto de padrões, modelos e convenções para construir um tipo de visão arquitetural.

Nakagawa et al., (2014) apresentam um conjunto de visões arquiteturais para representar o processo para criação de arquiteturas de referência, tais como: visão de módulo *(module view)*, visão em tempo de execução *(runtime view)*, visão de implantação *(deployment view)* e visão de processos *(process view)*.

2.4.4 Processo para a Construção de Arquitetura de Referência

O desenvolvimento de arquitetura de referência não é algo trivial e, por isso, exige conhecimento do domínio e processos sistemáticos para consolidar uma proposta de arquitetura de referência de qualidade. ProSA-RA, proposto por Nakagawa et al., (2014), é um processo iterativo que sistematiza os passos para construir, representar e avaliar arquitetura de referência. ProSA-RA, conforme ilustrado na Figura 2.7, é dividido em quatro etapas:

[4]Object Management Group - http://www.omg.org/

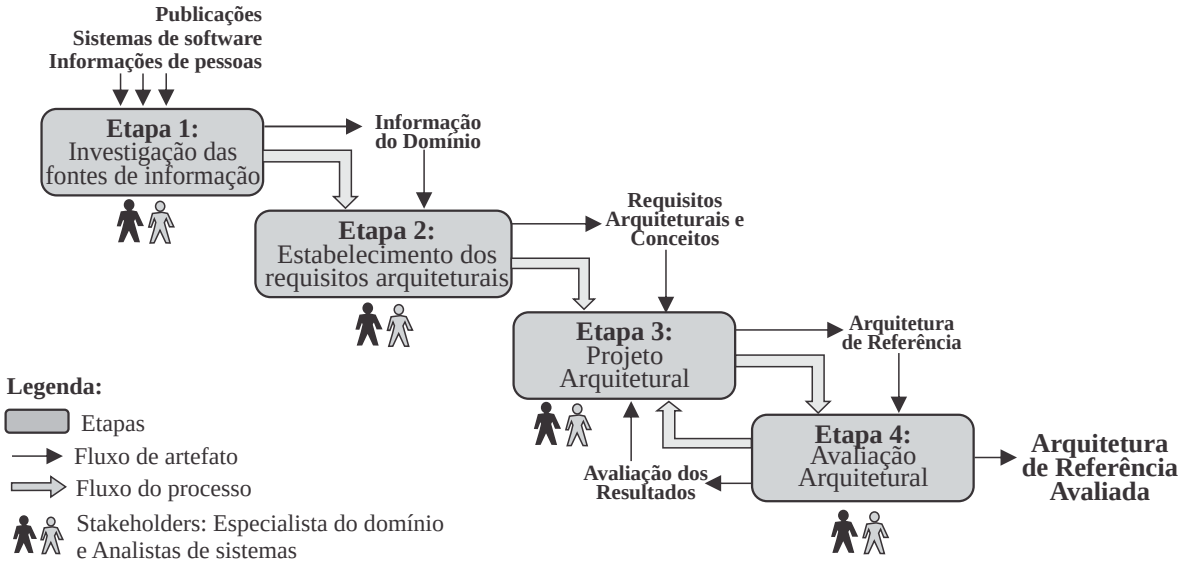


Figura 2.7: Etapas do processo ProSA-RA. Traduzido de (Nakagawa et al., 2014)

- **Etapa 1** - Investigação das fontes de informação: obtidas de pessoas, sistemas de softwares, publicações de revistas, livros, outros modelos de referência, ontologias de domínio, etc. As informações adquiridas devem envolver os processos e atividades que podem ser suportados por sistemas de software do domínio a ser desenvolvido. Deve-se, portanto, obter conhecimento abrangente sobre o domínio.

- **Etapa 2** - Estabelecimento dos requisitos arquiteturais: com base nas fontes selecionadas, um conjunto de requisitos de sistemas de software de domínio é identificado e, com base nesses requisitos, o conjunto de requisitos da arquitetura de referência é então estipulado.

- **Etapa 3** - Projeto arquitetural: nesta etapa a descrição arquitetural é realizada utilizando os requisitos arquiteturais e informações da Etapa 1. Textos e diagramas UML podem ser utilizados para representar a arquitetura de referência em diferentes visões arquiteturais, tais como: visão estrutural, visão de implantação e visão de tempo de execução.

- **Etapa 4** - Avaliação arquitetural: verifica a descrição da arquitetura junto a partes interessadas com o objetivo de detectar defeitos na descrição da arquitetura. O ProSA-RA recomenda o uso do *checklist* proposto por Santos et al., (2013) denominado *Framework for Evaluation of Reference Architectures* (FERA). Esse *checklist* avalia a completude das informações da arquitetura de referência, adequação da documentação para liberação ao público e a viabilidade de instanciação arquitetural com base na arquitetura de referência.

2.5 Considerações Finais

O GV tem sido amplamente adotada na indústria, devido à capacidade de aumentar o retorno de investimento (ROI) e melhorar a qualidade dos sistemas de software por meio da reutilização sistemática de artefatos de software. Diversos tipos de ferramentas de variabilidade de software para apoiar o GV têm sido desenvolvidos, tais como ferramentas de modelagem de variabilidade, configuração de variabilidade e derivação de produtos. Além disso, essas ferramentas destinam se a diferentes domínios, como militar, automotivo, energia, comércio eletrônico, aeroespacial, médico, governamental e telecomunicações. Dessa forma, é essencial agregar qualidade ao software que gerencia essas variabilidades em diferentes tipos de produtos e domínio. Soluções mais abstratas, como modelos de referência, foram desenvolvidos para permitir o aumento da qualidade em empresas que adotaram LPS e, consequentemente, guiar o gerenciamento de variabilidade. No entanto, observa-se a necessidade de uma solução mais próxima ao desenvolvimento, com base nas melhores práticas e que atenda também aspectos técnicos para que desenvolvedores possam idealizar novas ferramentas de variabilidade de software sem começar do zero. Nesse contexto, o próximo capítulo apresenta a especificação de uma arquitetura de referência para ferramentas de variabilidade de software, assim como o processo para seu desenvolvimento.

3

VMTools-RA

3.1 Considerações Iniciais

Neste capítulo é apresentada a arquitetura de referência para ferramentas de variabilidade de software, denominada *Variability Management Tools - Reference Architecture* (VMTools-RA). Essa arquitetura de referência tem por objetivo apoiar a fase de projeto arquitetural no desenvolvimento e evolução de ferramentas de variabilidade de software. A construção da VMTools-RA seguiu o processo ProSA-RA para o desenvolvimento de arquiteturas de referências. Tal processo prevê um conjunto de quatro etapas, desde a identificação das fontes de informação do domínio, até a fase de avaliação da arquitetura de referência.

Este capítulo está organizado de acordo com as etapas do ProSA-RA. Na Seção 3.2 são descritas as fontes de informação utilizadas na identificação dos requisitos arquiteturais da VMTools-RA. A Seção 3.3 apresenta a identificação dos requisitos arquiteturais para projetar a arquitetura de referência. Na Seção 3.4, o projeto da VMTools-RA é representado pela visão geral e outras seis visões arquiteturais. A Seção 3.5 discorre de forma geral sobre a avaliação da VMTools-RA. Na Seção 3.6 são apresentadas as considerações finais deste capítulo.

3.2 Etapa 1 - Investigação das Fontes de Informação

Na primeira etapa do ProSA-RA, foram identificadas diferentes fontes para obtenção de informações sobre o domínio de ferramentas de variabilidade de software. Em razão da falta de arquiteturas de referência para tal domínio, foram analisados três grupos de informações: (i) na Seção 3.2.1, serão apresentadas padronizações aplicáveis à LPS; (ii) MS sobre ferramentas de variabilidade de software apresentados na Seção 3.2.2; e (iii)

estudos secundários sobre ferramentas de variabilidade de software apresentados na Seção 3.2.3. A seguir, são discutidos os detalhes a respeito das fontes de informação utilizadas no estabelecimento da VMTools-RA.

3.2.1 Grupo 1: Padronizações Aplicáveis à LPS e ao GV

Com o objetivo de direcionar os grandes desafios na compreensão de LPS e GV e incorporar conceitos centrais da Engenharia de LPS buscou-se por padronizações internacionais estabelecidas. As ISO/IEC 26550 (ISO/IEC, 2013a) e ISO/IEC 26555 (ISO/IEC, 2013b) apresentadas na Seção 2.2, são padrões internacionais que estabelecem importantes processos e modelos de referência no contexto de LPS e, também evidências importantes dos processos relacionados ao GV.

A ISO/IEC 26550 incorpora os conceitos centrais da Engenharia de LPS e apoia o desenvolvimento de LPS. Essa norma fornece um modelo de referência contendo representações abstratas dos principais processos da engenharia de LPS. Esses processos evidenciaram a necessidade de uma infraestrutura que atenda, não só a lógica da aplicação, mas também, a estrutura organizacional ao implementar o GV.

A ISO/IEC 26555 tem o foco no gerenciamento técnico de LPS e envolve processos relacionados ao GV, que foram essenciais para o entendimento dos principais conceitos na construção da VMTools-RA. Além disso, tal padronização apresenta uma completa documentação sobre o grupo de processos que compõem o GV, sendo base para a especificação da arquitetura de referência. De acordo com a ISO/IEC 26555, o grupo de processos de GV requer explícita documentação das variabilidades e deve ser gerenciado corretamente por todo o ciclo de vida da engenharia de domínio e aplicação. Tal grupo de processos de GV envolve os seguintes processos:

- `Gerenciamento de Modelos de Variabilidade`: mantém a integridade do modelo de variabilidade da engenharia de domínio e da aplicação;
- `Gerenciamento da Resolução de Variabilidade`: mantém informação de resolução *(binding)* para alcançar o desenvolvimento eficiente de um produto e reutilização pró-ativa da plataforma de LPS;
- `Gerenciamento da Documentação de Variabilidade`: fornece suporte às descrições detalhadas dos modelos de variabilidade para fornecer as justificativas das decisões relacionadas às variabilidades;
- `Rastreamento de Variabilidade`: estabelece e mantém links de rastreabilidade entre elementos do modelo de variabilidade e ativos do domínio/aplicação associados; e
- `Controle e Evolução de Variabilidade`: lida com solicitações de alterações, análise de impacto das alterações, execução, verificação e validação das alterações.

É importante destacar que não foram encontrados modelos de referência específicos para o desenvolvimento de ferramentas de variabilidade de software além da ISO/IEC 26555, tampouco, arquiteturas de referência nesse contexto.

3.2.2 Grupo 2: Ferramentas de Variabilidade de Software

A fim de identificar os principais aspectos no desenvolvimento de ferramentas de variabilidade de software, um estudo secundário na forma de um MS foi realizado. O processo de condução, descrição das ferramentas e análise dos resultados foi apresentado brevemente na Seção 2.2 e com detalhes no Apêndice A. Em suma, foram identificadas **29** ferramentas de variabilidade de software que possuem diferentes características, funcionalidades e objetivos. Tais ferramentas fornecem conceitos e informações necessários na aquisição de conhecimento sobre o domínio de GV.

A Figura 3.1 apresenta uma visão conceitual da relação entre ferramentas de variabilidade de software e arquitetura de referência e resume a perspectiva da pesquisa sobre a criação de arquitetura de referência. Observe que uma ou mais ferramentas de variabilidade de software podem ser utilizadas para fornecer informações para projetar uma arquitetura de referência. Essa arquitetura de referência pode ser utilizada como base para novas ferramentas de variabilidade de software.

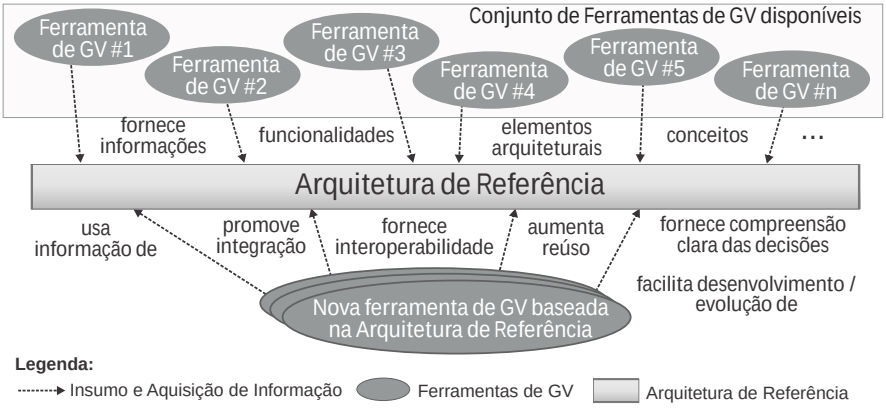


Figura 3.1: Relação entre ferramentas de variabilidade de software e arquitetura de referência. Traduzido de (OliveiraJr e Allian, 2015)

Dentre as principais características encontradas após a execução do MS sobre ferramentas de variabilidade de software estão os tipos de soluções disponíveis, tais como: derivação de produtos, configuração da variabilidade, modelagem da variabilidade e ferramentas de variabilidade de software com e sem solucionadores lógicos. Essa diversidade evidencia a existência de diferentes objetivos e mecanismos para realizar o GV. Além disso, a maior parte dessas ferramentas fornece suporte a arquivos XML, são *open source* e foram implementadas como *plugins*. Também foi possível evidenciar que aproximadamente 50% dessas ferramentas foram avaliadas pela indústria o que fornece um grau relevante de

maturidade dessas soluções. Além disso, diversas características foram identificadas nas ferramentas e agrupadas em três grupos: (i) abordagem de variabilidade, fornece os tipos de abordagens identificadas nas ferramentas e as notações utilizadas por tais abordagens; (ii) interoperabilidade, fornece os tipos de soluções de integração identificadas nessas ferramentas, tais como, ferramentas de raciocínio, ferramentas de requisitos, ferramentas de desenvolvimento orientadas a modelos; e (iii) tecnologias, fornece as tecnologias identificadas para a construção dessas ferramentas. Tais características foram resumidas na Tabela 3.1.

Essas características são importantes para o projeto de arquitetura de referência, mas não suficientes. Outras fontes de conhecimento foram necessárias para descobrir funcionalidades importantes do contexto de GV e são apresentadas no próximo grupo de informação.

Tabela 3.1: Características das ferramentas identificadas no MS

Abordagem de Variabilidade	**Notação de Variabilidade**
Modelos de Features	FODA, CBFM, OVM, CVL
Linguagem Estruturada	Clafer Language
Modelos de Decisão	Dopler
Ontologias	Kumbang Ontology
Interoperabilidade	**Soluções de Integração**
Ferramentas de Raciocínio	BDD, SAT Solvers CSP Solvers, Choco Solvers
Ferramentas de Requisitos	IBM Rational Doors, CaliberRMTM
Desenvolvimento Orientado a Modelo	MATLAB/Simulink, AUTOSAR, Enterprise Architect, IBM Rational Rhapsody, Eclipse Modeling Framework, Magic Draw, Rational Software Architect
Configuração de Software	CVS, Subversion, ClearCase, Serena Dimensions, Perforce, IBM Rational Synergy, IBM Rational Team Concert
Qualidade e Teste	ClearQuest, IBM Rational Quality
Documentação	Microsoft Office, MadCap Software Flare
Recursos Empresariais	SAP ERP
Relatórios	BIRT
Ferramentas de Modelagem	Guidsl, ModelBus (MDDi), Papyrus
Paradigmas de programação	Ahead, FeatureC++, FeatureHouse, AspectJ, DeltaJ,
Tecnologias	
Plataformas	EclipseIDE, VisualStudio
Visualização da Variabilidade	Diagramas de Features, Visão em Árvore das Features, Tabela de Features, Linguagem Textual, Mapas de Fluxo
Tecnologias de Integração	API REST, SOAP, Web Services, Interfaces HTTP/TCP, XML/XMI

3.2.3 Grupo 3: Estudos Secundários sobre Ferramentas de Variabilidade de Software

Os trabalhos deste grupo de informação referem se basicamente às RSL realizadas por Lisboa et al., (2010) e Pereira et al., (2015) sobre ferramentas de variabilidade de software

e podem ser vistos com mais detalhes na Seção 2.2 e no Apêndice A. Esses estudos foram considerados por apresentarem uma classificação de funcionalidades que devem ser consideradas em ferramentas que apoiam o GV (Lisboa et al., 2010; Pereira et al., 2015). Tais funcionalidades podem contribuir na elaboração de uma arquitetura de referência para esse domínio. Tais funcionalidades foram agrupadas em sete categorias: (i) configuração; (ii) interoperabilidade; (iii) modelagem; (iv) planejamento; (v) suporte técnico; (vi) usabilidade; e (vii) validação.

Configuração: trata de questões associadas à configuração e derivação de produtos de software fazendo uso de modelos de variabilidade. Essa categoria também é responsável por definir o momento e local em que as variabilidades ocorrerão. As funcionalidades identificadas nessa categoria são:

(1) Derivação de produto: consiste em selecionar/modificar cópias do conjunto de ativos para derivar um produto;

(2) Configuração de produto: é realizado com a identificação de um conjunto de ativos reutilizáveis;

(3) *Binding Time*: permite a realização das variabilidades e dos pontos de variação em diferentes momentos do ciclo de vida de desenvolvimento.

Interoperabilidade: representa os meios utilizados para realizar a integração entre ferramentas de variabilidade de software com outras soluções que fornecem suporte a esse gerenciamento, tais como ferramentas de análise de domínio, especificação de requisitos e modelagem de diagramas utilizando representação UML. Consiste nas seguintes funcionalidades:

(4) Importação/exportação: fornece a função de importação/exportação;

(5) Arquivos XML/XMI: permite fazer uso de arquivos de entrada e saída nos formatos XML/XMI;

(6) Integração: permite a interoperabilidade entre aplicações.

Modelagem: representam os mecanismos e técnicas utilizadas para modelar as variabilidades, além de definir as regras, dependências e restrições entre os modelos de variabilidade. As funcionalidades identificadas foram:

(7) Atributos de *Features*: permite a inclusão de informações específicas para cada *feature*;

(8) Regras de composição: cria restrições para representar e relacionar as *features*;

(9) *Features* obrigatórias: representam as *features* que estarão sempre presente nos produtos;

(10) Variabilidades: representam as variabilidades que uma *feature* pode ter.

Planejamento: responsável por coletar informações, identificar e representar as variabilidades, além de projetar os pontos de variação e dependências entre as *features*. Foram identificadas as seguintes funcionalidades para essa categoria:

(11) Documentação de pré-análise: armazena e recupera as informações para identificação das *Features*;

(12) Definição do escopo: identifica as *features* que devem fazer parte da infraestrutura de reutilização.

Suporte Técnico: representam funcionalidades que fornecem suporte ao processo de GV, como a geração de códigos e documentação. As funcionalidades identificadas são:

(13) Documentação do produto: fornece documentação para cada produto incluindo a versão do sistema;

(14) Disponível *online*: identifica se a ferramenta está disponível para acesso *online*;

(15) Código aberto (*open source*): permite acesso ao código fonte do software;

(16) Gerador de código fonte: responsável pela geração de código-fonte com base no modelo de variabilidade.

Usabilidade: fornece mecanismos para auxiliar ao usuário no processo de GV. Consiste nas seguintes funcionalidades:

(17) Guia do usuário: fornece documentação para guiar o usuário ao utilizar a ferramenta;

(18) Interface gráfica do usuário: fornece um ambiente mais intuitivo para os usuários.

Validação: responsável por funcionalidades que realizam a validação, análise, verificação de consistência dos produtos configurados. As funcionalidades identificadas são:

(19) Relatórios: responsável por gerar relatórios sobre as informações disponíveis no domínio;

(20) Rastreabilidade: relaciona as *features* existentes de um domínio com os requisitos;

(21) Verificação de consistência: verifica se o domínio gerado segue as regras de composição criadas.

A Tabela 3.2 contém as ferramentas de variabilidade de software identificadas no MS classificadas de acordo com as categorias citadas. A maior parte das ferramentas abrange as funcionalidades de modelagem, configuração e validação. Isso corrobora a importância de tais grupos de funcionalidades no projeto da VMTools-RA. Além disso, alguns requisitos não funcionais, tais como, usabilidade e interoperabilidade, também foram analisados para serem considerados no projeto da arquitetura de referência.

Tabela 3.2: Funcionalidades das ferramentas de variabilidade de software

Funcionalidades																				
	Configuração			**Interoperab.**			**Modelagem**				**Planejam.**		**Suporte Técnico**				**Usabilidade**		**Validação**		
Ferramentas	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
CaptainFeature		✓	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓			✓		✓	✓
Clafer		✓		✓		✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓		✓
Covamof-VS	✓	✓	✓	✓	✓	✓		✓	✓	✓								✓		✓	✓
CVL Tool		✓	✓	✓		✓	✓	✓	✓	✓				✓	✓		✓	✓		✓	
CVM		✓	✓		✓	✓		✓	✓	✓			✓	✓	✓		✓	✓		✓	✓
DecisionKing	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓						✓		✓	✓
DOPLER	✓			✓			✓	✓	✓	✓	✓	✓						✓	✓	✓	✓
FAMA		✓	✓	✓	✓		✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓		✓
FeatureIDE		✓			✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓			✓
FMP		✓	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓		✓			✓
FMT		✓		✓	✓	✓	✓	✓	✓	✓								✓		✓	✓
GEARS		✓	✓	✓	✓		✓	✓	✓	✓			✓	✓		✓	✓	✓	✓	✓	✓
GENARCH	✓	✓		✓	✓	✓	✓	✓	✓	✓			✓					✓			✓
Hephaestus	✓	✓		✓	✓	✓				✓								✓		✓	✓
Hydra		✓		✓	✓	✓	✓	✓	✓	✓								✓			✓
Kumbang tools		✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓		✓	✓	✓		✓
LISA toolkit				✓			✓	✓	✓	✓	✓	✓				✓		✓		✓	
Metadoc FM				✓	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓		✓	✓
PLUM		✓		✓	✓	✓			✓	✓			✓	✓			✓	✓			✓
pure variants		✓		✓	✓	✓	✓	✓	✓	✓			✓	✓		✓	✓	✓	✓		✓
s2t2		✓		✓	✓	✓		✓	✓	✓				✓	✓			✓			
SOASPL				✓	✓	✓				✓						✓		✓		✓	✓
SPLOT		✓		✓			✓	✓	✓	✓				✓				✓	✓	✓	✓
Variamos		✓		✓	✓	✓		✓	✓	✓			✓	✓	✓		✓	✓	✓		✓
Visit-FC		✓						✓	✓	✓								✓			✓
V-Menage		✓						✓	✓	✓								✓			
VMWT		✓	✓				✓	✓	✓	✓								✓			
WeCoTin		✓		✓	✓	✓	✓	✓	✓	✓								✓			✓
XFEATURE		✓		✓	✓	✓	✓	✓	✓	✓	✓			✓	✓			✓			✓

3.3 Etapa 2 - Estabelecimento dos Requisitos Arquiteturais

As informações e conceitos relacionados na primeira etapa do processo ProSA-RA são utilizados para estabelecer os requisitos arquiteturais da arquitetura de referência. Os requisitos arquiteturais da VMTools-RA foram divididos em dois grupos: (RA1) Requisitos arquiteturais para GV e; (RA2) Requisitos arquiteturais para evolução do GV. O primeiro grupo de requisitos é direcionado a parte mais técnica do GV fornecendo informações para a sua implementação. Tal grupo está direcionado aos especialistas de domínio, arquitetos de software e desenvolvedores. O segundo grupo de requisitos arquiteturais está direcionado aos elementos organizacionais da empresa e à análise de mercado, com o objetivo de manter a qualidade e a constante evolução do GV. Tal grupo de requisitos está direcionado aos executivos, profissionais de marketing, especialistas do domínio e gerentes de qualidade. É importante destacar que a lista de *stakeholders* mencionada pode variar em diferentes domínios e, por isso, é inviável prever todos os interessados necessários para atender a cada requisito arquitetural. A definição de cada *stakeholder* é apresentada na Seção 3.4.3.

3.3.1 Requisitos Arquiteturais para GV

O primeiro grupo com nove requisitos arquiteturais está relacionado ao GV e suas principais atividades.

RA1.1 A arquitetura de referência deve possibilitar o gerenciamento de ativos de domínio (por exemplo, *features*, componentes arquiteturais e requisitos) por meio da coleta de informações de variabilidades (por exemplo, pontos de variação e suas variantes, *binding time* para todos os pontos de variação, dependências e restrições) identificadas e analisadas durante a análise de domínio para construir/projetar os modelos de variabilidade. *Stakeholder* responsável: Especialista de Domínio.

RA1.2 A arquitetura de referência deve possibilitar a construção de modelos de variabilidade independente da abordagem selecionada (por exemplo, Modelo de *Feature*, Modelo de Decisão, Ontologias e Linguagens Estruturadas) para representar e gerenciar as variabilidade. *Stakeholders* responsáveis: Arquiteto de Software e Desenvolvedores.

RA1.3 A arquitetura de referência deve possibilitar que os modelos de variabilidade possam considerar as regras de composição (por exemplo, cardinalidade, regras de dependências e restrições das variabilidades) derivadas durante a análise de domínio e/ou requisitos. *Stakeholders* responsáveis: Especialista de Domínio e Arquiteto de Software.

RA1.4 A arquitetura de referência deve possibilitar o gerenciamento, estabelecimento e manutenção de links de rastreabilidade dos modelos de variabilidade com os ativos de domínio e/ou requisitos. *Stakeholders*: Arquiteto de Software e Desenvolvedores.

RA1.5 A arquitetura de referência deve possibilitar a documentação detalhada dos modelos de variabilidade anotando seus elementos (por exemplo, pontos de variação, variantes, dependências e restrições), e manter políticas de links de rastreamento entre modelos de variabilidade e ativos de domínio associados para manutenção da consistência. *Stakeholders* responsáveis: Especialista de Domínio.

RA1.6 A arquitetura de referência deve possibilitar a validação da conformidade entre os modelos de variabilidade e suas anotações fornecendo uma função para verificação automática da consistência (por exemplo, por meio da utilização de verificação aritmética de restrição, *reasoners e logic solvers*). *Stakeholders* responsáveis: Desenvolvedores.

RA1.7 A arquitetura de referência deve possibilitar a geração de relatórios para verificação de conflitos e inconsistências de todo modelo de variabilidade, a fim de validar os resultados. *Stakeholder* responsável: Desenvolvedores.

RA1.8 A arquitetura de referência deve possibilitar o gerenciamento de resolução das variabilidades *(variability binding)* que mantém informação necessária para resolver as variabilidades apropriadamente. Essa tarefa inclui o momento de resolução das variabilidades *(binding time)*, restrições entre as resoluções *(binding contraints)* e a razão pela qual uma variante foi resolvida. A escolha do tempo de resolução é independente da modelagem das variabilidades. *Stakeholders* responsáveis: Especialista de Domínio e Arquiteto de Software.

RA1.9 A arquitetura de referência deve possibilitar a seleção de um mecanismo de variabilidade, que são técnicas de representação / implementação das variabilidades de acordo com o tempo de resolução *(binding time)* em uma fase especifica do ciclo de vida (por exemplo, requisitos, projeto, realização, compilação, pós-tempo de compilação, tempo de execução e testes). Esse mecanismo permite adiar decisões relativas às variantes a serem selecionadas durante o ciclo de desenvolvimento, otimizando os objetivos gerais do negócio. *Stakeholders* responsáveis: Especialista de Domínio e Arquiteto de Software.

3.3.2 Requisitos Arquiteturais para Evolução do GV

O segundo grupo, com 12 requisitos, está relacionado às atividades de apoio organizacional, manutenção e evolução das ferramentas de variabilidade de software.

RA2.1 A arquitetura de referência deve possibilitar o gerenciamento do controle e evolução das variabilidades com objetivo de fornecer suporte a alterações no modelo de variabilidade por meio de requisições/*feedbacks*, melhorando o processo de evolução das variabilidades. *Stakeholders* responsáveis: Especialista do Domínio e Arquiteto de Software.

RA2.2 A arquitetura de referência deve possibilitar a implementação de análise de impacto para determinar quais alterações são necessárias para resolver as inadequações

das variabilidades, e quais os efeitos dessas alterações na LPS. *Stakeholders*: Gerente de Qualidade, Especialista de Domínio e Arquiteto de Software.

RA2.3 A arquitetura de referência deve possibilitar lidar com as capacidades organizacionais para iniciação, execução, controle e melhoria do GV. O controle do planejamento e gestão serve para definir e manter políticas e normas da organização, medir o desempenho de processos e, se necessário, planejar ações corretivas para solucionar problemas. *Stakeholders* responsáveis: Executivo e Especialista de Domínio.

RA2.4 A arquitetura de referência deve possibilitar o suporte de um ambiente de trabalho com comunicação e colaboração para compartilhar conhecimento em GV, independente da localização geográfica. *Stakeholders* responsáveis: Gerente Técnico, Arquiteto de Software e Desenvolvedor.

RA2.5 A arquitetura de referência deve possibilitar fornecer orientações (por exemplo, fornecendo um guia com *checklist online*) para apoiar a verificação dos modelos de variabilidade e suas regras de composição, rastreabilidade e tempo de resolução. *Stakeholders* responsáveis: Especialista de Domínio e Arquiteto de Software.

RA2.6 A arquitetura de referência deve possibilitar obter *feedbacks* (ou fornecer *feedbacks*) de usuários (por exemplo, ambiente baseado na Web para a coleta de opiniões sobre recursos), notificar os usuários sobre as mudanças no GV. *Stakeholders* responsáveis: Arquiteto de Software, Desenvolvedores e Usuários.

RA2.7 A arquitetura de referência deve possibilitar o apoio à análise de *trade-off* entre as alternativas de resolução das variabilidades, avaliação da flexibilidade e capacidade de reutilização de um mecanismo para variabilidades. *Stakeholders* responsáveis: Executivo, Profissional de Marketing e Especialista de Domínio.

RA2.8 A arquitetura de referência deve possibilitar o gerenciamento e armazenamento de múltiplas versões de modelos de variabilidade, documentação das variabilidades, etc. *Stakeholder* responsável: Desenvolvedores.

RA2.9 A arquitetura de referência deve possibilitar suporte quanto à importação/ exportação de ativos de variabilidade e informações relevantes relacionadas às variabilidades contidas em repositórios ou outras ferramentas disponíveis. *Stakeholders* responsáveis: Arquiteto de Software e Desenvolvedores.

RA2.10 A arquitetura de referência deve possibilitar acesso e busca em repositório de informações das variabilidades, persistir modelos de variabilidade, e manter registro dos links de rastreabilidade para permitir *roll-back*, quando necessário. *Stakeholders* responsáveis: Arquiteto de Software e Desenvolvedores.

RA2.11 A arquitetura de referência deve possibilitar uma usabilidade eficiente (para que o usuário, depois de o saber usar, possa atingir uma boa produtividade) com múltiplas

visões gráficas de interface de usuário para representar modelos de variabilidade, informações das variabilidades, zoom, consultas, etc. *Stakeholder* responsável: Arquiteto de Software e Desenvolvedores.

RA2.12 A arquitetura de referência deve possibilitar a opção de integração com ferramentas de análise de domínio e/ou de gerenciamento de requisitos por meio de um *middleware* para obter os ativos que serão utilizados no GV. *Stakeholders* responsáveis: Arquiteto de Software, Especialista de Domínio e Desenvolvedores.

3.3.3 Rastreamento dos Requisitos

O rastreamento de requisitos é uma técnica do gerenciamento de requisitos que possibilita uma compreensão dos relacionamentos existentes entre requisitos e soluções de software por meio do ciclo de vida de desenvolvimento em ambas as direções (*backward* e *forward*) (Gotel e Finkelstein, 1994; Ozkaya e Ömer Akin, 2007). O rastreamento de requisitos possibilita análise de impacto, manutenção e evolução do software (Gotel e Finkelstein, 1994; Nuseibeh e Easterbrook, 2000).

A maneira mais comum de representar o rastreamento de requisitos é utilizando uma matriz, que também é conhecida por matriz de rastreamento entre requisitos (Wiegers e Beatty, 2013). É possível realizar diversos tipos de mapeamentos entre requisitos, dentre eles estão o mapeamento de requisitos entre requisitos e o mapeamento entre requisitos e fontes de informações (Wiegers e Beatty, 2013). Esses mapeamentos possibilitam a identificação de dependências entre esses requisitos.

Nessa dissertação, foi projetada a matriz de rastreamento entre os 21 requisitos arquiteturais com os três grupos de informação apresentados na Seção 3.2. Pode-se observar na Tabela 3.3 que os requisitos identificados foram extraídos, em sua maioria, do primeiro grupo de informação correspondente às normas internacionais.

Além disso, foi projetada a matriz de rastreamento entre requisitos e requisitos apresentada na Tabela 3.4. Tal matriz mostra as ligações entre os requisitos, onde a linha é dependente da coluna e a coluna depende da linha. Essa matriz mostra de que forma um requisito influência em outros possibilitando a análise de impacto caso haja uma alteração no requisito.

Tabela 3.3: Matriz de rastreamento entre requisitos e fontes de informação

Requisitos	G1: ISO/IEC 26550 e ISO/IEC 26555	G2: Mapeamento Sistemático	G3: Estudos Secundários
RA1.1	X		
RA1.2	X	X	X
RA1.3	X		X
RA1.4	X		X
RA1.5	X	X	X
RA1.6	X	X	X
RA1.7		X	X
RA1.8	X		X
RA1.9	X		
RA2.1	X		
RA2.2	X		
RA2.3	X		
RA2.4	X		
RA2.5			X
RA2.6	X		
RA2.7	X		
RA2.8		X	X
RA2.9		X	X
RA2.10		X	X
RA2.11		X	X
RA2.12		X	X
Total	**14**	**9**	**13**

Tabela 3.4: Matriz de rastreabilidade entre requisitos arquiteturais

REQ	RA1.1	RA1.2	RA1.3	RA1.4	RA1.5	RA1.6	RA1.7	RA1.8	RA1.9	RA2.1	RA2.2	RA2.3	RA2.4	RA2.5	RA2.6	RA2.7	RA2.8	RA2.9	RA2.10	RA2.11	RA2.12
RA1.1																					
RA1.2	X																				
RA1.3	X	X																			
RA1.4	X	X																			
RA1.5	X	X	X	X																	
RA1.6		X	X																		
RA1.7		X	X			X															
RA1.8	X															X					
RA1.9	X							X								X					
RA2.1	X	X	X	X	X	X	X	X	X		X				X	X					
RA2.2	X	X	X	X	X	X	X	X	X						X	X					
RA2.3	X																				
RA2.4		X			X		X								X						
RA2.5	X	X	X	X	X	X	X	X	X												
RA2.6										X		X	X								
RA2.7	X							X	X		X				X						
RA2.8	X	X			X																
RA2.9	X	X	X	X	X	X	X	X	X										X		
RA2.10	X	X	X	X	X	X	X	X	X												
RA2.11	X	X		X	X		X	X	X												
RA2.12	X																				

3.4 Etapa 3 - Projeto da Arquitetura de Referência

A terceira etapa do processo ProSA-RA envolve o projeto da arquitetura de referência. Todos os requisitos arquiteturais identificados na etapa anterior são utilizados como base para a realização do projeto arquitetural. É importante destacar que a versão da VMTools-RA apresentada nesta seção já está refatorada com base nas avaliações dos especialistas. A documentação da primeira versão é apresentada no Apêndice E.

Os detalhes desse projeto são apresentados em um alto nível de abstração. Além disso, este documento fornece um guia de melhores práticas para o desenvolvimento de ferramentas de variabilidade de software. Na Seção 3.4.1, é apresentada uma descrição geral da VMTools-RA. Na Seção 3.4.2, é apresentado o escopo da VMTools-RA. Para uma melhor documentação da arquitetura de referência, também são apresentados os objetivos, identificação de possíveis riscos para a arquitetura de referência, *stakeholders* e interesses arquiteturais (Seção 3.4.3). Além disso, serão apresentadas seis visões da arquitetura de referência distribuídas em quatro pontos de vista arquitetural, tais como:

i) **Ponto de Vista Transversal:** Visão Conceitual, Visão de Variabilidade e Visão de Decisão de Projeto (Seção 3.4.4);

ii) **Ponto de Vista de Tempo de Execução:** Visão de Processo (Seção 3.4.5);

iii) **Ponto de Vista do Código Fonte:** Visão de Módulo (Seção 3.4.6); e

iv) **Ponto de Vista de Implantação:** Visão de Implantação (Seção 3.4.7).

Para cada visão arquitetural, será apresentado uma descrição, *stakeholders*, interesses capturados e a representação gráfica da arquitetura de referência. A notação UML foi utilizada para representar a maioria das visões arquiteturais. Apesar da existência de outras visões arquiteturais, tais como, visão arquitetural de serviços de sistema, visão arquitetural de componentes colaborativos, visão estrutural, visão lógica, entre outras (Nakagawa et al., 2014), acredita-se que as visões arquiteturais apresentadas neste trabalho são suficientes para a compreensão desta arquitetura de referência. É importante destacar que os conceitos apresentados nas visões arquiteturais podem ser implementados com diferentes tecnologias, linguagens de programação e diferentes notações para representar modelos de variabilidade.

3.4.1 Visão Geral da VMTools-RA

O projeto da VMTools-RA teve como base informações extraídas de ferramentas de variabilidade de software e, o mais importante, informações extraídas do modelo de referência ISO/IEC 26555 (ISO/IEC, 2013b), que envolve o gerenciamento técnico de engenharia de LPS.

A visão geral da VMTools-RA, representada na Figura 3.2, retrata os principais componentes, relação entre eles e as dependências com hardware. Optou-se por não definir o estilo arquitetural da VMTools-RA, uma vez que a maioria das ferramentas de variabilidade de software identificadas no MS (Apêndice A) foi desenvolvida como plugin do Eclipse IDE. Porém, como sugestão, pode-se adotar estilo arquitetural em camadas, estilo cliente servidor e padrão arquitetural *Model View Control* (MVC), pois tais estilos arquiteturais foram utilizados para desenvolver algumas das ferramentas de variabilidade de software identificadas no MS.

Na visão geral da VMTools-RA, podem-se notar quatro conjuntos de elementos: (i) `Elementos do Gerenciamento de Variabilidade`; (ii) `Elementos de Análise de Domínio`; (iii) `Elementos de Suporte`; e (iv) `Elementos do Gerenciamento Organizacional`. Além disso, há uma camada de `Middleware` e possíveis tipos de ferramentas disponíveis na indústria/academia para serem integradas. Outro elemento importante é o `Repositório` que fornece mecanismos para persistir as informações da aplicação. Para a identificação de tais elementos foram utilizados os requisitos arquiteturais identificados na Etapa 2 do ProSA-RA. A Tabela 3.5 apresenta um mapeamento dos requisitos arquiteturais com os elementos da VMTools-RA. Detalhes sobre o funcionamento dos elementos da VMTools-RA serão apresentados nas próximas visões arquiteturais.

Tabela 3.5: Mapeamento entre requisitos e elementos da VMTools-RA

Requisitos	**Elementos da VMTools-RA**
RA1.1	Análise de Domínio e Gerenciamento de Ativos de Domínio
RA1.2	Gerenciamento do Modelo de Variabilidade
RA1.3	Regras de Composição
RA1.4	Rastreabilidade
RA1.5	Documentação
RA1.6	Verificação de Consistência
RA1.7	Relatórios
RA1.8	Gerenciamento da Resolução da Variabilidade
RA1.9	Mecanismo de Variabilidade
RA2.1	Gerenciamento de Controle de Evolução
RA2.2	Análise de Impacto
RA2.3	Planejamento e Gerenciamento
RA2.4	Comunicação e Compartilhamento
RA2.5	Guia
RA2.6	Notificação e *Feedback*
RA2.7	Análise de *Trade-off*
RA2.8	Versionamento
RA2.9	Importação/Exportação
RA2.10	Persistência
RA2.11	Depende das tecnologias adotadas pelo *stakeholder* responsável
RA2.12	*Middleware*

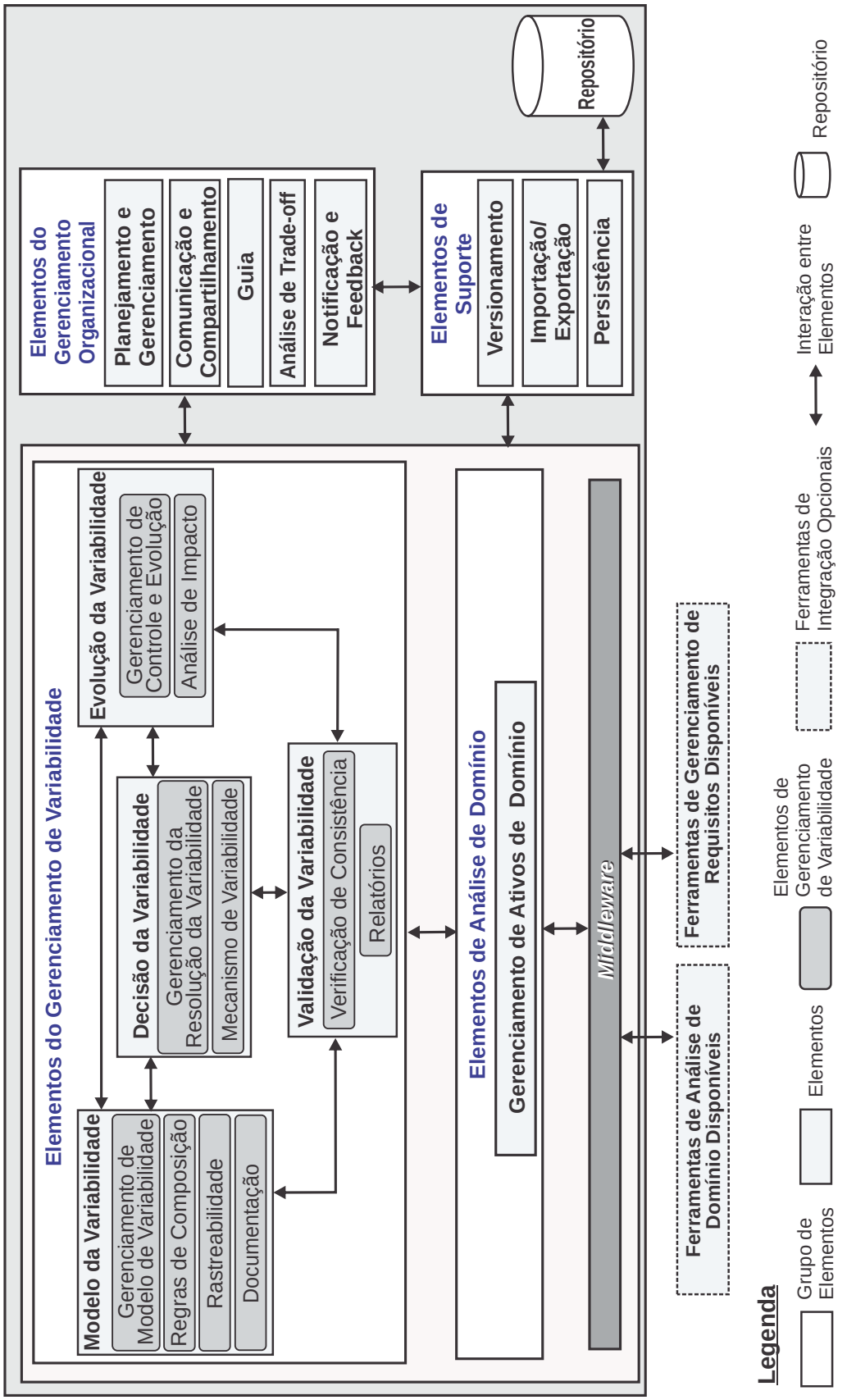


Figura 3.2: Visão geral da VMTools-RA

3.4.2 Escopo

O escopo da VMTools-RA pode ser definido considerando o conjunto de potenciais ferramentas que se pretende construir com base nesta arquitetura de referência. Além disso, o escopo da VMTools-RA abrange a integração apenas com ferramentas de análise de domínio e de especificação de requisitos. Seguem alguns exemplos de ferramentas que podem ser projetadas com base na VMTools-RA:

- **Ferramentas de GV:** identificam, modelam, configuram, realizam as variabilidades e derivam/geram produtos. O GV é realizado em todo o ciclo de vida de desenvolvimento do projeto. Exemplos de ferramentas já desenvolvidas: Gears[1], Pure::variants[2] e CVM[3].
- **Ferramentas de verificação de restrição das variabilidades:** modelam as variabilidades e fornecem mecanismos para realizar configuração interativa com base na seleção de *features* objetivando a verificação de consistência entre os modelos. Exemplos de ferramentas, S.P.L.O.T[4], FAMA[5] e FMP[6].
- **Ferramentas de modelagem das variabilidades:** modelam as variabilidades, geralmente estão relacionadas à análise de domínio. Por exemplo, Metadoc[7].
- **Ferramentas de configuração das variabilidades:** envolve a seleção e realização das variantes. Por exemplo, Visit-FC (Botterweck et al., 2007) e WeCoTin (Asikainen et al., 2004).

As ferramentas elencadas podem ser desenvolvidas com ou sem solucionadores lógicos *Solvers* (ex., BDD[8], SAT[9] e CSP[10]). Tais mecanismos lógicos fornecem raciocínio eficiente e serviços de configuração interativos. Além disso, tais ferramentas devem atender ao contexto de diferentes domínios de aplicação, por exemplo, automotivo, componentes eletrônicos, aeroespacial, comércio eletrônico e telecomunicação.

3.4.3 Objetivos, *Stakeholders*, Interesses e Riscos

O foco da VMTools-RA é fornecer uma estrutura geral para apoiar o desenvolvimento de novas ferramentas de variabilidade de software.

[1](http://www.biglever.com)
[2](http://www.pure-systems.com)
[3](http://www.cvm-framework.org)
[4](http://www.splot-research.org)
[5](http://www.isa.us.es/fama)
[6](http:// gp.uwaterloo.ca/fmp)
[7](http://www. metadoc.de)
[8]BDD - Binary Decision Diagrams
[9]SAT - boolean SATisfiability problem
[10]CSP - Constraint Satisfaction Problem

Objetivos

Dentre os principais objetivos estão:

- apoiar o desenvolvimento de ferramentas de variabilidade de software;
- apoiar e melhorar a capacidade de reutilização sistemática de software;
- apoiar a manutenção e evolução das ferramentas de variabilidade de software com base na VMTools-RA;
- especificar uma estrutura padronizada com base nos processos e modelos de referência fornecidos pela ISO/IEC 26555; e
- fornecer mecanismos de integração com ferramentas de análise de domínio e de especificação de requisitos.

Stakeholders

Os principais *stakeholders* foram identificados após a leitura das normas ISO/IEC 26550 (2013a) e ISO/IEC 26555 (2013b) que apresentam os *stakeholders* relevantes para gerenciar as atividades que envolvem o GV e LPS. São eles:

- **Executivo:** responsável pelos objetivos de negócio da organização e suas restrições;
- **Profissional de Marketing:** responsável pelas exigências do mercado;
- **Gerente Técnico:** responsável pela equipe de trabalho disponível;
- **Arquiteto de Software:** responsável por identificar as necessidades dos sistemas e instanciações da arquitetura de referência;
- **Especialista de Domínio:** responsável pelo fornecimento de informações específicas do domínio;
- **Gerente de Qualidade:** responsável por garantir que os requisitos de qualidade sejam atendidos;
- **Desenvolvedor:** responsável pela implementação da instância arquitetural;
- **Usuário:** utiliza o sistema desenvolvido.

Interesses

Os principais interesses identificados com base nos *stakeholders* são:

I1. Reúso: espera-se que a VMTools-RA possibilite, de maneira sistemática, a reutilização dos elementos do GV, gerenciamento de ativos e redução de custos, além de ROI;

I2. Conformidade entre arquitetura de referência e arquitetura instanciada: espera-se que todas as informações disponíveis neste documento possibilitem manter a conformidade entre a arquitetura de referência e arquitetura instanciada, por meio da instanciação passo a passo;

I3. Evolução: além da instanciação, espera-se que a VMTools-RA facilite a evolução de ferramentas de variabilidade de software, por meio da modificação de estruturas dos sistemas;

I4. Risco Técnico: espera-se que VMTools-RA lide com a identificação de riscos, avaliação, priorização e mitigação para evitar falhas que inibam o ROI e objetivos da empresa;

I5. Análise de Mercado: espera-se que a VMTools-RA lide com análise de mercados, tecnologias disponíveis, ofertas dos concorrentes e outros fatores.

Riscos

A VMtools-RA oferece alguns riscos, limitações e restrições que devem ser considerados ao derivar uma instância arquitetural de ferramenta de variabilidade de software, tais como:

R1. A falta de *stakeholders* qualificados: *stakeholders* devem ser qualificados nas tecnologias atuais e promissoras; eles precisam conhecer a aplicação de domínio, técnicas de projeto modernas, suporte de ferramentas e práticas profissionais;

R2. Escolha incorreta do mecanismo de variabilidade: a VMTools-RA fornece a oportunidade para *stakeholders* escolherem ou implementarem seu próprio mecanismo de variabilidade. Essa tarefa necessita de muita atenção, pois uma escolha errada pode resultar em componentes que não podem ser adaptados quando necessário;

R3. Escolha de ativos de domínio irrelevantes: a VMTools-RA fornece mecanismo para adquirir ativos de domínio; no entanto, a seleção dos ativos corretos depende do conhecimento do especialista de domínio;

R4. Falta de ferramentas de suporte: a VMTools-RA fornece a oportunidade de integração com ferramentas de especificação de requisitos, por exemplo, IBM Rational

DOORS[11], CaliberRM[12], PTC Integrity[13], Microsoft Office; no entanto, essa interoperabilidade pode falhar. Um plano de contingência deve ser desenvolvido/implementado pela organização para mitigar este risco, por exemplo, importação e exportação das informações para planilhas ou bancos de dados e *backups*.

