UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

RAModel_{TI}: agregando informações de teste de software no Modelo para Arquiteturas de Referência (RAModel)

Nilton Mendes de Souza

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências
de Computação e Matemática Computacional (PPG-CMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Nilton Mendes de Souza

RAModel_{TI}: agregando informações de teste de software no Modelo para Arquiteturas de Referência (RAModel)

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. José Carlos Maldonado

USP – São Carlos
Novembro de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

S719r Souza, Nilton Mendes
 RAModelTI: agregando informações de teste de
software no Modelo para Arquiteturas de Referência
(RAModel). / Nilton Mendes Souza; orientador José
Carlos Maldonado. -- São Carlos, 2017.
 165 p.

 Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática
Computacional) -- Instituto de Ciências Matemáticas
e de Computação, Universidade de São Paulo, 2017.

 1. Teste de Software. 2. Arquitetura de
Software. 3. Modelo para Arquiteturas de
Referência. I. Maldonado, José Carlos, orient. II.
Título.

Nilton Mendes de Souza

**RAModel_{TI}: towards embodying test information into the
Reference Architecture Model (RAModel)**

Master dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Master Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. José Carlos Maldonado

**USP – São Carlos
November 2017**

Dedico essa dissertação aos meus pais, José Nilton e Vanúbia Maria.

AGRADECIMENTOS

A muitos tenho que agradecer, pois não houve em minha vida, até hoje, um período tão frutuoso de aprendizado quanto os últimos três anos. Primeiramente, agradeço a eles que me subsidiaram emocionalmente, financeiramente e quaisquer outros “*mente” que se possa imaginar: meus pais e meu irmão. A esses, digo: amo-vos profundamente! Este trabalho que eu de defender também é de vocês.

Intelectualmente e academicamente devo muito ao professor José Carlos Maldonado, que muito se esforçou (mesmo a seu jeito próprio de ser carinhoso) para me fazer pensar, escrever e fazer um bom trabalho acadêmico. Muitas vezes eu abusei de sua paciência. Muito obrigado, professor! Agradeço também a professora Elisa Nakagawa, que com seu jeito doce, e muitas vezes quietinho, sempre soube dar bons conselhos. Agradeço ao Lucas Oliveira que também participou deste trabalho e por muitas vezes contribuiu.

A todos amigos do LabES, muito obrigado! Diógenes, Pedro, Silvana, Iohan, Ricardo, Faimison, Draylson, Joice, Cristiane, Paulão ... tantos e tantos que, com certeza, não conseguirei lembrar agora. Em especial, agradeço ao Diógenes e ao Pedro que trabalharam comigo, gastaram seu tempo e me deram muitas dicas. Agradeço a todos por tudo!

Aos amigos “de fora” da vida acadêmica também devo muito, especialmente os amigos do HESED, do GOU, do LIBCOM, da Associação Vigiai e do Huiós. A todos inúmeros amigos que fiz aqui no estado de São Paulo e em Goiás que insistentemente lutam para que eu seja um grande homem. A todos vocês, muito obrigado!

Agradeço a USP e ao ICMC pela infraestrutura cedida para realizar meu trabalho. Agradeço a CAPES pela concessão de bolsa de mestrado.

“Ad Majorem Dei Gloriam”
(Santo Inácio de Loyola)

RESUMO

SOUZA, N. M. **RAModel_T: agregando informações de teste de software no Modelo para Arquiteturas de Referência (RAModel)**. 2017. 165 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Arquitetura de software e teste de software são importantes áreas com o objetivo comum de prover meios para que sistemas de software sejam desenvolvidos com alta qualidade. Foi constatado, por meio de um mapeamento sistemático, que existem poucos estudos relacionando essas duas áreas de modo que uma possa contribuir para a outra, principalmente em arquiteturas de referência e teste de software. Esse projeto tem por objetivo estender o modelo para arquiteturas de referência (RAModel), para que arquiteturas de referência sejam estabelecidas com informações de teste de software. Assim, além de promover padronização, uniformização e reúso a informações relacionadas a arquiteturas de software, promove-se também o reúso, a padronização e a uniformização das informações de teste de software no desenvolvimento das soluções para um dado domínio de aplicação. Para isso, elementos de teste relacionados ao planejamento de teste foram extraídos do TMMi e da norma ISO/IEC/IEEE 29119 e inseridos no RAModel, definindo, assim, o RAModel_T. Dessa forma, a atividade de teste é incorporada aos estágios iniciais do desenvolvimento de software. Um estudo de viabilidade realizado para avaliar a extensão do RAModel_T e um exemplo de aplicação são apresentados.