R5. Cultura organizacional: *stakeholders* podem evitar a aceitação da VMTools-RA devido aos valores da organização, comportamentos e regras não estabelecidas. Sem uma articulação clara dos papéis, responsabilidades e procedimentos operacionais do dia a dia, qualquer estrutura organizacional vai deixar de aceitar uma arquitetura de referência;

R6. Falta de treinamento: se o plano de treinamento não é coordenado com a arquitetura de referência, a equipe pode provavelmente tornar-se frustrada e sobrecarregada.

É importante destacar que é inviável identificar todos os objetivos, *stakeholders*, interesses e riscos de todas as instâncias da arquitetura de referência. A Tabela 3.6 apresenta os *stakeholders* identificados neste projeto relacionados aos Riscos, Interesses e Requisitos Arquiteturais. Pode-se inferir que o primeiro grupo de requisitos arquiteturais exige especialistas com conhecimentos específicos tanto no domínio quanto nas tecnologias. O segundo grupo de requisitos arquiteturais abrange todos os *stakeholders* identificados. Quanto aos riscos para implantação da VMTools-RA, esses envolvem decisões organizacionais (executivos) e conhecimento de especialistas de domínio, arquitetos de software, gerentes técnicos e de qualidade. Por fim, os interesses identificados para implantação e instanciação da VMTools-RA abrangem todos os *stakeholders* identificados nesse projeto.

A seguir, serão apresentados os pontos de vistas e visões para esse projeto arquitetural, bem como os termos e informações relevantes para o entendimento da VMTools-RA. Cada visão arquitetural será representada na seguinte ordem:

i) Título e descrição do ponto de vista arquitetura;

ii) Título e descrição da visão arquitetural;

iii) *Stakeholders* interessados;

iv) Interesses capturados pela visão arquitetural; e

v) Representação da visão arquitetural por meio de diagramas UML ou outros meios gráficos.

[11]http://www-03.ibm.com/software/products/en/ratidoorng
[12]http://www.borland.com/en-GB/Products/Requirements-Management/Caliber
[13]http://www.ptc.com/application-lifecycle-management/integrity

Tabela 3.6: Relação entre *stakeholders*, requisitos, riscos e interesses

Stakeholders	**Requisitos Arquiteturais Grupo 1**	**Requisitos Arquiteturais Grupo 2**	**Risco**	**Interesse**
Executivo		RA2.3; RA2.7	R5; R6	I1; I4; I5
Profissional de Marketing		RA2.7		I1; I4; I5
Gerente Técnico		RA2.4	R1; R5; R6	I2
Arquiteto de Software	RA1.2; RA1.3. RA1.4; RA1.8; RA1.9	RA2.1; RA2.2; RA2.4; RA2.5; RA2.6; RA2.9; RA2.10; RA2.11; RA2.12	R1; R2; R4; R6	I1; I2; I3; I4
Especialista de Domínio	RA1.1; RA1.3; RA1.5; RA1.8; RA1.9	RA2.1; RA2.2; RA2.3; RA2.5; RA2.7; RA2.12	R1; R2; R3; R4; R5; R6	I1; I2; I3; I4
Gerente de Qualidade		RA2.2;	R1; R2; R4; R6	I1; I3; I4
Desenvolvedor	RA1.2; RA1.4; RA1.6; RA1.7	RA2.4; RA2.6; RA2.8; RA2.9; RA2.10; RA2.11; RA2.12		I2
Usuário		RA2.6		I3

3.4.4 V.1 - Ponto de Vista Transversal

Esse ponto de vista apresenta informações gerais sobre a arquitetura de referência, tais como termos/conceitos e variabilidades. Além disso, indica que as informações contidas são transversais para outros pontos de vistas, e captura o conhecimento necessário para criar visões particulares com o objetivo de representar sistemas complexos de uma maneira que *stakeholders* possam compreender.

VISÃO CONCEITUAL

Essa visão apresenta um glossário de todos os termos do domínio da VMTools-RA que serão utilizados em outras visões. A Figura 3.3 apresenta as relações diretas entre os conceitos usando um diagrama de classes UML.

Stakeholders: Especialista do Domínio, Arquitetos de Software e Desenvolvedores.

Interesses Capturados: I2.

Representação:

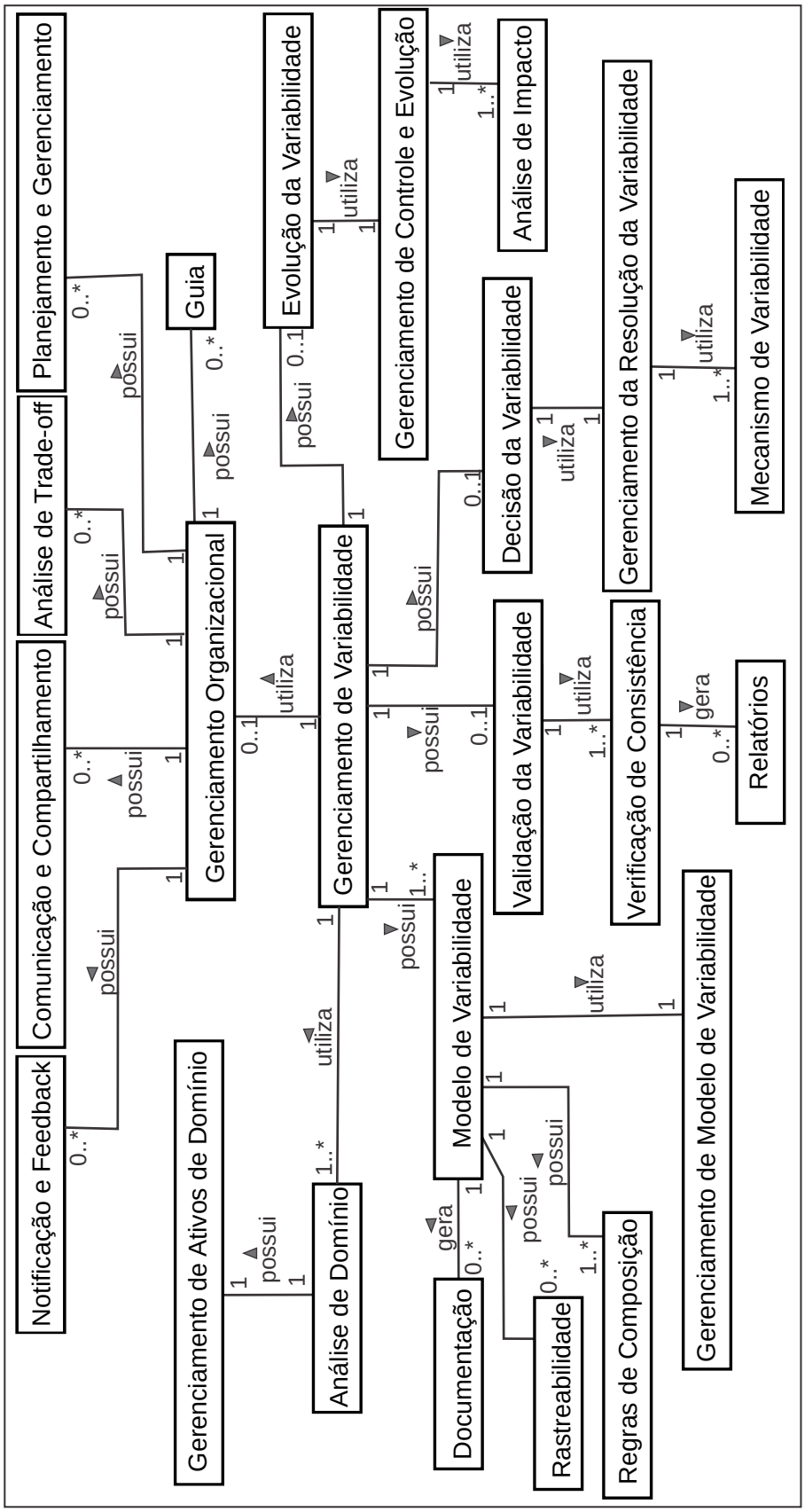


Figura 3.3: Visão conceitual da VMTools-RA

Termos do Domínio:

- Análise de Domínio: identifica as características comuns e as variabilidades de sistemas de um domínio específico;
- Análise de Impacto: avalia as consequências de uma futura alteração no GV;
- Análise de *Trade-off*: avalia os pontos positivos e negativos para a geração de produtos de software com base em aspectos definidos sobre os atributos de qualidade e objetivos do negócio, tais como flexibilidade, custo de manutenção, etc;
- Comunicação e Compartilhamento: promove a criação de ambientes de colaboração (comunicação) para o compartilhamento do GV independentemente da localização geográfica;
- Decisão da Variabilidade: seleciona a melhor opção no ciclo de vida de desenvolvimento para resolver as variabilidades;
- Documentação: apoia os *stakeholders* na documentação de modelos de variabilidade;
- Evolução da Variabilidade: lida com a avaliação do negócio e análise de mercado para evoluir a geração de produtos de software;
- Planejamento e Gerenciamento: estabelece e gerencia capacidades organizacionais e processos da empresa com relação ao GV;
- Gerenciamento da Resolução da Variabilidade: gerencia o momento em que a variabilidade será resolvida;
- Gerenciamento de Ativos de Domínio: gerencia os artefatos reutilizáveis produzidos durante a análise de domínio, tais como *features*, modelos de domínio, especificação de requisitos de domínio, arquitetura de domínio, componentes de domínio, casos de teste de domínio, etc.;
- Gerenciamento de Controle e Evolução: gerencia mudança e evolução nos produtos de software com base na análise de mercado;
- Gerenciamento Organizacional: gerencia os elementos que fornecem suporte à organização durante o GV;
- Gerenciamento de Variabilidade: gerencia todos os elementos responsáveis para modelar variabilidade, verificar consistência, resolver e evoluir a variabilidade durante o GV;

- Gerenciamento do Modelo de Variabilidade: gerencia os pontos de variação, variantes, abordagens para modelar a variabilidade (modelo de *feature*, modelo de decisão e ontologias) e notações para representá-las (FODA, CBFM, OVM, CVL e Ontologias);
- Guia: apoia os *stakeholders* com orientações passo a passo dos processos que envolvem o GV;
- Mecanismo de Variabilidade: trata a resolução das variabilidades com base no tempo de resolução *(binding time)* em diferentes fases do ciclo de vida de desenvolvimento;
- Modelo de Variabilidade: lida com os elementos responsáveis para criação dos modelos de variabilidade;
- Notificação e *Feedback*: promove a melhoria do GV por meio das avaliações dos *feedbacks* de *stakeholders*;
- Rastreabilidade: gerencia links de rastreabilidade de modelos de variabilidade com ativos de domínio;
- Regras de Composição: gerencia as restrições nos ativos de domínio para a criação das *features*. Elas podem ser mutuamente exclusivas, inclusivas e informações de dependência entre ativos;
- Relatórios: fornece informações sobre a verificação de consistência dos modelos de variabilidade;
- Validação da Variabilidade: gerencia os elementos responsáveis para a verificação de consistência entre os modelos de variabilidade;
- Verificação de Consistência: gerencia a consistência dos modelos de variabilidade por meio das regras de dependências e restrições derivadas durante a análise de domínio;

VISÃO DE VARIABILIDADE

A visão de variabilidade apresentada na Figura 3.4 foi criada com base no modelo de *features* da notação CBFM (Czarnecki e Eisenecker, 2000). Tal notação é uma extensão da notação FODA (Kang et al., 1990). O modelo de *feature* é uma estrutura hierárquica de tomada de decisão no desenvolvimento de software. Esse modelo facilita a identificação de componentes reutilizáveis por meio de elementos opcionais, obrigatórios, inclusivas ou exclusivas. É importante destacar que a visão de variabilidade é uma representação da VMTools-RA fazendo uso de modelos de *features* e não representa uma arquitetura de

LPS. Tal visão tem início no elemento `VMTools-RA`, considerado o elemento raiz para esse modelo de árvore hierárquica.

A consistência da visão de variabilidade da VMTools-RA foi avaliada utilizando a ferramenta *online* S.P.L.O.T, que tem objetivo de modelar, avaliar, verificar consistência, depurar, compartilhar e permitir download dos modelos de variabilidade. Conforme pode ser visualizado no Apêndice B, Seção B.1, a visão de variabilidade da VMTools-RA é consistente e possui até 737.280 configurações válidas. Aproveitando a versatilidade do modelo de *feature* e a ferramenta S.P.L.O.T, foi realizado três possíveis configurações de instâncias arquiteturais com base na VMTools-RA, que foram classificadas em `Básica`, `Média` e `Completa`. Tal classificação, resumida na Tabela 3.7, objetiva apresentar possíveis instanciações arquiteturais com base na VMTools-RA, além de servir como um guia para criar e personalizar novas instâncias. Segue abaixo a definição dessa classificação:

1. Básica: instância arquitetural de ferramenta de modelagem das variabilidades, apresentada na Seção B.2 do Apêndice B;

2. Média: instância arquitetural de ferramenta de modelagem e configuração da variabilidade com verificação de consistência, apresentada na Seção B.3 do ApêndiceB;

3. Completa: instância arquitetural de ferramenta de gerenciamento das variabilidade, apresentada na Seção B.4 do Apêndice B.

A definição de quais elementos são obrigatórios e opcionais foi realizada com base na ISO/IEC 26555 e na análise das funcionalidades atendidas pelas ferramentas de variabilidade de software do MS (Apêndice A) confrontadas com os elementos da VMTools-RA, conforme visualizado na Tabela 3.8. É importante destacar que as informações foram extraídas com base na leitura de artigos, relatórios técnicos e documentos disponíveis na web, e que nem todas as informações técnicas e funcionalidades foram encontradas nesses documentos. Essa classificação possibilitou identificar quais ferramentas estão mais completas de acordo com a VMTools-RA, neste caso, são as ferramentas Gears e pure::variants. A execução de uma engenharia reversa confrontando as ferramentas de variabilidade de software existentes com elementos da VMTools-RA forneceria mais evidências sobre a abrangência dessa arquitetura de referência. Tal atividade pode ser realizada como trabalhos futuros.

Vale ressaltar que é inviável identificar todos os elementos obrigatórios das instâncias arquiteturais, pois essa informação também está relacionada ao domínio da aplicação, por exemplo, uma instância arquitetural do tipo `Básica` para o domínio de empresas no contexto de usinas nucleares exigirá relatório de verificação de consistência para os modelos de variabilidade tornando esse elemento obrigatório para aquele contexto. Para tratar essas particularidades do domínio, existe o elemento do gerenciamento organizacional responsável por identificar e determinar as regras, políticas e processos que se enquadram

Tabela 3.7: Instâncias arquiteturais de ferramentas de variabilidade de software

Elementos da VMTools-RA	**Básica**	**Média**	**Completa**
Modelo de Variabilidade	++	++	++
Gerenciamento de Modelo de Variabilidade	++	++	++
Regras de Composição	++	++	++
Rastreabilidade	+	++	++
Documentação		+	++
Validação da Variabilidade	+	++	++
Verificação de Consistência	+	++	++
Relatórios		+	++
Decisão da Variabilidade	+	++	++
Gerenciamento da Resolução da Variabilidade	+	++	++
Mecanismo de Variabilidade	+	++	++
Evolução da Variabilidade			++
Gerenciamento de Controle e Evolução			++
Análise de Impacto			++
Análise de Domínio	+	+	++
Gerenciamento de Ativos de Domínio	+	++	++
Middleware			+
Ferramentas para Adquirir Ativos de Domínio			
Gerenciamento Organizacional			+
Planejamento e Gerenciamento			+
Comunicação e Compartilhamento			+
Guia			+
Análise de *Trade-off*			+
Notificação e *Feedback*			+
Repositório	+	+	+ +
Elementos de Suporte	+	+	+
Versionamento		+	+ +
Importar /Exportar		+	+
Persistência	+	+	+ +

++ Importante; + Recomentado

a cada domínio de aplicação. No modelo de variabilidade, tal elemento é opcional; porém, pode-se tornar imprescindível em determinados domínios.

Tabela 3.8: Relação entre ferramentas e elementos da VMTools-RA

Elementos da VMTools-RA		Ferramentas	CaptainFeature	Clafer	CovamoF-VS	CVL Tool	CVM	DecisionKing	DOPLER	FAMA	FeatureIDE	FMP	FMT	GEARS	GENARCH	Hephaestus	Hydra	Kumbang	LISA tool	Metadoc FM	PLUM	pure variants	s2t2	SOASPL	SPLOT	Variamos	Visit-FC	V-manage	VMWT	WeCoTin	XFEATURE	Total
Modelo de Variabilidade	Gerenciamento de Modelo de Variabilidade	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	29
	Regras de composição	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	29
	Rastreabilidade					x		x	x		x		x	x		x			x	x	x	x		x			x					14
	Documentação						x	x	x		x			x					x	x	x	x						x			x	11
Validação da Variabilidade	Verificação de consistência	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x			x	x	25
	Relatórios			x		x			x	x															x	x						6
Decisão da Variabilidade	Gerenciamento da Resolução da Variabilidade	x			x	x	x		x			x	x	x				x	x			x		x								14
	Mecanismo Variabilidade				x	x			x			x	x	x				x	x			x		x		x						11
Evolução da Variabilidade	Gerenciamento de Controle e Evolução		x	x				x						x	x	x				x				x								8
	Análise de Impacto													x	x																	2
Elementos de Análise de Domínio e Requisitos	Ativos de Domínio							x	x					x	x	x						x										6
	Especificação de Requisitos													x								x										2
	Middleware													x																		2
	Ferramentas de Integração		x											x								x										3
Gerenciamento Organizacional	Planejamento e Gerenciamento			x																												1
	Comunicação e Compartilhamento				x		x							x						x				x		x						5
	Guia		x			x	x			x	x			x				x							x	x						12
	Análise de Trade-off			x																									x			2
	Notificação e Feedback							x	x		x								x													4
Suporte	Versionamento										x				x	x						x		x								5
	Importar /Exportar		x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x		x	x	x				x	x	25
	Persistência								x					x											x		x						4
Total		5	8	9	9	9	8	9	11	6	8	6	7	17	10	8	4	7	9	9	6	14	3	10	7	7	5	5	4	4	5	

***Stakeholders*:** Especialista do Domínio, Arquiteto de Software e Desenvolvedor.
Interesses Capturados: I1, I2, I3 e I4.
Representação:

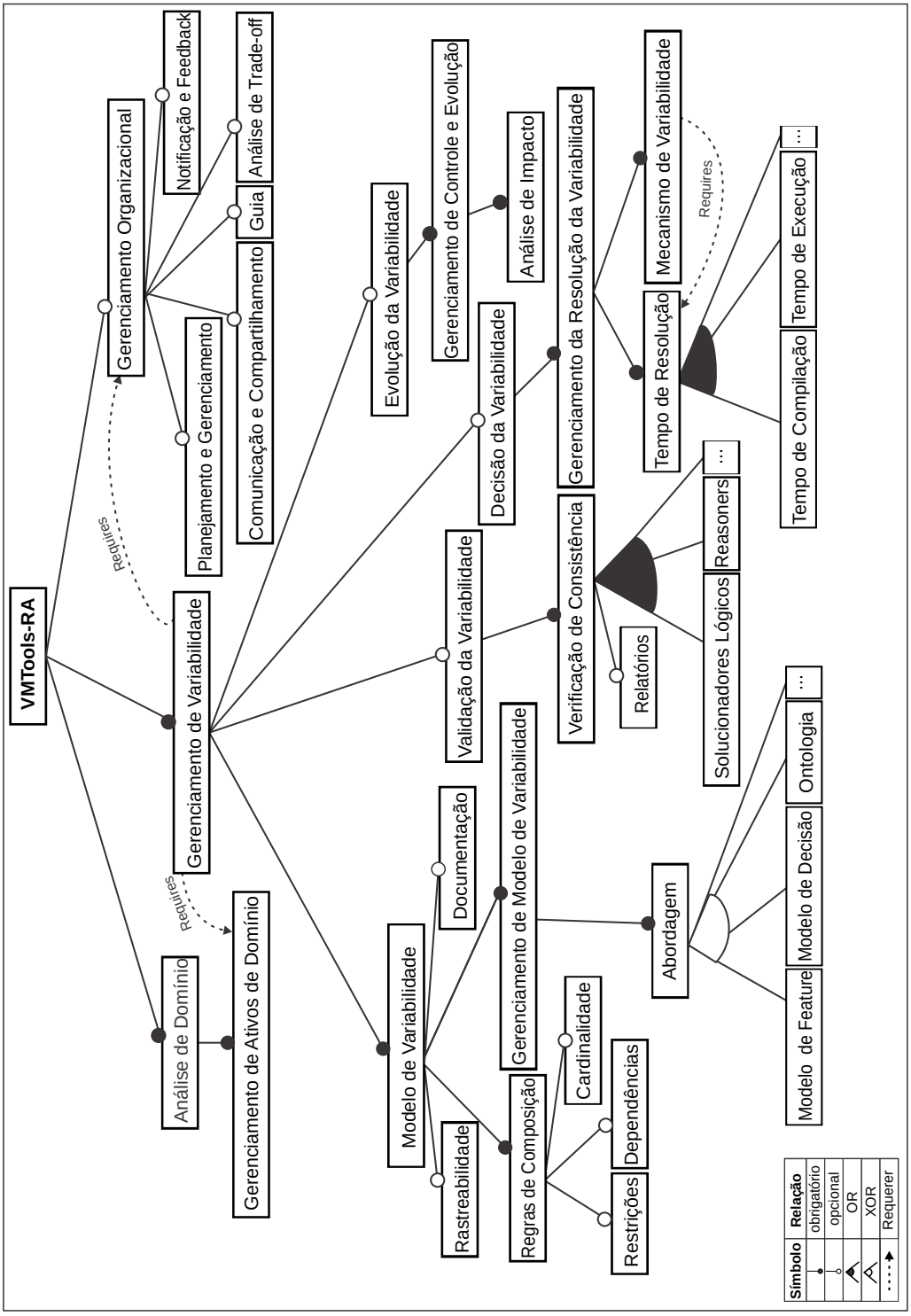


Figura 3.4: Visão de variabilidade da VMTools-RA utilizando modelo de *feature*

VISÃO DE DECISÃO DE PROJETO

A visão de decisão de projeto captura e compartilha o conhecimento das decisões arquiteturais da arquitetura de referência. Essa visão é o resultado do processo de projeto durante a construção inicial ou o desenvolvimento de um sistema de software. Clements et al., (2010) sugerem o uso de *templates* para documentar as decisões arquiteturais, por exemplo, tabelas e gráficos. Neste documento, é representada apenas uma decisão de projeto arquitetural para ilustrar seu funcionamento, outras decisões podem ser projetadas como trabalhos futuros. A decisão de projeto representada neste documento identifica duas questões relacionadas com a integração na VMTools-RA. Essas questões foram documentadas na Tabela 3.9 e na Tabela 3.10, e representadas na Figura 3.5.

Tabela 3.9: Documentação de decisão arquitetural - Integração (Q1)

Grupo	**Integração**
Questão	**Q1.** Se as ferramentas existentes sofrerem atualização e não for possível a integração?
Decisão	**D1.** Instanciar e projetar o módulo da ferramenta pertinente na AR permitindo importação dos dados existentes.
Suposições	**S1.** Supõe-se que as ferramentas disponíveis possam ser integradas e que manterão sua estrutura básica.
Alternativas	**A.1.1** Verificar possibilidades de integração com outras ferramentas que possuam funcionalidades similares. **A.1.2.** Desenvolver estrutura própria e importar informações para a aplicação.
Argumentos	**R.1.1.** A integração com ferramenta similar é possível, mas requer adaptações. **R.1.2.** O desenvolvimento da estrutura própria é possível, pois a AR permite a instanciação de módulos correspondentes às ferramentas de variabilidade de software em domínio e requisitos.
Implicações	**I.1.1**. Requer a negociação com o proprietário da aplicação externa para realizar modificações nas interfaces para permitir a integração. **I.1.2.** Requer um treinamento adicional aos *stakeholders*;

Tabela 3.10: Documentação de decisão arquitetural - Integração (Q2)

Grupo	**Integração**
Questão	**Q2.** É possível integrar novas ferramentas para gerenciar outros contextos das variabilidades?
Decisão	**D2.** Possibilitar que o *middleware* aceite tanto tecnologias sob protocolos de internet como de intranet.
Suposições	**S2.** Supõe-se que módulos definidos da AR já atendam o ciclo completo de gerenciamento de variabilidades.
Alternativas	**A.2.1.** Possibilitar a importação e exportação em todos os módulos da arquitetura de referência. **A.2.2.** Possibilitar a instanciação de um módulo genérico que atenda a diferentes contextos do GV.
Argumentos	**R.2.1.** Adaptações serão necessárias para que o novo módulo atenda as necessidades do cliente. **R.2.2.** A rastreabilidade e consistência das informações deverão ser mantidas com a integração de novas ferramentas.
Implicações	**I.1.1**. Requer a negociação com o proprietário da aplicação externa para realizar modificações nas interfaces para permitir a integração. **I.1.2.** Requer um treinamento adicional aos *stakeholders*;

***Stakeholders*:** Executivo, Profissional de Marketing, Arquiteto de Software, Especialista de Domínio e Gerente de Qualidade.

Interesses Capturados: I4.

Representação:

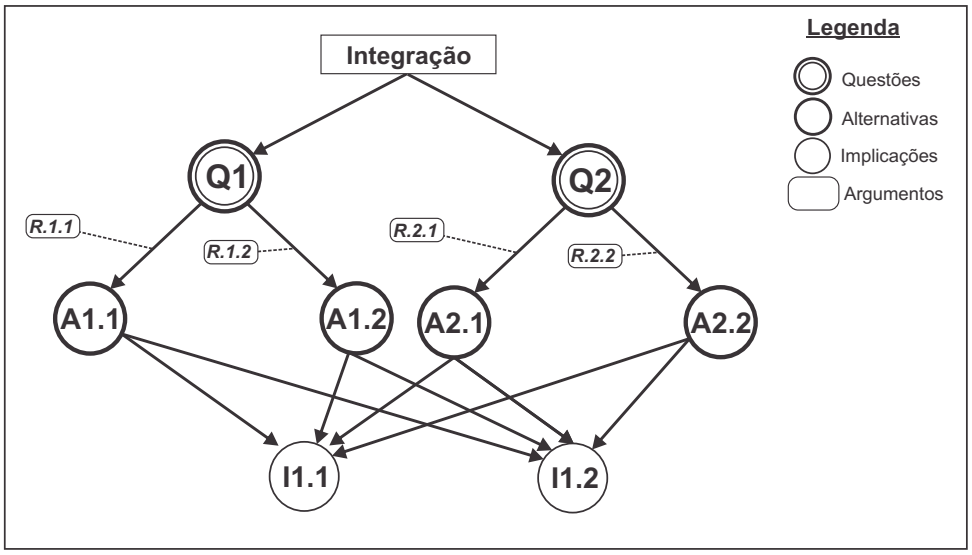


Figura 3.5: Visão de decisão arquitetural da VMTools-RA

HISTÓRICO DE MUDANÇA

A Tabela 3.11 apresenta o histórico de mudança para o ponto de vista transversal.

Tabela 3.11: Histórico de mudança para o ponto de vista transversal

Id do Ponto de Vista	Data	Versão	Histórico de Mudança
V.1	Out. 2015	1.0	Criação
V.1	Dez. 2015	1.1	Atualização com base em avaliação

3.4.5 V.2 - Ponto de Vista de Tempo de Execução

O ponto de vista de tempo de execução apresenta o comportamento dinâmico de sistemas (durante a sua execução) que serão construídos com base na arquitetura de referência. Ele descreve abstrações de elementos de software concretas, tais como atividades e processos. Também tem um impacto significativo sobre as propriedades de qualidade do sistema, por exemplo, capacidade para alteração, proteção e desempenho em tempo de execução.

VISÃO DE PROCESSOS

A visão de processos representada como um diagrama de atividades UML permite analisar atividades de GV, e as entradas e saídas de cada etapa do processo. Essa visão apresenta as principais linhas de realização de um sistema desenvolvido a partir de instanciação da VMTools-RA. A Figura 3.6 apresenta seis compartimentos de realização, quatro deles estão relacionadas com GV (`Modelo de Variabilidade`, `Validação de Variabilidade`, `Decisão da Variabilidade` e `Evolução da Variabilidade`), um compartimento relacionado a `Análise de Domínio` e um `Gerenciamento Organizacional`. Esse último é considerado transversal para os demais elementos.

Os processos de `Análise de Domínio` e `Gerenciamento Organizacional` são responsáveis pela aquisição de ativos de domínio, definição dos mecanismos de integração e gerenciamento dos processos organizacionais da empresa. São caracterizados por ciclos que podem garantir ganho de conhecimento e experiência sobre o domínio e ativos coletados, por isso não foram representados com um simbolo de terminação do processo. O processo de `Modelo de Variabilidade` pode utilizar informações do domínio para modelar as variabilidades com relação à rastreabilidade, regras de composição e informação das variabilidades. O processo de modelo de variabilidade é concluído quando um modelo de variabilidade é criado. O processo de `Decisão da Variabilidade` tem o objetivo de decidir o momento da resolução das variabilidades e variantes e, consecutivamente, gerar os produtos de software. Esse processo está diretamente ligado ao processo `Gerenciamento Organizacional` responsável pelas políticas e decisões de projeto. O processo de decisão da variabilidade é concluído quando produtos de software são gerados. O processo

de `Evolução da Variabilidade` pode gerenciar *feedbacks* dos *stakeholders*, administrar versões de modelos, e informações das variabilidades, a fim de garantir a manutenção do sistema ao passar por mudanças quando necessário. São caracterizados por ciclos que podem garantir evolução e qualidade do GV justificando a ausência do símbolo de término do processo.

Stakeholders: Todos.
Interesses Capturados: I1, I2, I3, I4 e I5.
Representação:

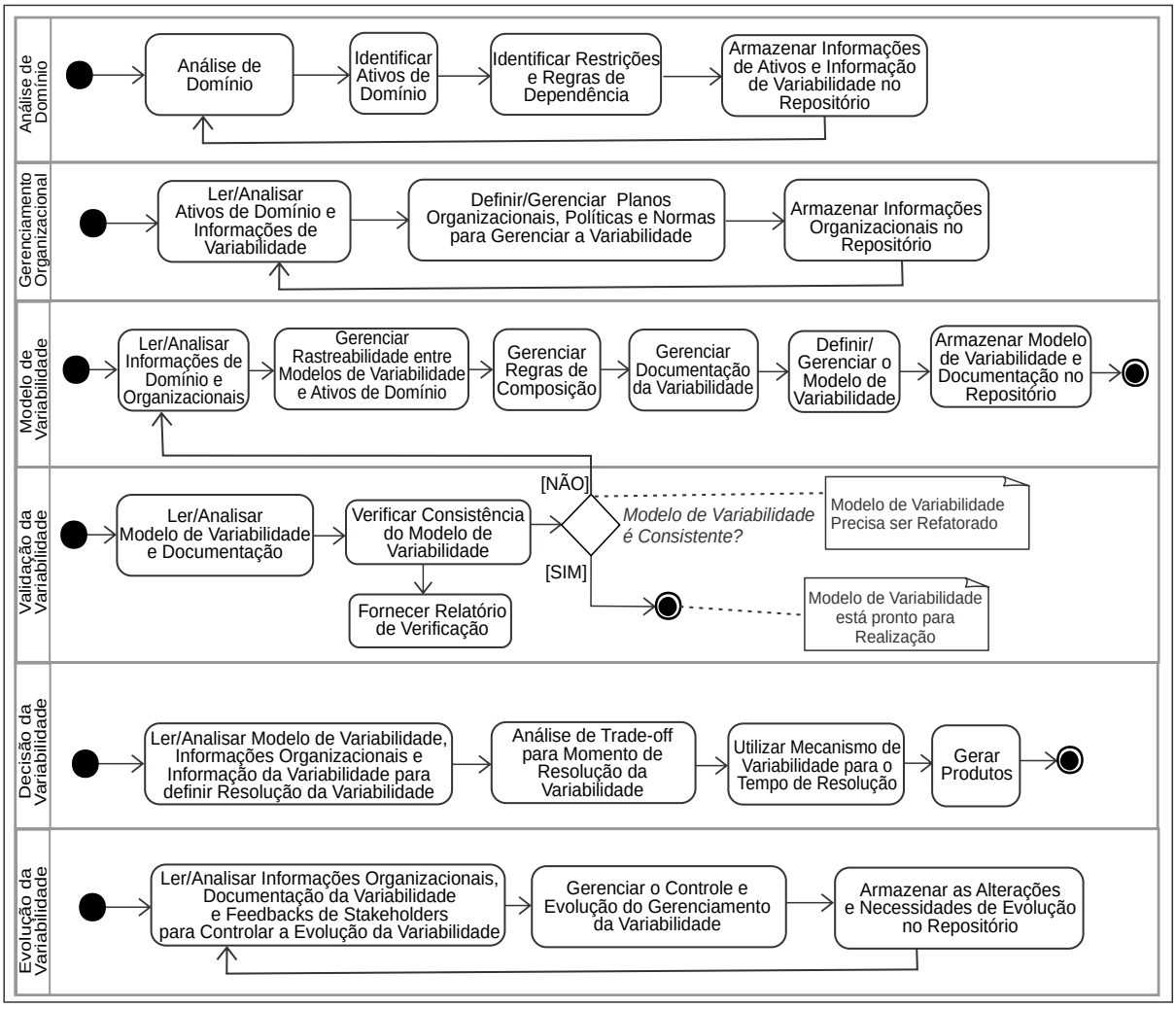


Figura 3.6: Visão de processos da VMTools-RA

HISTÓRICO DE MUDANÇA

A Tabela 3.12 apresenta o histórico de mudança para o ponto de vista de tempo de execução.

Tabela 3.12: Histórico de mudança para o ponto de vista de tempo de execução

Id do Ponto de Vista	Data	Versão	Histórico de Mudança
V.2	Out. 2015	1.0	Criação
V.2	Dez. 2015	1.1	Atualização com base em avaliação

3.4.6 V.3 - Ponto de Vista de Código Fonte

O ponto de vista de código fonte mostra detalhes específicos, tais como estruturas e os módulos de software, sobre a implementação dos sistemas resultantes da arquitetura de referência. Descrevem as relações, dependências e interações entre o sistema e seu ambiente (por exemplo, pessoas, sistemas e entidades externas com as quais interage). Para esse ponto de vista, é apresentado uma Visão em Módulo da VMTools-RA direcionada especialmente para Arquitetos de Software e Desenvolvedores.

VISÃO DE MÓDULO

A visão de módulo da VMTools-RA, representada como um diagrama de classes UML, mostra os módulos que contêm funcionalidades específicas, e podem ser usados para descrever módulos funcionais, fluxo de dados entre eles e interfaces. Além disso, essa visão é representada por pacotes, subpacotes, classes, interfaces e relacionamentos. A visão de módulo representada na Figura 3.7 apresenta quatro módulos principais (pacotes) que compõem a VMTools-RA, sendo eles: (i) `Elementos do Gerenciamento de Variabilidade`, (ii) `Elementos do Gerenciamento Organizacional`, (iii) `Elementos de Suporte` e (iv) `Elementos de Análise de Domínio`.

O pacote de `Elementos de Suporte` representa as funcionalidades espalhadas na aplicação e podem ser encapsuladas como elementos transversais. Dentre as tecnologias envolvidas no desenvolvimento desse pacote, pode-se utilizar JPA e AspectJ, por exemplo. Com relação à persistência, esse tem o objetivo de armazenar informações. Devido as diferentes tecnologias de armazenamento de informação para os modelos de variabilidade, esse elemento deve tratar funcionalidades, tais como, tratamentos de erros, serialização de informação e funções de *Rollback*.

O pacote `Elementos do Gerenciamento Organizacional`, contempla um grupo de aplicações direcionadas aos processos organizacionais. Por exemplo, o subpacote `Planejamento e Gerenciamento` pode fazer uso de processos de qualidade, tais como CMMI[14] com a missão de manter a qualidade do GV e produtos a serem gerados pela ferramenta.

[14]http://cmmiinstitute.com/

Os outros subpacotes auxiliam esse processo de qualidade com a construção de um ambiente de comunicação e compartilhamento de informações entre *stakeholders* fazendo uso de tecnologias como *Cloud* e comunicadores instantâneos. Além disso, podem-se utilizar tecnologias web para criação de guias de ajuda, e ambiente para armazenar as notificações e *feedbacks* dos *stakeholders* sobre o andamento do projeto. O subpacote `Análise de Trade-off` deve prover métricas de qualidade para auxiliar arquitetos de software e analistas na seleção do melhor momento para realizar as variabilidades. Além disso, esse subpacote deve prover suporte a métodos, por exemplo, *Architecture Tradeoff Analysis Method* (ATAM) para avaliação de atributos de qualidade da arquitetura (ex.: desempenho, segurança, modificabilidade e confiabilidade.).

O pacote `Elementos de Análise de Domínio` possui um subpacote `Gerenciamento de Ativos de Domínio` responsável pela identificação dos ativos. Tal subpacote pode fazer uso de tecnologias de armazenamento de ativos e integração com outras ferramentas por meio de um `Middleware`. A Tabela 3.13 apresenta um guia com as principais atividades, tecnologias, *stakeholders* responsáveis e atributos de qualidade exigidos para a criação da base de ativos. A Tabela 3.14 apresenta um guia com as principais atividades, tecnologias, *stakeholders* responsáveis e atributos de qualidade exigidos para realizar a integração com ferramentas que gerenciam ativos de domínio.

Tabela 3.13: Guia para criação da base de ativos

1 - BASE DE ATIVOS	
***Stakeholder*:**	Especialista do Domínio e Arquiteto de Software
Atributo de Qualidade:	Disponibilidade, Confidencialidade, Segurança e Integridade.
Principais Atividades:	a) Identificar ativos do domínio (ex.:*features*, modelos, requisitos, elementos arquiteturais, componentes, casos de testes e descrição de processos);
	b) Criar/gerenciar diferentes repositórios para diferentes ativos (ex.: repositório de *features*, modelos, elementos arquiteturais e componentes);
	c) A informação deve estar disponível apenas aos *stakeholders* envolvidos no projeto, independente de região geográfica (ex.: repositórios *online*, *cloud*);
	d) A base de ativos deve conter políticas de segurança (ex.: controle de acesso, criptografia, segurança contra falhas, *backups* e *recovery*.) para manter a integridade e disponibilidade da informação;
Importante:	A identificação e criação da base de ativos é uma tarefa complexa, e o foco desse projeto não é o gerenciamento e criação de ativos. Para mais informações sobre ativos ver ISO/IEC 26555.

Tabela 3.14: Guia para integração

2 - INTEGRAÇÃO	
***Stakeholder*:**	Especialista do Domínio e Arquiteto de Software
Atributos de Qualidade:	Disponibilidade, Interoperabilidade e Segurança.
Principais Atividades:	a) Integração com ferramentas de especificação de requisitos (ex.: IBM Rational DOORS, CaliberRM, PTC Integrity, Microsoft Office);
	b) Definição de contratos e políticas de integração com ferramentas de requisitos para garantir a disponibilidade das informações;
	c) Definição de portas intranet/internet e tecnologias para integração (ex.: Rest, Soap e Webservices) e tecnologias de segurança de dados (ex.: criptografia e autenticação de usuários);
Exemplos:	Exemplos de integração com ferramentas de especificação de requisitos podem ser encontrados nas ferramentas pure::variants, Gears, e no estudo realizado por Papendieck e Schulze (2014);

O pacote `Elementos do Gerenciamento de Variabilidade` é responsável por gerenciar as variabilidades e contém quatro subpacotes. O subpacote de `Modelo da Variabilidade` é responsável pela modelagem das variabilidades e está ligado ao subpacote `Validação da Variabilidade`, que fornece mecanismos para verificação de consistência dos modelos através de solucionadores lógicos ou verificadores aritméticos. A Tabela 3.15 apresenta as tecnologias que podem ser utilizadas nessa etapa, bem como *stakeholders* e atributos de qualidade identificados.

Tabela 3.15: Guia para modelo de variabilidade e verificação de consistência

3 - MODELO DE VARIABILIDADE E VERIFICAÇÃO DE CONSISTÊNCIA	
Stakeholder:	Especialista do Domínio
Atributos de Qualidade:	Usabilidade, Rastreabilidade, Consistência.
Principais Atividades:	a) Identificar elementos de variabilidade (ex.: pontos de variação e variantes, tempo de resolução para todos os pontos de variação, dependências e restrições). Tais informações devem corresponder às políticas e regras do domínio/negócio (ex.: automotivo, comércio eletrônico, aeroespacial, telecomunicação, etc.);
	b) Definir a abordagem (ex.: modelo de *feature*, modelo de decisão e ontologias) para modelar variabilidades;
	c) Definir representação dos modelos utilizando interfaces gráficas do usuário (ex.: diagramas UML, textos e gráficos 2D/3D) ou ferramentas de modelagem (Ex.: EMF, Simulink e IBM Rational SW Architect);
	d) Gerenciar informações de links de rastreabilidade entre modelos de variabilidade e ativo de domínio (ex.: *links* de navegação e notação);
	e) Armazenar modelos de variabilidade em repositórios e disponibilizar aos *stakeholders* interessados independente da região geográfica (ex.: repositório *online* e *cloud*);
	f) Gerenciar informações de tipos de restrições de dependências (ex.: *requires* e *excludes*), cardinalidades (ex.: min...max) e dependências das variabilidades (ex.: opcional, obrigatório, "OR"alternativa, "XOR" ou exclusivo) entre os modelos de variabilidade;
	g) Desenvolver ou utilizar mecanismos de verificação de consistência com solucionadores lógicos ou verificadores aritméticos (ex.: Lógica proposicional com SAT *solvers*; *Binary Decision Diagram* BDD; *constraint programming* CSP e Lógica Descritiva DL) para validar informações relevantes entre os modelos (ex.: número de *dead features*, número de *features* obrigatórias, opcionais e configurações válidas);
	h) Disponibilizar modelos de variabilidade em (ex.:XML, HTML e JPG) para intercambio entre diferentes aplicações e *stakeholders*;
Exemplos:	Ferramentas de variabilidade com verificadores de consistência: S.P.L.O.T, FAMA, FMP, Hydra, S2T2 e Variamos;

O subpacote `Decisão da Variabilidade` implementa mecanismos para realizar as variabilidades nas diferentes fases do ciclo de vida de desenvolvimento. As principais atividades, tecnologias, *stakeholders* e atributos de qualidade estão apresentadas na Tabela 3.16.

Tabela 3.16: Guia para realização das variabilidades

4 - DECISÃO DA VARIABILIDADE	
***Stakeholder*:**	Executivo, Profissional de Marketing, Especialista do Domínio, Gerente de Qualidade e Arquiteto de Software
Atributos de Qualidade:	Desempenho, Rastreabilidade, Consistência.
Principais Atividades:	a) Estabelecer, junto com a organização, as políticas (ex.:processo, documentação, custos, *trade-off*) para execução do *binding time* (ex.: tempo de compilação, tempo de construção e tempo de execução) em diferentes momentos das fases do ciclo de vida de desenvolvimento (ex.: requisitos, projeto arquitetural, implementação e teste);
	b) Estabelecer um ambiente de integração com mecanismos de análise de *trade-off* para avaliar as alternativas de *binding time* que melhor se adéquam ao negócio/ domínio da aplicação;
	c) Compartilhar informações de *binding time* com *stakeholders* interessados;
	d) Definir mecanismos para implementar as variabilidades. Tal mecanismo depende do ciclo de vida de desenvolvimento (Fase de Requisitos: modelos e diagramas de atividades; Fase de Projeto Arquitetural: diagramas de composição e diagramas de implantação; Fase de Implementação: estereótipos em modelo de entidade, abordagens de *model-driven* e polimorfismo; Fase de Teste: macros, ♯*ifdef*, diretivas e parâmetros);
	e) Armazenar informações sobre *binding time*, mecanismos de variabilidades e análises de *trade-off*;
Exemplos:	Uso do *binding time*: Tempo de compilação (ex.: com diretivas pré-processadas); Tempo de execução (ex.: dependendo da condição implementada no código, funcionalidades são ativadas ou desativadas); Tempo de Atualização (ex.:utilitários de atualização adicionam funcionalidades);
Importante:	A escolha do *binding time* é independente do modelo de variabilidade. Ele é uma consequência de tomada de decisão realizada desde a fase de requisitos até o tempo de execução. As exigências por flexibilidade e de ferramentas de suporte permitem o adiamento do *binding time* tornando dinâmico;

A `Evolução da Variabilidade` gerencia mudanças de modelos de variabilidade. As principais atividades, tecnologias, stakeholders e atributos de qualidade estão apresentados na Tabela 3.17.

Tabela 3.17: Guia para evolução das variabilidades

5 - EVOLUÇÃO DA VARIABILIDADE	
Stakeholder:	Executivo, Profissional de Marketing, Especialista do Domínio, Gerente de Qualidade e Arquiteto de Software
Atributos de Qualidade:	Escalabilidade, Viabilidade, Rastreabilidade, Consistência.
Principais Atividades:	a) Identificar e analisar solicitações (ex.:*feedback*/notificação/*trade-off*) sobre impacto de alterações de informações e de modelos de variabilidade;
	b) Realizar alterações das informações de variabilidade (ex.: remover/ adicionar variantes, pontos de variação, *binding time*, mecanismo de variabilidade, modelos de variabilidade, rastreabilidade) em conformidade com as políticas e regras de negócio;
	c) Armazenar versões de variabilidades para possibilitar restauração (ex.: *Rollback*) em caso de conflitos;
	d) Disponibilizar feedbacks/notificações e análises de trade-off sobre a evolução das informações e modelo de variabilidade para *stakeholders* interessados (ex.: sistema *online* de *feeckback* e comunicação entre *stakeholder*);
Importante:	A evolução das variabilidades está relacionada à decisão organizacional e à análise de mercado para o produto de software a ser gerado;

***Stakeholders*:** Todos.
Interesses Capturados: I1, I2, I3, I4 e I5.
Representação:

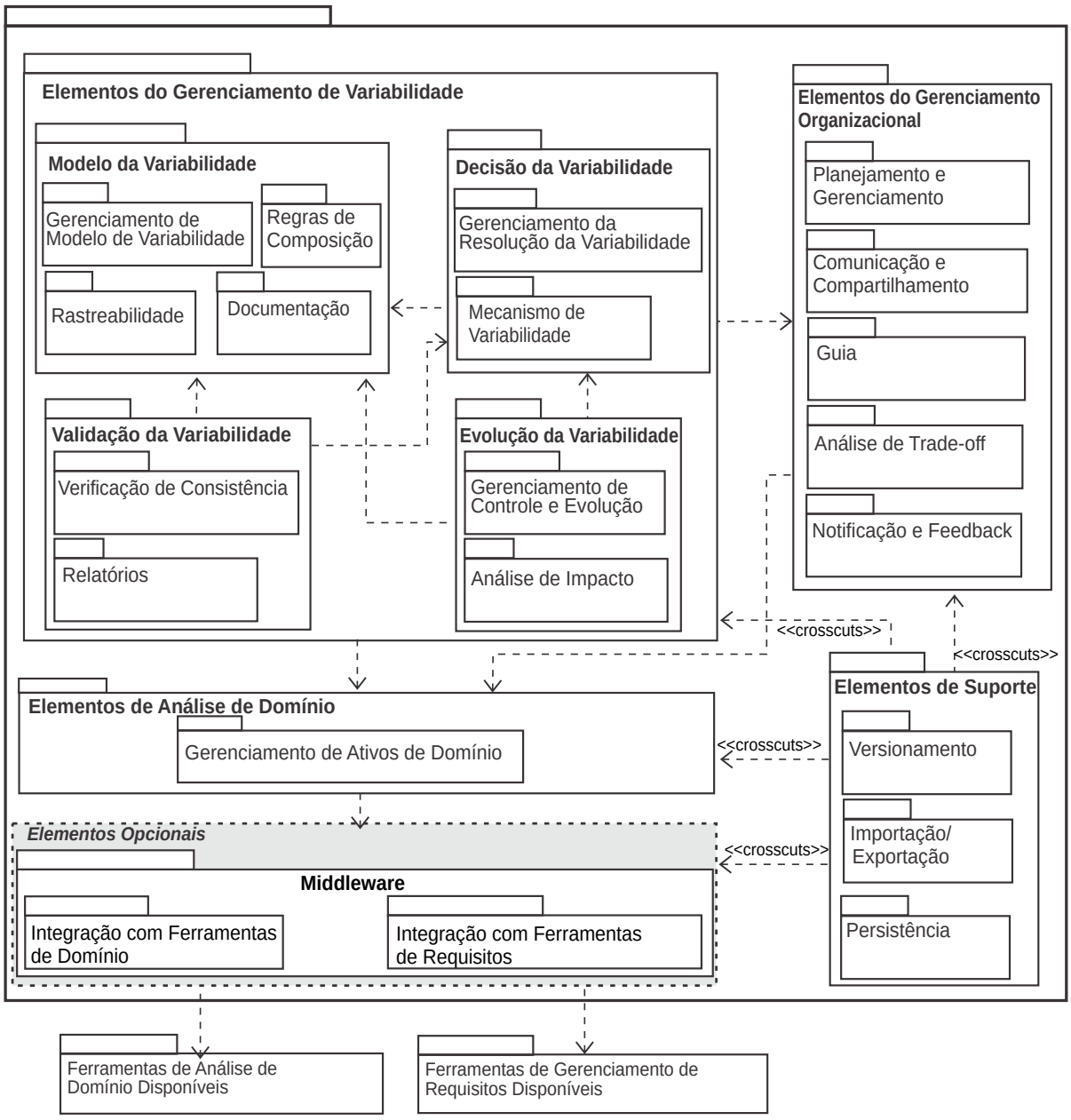


Figura 3.7: Visão em módulo da VMTools-RA

HISTÓRICO DE MUDANÇA

A Tabela 3.18 apresenta o histórico de mudança para o ponto de vista código fonte.

Tabela 3.18: Histórico de mudança para o ponto de vista código fonte

Id do Ponto de Vista	Data	Versão	Histórico de Mudança
V.3	Out. 2015	1.0	Criação
V.3	Dez. 2015	1.1	Atualização com base em avaliação

3.4.7 V.4 - Ponto de Vista de Implantação

O ponto de vista de implantação está preocupado com os elementos arquiteturais que apoiam a distribuição do sistema e inclui elementos, tais como localização, nós *(Nodes)*, dispositivos e as conexões entre eles.

VISÃO DE IMPLANTAÇÃO

A visão de implantação apresenta os componentes de hardware, servidores de aplicações, servidores de banco de dados e máquinas de clientes. Essa visão descreve o sistema de software ou subsistema que será instalado nesses hardwares. Para representar essa visão foram utilizados diagramas de implantação da UML. A visão de implantação representada na Figura 3.8 apresenta seis elementos que podem ser conectados utilizando interfaces em protocolos de internet / intranet.

O `Servidor de Aplicação` pode fornecer um ambiente integrado para a implantação e execução da lógica dos elementos de GV, elementos organizacionais, elementos de domínio e requisitos, e elementos de suporte. Também é possível incluir nesta infraestrutura segurança integrada e tecnologias de integração para gerenciar elementos organizacionais como *feedbacks*, análise de *trade-off*, comunicação e colaboração com *stakeholders*, independentemente da localização geográfica. O `Servidor de Aplicação` pode ser responsável pelo gerenciamento da persistência no `Servidor de Banco de Dados` dos ativos de domínio, informações das variabilidades e informações gerais.

O `Middleware` é um elemento opcional que pode ser implementado para facilitar o intercâmbio de dados entre duas ou mais aplicações dentro do mesmo ambiente, ou por meio de diferentes ambientes de hardware e de rede. Tal elemento pode se integrar com ferramentas existentes de análise de domínio e de requisitos; eles devem seguir as normas organizacionais e políticas que permitam a disponibilidade dos sistemas. O `Middleware` pode utilizar interfaces em protocolos de internet / intranet, e algumas APIs, tais como tecnologias REST, SOAP, RMI e Web Services.

***Stakeholders*:** Desenvolvedor e Arquiteto de Software.

Interesses Capturados: I1 e I2.

Representação:

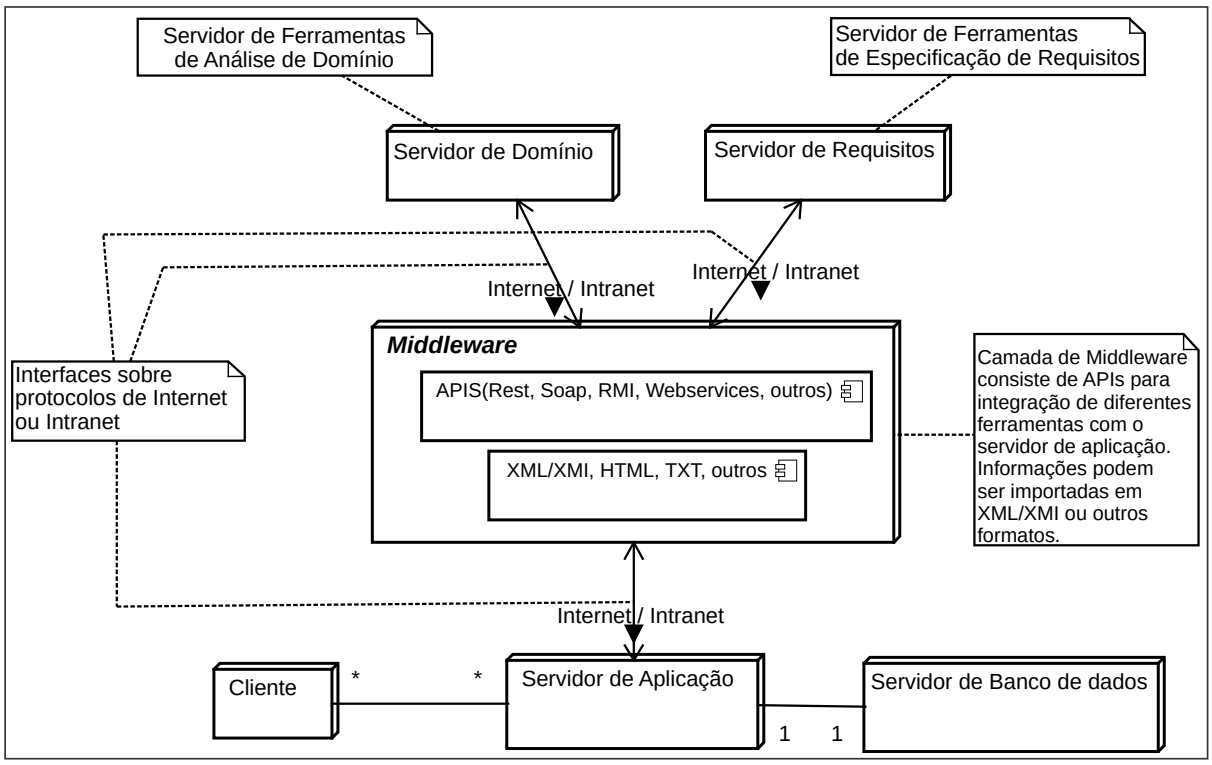


Figura 3.8: Visão de implantação da VMTools-RA

HISTÓRICO DE MUDANÇA

A Tabela 3.19 apresenta o histórico de mudança para o ponto de vista de implantação.

Tabela 3.19: Histórico de mudança para o ponto de vista implantação

Id do Ponto de Vista	Data	Versão	Histórico de Mudança
V.4	Out. 2015	1.0	Criação
V.4	Dez. 2015	1.1	Atualização com base em avaliação

3.5 Etapa 4 - Avaliação da Arquitetura de Referência

O propósito da quarta etapa do processo ProSA-RA é avaliar a arquitetura de referência proposta. Essa avaliação tem o objetivo de detectar defeitos em documentos relacionados à VMTools-RA, e analisar características de qualidade (manutenibilidade, desempenho, segurança, usabilidade, portabilidade e reúso), além da própria descrição da arquitetura.

Para este trabalho, além de avaliações informais realizadas por meio de entrevistas com especialistas em ferramentas de variabilidade de software, foi realizada uma inspeção por *checklist* e um exemplo de aplicação, a fim de identificar defeitos relacionados à omissão, ambiguidade, inconsistência e possíveis informações incorretas. No Capítulo 4

são apresentados os detalhes da avaliação e estudo qualitativo realizado com especialistas em arquiteturas de referência e GV, no Capítulo 5 são apresentados os detalhes do exemplo de aplicação sobre a instanciação de uma arquitetura de software para uma ferramenta de variabilidade com solucionadores lógicos.

3.6 Considerações Finais

Neste capítulo foi apresentada a especificação da VMTools-RA, uma arquitetura de referência que tem dentre os principais objetivos orientar o desenvolvimento de ferramentas de variabilidade de software, apoiar a evolução e manutenção de ferramentas de variabilidade de software, especificar uma estrutura padronizada com base na ISO/IEC 26555, melhorar a capacidade de reutilização e fornecer mecanismos de integração com ferramentas de análise de domínio e de especificação de requisitos. Devido à importância do GV, vários tipos de ferramentas têm sido desenvolvidas para apoiar empresas de diferentes domínios na customização de novos produtos de software. A especificação da VMTools-RA pretende contribuir para o desenvolvimento de ferramentas de variabilidade de software aprimorando a capacidade de reúso e contribuindo como um guia na criação de tais aplicações.

Neste trabalho de mestrado, a VMTools-RA ainda é um projeto no contexto acadêmico; e para torná-la conhecida e largamente utilizada pela academia e indústria é necessário promover e apoiar sua evolução e maturidade por meio de novas avaliações e estudos de caso. O *feedback* de empresas envolvendo estudos de caso com a utilização da VMTools-RA podem fornecer mais evidências de sua real aplicabilidade e, dessa forma, fornecer um consenso envolvendo *stakeholders* interessados em utilizar e influenciar na evolução da VMTools-RA. Orientações e tutoriais juntamente com materiais didáticos adequados sobre o seu funcionamento devem ser disponibilizados *online*. Além disso, a realização de uma engenharia reversa, avaliando as ferramentas de variabilidade de software existentes, pode contribuir para que empresas adotem a VMTools-RA como um guia para projetar suas próprias ferramentas de variabilidade inspiradas em ferramentas já existentes. O capítulo a seguir concentra-se na primeira avaliação da VMTools-RA por especialistas tanto da academia como da indústria. Novas avaliações e uma reengenharia podem ser realizadas como trabalhos futuros.

4

Estudo Empírico Qualitativo da VMTools-RA

4.1 Considerações Iniciais

Este capítulo apresenta a avaliação da VMTools-RA por especialistas e um estudo qualitativo baseado em procedimentos de *Grounded Theory* com o propósito de identificar as principais melhorias para a refatoração da VMTools-RA. A execução deste estudo foi realizada por meio da aplicação de um *checklist* aos especialistas em arquiteturas de referência e GV. Os comentários/*feedbacks* dos participantes e as respostas não atendidas ou atendidas parcialmente também foram analisadas com o objetivo de refatorar/refinar a proposta da arquitetura de referência deste projeto.

Este capítulo está organizado da seguinte forma: A Seção 4.2 apresenta o planejamento da avaliação da VMTools-RA. A Seção 4.3 apresenta a execução da avaliação da VMTools-RA. Na Seção 4.4 são descritos os principais conceitos sobre o estudo qualitativo e *Grounded Theory*. A Seção 4.5 apresenta análise e interpretação dos resultados apresentados no estudo qualitativo. A Seção 4.6 apresenta as possibilidades de refatoração e melhorias para a VMTools-RA. A Seção 4.7 apresenta as ameaças à validade do estudo qualitativo. A Seção 4.8 apresenta as considerações finais deste capítulo.

4.2 Planejamento da Avaliação

Para a avaliação da VMTools-RA, foi utilizado o *checklist* proposto por Santos et al., (2013) denominado *Framework for Evaluation of Reference Architectures* (FERA). A escolha por esse *checklist* foi realizada com base nas recomendações da etapa de avaliação do ProSA-RA. O FERA possui um total de 114 questões (incluindo subquestões) divididas em três grupos:

- **Grupo I:** verificar a completude das informações relacionadas com a construção e conteúdo das visões arquiteturais da arquitetura de referência (58 questões);
- **Grupo II:** verificar adequação da documentação da arquitetura de referência para liberação e uso público (52 questões); e
- **Grupo III:** verificar possibilidades de evolução e viabilidade de instanciação arquitetural com base na arquitetura de referência (quatro questões).

Além disso, FERA é distribuídos entre 9 *stakeholders*, sendo eles: analista, arquiteto, projetista, desenvolvedor, especialista de domínio, integrador, gerente de garantia de qualidade, gerente de software e testador. O *checklist* pode ser visualizado no Anexo F e é composto por questões objetivas de múltipla escolha, cujas respostas oscilam de totalmente satisfatório para não satisfatório com campos para adicionar comentários. As alternativas de respostas para cada pergunta são: *Yes; No; Partially; I am not expert to answer this question.* A última alternativa foi incluída a cada pergunta do questionário para evitar viés dos participantes; e, também, para evitar que o *checklist* fosse dividido de acordo com a qualificação de cada participante, pois seria praticamente impossível prever o conhecimento de cada especialista e as questões destinadas a ele.

Foram convidados quatro especialistas para avaliar a VMTools-RA. Um questionário de caracterização foi aplicado para avaliar o conhecimento dos especialistas, conforme visualizado no Apêndice C e apresentadas nas Tabela 4.1 e Tabela 4.2.

Tabela 4.1: Especialistas em arquitetura de referência

Questões	**Especialista 1 (E1)**	**Especialista 2 (E2)**
Nível Educacional	Doutorando(a)	Doutorando(a)
Profissão	Arquiteto de Software	Pesquisador
Experiência	5 anos	5 anos
Uso de arquitetura de referência	Avançado	Avançado
Projeto de arquitetura de referência	Avançado	Avançado
Avaliação de arquitetura de referência	Avançado	Intermediário
Arquitetura de referência Projetadas	2	1
Domínio da arquitetura de referência Projetada	*Telemedicine* e *Ambient Assisted Living*	*Multi-Agent*

4.3 Execução da Avaliação

A documentação da primeira versão da VMTools-RA (Apêndice E) foi enviada por e-mail aos especialistas juntamente com o questionário de caracterização (Apêndice C) e o *checklist* para avaliação (Anexo F). Apenas três especialistas (E1, E2 e E3) aceitaram responder ao *checklist*. As respostas desses especialistas podem ser vistas na íntegra no

Tabela 4.2: Especialistas em ferramentas de variabilidade de software

Questões	**Especialista 3 (E3)**	**Especialista 4 (E4)**
Nível Educacional	Doutor	Doutor
Profissão	Engenheiro de SW Senior	Arquiteto de Soluções *Cloud*
Experiência	Projetos de Engenheira de SW: 8 anos Engenharia de Requisitos: 5 anos; Desenvolvimento e teste: 8 anos.	Arquiteto/Engenheiro de SW:10 anos; Pesquisador: 4 anos; Arquiteto de Soluções *Cloud*: 1 ano.
Uso de Ferramentas de GV	Avançado	Avançado
Projeto de Ferramentas de GV	Avançado	Avançado
Experiência em GV	Avançado	Avançado
Ferramentas de Variabilidade de Software Projetadas ou Avaliadas	Desenvolveu/Testou: Extensão da FeatureIDE; Utilizou/Avaliou/Testou: pure::variants e ToolDay.	Desenvolveu (SPLOT); Pesquisou: FAMA tool suite, FMP, pure::variants, Gears, FeatureIDE, XFeature.

Apêndice D. O especialista (E4) contribuiu na forma de consultoria sobre o funcionamento de ferramentas de variabilidade de software e avaliação geral das visões arquiteturais da VMTools-RA.

Após a obtenção das respostas do *checklist* realizado pelos especialistas (E1, E2 e E3), foi possível iniciar a avaliação dos resultados. O primeiro grupo de questões do *checklist*, preocupado com a completude da arquitetura de referência, apresentou os seguintes resultados: 47,13% das questões obtiveram resposta positiva *(Yes)* representando que a completude da VMTools-RA está totalmente satisfatória, 13,79% das questões obtiveram a resposta *(Partially)* representando que a completude da VMTools-RA está parcialmente satisfatória, 20,11% das questões obtiveram resposta *(No)* representando que a completude da VMTools-RA não está satisfatória e 18,97% das questões obtiveram resposta *(I am not expert to answer this question)* indicando que os especialistas não se sentiram seguros para respondê-las. Esses resultados revelaram que, apesar de aproximadamente 60% da documentação da VMTools-RA estar totalmente ou parcialmente satisfatória, percebeu-se a falta de informações sobre tomada de decisão, atributos de qualidade e tabelas relacionando *stakeholders* com requisitos arquiteturais e com visões arquiteturais.

O segundo grupo de questões do *checklist*, preocupado com a adequação da documentação da VMTools-RA para liberação ao público, apresentou os seguintes resultados: 12,18% das questões obtiveram resposta positiva *(Yes)* representando que a adequação da documentação está totalmente satisfatória para liberação, 12,18% das questões obtiveram resposta *(Partially)* representando que a adequação da documentação está parcialmente satisfatória, 26,28% das questões obtiveram resposta *(No)* representando que a adequação da documentação da VMTools-RA não está satisfatória, 31,41% das questões obtiveram resposta *(I am not expert to answer this question)* e 17,95% das questões não foram respondidas (em branco). Com esses resultados observou-se que aproximadamente 40% das questões não foram respondidas pelos especialistas em arquitetura de referência, pois

esse segundo grupo de questões abrangem, em sua maioria, questões sobre o domínio de GV que foram respondidas pelo especialista (E3). Após avaliar as respostas, contatou-se que importantes alterações são necessárias antes da liberação da VMTools-RA, tais como, um guia de instanciação arquitetural, descrição mais detalhada dos pontos que são fixos e variáveis da arquitetura e de como os pontos variáveis afetam os pontos fixos.

O terceiro grupo de questões do *checklist* foi relacionado à evolução e viabilidade de instanciação arquitetural da VMTools-RA. As respostas dos especialistas indicaram que a VMTools-RA certamente pode ser instanciada, mas ainda falta informações importantes, tais como, um guia de instanciação passo-a-passo ou exemplos de instanciação arquitetural. Os resultados resumidos do *checklist* podem ser visualizados na Tabela 4.3 e os detalhes dos comentários dos especialistas (E1, E2 e E3) podem ser visualizados no Apêndice D.

Tabela 4.3: Resultados do *checklist* obtidos por especialistas

Respostas	**E1**		**E2**		**E3**		**Média**	
Grupo 1	**Qtd**	**%**	**Qtd**	**%**	**Qtd**	**%**	**Qtd**	**%**
Sim	20	34,48	32	55,17	30	51,72	27,33	47,13
Não	7	12,07	10	17,24	18	31,03	11,67	20,11
Parcialmente	6	10,34	12	20,69	6	10,34	8,00	13,79
Não é Especialista	25	43,10	4	6,90	4	6,90	11,00	18,97
Em branco	0	0,00	0	0,00	0	0,00	0,00	0,00
Total	58	100,00	58	100,00	58	100,00	58,00	100,00
Grupo 2	**Qtd**	**%**	**Qtd**	**%**	**Qtd**	**%**	**Qtd**	**%**
Sim	0	0,00	12	23,08	7	13,46	6,33	12,18
Não	13	25,00	2	3,85	26	50,00	13,67	26,28
Parcialmente	0	0,00	7	13,46	12	23,08	6,33	12,18
Não é Especialista	16	30,77	29	55,77	4	7,69	16,33	31,41
Em branco	23	44,23	2	3,85	3	5,77	9,33	17,95
Total	52	100,00	52	100,00	52	100,00	52,00	100,00
Grupo 3	**Qtd**	**%**	**Qtd**	**%**	**Qtd**	**%**	**Qtd**	**%**
Sim	0	0,00	2	50,00	0	0,00	0,67	16,67
Não	0	0,00	0	0,00	3	75,00	1,00	25,00
Parcialmente	0	0,00	2	50,00	0	0,00	0,67	16,67
Não é Especialista	0	0,00	0	0,00	1	25,00	0,33	8,33
Em branco	4	100,00	0	0,00	0	0,00	1,33	33,33
Total	4	100,00	4	100,00	4	100,00	4,00	100,00

Além dos resultados do *checklist* realizado pelos especialistas (E1, E2 e E3), também foi considerada a entrevista via Skype[1] com o especialista (E4). A entrevista durou 40

[1]https://www.skype.com

minutos e o especialista contribuiu com avaliações da visão arquitetural de variabilidade e sugeriu mecanismos para verificar a consistência das *features* por meio da ferramenta S.P.L.O.T. Também contribuiu com informações sobre ferramentas de variabilidade de software que foram úteis para a obtenção de mais conhecimento sobre o domínio. A seguir, é apresentado um resumo das sugestões do especialista (E4).

- utilização da ferramenta S.P.L.O.T para modelar e avaliar a consistência da visão de variabilidade da VMTools-RA;
- realizar um estudo de caso ou exemplo de aplicação instanciando uma arquitetura de software de ferramentas de variabilidade com base na VMTools-RA;
- apresentar um guia de instanciação arquitetural baseada na VMTools-RA;
- resumir as informações em tabelas;
- correlacionar as ferramentas de variabilidade encontradas com os elementos da VMTools-RA.

Após os resultados e informações adquiridas concluiu-se que há necessidade de refatoração da VMTools-RA. Para isso, todas as questões que obtiveram respostas *(No)* e *(Partially)* foram utilizados no estudo qualitativo, bem como, os comentários redigidos pelos especialistas. A Seção 4.4 apresenta os principais conceitos sobre estudo qualitativo utilizados nesta dissertação.

4.4 Definição do Estudo Qualitativo

Estudos qualitativos têm sido utilizados em diversas áreas da ciência para lidar com a complexidade de questões envolvendo subjetividade e comportamento humano. Recentemente, é possível notar o crescente uso de métodos qualitativos em pesquisas sobre diferentes tópicos em Engenharia de Software (Dybå et al., 2011). Neste trabalho, é utilizado o método proposto por Strauss e Corbin (1998) conhecido por *Grounded Theory*. Esse método oferece um conjunto flexível e indutivo de estratégias para coletar e analisar informações, e é baseado na ideia de codificação (*coding*) para análise de dados.

Em um estudo qualitativo, *coding* representa uma palavra ou pequena sentença que sumariza uma ideia ou informação de dados (Saldana, 2009). Os dados podem consistir de informações retiradas de entrevistas, observações, comentários, informações retiradas de vídeos, web sites, e-mails, imagens, etc. O processo de codificação pode ser dividido em três fases (Strauss e Corbin, 1998): codificação aberta (*Open Coding*), axial (*Axial Coding*) e seletiva (*Selective Coding*), como seguem:

- *Open Coding* é um processo analítico no qual conceitos são identificados e separados em partes discretas para análise, comparação e categorização dos dados. Pode

ser realizada manualmente com a leitura dos dados recuperados agrupando as informações semelhantes em códigos;

- *Axial Coding* realiza a conexão entre os códigos identificados no processo anterior (*Open Coding*) e agrupa-os de acordo com suas propriedades para representar categorias; e
- *Selective Coding* é o processo de refinar a categoria central da teoria com a qual todas as outras estão relacionadas.

Nesta dissertação, foram utilizados os processos *Open Coding* e *Axial Coding*. O *Selective Coding* (Strauss e Corbin, 1998) não foi utilizado, pois essa avaliação não trabalha com uma categoria central, mas sim com diferentes categorias.

4.5 Análise e Interpretação dos Resultados

Os dados qualitativos extraídos do *checklist* de avaliação, que incluem os comentários dos participantes, questões não atendidas e questões atendidas parcialmente, foram analisados manualmente utilizando um subconjunto das fases do processo de codificação sugerido por Strauss e Corbin, (1998) para *Grounded Theory* utilizando os processos *Open Coding* e *Axial Coding*. As informações obtidas por meio do *checklist* e da entrevista informal com especialista (E4) foram sintetizados nos seguintes códigos:

1. Completar com atributos de qualidade e exemplos de aplicação;
2. Completar informações sobre ponto de vista de execução;
3. Completar informações sobre restrições, padrões, políticas;
4. Esclarecer informação de integração da arquitetura de referência no ambiente de TI;
5. Esclarecer no modelo de variabilidade, as partes fixas e opcionais;
6. Esclarecer os recursos que podem afetar os padrões de qualidade;
7. Esclarecer rastreabilidade entre o domínio, requisitos e soluções de tecnologias;
8. Esclarecer visão conceitual;
9. Esclarecer informações de hardware e comunicação;
10. Esclarecer Informações de histórico de mudança e lançamento;
11. Inserir Informação de como a arquitetura de referência pode impactar o projeto;

12. Inserir informação de dependências entre atributos de qualidade e instanciações;
13. Inserir informação de incompatibilidade e dependência relacionadas ao ponto de vista;
14. Inserir Informação sobre critérios de testes;
15. Inserir informação sobre tomada de decisões;
16. Inserir lista de riscos e planos de mitigação para cada visão;
17. Inserir referências e citar materiais de apoio;
18. Inserir um guia para Instanciação;
19. Justificar a falta de outras visões e pontos de vista e tecnologias utilizadas;
20. Manter as visões consistentes com os mesmos elementos e componentes;
21. Relacionar *Stakeholder* com Interesses, visões, requisitos e riscos; e
22. Resumir informações importantes em tabelas.

Os procedimentos de *Open Coding* estimulam a constante criação de novos códigos e a fusão com os códigos existentes quando novos dados de evidência e interpretação emergem. Diante disso, foi possível identificar seis categorias:

- **EI**: Esclarecer Informação;
- **IG**: Inserir Guia;
- **II**: Inserir Informação;
- **MC**: Manter Consistência;
- **RI**: Relacionar Informação; e
- **RU**: Resumir Informação.

Para facilitar a compreensão, os códigos gerados, as categorias identificadas e os especialistas (E1, E2, E3 e E4) foram relacionados gerando a Tabela 4.4. Nessa classificação, é possível notar os pontos mais importantes identificados pelos especialistas para serem considerados na documentação da VMTools-RA. A ordem de leitura dessa classificação tem início no especialista, depois categoria e finalmente código. Por exemplo, E1.II.12 refere se ao Especialista (E1), à categoria "Inserir informação", e o *coding* 12 "Inserir informação de dependência entre atributos de qualidade e instanciação".

Tabela 4.4: Relação de códigos, categorias e especialistas

	E1	**E2**	**E3**	**E4**
II	1; 3; 12; 15; 17	1; 2; 3; 11; 12; 13; 16; 17	11; 12; 13; 14 15; 16; 17	
RI	21	20; 21	21	
MC	20	20		20
IG	18	18	18	18
EI	6; 5; 19	5; 8; 9; 10; 19	4; 5; 6; 7; 8; 9; 10	
RU			22	22

De acordo com as respostas dos especialistas, as solicitações que precisaram de mais atenção na evolução da documentação da VMTools-RA são (II.1), (IG.18) e (RI.21). A Tabela 4.5 apresenta as categorias e os códigos que foram identificados pelos especialistas, bem como a quantidade de solicitações.

4.6 Refatoração e Melhorias para VMTools-RA

A avaliação qualitativa, com base nas questões respondidas pelos especialistas, indicou a possibilidade de mudanças e evolução da VMTools-RA. Portanto, para que haja viabilidade de utilização da arquitetura de referência, importantes pontos foram revistos dentro do projeto arquitetural. Abaixo estão todos os itens refatorados na documentação avaliada pelos especialistas e que correspondem à terceira etapa do processo Prosa-RA (Seção 3.4):

II. 1: A visão de módulo (Seção 3.4.6) foi refatorada com a inserção de tabelas contendo guias para as principais atividades da VMTools-RA, além de informações sobre atributos de qualidade que devem ser considerados na instanciação da VMTools-RA, tais como, disponibilidade e interoperabilidade ao lidar com integração com ferramentas de requisitos, e consistência ao desenvolver modelos de variabilidade; além disso, exemplos de aplicação foram apresentados;

II. 2: A descrição do ponto de vista de tempo de execução (Seção 3.4.5) foi rescrita destacando que esse ponto de vista tem o objetivo de apresentar o comportamento dinâmico de sistemas;

EI. 4: As informações de integração da arquitetura de referência no ambiente de TI foram esclarecidas na visão de módulo (Seção 3.4.6). Há mais detalhes das tecnologias utilizadas neste projeto, tais como, a possibilidade de se utilizar tecnologias HTTP/TCP para realizar integrações por meio de APIs REST, SOAP e Webservices;

Tabela 4.5: Solicitações de melhorias e refatoração da VMTools-RA

Cod	Descrição	Qtd
II. 1	Completar com atributos de qualidade e exemplos de aplicação	19
II. 2	Completar informações sobre ponto de vista de execução	4
II. 3	Completar informações sobre restrições, padrões, políticas	3
EI. 4	Esclarecer informação de integração da AR no ambiente de TI	2
EI. 5	Esclarecer no modelo de Variabilidade as partes fixas e opcionais	7
EI. 6	Esclarecer os recursos que podem afetar os padrões de qualidade	2
EI. 7	Esclarecer rastreabilidade entre o domínio, requisitos e soluções de tecnologias	8
EI. 8	Esclarecer visão conceitual	3
EI. 9	Esclarecer informações de hardware e comunicação	2
EI. 10	Esclarecer Informações de histórico de mudança e lançamento	3
II. 11	Inserir Informação de como a AR pode impactar o projeto	3
II. 12	Inserir informação de dependências entre atributos de qualidade e instanciações	4
II. 13	Inserir informação de incompatibilidades e dependências relacionada ao ponto de vista	4
II. 14	Inserir Informação sobre critérios de testes	2
II. 15	Inserir informação sobre tomada de decisões	3
II. 16	Inserir lista de riscos e planos de mitigação para cada visão	2
II. 17	Inserir referências e citar materiais de apoio	3
IG. 18	Inserir um guia para Instanciação	18
EI. 19	Justificar a falta de outras visões e pontos de vista e tecnologias utilizadas	3
MC. 20	Manter as visões consistentes com os mesmos elementos e componentes	2
RI. 21	Relacionar Stakeholder com Interesses, visões, requisitos e riscos	20
RU. 22	Resumir informação importantes em tabelas	2
	Total	**119**

EI. 5: A visão de variabilidade Figura 3.4 foi alterada para atender aos pontos obrigatórios e opcionais que foram identificados após avaliar as ferramentas de variabilidade de software e os processos que envolvem o GV na ISO/IEC 26555;

EI. 8: Os termos da visão conceitual foram reescritos (Seção 3.4.4);

EI. 9: As informações de Hardware e comunicação foram reescritas na visão de implantação (Seção 3.4.7);

EI. 10: O histórico de mudanças de cada ponto de vista foi acrescentado;

II. 15: Foram acrescentados mais detalhes sobre variabilidades da arquitetura de referência expondo pontos opcionais e obrigatórios o que favorece na tomada de decisão para realização da instância arquitetural;

II. 17: Foram acrescentadas mais referências de materiais de apoio para investigação das tecnologias mencionadas no projeto da VMTools-RA, além de exemplos de ferramentas que implementam tais tecnologias/funcionalidades;

IG. 18: Foi acrescentado um guia para a instanciação na visão de variabilidade (Seção 3.4.4), e exemplos de configuração de instâncias arquiteturais podem ser vistos no Apêndice B. Além disso, um exemplo de aplicação instanciando uma arquitetura de ferramentas de variabilidade de software com base na VMTools-RA é apresentado no Capítulo 5;

EI. 19: A falta de outras visões arquiteturais, tais como, visão arquitetural de serviços de sistema e visão arquitetural de componentes colaborativos, foi justificada na (Seção 3.4);

RI. 21: *Stakeholders*, requisitos, interesses e riscos foram relacionados na Tabela 3.6. Além disso, cada visão apresenta *stakeholders* responsáveis;

RU. 22: A visão de processos (Seção 3.4.5) estava muito extensa e foi resumida.

A Tabela 4.5 apresenta as solicitações de melhoria e refatoração identificadas pelos especialistas somando 22 melhorias, que totalizam 119 solicitações (solicitações significam as questões que cada especialista selecionou *No* e *Partially*). Para este trabalho foram realizadas 13 melhorias/refatoração que correspondem a 89 solicitações dos especialistas. As requisições de melhorias não atendidas somam 33 e são justificadas a seguir:

II.3: Foi citado na documentação da VMTools-RA padronizações ISO/IEC 26555 para criação do projeto arquitetural, porém não foram encontradas políticas para a criação de ferramentas de variabilidade de software;

EI.6: As informações sobre recursos que podem afetar os padrões de qualidade estão relacionadas aos elementos organizacionais e domínio da aplicação em que a ferramenta de variabilidade de software será implementada e, por isso, esses recursos podem variar dificultando sua representação. Uma avaliação sobre as consequências de tais atributos em diferentes domínios pode fornecer evidências de como realizar essa representação na VMTools-RA;

EI.7: A rastreabilidade entre domínio, requisitos e soluções de tecnologias foi representada parcialmente por meio de tabelas e representações gráficas de cada visão. Porém, um estudo de caso propondo uma avaliação da rastreabilidade em diferentes domínios é importante para obtenção de evidências para essa representação;

II.11: O impacto que a VMTools-RA pode gerar em projetos está escrito de forma parcial e superficial, uma análise de impacto em um estudo de caso seria mais adequado para obtermos essa avaliação;

II.12: Idem à resposta EI.6;

II.13: Informações de incompatibilidade e dependência relacionadas aos pontos de vista foram atendidos parcialmente, pois há pontos de vistas com informações transversais para outros pontos de vistas;

II.14: Informações sobre critérios de testes não foram tratados nesse projeto;

II.16: Informações sobre riscos e planos de mitigação para cada visão não foram tratados nesse projeto;

MC.20: As visões arquiteturais foram projetadas para apresentarem os mesmos elementos, porém a visão de variabilidade apresenta mais elementos que outras visões, pois observou-se a necessidade de representar todas as relações (OR, XOR, Opcional e Obrigatório) da visão de variabilidade e, para isso, a árvore hierárquica foi ampliada para apresentar essas alternativas (Figura 3.4).

4.7 Ameaças à Validade do Estudo

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

4.7.1 Ameaças à Validade de Conclusão

A principal ameaça para esse estudo está relacionada ao número de participantes/especialistas (4). Esse número é pequeno, mas o conhecimento dos especialistas que colaboraram com esse estudo foi significativo. Além disso, o efeito de fadiga pode ter influenciado nos resultados, uma vez que o documento para avaliação consistia de textos extensos e o *checklist* para avaliação também era extenso com 114 questões.

4.7.2 Ameaças à Validade de *Constructo*

As principais ameaças de *constructo* são referentes à validade do *checklist* para a avaliação da arquitetura de referência e à validade da documentação e notações utilizadas na VMTools-RA. Optou-se pela utilização de um *checklist* consolidado na área acadêmica para avaliação de arquiteturas de referência conhecido por FERA (Nakagawa et al., 2014;

Santos et al., 2013). Tal *checklist* já foi utilizado na avaliação de outros projetos, tais como, (Duarte, 2012; Feitosa, 2013) amenizando essa ameaça.

Quanto à documentação da VMTools-RA, buscou-se pela utilização da notação UML para representar a maioria das visões arquiteturais. O diagrama de variabilidades, por sua vez, foi representado por um modelo de *feature* utilizando a notação CBFM (Czarnecki e Eisenecker, 2000); porém, descrição de seu funcionamento e legendas foram utilizadas para esclarecer seu funcionamento e assim amenizar essa ameaça.

Com relação ao nível de conhecimento dos especialistas, buscou-se por especialistas com mais de cinco anos de experiência em avaliação e projeto de arquiteturas de referência e de ferramentas de variabilidade de software. Um questionário de caracterização foi aplicado para avaliar o conhecimento dos especialistas. Tal questionário evidenciou que os especialistas possuem conhecimento avançado para realizar a avaliação da VMTools-RA.

4.7.3 Ameaças à Validade Interna

Tem a intenção de verificar se o tratamento realmente causou o resultado obtido. Nesse estudo, as seguintes ameaças foram identificadas:

Diferenças entre os especialistas: Devido à natureza do estudo envolver duas áreas da Engenharia de Software, a avaliação contou com quatro especialistas sendo dois em arquitetura de referência e dois em GV. Para evitar algum tipo de viés para os especialistas que responderam ao *checklist* foi acrescentado a cada pergunta uma alternativa para não especialistas (*"I am not expert to answer this question."*).

Acurácia das respostas dos especialistas: Uma vez que os especialistas em arquitetura de referência possuem experiência em avaliação e projetos de tais arquiteturas; e o especialista em GV possui experiência no desenvolvimento projeto, teste e avaliação de ferramentas de GV, considera-se que a avaliação da documentação do projeto VMTools-RA, por meio do *checklist* é válida.

Efeito de fadiga: O *checklist* do FERA consiste de 114 questões e o documento da primeira versão da VMTools-RA (Apêndice E) avaliado consiste de 12 páginas com diagramas e textos descrevendo os requisitos, visões e etapas do projeto arquitetural. Os efeitos de fadiga são uma das principais ameaças, por isso, para tentar minimizar esse problema, a documentação e o *checklist* foram enviados aos especialistas por e-mail com um prazo de 15 dias para enviarem as respostas.

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

4.7.4 Ameaças à Validade Externa

Participantes: A obtenção de especialistas qualificados foi uma das grandes dificuldades encontradas para esse estudo. Assim, especialistas do meio acadêmico com um nível de conhecimento avançado na indústria (5 ou + anos) em engenharia de software foram convidados, buscando amenizar tal ameaça.

4.8 Considerações Finais

Os resultados obtidos por meio da análise qualitativa com base no *checklist* e comentários/*feedbacks* dos especialistas permitiram avaliar a viabilidade da arquitetura de referência e propor possíveis melhorias e refatoração dos pontos cruciais ainda não atendidos na primeira versão da VMTools-RA. Assim, após a análise do estudo qualitativo, diversas melhorias foram realizadas na segunda versão da VMTools-RA. Porém, detalhes de alguns módulos e elementos ainda necessitam ser futuramente investigados. No próximo capítulo, é apresentado um exemplo de aplicação de uma instância arquitetural de uma ferramenta de variabilidade de software com base na VMTools-RA avaliada. Esse exemplo de aplicação foi sugerido durante a avaliação com especialistas, pois é uma forma de avaliar o funcionamento da VMTools-RA ao instanciar uma arquitetura de software para uma ferramenta de GV, além de servir como um guia para *stakeholders* instanciarem diferentes ferramentas de variabilidade de software baseada na VMTools-RA.

5

Exemplo de Aplicação: Instanciação da VMTools-RA

5.1 Considerações Iniciais

Neste capítulo é apresentado um exemplo de aplicação em que, a partir da VMTools-RA, é especificada uma ferramenta de variabilidade de software com solucionadores lógicos.

Para a instanciação da ferramenta proposta, os requisitos arquiteturais a serem propostos foram elicitados, assim como a identificação dos *stakeholders* e interesses. Portanto, cada visão da VMTools-RA foi instanciada com o objetivo de atender aos requisitos arquiteturais e interesses.

Na Seção 5.2 é apresentada uma descrição da ferramenta a ser especificada. Na Seção 5.3 é apresentada a instanciação arquitetural com base na VMTools-RA. Na Seção 5.4 são apresentadas discussões sobre a instanciação da ferramenta. Por fim, na Seção 5.5 são apresentadas as considerações finais deste capítulo.

5.2 Descrição Geral da Ferramenta

A ferramenta instanciada com base na VMTools-RA, conforme apresentado na Figura 5.1, constitui-se de uma aplicação desenvolvida em JavaEE[1] com tecnologias web, tal como, Ajax e HTML. Optou-se por especificar essa instância arquitetural tendo como base uma arquitetura cliente-servidor em três camadas (`apresentação`, `aplicação` e `persistência`) e pela utilização do padrão arquitetural *Model View Control* (MVC). Esse padrão consiste na separação da lógica da aplicação (*Model*) da visão (*View*) e o (*Controller*) que faz a mediação da entrada de dados convertendo em comando para as

[1] http://www.oracle.com/technetwork/java/javaee/overview/index.html

camadas de modelo e visão. A ferramenta tem o objetivo de realizar a modelagem de variabilidades utilizando modelos de *features* por meio da notação *Cardinality Based Feature Model* (CBFM) (Czarnecki e Eisenecker, 2000) e verificação de consistência fazendo uso de solucionadores lógicos. Além disso, essa ferramenta tem o propósito de gerenciar artefatos variáveis e apoiar atividades que estão relacionadas ao GV.

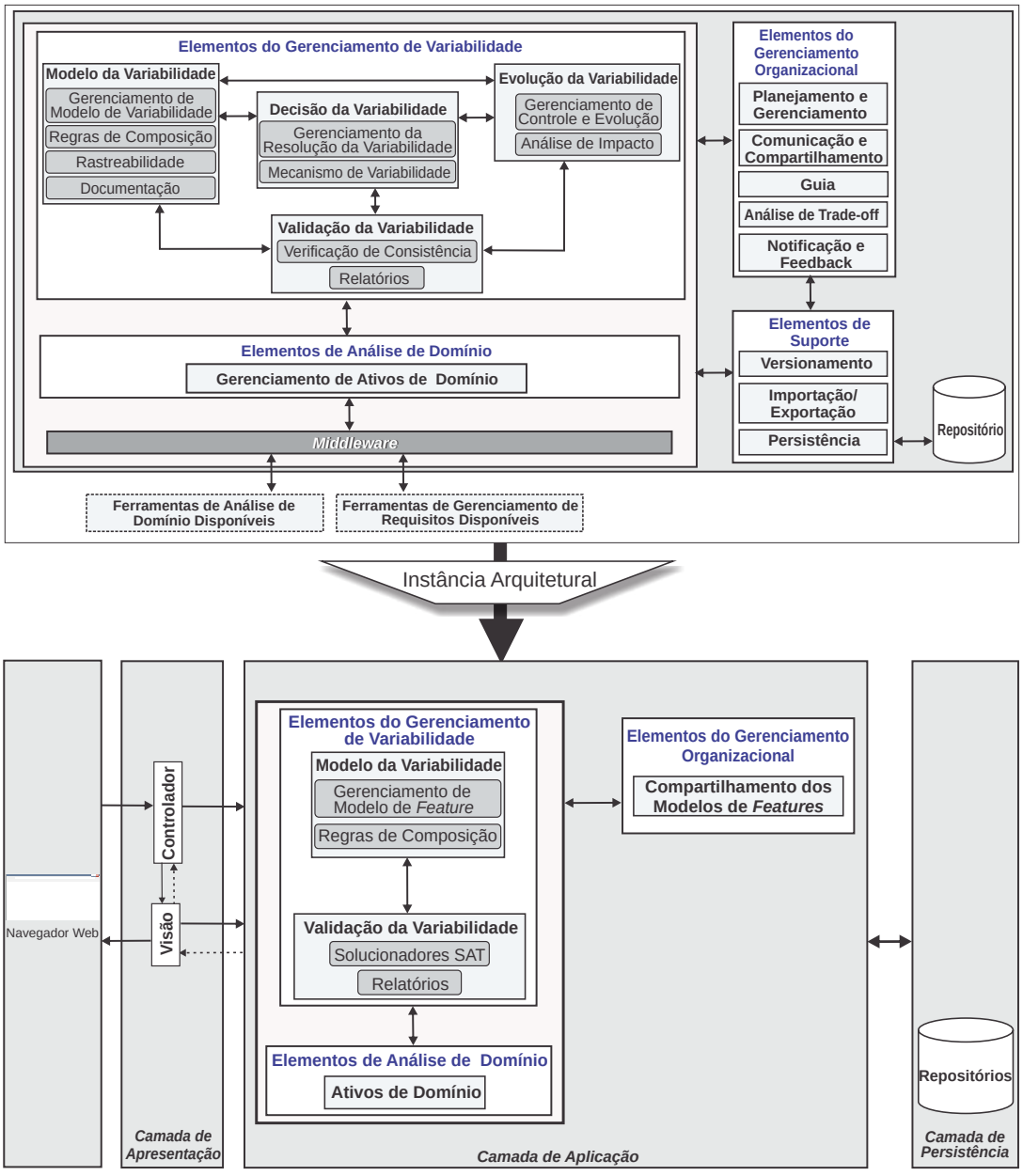


Figura 5.1: Instância da arquitetura de referência

O modelo de *feature* a ser implementado nessa ferramenta tem a importante função de fornecer meios de representação dos artefatos comuns e variáveis facilitando a visualização e gerenciamento de produtos. Para isso, será utilizado o *Eclipse Modelling Framework*

(EMF)[2], que é um *framework* genérico orientado a modelos. O EMF apoia o gerenciamento básico de modelos, tais como, criação, edição e persistências. Além disso, o EMF pode ser integrado com vários outros projetos para validação de modelos, pois apoia a produção de um conjunto de classes Java para os modelos de *features*.

As regras de composição são responsáveis pela validação das combinações de *features*, que garantem a correta semântica entre as combinações realizadas. Nessa instanciação será utilizada uma fórmula proposicional de lógica matemática de problemas de satisfabilidade conhecidos por Solucionadores SAT (*SAT Solvers*) (Cook, 1971), que consiste em decidir se uma proposição pode ser satisfeita. Por exemplo, um valor lógico pode ser atribuído às suas variáveis de uma maneira que torne a fórmula verdadeira. Tais fórmulas podem ser expressas com um conjunto de variáveis booleanas conectadas por operadores lógicos ($\neg$, $\wedge$, $\vee$, $\rightarrow$,$\leftrightarrow$). Além disso, os *SAT Solvers* permitem uma análise relacional de recomendações indicando a melhor situação para que uma determinada *feature* seja ou não selecionada.

Os elementos organizacionais são responsáveis pela definição dos processos responsáveis ao gerenciamento, planejamento e identificação dos ativos de domínio. Consequentemente, uma organização aumentará continuamente os seus ativos de domínio para adequar sua infraestrutura e conduzir a criação de novos produtos. Além disso, os elementos organizacionais são responsáveis pela implementação de meios de compartilhamento dos modelos de *features* entre *stakeholders* interessados. O compartilhamento de modelos de *features* será facilitado por meio de um repositório *online* disponível aos *stakeholders*.

5.3 Instanciação da VMTools-RA

Com o objetivo de documentar a arquitetura sendo instanciada, são apresentados requisitos arquiteturais, *stakeholders* e interesses (Seção 5.3.1), os quais foram obtidos com base na descrição da ferramenta. Utilizando essas informações, cinco visões foram criadas a partir da instanciação da VMTools-RA, sendo a visão conceitual e visão de variabilidade para o ponto de vista transversal (Seção 5.3.2); visão de implantação para o ponto de vista de implantação (Seção 5.3.3); visão de módulo para o ponto de vista de código fonte (Seção 5.3.4); e visão de processos para o ponto de vista de tempo de execução (Seção 5.3.5). Para cada visão, serão apresentados: descrição da instanciação, *stakeholders*, interesses capturados e representação gráfica.

5.3.1 Requisitos Arquiteturais, *Stakeholders* e Interesses

É importante destacar que o sistema a ser construído não provê evolução dos modelos de variabilidade e também não realiza o *binding time*, que é o tempo em que as variabilidades

[2]https://eclipse.org/modeling/emf/

serão resolvidas para gerar um produto. O objetivo dessa ferramenta é a modelagem das variabilidades e os mecanismos de verificação de consistência entre os modelos gerados. Com base na descrição do sistema, os requisitos arquiteturais identificados foram:

- a ferramenta deve viabilizar o gerenciamento de ativos de domínio para serem utilizados no modelo de variabilidade;
- a ferramenta deve viabilizar a construção de modelos de *features* utilizando a notação CBFM;
- a ferramenta deve viabilizar que os modelos de *features* considerem as regras de composição de dependência, restrição e cardinalidade;
- a ferramenta deve viabilizar uma função para verificação automática da consistência utilizando *SAT Solvers*;
- a ferramenta deve viabilizar a geração de relatórios de verificação de consistência;
- a ferramenta deve compartilhar modelos de variabilidade *online*;
- a ferramenta deve persistir modelos de variabilidade em repositório *online*; e
- a ferramenta deve gerenciar gráficos de modelos de *features*, configuração de *features* e relatórios de consistência.

Com base nos requisitos arquiteturais e descrição, foram identificados os *stakeholders* do sistema. Entretanto, vale destacar que é inviável prever todos os *stakeholders* envolvidos. Os *stakeholders* são:

Arquiteto de Software: responsável pela identificação das necessidades da ferramenta e instanciação da arquitetura;

Especialista de Domínio: responsável por fornecer informações específicas de domínio e verificar se os requisitos referentes ao domínio foram atendidos;

Desenvolvedor: responsável por definir as tecnologias, implementação da ferramenta e realizar manutenções; e

Usuário: utiliza a ferramenta desenvolvida.

Assim como os *stakeholders*, o conjunto de interesses pode sofrer modificações. A seguir, estão os interesses identificados para os *stakeholders* apresentados:

I1. Modelar e gerenciar variabilidade: é o principal interesse desejado para essa ferramenta;

I2. Conformidade entre arquiteturas de referência e instanciada: espera-se que a documentação da ferramenta possibilite manter a conformidade entre a arquitetura de referência e a arquitetura instanciada;

I3. Rastreabilidade: espera-se que a organização da ferramenta facilite o rastreio dos problemas arquiteturais e de implementação com base na VMTools-RA;

I4. Requisitos de usabilidade e integração com tecnologias disponíveis: espera-se que a organização da ferramenta facilite a incorporação de tecnologias para gerar modelos gráficos de *features*;

I5. Disponibilidade e segurança: por se tratar de uma aplicação *online*, espera-se que os requisitos de segurança e disponibilidade sejam facilmente identificados para a ferramenta.

5.3.2 Ponto de Vista Transversal

O ponto de vista transversal apresenta informações gerais sobre a arquitetura de software, tais como termos e conceitos. Para este ponto de vista, a visão conceitual e a visão de variabilidade são apresentadas.

VISÃO CONCEITUAL

Essa visão apresenta um glossário dos termos utilizados nos outros pontos de vista e está representada na Figura 5.2.