Palavras-chave: Teste de Software, Arquitetura de Software, Modelo para Arquiteturas de Referência.

ABSTRACT

SOUZA, N. M. **RAModel_{TI}: towards embodying test information into the Reference Architecture Model (RAModel)**. 2017. 165 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Software architecture and software testing are important areas with the common goal of providing the means to develop high-quality software systems. It was found through a systematic mapping that there are few studies relating these two areas so that one can contribute to the other, especially in reference architectures and software testing. This project aims at extending the Reference Architecture Model (RAModel), so that reference architectures are established with software test information. Thus, in addition to promoting standardization, standardization, and reuse of information related to software architectures, it is also promoted the reuse, standardization, and standardization of software test information in the development of solutions for a given application domain. For this, test elements related to the test planning were extracted from the TMMi and the ISO/IEC/IEEE 29119 standard and inserted into the RAModel, thus defining the RAModel_T. In this way, the test activity is incorporated into the early stages of software development. A feasibility study performed to evaluate the extent of the RAModel_T and an application example are presented.

Keywords: Software Test, Software Architecture, Reference Architecture Model.

LISTA DE ILUSTRAÇÕES

Figura 1 – Visão geral do objetivo do grupo de pesquisa em que esse projeto de mestrado está inserido	28
Figura 2 – Metodologia Experimental para Introdução de Tecnologias.	30
Figura 3 – Atividades de projeto arquitetural.	37
Figura 4 – Relação entre modelo de referência, padrão arquitetural, arquitetura de referência e arquitetura de software	38
Figura 5 – Contexto conceitual das arquiteturas de referência	38
Figura 6 – Representação do papel da arquitetura de referência no desenvolvimento de software	40
Figura 7 – Visão geral do ProSA-RA	42
Figura 8 – Representação da estrutura do RAModel	44
Figura 9 – Processo de teste em níveis.	51
Figura 10 – Processo de Planejamento de Teste.	53
Figura 11 – Estrutura do TMMi	55
Figura 12 – Procedimento para definir o <i>Reference Architecture Model with Testing Information</i> (RAModel _{TI})	60
Figura 13 – Visão geral do RAModel _{TI} (extensão do (NAKAGAWA; OQUENDO; BECKER, 2012).	61
Figura 14 – Estrutura dos elementos de teste no RAModel _{TI}	62
Figura 15 – Estrutura geral do RAModel _{TI}	62
Figura 16 – Diagrama que compõe a Visão de Ambiente de Teste	69
Figura 17 – Diagrama que compõe a Visão de Riscos	74
Figura 18 – Diagrama que compõe a Visão de Defeito	75
Figura 19 – Diagrama Projeto de Teste	76
Figura 20 – Visão de Capacidade da RefSORS	80
Figura 21 – Diagrama de Visão de Riscos	90
Figura 22 – Diagrama de Visão de Infraestrutura	91
Figura 23 – Diagrama de Estratégia de Teste	92
Figura 24 – Nível de escolaridade dos sujeitos	98
Figura 25 – Perfil dos usuários em teste de software	99
Figura 26 – Perfil dos usuários em arquitetura de software	99
Figura 27 – Perfil dos usuários em arquitetura de referência	100
Figura 28 – Experiência dos sujeitos nas áreas do estudo de viabilidade	100

Figura 29 – Visão geral do RAModel_{TI}. 139

Figura 30 – Estrutura dos elementos de teste no RAModel_{TI} 140

Figura 31 – Estrutura geral do RAModel_{TI}. 141

LISTA DE QUADROS

Quadro 1 – Descrição dos elementos do grupo Domínio	45
Quadro 2 – Descrição dos elementos do grupo Infraestrutura	45
Quadro 3 – Descrição dos elementos do grupo Aplicação	46
Quadro 4 – Descrição dos elementos do grupo Elementos Transversais	46
Quadro 5 – Descrição do subgrupo Política de Teste	63
Quadro 6 – Descrição do subgrupo Performance de Teste	64
Quadro 7 – Descrição do subgrupo Melhoria de Teste	65
Quadro 8 – <i>Viewpoint</i> para descrição de Gerenciamento de Teste	66
Quadro 9 – Descrição do subgrupo Ambiente de Teste	67
Quadro 10 – Descrição do subgrupo Automatização de Teste	68
Quadro 11 – <i>Viewpoint</i> para descrição de Gerenciamento de Teste	68
Quadro 12 – Descrição do subgrupo de teste Gerenciamento de Defeitos	70
Quadro 13 – Descrição do subgrupo de teste Gerenciamento de Defeitos (Continuação do Quadro 12).	71
Quadro 14 – Descrição do subgrupo de teste Análise de Riscos	71
Quadro 15 – Descrição do subgrupo de teste Análise de Riscos (Continuação do Quadro 14.)	72
Quadro 16 – Descrição do subgrupo de teste Estratégia de Teste	72
Quadro 17 – Descrição do subgrupo de teste Estratégia de Teste	73
Quadro 18 – <i>Viewpoint</i> para descrição de Projeto de Teste	73
Quadro 19 – Relacionamento entre os subgrupos do RAModel _{TI} e o TMMi	77
Quadro 20 – Exemplos de riscos para área de robótica	83
Quadro 21 – Exemplos de riscos para SOA	84
Quadro 22 – Exemplo de mitigações para riscos identificados	85
Quadro 23 – Resumo dos comentários dos sujeitos	103
Quadro 24 – Quadro comparativo entre estudos com relação ao teste de software	124
Quadro 25 – Informações e artefatos de teste gerados nos estudos	125
Quadro 26 – Características gerais de teste	125
Quadro 27 – Linguagens utilizadas para representar a arquitetura	127
Quadro 28 – Descrição do subgrupo de teste Política de Teste	142
Quadro 29 – Descrição do subgrupo de teste Performance de Teste	142
Quadro 30 – Descrição do subgrupo de teste Performance de Teste (Continuação do Quadro 29).	142
Quadro 31 – Descrição do subgrupo de teste Melhoria de Teste	143

Quadro 32 – Descrição do subgrupo de teste Ambiente de Teste.	144
Quadro 33 – Descrição do subgrupo de teste Automatização de Teste.	144
Quadro 34 – Descrição do subgrupo de teste Automatização de Teste (Continuação do Quadro 33).	145
Quadro 35 – Descrição do subgrupo de teste Gerenciamento de Defeitos.	145
Quadro 36 – Descrição do subgrupo de teste Gerenciamento de Defeitos (Continuação do Quadro 35).	146
Quadro 37 – Descrição do subgrupo de teste Análise de Riscos	146
Quadro 38 – Descrição do subgrupo de teste Estratégia de Teste	147
Quadro 39 – Descrição do subgrupo Política de Teste	150
Quadro 40 – Descrição do subgrupo Performance de Teste	152
Quadro 41 – Descrição do subgrupo Melhoria de Teste	154
Quadro 42 – Descrição do subgrupo Ambiente de Teste.	156
Quadro 43 – Descrição do subgrupo Automatização de Teste.	158
Quadro 44 – Descrição do subgrupo Gerenciamento de Defeitos.	160
Quadro 45 – Descrição do subgrupo Análise de Riscos	162
Quadro 46 – Descrição do subgrupo Estratégia de Teste	164

LISTA DE TABELAS

Tabela 1 – Resultado da avaliação para o grupo Gerenciamento de Teste	101
Tabela 2 – Resultado da avaliação para o grupo Infraestrutura de Teste	102
Tabela 3 – Resultado da avaliação para o grupo Estratégia de Teste	102
Tabela 4 – Estudos incluídos no mapeamento sistemático	122