Termos do Domínio:

Análise do Domínio: identifica as características comuns e as variabilidades de sistemas de um domínio específico;

Compartilhamento dos Modelos de *Features*: promove o compartilhamento dos modelos de *features*;

Gerenciamento de Ativos de Domínio: gerencia os artefatos reutilizáveis produzidos durante a análise de domínio (*features*, modelos de domínio, especificação de requisitos de domínio, arquitetura de domínio, componentes de domínio, casos de teste de domínio, etc.);

Gerenciamento de Modelo de *Feature*: gerencia os pontos de variação, variantes utilizando modelos de *features* e notação CBFM;

Gerenciamento de Variabilidade: gerencia todos os elementos responsáveis para modelar variabilidade e verificar consistência entre os modelos de *features*;

Gerenciamento Organizacional: gerencia os elementos que fornecem suporte à organização durante o GV;

Modelo de Variabilidade: lida com os elementos responsáveis para criação dos modelos de variabilidade;

Regras de Composição: gerencia as restrições nos ativos de domínio para criação das *features*. Elas podem ser mutuamente exclusivas, inclusivas e informações de dependência entre os artefatos de software;

Relatórios: fornece informações sobre a verificação de consistência dos modelos de *features*;

Solucionadores SAT: realizam a análise automática dos modelos de *features* por meio de lógica proposicional;

Validação da Variabilidade: gerencia a consistência dos modelos de *features* por meio de regras de dependência e restrições derivadas durante a análise de domínio;

***Stakeholders*:** Especialista do domínio, Arquiteto de Software e Desenvolvedor.
Interesses Capturados: I2 e I3.
Representação:

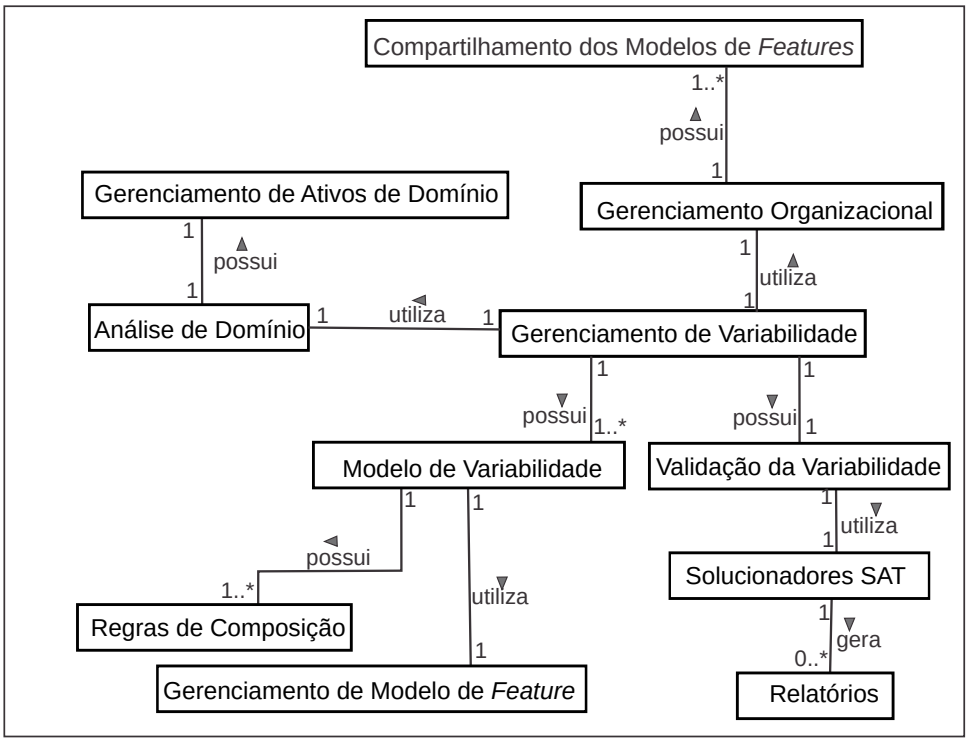


Figura 5.2: Relação entre conceitos da ferramenta de variabilidade de software

VISÃO DE VARIABILIDADE

A visão de variabilidade apresentada na Figura 5.3 foi criada com base no modelo de *features* da notação CBFM (Czarnecki e Eisenecker, 2000). Os elementos destacados

(cinza) representam as *features* selecionas para esta instância arquitetural. A consistência dessa configuração foi validada com a ferramenta S.P.L.O.T e é apresentada no Apêndice B na Seção B.5

Stakeholders: Especialista do Domínio, Arquiteto de Software e Desenvolvedor.

Interesses Capturados: I2 e I3.

Representação:

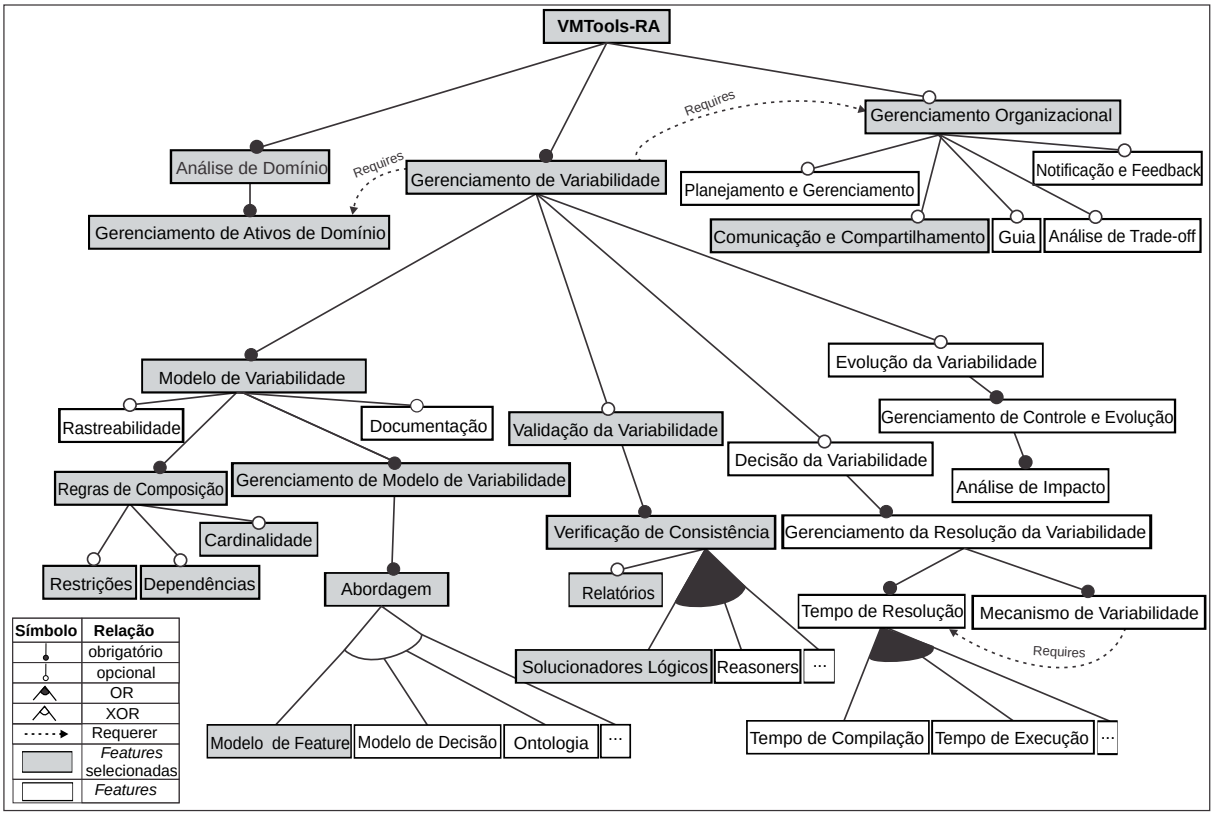


Figura 5.3: Visão de variabilidades da VMTools-RA destacando a seleção das *features* correspondentes à instância arquitetural

5.3.3 Ponto de Vista de Implantação

Objetiva apresentar os componentes de hardware para a implantação da ferramenta de variabilidade de software.

VISÃO DE IMPLANTAÇÃO

A Figura 5.4 apresenta os componentes de hardware e software presentes no ambiente de implantação. O `Servidor de Aplicação` fornece um ambiente integrado para a implantação e execução da lógica dos elementos do gerenciamento do modelo de vari-

abilidade. O `Servidor de Aplicação` é responsável pelo gerenciamento da persistência no `Servidor de Banco de Dados` dos modelos de variabilidade e ativos de domínio.

***Stakeholders*:** Desenvolvedores e Arquitetos de Software.

Interesses Capturados: I2, I4 e I5.

Representação:

Figura 5.4: Visão de implantação da ferramenta de variabilidade de software

5.3.4 Ponto de Vista de Código Fonte

O Ponto de vista de código fonte mostra detalhes específicos, tais como estruturas e os módulos e respectivos submódulos, sobre a implementação do sistema.

VISÃO DE MÓDULO

A visão de módulo representada na Figura 5.5 apresenta os módulos principais para a criação da instância arquitetural de ferramentas de variabilidade com solucionadores lógicos. Pode-se observar três módulos principais, `Elementos do Gerenciamento de Variabilidade`, `Elementos de Análise de Domínio` e `Elementos do Gerenciamento Organizacional`.

Dentro do módulo `Elementos do Gerenciamento Organizacional` encontra-se o pacote `Compartilhamento dos Modelos de` ***`Features`***, que objetiva a construção de um ambiente de comunicação e compartilhamento de informações entre *stakeholders* fazendo uso de tecnologias como *Cloud* e comunicadores instantâneos.

O pacote `Elementos de Análise de Domínio` possui um subpacote `Ativos de Domínio` responsável pela identificação e gerenciamento dos ativos do domínio. O pacote `Elementos do Gerenciamento de Variabilidade` é responsável por gerenciar variabilidades e é composto por dois subpacotes. O subpacote `Modelo de Variabilidade` contém o `Gerenciamento de Modelo de` ***`Feature`***, que é responsável por modelar variabilidades e, nesta instanciação, fará uso de frameworks de modelagem do EMF. E o subpacote `Regras de Composição` que se refere a restrições e dependências entre combinações de *features*, permitindo ao módulo de validação das variabilidades realizar verificações de consistências entre os modelos.

O subpacote `Validação da Variabilidade` está ligado ao modelo de variabilidade, pois fornece mecanismos para verificação de consistência dos modelos de *features*. É composto por dois subpacotes, `Solucionadores SAT` responsável por implementar a

lógica proposicional com *SAT solvers*, e `Relatórios` responsável por emitir relatórios de verificação de consistência entre os modelos.

O pacote de `Persistência` utilizará *Java Persistence API* (JPA), que é um *framework lightweight*, baseado em POJOS (*Plain Old Java Objects*) para persistir objetos Java. O objetivo desse pacote é armazenar informações e tratar funcionalidades específicas de persistência, tais como, tratamentos de erros, serialização de informação e funções de *Rollback*.

***Stakeholders*:** Arquitetos de Software, Desenvolvedores.

Interesses Capturados: I1, I2, I3, I4 e I5.

Representação:

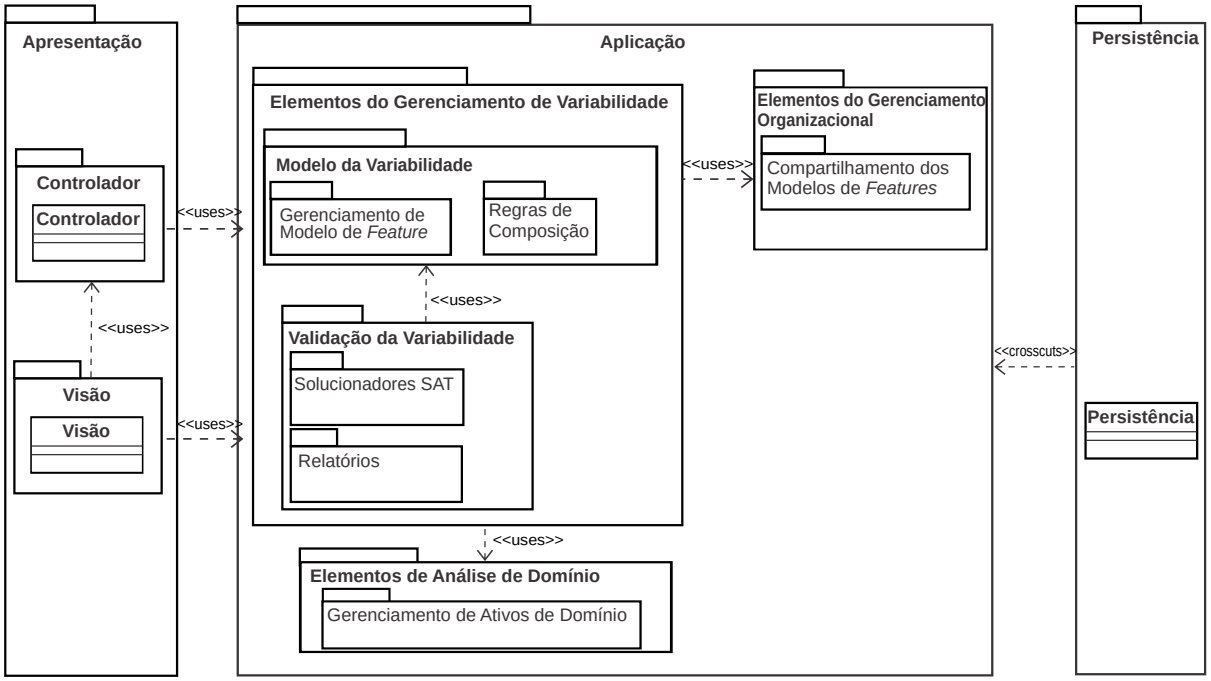


Figura 5.5: Visão em módulo da ferramenta de variabilidade de software

5.3.5 Ponto de Vista de Tempo de Execução

Esse ponto de vista apresenta o comportamento dinâmico de sistemas durante a sua execução e descreve abstrações de elementos de software concretas, tais como atividades e processos.

VISÃO DE PROCESSOS

A visão de processos representado na Figura 5.6 permite analisar atividades de gerenciamento do modelo de variabilidade com solucionadores SAT, e as entradas e saídas de cada etapa do processo.

O processo `Informação do Domínio` é responsável pela aquisição de ativos de domínio (*features*) e informações de variabilidades desses ativos. São caracterizados por ciclos que podem garantir ganho de conhecimento, experiência e evolução das *features* identificadas no domínio, por isso não foram representados com um simbolo de terminação do processo. As atividades do processo `Gerenciamento Organizacional` têm o foco de implementar uma estrutura de compartilhamento dos modelos de *features* criados na ferramenta e também são caracterizados por ciclos que podem garantir ganho de conhecimento e experiência sobre o domínio; por isso, não foram representados com um símbolo de terminação do processo.

O processo de `Modelo de Variabilidade` fará uso das *features* e informações de variabilidades, tais como restrições e regras de dependências identificadas na análise de domínio. O processo de modelo de variabilidade é concluído quando um modelo de *feature* é criado.

O processo de `Validação da Variabilidade` fará uso dos modelos de *features* e regras de composição (informações sobre restrições e dependências) para verificar a consistência dos modelos. Essa verificação fará uso de solucionadores SAT com regras de lógica proposicional permitindo uma análise automática dos modelos. Além disso, fornecerá relatório com os resultados da verificação permitindo uma refatoração do modelo caso esse seja inválido.

Stakeholders: Arquitetos de Software e Desenvolvedores.

Interesses Capturados: I1 e I2.

Representação:

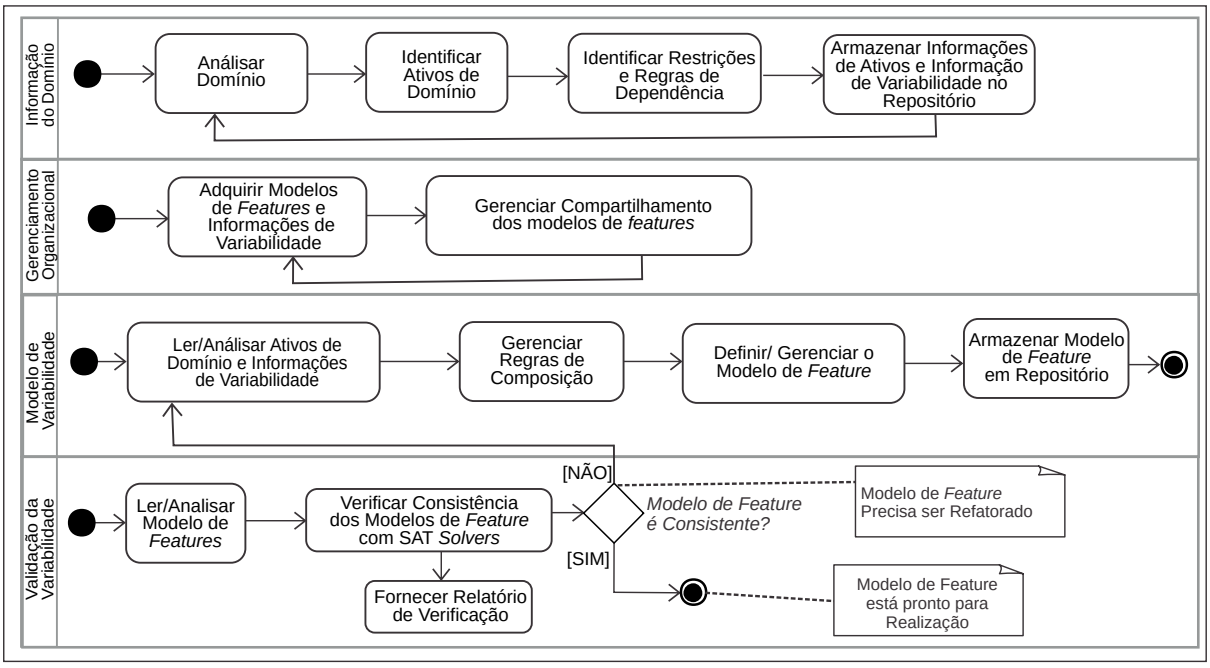


Figura 5.6: Visão de processos da ferramenta de variabilidade de software

5.4 Discussão sobre a Instanciação

Importantes pontos de discussão surgiram durante a instanciação arquitetural da ferramenta de variabilidade de software. O primeiro foi o possível viés do autor na instanciação da ferramenta com base na VMTools-RA. Por isso, para amenizar essa ameaça, buscou-se como base uma ferramenta de variabilidade de software já consolidada na academia conhecida por S.P.L.O.T (Mendonça et al., 2009a). Essa ferramenta faz uso de solucionadores lógicos, relatórios de verificação de consistência, utiliza notação CBFM para modelar as *features* e disponibiliza um repositório online com os modelos de *features* criados pelos usuários. É importante destacar que a arquitetura de software da ferramenta S.P.L.O.T. não estava disponível para ser utilizada nessa instanciação.

Outro fator importante foi o processo de instanciação com base nos artefatos existentes da arquitetura de referência, tais como, descrição, requisitos arquiteturais e representação das visões arquiteturais. Foi possível seguir a documentação da arquitetura de referência para instanciar a ferramenta de variabilidade de software.

5.5 Considerações Finais

Neste capítulo foram discutidos os detalhes relacionados à instanciação da arquitetura de uma ferramenta de variabilidade com solucionadores lógicos. A elaboração dessa instância arquitetural apresentou evidências da aplicabilidade da VMTools-RA, pois foi possível seguir a documentação da VMTools-RA para identificar os componentes necessários para a instanciação arquitetural da ferramenta de variabilidade de software. Além disso, esse exemplo de aplicação complementa o estudo qualitativo apresentado no Capítulo 4, pois serve com um guia para novas instanciações arquiteturais. Foi possível reutilizar as estruturas elaboradas na VMTools-RA, como visões arquiteturais, requisitos e descrições. Dentre as limitações encontradas, está a falta da posterior implementação dessa instância arquitetural. Uma implementação permitiria a descoberta de possíveis irregularidades presentes na arquitetura de referência.

6

Conclusão

O GV tem possibilitado a criação em larga escala de produtos de software singulares por meio do gerenciamento de suas variabilidades, possibilitando mais reúso e ROI à indústria. Consequentemente, diversas empresas têm adotado meios para automatizar o processo de GV com apoio de ferramentas de variabilidade de software. Apesar do grande número de ferramentas disponíveis, ainda há evidências de um número crescente de empresas que criam suas próprias soluções para gerenciar variabilidades. Nessa perspectiva, a complexidade dessas ferramentas tende a aumentar, criando um desafio considerável para o seu desenvolvimento, manutenção e integração.

Entretanto, estudos relacionados às padronizações que englobam conhecimento sobre como projetar, padronizar e desenvolver sistemas têm ganhado destaque na área de engenharia de software. Nessa perspectiva, normas internacionais, tais como, ISO/IEC 26550 e ISO/IEC 26555 fornecem importantes informações sobre o funcionamento dos processos que envolvem o GV em LPS. Tais normas permitem especificar, verificar e validar práticas da engenharia de LPS com base em documentação estabelecida. Dessa forma, esforços no sentido de estabelecer padrões para engenharia de software possuem grande relevância e podem auxiliar no desenvolvimento de sistemas mais facilmente reusáveis. Nesse contexto, arquiteturas de referência, um tipo especial de arquitetura de software, têm sido considerada como um elemento significativo para melhorar a produtividade e a qualidade dos sistemas de software. Dessa forma, uma arquitetura de referência que apoie a criação de ferramentas de variabilidade de software constitui uma importante solução para padronização dessas ferramentas.

6.1 Contribuições

O resultado principal dessa pesquisa foi a especificação de uma arquitetura de referência para ferramentas de variabilidade de software, denominada VMTools-RA, com o objetivo

principal de facilitar o desenvolvimento de novas ferramentas desse domínio. Além disso, essa pesquisa também teve como objetivo apoiar a evolução e manutenção de ferramentas de variabilidade, melhorar a capacidade de reutilização sistemática de software, especificar uma estrutura padronizada com base nos processos e modelos de referência da ISO/IEC 26555 e fornecer mecanismos de integração com ferramentas de análise de domínio e especificação de requisitos. É importante ressaltar que, dentro do referencial teórico levantado, a arquitetura de referência proposta é exclusiva para promover o desenvolvimento de ferramentas de variabilidade de software. Ainda nesse contexto, os componentes arquiteturais especificados dão suporte para construções de ferramentas de modelagem, configuração e GV com ou sem solucionadores lógicos.

Além da contribuição principal, o conjunto de requisitos arquiteturais, *stakeholders*, interesses e riscos identificados durante a condução deste trabalho podem ser considerados como contribuições, pois podem servir de base para o desenvolvimento e integração de novas ferramentas de variabilidade de software ou para o estabelecimento de outras arquiteturas de referência no contexto de GV. Outra contribuição importante deste trabalho foi a realização de um MS, apresentado no Apêndice A, sobre ferramentas de variabilidade de software. Esse MS viabilizou a aquisição de conhecimento técnico sobre as ferramentas de variabilidade de software existentes contribuindo para o estabelecimento da VMTools-RA.

Também pode-se considerar as avaliações qualitativas como contribuição para esse estudo, pois os especialistas avaliados forneceram importantes comentários para a melhoria deste trabalho. O exemplo de aplicação que apresenta a especificação de uma ferramenta de variabilidade de software com solucionadores lógicos é também uma importante contribuição. Tal exemplo de aplicação permite que desenvolvedores possam utilizar a especificação arquitetural para o desenvolvimento de ferramentas de variabilidade de software. Além disso, o referencial teórico deste trabalho complementa as normas internacionais ISO / IEC 26550 (2013a) e ISO / IEC 26555 (2013b), contribuindo com conhecimento sobre o funcionamento do GV.

Em síntese, buscou-se com o desenvolvimento desse projeto contribuir para a comunidade acadêmica e industrial de LPS com uma arquitetura de referência para apoiar o desenvolvimento de ferramentas de variabilidade de software e, além disso, fornecer uma documentação de referência ao contexto de GV.

Publicações dos Resultados e Contribuições:

- OLIVEIRAJR, E.; ALLIAN, A. P. Do Reference Architectures can Contribute to Standardizing Variability Management Tools? In: International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA 2015) @ WICSA 2015. Montreal, Canadá. p. 9-12.

- ALLIAN, A. P.; OLIVEIRAJR, E.; NAKAGAWA, E. Y. Estabelecimento de uma Arquitetura de Referência para Ferramentas de Gerenciamento de Variabilidades.

In: 2nd Latin-American School on Software Engineering (ELA-ES 2015), Porto Alegre: UFRGS, 2015. v. 1. p. 82-85.

- ALLIAN, A. P.; OLIVEIRAJR, E.; NAKAGAWA, E. Y. VMTools-RA - Uma Proposta de Arquitetura de Referência para Ferramentas de Gerenciamento de Variabilidades. In: Workshop de Teses e Dissertações em Sistemas de Informações, 2015, Goiânia. Workshop de Teses e Dissertações em Sistemas de Informações, 2015. v. 1. p. 1-4.

- ALLIAN, A. P.; OLIVEIRAJR, E.; CAPILLA, R.; NAKAGAWA, E. Y. A Systematic Mapping Study of Software Variability Management Tools, ACM Computing Surveys, p. 1-35. (Submetido em Maio de 2016); e

- ALLIAN, A. P.; OLIVEIRAJR, E.; NAKAGAWA, E. Y. A Reference Architecture for Variability Management Tools, Journal of Systems and Software, 2016. p. 1-45. (a ser submetido em Julho de 2016).

6.2 Dificuldades e Limitações

Uma das dificuldades encontradas durante a realização deste trabalho foi a falta de arquiteturas de referência para o GV. Como solução, foram utilizados modelos de referência de LPS ISO/IEC 26555 (2013b) e ISO/IEC 26550 (2013a), que são mais abrangentes e abstratos. Além disso, foi necessário investigar, de maneira mais detalhada, as tecnologias utilizadas no desenvolvimento das ferramentas de variabilidade de software para apoiar a especificação da VMTools-RA.

As limitações deste trabalho estão relacionadas, principalmente, à representação da arquitetura de referência proposta. Mais detalhes acerca das visões e relações entre elas poderiam ser apresentados permitindo a identificação de mais pontos para a evolução da VMTools-RA. Além disso, um guia de instanciação detalhado poderia estar mais explícito, o que poderia facilitar o uso da VMTools-RA durante a instanciação e geração de arquiteturas concretas. Outra limitação importante foi o número de especialistas na avaliação da VMTools-RA. Apesar do conhecimento avançado dos participantes, acredita-se que novas avaliações são necessárias para permitir a liberação e adoção da VMTools-RA.

Por fim, uma vez que este trabalho foi desenvolvido em contexto acadêmico, uma participação maior da indústria poderia contribuir de forma mais efetiva para a especificação da VMTools-RA, proporcionando alguns conceitos relacionados ao domínio que podem não ter sido previstos durante o estabelecimento desta arquitetura de referência.

6.3 Trabalhos Futuros

De acordo com a experiência adquirida com a especificação da VMTools-RA somados ao estudo empírico e exemplo de aplicação, e visando a continuidade do trabalho desenvolvido, a seguir, são apresentadas as principais perspectivas de trabalhos futuros relacionadas a este trabalho.

Reavaliação da VMTools-RA: após as correções sugeridas na primeira avaliação da arquitetura de referência, uma segunda avaliação deve ser conduzida após a evolução da VMTools-RA. Além do *checklist* já utilizado, métodos de avaliação arquitetural, como SAAM e ATAM, poderão ser adaptados e aplicados com o objetivo de aprimorar a qualidade da arquitetura de referência.

Implementação do Exemplo de Aplicação: a implementação da ferramenta do exemplo de aplicação apresentado no Capítulo 5 possibilitaria evidenciar possíveis limitações e irregularidades da VMTools-RA.

Implementação de Outras Ferramentas de Variabilidade de Software com Base na VMTools-RA: a implementação de diferentes ferramentas de variabilidade de software com base na VMTools-RA poderia contribuir para fornecer evidências da efetividade da arquitetura de referência, possibilitando até uma forma de avaliação comparativa.

Planejamento e Execução de Estudos Quantitativos: análises quantitativas do aumento da capacidade de reúso e diminuição do tempo de desenvolvimento proporcionado pela VMTools-RA poderão ser realizados contribuindo para a análise de sua real efetividade.

Requisições de Melhorias: as requisições de melhorias para a VMTools-RA de acordo com especialistas apresentadas na Seção 4.6 podem ser utilizadas como trabalhos futuros na evolução dessa arquitetura de referência.

Documentar Outras Decisões de Projeto Arquitetural: Neste trabalho de mestrado é documentado apenas uma decisão de projeto arquitetural na Seção 3.4.4 com relação à integração. Outras decisões de projeto podem ser identificadas e documentadas complementando a VMTools-RA.

Reengenharia das Ferramentas de Variabilidade de Software com Base na VMTools-RA: a execução de uma engenharia reversa, analisando as funcionalidades das ferramentas de variabilidade de software com relação aos elementos da VMTools-RA, possibilitaria uma evolução da arquitetura de referência.

REFERÊNCIAS

AALIANCE *AALIANCE ambient assisted living roadmap*, v. 6 de *Ambient Intelligence and Smart Environments*. IOS Press, 2010.

Abele, A.; Lönn, H.; Reiser, M.; Weber, M.; Glathe, H. EPM: a prototype tool for variability management in component hierarchies. In: *16th International Software Product Line Conference (SPLC)*, Salvador, Brazil, 2012, p. 246–249.

Abele, A.; Papadopoulos, Y.; Servat, D.; Törngren, M.; Weber, M. The CVM framework - A prototype tool for compositional variability management. In: *4th International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS)*, Linz, Austria, 2010, p. 101–105.

Abu-Matar, M.; Gomaa, H. An automated framework for variability management of service-oriented software product lines. In: *7th IEEE International Symposium on Service-Oriented System Engineering (SOSE)*, San Francisco, CA, USA, 2013, p. 260–267.

Angelov, S.; Grefen, P. W. P. J.; Greefhorst, D. A classification of software reference architectures: Analyzing their success and effectiveness. In: *8th European Conference on Software Architecture (WICSA/ECSA)*, Cambridge, UK, 2009, p. 141–150.

Angelov, S.; Grefen, P. W. P. J.; Greefhorst, D. A framework for analysis and design of software reference architectures. *Information & Software Technology*, v. 54, n. 4, p. 417–431, 2012.

ANSI/IEEE IEEE recommended practice for architectural description of software-intensive systems. *IEEE Std 1471-2000*, p. i–23, 2000.

Antkiewicz, M.; Bak, K.; Murashkin, A.; Olaechea, R.; Liang, J. H. J.; Czarnecki, K. Clafer tools for product line engineering. In: *17th International Software Product Line Conference co-located workshops (SPLC)*, Tokyo, Japan, 2013, p. 130–135.

Apel, S.; Kästner, C.; Lengauer, C. FEATUREHOUSE: language-independent, automated software composition. In: *31st International Conference on Software Engineering (ICSE)*, Vancouver, Canada, 2009, p. 221–231.

Apel, S.; Speidel, H.; Wendler, P.; von Rhein, A.; Beyer, D. Detection of feature interactions using feature-aware verification. In: *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Lawrence, KS, USA, 2011, p. 372–375.

Asikainen, T.; Männistö, T.; Soininen, T. Using a configurator for modelling and configuring software product lines based on feature models. In: *3rd International Conference on Software Product Line (SPLC)*, Boston, MA, USA, 2004, p. 24–35.

AUTOSAR Autosar (automotive open system architecture). URL: http://www.autosar.org/, acessado em 2016, 2016.

Bachmann, F.; Clements, P. *Variability in software product lines.* Relatório Técnico 012, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

Bashroush, R. A NUI based multiple perspective variability modeling CASE tool. In: *4th European Conference on Software Architecture (ECSA)*, Copenhagen, Denmark, 2010, p. 523–526.

Bashroush, R.; Al-Nemrat, A.; Bachrouch, M.; Jahankhani, H. Visualizing variability models using hyperbolic trees. In: *2011 Conference on Advanced Information Systems Engineering (CAiSE)*, London, UK, 2011, p. 113–120.

Bass, L.; Clements, P.; Kazman, R. *Software architecture in practice.* 2 ed. Boston, MA, USA: Addison-Wesley Longman Publishing, 560 p., 2003.

Batory, D. S.; Sarvela, J. N.; Rauschmayer, A. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, v. 30, n. 6, p. 355–371, 2004.

Bécan, G.; Nasr, S. B.; Acher, M.; Baudry, B. Webfml: synthesizing feature models everywhere. In: *18th International Software Product Lines Conference (SPLC)*, Florence, Italy, 2014, p. 112–116.

Bednasch, T.; Endler, C.; Lang, M. Captainfeature. Available at `http://sourceforge.net/projects/captainfeature/` Acessado em 24 de Fevereiro 2016., 2002.

Berger, T.; Nair, D.; Rublack, R.; Atlee, J. M.; Czarnecki, K.; Wasowski, A. Three cases of feature-based variability modeling in industry. In: *17th International Conference on Model-Driven Engineering Languages and Systems (MODELS)*, Valencia, Spain, 2014, p. 302–319.

Berger, T.; Rublack, R.; Nair, D.; Atlee, J. M.; Becker, M.; Czarnecki, K.; Wsowski, A. A survey of variability modeling in industrial practice. In: *7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, Pisa, Italy, 2013, p. 7:1–7:8.

Beuche, D. Modeling and building product lines with pure: : variants. In: *17th International Software Product Line Conference co-located workshops (SPLC)*, Tokyo, Japan, 2013, p. 147–149.

Beuche, D.; Hellebrand, R. Using pure: : variants across the product line lifecycle. In: *19th International Conference on Software Product Line (SPLC)*, Nashville, TN, USA, 2015, p. 352–354.

Beuche, D.; Papajewski, H.; Schröder-Preikschat, W. Variability management with feature models. *Science of Computer Programming*, v. 53, n. 3, p. 333–352, 2004.

Bhumula, M. R. Comparative study and analysis of variability tools. *Computing Research Repository ACM (CoRR)*, v. abs/1304.3912, 2013.

Borg, Z. *A reference architecture for marine systems.* Dissertação de Mestrado, University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany, Master's Thesis, 2011.

Bosch, J.; Capilla, R.; Hilliard, R. Trends in systems and software variability. *IEEE Software*, v. 32, n. 3, p. 44–51, 2015.

Botterweck, G.; Nestor, D.; Preussner, A.; Cawley, C.; Thiel, S. Towards supporting feature configuration by interactive visualisation. In: *11th International Conference on Software Product Lines (SPLC)*, Kyoto, Japan, 2007, p. 125–131.

Botterweck, G.; Thiel, S.; Cawley, C.; Nestor, D.; Preussner, A. Visual configuration in automotive software product lines. In: *32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, Turku, Finland, 2008, p. 1070–1075.

Braga, R. M. M.; Werner, C. M. L.; Mattoso, M. Odyssey: a reuse environment based on domain models. In: *IEEE Symposium on the Application-Specific Systems and Software Engineering and Technology (ASSET)*, Richardson, TX, USA, 1999, p. 50–57.

Buchgeher, G.; Weinreich, R. Towards continuous reference architecture conformance analysis. In: *Software Architecture*, v. 7957 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 332–335, 2013.

Capilla, R.; Bosch, J.; Kang, K. C. *Systems and software variability management, concepts, tools and experiences*, v. 14. Springer, 317 p., 2013.

Capilla, R.; Sánchez, A.; Dueñas, J. C. An analysis of variability modeling and management tools for product line development. In: *Software and Services Variability Management Workshop–Concept Models and Tools*, Helsinki, Finland, 2007, p. 32–47.

Chen, L.; Babar, M. A.; Ali, N. Variability management in software product lines: A systematic review. In: *13th International Software Product Line Conference(SPLC9)*, San Francisco, California, USA: Carnegie Mellon University, 2009, p. 81–90.

Cirilo, E.; Kulesza, U.; de Lucena, C. J. P. Genarch-a model-based product derivation tool. In: *1st Brazilian Symposium on Software Components Architectures and Reuse (SBCARS)*, Campinas, Brazil, 2007, p. 31–46.

Cirilo, E.; Nunes, I.; Kulesza, U.; de Lucena, C. J. P. Automating the product derivation process of multi-agent systems product lines. *Journal of Systems and Software*, v. 85, n. 2, p. 258–276, 2012.

Classen, A.; Cordy, M.; Heymans, P.; Legay, A.; Schobbens, P.-Y. Model checking software product lines with snip. *International Journal on Software Tools for Technology Transfer*, v. 14, n. 5, p. 589–612, 2012.

Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Merson, P.; Nord, R.; Stafford, J. *Documenting software architectures: Views and beyond*. 2 ed. Portland, Oregon, USA: Pearson Education, 740–741 p., 2010.

Cloutier, R. J.; Muller, G.; Verma, D.; Nilchiani, R.; Hole, E.; Bone, M. A. The concept of reference architectures. *System Engineer*, v. 13, n. 1, p. 14–27, 2010.

Continua Continua health alliance. URL: http://www.continuaalliance.org/, acessado em 2016, 2016.

Cook, S. A. The complexity of theorem-proving procedures. In: *3rd Annual ACM Symposium on Theory of Computing*, Shaker Heights, Ohio, USA, 1971, p. 151–158.

Czarnecki, K.; Antkiewicz, M.; Kim, C. H. P.; Lau, S.; Pietroszek, K. fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates. In: *20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, San Diego, CA, USA, 2005a, p. 200–201.

Czarnecki, K.; Eisenecker, U. W. *Generative programming: Methods, tools, and applications*. New York, NY, USA: Addison-Wesley, 2000.

Czarnecki, K.; Helsen, S.; Eisenecker, U. W. Formalizing cardinality-based feature models and their specialization. *Software process: Improvement and practice*, v. 10, n. 1, p. 7–29, 2005b.

Deelstra, S.; Sinnema, M.; Bosch, J. Variability assessment in software product families. *Information & Software Technology*, v. 51, n. 1, p. 195–218, 2009.

Dehlinger, J.; Humphrey, M.; Suvorov, L.; Padmanabhan, P.; Lutz, R. Decimal and plfaultcat: From product-line requirements to product-line member software fault trees. In: *29th International Conference on Software Engineering (ICSE)*, Washington, DC, USA, 2007, p. 49–50.

Dhungana, D.; Grünbacher, P.; Rabiser, R. Decisionking: A flexible and extensible tool for integrated variability modeling. In: *1st International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, Limerick, Ireland, 2007, p. 119–127.

Dhungana, D.; Grünbacher, P.; Rabiser, R. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, v. 18, n. 1, p. 77–114, 2011.

Dhungana, D.; Seichter, D.; Botterweck, G.; Rabiser, R.; Grünbacher, P.; Benavides, D.; Galindo, J. A. Integrating heterogeneous variability modeling approaches with invar. In: *7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, Pisa , Italy, 2013, p. 8:1–8:5.

Duarte, L. S. *Establishment of a reference architecture for digital television applications*. Dissertação de Mestrado, University of São Paulo, São Carlos, Brazil, master Thesis, 2012.

Duisburg-Essen, S. S. E. R. G. U. Varmod-prime - process integration of modelling workplaces. Available at `https://sse.uni-due.de/en/research/projects/varmod-prime/` Acessado em 6 de Março de 2016, 2005.

Dybå, T.; Dingsøyr, T.; Hanssen, G. K. Applying systematic reviews to diverse study types: An experience report. In: *1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Madrid,Spain, 2007, p. 225–234.

Dybå, T.; Prikladnicki, R.; Rönkkö, K.; Seaman, C. B.; Sillito, J. Qualitative research in software engineering. *Empirical Software Engineering*, v. 16, n. 4, p. 425–429, 2011.

Eeles, P.; Cripps, P. *The process of software architecting*. Addison-Wesley Professional, 2010.

Eichelberger, H.; Schmid, K. Easy-producer - A product line production environment. In: *12th International Conference on Software Product Lines (SPLC)*, Limerick, Ireland, 2008, p. 357.

Eklund, U.; Örjan Askerdal; Granholm, J.; Alminger, A.; Axelsson, J. Experience of introducing reference architectures in the development of automotive electronic systems. *SIGSOFT Software Engineering*, v. 30, n. 4, p. 1–6, 2005.

Eklund, U.; Jonsson, N.; Bosch, J.; Eriksson, A. A reference architecture template for software-intensive embedded systems. In: *Proceedints on the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA)*, Helsinki, Finland, 2012, p. 104–111.

Eriksson, M.; Morast, H.; Börstler, J.; Borg, K. The PLUSS toolkit - extending telelogic DOORS and ibm-rational rose to support product line use case modeling. In: *20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Long Beach, CA, USA, 2005, p. 300–304.

Feitosa, D. *Simus - reference architecture for service multirobotics systems.* Dissertação de Mestrado, University of São Paulo, São Carlos, Brazil, master Thesis, 2013.

Fioravanti, M. L.; Nakagawa, E. Y.; Barbosa, E. F. Educar: Uma arquitetura de referência para ambientes educacionais. In: *Anais do Simpósio Brasileiro de Informática na Educação*, 2010.

Frakes, W. B.; Díaz, R. P.; Fox, C. J. DARE-COTS: A domain analysis support tool (invited paper). In: *17th International Conference of the Chilean Computer Science Society (SCCC)*, Valpariso, Chile, 1997, p. 73–77.

Frank, A.; Brenner, E. Model-based variability management for complex embedded networks. In: *5th International Multi-Conference on Computing in the Global Information Technology (ICCGI)*, Washington, DC, USA, 2010, p. 305–309.

Galindo, J. A.; Dhungana, D.; Rabiser, R.; Benavides, D.; Botterweck, G.; Grünbacher, P. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information & Software Technology*, v. 62, p. 78–100, 2015.

Galster, M.; Avgeriou, P.; Tofan, D. Constraints for the design of variability-intensive service-oriented reference architectures - an industrial case study. *Information and Software Technology*, v. 55, n. 2, p. 428 – 441, 2013.

Galster, M.; Weyns, D.; Tofan, D.; Michalik, B.; Avgeriou, P. Variability in software systems - A systematic literature review. *IEEE Trans. Software Eng.*, v. 40, n. 3, p. 282–306, 2014.

Gamez, N.; Fuentes, L. Software product line evolution with cardinality-based feature models. In: Schmid, K., ed. *Top Productivity through Software Reuse*, v. 6727 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 102–118, 2011.

Gauthier, C.; Classen, A.; Boucher, Q.; Heymans, P.; Storey, M. D.; Mendonça, M. Xtof - A tool for tag-based product line implementation. In: *4th*

International Workshop on Variability Modelling of Software-Intensive Systems (Vamos), Linz, Austria, 2010, p. 163–166.

Gómez, A.; del Carmen Penadés, M.; Canós, J. H.; Borges, M. R. S.; Llavador, M. Dplfw: a framework for variable content document generation. In: *16th International Software Product Line Conference (SPLC)*, Salvador, Brazil, 2012, p. 96–105.

Gotel, O. C. Z.; Finkelstein, A. An analysis of the requirements traceability problem. In: *Proceedings of the First IEEE International Conference on Requirements Engineering (ICRE)*, Colorado, USA: IEEE, 1994, p. 94–101.

Greefhorst, D.; Proper, E. *Architecture principles: The cornerstones of enterprise architecture*, v. 4 de *The Enterprise Engineering Series*. Springer, 197 p., 2011.

Groher, I.; Weinreich, R. Integrating variability management and software architecture. In: *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA)*, Helsinki, Finland, 2012, p. 262–266.

Groher, I.; Weinreich, R. Supporting variability management in architecture design and implementation. In: *46th Hawaii International Conference on System Sciences (HICSS)*, Wailea, HI, USA, 2013, p. 4995–5004.

Gurp, J. V.; Bosch, J.; Svahnberg, M. On the notion of variability in software product lines. In: *2001 Working IEEE / IFIP Conference on Software Architecture (WICSA)*, Amsterdam, The Netherlands, 2001, p. 45–54.

Halmans, G.; Pohl, K. Communicating the variability of a software-product family to customers. *Software and System Modeling*, v. 2, n. 1, p. 15–36, 2003.

Hamilton, M. H. 001: A full life cycle systems engineering and software development environment development before the fact in action. Available at `http://www.htius.com/Articles/Full_Life_Cycle.htm` Acessado em 20 de Junho de 2015, 1991.

Heidenreich, F.; Kopcsek, J.; Wende, C. Featuremapper: mapping features to models. In: *30th International Conference on Software Engineering (ICSE)*, Leipzig, Germany, 2008, p. 943–944.

Hervieu, A.; Baudry, B.; Gotlieb, A. PACOGEN: automatic generation of pairwise test configurations from feature models. In: *IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE)*, Hiroshima, Japan, 2011, p. 120–129.

Hilliard, R. Using the UML for architectural description. In: *2nd International Conference Beyond the Standard UML The Unified Modeling Language*, Fort Collins,CO, USA, 1999, p. 32–48.

van der Hoek, A. Design-time product line architectures for any-time variability. *Science of Computer Programming*, v. 53, n. 3, p. 285–304, 2004.

Hofmeister, C.; Nord, R.; Soni, D. *Applied software architecture.* The Addison-Wesley object technology series. Addison-Wesley, 397 p., 2000.

ISO/IEC Software and systems engineering - Reference model for product line engineering and management (ISO/IEC 26550). 2013a.

ISO/IEC Software and systems engineering - Tools and methods for product line technical management (ISO/IEC 26555). 2013b.

Jain, A.; Biesiadecki, J. Yam - a framework for rapid software development. In: *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, Pasadena, CA, USA, 2006, p. 182–194.

Jézéquel, J.-M. Model-driven engineering for software product lines. *ISRN Software Engineering*, v. 2012 (2012), Article ID 670803, 2012.

Käkölä, T. Standards initiatives for software product line engineering and management within the international organization for standardization. In: *43rd Hawaii International International Conference on Systems Science (HICSS-43*, Hawaii, USA: IEEE Computer Society, 2010, p. 1–10.

Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S. *Feature-oriented domain analysis (foda): Feasibility study ;.* Relatório Técnico Technical Report CMU/SEI-90-TR-21 - ESD-90-TR-222, CMU/SEI, 1990.

Kang, K. C.; Kim, S.; Lee, J.; Kim, K.; Shin, E.; Huh, M. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annual Software Engeener*, v. 5, p. 143–168, 1998.

Kästner, C.; Apel, S.; Kuhlemann, M. Granularity in software product lines. In: *30th International Conference on Software Engineering (ICSE)*, Leipzig, Germany, 2008, p. 311–320.

Khan, A. I.; Alam, M. M.; Al Jedaibi, W. Variability management in software development using featureide: A case study. *IJSER - International Journal of Scientific e Engineering Research*, v. 6, 2015.

Kim, K.; Kim, H.; Ahn, M.; Seo, M.; Chang, Y.; Kang, K. C. ASADAL: a tool system for co-development of software and test environment based on product line engineering. In: *28th International Conference on Software Engineering (ICSE)*, Shanghai, China, 2006, p. 783–786.

Kitchenham, B.; Brereton, P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. G. Systematic literature reviews in software engineering. *Information and Software Technology*, v. 51, n. 1, p. 7–15, 2009.

Kitchenham, B.; Charters, S. *Guidelines for performing systematic literature reviews in software engineering.* Relatório Técnico, Keele University and Durham University Joint Report, 2007.

Krueger, C. W. Biglever software gears and the 3-tiered SPL methodology. In: *22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Montreal, Quebec, Canada, 2007, p. 844–845.

Krueger, C. W. The biglever software gears unified software product line engineering framework. In: *12th International Conference on Software Product Lines (SPLC)*, Limerick, Ireland, 2008, p. 353.

Krueger, C. W.; Clements, P. C. Systems and software product line engineering with gears from biglever software. In: *18th International Software Product Lines Conference (SPLC)*, Florence, Italy, 2014, p. 121–125.

Laguna, M. A.; Hernández, C. A software product line approach for e-commerce systems. In: *7th International Conference on e-Business Engineering (ICEBE)*, Shanghai, China, 2010, p. 230–235.

Lamprecht, A.; Naujokat, S.; Schaefer, I. Variability management beyond feature models. *IEEE Computer*, v. 46, n. 11, p. 48–54, 2013.

Lee, H.; Yang, J.; Kang, K. C. VULCAN: architecture-model-based workbench for product line engineering. In: *16th International Software Product Line Conference (SPLC)*, Salvador, Brazil, 2012, p. 260–264.

Lettner, D.; Petruzelka, M.; Rabiser, R.; Angerer, F.; Prähofer, H.; Grünbacher, P. Custom-developed vs. model-based configuration tools: experiences from an industrial automation ecosystem. In: *17th International Software Product Line Conference co-located workshops (SPLC)*, Tokyo, Japan, 2013, p. 52–58.

Linden, F. J. V. D.; Schmid, K.; Rommes, E. *Software product lines in action: The best industrial practice in product line engineering*, v. 20. Springer-Verlag New York, Inc., 333 p., 2007.

Lisboa, L. B.; Garcia, V. C.; de Almeida, E. S.; de Lemos Meira, S. R. Toolday: a tool for domain analysis. *STTT*, v. 13, n. 4, p. 337–353, 2011.

Lisboa, L. B.; Garcia, V. C.; Lucrédio, D.; de Almeida, E. S.; de Lemos Meira, S. R.; de Mattos Fortes, R. P. A systematic review of domain analysis tools. *Information and Software Technology*, v. 52, n. 1, p. 1–13, 2010.

López, C.; Mansell, J. X. PLUM: product line unified modeler tool. In: *Systems and Software Variability Management, Concepts, Tools and Experiences*, Springer Heidelberg, p. 151–161, 2013.

Machado, L.; Pereira, J.; Garcia, L.; Figueiredo, E. Splconfig product configuration in software product line. In: *Congress on Brazilian Software (CBSOFT)*, Maceio, Alagoas, BRA, 2014, p. 1–8.

Marijan, D.; Gotlieb, A.; Sen, S.; Hervieu, A. Practical pairwise testing for software product lines. In: *17th International Software Product Line Conference (SPLC)*, Tokyo, Japan, 2013, p. 227–235.

Martínez-Fernández, S. Towards supporting the adoption of software reference architectures: An empirically-grounded framework. In: *11th International Doctoral Symposium on Empirical Software Engineering((IDoESE)*, Baltimore, 2013, p. 1–8.

Martínez-Fernández, S.; Ayala, C. P.; Franch, J.; Marques, H. M.; Ameller, D. A framework for software reference architecture analysis and review. In: *Memorias del X Workshop Latinoamericano Ingeniería de Software Experimental (ESELAW)*, Montevideo, Uruguay: Experimental Software Engineering Latin American Workshop, 2013a, p. 89–102.