SUMÁRIO

1	INTRODUÇÃO	25
1.1	Contextualização e Motivação	25
1.2	Objetivo	29
1.3	Métodos	29
1.4	Organização	31
2	CONCEITOS BÁSICOS	33
2.1	Considerações Iniciais	33
2.2	Conceitos básicos sobre Arquitetura de Software	34
2.2.1	<i>Projeto de Arquitetura de Software</i>	36
2.2.2	<i>Arquitetura de Referência</i>	37
2.2.3	<i>Classificação de Arquiteturas de Referência</i>	39
2.2.4	<i>Engenharia de Arquiteturas de Referência</i>	41
2.2.5	<i>RAModel: Modelo para Arquiteturas de Referência</i>	43
2.2.6	<i>Exemplos de Arquitetura de Referência</i>	46
2.3	Conceitos básicos sobre Teste de Software	48
2.3.1	<i>Técnicas de Teste de Software</i>	49
2.3.1.1	<i>Teste de Especificação</i>	50
2.3.2	<i>Processo de Teste de Software</i>	50
2.3.2.1	<i>Processo de Teste em Nível Organizacional</i>	51
2.3.2.2	<i>Processo de Teste em nível Gerencial</i>	52
2.3.2.3	<i>Processo de Teste Dinâmico</i>	53
2.3.3	<i>Modelo de Maturidade em Teste de Software</i>	55
2.3.4	<i>UTP - Um perfil de Teste para a UML</i>	57
2.4	Considerações Finais	57
3	RAMODEL_T: UMA EXTENSÃO DO RAMODEL COM INFORMAÇÕES DE TESTE	59
3.1	Considerações Iniciais	59
3.2	Definição do RAModel _{TI}	60
3.3	Descrição do RAModel _{TI}	62
3.3.1	<i>Gerenciamento de Teste</i>	63
3.3.1.1	<i>Política de Teste</i>	63

3.3.1.2	<i>Performance de Teste</i>	64
3.3.1.3	<i>Melhoria de Teste</i>	64
3.3.1.4	<i>Modo de Descrição</i>	65
3.3.2	<i>Infraestrutura de Teste</i>	66
3.3.2.1	<i>Ambiente de Teste</i>	66
3.3.2.2	<i>Automatização de Teste</i>	67
3.3.2.3	<i>Modo de Descrição</i>	68
3.3.3	<i>Projeto de Teste</i>	70
3.3.3.1	<i>Gerenciamento de Defeitos</i>	70
3.3.3.2	<i>Análise de Riscos</i>	71
3.3.3.3	<i>Estratégia de Teste</i>	72
3.3.3.4	<i>Modo de Descrição</i>	73
3.4	Elementos de teste e o TMMi	76
3.5	Considerações Finais	77
4	EXEMPLO DE APLICAÇÃO DO RAMODEL_T	79
4.1	Considerações Iniciais	79
4.2	Extensão da RefSORS	80
4.2.1	<i>RA-1: Investigação de Fontes de Informação</i>	81
4.2.2	<i>RA-T.1: Levantar Conhecimento de Teste</i>	81
4.2.3	<i>RA-T.2: Definir Contexto Organizacional de Teste</i>	82
4.2.4	<i>RA-T.3: Investigar Riscos do Domínio</i>	82
4.2.5	<i>RA-T.4: Estabelecer Estratégia de Teste</i>	85
4.2.5.1	<i>Requisitos de Teste</i>	85
4.2.5.2	<i>Técnicas de Teste</i>	88
4.2.6	<i>RA-T.5: Estabelecer Infraestrutura de Teste</i>	88
4.2.6.1	<i>Ferramentas de Teste</i>	88
4.2.6.2	<i>Ambiente de Teste</i>	89
4.3	Visão de Riscos	89
4.4	Visão de Infraestrutura	91
4.5	Visão de Estratégia	91
4.6	Considerações Finais	92
5	AVALIAÇÃO DO RAMODEL_T	95
5.1	Considerações Iniciais	95
5.2	Estudo de Viabilidade	96
5.3	Planejamento do Estudo de Viabilidade	96
5.3.1	<i>Questões de Pesquisa</i>	96
5.3.2	<i>Objetivo</i>	97
5.3.3	<i>Seleção de Sujeitos</i>	97