Martínez-Fernández, S.; Ayala, C. P.; Franch, X.; Marques, H. M. Rearm: A reuse-based economic model for software reference architectures. In: *Safe and Secure Software Reuse*, v. 7925 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 97–112, 2013b.

Massen, T.; Lichter, H. Requiline: A requirements engineering tool for software product lines. In: *5th International Workshop Software Product-Family Engineering (PFE)*, Siena, Italy, 2004, p. 168–180.

Matos, P.; Santos, D.; Neto, C.; Almeida, E. *Spl-tupi: A tool to support product instantiation of software product line projects.* Relatório Técnico, Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA), 2015.

Mazo, R.; Muñoz-Fernández, J. C.; Rincón, L.; Salinesi, C.; Tamura, G. Variamos: an extensible tool for engineering (dynamic) product lines. In: *19th*

International Conference on Software Product Line (SPLC), Nashville, TN, USA, 2015, p. 374–379.

Mendonça, M.; Branco, M.; Cowan, D. Splot: Software product lines online tools. In: *24th Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, Orlando, FL, USA, 2009a, p. 761–762.

Mendonça, M.; Branco, M.; Cowan, D. S.p.l.o.t.: Software product lines online tools. In: *24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, Orlando, Florida, USA, 2009b, p. 761–762.

Metzger, A.; Pohl, K. Software product line engineering and variability management: achievements and challenges. In: *Future of Software Engineering (FOSE)*, Hyderabad, India, 2014, p. 70–84.

Myllärniemi, V.; Raatikainen, M.; Männistö, T. Kumbang tools. In: *11th International Conference on Software Product Lines (SPLC)*, Kyoto, Japan, 2007, p. 135–136.

Nakagawa, E. Y.; Antonino, P. O.; Becker, M. Reference architecture and product line architecture: A subtle but critical difference. In: *5th European Conference on Software Architecture(ECSA)*, Berlin, Heidelberg: Springer-Verlag, 2011, p. 207–211.

Nakagawa, E. Y.; Becker, M.; Maldonado, J. C. Towards a process to design product line architectures based on reference architectures. In: *17th International Software Product Line Conference (SPLC)*, Tokyo, Japan: ACM, 2013, p. 157–161.

Nakagawa, E. Y.; Guessi, M.; Maldonado, J. C.; Feitosa, D.; Oquendo, F. Consolidating a process for the design, representation, and evaluation of reference architectures. In: *2014 IEEE/IFIP Conference on Software Architecture*, Washington, DC, USA: IEEE Computer Society, 2014, p. 143–152.

Nakagawa, E. Y.; Martins, R.; Felizardo, K.; Maldonado, J. C. Towards a process to design aspect-oriented reference architectures. In: *XXXV Latin American Informatics Conference (CLEI'2009)*, Pelotas, Brazil, 2009, p. 1–10.

Nakagawa, E. Y.; da Silva Simão, A.; Ferrari, F. C.; Maldonado, J. C. Towards a reference architecture for software testing tools. In: *9th International Conference on Software Engineering & Knowledge Engineering (SEKE)*, Boston, Massachusetts, 2007, p. 157–162.

Nestor, D.; Thiel, S.; Botterweck, G.; Cawley, C.; Healy, P. Applying visualisation techniques in software product lines. In: *2008 Symposium on Software Visualization (SoftVis)*, Ammersee, Germany, 2008, p. 175–184.

Nuseibeh, B.; Easterbrook, S. Requirements engineering: A roadmap. In: *22nd International Conference on on Software Engineering, Future of Software Engineering Track (ICSE)*, Limerick, Ireland: ACM, 2000, p. 35–46.

Oliveira, L. B. R.; Nakagawa, E. Y. A service-oriented reference architecture for software testing tools. In: *5th European Conference on Software Architecture (ECSA)*, Essen, Germany, 2011, p. 405–421.

OliveiraJr, E.; Allian, A. P. Do reference architectures can contribute to standardizing variability management tools? In: *1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures*, Montreal, QC, Canada: ACM, 2015, p. 9–12.

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

Oster, S.; Zorcic, I.; Markert, F.; Lochau, M. Moso-polite: tool support for pairwise and model-based software product line testing. In: *5th International Workshop on Variability Modelling of Software-Intensive Systems (Vamos)*, Namur, Belgium, 2011, p. 79–82.

Ozkaya, I.; Ömer Akin Tool support for computer-aided requirement traceability in architectural design: The case of designtrack. *Automation in Construction*, v. 16, n. 5, p. 674 – 684, 2007.

Papendieck, M.; Schulze, M. Concepts for consistent variant-management tool integrations. In: *Workshops der Tagung Software Engineering Gemeinsamer Tagungsband*, Kiel, Deutschland, 2014, p. 37–46.

Park, J.; Moon, M.; Yeom, K. Dream: domain requirement asset manager in product lines. In: *International Symposium on Future Software Technology (ISFST)*, Xian, China, 2004, p. 1–6.

Park, K.; Ryu, D.; Baik, J. An integrated software management tool for adopting software product lines. In: *11th International Conference on Computer and Information Science (ICIS)*, Shanghai, China, 2012, p. 553–558.

Pasetti, A.; Rohlik, O. *Technical note on a concept for the xfeature tool.* Relatório Técnico, PnP Software and GmbH / ETH Zurich, 2005.

Pereira, J. A.; Constantino, K.; Figueiredo, E. A systematic literature review of software product line management tools. In: *14th International Conference on Software*

Reuse for Dynamic Systems in the Cloud and Beyond (ICSR), Miami, FL, USA, 2015, p. 73–89.

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

Pleuss, A.; Botterweck, G. Visualization of variability and configuration options. *STTT*, v. 14, n. 5, p. 497–510, 2012.

Pohl, K.; Böckle, G.; van der Linden, F. J. *Software product line engineering: Foundations, principles and techniques*, v. 26. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 467 p., 2005.

Rabiser, R.; Wolfinger, R.; Grünbacher, P. Three-level customization of software products using a product line approach. In: *42nd Hawaii International International Conference on Systems Science (HICSS)*, Big Island, HI, USA, 2009, p. 1–10.

Rodrigues, E. M.; Zorzo, A. F.; OliveiraJr, E.; de Souza Gimenes, I. M.; Maldonado, J. C.; Domingues, A. R. P. Plugspl: An automated environment for supporting plugin-based software product lines. In: *24th International Conference on Software Engineering & Knowledge Engineering (SEKE)*, Francisco Bay, USA, 2012, p. 647–650.

Roos-Frantz, F.; Galindo, J. A.; Benavides, D.; Cortés, A. R. Fama-ovm: a tool for the automated analysis of ovms. In: *16th International Software Product Line Conference (SPLC)*, Salvador, Brazil, 2012, p. 250–254.

Rozanski, N.; Woods, E. *Software systems architecture: Working with stakeholders using viewpoints and perspectives*. Addison-Wesley Professional, 2005.

Saldana, J. *The coding manual for qualitative researchers*. SAGE Publications, 240 p., 2009.

Samih, H.; Bogusch, R. MPLM - matelo product line manager: [relating variability modelling and model-based testing]. In: *18th International Software Product Lines Conference (SPLC)*, Florence, Italy, 2014, p. 138–142.

Santos, J. F. M.; Guessi, M.; Galster, M.; Feitosa, D.; Nakagawa, E. Y. A checklist for evaluation of reference architectures of embedded systems (S). In: *25th International Conference on Software Engineering and Knowledge Engineering*, Boston, MA, USA, 2013, p. 451–454.

Saratxaga, C. L.; Alonso-Montes, C.; Haugen, Ø.; Ekelin, C.; Mitschke, A. Product line tool-chain: variability in critical systems. In: *3rd International Workshop on Product LinE Approaches in Software Engineering (PLEASE)*, Zurich, Switzerland, 2012, p. 57–60.

Schmid, K.; Krennrich, K.; Eisenbarth, M. Requirements management for product lines: Extending professional tools. In: *10th International Conference on Software Product Lines (SPLC)*, Baltimore, Maryland, USA, 2006, p. 113–122.

Schmid, K.; Schank, M. Pulse-beat – A decision support tool for scoping product lines. In: *International Workshop on Software Architectures for Product Families (ISAPF)*, Las Palmas de Gran Canaria, Spain, 2000, p. 65–75.

Schnabel, T.; Weckesser, M.; Kluge, R.; Lochau, M.; Schürr, A. Cardygan: Tool support for cardinality-based feature models. In: *10th International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, Salvador, Brazil, 2016, p. 33–40.

Segura, S.; Benavides, D.; Ruiz-Cortés, A.; Trinidad, P. Open source tools for software product line development. *Open Source and Product Lines*, 2007.

SEI Defining software architecture. Disponível em `http://www.sei.cmu.edu/architecture/index.cfm`. Último acesso em 20 de Maio de 2016, 2016.

Shaw, M.; Garlan, D. *Software architecture: Perspectives on an emerging discipline.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

Sinnema, M.; Deelstra, S. Industrial validation of COVAMOF. *Journal of Systems and Software*, v. 81, n. 4, p. 584–600, 2008.

Soares, S.; Calheiros, F.; Nepomuceno, V.; Menezes, A.; Borba, P.; Alves, V. Supporting software product lines development: Flip - product line derivation tool. In: *23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Nashville, TN, USA, 2008, p. 737–738.

Strauss, A. L.; Corbin, J. M. *Basics of qualitative research: Techniques and procedures for developing grounded theory.* second ed. SAGE Publications, 1998.

Stuart, D.; Sull, W.; Pruitt, S.; Cobb, D.; Waskiewicz, F.; Cook, T. W. The SSEP toolset for product line development. In: *1st International Conference on Software Product Lines (SPLC)*, Denver, Colorado, USA, 2000, p. 413–436.

Succi, G.; Yip, J.; Liu, E.; Pedrycz, W. Holmes: a system to support software product lines. In: *22nd International Conference on on Software Engineering (ICSE)*, Limerick Ireland, 2000, p. 786.

Succi, G.; Yip, J.; Pedrycz, W. Holmes: An intelligent system to support software product line development. In: *23rd International Conference on Software Engineering (ICSE)*, Toronto, Ontario, Canada, 2001, p. 829–830.

Svendsen, A.; Zhang, X.; Fleurey, F.; Haugen, Ø.; Olsen, G. K.; Møller-Pedersen, B. CVL tool - modeling variability in spls. In: *14th International Conference on Software Product Lines (SPLC)*, Jeju Island, South Korea, 2010, p. 299.

Thao, C.; Munson, E. V.; Nguyen, T. N. Software configuration management for product derivation in software product families. In: *15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS)*, Belfast, Northern Ireland, 2008, p. 265–274.

Thurimella, A.; Bruegge, B.; Janzen, D. Variability plug-ins for requirements tools: A case-based theory building approach. *Systems Journal, IEEE*, v. PP, n. 99, p. 1–12, 2015.

Thurimella, A. K.; Bruegge, B. Issue-based variability management. *Information & Software Technology*, v. 54, n. 9, p. 933–950, 2012.

Thurimella, A. K.; Janzen, D. Metadoc feature modeler: A plug-in for IBM rational DOORS. In: *15th International Conference in Software Product Lines (SPLC)*, Munich, Germany, 2011, p. 313–322.

Tolvanen, J. Metaedit+: Domain-specific modeling and product generation environment. In: *11th International Conference on Software Product Lines (SPLC)*, Kyoto, Japan, 2007, p. 145–146.

Tracz, W. Dssa (domain-specific software architecture): Pedagogical example. *SIGSOFT Software Engineering Notes*, v. 20, n. 3, p. 49–62, 1995.

Turnes, L.; Bonifácio, R.; Alves, V.; Lämmel, R. Techniques for developing a product line of product line tools: A comparative study. In: *5th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, São Paulo, Brazil, 2011, p. 11–20.

UniversAAL The universaal reference architecture. URL: http://www.universaal.org/, acessado em 2016, 2016.

Varela, P.; Araújo, J.; Brito, I. S.; Moreira, A. Aspect-oriented analysis for software product lines requirements engineering. In: *2011 ACM Symposium on Applied Computing (SAC)*, TaiChung, Taiwan, 2011, p. 667–674.

Vasilevskiy, A.; Haugen, Ø.; Chauvel, F.; Johansen, M. F.; Shimbara, D. The BVR tool bundle to support product line engineering. In: *19th International Conference on Software Product Line (SPLC)*, Nashville, TN, USA, 2015, p. 380–384.

Wagner, M.; Dudeck, G.; Hein, C.; Tcholtchev, N.; Gebhardt, C.; Korff, A. VARIES framework to support tool integration in product line engineering. In: *18th International Software Product Lines Conference (SPLC)*, Florence, Italy, 2014, p. 117–120.

Weber, S.; Chan, H.; Degenaro, L.; Diament, J.; Fokoue-Nkoutche, A.; Rouvellou, I. Fusion: A system for business users to manage program variability. *IEEE Trans. Software Eng.*, v. 31, n. 7, p. 570–587, 2005.

Wiegers, K.; Beatty, J. *Software requirements, developer best practices.* 3 ed. Redmond, WA, USA: Pearson Education, 672 p., 2013.

Wohlin, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *8th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, London, England, 2014, p. 1–10.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in software engineering: An introduction.* Norwell, MA, USA: Springer-Verlag Berlin Heidelberg, 2000.

Wong, P. Y. H.; Albert, E.; Muschevici, R.; Proença, J.; Schäfer, J.; Schlatte, R. The ABS tool suite: modelling, executing and analysing distributed adaptable object-oriented systems. *International Journal on Software Tools for Technology Transfer (STTT)*, v. 14, n. 5, p. 567–588, 2012.

Apêndice A - Mapeamento Sistemático da Literatura de Ferramentas de Variabilidade de Software

A.1 Método de Pesquisa

Um Mapeamento Sistemática da Literatura (MS) é um meio de identificar, sintetizar e interpretar informações de pesquisa relevantes para uma questão de pesquisa especifica, um tópico, ou fenômeno de interesse (Kitchenham et al., 2009; Petersen et al., 2015). Os estudos individuais que contribuem para um MS são chamados de estudos primários, enquanto que a MS é uma forma de estudo secundário (Petersen et al., 2015). De acordo com Kitchenham e Charters 2007 e Petersen et al., (2015), algum dos motivos para executar um MS são:

1. Identificação da necessidade do estudo;
2. Definição da questão de pesquisa;
3. Definição da estratégia de pesquisa;
4. Seleção inicial dos estudos primários;
5. Aplicação dos critérios de inclusão/exclusão;
6. Seleção dos estudos; e
7. Análise dos resultados;

A.1.1 Necessidade do Estudo

A necessidade de um estudo é identificada durante a fase de planejamento e foi conduzida pela percepção da falta de uma análise precisa sobre ferramentas de variabilidades de software. MSs têm sido conduzidas em torno de um tópico semelhante, mas com escopo mais abrangente uma vez que incluem pesquisas sobre LPS. Para nosso conhecimento,

nenhum dos estudos anteriores está focado exclusivamente em notações e ferramentas de variabilidades de software. Portanto, foi identificado um nicho para ser explorado abrangendo o estado da arte em que o GV avançou nos últimos anos e quais ferramentas de variabilidades de software têm sido utilizadas na indústria.

A.1.2 Questões de Pesquisa

O principal objetivo desse MS inclui: (i) identificar na literatura ferramentas de variabilidades de software; e (ii) apresentar ao leitor informações relacionados ao tipo de ferramentas de variabilidades de software disponíveis, se as ferramentas foram avaliadas pela indústria e aspectos relacionados a tecnologias utilizadas para a implementação. Deste modo, foram definidas as seguintes questões de pesquisa:

Questão de Pesquisa (RQ1): *Quais tipos de ferramentas de variabilidades de software têm sido desenvolvidas e utilizadas?*

Ao responder **RQ1**, identificou-se os tipos de ferramentas disponíveis desde ferramentas para pesquisa até as comerciais, que cobrem a maioria das atividades de GV. Contudo, a fim de proporcionar um contexto mais completo para a compreensão de ferramentas de variabilidades de software, a questão de pesquisa **RQ1** foi refinada na seguinte sub-questão:

- **RQ1.1** *Qual notação e extensões de modelagem de features são suportadas nas ferramentas de variabilidades de software existentes?*

Questão de Pesquisa (RQ2): *Quais são as tecnologias fundamentais que apoiam as ferramentas de variabilidades de software?*

A questão de pesquisa **RQ2** concentra-se nas tecnologias e interoperabilidade das ferramentas capazes de interagir com outras ferramentas e/ou tecnologias relacionadas.

A.1.3 Estratégia de Pesquisa

A estratégia de pesquisa tem o objetivo de identificar o máximo de estudos relevantes possíveis, por isso foi conduzido um MS consultando sete fontes de literatura digitais: Association for Computing Machinery Digital Library (ACM DL)[1], IEEE Xplore Digital Library[2], Google Scholar[3], Ei Compendex[4], Science Direct[5], Scopus[6], e Springer[7]. Os estudos primários selecionados e outros estudos relevantes utilizando um conjunto de

[1]http://dl.acm.org
[2]http://ieeexplore.ieee.org
[3]https://scholar.google.com.br
[4]https://www.engineeringvillage.com
[5]http://www.sciencedirect.com
[6]http://www.scopus.com
[7]http://link.springer.com

palavras chaves que foram definidas para diferenciar nossa pesquisa de outros estudos relacionados. A *string* de pesquisa utilizada nessa estratégia envolve as palavras mais adequadas relativas ao escopo desse MS, e é expressada como uma formula booleana como segue:

Query: a *string* de pesquisa utilizada neste estudo foi construída utilizando uma combinação de palavras chaves e variações relevantes utilizando os operadores lógicos "AND" e "OR" como observado na Figura 1.1. Devido à diversidade dos meios de pesquisa e recursos fornecidos pelas principais fontes digitais, foi utilizado diferentes *strings* de pesquisa para cada fonte bibliográfica, conforme apresentado na Tabela 1.1.

((“gerenciamento de variabilidade”OR “customização da variabilidade”OR “configuração da variabilidade”OR “realização da variabilidade”) AND (“software”) AND (ferramenta OR framework OR plugin))

Figura 1.1: Query geral

Tabela 1.1: Fontes de literatura e respectivas queries

Fontes de Literatura	**Queries**
ACM DL	(+“gerenciamento variabilidade”+software ferramenta* framework plug* “variabilidade customização” “variabilidade configuração” “realização variabilidade”)
IEEE Xplore Springer Google Scholar	((“gerenciamento variabilidade”OR “customização variabilidade”OR “configuração variabilidade”OR “realização variabilidade”) AND (“software”) AND (ferramenta* OR framework OR plug*))
Ei Compendex	(gerenciamento variabilidade OR customização variabilidade OR configuração variabilidade OR realização variabilidade) AND (software) AND (ferramenta* OR framework OR plug*)
Science Direct	TITLE-ABSTR-KEY (“gerenciamento variabilidade”OR “customização variabilidade” OR “configuração variabilidade”OR “realização variabilidade”) AND (software) AND (ferramenta* OR framework OR plug*)
Scopus	TITLE-ABS-KEY (“gerenciamento variabilidade”OR “customização variabilidade”OR “configuração variabilidade”OR “realização variabilidade”AND software AND ferramenta* OR framework OR plug*)

Como a estratégia de busca utilizada pode não recuperar todos os estudos importantes, utilizou-se a técnica de *snowballing* tanto o método *backward* como o *forward* (Wohlin, 2014; Wohlin et al., 2000) para identificar potenciais estudos. No método *backward*, as referências dos estudos selecionados são verificadas para aquisição de novas publicações, enquanto que no método *forward* as publicações que citam os estudos selecionados são examinadas. Em ambos os casos, a técnica *snowballing* termina quando a convergência é alcançada e nenhum novo estudo pode ser encontrado.

Fontes de Pesquisa: Foi realizado dois tipos de pesquisa, uma pesquisa eletrônica em sete fontes de literatura digital apresentado na Tabela 1.1, e uma pesquisa manual em 15 específicos anais de conferências, como apresentado na Tabela 1.2. Embora, as fontes

de literatura digital apresentados previamente na Tabela 1.1 cobrem a maioria dos anais de conferência apresentados na Tabela 1.2, decidiu-se conduzir uma pesquisa manual em tais conferências para evitar a falta de estudos importantes devido ao fato de que um MS pode experimentar variações nos mecanismos de busca disponíveis (Dybå et al., 2007). Adicionalmente, foi realizado uma pesquisa manual em sites da web e relatórios técnicos para reunir mais informações quando uma descrição de uma ferramenta de variabilidades de software estava disponível online.

Tabela 1.2: Locais de publicação

Acrônimo	**Locais de Publicação**
CAiSE	Conference on Advanced Information Systems Engineering
COMPSAC	Computer Software and Applications Conference
ECSA	European Conference on Software Architecture
ICEIS	International Conference on Enterprise Information Systems
ICSE	International Conference on Software Engineering
ICSR	International Conference on Software Reuse
MoDELS	Model Driven Engineering Languages and Systems
OOPSLA	Conference on Object-Oriented Programming Systems, Languages, and Applications
SAC	ACM Symposium on Applied Computing
SBCARS	Brazilian Symposium on Software Components, Architectures and Reuse
SBES	Brazilian Symposium on Software Engineering
SEKE	Software Engineering and Knowledge Engineering
SPLC	Software Product Lines Conference
VaMoS	Variability Modeling of Software-Intensive Systems
WICSA	Working IEEE/IFIP Conference on Software Architecture

Idioma: foi considerado somente estudos escritos em Inglês;

Tipos de Publicação: pesquisou-se anais de conferência, revistas, capítulos de livros, web sites e relatórios técnicos;

Ano de Publicação: pesquisou-se por estudos publicados entre 1990 e (incluindo) 2016; 1990 foi escolhido porque a primeira notação de variabilidades, FODA (*Feature Oriented Domain Analysis*) (Kang et al., 1990), foi desenvolvida durante esse ano.

A.1.4 Seleção Inicial

A pesquisa pelos estudos envolveu duas fases: primeiro, uma busca automática por estudos através de fontes de literatura digital que retornaram 682 documentos e segundo, uma pesquisa manual em anais de conferência que retornaram 197 documentos. Essas pesquisas sumarizaram 879 estudos.

Figura 1.2 apresenta os estudos obtidos por fontes digitais da literatura. Foi aplicado a *string* de busca no Google Scholar que indexou o texto completo da literatura entre revistas, artigos científicos, relatórios técnicos, teses e livros. Google Scholar retornou 360 contribuições. Com a finalidade de se concentrar nos estudos mais relevantes, restringiu-se a pesquisa para os primeiros 200 documentos que representaram (29%) dos estudos recuperados. Scopus e Ei Compendex retornaram aproximadamente a mesma quantia de estudos, 126 (19%) e 127 (19%) respectivamente. IEEE Xplore, outro importante

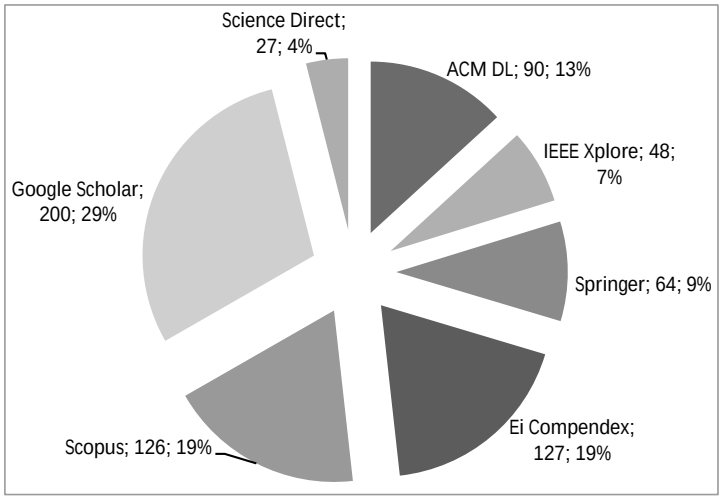


Figura 1.2: Estudos obtidos por motores de busca de dados eletrônicos

mecanismo de pesquisa, apresentou um total de 48 estudos (13%). Springer apresentou 64 (9%), ACM DL retornou 90 (13%) e Science Direct 27 (4%) dos estudos.

Os estudos obtidos por pesquisa manual foram adquiridos de 15 conferências que endereçavam estudos em LPS, GV entre outras importantes áreas da engenharia de software, como apresentado na Figura 1.3. A maioria dos estudos foi adquirida da conferência SPLC, seguida das conferências ICSE e VaMoS.

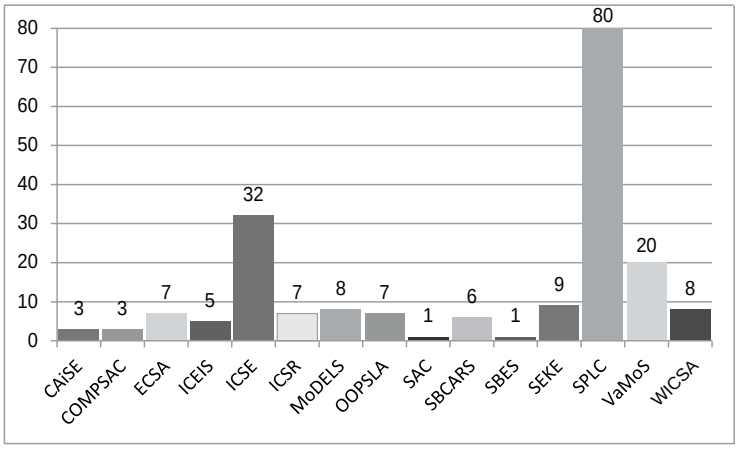


Figura 1.3: Distribuição dos estudos em diferentes locais de publicação

No primeiro estágio do processo do MS, excluímos estudos duplicados (272) e estudos que eram capas de apresentação de eventos (30). Finalmente, 577 estudos foram recuperados para a aplicação dos critérios de inclusão e exclusão. Os estágios e resultados do processo de MS estão apresentados na Figura 1.4.

Figura 1.4: Estágios e resultados do processo de seleção de estudos.

A.1.5 Critérios de Inclusão e Exclusão

Durante essa etapa, a seleção dos estudos primários foi baseada em critérios de inclusão e exclusão previamente definidos, como segue:

Critérios de Inclusão:

a) Estudos sobre ferramentas de variabilidades de software;

b) Estudos sobre avaliação de ferramentas de variabilidades de software;

c) Estudos sobre tecnologias para a implementação de ferramentas de variabilidades de software; e

d) Estudos relacionados a características das ferramentas de variabilidades de software.

Critérios de Exclusão:

a) Estudos que não abordam ferramentas de variabilidades de software;

b) Estudos relacionados a ferramentas de variabilidades de software, mas sem descrição das funcionalidades ou documentação disponível, pois não fornecem meios para a análise da ferramenta; e

c) Documentos com menos de três páginas.

A seleção dos estudos primários foi inicialmente baseada na revisão do titulo e resumo; embora, essa análise se estendeu para a leitura da seção de conclusão nos casos onde o título e resumo forneciam informações insuficientes. Depois de realizar tal revisão, 512 estudos que não abordavam ferramentas de variabilidades de software foram excluídos, e **65 estudos potenciais foram selecionados**.

Uma vez que o conjunto inicial de documentos foi identificado, decidiu-se aplicar a técnica de *snowballing* fazendo uso dos métodos *backward* e *forward* (Wohlin, 2014). *Backward snowballing* significa utilizar a lista de referência para identificar novos estudos para serem incluídos. Deste modo, na primeira iteração de *backward snowballing*, foram

encontrados 69 estudos sendo 44 que já foram incluídos no conjunto inicial de estudos, e 25 novos estudos que foram adicionados a lista de estudos a serem incluídos. Na primeira iteração de *forward snowballing*, os estudos citando o conjunto de estudos foram avaliados. A análise de tais citações foi conduzida utilizando o Google Scholar. Consequentemente, foram identificados 34 estudos sendo 29 já incluídos no conjunto inicial de estudos, e cinco novos estudos que foram incluídos a listas de estudos. Resumindo, 30 novos estudos potenciais foram encontrados na primeira iteração do *snowballing*.

Como o objetivo desse MS é reunir todas as ferramentas de variabilidades de software que suportam a maior parte do processo de GV, foi realizado uma segunda iteração de *snowballing* com os novos estudos identificados na primeira iteração. Assim, o *backward snowballing* foi conduzido para esses 30 novos estudos, e então um *forward snowballing* foi realizado também. Em resumo, na segunda iteração do *backward snowballing*, foram encontrados 17 estudos que já estavam incluídos no conjunto inicial de estudos e nenhum novo estudo foi encontrado. No segundo *forward snowballing*, 29 estudos já examinados foram identificados e nenhum novo estudo foi encontrado. Portanto, incluímos **30 novos estudos** utilizando a técnica de *snowballing*.

Todos os potenciais estudos e possíveis ferramentas de variabilidades de software podem ser visualizados na Tabela 1.3. Em resumo, tal tabela apresenta **79 possíveis ferramentas** entre **95 potenciais estudos** incluindo aquelas ferramentas encontradas pela técnica *snowballing* e dois importantes estudos secundários de Lisboa et al., 2010 e Pereira et al., 2015 que cobrem ferramentas de LPS. Além disso, é possível observar os tipos de publicação e onde foi publicado.

Tabela 1.3: Possíveis ferramentas de variabilidades de software

♯ Possíveis Ferramentas	Estudos Relacionados	Publicação	Local	Snow ball
1 - 001	Hamilton	Online	Hamilton Tech.	Sim
2 - ABS	Wong et al.	Revista	STTT	Não
3 - AHEAD tool	Batory et al.	Revista	TSE	Sim
4 - Aora	Varela et al.	Conferência	SAC	Sim
5 - Asadal	Kim et al.	Conferência	ICSE	Não
6 - BVR	Vasilevskiy et al.	Conferência	SPLC	Não
7 - CaptainFeature	Bednasch et al.	Online	Kaiserslautern	Sim
8 - CardyGAn	Schnabel et al.	Conferência	VaMoS	Não
9 - CIDE	Kästner et al.	Conferência	ICSE	Sim
10 - Clafer Tools	Antkiewicz et al.	Conferência	SPLC	Não
11 - Consul	Lamprecht et al.	Revista	IEEE	Não
12 - Covamof-VS	Deelstra et al.; Sinnema e Deelstra	Revista, Revista	IST, JSS	Não
13 - CVL Tool	Saratxaga et al.; Svendsen et al.	Conferência, Conferência	PLEASE, SPLC	Não
14 - CVM	Abele et al.	Conferência	VaMoS	Sim
15 - DARE	Frakes et al.	Conferência	SCCC	Sim
16 - Decimal	Dehlinger et al.	Conferência	ICSE	Sim
17 - DecisionKing	Dhungana et al.	Conferência	VaMoS	Sim
18 - Domain	Tracz	Revista	SIGSOFT	Sim
19 - Doors	Thurimella et al.	Revista	IEEE	Não
20 - Dopler	Lettner et al.;Dhungana et al.	Conferência, Revista	SPLC, ASE	Não
21 - DPLfw	Gómez et al.	Conferência	SPLC	Não
22 - Dream	Park et al.	Conferência	ISFST	Sim
23 - Easy-Producer	Eichelberger e Schmid	Conferência	SPLC	Não
24 - EPM	Abele et al.	Conferência	SPLC	Não
25 - ESCAPE	Frank e Brenner	Conferência	ICCGI	Não
26 - Fama	Roos-Frantz et al.	Conferência	SPLC	Não
27 - FeatureHouse	Apel et al.	Conferece	ICSE	Sim
28 - FeatureIDE	Khan et al.	Revista	IJSER	Não
29 - FeatureMapper	Heidenreich et al.	Conferência	ICSE	Sim
30 - Flip	Soares et al.	Conferência	OOPSLA	Não
31 - fmp	Czarnecki et al.	Conferência	OOPSLA	Não
32 - FMT	Laguna e Hernández	Conferência	ICEBE	Não
33 - Fusion	Weber et al.	Revista	IEEE	Não
34 - Gears	Krueger; Krueger; Krueger e Clements	Conferência, Conferência, Conferência	OOPSLA, SPLC, SPLC	Não
35 - Genarch	Cirilo et al.; Cirilo et al.	Conferência, Revista	SBCARS, JSS	Não
36 - Hephaestus	Turnes et al.	Conferência	SBCARS	Não
37 - Holmes	Succi et al.; Succi et al.	Conferência, Conferência	ICSE, ICSE	Não
38 - Hydra	Gamez e Fuentes	Livro	Springer	Sim
39 - Invar	Dhungana et al.; Galindo et al.	Conferência, Revista	VaMoS, IST	Não
40 - ISMT4SPL	Park et al.	Conferência	ICIS	Não
41 - Kumbang Tools	Myllärniemi et al.	Conferência	SPLC	Não
42 - Lisa Toolkit	Groher e Weinreich; Groher e Weinreich	Conferência, Conferência	ECSA, HICSS	Não
43 - Matelo	Samih e Bogusch	Conferência	SPLC	Não
44 - Metadoc	Thurimella e Janzen	Conferência	SPLC	Não
45 - MetaEdit+	Tolvanen	Conferência	SPLC	Não
46 - Moso-Polite	Oster et al.	Conferência	VaMoS	Não
47 - MoSPL	Thao et al.	Conferência	ECBS	Sim
48 - Musa	Bashroush et al.; Bashroush	Conferência, Conferência	CAiSE, ECSA	Não
49 - Odyssey	Braga et al.	Conferência	ASSET	Sim
50 - Pacogen	Marijan et al.; Hervieu et al.	Conferência, Conferência	SPLC, ISSRE	Não
51 - PlugSPL	Rodrigues et al.	Conferência	SEKE	Sim
52 - Plum	López e Mansell	Livro	Springer	Não
53 - Pluss Toolkit	Eriksson et al.	Conferência	ASE	Sim
54 - PuLSE-BEAT	Schmid e Schank	Conferência	ISAPF	Sim
55 - pure::variants	Beuche e Hellebrand; Beuche et al.; Beuche	Conferência, Revista, Conferência	SPLC, SCP, SPLC	Não
56 - Remap	Schmid et al.	Conferência	SPLC	Não
57 - Requiline	Maßen e Lichter	Conferência	PFE	Não
58 - s2t2	Pleuss e Botterweck	Revista	STTT	Não
59 - SNIP	Classen et al.	Revista	STTT	Sim
60 - SoaSPL	Abu-Matar e Gomaa	Conferência	SOSE	Não
61 - SPL Config	Machado et al.	Conferência	CBSoft	Sim
62 - SPLOT	Mendonça et al.	Conferência	OOPSLA	Não
63 - SPL-TuPI	Matos et al.	Online	IFBA	Sim
64 - SPLVerifier	Apel et al.	Conferência	ASE	Sim
65 - SSEP toolset	Stuart et al.	Conferência	SPLC	Sim
66 - Sysiphus-IVVM	Thurimella e Bruegge	Revista	IST	Não
67 - Toolday	Lisboa et al.	Revista	STTT	Não
68 - Variamos	Mazo et al.	Conferência	SPLC	Não
69 - Varies	Wagner et al.	Conferência	SPLC	Não
70 - Varmod	Duisburg-Essen	Online	Duisburg	Sim
71 - Visit-FC	Botterweck et al.; Nestor et al.	Conferência	COMPSAC	Não
72 - V-Menage	van der Hoek	Revista	SCP	Não
73 - VMWT	Capilla et al.	Conferência	SVM-WS	Sim
74 - Vulcan	Lee et al.	Conferência	SoftVis	Não
75 - WebFML	Bécan et al.	Conferência	SPLC	Sim
76 - WeCoTin	Asikainen et al.	Conferência	SPLC	Sim
77 - XFeature	Pasetti e Rohlik	Online	PnP Software	Sim
78 - Xtof	Gauthier et al.	Conferência	VaMoS	Não
79 - Yam	Jain e Biesiadecki	Conferência	SMC-IT	Sim

A.1.6 Estudos Selecionados

Uma vez obtidos os estudos primários considerados relevantes para os objetivos desta pesquisa, foram revisados cuidadosamente cada artigo para filtrar e selecionar somente aqueles estudos relevantes para responder as questões de pesquisa. Em particular, teve-se como objetivo identificar se os estudos descreviam características técnicas sobre ferramentas de variabilidades de software e se as ferramentas foram avaliadas na indústria ou academia.

Como resultado da revisão manual dos 79 possíveis ferramentas, excluímos quatro ferramentas: *AHEAD* (Algebraic Hierarchical Equations for Application Development) (Batory et al., 2004), *CIDE ColoredIDE* (Kästner et al., 2008), *FeatureHouse Language-Independent* (Apel et al., 2009) e SPLConfig (Machado et al., 2014), pois elas estão integradas ou utilizadas em combinação com FeatureIDE, uma ferramenta que foi considerada neste estudo. Além do mais, nota-se que 46 possíveis ferramentas de variabilidades de software não cobriam a maioria das atividades de GV sugeridas por Pohl et al., (2005). Tais ferramentas foram excluídas da fase de análise e são listados no material suplementar da Web[8] em conjunto com uma breve descrição. Finalmente, foram selecionados e analisados **29** ferramentas de variabilidades de software. Tabela 1.4 apresenta tais ferramentas e uma classificação geral incluindo ano de publicação, se a ferramenta foi desenvolvida pela indústria ou academia e licença de software.

A.2 Análise dos Resultados

As seguintes seções descrevem os resultados detalhados do MS utilizando as ferramentas selecionadas, e cada questão de pesquisa é respondida separadamente. Seção A.2.1 apresenta uma classificação para os tipos de ferramentas de variabilidades de software e em qual configuração foram avaliadas; Seção A.2.2 apresenta as notações de modelagem utilizadas pelas ferramentas; e, finalmente, Seção A.2.3 descreve as tecnologias subjacentes que apoiam tais ferramentas.

A.2.1 Classificação dos Tipos de Ferramentas de Variabilidades de Software

A primeira RQ refere-se à classificação dos tipos de ferramentas de variabilidades de software e responde a questão de pesquisa **RQ1** (*Quais tipos de ferramentas de variabilidades de software tem sido desenvolvidas e utilizadas?*). Essa classificação foi focada nas ferramentas que reuniam a maioria das atividades de GV, tal classificação está apresentada na Tabela 1.5 e descrita abaixo:

[8]http://www.din.uem.br/ edson/svtools/SMS.html

Tabela 1.4: Ferramentas de variabilidades de software selecionadas

♯	**Ferramenta**	**Ano de Publicação**	**Instituição**	**Licença de Software**
T1	Captain Feature	2002	Academia	*Open source*
T2	Clafer Tools	2013	Academia	*Open source*
T3	COVAMOF-VS	2004	Academia	Pesquisa Acadêmica
T4	CVL Tool	2009	Academia	*Open source*
T5	CVM	2009	Academia	*Open source*
T6	DecisionKing	2007	Ambos	Protótipo de Pesquisa
T7	DOPLER	2007	Ambos	Protótipo de Pesquisa
T8	FAMA	2007	Ambos	*Open source*
T9	FeatureIDE	2005	Academia	*Open source*
T10	FMP	2004	Academia	*Open source*
T11	FMT	2009	Academia	*Open source*
T12	GEARS	2002	Indústria	Comercial
T13	GENARCH	2007	Academia	Pesquisa Acadêmica
T14	Hephaestus	2009	Academia	*Open source*
T15	Hydra	2009	Academia	*Open source*
T16	Kumbang tools	2007	Ambos	*Open source*
T17	LISA toolkit	2012	Academia	Protótipo de Pesquisa
T18	Metadoc	1998	Indústria	Comercial
T19	PLUM	2008	Ambos	Comercial
T20	pure variants	2004	Indústria	Comercial
T21	s2t2	2009	Academia	*Open source*
T22	SOASPL	2013	Academia	Protótipo de Pesquisa
T23	SPLOT	2009	Academia	*Open source*
T24	Variamos	2012	Academia	*Open source*
T25	Visit-FC	2008	Academia	Protótipo de Pesquisa
T26	V-Menage	2006	Ambos	Pesquisa da Indústria
T27	VMWT	2007	Academia	Protótipo de Pesquisa
T28	WeCoTin	2000	Academia	Protótipo de Pesquisa
T29	XFEATURE	2005	Ambos	*Open source*

1) **Ferramentas de GV** contém a identificação e representação da variabilidade, como o modelo de variabilidade é representado, selecionado e instanciado, e como ele pode evoluir alterando as variantes e pontos de variação.

2) **Ferramenta de Verificação de Restrição da Variabilidade** é utilizada para automatizar a análise dos modelos de *features* e determinar se um modelo de *feature* é válido.

3) **Ferramenta de Modelagem de Variabilidades** engloba ferramentas voltadas para a modelagem, representação e visualização de modelos de variabilidades.

4) **Ferramenta de Derivação de Produtos** envolve o processo de construção de produtos por meio de artefatos de uma LPS, parcialmente através da exploração de pontos de variação e variantes.

5) **Ferramentas de Configuração das Variabilidades** facilita a seleção e realização das variantes e é limitada por variabilidades ou restrições dos produtos.

Tabela 1.5: Tipo da ferramentas de variabilidades de software

♯	**Ferramenta**	**Tipo da Ferramenta**
T1	Captain Feature	Gerenciamento de Variabilidade
T2	Clafer Tools	Verificação de Restrição das Variabilidades
T3	COVAMOF-VS	Derivação de Produto
T4	CVL Tool	Gerenciamento de Variabilidade
T5	CVM	Gerenciamento de Variabilidade
T6	DecisionKing	Derivação de Produto
T7	DOPLER	Derivação de Produto
T8	FAMA	Verificação de Restrição das Variabilidades
T9	FeatureIDE	Gerenciamento de Variabilidade
T10	FMP	Verificação de Restrição das Variabilidades
T11	FMT	Gerenciamento de Variabilidade
T12	GEARS	Gerenciamento de Variabilidade
T13	GENARCH	Derivação de Produto
T14	Hephaestus	Derivação de Produto
T15	Hydra	Verificação de Restrição das Variabilidades
T16	Kumbang tools	Gerenciamento de Variabilidade
T17	LISA toolkit	Gerenciamento de Variabilidade
T18	Metadoc	Modelagem das Variabilidades
T19	PLUM	Gerenciamento de Variabilidade
T20	pure variants	Gerenciamento de Variabilidade
T21	s2t2	Verificação de Restrição das Variabilidades
T22	SOASPL	Gerenciamento de Variabilidade
T23	SPLOT	Verificação de Restrição das Variabilidades
T24	Variamos	Verificação de Restrição das Variabilidades
T25	Visit-FC	Configuração das Variabilidades
T26	V-Menage	Gerenciamento de Variabilidade
T27	VMWT	Gerenciamento de Variabilidade
T28	WeCoTin	Configuração das Variabilidades
T29	XFEATURE	Gerenciamento de Variabilidade

Figura 1.5 descreve a distribuição das ferramentas por ano e por tipo de ferramenta. Os resultados apresentaram que **14** ferramentas **(48.3%)** são ferramentas de GV, **7** ferramentas **(24.1%)** são ferramentas de verificação de restrição das variabilidades, **5**

ferramentas **(17.2%)** são ferramentas de derivação de produtos, **2** ferramentas **(6.9%)** são ferramentas de configuração das variabilidades, e **1** ferramenta **(3.5%)** é de modelagem de variabilidades.

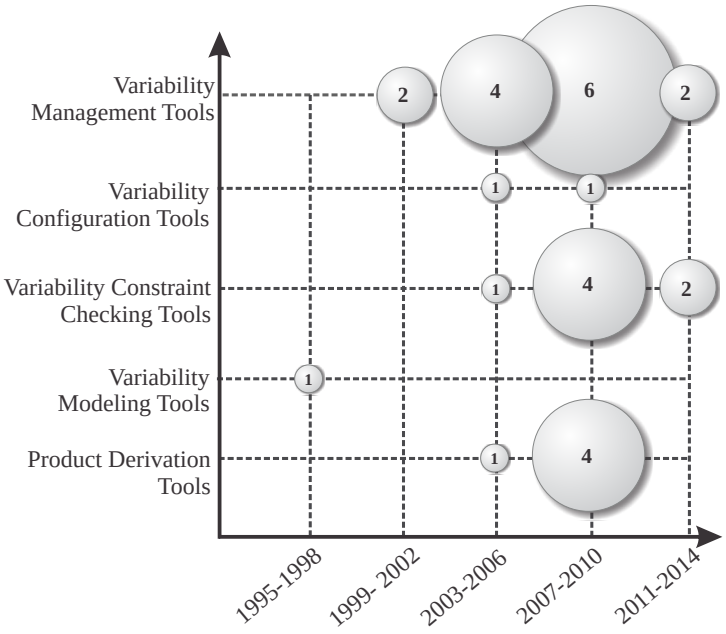


Figura 1.5: Distribuição das ferramentas por ano relacionado aos tipos

Vale ressaltar que a classificação de (2) a (5) estão integradas no GV. Por essa razão, algumas ferramentas de GV são conhecidas como ferramentas de modelagem ou derivação de produtos, e ferramentas de verificação de consistência são também conhecidas por serem ferramentas de modelagem ou de configuração das variabilidades. A intenção da classificação apresentada anteriormente é envolver as principais atividades de tais ferramentas, incluindo a atividade de GV.

Com relação à disponibilidade das ferramentas, descobri-se que 23 ferramentas possuem página web disponíveis, e somente **18** ferramentas estão disponíveis para download, como apresentado na Tabela 1.6. As Ferramentas **T12, T18, T19** e **T20** são comercias, enquanto que a ferramenta **T20** tem uma versão gratuita para comunidade.

Avaliação das Ferramentas

Porque é interessante saber se as ferramentas selecionadas foram ou estão atualmente em uso na indústria, foram analisados os trabalhos de pesquisa e outras documentações complementares e web sites de LPS conhecido por Hall of Fame (HoF)[9] para encontrar evidências de uso destas ferramentas em ambientes industriais. Como resultado, encontrou-se que **48%** destas ferramentas (14 ferramentas) foram reportadas por terem sido utilizadas na indústria ou por estudos de caso na indústria, como apresentado na

[9]http://splc.net/fame.html

Tabela 1.6: Ferramentas disponíveis para download

Ferramenta	T	Download	Website
Covamof-VS	T3		Web page não encontrada
Hephaestus	T14		Web page não encontrada
SOASPL	T22		Web page não encontrada
Visit-FC	T25		Web page não encontrada
V-Manage	T26		Web page não encontrada
VMWT	T27		Web page não encontrada
CaptainFeature	T1	Sim	https://sourceforge.net/projects/captainfeature
Clafer Tools	T2	Sim	http://www.clafer.org
CVL Tool	T4	Sim	http://www.omgwiki.org/variability/doku.php
CVM	T5	Sim	http://www.cvm-framework.org/
DecisionKing	T6	Não	http://ase.jku.at/modules/product-lines/
Dopler	T7	Não	http://ase.jku.at/modules/product-lines/
Fama	T8	Sim	http://www.isa.us.es/fama/
FeatureIDE	T9	Sim	http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/
FMP	T10	Sim	http://gp.uwaterloo.ca/fmp
FMT	T11	Sim	http://www.giro.infor.uva.es/FeatureTool.html
Gears	T12	Sim	http://www.biglever.com/
GenArch	T13	Não	http://twiki.cin.ufpe.br/twiki/bin/view/ProjetoProcad/GenArch-P
Hydra	T15	Sim	http://caosd.lcc.uma.es/spl/hydra/
Kumbang Tools	T16	Sim	http://www.soberit.hut.fi/KumbangTools/
Lisa	T17	Sim	http://lisa.se.jku.at/
Metadoc	T18	Sim	http://www.metadoc.de
PLUM	T19	Sim	http://www.esi.es/plum/index.php
pure::Variants	T20	Sim	http://www.pure-systems.com
s2t2	T21	Sim	http://download.lero.ie/spl/s2t2/index.html
SPLOT	T23	Não	http://www.splot-research.org/
Variamos	T24	Sim	http://variamos.com/home/
WeCoTin	T28	Não	http://www.soberit.hut.fi/WeCoTin/
XFeature	T29	Sim	http://www.pnp-software.com/XFeature/

Tabela 1.7, porém não foi encontrado informações de avaliações pela indústria com relação a 15 ferramentas. Mais informações sobre as ferramentas avaliadas pela indústria estão a seguir.

A pesquisa de Berger et al., (2013) recebeu respostas de 42 países de uma ampla variedade de domínios de aplicação, tais como: automotivo, energia, empreendimento, comércio eletrônico, aeroespacial e de defesa, medico, eletrônico, governo, telecomunicação e outros. Não foi possível identificar os nomes das empresas que responderam a pesquisa. Em tal pesquisa, os autores identificaram uma variedade de ferramentas utilizadas pela indústria, pure::variants (T20) da empresa pure::systems foi a ferramenta mais frequentemente selecionadas entre os participantes (35% respostas), seguida da ferramenta Gears (T12) da empresa BigLever Software (23%). Pouco menos de um terço dos participantes usaram uma ferramenta *open source*, tal como a ferramenta CVL (T4), CVM (T5), Hephaestus (T14), Dopler (T7), FeatureIDE (T9) e XFeature (T29). 27% dos entrevistados utilizaram uma ferramenta comercial diferente, tal como a ferramenta PLUM (T19) baseada em modelo de decisão e desenvolvida pela empresa Tecnalia. Em resumo, os pesquisadores observaram uma elevada variedade de ferramentas, porém ferramentas baseadas em modelos de *features* são claramente dominantes.

Tabela 1.7: Ferramentas avaliadas pela indústria

Avaliado por	**Ferramenta**	**Total**	%
Utilizada pela Indústria	T1, T4, T5, T7, T9, T12, T14, T18, T19, T20, T29	11	38
Estudo de Caso pela Indústria	T3, T6, T16	3	10

Além disso, Berger et al., (2014) conduziu outra pesquisa na indústria com três estudos de caso de ferramentas de modelagem de variabilidades baseada em *features* e encontrou mais duas ferramentas de variabilidades de software utilizadas pela indústria, CaptainFeature (T1) e pure::variants (T20). Em tal estudo, CaptainFeature foi utilizado em uma empresa de consultoria com menos de 50 empregados. Tal empresa entrega aplicações baseada na web, comercio eletrônico e aplicações empresariais personalizadas, porém não foi possível identificar o nome da empresa. Pure::variants foi utilizado em uma grande empresa de componentes com ao menos 25 mil empregados. Tal empresa é fornecedora de componentes eletrônicos e mecânicos para usuários finais e aplicações industriais. As experiências relatadas mostraram que os modelos de *features* são considerados intuitivos e simples, e em vez de declarar e manter restrições, os participantes preferem gerenciar um conjunto de configurações ou deixar que os especialistas configurem os produtos.

Outros estudos reportaram o uso de ferramentas de variabilidades na indústria. Por exemplo, a ferramenta DecisionKing (T6) foi utilizada em um estudo de caso com um parceiro industrial da BMD[10], uma empresa de médio porte que oferece produtos de software empresariais para 18.400 clientes e 45.000 usuários ativos em diferentes países europeus (Rabiser et al., 2009).

A ferramenta Kumbang (T16) foi utilizada em um estudo de caso em colaboração com Robert Bosch GmbH, uma empresa que desenvolve sistemas automotivos embarcados e validados por um sistema periférico de carros (Myllärniemi et al., 2007). A ferramenta Metadoc Feature Model (T18) foi avaliada por uma empresa de eletrodomésticos de médio porte que precisavam de uma solução para gerenciar o crescente número de variantes em seus sistemas (Thurimella e Janzen, 2011). A ferramenta Covamof-VS (T3) foi avaliada por uma empresa Holandesa que desenvolve módulos de software de propriedade intelectual para sistemas de tráfego inteligentes (ITS) utilizando família de produtos conhecida por Intrada (Deelstra et al., 2009). A base de ativos reutilizáveis dessa família de produtos consiste de aproximadamente 11 milhões de linha de código fonte (Deelstra et al., 2009; Sinnema e Deelstra, 2008).

Embora quase 50% das ferramentas de variabilidades de software foram avaliadas pela indústria, observa-se a falta de informações mais completas sobre as práticas industriais. Também descobriu-se que não está claro como essas ferramentas foram adotadas pela indústria. Acredita-se que mais relatos de experiência industrial e estudos de caso devem

[10]www.bmd.com

ser realizados para reunir provas sobre o uso de tais ferramentas. As avaliações por parte da indústria podem ser usadas como uma indicação de maturidade, bem como para estimar o risco potencial da adoção de uma determinada ferramenta de variabilidades de software.

A.2.2 Notações Utilizadas nas Ferramentas de Variabilidades de Software

Modelar semelhanças e variabilidades de sistemas é uma importante atividade quando se utiliza a técnica de LPS. Nos últimos anos, várias técnicas de modelagem da variabilidade têm sido desenvolvidas, com objetivo de representar explicitamente e eficazmente a variabilidade em LPS, como pode ser visto na RSL realizada por Chen et al., (2009).

Modelagem de *features* é a técnica mais utilizada para capturar e gerenciar as semelhanças e variabilidades de LPS, mas pode-se encontrar extensões e notações complementares desenvolvidas ao longo dos últimos anos. FODA (Kang et al., 1990) foi a primeira notação criada em 1990 para gerenciar variabilidades. Esta notação centra-se na engenharia de requisitos orientada a *features* e foi estendida pelo Método de Reúso Orientado a *Feature* (FORM) (Kang et al., 1998) para apoiar o GV nas fases de projeto e implementação. Depois disso, muitas outras notações foram baseadas na modelagem de *feature* ou suas extensões (Bosch et al., 2015; Chen et al., 2009). Algumas das mais importantes extensões de modelagem de *features* são as seguintes.

O Modelo de *Feature* Baseado em Cardinalidade (CBFM) (Czarnecki et al., 2005b) aborda os desafios da engenharia de família de produtos com modelagem de variabilidades em termos de seleção de *features*. Nesta técnica, *Features* denotam qualquer característica funcional ou não funcional nos artefatos de família de produtos. Para permitir abstração, as *features* são hierarquicamente organizadas em uma árvore. Cada *feature* pode ter um atributo que representa uma propriedade numérica ou textual. *Orthogonal Variability Management* (OVM) é um modelo que define variabilidades de uma LPS. Relaciona variabilidades definidas a outros modelos de desenvolvimento de software, tal como modelos de *feature*, modelos de caso de uso, modelos de componentes e modelos de testes (Pohl et al., 2005). *Common Variability Language* (CVL), do consorcio OMG[11], é uma linguagem independente do domínio para especificar e resolver variabilidades (Svendsen et al., 2010).

A Figura 1.6 apresenta uma cronologia da evolução da notação FODA e suas extensões. As notações foram alocadas em diferentes períodos de anos com base na data de publicação da notação. Essa classificação indica um grande esforço para construir notações entre 2000 e 2005. Estes períodos de tempo abrangem o maior número de notações de GV desenvolvidas.

[11]http://www.omg.org

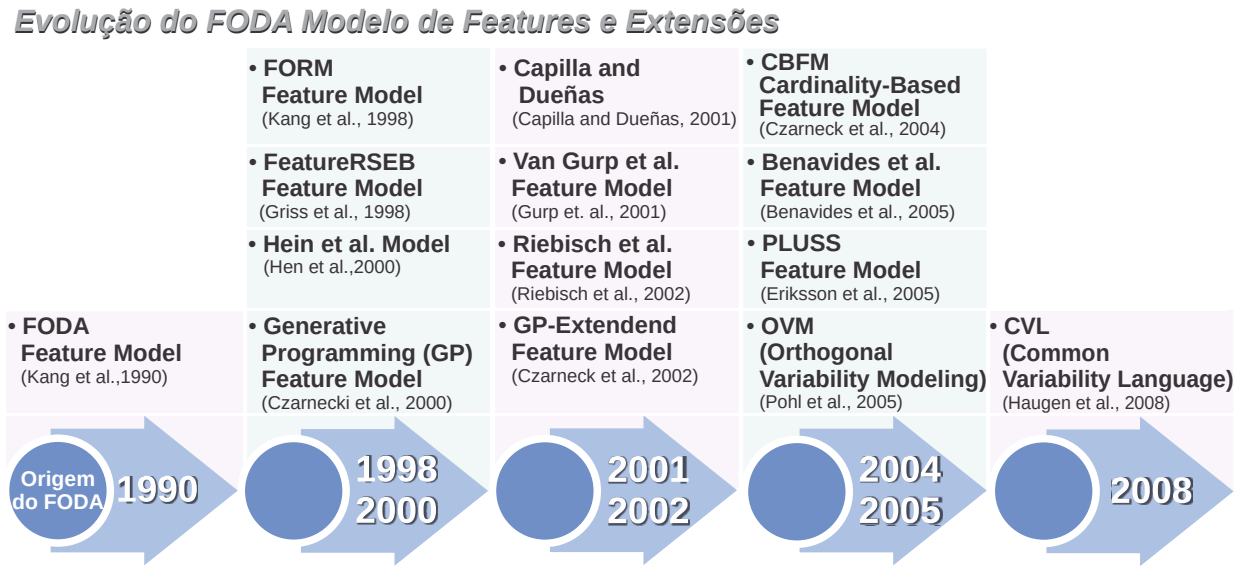


Figura 1.6: Cronologia da notação FODA e suas extensões

Em resposta à questão de pesquisa **RQ1.1** *(Quais notações de modelagem de feature e extensões são suportadas pelas ferramentas de variabilidades de software?)*, foram apresentadas na Tabela 1.8 a abordagem de modelagem de *feature* utilizada pelas ferramentas. CBFM é a notação mais utilizada nas ferramentas de variabilidades de software. Ela aparece em **9** ferramentas **(31%)**; além disso, **7** ferramentas **(24.1%)** utilizam notação FODA; **2** ferramentas **(6,9%)** utilizam OVM e **1** ferramenta **(3,4%)** utiliza notação CVL.

As **10** ferramentas restantes **(34,5%)** não utilizam abordagem de modelagem de *feature* e não foram listadas na tabela anterior. No entanto, foi realizada uma breve descrição sobre elas e suas notações como seguem.

A ferramenta Clafer (T2) utiliza a notação Clafer (*class feature reference*), que é uma linguagem de modelagem estrutural com sintaxe e semântica minimalista equivalente a lógica relacional de primeira ordem (Antkiewicz et al., 2013). As ferramentas DecisionKing (T6), Dopler (T7), PLUM (T19) e V-Manage (T26) utilizam abordagens orientada a decisão para representar variabilidades em LPS. Tal abordagem basicamente utiliza uma tabela onde cada linha representa uma decisão e cada coluna uma propriedade de uma decisão (Jézéquel, 2012). A ferramenta Kumbang (T16) utiliza uma ontologia para representar as variabilidades de família de produtos de software. Essa ontologia inclui conceitos para modelar variabilidades, tanto do ponto de vista de *feature* como do ponto de vista arquitetural, incluindo o inter-relacionamento entre eles (Myllärniemi et al., 2007). Finalmente, não foi possível classificar a notação de variabilidades adotada ou utilizada pelas ferramentas T12, T22, T24, e T28.

Abordagens de modelagem de variabilidades são essenciais para apoiar o GV em diferentes fases de uma LPS. A maioria delas enfatiza melhorias da notação, novos tipos de *features*, cardinalidades, atributos de *features*, e as relações estendidas para definir

Tabela 1.8: Ferramentas classificadas por notações do modelo de *feature*

Notações da abordagem de Modelo de Feature	♯	**Ferramenta**
FODA	T1	Captain Feature
	T9	FeatureIDE
	T11	FMT
	T15	Hydra
	T18	Metadoc
	T20	pure variants
	T27	VMWT
CBFM	T5	CVM
	T8	FAMA
	T10	FMP
	T13	GENARCH
	T14	Hephaestus
	T21	s2t2
	T23	SPLOT
	T25	Visit-FC
	T29	XFEATURE
OVM	T3	Covamof-VS
	T17	Lisa Toolkit
CVL	T4	CVL Tool

com mais precisão limitações e relações entre *features* (Bosch et al., 2015). Em nosso MS, **65.5%** das ferramentas de variabilidades de software são baseadas em modelos de *feature* através da notação FODA e suas extensões. A escolha por notações de abordagens de modelos de *features* pode ser explicada devido ao grande número de extensões existentes e pelos benefícios gerados através das capacidades de visualização envolvendo a identificação das variabilidades na análise do domínio.

Figura 1.7 apresenta uma visão geral de todas as ferramentas identificadas nesta pesquisa. As ferramentas foram colocadas em diferentes períodos de anos, com base na data de publicação, mas o momento em que a ferramenta foi desenvolvida pode ser diferente. Cada ferramenta está organizada pelo nome, licença de software, tipo da ferramenta de variabilidades de software, desenvolvedor, ano de desenvolvimento e abordagem das variabilidades.

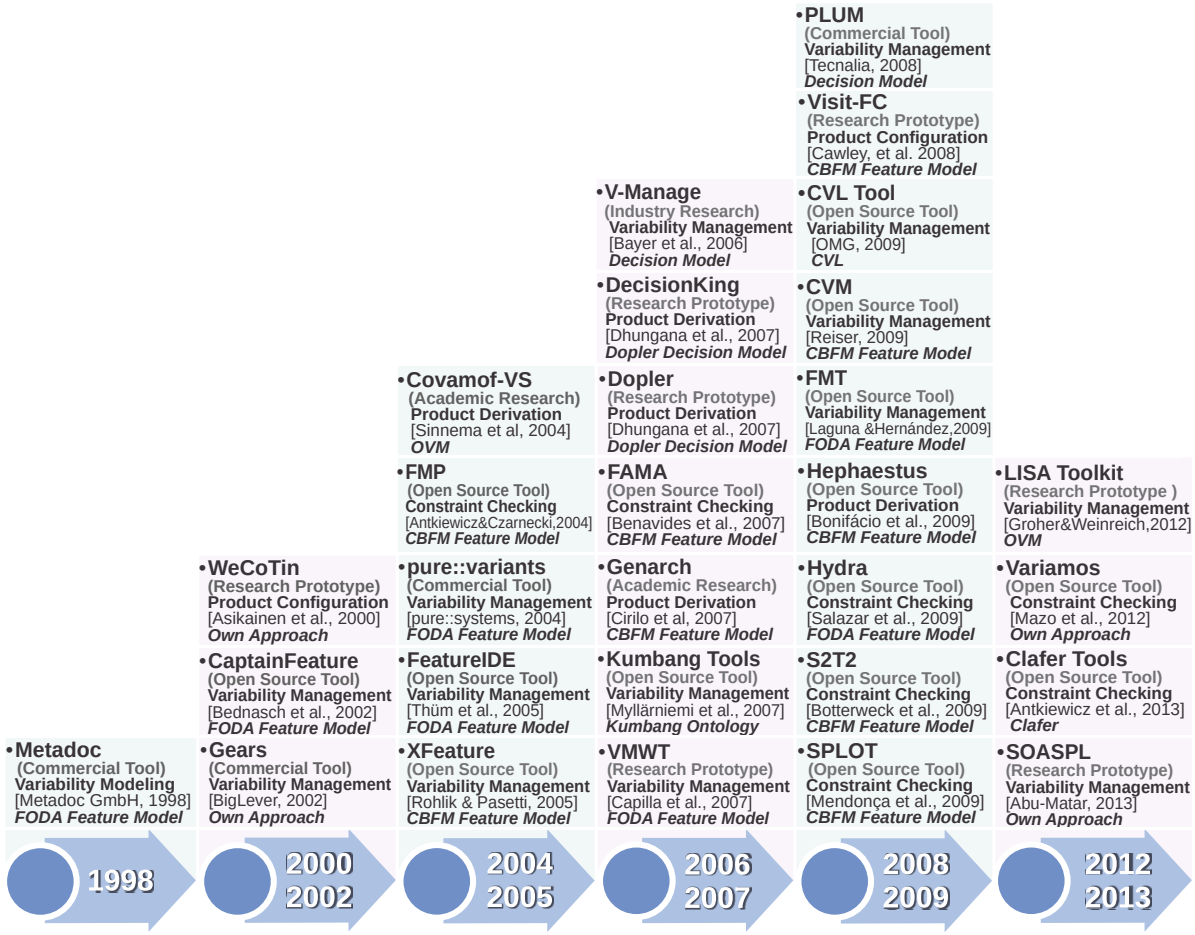


Figura 1.7: Ferramentas de variabilidades de software open source e comerciais

A.2.3 Tecnologias Utilizadas nas Ferramentas de Variabilidades de Software

A maioria das ferramentas de variabilidades de software integra uma variedade de tecnologias. Em resposta à questão de pesquisa **RQ2** (*Quais são as tecnologias fundamentais que apoiam as ferramentas de variabilidades de software?*), observa-se que **19** ferramentas, isto é, **65.5%** foram construídas como plugins. Tais ferramentas são T3, T4, T5, T6, T7, T8, T9, T10, T11, T13, T15, T17, T18, T19, T20, T21, T22, T24, e T29. Um plugin possibilita a integração com outras funcionalidades específicas (Dhungana et al., 2007); além disso, plugins representam unidades de funcionalidades que podem ser desenvolvidas e implementadas de forma isolada (Dhungana et al., 2011). A integração de modelos de *feature* através de plugins é facilitada com o uso da plataforma EclipseIDE, que apoia de forma otimizada as variabilidades e modelagem em diferentes artefatos (Czarnecki et al., 2005a).

Foram analisadas as linguagens de programação em que as ferramentas foram construídas, como apresentado na Tabela 1.9. Pode-se observar que **23** ferramentas **(79.3%)** foram desenvolvidas em Java, enquanto **2** ferramentas **(6.9%)** foram implementadas em C♯, **1** ferramenta **(3.4%)** em Haskel, e **1** ferramenta **(3.4%)** em PHP. Uma das razões para utilização da linguagem de programação Java para implementar tais ferramentas de variabilidades de software está na possibilidade de aumentar a reutilização de componentes entre diferentes ambientes através do uso da plataforma EclipseIDE. Assim, com um conjunto adequado de plugins, Eclipse pode ser utilizado como uma poderosa plataforma IDE transversal para multiplos tipos de desenvolvimento, tais como programação, modelagem visual, ou edição de textos (Segura et al., 2007). Além do mais, a linguagem de programação Java envolve importantes características, tais como, execução condicional, parâmetros de funções e constantes, que podem ajudar a resolver variabilidades em tempo de compilação e tempo de execução (Beuche et al., 2004). Portanto, concluímos que Java é a linguagem preferida para apoiar novas extensões utilizando tecnologias de plugins para o desenvolvimento de ferramentas de variabilidades de software. Outras características relevantes observadas estão descritas abaixo.

Tabela 1.9: Linguagem de programação

Linguagem	**Ferramentas**	**Total**	**%**
Java	T1, T4-T10, T12, T13, T15-T25, T28, T29	23	79.3
C♯	T3, T11	2	6.9
Haskel	T14	1	3.4
PHP	T27	1	3.4
N/A	T2, T26	2	6.9

Visualização das Variabilidades

A visualização de grandes modelos de variabilidades é uma preocupação e um desafio para a indústria. Assim, a fim de melhorar a usabilidade da capacidade de visualização nas ferramentas de variabilidades de software, diversas notações têm sido utilizadas para representar modelos de *features*, pode-se destacar algumas a seguir:

- *Diagrama de Feature* usa uma árvore hierárquica (estrutura de grafos) para apresentar as *features* e seus *features* filhos. Essa estrutura também é conhecida como a clássica árvore FODA. As conexões entre *features* indica se elas são opcionais, alternativas ou mandatórias;

- *Visão de Árvore* fornece uma representação vertical de árvore hierárquica de *features* em termos de variantes e invariantes de uma LPS. Uma *feature* pode ser expandida para revelar sub *features* e recolhida para escondê-las.

- *Tabela de Feature* representa um modelo de *feature* e produtos em linhas e colunas e é utilizada para resumir as *features* semelhantes em um modelo de variabilidades.

- *Linguagem Textual* inclui uma fórmula que define a composição e restrição das *features*.

- *Mapa de Fluxo* é uma técnica de visualização interativa, que combina modelos de *features* orientado a árvores e visualização em mapas de fluxo. Mapas de Fluxo podem indicar qual direção o fluxo está se movendo, e foi utilizado pela primeira vez na cartografia como uma mistura de mapas e fluxogramas.

Pode-se observar na Tabela 1.10 que o uso de Visão de Árvore é comumente mais utilizado para modelar as *features*, aparecendo em **25** ferramentas **(86.2%)**; Linguagem Textual aparece em **2** ferramentas **(6.9%)**; Diagramas de *Feature* usado em **12** ferramentas **41.4%**; **3** ferramentas **(10.3%)** utilizam Tabela de *Feature*; e **1** utiliza Mapas de Fluxo. Uma observação interessante sobre as alternativas de visualização é que várias dessas notações gráficas aparecem simultaneamente nos estudos analisados. Por isso, o número total de técnicas de visualização das variabilidades é maior do que o número de ferramentas da nossa pesquisa.

Tabela 1.10: Visualização das variabilidades

GUI	**Ferramenta**	**Total**
Diagrama de Feature	T1, T3-T6, T9-T11, T15, T19, T24, T29	12
Visão de Árvore	T3-T13, T15-T23, T25-T29	25
Tabela de *Feature*	T2, T7, T14	3
Linguagem Textual	T2, T14	2
Mapas de Fluxo	T21	1

Além disso, a escolha por um modelo de Visão de Árvore para modelar as *features* pode ser justificado devido a grande escalabilidade das funcionalidades de barra de rolagem, recolhimento e expansão das *features* disponíveis na interface gráfica do usuário comparado à tradicional árvore FODA (Diagrama de *Feature*). Além disso, considerando que a maioria das ferramentas são plugins desenvolvidas utilizando a plataforma EclipseIDE, foram encontradas evidências que o EMF foi utilizado como uma plataforma base para desenvolver os modelos de *features* na visão de árvore vertical.

Interoperabilidade com outras Tecnologias

Algumas das ferramentas de variabilidades de software utilizam banco de dados como um repositório para armazenar os modelos de *features*, permitindo a criação e manipulação dos modelos de variabilidades (Mendonça et al., 2009b). Algumas outras ferramentas utilizam um repositório de ativos reutilizáveis que fornece a base para a implementação do domínio (Frakes et al., 1997). Como apresentado na Tabela 1.11, identificou-se que **3** ferramentas **(10.3%)** oferece suporte ao banco de dados, enquanto que tal informação não estava disponível em 26 ferramentas.

Os arquivos XML/XMI podem ser utilizados como entrada/saída de geradores de códigos ou também podem ser acessados por um sistema durante o tempo de execução.

Tabela 1.11: Serviços de suporte

Suporte		Ferramentas	Total	%
Banco de Dados	Sim	T7, T23, T25	3	10.3
	N/A	T1-T6, T8-T22, T24, T26-T29	26	86.2
XML/XMI	Sim	T1, T3, T5-T25, T28, T29	25	86.2
	N/A	T2, T4, T26, t27	4	13.8
Ferramentas de Raciocínio	Sim	T2, T4, T8, T10, T15, T21, T23, T24	8	27.6
	N/A	T1, T3, T5-T7, T9, T11-T14, T16-T20, T22, T25-T29	21	72.4
Outras Integrações	Sim	T4, T9, T10, T12, T18, T19, T20	7	24.2
	N/A	T1-T3, T5-T8, T11, T13-T17, T21-T29	22	75.8

A Tabela 1.11 apresenta que **86.2%** das ferramentas tem suporte a arquivos XML/XMI. Também foram encontrados integração com ferramentas de raciocínio e análise em ferramentas de variabilidades de software, tais como apresentado na Tabela 1.11. Os mecanismos de raciocínio apoiam funcionalidades interativas tais como calcular as consequências das decisões do usuário baseada na semântica formal de linguagens de modelagens de *features* (Pleuss e Botterweck, 2012) e que podem ser implementadas utilizando um plugin do Eclipse. Além disso, há certo número de abordagens que oferecem análise automatizada de modelos de *features* que dependem de diferentes mecanismos (ex.: Lógica Descritiva, Lógica Proposicional, Programação por Restrição) e a maioria deles utilizam uma biblioteca BDD (ex.: JavaBDD[12]), SAT Solvers (i.e., SAT4J[13]), CSP solvers[14], Choco Solvers[15] para implementar e interpretar a semântica de modelos (Roos-Frantz et al., 2012).

Além das tecnologias já mencionadas, algumas ferramentas de variabilidades de software também fornecem um conjunto de ferramentas integradas para um melhor apoio do processo de desenvolvimento de LPS, como apresentado na última linha da Tabela 1.11. Tais ferramentas são: CVL Tool (T4), FeatureIDE (T9), FMP (T10), Gears (T12), Metadoc (T18), PLUM (T19) e pure::variants (T20).

Pure::variants (T20) da empresa pure-systems lançou pure::variants 3.2.18 em 1 de Fevereiro de 2016. Os destaques desta versão principal são uma série de novas e melhoradas integrações. Uma das novas integrações conecta pure::variants com Microsoft Office 2007/2010. Também, o novo lançamento do Conector para ferramenta AUTOSAR[16]. Esse conector permite usuários do AUTOSAR obter acesso a um conceito simples, mas poderoso na manipulação de variantes. A integração para Enterprise Architect[17] e Rational Rhapsody[18] contém integração nativa UI, permitindo, por exemplo, visualização

[12]JavaBDD - http://javabdd.sourceforge.net/
[13]SAT4J - http://www.sat4j.org/
[14]CSP solvers - http://4c.ucc.ie/web/outreach/tutorial.html
[15]Choco Solvers - http://choco-solver.org/
[16]AUTOSAR - http://www.autosar.org/
[17]Enterprise Architect - http://www.sparxsystems.com.au/
[18]Rational Rhapsody - http://www-03.ibm.com/software/products/pt/ratirhapfami

imediata de variantes na ferramenta UML. Da mesma forma, o EMF Feature Mapper[19] integra perfeitamente ao pure::variants e ferramenta baseada em EMF (ex.: Papyrus, Topcased ou Rational Software Architect). O Conector Matlab/Simulink[20] fornece GV para modelos Simulink incluindo Fluxo de Estado e modelos *Target Link*. O Conector Pure::variants para BIRT[21] permite relatórios imprimíveis sem um formato comum de texto. A integração com CaliberRM[22] permite que modelos de requisitos sejam modelados como modelos de *Features* do pure::variants. Há também uma integração com ClearQuest[23] onde variantes específicas de defeitos e testes podem ser gerenciadas facilmente e eficientemente para tanto novos como e já existentes banco de dados ClearQuest. Além disso, pure::variants sincroniza com IBM Rational DOORS[24], que é uma ferramenta certificada IBM Ready for Rational Software para gerenciar variabilidades em requisitos e apoiar na criação e manutenção de requisitos reutilizáveis com os módulos do DOORS.

A ferramenta Gears (T12) permite a integração com outras ferramentas através de *Bridges* (pontes) que apoiam a engenharia de LPS através dos sistemas completos e do ciclo de vida de desenvolvimento de software. Gears pode integrar com Microsoft Visual Studio[25] e EclipseIDE. Gears fornece pontes para gerenciamento da engenharia de requisitos por meio da integração com a Ferramenta IBM Rational DOORS. Também integra com IBM Rational Rhapsody, MagicDraw[26], e Enterprise Architect para gerenciar e modelar *features*. Integrações com gerenciamento de qualidade e ferramentas de teste permitem uma diversidade de caso de testes utilizando IBM Rational Quality Manager[27]. A integração com MadCap Software Flare[28] permite o gerenciamento de documentos na ferramenta Gears. Além disso, Gears integra com *Configuration Management* para permitir execução automática de operações utilizando Serena Dimensions[29], Perforce[30], IBM Rational ClearCase[31], IBM Rational Synergy[32], IBM Rational Team Concert[33], e ferramentas Subversion[34].

[19]EMF Feature Mapper - http://wiki.eclipse.org/EMF/FAQ
[20]Matlab/Simulink http://www.mathworks.com/products/simulink/
[21]BIRT - http://www.eclipse.org/birt
[22]CaliberRM - www.borland.com/pt-BR/Products/Requirements-Management/Caliber
[23]ClearQuest - www.ibm.com/software/products/en/clearquest
[24]DOORS - http://www-01.ibm.com/software/awdtools/doors/
[25]Microsoft Visual Studio - https://www.visualstudio.com/
[26]MagicDraw - www.nomagic.com/products/magicdraw.html
[27]IBM Rational Quality Manager - www.ibm.com/software/products/pt/ratiqualmana
[28]MadCap Software Flare - www.madcapsoftware.com/products/flare/
[29]Serena Dimensions - www.serena.com
[30]Perforce - https://www.perforce.com/
[31]IBM Rational ClearCase - www.ibm.com/software/products/pt/clearcase
[32]IBM Rational Synergy - www.ibm.com/software/products/pt/ratisyne
[33]IBM Rational Team Concert - www.ibm.com/software/products/pt/rtc
[34]Subversion - https://subversion.apache.org/

FeatureIDE (T9) suporta a integração com ferramentas de programação orientada a *feature* (FOP), tais como ferramentas AHEAD (Batory et al., 2004), FeatureC++[35], e FeatureHouse (Apel et al., 2009). Há também integração com ferramentas orientadas a aspectos (AOP) através da integração com AspectJ[36] que permite uma extensão orientada a aspectos transparente para a linguagem Java. FeatureIDE suporta a integração com programação orientada a delta (DOP) através do uso de tecnologias de programação orientada a delta, tais como DeltaJ[37] e DeltaEcore[38]. Além do mais, FeatureIDE integra com tecnologias pré processadas, tais como Colligens[39], TypeChef[40], Munge[41], Antenna[42] que podem habilitar e desabilitar fragmentos de código. Além disso, FeatureIDE tem suporte para gramática de modelos de *feature* com as seguintes ferramentas: Guidsl Tool[43], S.P.L.O.T. (Mendonça et al., 2009b), Velvet[44], FMP (Czarnecki et al., 2005a) e SPL Conqueror[45].

PLUM (T19) está integrado no Eclipse e utiliza uma ampla gama de tecnologias, tais como EMF para modelar as variabilidades, *Object Constraint Language* (OCL)[46] para modelos de decisão com dependência, BIRT para gerar valiosos relatórios de métricas da LPS, ModelBus (MDDi)[47] para permitir aos clientes que façam solicitações por geração de produtos em um servidor remoto e Subversion[48] para controlar versões do sistema e histórico de métricas.

CVL tool (T4) integra com Papyrus[49], uma ferramenta de nível industrial e *open source* que projeta modelos de caso de uso. A ferramenta FMP (T10) integra com Rational Software Modeler (RSM) ou Rational Software Architect (RSA) para permitir a modelagem de LPS em UML. Metadoc (T18) integra com IBM Rational DOORS para suportar a modelagem de variabilidades para o gerenciamento de requisitos.

É importante destacar que outras ferramentas de variabilidades de software devem possuir funcionalidades de integração; porém, tal informação não estava explicitamente disponível nos estudos analisados.

[35]FeatureC++ - wwwiti.cs.uni-magdeburg.de/itidb/fcc/
[36]AspectJ - https://eclipse.org/aspectj/
[37]DeltaJ - deltaj.sourceforge.net/
[38]DeltaEcore - deltaecore.org/
[39]Colligens - https://sites.google.com/a/ic.ufal.br/colligens/
[40]TypeChef - http://ckaestne.github.io/TypeChef/
[41]Munge - http://sonatype.github.com/munge-maven-plugin/
[42]Antenna - http://antenna.sourceforge.net/
[43]Guidsl Tool - http://www.cs.utexas.edu/schwartz/ATS/fopdocs/guidsl.html
[44]Velvet - http://wwwiti.cs.uni-magdeburg.de/itidb/research/MultiPLe/modeling.htm
[45]SPL Conqueror - http://fosd.de/SPLConqueror
[46]OCL - http://www.omg.org/spec/OCL/
[47]MDDI - https://wiki.eclipse.org/Mddi
[48]Subversion - https://subversion.apache.org/
[49]Papyrus - https://eclipse.org/papyrus/

A.3 Estudos Relacionados

O objetivo desse artigo é fornecer uma compreensão fundamental sobre ferramentas de variabilidades de software com relação aos tipos de ferramentas, evidências de uso pela indústria, tecnologias utilizadas para implementação das ferramentas, entre outras características relevantes. Para esse propósito, realizou-se um MS relatando o atual status das ferramentas de variabilidades de software, uma vez que, para o nosso conhecimento, não há estudo similar focado em ferramentas de variabilidades de software. Alguns estudos relacionados são os seguintes.

Lisboa et al., (2010) realizou uma RSL sobre ferramentas de análise de domínio. Um conjunto de questões de pesquisa foi definido para identificar se ferramentas oferecem suporte a um processo específico ou genérico. Para mapear a forma como o processo de análise de domínio é suportado, uma série de funcionalidades e características foi extraída incluindo o contexto em que as ferramentas foram desenvolvidas (para a indústria ou ambiente acadêmico). No final, **19** ferramentas foram selecionadas. A maioria das ferramentas suportam processos específicos como *Feature-Oriented Domain Analysis* (FODA). A maioria destas ferramentas apresentam funcionalidades semelhantes com os objetivos análogos, que são: *planejamento*, para a coleta de dados para identificar características do domínio baseado em documentação de pré-análise, matriz do domínio, funções de avaliação e definição do escopo; *modelagem*, para representar o escopo de domínio baseado em diagramas e tabelas, variabilidades, *features* obrigatórias, regras de composição, identificação do grupo de *features*, tipos de relacionamento, atributos de *features*; e *validação*, baseada na documentação das *features*, documentação dos requisitos, relação entre as *features* e requisitos, dicionários, relatórios e verificadores de consistência. Configuração e documentação de produtos são funcionalidades extras. Os autores encontraram que **63.1%** das ferramentas foram desenvolvidas na academia, **21.05%** na indústria e **15.78%** em ambos, academia e indústria.

Em Pereira et al., (2015) os autores descrevem um estudo similar para LPS. Eles listaram as principais características e funcionalidades das ferramentas de LPS e forneceram uma atualização para o estudo de Lisboa et al. Os autores reportaram **41** ferramentas incluindo ferramentas de análise de domínio, variabilidades de software, gerenciamento de requisitos e suporte a LPS. Os autores descobriram que **63.4%** das ferramentas foram desenvolvidas na academia, **12.2%** na indústria e **24.4%** ambos em academia e indústria, e a maioria das ferramentas compartilham funcionalidades semelhantes, que são: **73%** apoiam atividades de planejamento, **81%** fornecem derivação de produtos e **71%** oferece funcionalidades de importação/exportação.

Galster et al. (2014) executou uma RSL sobre variabilidades em sistemas de software. O objetivo era avaliar evidências sobre a manipulação das variabilidades e identificar tendências nas pesquisas de variabilidades de software. Eles analisaram atividades de GV em sistemas de software e realizaram uma extensa pesquisa manual sobre milhares

de documentos e selecionaram 196 trabalhos para a revisão. Como resultado, eles encontraram que 128 estudos focavam nas variabilidades em tempo de projeto e 50 estudos nas variabilidades em tempo de execução. Essa RSL não informa sobre as ferramentas de variabilidades de software, mas discute as tecnologias disponíveis para gerenciar variabilidades.

As pesquisas de Berger et al., (2013, 2014) apresentam um questionário distribuído aos profissionais da indústria sobre modelagem de variabilidades. Tais pesquisas destacam a importância das ferramentas de variabilidades de software, tais como: ferramentas específicas para domínio (38%), ferramentas *open source* (29.4%) e comerciais (26.5%) usadas na indústria como pure:variants e GEARS.

Bhumula (2013) realizou uma pesquisa online para identificar ferramentas de variabilidades de software em uso. Os autores realizaram uma análise comparativa sobre 14 ferramentas em torno de diferentes critérios, tais como: questões de governança e aspectos técnicos como abordagem de variabilidades, técnicas de visualização dos modelos de *features*, *binding time*, tipos de *feature*, etc. No final, os autores recomendaram um conjunto de melhorias para a indústria e melhores praticas para satisfazer os critérios propostos.

Em termos de estudos preliminares, o estudo de Lisboa et al. composto apenas de ferramentas de análise de domínio, não inclui muitas outras ferramentas que têm o objetivo de gerenciar variabilidades. Em contraste, o estudo de Pereira et al. compreende ferramentas utilizadas na LPS ou com este objetivo. Por outro lado, nosso estudo compreende todas as ferramentas de variabilidades de software disponíveis que abrangem a maioria ou todas as atividades da GV. A sobreposição dos estudos é elevada quando comparada ao estudo de Pereira et al., explicando que a grande maioria das ferramentas de variabilidades de software está no domínio de LPS.

As pesquisas realizadas por Berger et al.,(2014; 2013) sobrepõe nove ferramentas, o que indica que 32,1% das ferramentas encontradas por nosso MS estão em uso pela indústria. O estudo da Bhumula (2013) sobreposta oito ferramentas. Esse estudo avaliou alguns aspectos técnicos das ferramentas como o nosso estudo; no entanto, tal estudo não é uma estudo secundário o que compromete a avaliação de importantes ferramentas de variabilidades de software disponíveis.

A.4 Ameaças à Validade

O resultado deste trabalho pode ser afetado por um conjunto de limitações que podem gerar viés a seleção e análise relatada neste MS. Nesta seção, explorou-se as principais ameaças à validade dos nossos resultados, incluindo ameaças à validade interna, validade externa, validade de construto e validade conclusão.

Validade de Construto: está relacionada ao estabelecimento de uma relação entre a teoria por trás do estudo e as observações. Abrange questões que estão relacionadas com projeto do estudo, analise de sua capacidade de representar o que os pesquisadores têm em mente, e o que é investigado de acordo com as questões de pesquisa. Para evitar esta ameaça, foram aplicados os conceitos de "gerenciamento de variabilidade", "ferramenta"e "software"como as principais construções do nosso estudo. Também seguiu-se as orientações de Kitchenham (2007) para projetar as questões de pesquisa. Outro aspecto importante é garantir a descoberta de todos os estudos relevantes no tópico sob investigação. Para este efeito, foram combinadas pesquisas automáticas e manuais para aumentar a cobertura dos estudos primários e representativos selecionados. A técnica de *snowballing* também foi usada para reunir estudos complementares; além disso, o procedimento de busca foi validado contra outras revisões sistemáticas, a fim de garantir a completude.

Validade Interna: está preocupada se algum tratamento que foi introduzido pode causar algum efeito no resultado. Esta ameaça pode afetar a variável independente no que diz respeito à causalidade, sem o conhecimento do pesquisador. Em particular, as principais ameaças à validade interna neste trabalho são a seleção de estudos primários e viés individual na avaliação. A fim de garantir que este MS esteja completa e nenhuma literatura importante esteja faltando, foram utilizados sete fontes bibliográficas digitais. Além disso, foi realizado uma pesquisa manual em anais de eventos sobre ferramentas de variabilidades de software e LPS. Adicionalmente, os motores de busca web também foram incluídos, a fim de incorporar os estudos primários mais completos possíveis para aumentar a confiabilidade das conclusões. A outra ameaça resulta do viés individual dos pesquisadores ao avaliar os estudos primários. A fim de minimizar o viés na seleção, o processo de seleção foi executado pelo autor principal e revisado constantemente pelos coautores, junto com aplicação de critérios de inclusão e exclusão. Também tentou-se evitar equívocos de conceitos que poderiam causar resultados tendenciosos. Essa ameaça foi atenuada usando o protocolo discutido no início do MS para garantir um processo de seleção imparcial, tanto quanto possível.

Validade Externa: investiga se os resultados dos experimentos são generalizáveis. Uma dessas condições são as palavras-chave usadas no título e resumo. Essas palavras-chaves podem ter influenciado a completude e identificação de estudos primários. Isto significa que a nossa pesquisa pode ter perdido estudos cujos autores teriam usado outros termos para especificar ferramentas de variabilidades de software. Além disso, como buscou-se por ferramentas, é possível que haja ferramentas não registradas na literatura. Também, o escopo do nosso MS é sobre ferramentas de variabilidades de software desenvolvidas entre 1990 e 2016 para cobrir conhecimento recente da pesquisa.

Validade de Conclusão: se concentra em questões que afetam a capacidade de tirar as conclusões corretas a partir do estudo. Refere-se à forma como os dados foram extraídos a partir das ferramentas, uma vez que nem todas as informações eram evidentes

para responder às perguntas e alguns dados tiveram de ser interpretados. Portanto, a fim de garantir a validade, foram analisadas várias fontes de dados, ou seja, publicações, protótipos, relatórios técnicos, documentação técnica, páginas web e manuais, além de executáveis das ferramentas. Além disso, o autor principal realizou a extração de dados e da análise que pode incluir decisões equivocadas. Para atenuar esse risco todos os dados foram revistos por todos os coautores e um processo de extração rigorosa foi aplicado com base nas diretrizes de Kitchenham et al., (2007).

A.5 Tendências Futuras

Na última década, a comunidade de pesquisa de engenharia LPS tem crescido significativamente. Essa comunidade provou capacitar empresas de diferentes domínios, tais como componentes eletrônicos, setor automotivo, energia, telecomunicações e aviação para investir esforços significativos no desenvolvimento de uma diversidade de sistemas semelhantes a um custo menor, em menor tempo e com maior qualidade, quando comparado com o desenvolvimento de sistemas individuais.

Desenvolvimento de software configurável, personalizado com variabilidades e criando famílias de softwares estão entre as tendências recentes mais bem sucedidas. No entanto, a crescente demanda dos clientes por produtos configuráveis levaram à exigência de que linhas de produtos convencionais e mecanismos de variabilidades atuais não podem resolver. Além disso, outro problema é a visualização de grandes modelos de variabilidades, pois as ferramentas comerciais atuais oferecem suporte limitado para mecanismos de visualização avançado (Bosch et al., 2015).

Embora uma espécie de diferentes ferramentas e técnicas de variabilidades software tem sido desenvolvida, a indústria adotou técnicas diferentes para gerenciar variabilidades, incluindo a produção de suas próprias ferramentas sem padronização. Essa diversidade pode ser entendida pela necessidade de abordar aspectos específicos de domínio para lidar com variabilidades do produto e, se este for o caso, é pouco provável que alguma vez será possível observar uniformização da notação de variabilidades e ferramentas (Berger et al., 2013).

Portanto, GV ainda enfrenta muitos desafios, incluindo a evolução e visualização de modelos de variabilidades, técnicas de representação dos modelos, gerenciamento de restrição (Berger et al., 2013; Bosch et al., 2015) e preocupações com o tempo de execução pode modificar dinamicamente as variabilidades estruturais. Também novas tendências na tentativa de proporcionar melhores maneiras de gerenciar a evolução das variabilidades, no que se refere à resolução dinâmica, permite a reconfiguração de produtos para alterar flexivelmente decisões, em oposição a mecanismos de licãogação estática (Capilla et al., 2013).

Metzger e Pohl (2014) destacaram em seu estudo três tendências que terão um impacto na pesquisa LPS na próxima década, que são: (i) gerenciar variabilidades em ambientes que não são de LPS, (ii) melhorar o *feedback* imediato de *big data* e *cloud computing* durante o LPS, e (iii) direcionar suposições de mundo aberto em ambientes LPS. Todas as tendências mencionadas indicam novas oportunidades de pesquisa no contexto de LPS e GV.

A.6 Conclusão

Diferentemente de outros estudos relacionados com variabilidades de software e LPS, realizou-se nesta pesquisa um MS com foco apenas no estado da arte e características das ferramentas de variabilidades de software utilizadas em projetos de pesquisa e indústria. Foram identificados 879 estudos ao pesquisar a literatura, dos quais 95 estudos (incluindo estudos de *snowballing*) foram qualificados para a fase de extração de dados após a aplicação dos critérios de inclusão/exclusão conforme descrito na Seção A.1. Estes 94 trabalhos resultaram em 29 diferentes ferramentas de variabilidades de software. A grande maioria delas são ferramentas GV seguidas de verificação de restrição das variabilidades, derivação de produtos, modelagem de variabilidades e configuração de produtos. Outras ferramentas identificadas em nosso MS cobriam fases especificas da LPS, como ferramentas de análise de domínio e ferramentas de gerenciamento de requisitos.

Também descobriu-se que quase 50% são ferramentas utilizadas ou avaliadas pela indústria. Quanto à abordagem de modelo de *features*, *Cardinality Based Feature Model* (CBFM), é notação mais frequentemente relatada seguida por *Feature Oriented Domain Analysis* (FODA). Nossa mapeamento sistemático revelou que a grande maioria das ferramentas encontradas foi implementada através da linguagem de programação Java como plugins e fornecem interoperabilidade adicionada a facilidades de visualização. O escopo abrangente deste estudo revelou não só a maturidade da tecnologia usada pelas ferramentas, mas também as tendências de pesquisa e desafios para novas melhorias. Espera-se para o futuro uma melhor integração entre GV e outras ferramentas de engenharia de software, em particular, uma melhor conexão com arquiteturas de software, ferramentas de modelagem e apoio específico para questões de tempo de execução das variabilidades.

Apêndice B - Verificação de Consistência da VMTools-RA

Este apêndice apresenta as verificações de consistência para as visões de variabilidade da VMTools-RA. Tais avaliações foram realizadas pela ferramenta S.P.L.O.T..

B.1 Verificação de Consistência da VMTools-RA

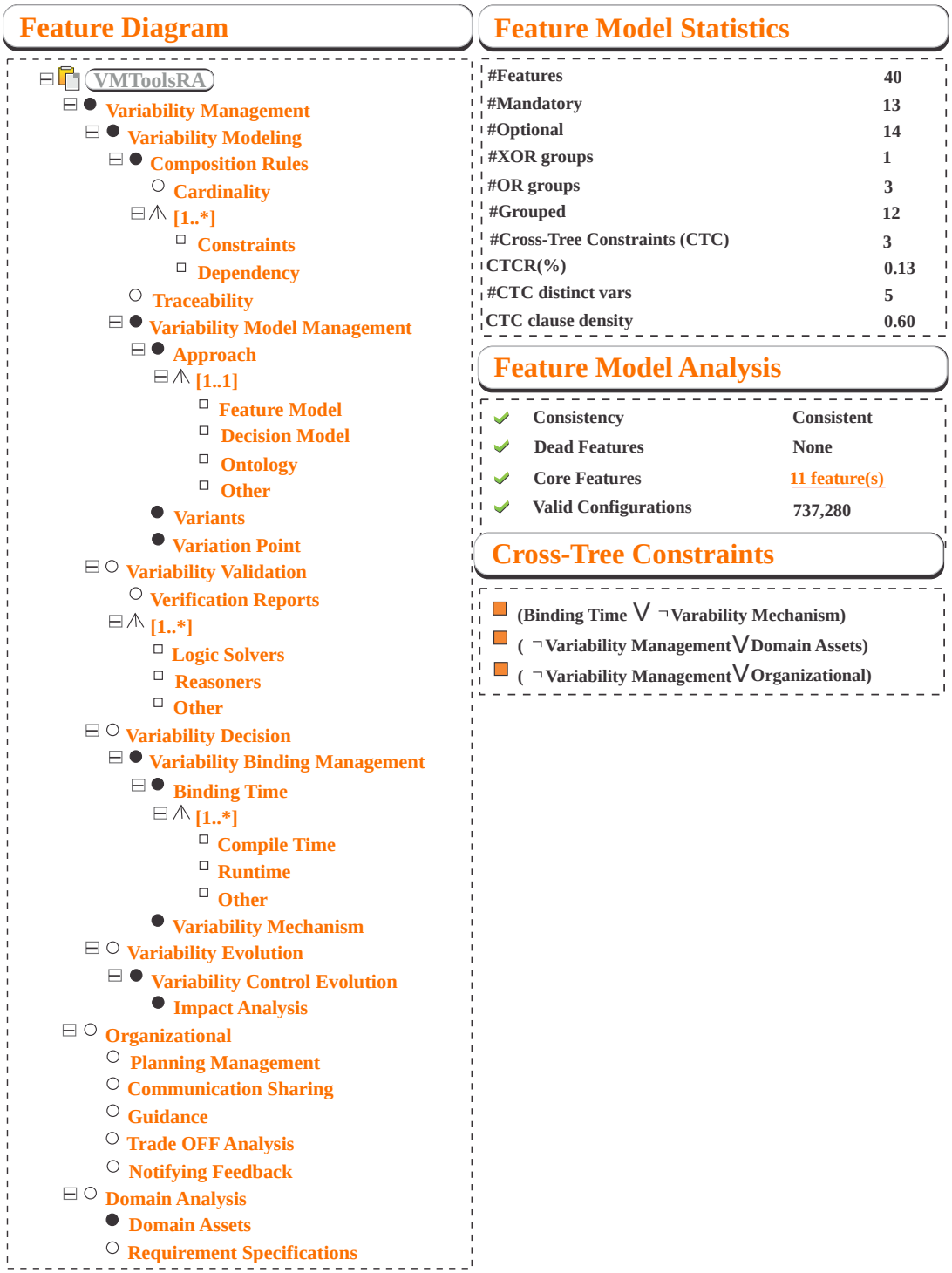
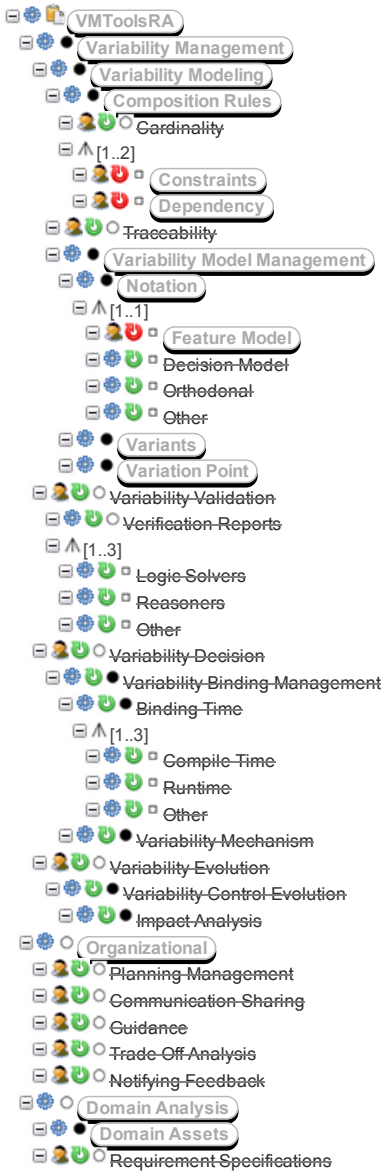


Figura 2.1: Verificação de consistência da VMTools-RA.

B.2 Instância Arquitetural de Ferramenta de Modelagem de Variabilidades

Exemplo de configuração de ferramenta de modelagem de variabilidades baseada na VMTools-RA. A Ferramenta abaixo apenas modela as variabilidades utilizando modelo de *feature* e regras de composição.