5.3.4	<i>Instrumentação</i>	97
5.4	Execução do Estudo de Viabilidade	98
5.5	Análise dos Resultados	101
5.6	Discussão dos Resultados	102
5.7	Ameaças à Validade	104
5.8	Considerações Finais	105
6	CONCLUSÕES E TRABALHOS FUTUROS	107
6.1	Visão Geral da Pesquisa	107
6.2	Contribuições e Limitações da Dissertação	108
6.3	Trabalhos Futuros	108
6.4	Produção Científica	108
REFERÊNCIAS		111
APÊNDICE A RESUMO DO MAPEAMENTO SISTEMÁTICO		121
A.1	Considerações Iniciais	121
A.2	Resumo dos resultados do mapeamento	121
A.3	Considerações Finais	129
APÊNDICE B FORMULÁRIO DE CONSENTIMENTO		131
B.1	Termo de Consentimento	131
APÊNDICE C FORMULÁRIO DE CARACTERIZAÇÃO DE PERFIL		133
C.1	Dados Pessoais	133
C.2	Conhecimento em Teste de Software	134
C.3	Conhecimento em Arquitetura de Software	135
C.4	Conhecimento em Arquitetura de Referência	136
C.5	Conhecimento sobre o RAModel	136
APÊNDICE D SÍNTESE DO RAMODEL _T		139
D.1	Introdução ao RAModel _{TI}	139
D.2	Elementos do RAModel _{TI}	141
D.2.1	<i>Grupo Gerenciamento de Teste</i>	141
D.2.2	<i>Grupo Infraestrutura de Teste</i>	143
D.2.3	<i>Grupo Projeto de Teste</i>	145
APÊNDICE E FORMULÁRIO DE AVALIAÇÃO		149
E.1	Objetivo do Estudo	149
E.2	Grupo: Gerenciamento de Teste	150
E.2.1	<i>Subgrupo: Política de Teste</i>	150

<i>E.2.2</i>	<i>Subgrupo: Performance de Teste</i>	<i>152</i>
<i>E.2.3</i>	<i>Subgrupo: Melhoria de Teste</i>	<i>154</i>
<i>E.3</i>	<i>Grupo: Infraestrutura de Teste</i>	<i>156</i>
<i>E.3.1</i>	<i>Subgrupo: Ambiente de Teste</i>	<i>156</i>
<i>E.3.2</i>	<i>Subgrupo Automação de Teste</i>	<i>158</i>
<i>E.4</i>	<i>Grupo: Projeto de Teste</i>	<i>160</i>
<i>E.4.1</i>	<i>Subgrupo: Gerenciamento de Defeitos</i>	<i>160</i>
<i>E.4.2</i>	<i>Subgrupo: Análise de Riscos</i>	<i>162</i>
<i>E.4.3</i>	<i>Subgrupo: Estratégia de Teste</i>	<i>164</i>

INTRODUÇÃO

1.1 Contextualização e Motivação

Diversos e importantes setores da sociedade são cada vez mais dependentes de sistemas de software. Em decorrência das necessidades dos usuários e dos contextos em que os sistemas estão inseridos, eles têm se tornado cada vez maiores, mais complexos e devem atender padrões de qualidade (CÁMARA *et al.*, 2013). A Engenharia de Software é uma área que se destaca por prover ferramentas, métodos e procedimentos para o desenvolvimento de sistemas de software com qualidade e baixo custo (PRESSMAN, 2010).

Arquiteturas de software têm um papel fundamental para que organizações alcancem seus objetivos em um sistema de software (BASS; CLEMENTS; KAZMAN, 2012). Arquitetura de software representa as estruturas do sistema, que são representadas por um conjunto de elementos, suas propriedades externas e o relacionamento entre eles (BASS; CLEMENTS; KAZMAN, 2012). Por meio de um projeto de arquiteturas de software, a partir de abstrações adequadas, sistemas grandes e complexos tornam-se gerenciáveis (MUCCINI; BERTOLINO; INVERARDI, 2004; CLEMENTS *et al.*, 2010). Ao projetar uma arquitetura de software, detalhes de implementação são abstraídos e a representação do sistema em desenvolvimento concentra-se no comportamento de elementos e na interação entre eles (CLEMENTS *et al.*, 2010). Durante o projeto arquitetural, existem diversas necessidades e cabe ao arquiteto decidir a maneira mais adequada para que uma arquitetura atenda aos requisitos do sistema (HOFMEISTER *et al.*, 2007).