Configuration Steps [reset]

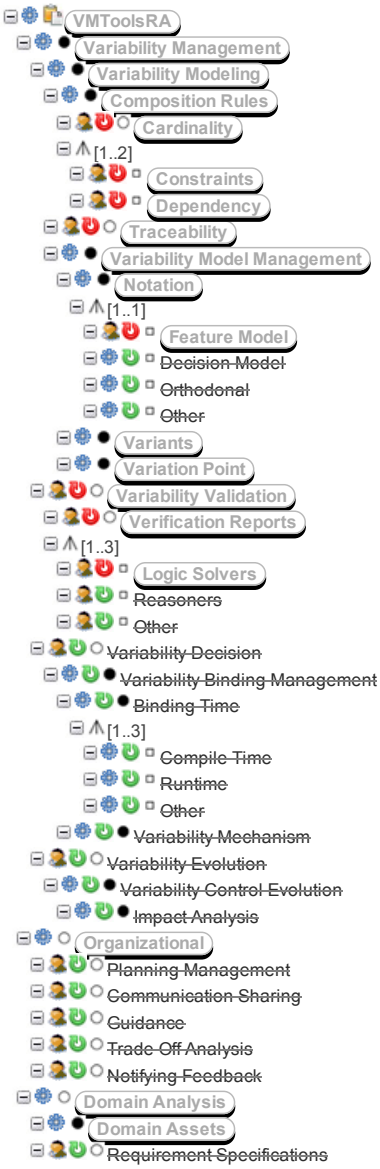
100%

Step	Decision	#Decisions (cummulative)	#Propagations (at step)	#SAT checks (at step)	SAT time (at step)
1	✔ VMToolsRA	11 (27.5%)	10	14	2 ms
2	✔ Constraints	12 (30.0%)	0	4	0 ms
3	✔ Dependency	13 (32.5%)	0	4	0 ms
4	✘ Cardinality	14 (35.0%)	0	4	0 ms
5	✘ Traceability	15 (37.5%)	0	4	9 ms
6	✔ Feature Model	19 (47.5%)	3	6	1 ms
7	✘ Variability Validation	24 (60.0%)	4	7	1 ms
8	✘ Variability Decision	31 (77.5%)	6	3	0 ms
9	✘ Variability Evolution	34 (85.0%)	2	3	0 ms
10	✘ Planning Management	35 (87.5%)	0	2	0 ms
11	✘ Communication Sharing	36 (90.0%)	0	2	0 ms
12	✘ Guidance	37 (92.5%)	0	2	0 ms
13	✘ Trade Off Analysis	38 (95.0%)	0	2	0 ms
14	✘ Notifying Feedback	39 (97.5%)	0	2	0 ms
15	✘ Requirement Specifications	40 (100.0%)	0	0	0 ms

Done! (Export configuration: CSV file | XML)

B.3 Instância Arquitetural de Ferramenta de Modelagem e Configuração das Variabilidades com Verificação de Consistência

Exemplo de configuração de ferramenta de modelagem de variabilidades com verificadores lógicos de consistência e rastreabilidade entre modelos e ativos de domínio. A Ferramenta abaixo modela as variabilidades utilizando modelo de *feature* e regras de composição.



Configuration Steps [reset]

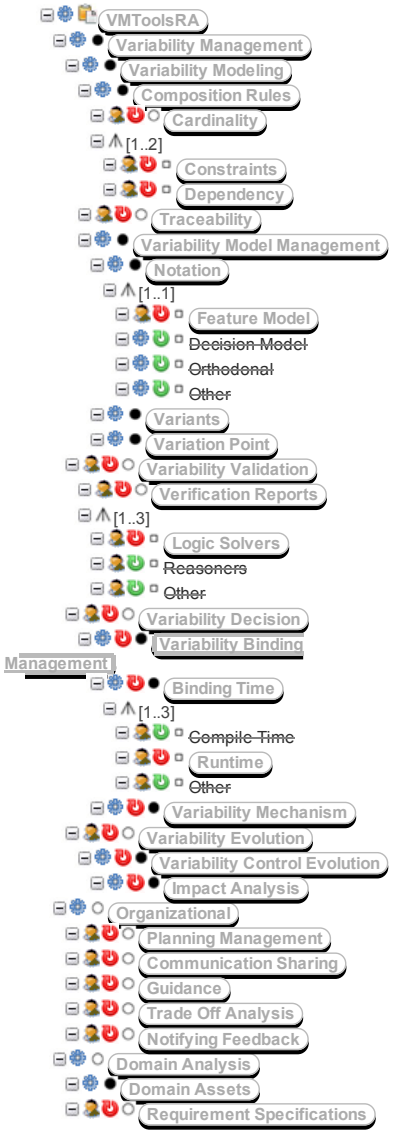
100%

Step	Decision	#Decisions (cummulative)	#Propagations (at step)	#SAT checks (at step)	SAT time (at step)
1	✔ VMToolsRA	11 (27.5%)	10	14	1 ms
2	✔ Cardinality	12 (30.0%)	0	4	0 ms
3	✔ Constraints	13 (32.5%)	0	4	0 ms
4	✔ Dependency	14 (35.0%)	0	4	0 ms
5	✔ Traceability	15 (37.5%)	0	4	1 ms
6	✔ Feature Model	19 (47.5%)	3	6	1 ms
7	✔ Variability Validation	20 (50.0%)	0	2	0 ms
8	✔ Verification Reports	21 (52.5%)	0	2	1 ms
9	✔ Logic Solvers	22 (55.0%)	0	2	0 ms
10	✘ Reasoners	23 (57.5%)	0	2	0 ms
11	✘ Other	24 (60.0%)	0	3	1 ms
12	✘ Variability Decision	31 (77.5%)	6	3	0 ms
13	✘ Variability Evolution	34 (85.0%)	2	3	1 ms
14	✘ Planning Management	35 (87.5%)	0	2	1 ms
15	✘ Communication Sharing	36 (90.0%)	0	2	0 ms
16	✘ Guidance	37 (92.5%)	0	2	1 ms
17	✘ Trade Off Analysis	38 (95.0%)	0	2	1 ms
18	✘ Notifying Feedback	39 (97.5%)	0	2	0 ms
19	✘ Requirement Specifications	40 (100.0%)	0	0	0 ms

Done! (Export configuration: CSV file | XML)

B.4 Instância Arquitetural de Ferramenta de Gerenciamento de Variabilidade

Exemplo de configuração de ferramenta de gerenciamento de variabilidade. A Ferramenta abaixo possui quase todos os elementos propostos na VMTools-RA.



Configuration Steps [reset]

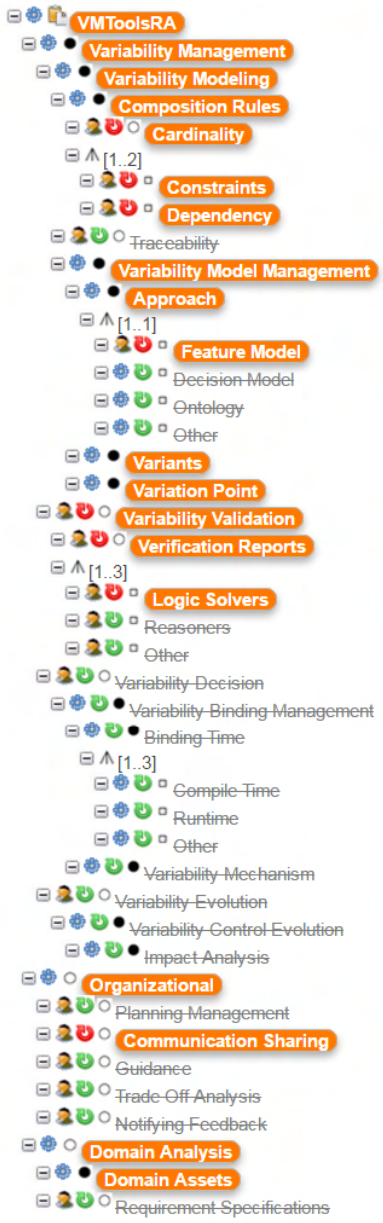
100%

Step	Decision	#Decisions (cummulative)	#Propagations (at step)	#SAT checks (at step)	SAT time (at step)
1	✔ VMToolsRA	11 (27.5%)	10	14	1 ms
2	✔ Cardinality	12 (30.0%)	0	4	1 ms
3	✔ Constraints	13 (32.5%)	0	4	0 ms
4	✔ Dependency	14 (35.0%)	0	4	1 ms
5	✔ Traceability	15 (37.5%)	0	4	0 ms
6	✔ Feature Model	19 (47.5%)	3	6	0 ms
7	✔ Variability Validation	20 (50.0%)	0	2	0 ms
8	✔ Verification Reports	21 (52.5%)	0	2	0 ms
9	✔ Logic Solvers	22 (55.0%)	0	2	0 ms
10	✔ Variability Decision	26 (65.0%)	3	5	1 ms
11	✔ Runtime	27 (67.5%)	0	2	0 ms
12	✔ Variability Evolution	30 (75.0%)	2	4	0 ms
13	✔ Planning Management	31 (77.5%)	0	2	0 ms
14	✔ Communication Sharing	32 (80.0%)	0	2	0 ms
15	✔ Trade Off Analysis	33 (82.5%)	0	2	0 ms
16	✔ Notifying Feedback	34 (85.0%)	0	2	0 ms
17	✔ Requirement Specifications	35 (87.5%)	0	2	0 ms
18	✔ Guidance	36 (90.0%)	0	2	0 ms
19	✘ Other	37 (92.5%)	0	2	0 ms
20	✘ Compile Time	38 (95.0%)	0	2	0 ms
21	✘ Other	39 (97.5%)	0	2	0 ms
22	✘ Reasoners	40 (100.0%)	0	0	0 ms

Done! (Export configuration: CSV file | XML)

B.5 Verificação da Consistência da Instância Arquitetural de Ferramenta de Variabilidade

A Figura abaixo representa a consistência da instância arquitetural para a ferramenta de variabilidade de software apresentada no exemplo de aplicaÃ§Ã£o do Capítulo 5.



Configuration Steps [reset]

100%

Step	Decision	#Decisions (cummulative)	#Propagations (at step)	#SAT checks (at step)	SAT time (at step)
1	✔ VMToolsRA	11 (27.5%)	10	14	1 ms
2	✔ Cardinality	12 (30.0%)	0	4	0 ms
3	✔ Constraints	13 (32.5%)	0	4	0 ms
4	✔ Dependency	14 (35.0%)	0	4	0 ms
5	✘ Traceability	15 (37.5%)	0	4	0 ms
6	✔ Feature Model	19 (47.5%)	3	6	0 ms
7	✔ Variability Validation	20 (50.0%)	0	2	0 ms
8	✔ Verification Reports	21 (52.5%)	0	2	0 ms
9	✔ Logic Solvers	22 (55.0%)	0	2	0 ms
10	✘ Reasoners	23 (57.5%)	0	2	0 ms
11	✘ Other	24 (60.0%)	0	3	0 ms
12	✘ Variability Decision	31 (77.5%)	6	3	0 ms
13	✘ Variability Evolution	34 (85.0%)	2	3	0 ms
14	✘ Planning Management	35 (87.5%)	0	2	0 ms
15	✔ Communication Sharing	36 (90.0%)	0	2	0 ms
16	✘ Guidance	37 (92.5%)	0	2	0 ms
17	✘ Trade Off Analysis	38 (95.0%)	0	2	0 ms
18	✘ Notifying Feedback	39 (97.5%)	0	2	0 ms
19	✘ Requirement Specifications	40 (100.0%)	0	0	0 ms

Done! (Export configuration: CSV file | XML)

Apêndice C - Questionário de Caracterização dos Especialistas

C.1 Questionário de Caracterização dos Especialistas em Arquitetura de Referência

Participant Characterization Questionnaire

Study Adhesion Term

I declare to be aware of participating in the evaluation of VMTools-RA, a Reference Architecture for Variability Management Tools. I also declare to be aware that the results collected about me will be confidential and that I will not receive any charge for participation, except to learning new techniques / technologies. ( ) I accept

Participant Name: _____ Institution: _____
What is your education level?
( ) Masters student ( ) PhD candidate ( ) Master ( ) PhD
What role describes you best?
( ) Software architect
( ) Enterprise architect
( ) Senior software engineer
( ) Junior software engineer
( ) Business, technical, project manager
( ) Business analyst, requirements engineer
( ) Software developer
Years of experience in your role (e.g.: Software architect 3 years, Software Developer 5 years)
_____
How would you rate your experience in using reference architectures?
( ) None
( ) Beginner
( ) Intermediate
( ) Advanced
( ) Expert
How would you rate your experience in reference architecture design?
( ) None
( ) Beginner
( ) Intermediate
( ) Advanced
( ) Expert
How many reference architectures have you designed, and for what domain(s)? (e.g.: 1 for robot, 1 for automobile)
_____
How would you rate your experience in reference architecture evaluation?
( ) None
( ) Beginner
( ) Intermediate
( ) Advanced
( ) Expert

C.2 Questionário de Caracterização dos Especialistas em Gerenciamento de Variabilidade

Participant Characterization Questionnaire

Study Adhesion Term

I declare to be aware of participating in the evaluation of VMTools-RA, a Reference Architecture for Variability Management Tools. I also declare to be aware that the results collected about me will be confidential and that I will not receive any charge for participation, except to learning new techniques / technologies. ( ) I accept

Participant Name: _____ Institution: _____

What is your education level?

( ) Masters student ( ) PhD candidate ( ) Master ( ) PhD

What role describes you best?

( ) Software architect
( ) Enterprise architect
( ) Senior software engineer
( ) Junior software engineer
( ) Business, technical, project manager
( ) Business analyst, requirements engineer
( ) Software developer

Years of experience in your role (e.g.: Software architect 3 years, Software Developer 5 years)

_____

How would you rate your experience in using reference architectures?

( ) None
( ) Beginner
( ) Intermediate
( ) Advanced
( ) Expert

How would you rate your experience in reference architecture design?

( ) None
( ) Beginner
( ) Intermediate
( ) Advanced
( ) Expert

How many reference architectures have you designed, and for what domain(s)? (e.g.: 1 for robot, 1 for automobile)

_____

How would you rate your experience in reference architecture evaluation?

( ) None
( ) Beginner
( ) Intermediate
( ) Advanced
( ) Expert

Apêndice D - Respostas dos Especialistas no Estudo Empírico Qualitativo da VMTools-RA

Este apêndice apresenta as respostas na integra dos especialistas E1, E2 e E3 para a inspeção por *checklist* realizada para avaliar a VMTools-RA.

Respostas do Especialista E1

1. General Information

1. The reference architecture presents:
i) overview information
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: It is necessary to offer a glossary of terms.
ii) release date
Yes (x) No ( ) I am not expert to answer this question ( )
iii) version
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( ).
iv) owner (e.g. organization)
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
v) change history
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
vi) short description
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
vii) scope
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
viii) domain terminology
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: It is offered in the conceptual view, but I suggest to move in front to support document reading
xii) open decisions, and
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
xiii) supporting material.
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

2. Are the concepts underlying the reference architecture document clearly explained (e.g., they consistently use a domain terminology, labels for diagrams)?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Are all the selected stakeholders identified?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: I consider that it is important to specify which stakeholder is related with which requirement

4. Are all the selected stakeholders' concerns identified?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: It must be linked each concern with the related stakeholder and requirements

5. Does the reference architecture present the potential risk to its stakeholders through its life cycle?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Questions referring to viewpoints

6. All selected viewpoints for the reference architecture were considered?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: It is important to expose why others viewpoints were not considered. Are the selected viewpoints enough to represent the solution for all requirements?
6.1. If yes, there is a description of each viewpoint used in the reference architecture document?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: There is a lack of reference citation in all the document including for each viewpoint.

6.2. Does each viewpoint description include:
i) Viewpoint name;
Yes (x) No ( ) I am not expert to answer this question ( )
ii) Identification of the stakeholders addressed by that viewpoint;
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: I consider important to relate each viewpoint, with the requirements, stakeholders, and Interests. Maybe using a table at the final of the document
iii) The architectural concerns addressed by that viewpoint; and
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iv) The architectural views used by the viewpoint.
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Questions referring to views

7. Does each view contains:
i) An identifier;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
ii) Introductory and additional information;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iii) Know incompatibilities related to its viewpoint;
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
iv) One or more architectural models;
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: What do you mean by architectural models here ? Architectural representation, styles...?

Questions referring to architectural models

8. Does each model contain:
i) Version identification; and
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

9. Does each model use well known notation (e.g., SysML and UML)
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Construction and Content

1. Does the reference architecture satisfy:
i) legal regulation;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
ii) international standards;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
iii) company policies and rules; and
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
iv) Best practices and guidelines.
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

2. Does the reference architecture presents:
i) Feasibility to derive it in instances;
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: It is important to offer guidelines to use this RA at insantiating it in concrete architectures
ii) Maintainability related issues;
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

3. Does every viewpoint description contain only the required information (i.e., there's no useless information in the viewpoint definition)?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: I consider that it is a lack of information. How views are related among them ? Which stakeholders are related, and which requirements are addressed by each viewpoint and view?

4. Does the detail level favors the reference architecture understanding?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: The understanding could be improved at offering a case study, e.g., using a previous software architecture of a variability tool, and represented it using the RA

5. Is the abstraction level consistent with the goals of the reference architecture?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
6. Does each view correctly represent its viewpoint?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

7. Do the diagrams address all the concerns framed by the viewpoint?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Domain Expert:

8. Have the following quality attributes have been considered:
Observation: It is not possible to verify if the quality attributes have been addressed, due to the document does not present quality attribute requirements for the variability tools. I consider extremely important to include such requirements into the document.

Continuação das Respostas do Especialista E1

i) Availability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
ii) Interoperability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
iii) Maintainability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
iv) Performance
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
v) Security
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
vi) Reliability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
vii) Scalability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
viii) Protection
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
ix) Usability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

9. Does the reference architecture specifies which are the variability mechanisms
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

10. Are there correspondence rules?
Yes ( ) No ( ) I am not expert to answer this question (x)

10.1 For each such rule, is there at least one correspondence satisfying each rule?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

10.2 Are all the correspondences identified and documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

10.3 Are all the correspondences identified in its participating elements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
11. Can you identify the full set of functional modules?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

12. Can you determine development relationships between these modules?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

12.1 Can you identify runtime dependencies between these modules?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

13. Can you identify required hardware elements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

14. Does the reference architecture specifies how the communication with the outer environment will be treated?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

15. Is the reference architecture in conformance with the requirements document?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

16. Have the used domain data been clearly defined?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

Stage 2:

1. Does the reference architecture clearly state its stakeholders (including domain-specific stakeholders) and their concerns?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

2. Does the reference architecture clearly state potential stakeholders of an instantiated architecture?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

3. Do the selected viewpoints frame the concerns of all stakeholders (including domain-specific stakeholders)?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

4. Do the selected viewpoints frame the concerns of potential stakeholders for an instantiated architecture?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

5. Is the reference architecture consistent with domain's practices and mandated standards?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

6. Do the viewpoints include concerns that are not domain-specific stakeholder's concerns?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

7. For each viewpoint, are its models clear and well-defined? Do the models provide enough information for determining whether the concerns framed by the viewpoint have been satisfied?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

1. Can you show how you produced the list of RA stakeholders and their concerns. I do not understand this question.

2. Have the requirements, constraints, standards, and quality assurance policies of the domain been clearly defined?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

3. Is there awareness for different runtime resources which might affect quality standards used in the RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

4. Is there a clear description of what parts of the RA are fixed, and what parts are subject to instantiation in a concrete organization / context?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

5. Is there a clear description of potential contexts in which the RA is instantiated? (Can you describe potential contexts for instantiation of the reference architecture?)
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

6. Are there guidelines for how to instantiate the RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

7. Is there a description of how variable parts of the RA affect non-variable parts when instantiated in a concrete organization / context? (Can you describe how variable parts of the RA affect non-variable parts when instantiated in a concrete organization / context?)
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

8. Does the RA support evolution? If so, how? If not, why not?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: I consider that guidelines of how using this RA to evolve variability tools are needed.

9. How will the RA be introduced and integrated in organizations?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

10. Are there clear dependencies between quality attributes and the behavior of instantiated RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

11. Are specific architecturally significant requirements identified?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

12. Are there additional architecturally significant requirements that occur when instantiating the RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

Continuação das Respostas do Especialista E1

1. Are the domain goals the system must satisfy clearly articulated and prioritized?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

2. Is there traceability between the domain goals and the requirements the determin?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

3. Is there traceability between the domain goals and the technical solution ?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

4. What criteria are used to determine whether the RA is supporting the domain goals?

5. Is there a clear outline of how instances of the RA could look like?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

6. Are the threats of introducing the RA clearly documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

1. Does the RA description allow an estimate of the effort for implementing it?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

2. Is there a way to show the benefit of using the reference architecture?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

3. Does the reference architecture description show what parts can be implemented using OTS(?) or OSS(?) components?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

4. Can you determine development dependencies between different parts of the RA and existing IT landscapes?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

5. Can you lay out a schedule for RA implementation and integration?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

6. Can you tell if there are enough development resources?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

1. Can you identify the allowed and prohibited dependencies between parts of the RA?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Can you identify applicable architectural constraints, rules, principles, styles, patterns, etc. that can be used for RA implementation?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Can you determine approaches for error handling, resource management, human-computer interaction, data management and persistence, variation and variability?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

4. Can you determine what is likely to change and how it impacts the RA design?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

1. Do you understand what needs to be done to integrate the RA in an existing IT landscape?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Do you understand the dependencies of a RA and existing IT landscapes?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Do you understand the adaptation points of the RA? (question 2.13)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

1. For each partition of the RA, can you determine what is needed (e.g., data, special hardware, other units) to test it?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

2. For each partition of the RA, can you determine what constitutes test success criteria?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Is it possible to identify “representative” challenges that occur in a domain to test the RA?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

1. Does the RA description articulate “open decisions”?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Are potential inconsistencies in the RA description (and for instances) known and documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Are the test approaches and artifacts consistent with the RA description?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

4. Is the RA understandable for all involved stakeholders?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

5. Does the RA address the key issues of the domain?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

6. Is the RA applicable to the problem is claims to address?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

7. Is the RA complete with regard to domain requirements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

8. Is the RA buildable with regard to constraints of the domain and instantiation contexts?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

9. Is there a process to ensure conformance with quality attribute requirements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

1. If the RA is part of a life cycle or process that includes a procurement decision, does the RA contain the appropriate information to support the procurement process?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Do the customers/acquirers have the right information to understand the key decision and how that decision meets the system requirements and constrains the design and implementation of the system?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Step 3. Conclusion / Final Analysis

1. Can you determine what is likely to change?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

1.1 Can you determine how does it impact your design?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Is the current document complete in the sense that all information is documented? If not, are there placeholders for what has yet to be documented along with descriptions of what still needs to be worked out?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Is it feasible that the views drawing upon viewpoints can be instantiated in concrete architectures, within the time and funding available?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Respostas do Especialista E2

1. General Information
1. The reference architecture presents:
i) overview information
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
ii) release date
Yes ( ) No (x) I am not expert to answer this question ( )
iii) version
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
I find a bit odd that all views were simultaneously created. This makes it difficult to see how the description envolved.
iv) owner (e.g. organization)
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
v) change history
Yes (x) No ( ) Partially () I am not expert to answer this question ( )
vi) short description
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
vii) scope
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
viii) domain terminology
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
xii) open decisions, and
Yes (x) No ( ) Partially () I am not expert to answer this question ( )
xiii) supporting material.
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
The authors make some references that are missing the actual document.

2. Are the concepts underlying the reference architecture document clearly explained ?
Yes (x) No ( ) Partially () I am not expert to answer this question ( )

3. Are all the selected stakeholders identified?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

4. Are all the selected stakeholders' concerns identified?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Concerns are generally defined (they are not connected to a particular stakeholder)
5. Does the reference architecture present the potential risk to its stakeholders through its life cycle?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: Although some risks are identified, they are shown together with interests. How these risks impact the lifecycle of the RA is not clear.

Questions referring to viewpoints
6. All selected viewpoints for the reference architecture were considered?
Yes () No (x) Partially ( ) I am not expert to answer this question ( )
It is not possible to know that because an initial set of viewpoints is not presented. Otherwise, there IS at least one view of each viewpoint.

6.1. If yes, there is a description of each viewpoint used in the reference architecture document?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

6.2. Does each viewpoint description include:
i) Viewpoint name;
Yes (x) No ( ) I am not expert to answer this question ( )
ii) Identification of the stakeholders addressed by that viewpoint;
Yes () No ( ) Partially (x) I am not expert to answer this question ( )
v.4 stakeholders are not completely defined.
iii) The architectural concerns addressed by that viewpoint; and
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iv) The architectural views used by the viewpoint.
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Questions referring to views
7. Does each view contains:
i) An identifier;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
ii) Introductory and additional information;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iii) Know incompatibilities related to its viewpoint;
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
It is not possible to determine it.
iv) One or more architectural models;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Questions referring to architectural models
8. Does each model contain:
i) Version identification; and
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
9. Does each model use well known notation (e.g., SysML and UML)
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Design decision view uses ad-hoc notation and text.

2. Construction and Content
1. Does the reference architecture satisfy:
i) legal regulation;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
ii) international standards;
Yes () No ( ) Partially (x) I am not expert to answer this question ( )
ISO 1471 has been replaced by ISO 42010
iii) company policies and rules; and
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
iv) Best practices and guidelines.
Yes (x) No () Partially ( ) I am not expert to answer this question ( )

2. Does the reference architecture presents:
i) Feasibility to derive it in instances;
Yes ( ) No () Partially (x) I am not expert to answer this question ( )
An instance architecture conforming of this RA is not shown.
iii) Maintainability related issues;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
The purpose of the RA is to cope better with change, including evolution and maintenance.
3. Does every viewpoint description contain only the required information (i.e., there's no useless information in the viewpoint definition)?
Yes (x) No () Partially ( ) I am not expert to answer this question ( )

4. Does the detail level favors the reference architecture understanding?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

5. Is the abstraction level consistent with the goals of the reference architecture?
Yes (x) No ( ) Partially () I am not expert to answer this question ( )
6. Does each view correctly represent its viewpoint?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
The runtime viewpoint is partially represented in the process view. Information about functional elements (e.g., responsibilities interfaces and primary interactions are missing).
7. Do the diagrams address all the concerns framed by the viewpoint?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Due to the reasons commented previously.
Domain Expert:
8. Have the following quality attributes have been considered:
i) Availability
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
ii) Interoperability
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iii) Maintainability
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: Only superficial information is provided. Not applicable in practice as is.

Continuação das Respostas do Especialista E2

v) Performance
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
v) Security
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
vi) Reliability
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
vii) Scalability
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
viii) Protection
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
ix) Usability
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

9. Does the reference architecture specifies which are the variability mechanisms?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Questions referring to correspondence rules
10. Are there correspondence rules?
Yes (x) No ( ) I am not expert to answer this question ( )

10.1 For each such rule, is there at least one correspondence satisfying each rule?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

10.2 Are all the correspondences identified and documented?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ()

10.3 Are all the correspondences identified in its participating elements?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ()
Some correspondences are hinted (e.g., binding time, binding constraints, variants, and cardinality) but they are not explored in al related views.

11. Can you identify the full set of functional modules?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

12. Can you determine development relationships between these modules?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

12.1 Can you identify runtime dependencies between these modules?
Yes (x) No () Partially ( ) I am not expert to answer this question ( )

13. Can you identify required hardware elements?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

14. Does the reference architecture specifies how the communication with the outer environment will be treated?
Yes () No (x) Partially ( ) I am not expert to answer this question ( )

15. Is the reference architecture in conformance with the requirements document?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
Comments: I have seen no requirements documents so I cant answer this question.

16. Have the used domain data been clearly defined?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

Stage 2:
1. Does the reference architecture clearly state its stakeholders (including domain-specific stakeholders) and their concerns?
Yes () No ( ) Partially (x) I am not expert to answer this question ( )
Stakeholders of the RA and its instances are not distinguished from each other.
2. Does the reference architecture clearly state potential stakeholders of an instantiated architecture?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

3. Do the selected viewpoints frame the concerns of all stakeholders (including domain-specific stakeholders)?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
For some stakeholders, only the description of their role is presented. Thereby, not all concerns are framed.

4. Do the selected viewpoints frame the concerns of potential stakeholders for an instantiated architecture?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
For some stakeholders, only the description of their role is presented. Thereby, not all concerns are framed.

5. Is the reference architecture consistent with domain's practices and mandated standards?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
ISO 25010 is not used to refer the quality characteristics. There is an outdated to the iso 1471, which has been replaced by ISO 42010
6. Do the viewpoints include concerns that are not domain-specific stakeholder's concerns?
Yes (x) No () Partially ( ) I am not expert to answer this question ( )
All concerns framed in this description are general.
7. For each viewpoint, are its models clear and well-defined? Do the models provide enough information for determining whether the concerns framed by the viewpoint have been satisfied?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
The conceptual view is not clear because it covers activities, concepts, and artifacts.
1. Can you show how you produced the list of RA stakeholders and their concerns.
--
2. Have the requirements, constraints, standards, and quality assurance policies of the domain been clearly defined?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Requirements OK, constraints are not clear, some standards are missing, explicit policies are not defined.
3. Is there awareness for different runtime resources which might affect quality standards used in the RA?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

4. Is there a clear description of what parts of the RA are fixed, and what parts are subject to instantiation in a concrete organization / context?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
All of them should be instantiated (apart from the Repository)
5. Is there a clear description of potential contexts in which the RA is instantiated?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

6. Are there guidelines for how to instantiate the RA?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: The variability view helps in that sense.

7. Is there a description of how variable parts of the RA affect non-variable parts when instantiated in a concrete organization / context?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
8. Does the RA support evolution? If so, how? If not, why not?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: The module variability evolution supports impact analysis and trade-off.
9. How will the RA be introduced and integrated in organizations?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
Comments: It can be used as guide that shows which activities are needed to support variability.

10. Are there clear dependencies between quality attributes and the behavior of instantiated RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: It would be possible to infer that from the runtime viewpoint, but the process view only covers the relationship between the activities. .

Continuação das Respostas do Especialista E2

11. Are specific architecturally significant requirements (i.e., the subset of functional, quality attribute, and business requirements that “shape” the reference architecture) identified?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Integrability, interoperability, maintenance.
12. Are there additional architecturally significant requirements that occur when instantiating the RA?
Yes (x) No () Partially ( ) I am not expert to answer this question ( )
Comments: performance.

Domain expert:
1. Are the domain goals the system must satisfy clearly articulated and prioritized?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

2. Is there traceability between the domain goals and the requirements the determin?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

3. Is there traceability between the domain goals and the technical solution (i.e., is it possible to navigate from domain goals to architecturally significant requirements, to technical decisions, and finally back to implications on achieving the domain goals)?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

4. What criteria are used to determine whether the RA is supporting the domain goals?
---
5. Is there a clear outline of how instances of the RA could look like?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

6. Are the threats of introducing the RA clearly documented?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

1. Does the RA description allow an estimate of the effort for implementing it?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

2. Is there a way to show the benefit of using the reference architecture?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

3. Does the reference architecture description show what parts can be implemented using OTS(?) or OSS(?) components?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

4. Can you determine development dependencies between different parts of the RA and existing IT landscapes?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

5. Can you lay out a schedule for RA implementation and integration?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

6. Can you tell if there are enough development resources?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

1. Can you identify the allowed and prohibited dependencies between parts of the RA?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Can you identify applicable architectural constraints, rules, principles, styles, patterns, etc. that can be used for RA implementation?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Can you determine approaches for error handling, resource management, human-computer interaction, data management and persistence, variation and variability?
Yes (x) No () Partially ( ) I am not expert to answer this question ( )

4. Can you determine what is likely to change and how it impacts the RA design?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

1. Do you understand what needs to be done to integrate the RA in an existing IT landscape?
Yes ( ) No ( ) Partially () I am not expert to answer this question (x)

2. Do you understand the dependencies of a RA and existing IT landscapes?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

3. Do you understand the adaptation points of the RA? (question 2.13)
Yes ( ) No ( ) Partially () I am not expert to answer this question (x)

1. For each partition of the RA, can you determine what is needed (e.g., data, special hardware, other units) to test it?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

2. For each partition of the RA, can you determine what constitutes test success criteria?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

3. Is it possible to identify “representative” challenges that occur in a domain to test the RA?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

1. Does the RA description articulate “open decisions”?
Yes ( ) No () Partially ( ) I am not expert to answer this question (x)

2. Are potential inconsistencies in the RA description (and for instances) known and documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

3. Are the test approaches and artifacts consistent with the RA description?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

4. Is the RA understandable for all involved stakeholders?
Yes ( ) No ( ) Partially () I am not expert to answer this question (x)

5. Does the RA address the key issues of the domain?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

6. Is the RA applicable to the problem is claims to address?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

7. Is the RA complete with regard to domain requirements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

8. Is the RA buildable with regard to constraints of the domain and instantiation contexts?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

9. Is there a process to ensure conformance with quality attribute requirements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

1. If the RA is part of a life cycle or process that includes a procurement decision, does the RA contain the appropriate information to support the procurement process?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

2. Do the customers/acquirers have the right information to understand the key decision and how that decision meets the system requirements and constrains the design and implementation of the system?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

Continuação das Respostas do Especialista E2

Step 3. Conclusion / Final Analysis
1. Can you determine what is likely to change?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Notations, binding time, tools
1.1 Can you determine how does it impact your design?
Yes ( ) No () Partially (x) I am not expert to answer this question ( )

2. Is the current document complete in the sense that all information is documented? If not, are there placeholders for what has yet to be documented along with descriptions of what still needs to be worked out?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: More information should be added in regards to the runtime view and process view. However, there are no placeholders for it.

3. Is it feasible that the views drawing upon viewpoints can be instantiated in concrete architectures, within the time and funding available?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
They can certainly be instantiated, but information is still missing.

Respostas do Especialista E3

1. General Information

1. The reference architecture presents:
i) overview information
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
ii) release date
Yes (x) No ( ) I am not expert to answer this question ( )
iii) version
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iv) owner (e.g. organization)
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
v) change history
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
vi) short description
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
vii) scope
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
viii) domain terminology
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
xii) open decisions, and
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: There are some decisions left to the user, however the information is kind of scattered throughout the text. A dedicated session explaining such elements would be valuable.
xiii) supporting material.
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

2. Are the concepts underlying the reference architecture document clearly explained ?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: Some diagrams are hard to understand.

3. Are all the selected stakeholders identified?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: It is important to make it clear who is responsible for each view and viewpoint. A summary (e.g., a table) would be helpful.

4. Are all the selected stakeholders' concerns identified?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

5. Does the reference architecture present the potential risk to its stakeholders through its life cycle?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: There is a general description on the risks someone could face, but the information should be more specific, i.e., a list of risks, and mitigation plans, for each particular view or viewpoint; then it would be possible to sketch a relationship between risks and their associated life cycle "phases".

Questions referring to viewpoints

6. All selected viewpoints for the reference architecture were considered?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

6.1. If yes, there is a description of each viewpoint used in the reference architecture document?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

6.2. Does each viewpoint description include:
i) Viewpoint name;
Yes (x) No ( ) I am not expert to answer this question ( )
ii) Identification of the stakeholders addressed by that viewpoint;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iii) The architectural concerns addressed by that viewpoint; and
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iv) The architectural views used by the viewpoint.
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Questions referring to views

7. Does each view contains:
i) An identifier;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
ii) Introductory and additional information;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iii) Know incompatibilities related to its viewpoint;
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
iv) One or more architectural models;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Questions referring to architectural models

8. Does each model contain:
i) Version identification; and
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
9. Does each model use well known notation (e.g., SysML and UML)
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Construction and Content

1. Does the reference architecture satisfy:
i) legal regulation;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
ii) international standards;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iii) company policies and rules; and
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
iv) Best practices and guidelines.
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Only a brief description about "what", without detailing "how". It doesn't suffice.

2. Does the reference architecture presents:
i) Feasibility to derive it in instances;
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
iii) Maintainability related issues;
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

3. Does every viewpoint description contain only the required information (i.e., there's no useless information in the viewpoint definition)?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: There is too much text. Maybe a Table summarizing every important concern would suffice.

4. Does the detail level favors the reference architecture understanding?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Please read the answer for the previous question.

5. Is the abstraction level consistent with the goals of the reference architecture?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

6. Does each view correctly represent its viewpoint?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

7. Do the diagrams address all the concerns framed by the viewpoint?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

Domain Expert:

8. Have the following quality attributes have been considered:
i) Availability
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Only superficial information is provided. Not applicable in practice as is.
ii) Interoperability
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Only superficial information is provided. Not applicable in practice as is.
iii) Maintainability
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Only superficial information is provided. Not applicable in practice as is.

Respostas do Especialista E3

iv) Performance
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Only superficial information is provided. Not applicable in practice as is.
v) Security
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Only superficial information is provided. Not applicable in practice as is.
vi) Reliability
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
vii) Scalability
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
viii) Protection
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
ix) Usability
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

9. Does the reference architecture specifies which are the variability mechanisms?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: Do you mean variability activities?! If so, the answer is YES. Regarding variability mechanisms, the answer is no. Indeed, what is surrounded by parenthesis does not refer to mechanisms, as widely accepted in the literature.

Questions referring to correspondence rules

10. Are there correspondence rules?
Yes (x) No ( ) I am not expert to answer this question ( )

10.1 For each such rule, is there at least one correspondence satisfying each rule?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

10.2 Are all the correspondences identified and documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

10.3 Are all the correspondences identified in its participating elements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

11. Can you identify the full set of functional modules?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

12. Can you determine development relationships between these modules?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

12.1 Can you identify runtime dependencies between these modules?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

13. Can you identify required hardware elements?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

14. Does the reference architecture specifies how the communication with the outer environment will be treated?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

15. Is the reference architecture in conformance with the requirements document?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
Comments: I have seen no requirements documents so I cant answer this question.

16. Have the used domain data been clearly defined?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

Stage 2:

1. Does the reference architecture clearly state its stakeholders (including domain-specific stakeholders) and their concerns?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

2. Does the reference architecture clearly state potential stakeholders of an instantiated architecture?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

3. Do the selected viewpoints frame the concerns of all stakeholders (including domain-specific stakeholders)?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

4. Do the selected viewpoints frame the concerns of potential stakeholders for an instantiated architecture?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

5. Is the reference architecture consistent with domain's practices and mandated standards?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

6. Do the viewpoints include concerns that are not domain-specific stakeholder's concerns?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

7. For each viewpoint, are its models clear and well-defined? Do the models provide enough information for determining whether the concerns framed by the viewpoint have been satisfied?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

1. Can you show how you produced the list of RA stakeholders and their concerns.
2. Have the requirements, constraints, standards, and quality assurance policies of the domain been clearly defined?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

3. Is there awareness for different runtime resources which might affect quality standards used in the RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
4. Is there a clear description of what parts of the RA are fixed, and what parts are subject to instantiation in a concrete organization / context?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

5. Is there a clear description of potential contexts in which the RA is instantiated?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

6. Are there guidelines for how to instantiate the RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: A running example should be helpful.

7. Is there a description of how variable parts of the RA affect non-variable parts when instantiated in a concrete organization / context?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

8. Does the RA support evolution? If so, how? If not, why not?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: Again, only a brief description about such a capability is provided; however, it is hard to understand how the instantiation could actually take place.

9. How will the RA be introduced and integrated in organizations?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: The "how" question cannot be answered with Yes/No answers. The point is "I don't know how one could instantiate. The same answer provided in the previous question may be applicable to this one.

10. Are there clear dependencies between quality attributes and the behavior of instantiated RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: QA are barely discussed in the text. It generally sound more as a process than practical discussion, thus including QA-related issues.

Respostas do Especialista E3

11. Are specific architecturally significant requirements (i.e., the subset of functional, quality attribute, and business requirements that “shape” the reference architecture) identified?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

12. Are there additional architecturally significant requirements that occur when instantiating the RA?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Regarding instantiation, the discussion above detail my standpoint about instantiation.

Domain expert:
1. Are the domain goals the system must satisfy clearly articulated and prioritized?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
2. Is there traceability between the domain goals and the requirements the determin?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

3. Is there traceability between the domain goals and the technical solution (i.e., is it possible to navigate from domain goals to architecturally significant requirements, to technical decisions, and finally back to implications on achieving the domain goals)?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

4. What criteria are used to determine whether the RA is supporting the domain goals?
---
5. Is there a clear outline of how instances of the RA could look like?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

6. Are the threats of introducing the RA clearly documented?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

1. Does the RA description allow an estimate of the effort for implementing it?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

2. Is there a way to show the benefit of using the reference architecture?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: Documenting and following a plan is way better than an adhoc approach.

3. Does the reference architecture description show what parts can be implemented using OTS(?) or OSS(?) components?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

4. Can you determine development dependencies between different parts of the RA and existing IT landscapes?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: Considering your last diagram, it’s ok!

5. Can you lay out a schedule for RA implementation and integration?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: Based on your document? No, Indeed I have no hint about the order in which things should occur, I mean, what comes first, when reading the document.

6. Can you tell if there are enough development resources?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)
I don’t know if such questions refer to either my daily work or your document. At this point in this too long survey, I get confused. Maybe my answers are not useful.

1. Can you identify the allowed and prohibited dependencies between parts of the RA?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

2. Can you identify applicable architectural constraints, rules, principles, styles, patterns, etc. that can be used for RA implementation?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: Identifying? Ok! I can. However, there is too much not sequential information, that makes understanding a hard task.

3. Can you determine approaches for error handling, resource management, human-computer interaction, data management and persistence, variation and variability?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

4. Can you determine what is likely to change and how it impacts the RA design?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

1. Do you understand what needs to be done to integrate the RA in an existing IT landscape?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

2. Do you understand the dependencies of a RA and existing IT landscapes?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

3. Do you understand the adaptation points of the RA? (question 2.13)
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

1. For each partition of the RA, can you determine what is needed (e.g., data, special hardware, other units) to test it?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

2. For each partition of the RA, can you determine what constitutes test success criteria?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

3. Is it possible to identify “representative” challenges that occur in a domain to test the RA?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )
Comments: Functional entities, ok! QA: not ok!

1. Does the RA description articulate “open decisions”?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

2. Are potential inconsistencies in the RA description (and for instances) known and documented?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

3. Are the test approaches and artifacts consistent with the RA description?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

4. Is the RA understandable for all involved stakeholders?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

5. Does the RA address the key issues of the domain?
Yes ( ) No ( ) Partially (x) I am not expert to answer this question ( )

6. Is the RA applicable to the problem is claims to address?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

7. Is the RA complete with regard to domain requirements?
Yes (x) No ( ) Partially ( ) I am not expert to answer this question ( )

8. Is the RA buildable with regard to constraints of the domain and instantiation contexts?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

9. Is there a process to ensure conformance with quality attribute requirements?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

Respostas do Especialista E3

1. If the RA is part of a life cycle or process that includes a procurement decision, does the RA contain the appropriate information to support the procurement process?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

2. Do the customers/acquirers have the right information to understand the key decision and how that decision meets the system requirements and constrains the design and implementation of the system?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

Step 3. Conclusion / Final Analysis
1. Can you determine what is likely to change?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

1.1 Can you determine how does it impact your design?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )

2. Is the current document complete in the sense that all information is documented? If not, are there placeholders for what has yet to be documented along with descriptions of what still needs to be worked out?
Yes ( ) No (x) Partially ( ) I am not expert to answer this question ( )
Comments: There is much information about the elements, but a process (step-by-step to follow) isn't clearly described nor an example on how to instantiate an architecture is provided. I guess this would be more interesting.

3. Is it feasible that the views drawing upon viewpoints can be instantiated in concrete architectures, within the time and funding available?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question (x)

Apêndice E - Documentação da Primeira Versão da VMTools-RA

O documento apresentado neste apêndice representa a primeira versão da VMTools-RA avaliada pelos especialistas no estudo qualitativo.

VMTools-RA

A Reference Architecture for Variability Management Tools

Owner

Ana Paula Allian, Master's degree student (UEM), Maringá, Paraná, Brazil
Prof. Dr. Edson OliveiraJr, *Professor of Software Engineering* (UEM), Maringá, Paraná, Brazil
Prof. Dr. Elisa Yumi Nakagawa, Dept. of Computer Systems, (USP-ICMC), São Carlos, São Paulo, Brazil.
Document Creation: 26/10/2015 – Version 1.0.

Overview Information

This document presents VMTools-RA, a reference architecture for variability management tool. More specifically, it contains the architecture requirements, architecture viewpoints, architecture views, and architecture description of VMTools-RA. Variability Management is the most central concept of product line approach and is also, a key differentiator between single-system engineering and software product line engineering. Variability management must be defined, modeled, implemented, versioned, verified, and validated. It must also be traced within and across domain and application engineering life cycles.

Goal

The main goal of VMTools-RA is to provide a general structure for supporting the development of new variability management tools.

Scope

The scope of VMTools-RA can be defined by considering the set of tools that are intended to be produced based on this reference architecture. These tools can be Variability Management Tools with or without Solvers, Variability Modeling Tools, Variability Product Derivation Tools and Variability Configuration Tools. Besides, all these tools can be developed considering the organizational elements that will guide for specific needs from different domains and market segments.

Needs

VMTools-RA covers the necessity of integration to domain analysis and or requirement specification tools in order to acquire domain assets and composition rules from the business context. Besides, VMTools-RA provides organizational elements that can attend the evolution need of variability management tool by getting feedbacks from stakeholders.

Risks

VMTools-RA provides some risk, limitations, and constraints that must be considered when designing an architecture for a variability management tool:

- **Lack of skilled stakeholders**: Stakeholders must be skilled in current and promising technologies; they need to know the domain application, modern design techniques and tool support, and professional practices.
- **Choice of wrong variability mechanism(s):** The VMTools-RA gives the opportunity for stakeholders to choose or implement their own variability mechanism. This task needs close attention, because a wrong choice can result in components that cannot be tailored when necessary.
- **Choice of irrelevant assets:** The VMTools-RA provides mechanism to acquire domain assets; however, the selection of the correct assets depends on the domain specialist knowledge.
- **Lack of tool support:** VMTools-RA gives the opportunity of integration of to domain analysis and requirement specification tools; however, this interoperability can failure. A contingency plan must be elaborated by the organization to minimize such risk by developing their own domain analysis or requirement specification tools, or perhaps adapting spreadsheets or databases to manage this information.
- **Organizational culture:** Stakeholders can avoid the acceptance of VMTools-RA due to organization's values, behaviors, and unwritten rules. Without a clear articulation of the roles, responsibilities, and day-to-day operating procedures, any organizational structure will fail to accept this reference architecture.
- **Lack of training:** If the training plan is not coordinated with the reference architecture, the staff will probably become frustrated and overwhelmed.

VMTools-RA Requirements

We identify 21 requirements, as it follows:

1. The RA must make possible to manage the domain assets (e.g., features, architectural components, requirements, etc) by collecting variability information (e.g., variation points and variants, binding time, dependencies, and constraints) elicited and analyzed during domain analysis and/or requirement analysis to make variability models.
2. The RA must make possible to build variability models independent from the variability approach selected to represent and manage the variability.
3. The RA must make possible for variability models to consider the variability composition rules (e.g., cardinalities, variability dependencies rules and constraints) derived during domain and/or requirement analysis.
4. The RA must make possible to manage, to establish, and maintain trace links (traceability) of variability models with domain and/or requirement assets.
5. The RA must make possible to document variability models in detail by annotating the elements of the variability models (e.g., variation points, variants, dependencies, and constraints), and maintains policies for forward and backward traceability links between variability models and associated domain assets for maintaining consistencies.
6. The RA must make possible to validate the conformance among variability models and its annotations by providing a function for an automatic consistency check (e.g., by using arithmetic constraint check, reasoners, logic solvers, etc).
7. The RA must make possible to generate reports for checking conflicts and inconsistencies within the variability model in order to validate the results.
8. The RA must make possible the variability binding management that maintains information necessary to resolve variability appropriately. It includes binding time, binding constraints, and its rationale for why a specific variant has been bound. (NOTE the choice of binding time is independent from variability modeling. It is consequence of decisions made from requirements through runtime).
9. The RA must make possible to select a variability mechanism which is a representation/implementation technique for implementing variability in accordance with its binding time at the specific life cycle stage (e.g. requirements, design, realization, compilation, post-compile time, runtime and testing). This mechanism allows delaying the decisions concerning the variants to choose during the development lifecycle that is optimized overall business goals.
10. The RA must make possible to manage variability control and evolution that support change requests/feedbacks for variability models, improving variability evolution process.
11. The RA must make possible to implement impact analysis to determine what changes are required to resolve the variability mismatches, and what the effects of these changes are on the software product line.
12. The RA must make possible to deal with the organization's capabilities for initiation, execution, control, and improvement of variability management. Planning and management control serves to define and maintain the organization's policies and norms, measure the performance of processes and if necessary, planning corrective actions to resolve problems.
13. The RA must make possible to support communication and collaboration work environment for variability management knowledge sharing regardless of geographical location.
14. The RA must make possible to provide guidance (e.g. providing web-based checklist) to support verifying the variability models and its composition rules, traceability, binding time, etc.
15. The RA must make possible to get feedbacks (or providing feed-forwards) from users (e.g. Web-based environment for collecting opinions about resources), notify users about variability management changes.
16. The RA must make possible to support trade-off analysis among alternatives of variability binding time, measurements for the flexibility and reusability of a variability mechanism.
17. The RA must make possible to manage and store multiple versions of variability models, variability documentation, etc.
18. The RA must make possible to support import/export of variability assets and relevant information related to variability from repositories and other tools available.
19. The RA must make possible to access and browse repository of variability information, persist variability models, and maintaining old trace links to enable roll-backing, when necessary.
20. The RA must make possible to offer an efficiency usability (for the user to achieve good productivity) with multiple graphical user interface views to present variability models, variability information, zoom, queries, etc.
21. The RA must make possible the option of integrating domain analysis and requirement specification tools through a middleware to obtain candidates assets that will be used in variability management.