Arquiteturas de referência têm emergido como um tipo especial de arquitetura que contribuem para o projeto de arquiteturas de software com qualidade (NAKAGAWA; OQUENDO; BECKER, 2012). Arquiteturas de referência são constituídas de conhecimento de como projetar arquiteturas de software para sistemas de um dado domínio de aplicação (NAKAGAWA *et al.*, 2014; ANGELOV; GREFFEN; GREEFHORST, 2012). Para isso, arquiteturas de refe-

rência contemplam informações sobre regras de negócio, estilos arquiteturais, boas práticas de desenvolvimento, decisões arquiteturais, restrições de domínio, elementos de software que proveem suporte ao desenvolvimento de sistemas naquele domínio de aplicação, legislação e normas (NAKAGAWA; OQUENDO; MALDONADO, 2014). Considerando sua relevância para o desenvolvimento de software, arquiteturas de referência são estabelecidas para vários domínios, tais como automotivo (AUTOSAR, 2015), sistemas robóticos (OLIVEIRA, 2015), computação em nuvem (FERNANDEZ; MONGE; HASHIZUME, 2015), computação ubíqua (MACHADO *et al.*, 2014), e sistemas com arquitetura orientada a serviço (OASIS, 2012).

Segundo Nakagawa *et al.* (2014), a abordagem mais utilizada para o estabelecimento de arquiteturas de referência é a *ad-hoc*. De acordo com esses autores, utilizar a abordagem *ad-hoc* pode causar esforço desnecessário, consumo excessivo de tempo e retrabalho. Como alternativa, uma abordagem sistemática pode motivar o estabelecimento desse tipo de arquitetura e contribuir para a evolução das existentes. Além disso, pode facilitar o estabelecimento de arquiteturas de referência adequadas ao domínio de aplicação. Nesse contexto, inserem-se o *Reference Architecture Model* (RAModel) (NAKAGAWA; OQUENDO; BECKER, 2012) e o *Process Software Architecture – Reference Architecture* (ProSA-RA) (NAKAGAWA; OQUENDO; MALDONADO, 2014). Segundo seus autores, o modelo de referência apresenta os elementos essenciais e o processo apresenta os passos para o estabelecimento de arquiteturas de referência efetivas. O RAModel é estruturado em quatro grupos de elementos, sendo eles (NAKAGAWA; OQUENDO; BECKER, 2012): domínio, aplicação, infraestrutura e elementos transversais. Esses grupos foram definidos para prover um melhor entendimento dos elementos e do projeto de arquiteturas de referência. A partir de uma revisão da literatura, RAModel e ProSA-RA foram identificados como mais completos e mais recentes até o momento, visto que são baseados nos demais modelos e processos presentes na literatura. O RAModel e o ProSA-RA são utilizados para estabelecer arquiteturas de referência, como a *Reference Architecture for Service-Oriented Robotic Systems* (RefSORS) estabelecida por Oliveira (2015).

O processo de desenvolvimento de software é uma tarefa propensa a erros, mesmo que o projeto arquitetural seja realizado adequadamente. Segundo ISO/IEC/IEEE (2013b), é amplamente aceito que não é possível desenvolver um sistema de software perfeito, gerando, assim, a necessidade de testá-lo antes que seja entregue aos usuários. O teste de software é uma atividade que evidencia a qualidade e a confiabilidade de sistemas de software por meio da execução de um sistema com entradas específicas, analisando a conformidade entre a saída do sistema e a saída esperada (DELAMARO; MALDONADO; JINO, 2016b). A atividade de teste, geralmente, é dividida em quatro etapas (planejamento de teste, projeto de casos de teste, execução e avaliação) e em três fases principais (unidade, integração e sistema) (DELAMARO; MALDONADO; JINO, 2016a).

O teste de software é uma das atividades que permeiam o desenvolvimento e, assim como as demais, deve ser conduzida rigorosamente. Nesse sentido, o *Testing Maturity Model*

Integration (TMMi) foi definido com o objetivo de sistematizar a melhoria de processos de teste de software. Esse modelo possui cinco níveis de maturidade e uma série de tópicos pontuais que permitem avaliar o nível de maturidade do processo de teste implantado em uma organização. Entre os anos de 2014 e 2017, três revisões sistemáticas da literatura (AFZAL *et al.*, 2016; GAROUSI; FELDERER; HACALOGLU, 2017; GARCIA; DAVILA; PESSOA, 2014) sobre melhoria do processo de teste foram conduzidas, em todas o TMMi foi apontado como o modelo de maturidade mais completo e mais utilizado. De forma complementar, foi definida a família de normas ISO/IEC/IEEE 29119 para a atividade de teste de software. Essa família de normas contempla documentos a respeito de: 1 - Conceitos e Definições, 2 - Processo de Teste de Software, 3 - Documentação de Teste, 4 - Técnicas de Teste e 5 - Teste Orientado por Palavras-chave. Tanto a norma quanto o modelo propõem que o teste de software seja considerado desde o nível mais alto de abstração, em nível organizacional, até o mais concreto, em casos e procedimentos de teste; que seja realizado desde os estágios iniciais do desenvolvimento até que seja entregue ao usuário e, caso necessário, após o estágio de manutenção.