Architectural Design

All the architectural requirements elicited are used to be the base for the architectural design realization. For a better documentation of the reference architecture, it will be presented the objectives, stakeholders, interests, and an overview representation of VMTools-RA divided in layers. Furthermore, it will be presented six architectural views distributed in four architectural viewpoints. An architectural view is a representation of a whole system from the perspective of a related set of concerns. An architectural viewpoint addresses the activities of the creation, analysis, and sustainment of architectures of software-intensive systems, and the recording of such architectures in terms of architectural descriptions. Architectural view and architectural viewpoint are recommended practices from IEEE 1471-2000 standard.

This document presents four architectural viewpoints and six architectural views, such as: *(i) Crosscutting viewpoint*: Conceptual view, Variability view, and Design Decision view; *(ii) Runtime viewpoint*: Process view; *(iii) Source Code viewpoint*: Module view; and *(iv) Deployment viewpoint*: Deployment view. For each architectural view it will be presented description, captured interests, and the reference architecture representation. We used UML to represent most of architectural views.

It is important to highlight that the concepts presented in the architectural views can be implemented with different technologies, programming languages, and different variability notation to represent variability models.

Objectives, Stakeholders and Interests

The main objectives for VMTools-RA are:

- Support the variability management tools development;
- Support the software reuse;
- Support the maintenance and evolution of variability management tools based on VMTools-RA;

The main stakeholders were defined based on the architectural requirements and VMTools-RA objectives:

Chief executive officer (CEO): Responsible for large productivity gains; greatly improved time to market; sustained growth and market presence; the options and ability to economically capture a market niche;
Chief operating officer (COO): Efficient use of workforce; ability to explore new markets, new technology, and/or new products; fluid personnel pool;
Executives: Responsible for the organization's business goals and constraints;
Marketers: Responsible for the market demands;
Technical managers: Responsible for the available personnel;
Architect: Responsible for identifying the needs of systems and further reference architecture instantiation.
Domain Specialist: Responsible for providing specific domain information and verifying whether the related requirements were attended.
Analyst: Responsible for validating the group of information from architectural instantiation;
QA Manager (Quality Assurance): Responsible for ensuring that the quality requirements are satisfied;
Developers: Responsible for the implementation of the instantiated architecture;
Maintainer: Responsible for performing any maintenance and upgrades in the developed system; and
User: Uses the system developed.

The main interests identified based on the stakeholders:

I1. Reuse: It is expected that VMTools-RA enables the efficient reuse and management of variability, assets, and cost saving by enhancing reusability;
I2. Conformance between reference architectures and instantiated architecture: It is expected that all the information available in this document enable maintain conformance between the reference architectures and instantiated architecture through the simplest instantiation step-by-step;
I3. Evolution: Besides the instantiation, it is expected that VMTools-RA facilitates the evolution of variability management tools, by modifying the structure of the systems.
I4. Technical Risk: It is expected that VMTools-RA deals with risk identification, assessment, prioritization, and mitigation to prevent failures that inhibit the achievement of business values and product line objectives.
I5. Market Analysis: It is expected that VMTools-RA deals with analysis of target markets, available technologies, offerings of competitors, and other factors.

It is important to highlight that is impossible to identify all the objectives, stakeholders and interests from all the architectural instances.

VMTools-RA Overview

The VMTools-RA overview (Figure 1) shows the main components, relationship between them, and dependencies with hardware. It can be observed five elements: *(i)Variability Management Elements*; *(ii)Domain Analysis Elements*; *(iii)Domain Requirements Elements*; *(iv)Support Elements;* and *(v)Organizational Elements*. Furthermore, it can be observed a *Middleware* layer where optional tools can be integrated. The *Repository* element can provide mechanisms to persist the application information. These elements are organized in layers where the lower layers (*Domain Analysis Elements*, *Domain Requirements Elements, Middleware)* can support the upper layer. The *Support Elements* layer and *Organizational Elements* layer acts like crosscuts elements, and can modify *Variability Manager Elements, Domain Analysis* and *Domain Requirements Elements*, in order to insert or remove functionalities. The lower layers can acquire domain assets, constraints, dependencies, and variability information for the upper layer (*Variability Management Elements*) where are implemented the solutions to manage the variability.

The *Variability Management Elements* are based on the reference model ISO/IEC 26555 for the technical management of software product line engineering. This International standard consists on the following sub-processes: *Variability Model, Variability Binding, Variability Documentation, Variability Tracing* and *Variability Control and Evolution.* The VMTools-RA can also satisfy best practices enabling documentation and versioning of variability, impact analysis, trade-off analysis, guidance, organizational standards, etc. The *Organizational Elements* from the reference architecture can be projected to support a particular industry standard that all team members conform to when instantiating a tool based on the VMTools-RA. Such organizational standards can include architecture and design patterns to be applied, modeling standards, and programming standards. Furthermore, it can include particular policies, rules, legal regulation to attend to specific market needs and contribute to the evolution of the reference architecture. Thus, it is expected that this overview promote the reuse, traceability and evolution of instantiated architecture based on VMTools-RA.

Captured Interests: I1, I2, I3 and I5.

Representation:

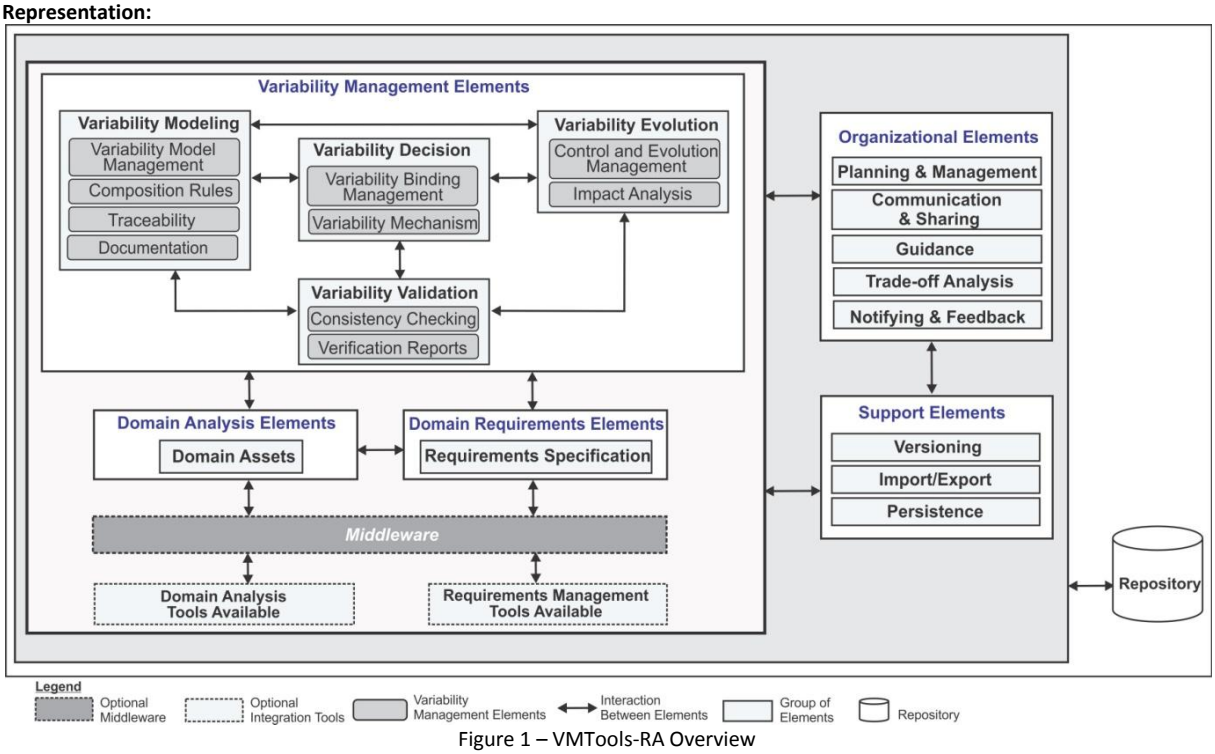


Figure 1 – VMTools-RA Overview

V.1 Crosscutting Viewpoint

This viewpoint shows general information about the reference architecture, such as terms/concepts and variabilities. It indicates that the information contained in is transversal to other viewpoints, and captures the knowledge necessary to create particular views aimed to represent complex systems in a way stakeholders can understand. Such viewpoint also provides the basis from which other viewpoints are going to be built and it is addressed for all stakeholders. This viewpoint addresses some concerns to stakeholders like information domain, reuse, terminologies and instantiated architecture decision. The history change of this viewpoint is presented in the table 1.

1.1 Conceptual View

This view presents a glossary of all the concepts that will be used in other views. Figure 2 shows the direct relationships between the concepts by using a UML Class Diagram.

Captured Interests: I2.
Representation:

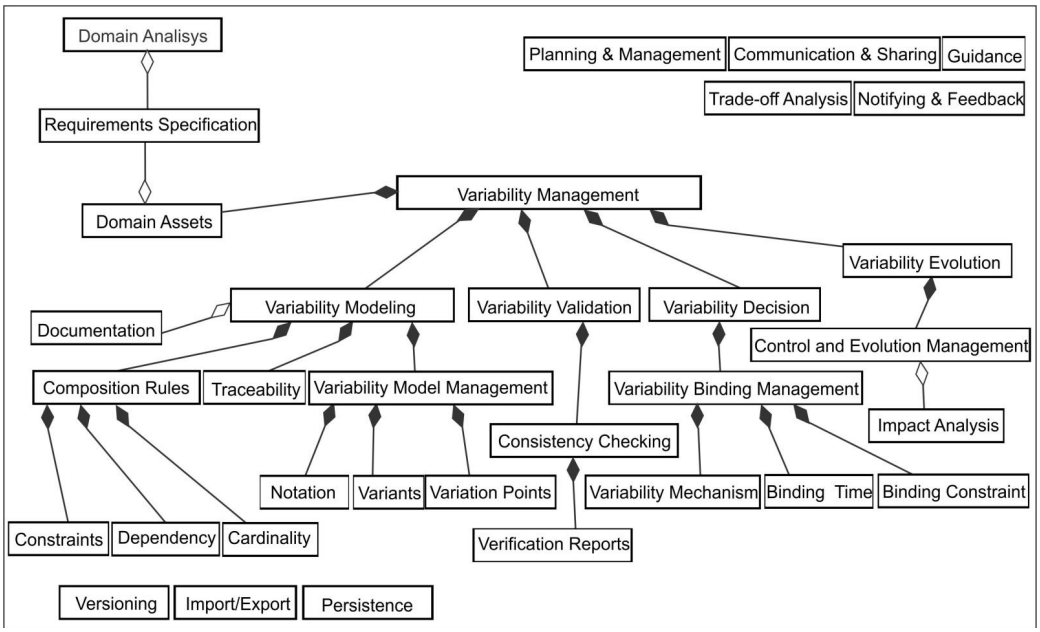


Figure 2 – VMTools-RA Conceptual View

Domain terms:

1. **Domain Analysis**: It is a process used to identify and document common and variable characteristics of systems in a specific domain.
2. **Domain Assets**: Reusable artifacts produced during domain analysis (features, domain models, domain requirements specification, domain architecture, domain components, domain test cases, and etc).
3. **Requirements Specification:** Documents domain requirements for a product line based on domain analysis results.
4. **Planning & Management**: Serves for establishing and managing a product line organization's capabilities of implementing product line processes.
5. **Communication & Sharing**: Servers to promote collaboration (communication) environments for knowledge sharing regardless of geographical location.
6. **Guidance**: Guide the decision execution, decision rationale, the measurement of decision effectiveness.
7. **Trade-off Analysis**: Analysis based on defined aspects: quality attributes or product line goals such as flexibility, maintenance cost, etc.
8. **Notifying & Feedback**: Promotes the improvement of variability models and the variability evolution process.
9. **Variability Management**: Defines how member of products in a product line can vary.
10. **Variability Modeling**: All the activities related to model the variability.
11. **Variability Model Management:** It serves to collect variability information for identifying variability elements and bases information for constructing variability models.
12. **Notations**: It is the approach used to represent and to model the variability.
13. **Variants:** It is one alternative that may be used to realize particular variation points.
14. **Variation Points:** Places in design artifacts where a specific decision has been narrowed to several options but the option to be chosen for a particular system has been left open.
15. **Composition Rules**: Create restrictions in the domain for representing and relating the features. They can be mutual exclusion and dependency, regular expressions, or artificial intelligence, among others.
16. **Dependencies**: It is a relationship between a variation point and a set of variants; it indicates that the variation point implies a decision about the variants.
17. **Constraints**: Denotes constraint relationships between a variant and a variation point, between two variants, and between two variation points.
18. **Cardinality:** Shows the number of features that must be selected. This cardinality is represented by a minimum and maximum number.

19. **Traceability**: Manages to establish and maintain trace links of variability models with domain and application assets according to the established policies for traceability management.
20. **Documentation**: Supports stakeholders to document variability models in detail with annotations.
21. **Variability Validation**: It is served for ensuring that the models consider variability dependencies and constraints derived during domain engineering.
22. **Consistency Checking**: Verifies if the generated domain follows the composition rules created.
23. **Verification Reports**: It should be able to provide information about the validation of variability.
24. **Variability Decision:** It aims to select the best option where alternatives exist.
25. **Variability Binding Management:** Maintains information that is necessary to resolve variability appropriately.
26. **Binding Time:** It is the time that the value of a variation point is determined. It can be compile time, link time, build time, or runtime.
27. **Binding Constraints:** Denotes constraint relationships during the binding management.
28. **Variability Mechanism:** It is a variability representation/implementation technique for the product line variability. It deals with variabilities based on the binding time at the specific life cycle stage.
29. **Variability Evolution**: Deals with change requests, change impact analysis, change execution, and verification/validation for the change.
30. **Control and Evolution Management:** Serves to manage changes of variability models and their established traceability.
31. **Impact Analysis**: To evaluate the consequences of a future change in the variability model.
32. **Versioning:** Maintain versions of variability models.
33. **Import/Export**: Importing and exporting the relevant information from/to other tools.
34. **Persistence**: Persist the information in a repository and providing a roll-back function for restore trace links to their former state.

1.2 Variability View

It describes elements of the reference architecture, and how they can be exercised to build instantiated architecture based on VMTools-RA. For example, it can show alternatives (or-exclusive relationship), optional elements, and mandatory elements. The Variability View (Figure 3) based on the Cardinality Based Feature Model (CBFM) is an extension of Feature Oriented Domain Analysis (FODA). The feature model is a hierarchical structure of decision making in software development, and facilitates the identification of reusable components. It is important to highlight that the variability view is a representation of VMTools-RA making use of Feature Models and does not represent a product line architecture.

The main activity *Variability Management* is the starting point (root element) of hierarchical tree. We can also consider two others trees: *Domain Analysis* and *Organizational*. The *Domain Analysis* tree is responsible for the identification of domain assets, and can be responsible for the requirements specification. The *Organizational* tree can deal with important policies and norms related to the company and stakeholders, and it requires domain analysis tree to plan the general organization structure to manage the variability.

The *Variability Management* tree supports four mandatory activities *Variability Modeling, Variability Validation, Variability Decision* and *Variability Evolution*. The *Variability Model Management* consists in three mandatory elements (*Notation*, *Variants* and *Variation Point*) that need to be implemented in order to get a variability model. The *Notation* activity requires an organization decision to decide which variability notation approach will be used to model the variability (feature model, decision model, own variability notation, etc). Such notation is represented in a *XOR* group (select exactly one). The *Consistency Checking* activity is implemented in an *OR* group (select at least one), because it is possible to use any arithmetic constraint check available (logic solvers, reasoners, etc), or it can be developed by the organizational or responsible stakeholder. The *Binding Time* activity is a mandatory element, and it is represented in an *OR* group (select at least one). The *Variability Mechanism* requires binding time to realize the variability. *Versioning*, *Import/Export* and *Persistence* activities represent the crosscuts elements in this view, and are not organized in any tree. This Feature Model View of VMTools-RA gives the opportunity for companies to analyze which elements are mandatory and options in a possible VMTools-RA instantiation.

Interests Captured: I1, I2, I3, I4 and I5.
Representation:

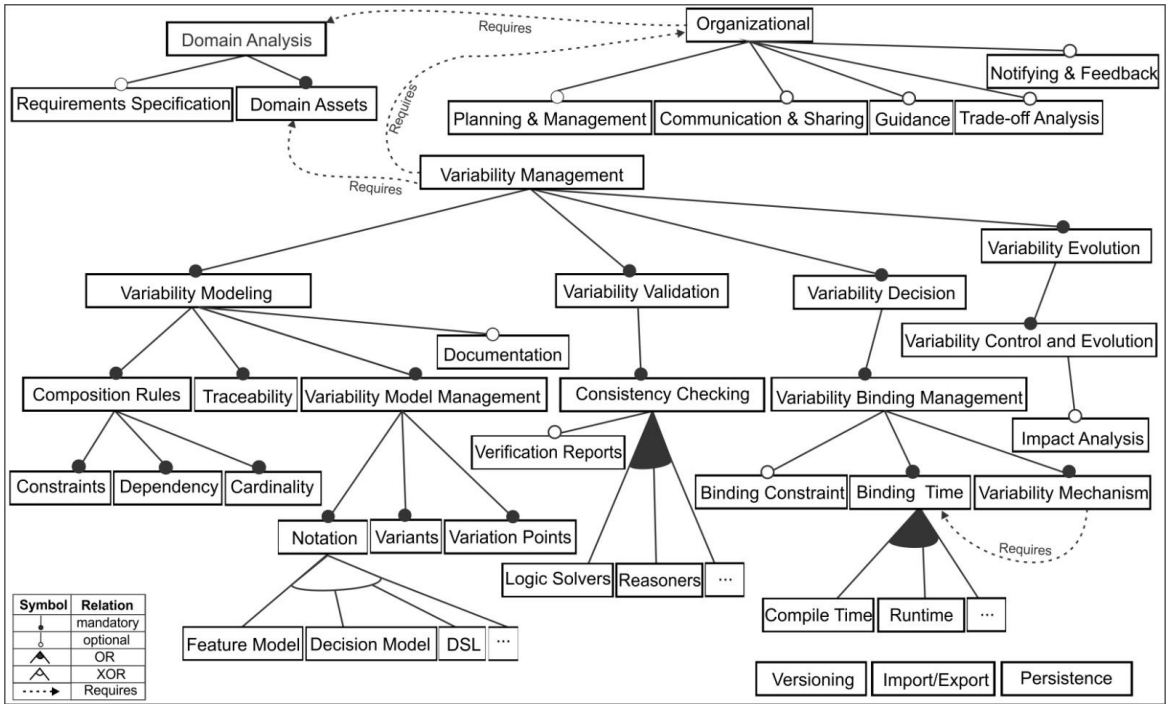


Figure 3 – VMTools-RA Variability View

1.3 Design Decision View

The Design Decision view captures and shares the knowledge of architectural decisions of reference architecture. This view is the result of the design process during initial construction or development of a software system. Garlan et al., (2010) suggest the use of templates to document the decisions. Such templates can make use of tables and graphical to represent architectural decisions. In this document, we represent one architectural design decision related to the integration element in the VMTools-RA. This design decision identifies two issues related to the integration. These issues were documented in the tables **a)** and **b)**, and represented in the Figure 4.

Interests Captured: I4.
Representation:

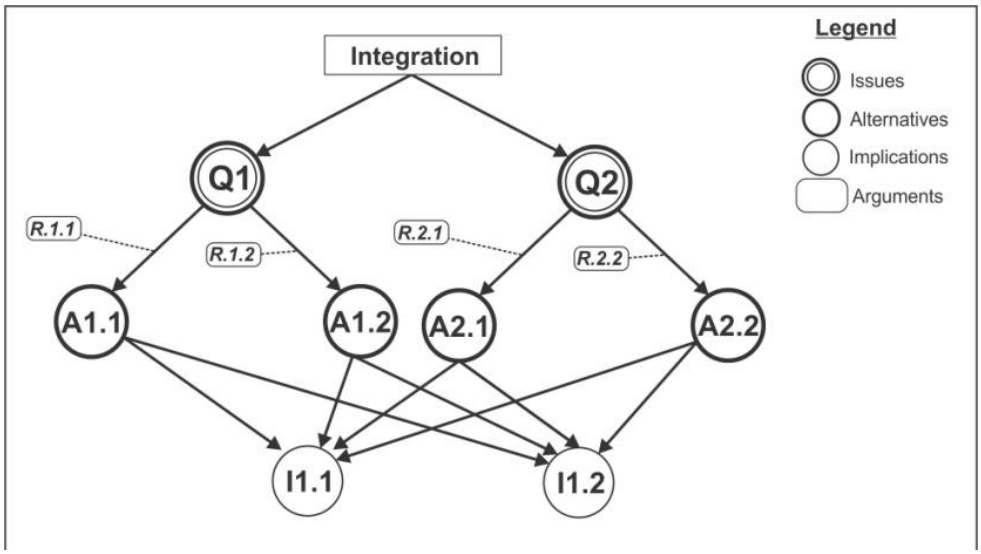


Figure 4 – VMTools-RA Design Decision View

Table a) - Decision Architectural Documentation – Integration (Q1)

Group	Integration
Issues	Q1. If existing tools get updated and the integration became not possible anymore?
Decision	D1. Instantiate and protect the relevant tool in the AR module allowing import of existing data.
Assumptions	S1. It is assumed that the available tools can be integrated and they will maintain its basic structure.
Alternatives	A.1.1 Check integration possibilities with other tools that have similar features. A.1.2. Develop own structure and import information for the application.
Reasoning	R.1.1. Integration with similar tool is possible, but requires adaptations. R.1.2. The development of the structure itself is possible as the VMTools-RA allows the instantiation of corresponding modules to domain and requirements analysis.
Implications	I.1.1. It requires negotiation with the owner of the external application to make changes in interfaces to allow integration. I.1.2. Requires additional training to stakeholders.

Table b) - Decision Architectural Documentation – Integration (Q2)

Group	Integration
Issues	Q2. Is it possible to integrate new tools to manage other variability management elements, like consistency check, evolution of variability, documentation, etc?
Decision	2. Make it possible for the middleware to accept both technologies over Internet/intranet protocols.
Assumptions	S2. It is assumed that the modules defined in the VMTools-RA already cover the full variability management process.
Alternatives	A.2.1 Make possible the import and export in the VMTools-RA. A.2.2. Make possible the instantiation of a generic module that addresses the different contexts of variability management.
Reasoning	R.2.1. Adaptations are required for the new module to meets the customer needs. R.2.2. Traceability and consistency of the information should be maintained with the integration of new tools.
Implications	I.1.1. It requires negotiation with the owner of the external application to make changes in interfaces to allow integration. I.1.2. Requires additional training to stakeholders.

CHANGE HISTORY:

Table 1 - Change history for Crosscutting Viewpoint

Viewpoint ID	Date:	Version:	Change history
V.1	Out. 2015	1.0	Creation

V.2 Runtime Viewpoint

The runtime viewpoint shows the dynamic behavior of systems (during their execution) that will be built based on the reference architecture. It describes the system's functional elements, their responsibilities, interfaces, and primary interactions. Besides, it describes abstractions of concrete software elements such as tasks, processes, or execution threads. It also has a significant impact on the system's quality properties, such as its ability to change, its ability to be secured, and its runtime performance. It is addressed for all stakeholders. This viewpoint addresses some concerns to stakeholders like system scope and responsibilities, evolution, and support. The history change of this view point is presented in the table 2.

2.1 Process View

The Process View represented as a UML Activity Diagram allows analyzing the variability management activities, and the inputs and outputs of each stage of the process. It presents the main realization lines of a system developed from VMTools-RA instantiation, indicating responsibilities and collaborations. The Figure 5 presents six realization lines, four of them are related to *Variability Management (Variability Model, Variability Validation, Variability Decision* and *Variability Evolution*), one related to *Domain Information* and one related to the *Organization Information.* The Middleware layer and the integrated tools are not represented in this view because they are part of domain information acquisition.

There are two processes that can be responsible for collecting information from the domain and from the organization. Such processes are characterized by cycles that can guarantee gain in knowledge and experience. The *Domain Information* process requires a *Domain Specialist* who can be responsible for domain analysis in order to identify and persists domain assets, variability information, constraints, and dependencies rules in the repository. The *Organizational* process requires *Executives*, *Marketers*, *Chief Executive Officer (CEO), Chief Operating Officer (COO),* and any *Stakeholder* who can contribute with feedbacks for the improvement of variability management process. Such *Organizational* process can be responsible for many tasks like: define policies and norms, promote a collaboration environment, define which variability notation will be used to model the variability, trade-off analysis, and other relevant information related to the organization group.

The *Variability Model* process can use information from the domain to model the variability regarding to its traceability, composition rules and variability information. The variability model process is completed when a variability model is concluded. The variability documentation can be created with all the information related to the variability model. This process can be managed by

Domain Specialist, *Architect* and *Developers*. The *Variability Validation* process can be responsible for the consistency of variability model. This validation assure if the variability models are consistent according to its composition rules, traceability, and documentation. If any inconsistency is found, the *Variability Validation* process guides the model back to the *Variability Model* process to force its correctness. It is completed when the variability model is consistent. This process can be managed by *Quality Assurance Manager*, *Analysts* and *Developers.*

Once the variability model is correct, it is possible to apply the *Variability Decision* process, and decide the variability resolution on relevant variants using variability model and organizational information. Such process can be managed by *Analyst*, *Executives*, *Marketers*, *Quality Assurance Manager* and *Developers*. Binding information and Binding time can significantly affect the flexibility, runtime performance, and production costs of product line members. Because these decisions are influential, alternatives of binding time must be reviewed carefully in the activity related to trade-off analysis, in order to guide among alternatives of binding time analysis and compares the results. To support the *Variability Decision* process, it is possible to use a variability mechanism to deal with representation/ implementation of variability. After the correct selection of binding decision and variability mechanism it is possible to generate products.

The *Variability Evolution* process can manage requests/feedbacks from *stakeholders;* manage versions of models and variability information, in order to assure *Maintainability* of the system to undergo changes when necessary. Besides, this process can be responsible for impact analysis and trade-off analysis. Approved changes are communicated to all affected participants, new information can be updated into the repository, and organizational policies can be modified. *Maintainer*, *Analyst*, *Marketers*, *Executives* are examples of stakeholders who could be responsible for this process.

Interests Captured: I1, I2, I3, I4 and I5.
Representation:

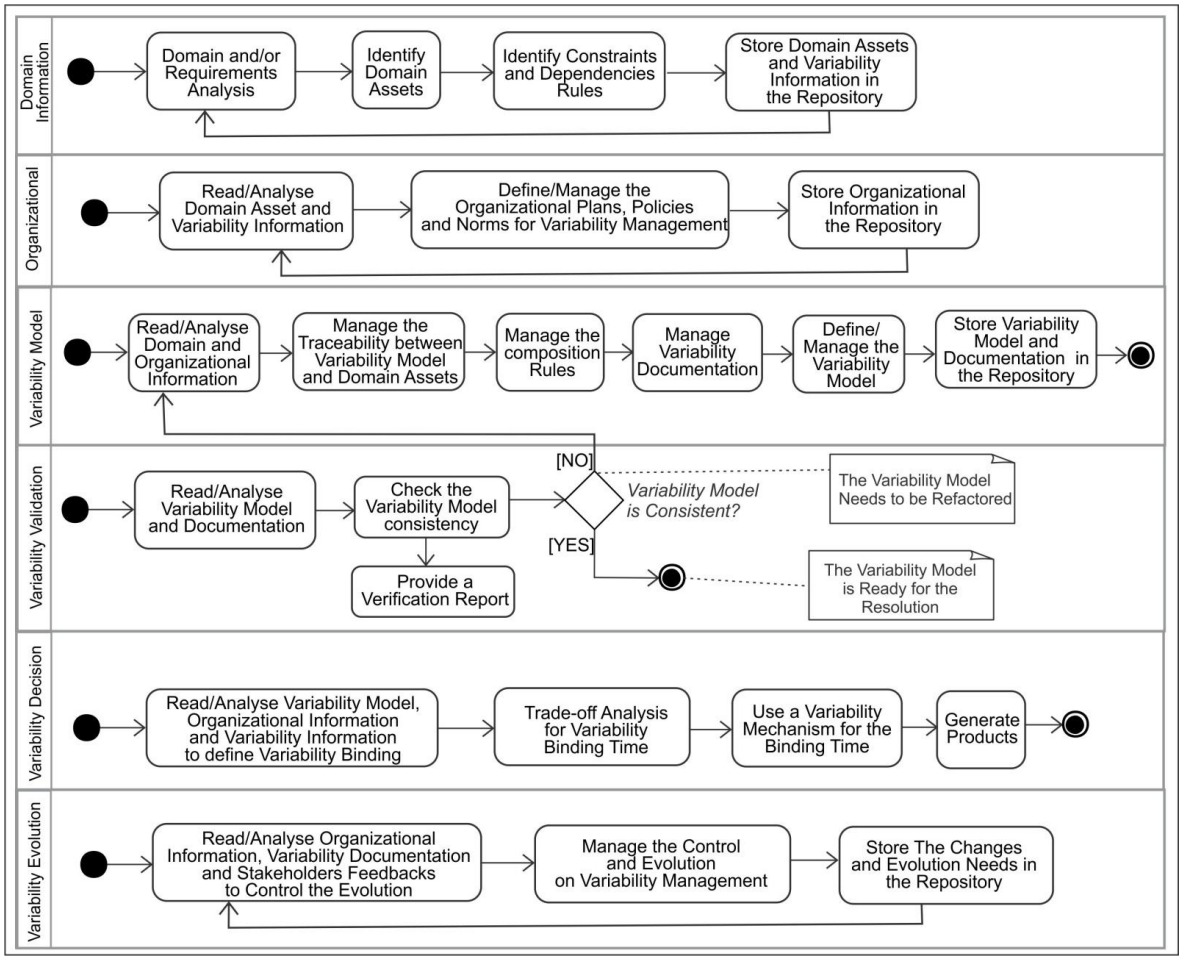


Figure 5 – VMTools-RA Process View

CHANGE HISTORY:

Table 2 - Change history for Runtime Viewpoint

Viewpoint ID	Date:	Version:	Change history
V.2	Out. 2015	1.0	Creation

V.3 Source Code Viewpoint

It shows specific details, such as software structures and modules, about the implementation of the systems resulted from the reference architecture. Describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts). For this viewpoint, we will present a Module View of VMTools-RA addressed especially for *architects* and *developers*. This viewpoint addresses some concerns to stakeholders like application-specific requirements, module organization and their realization. The history change of this viewpoint is presented in the table 3.

3.1 VMTools-RA Module View

The VMTools-RA module view, represented as a UML class diagram, shows modules containing specific functionality, and can be used to depict functional modules, the data flow among them, and the interfaces. It is represented by packages, sub packages, classes, interfaces and relationships. The Module View (Figure 6) presents the general structure of VMTools-RA and it shows five modules concern to main elements presented in previous views, *i) Variability Management Elements, ii)Organizational Elements, iii)Support Elements, iv)Domain Analysis Elements, v)Domain Requirement Elements.*

The *Domain Analysis* and *Domain Requirement* modules can be necessary to identify domain assets and composition rules. Such modules can use a *Middleware* to integrate to other related tools and acquire the information needed to save in a repository. This *Interoperability* provides communication and exchange information and can be followed by organizational policies and rules to ensure the *Availability* of such systems.

The *Organizational Elements* module can plan and manage all the organization needs related to the variability management. It can provide *Performance* of processes when planning corrective actions to resolve problems for fixing the deviations. This module can also prevents malicious actions outside of the designed usage and loss of information; it can ensure *Security* to the variability management by using *Notifying & Feedbacks* and *Guidance* sub-modules. *Reliability* is another requirement that can be attended by the *Organizational module;* it can provide a geographically distributed workspace for stakeholders by using the *Communication & Sharing* sub-module. Besides, the *Usability* of the system can be defined by the *Organizational module* or responsible stakeholder; it needs to meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience.

The *Variability Management* module is responsible to manage the variability. It consists in four sub-modules: *Variability Modeling*, *Variability Evolution*, *Variability Decision* and *Variability Validation*. The *Variability Modeling* supports developing and validating variability models by using variability information derived from the domain/requirement analysis. The *Variability Decision* maintains information that is necessary to resolve variability appropriately, and use the variability model, domain/requirement information and organizational functionalities, like guidance and trade-off analysis, to decide the best moment to realize the variability. The *Variability Validation* verifies the consistency of variability models related to its composition rules. The *Variability Evolution* manages changes of variability models and it considers *feedbacks* to ensure success in what is going to be modified. *Variability Evolution* ensures *Maintainability* ability of the system to undergo changes. These changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements. The *Variability Management* module use some support elements like *Versioning* that deals with different versions of variability models to be saved and restored when necessary. *Import/Export* information from/to other tools, and *Persistence* to provide roll-back functions, when necessary.

Interests Captured: I1, I2, I3, I4, I5.
Representation:

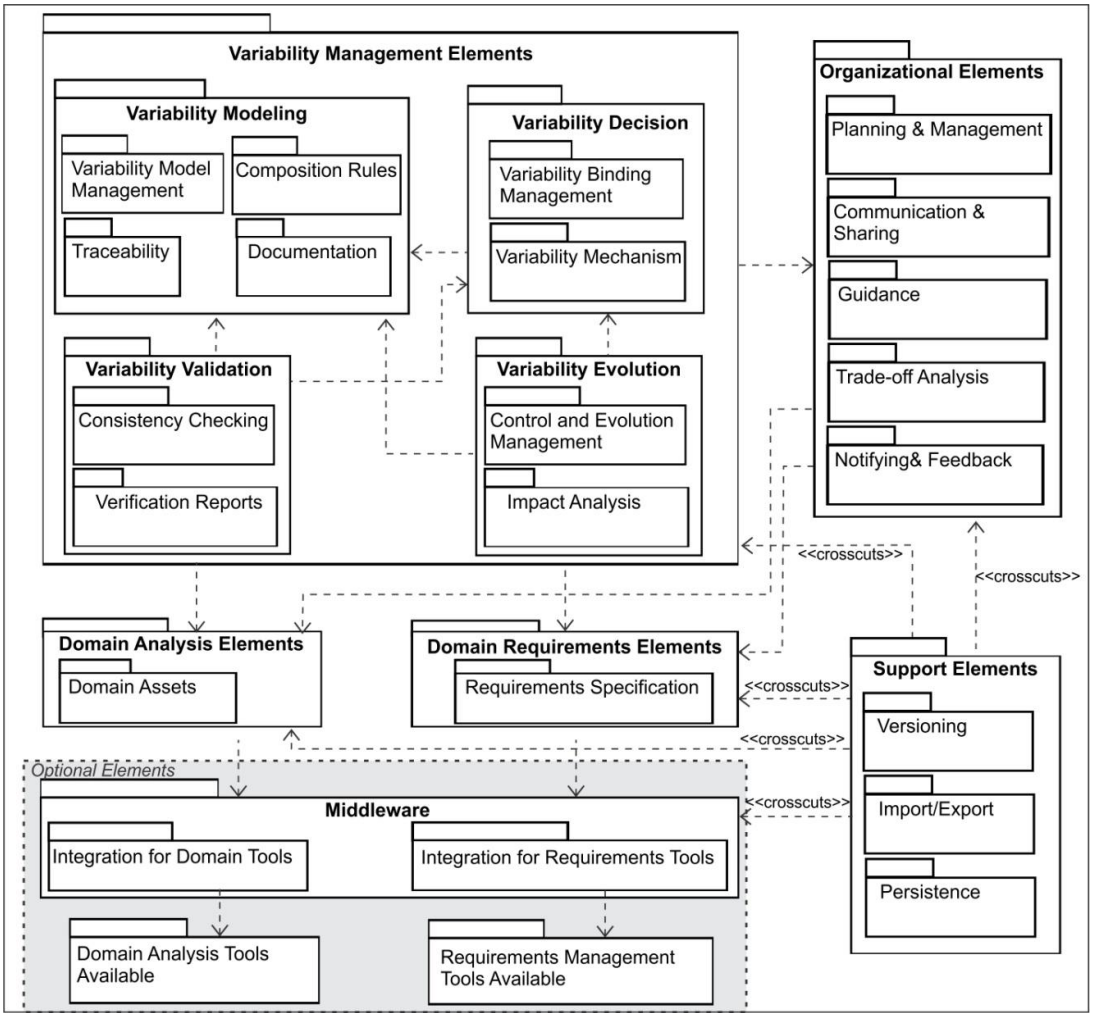


Figure 6 – VMTools-RA Module View

CHANGE HISTORY:

Table 3 - Change history for Source Code Viewpoint

Viewpoint ID	Date:	Version:	Change history
V.3	Out. 2015	1.0	Creation

V.4 Deployment Viewpoint

This viewpoint is concerned with those architectural elements that supports the distribution of the system and includes elements such as location, nodes, devices, and the connections among them. Some potentially stakeholder interested in this viewpoint are *developers, maintainers, architectures, etc.* This viewpoint addresses some concerns to stakeholders like system distribution, hardware node (and devices) specification, integration middleware, and the connection among them. The history change of this view point is presented in the table 4.

4.1 Deployment View

The deployment view presents the hardware components, application servers, database servers and client machines. This view describes the software system or subsystems that will be installed on this hardware. To represent this view we use UML deployment diagram. The Deployment View (Figure 7) presents six elements that can be connected using interfaces over internet / intranet protocols.

The *Application Server* can provide an integrated environment for deploying, and running the logic from the variability management, organizational, domain, requirements, and support elements. It is also possible to include in this infrastructure the integrated security, and integration technologies to manage organizational elements like feedbacks, trade-off analysis,

communication and collaboration to other stakeholders regardless of geographical location. The *Application Server* can be responsible for managing and persisting domain assets, variability information, and general information in the *Database Server*.

The *Middleware* is an option element that can be implemented to facilitate exchange of data between two or more application programs within the same environment, or across different hardware and network environments. Such element can integrate with existing domain analysis tools and requirements specification tools; they must follow the organizational norms and policies allowing *Availability* of the systems. The *Middleware* can use interfaces over internet/intranet protocols, and it can use some APIS like REST technologies, SOAP, RMI, Web Services, etc.

Interests Captured: I2, I3, I4 and I5.
Representation:

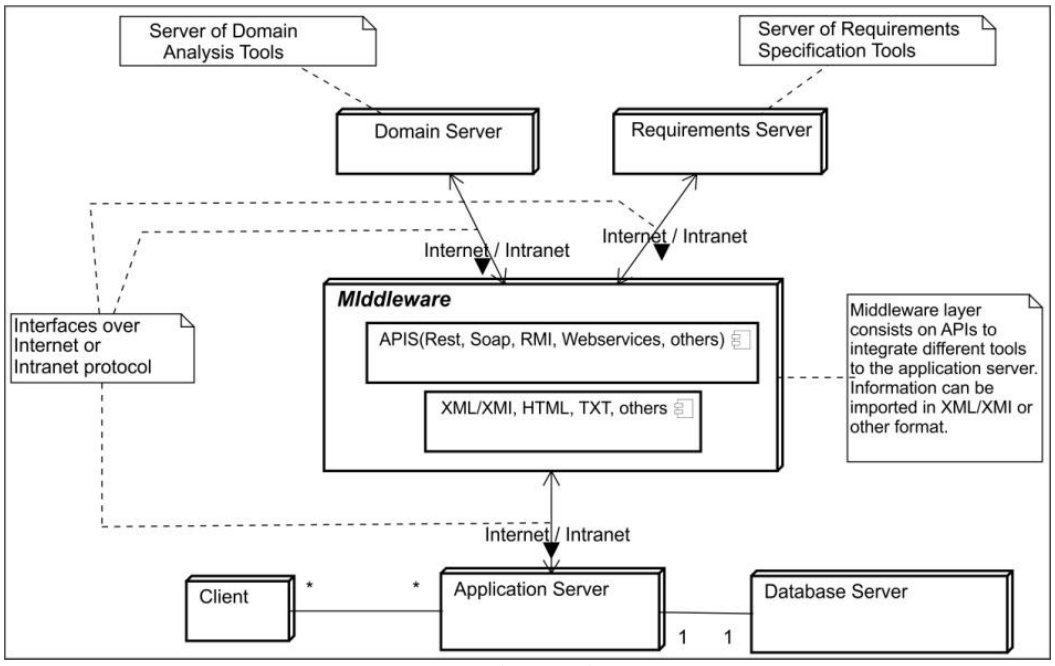


Figure 7 – VMTools-RA Deployment View

CHANGE HISTORY:

Table 4 - Change history for Deployment Viewpoint

Viewpoint ID	Date:	Version:	Change history
V.4	Out. 2015	1.0	Creation

Anexos

F.1 *Checklist* de Avaliação do FERA

Este Anexo apresenta o *checklist* do *framework* FERA utilizado para avaliação da VMTools-RA.

VMTools-RA Checklist

Checklist for the Evaluation of VMTools-RA, A Reference Architecture for Variability Management Tools

Step 1: Check for completeness

Purpose: This is the first step: to check whether the document is complete in the sense that all the required fields are filled, so then, the document is ready for a deeper evaluation.

1. General Information

Stakeholders: Architects

1. The reference architecture presents:

i) overview information

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

ii) release date

Yes ( ) No ( ) I am not expert to answer this question ( )

iii) version

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

iv) owner (e.g. organization)

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

v) change history

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

vi) short description

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

vii) scope

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

viii) domain terminology

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

ix) open decisions, and

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

x) supporting material.

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

2. Are the concepts underlying the reference architecture document clearly explained (e.g., they consistently use a domain terminology, labels for diagrams)?

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

3. Are all the selected stakeholders identified?

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

4. Are all the selected stakeholders' concerns identified?

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

5. Does the reference architecture present the potential risk to its stakeholders through its life cycle?

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

Questions referring to viewpoints

6. All selected viewpoints for the reference architecture were considered?

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

6.1. If yes, there is a description of each viewpoint used in the reference architecture document?

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____

6.2. Does each viewpoint description include:

i) Viewpoint name;

Yes ( ) No ( ) I am not expert to answer this question ( )
ii) Identification of the stakeholders addressed by that viewpoint;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iii) The architectural concerns addressed by that viewpoint; and
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iv) The architectural views used by the viewpoint.
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Questions referring to views
7. Does each view contains:
i) An identifier;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
ii) Introductory and additional information;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iii) Know incompatibilities related to its viewpoint;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iv) One or more architectural models;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Questions referring to architectural models
8. Does each model contain: i) Version identification;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
9. Does each model use well known notation (e.g., SysML and UML)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

2. Construction and Content

Questions

1. Does the reference architecture satisfy:
i) legal regulation;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
ii) international standards;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iii) company policies and rules; and
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iv) Best practices and guidelines.
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Does the reference architecture presents:
i) Feasibility to derive it in instances;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iii) Maintainability related issues;
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Does every viewpoint description contain only the required information (i.e., there's no useless information in the viewpoint definition)?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
4. Does the detail level favors the reference architecture understanding?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____
5. Is the abstraction level consistent with the goals of the reference architecture?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
6. Does each view correctly represent its viewpoint?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
7. Do the diagrams address all the concerns framed by the viewpoint?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Domain Expert:
8. Have the following quality attributes have been considered:
i) Availability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
ii) Interoperability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iii) Maintainability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
iv) Performance
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
v) Security
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
vi) Reliability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
vii) Scalability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
viii) Protection
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
ix) Usability
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
9. Does the reference architecture specifies which are the variability mechanisms (e.g. omission, alternatives, and mandatory elements)?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Questions referring to correspondence rules
10. Are there correspondence rules?
Yes ( ) No ( ) I am not expert to answer this question ( )

10.1 For each such rule, is there at least one correspondence satisfying each rule?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
10.2 Are all the correspondences identified and documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
10.3 Are all the correspondences identified in its participating elements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Developer:
11. Can you identify the full set of functional modules?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Comments: _____
12. Can you determine development relationships between these modules?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
12.1 Can you identify runtime dependencies between these modules?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
13. Can you identify required hardware elements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
14. Does the reference architecture specifies how the communication with the outer environment will be treated?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Analyst:
15. Is the reference architecture in conformance with the requirements document?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Domain Expert:
16. Have the used domain data been clearly defined?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Step 2: Check if it is adequate for release

Purpose: Raise discussions between the evaluators and the architects in order to check if the reference architecture is adequate for release.
All Stakeholders:
1. Does the reference architecture clearly state its stakeholders (including domain-specific stakeholders) and their concerns?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Does the reference architecture clearly state potential stakeholders of an instantiated architecture?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Do the selected viewpoints frame the concerns of all stakeholders (including domain-specific stakeholders)?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
4. Do the selected viewpoints frame the concerns of potential stakeholders for an instantiated architecture?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
5. Is the reference architecture consistent with domain's practices and mandated standards?
(Are there parts in the architecture that are not consistent with domain's practices or mandated standards? If yes, how these inconsistencies will be handled?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
6. Do the viewpoints include concerns that are not domain-specific stakeholder's concerns?
(From all the concerns, can you identify concerns that are not domain-specific stakeholder's concerns?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
7. For each viewpoint, are its models clear and well-defined? Do the models provide enough information for determining whether the concerns framed by the viewpoint have been satisfied?
(For each viewpoint, do you fully understand its models? Can you determine whether the framed concerns have been satisfied or not?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Architect:
1. Can you show how you produced the list of RA stakeholders and their concerns.
_____
2. Have the requirements, constraints, standards, and quality assurance policies of the domain been clearly defined?
(Can you explain how the requirements, constraints, standards, and quality assurance policies have been defined and applied to the reference architecture?)

Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Is there awareness for different runtime resources which might affect quality standards used in the RA?
(Can you show which parts of the reference architecture may be affected by resources variability?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
4. Is there a clear description of what parts of the RA are fixed, and what parts are subject to instantiation in a concrete organization / context?
(Can you show which parts of the reference architecture are fixed and which are subject to instantiation in a concrete organization / context?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
5. Is there a clear description of potential contexts in which the RA is instantiated?
(Can you describe potential contexts for instantiation of the reference architecture?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
6. Are there guidelines for how to instantiate the RA?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
7. Is there a description of how variable parts of the RA affect non-variable parts when instantiated in a concrete organization / context?
(Can you describe how variable parts of the RA affect non-variable parts when instantiated?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
8. Does the RA support evolution? If so, how? If not, why not?
(How does the reference architecture support evolution? If it doesn't, can you explain why?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
9. How will the RA be introduced and integrated in organizations?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
10. Are there clear dependencies between quality attributes and the behavior of instantiated RA?
(Can you show dependencies between quality attributes and the behavior of instantiated RA?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
11. Are specific architecturally significant requirements (i.e., the subset of functional, quality attribute, and business requirements that "shape" the reference architecture) identified?
(Can you state the architecturally significant requirements (i.e., the subset of functional, quality attribute, and business requirements that "shape" the reference architecture)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

12. Are there additional architecturally significant requirements that occur when instantiating the RA?
(Can you state additional architecturally significant requirements that occur when instantiating the RA?)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Domain expert:
1. Are the domain goals the system must satisfy clearly articulated and prioritized?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Is there traceability between the domain goals and the requirements the determin?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Is there traceability between the domain goals and the technical solution (i.e., is it possible to navigate from domain goals to architecturally significant requirements, to technical decisions, and finally back to implications on achieving the domain goals)?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
4. What criteria are used to determine whether the RA is supporting the domain goals?
_____
5. Is there a clear outline of how instances of the RA could look like?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

6. Are the threats of introducing the RA clearly documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )

Software manager:
1. Does the RA description allow an estimate of the effort for implementing it?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Is there a way to show the benefit of using the reference architecture?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Does the reference architecture description show what parts can be implemented using OTS(?) or OSS(?) components?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
4. Can you determine development dependencies between different parts of the RA and existing IT landscapes?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
5. Can you lay out a schedule for RA implementation and integration?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
6. Can you tell if there are enough development resources?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Designers and implementers:
1. Can you identify the allowed and prohibited dependencies between parts of the RA?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Can you identify applicable architectural constraints, rules, principles, styles, patterns, etc. that can be used for RA implementation?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Can you determine approaches for error handling, resource management, human-computer interaction, data management and persistence, variation and variability?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

4. Can you determine what is likely to change and how it impacts the RA design?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Integrators
1. Do you understand what needs to be done to integrate the RA in an existing IT landscape?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Do you understand the dependencies of a RA and existing IT landscapes?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Do you understand the adaptation points of the RA? (question 2.13)
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Testers
1. For each partition of the RA, can you determine what is needed (e.g., data, special HW, other units) to test it?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. For each partition of the RA, can you determine what constitutes test success criteria?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Is it possible to identify “representative” challenges that occur in a domain to test the RA?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

QA stakeholders
1. Does the RA description articulate “open decisions”?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Are potential inconsistencies in the RA description (and for instances) known and documented?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Are the test approaches and artifacts consistent with the RA description?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
4. Is the RA understandable for all involved stakeholders?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
5. Does the RA address the key issues of the domain?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
6. Is the RA applicable to the problem is claims to address?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
7. Is the RA complete with regard to domain requirements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
8. Is the RA buildable with regard to constraints of the domain and instantiation contexts?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
9. Is there a process to ensure conformance with quality attribute requirements?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Analysts:
1. If the RA is part of a life cycle or process that includes a procurement decision, does the RA contain the appropriate information to support the procurement process?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Do the customers/acquirers have the right information to understand the key decision and how that decision meets the system requirements and constrains the design and implementation of the system?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____

Step 3. Conclusion / Final Analysis

All stakeholders:
1. Can you determine what is likely to change?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
1.1 Can you determine how does it impact your design?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
2. Is the current document complete in the sense that all information is documented? If not, are there placeholders for what has yet to be documented along with descriptions of what still needs to be worked out?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____
3. Is it feasible that the views drawing upon viewpoints can be instantiated in concrete architectures, within the time and funding available?
Yes ( ) No ( ) Partially ( ) I am not expert to answer this question ( )
Comments: _____