O teste de software é uma das principais atividades no processo de desenvolvimento de software, uma vez que evidencia a confiabilidade e a qualidade de sistemas de software (MYERS; SANDLER; BADGETT, 2011). No entanto, apesar dessa atividade ser essencial e investigada, ela ainda é dispendiosa em relação ao custo total do projeto de software (MYERS; SANDLER; BADGETT, 2011). De acordo com Britton *et al.* (2013), o custo anual para detectar e corrigir defeitos ultrapassa 312 bilhões de dólares e utiliza a metade do tempo de desenvolvimento de todo o projeto de software. Desde a gênese de pesquisas em teste de software, pesquisadores têm proposto métodos, técnicas, critérios, ferramentas, entre outros meios para tornar o teste viável e eficiente. Segundo Andrade *et al.* (2013), propor ferramentas ou técnicas para o teste de software não é o bastante, segundo eles, é necessário que existam meios eficientes para que o conhecimento produzido durante a atividade de teste seja reutilizado.

O *Capability Maturity Model Integration CMMi* é uma coleção de boas práticas que ajudam organizações a melhorarem seus processos (CMMI Product Team, 2010). Nesse sentido, o modelo contém os elementos essenciais para tornar processos efetivos. Assim, uma empresa que pretenda certificar a qualidade de seu processo, e consequentemente seus produtos, deve apresentar em qual nível de maturidade do CMMi ele se encontra. No CMMi, desde o nível de maturidade três, o estabelecimento de arquiteturas de software é requerido. De forma semelhante, para certificar o processo de teste de software, tem-se o TMMi. No TMMi, o nível de maturidade dois requer o planejamento de teste. No nível de maturidade três, o planejamento de teste é requerido desde os níveis iniciais do desenvolvimento de software. Torna-se, assim, ainda mais clara a importância das áreas de arquitetura de software e teste de software. Ademais, como constatado por meio de um mapeamento sistemático (SOUZA *et al.*, 2017b), existem poucos estudos que possam relacionar essas duas áreas. De modo que uma possa contribuir para a outra e promover o reúso de informação de teste de software.

Na Figura 1, uma visão geral do objetivo do grupo de pesquisa em que esse projeto de mestrado está inserido é apresentada. Participaram desse objetivo comum dois projetos de mestrado, um que deu origem ao *Process Software Architecture – Reference Architecture - Test* (*ProSA-RA_T*) elaborado por Simão (2017) e este projeto em que *RAModel_{TI}* é definido. Do lado esquerdo da figura, é apresentado o procedimento para estabelecimento de arquiteturas de referência seguindo o *ProSA-RA* e o *RAModel*. A partir deles, uma ou mais arquiteturas de referência podem ser estabelecidas. Nessas, as informações de teste de software não fazem parte da arquitetura e são geradas de modo *ad-hoc* pelos profissionais da área de teste. Desse modo, no final do procedimento, as informações de teste geradas para as aplicações desenvolvidas não possuem uniformidade e nem foram projetadas para serem padronizadas. As elipses com “App *” representam aplicações derivadas da arquitetura concreta e, para uma delas, um conjunto de teste “T*” com várias formas representam a falta de uniformidade entre os conjuntos de teste.

Figura 1 – Visão geral do objetivo do grupo de pesquisa em que esse projeto de mestrado está inserido

Fonte: Elaborada pelo autor.

No procedimento apresentado à direita, as arquiteturas de referência são estabelecidas com o suporte do *ProSA-RA_T* (SIMÃO, 2017) e do *RAModel_{TI}*. Nesse procedimento, informa-

ções de teste de software fazem parte das arquiteturas de referência. Desse modo, as informações de teste possuem uniformidade durante todo o desenvolvimento e resultam em informações de teste padronizadas para as aplicações. Pode-se perceber que os retângulos com “Teste” estão inseridos dentro das arquiteturas de referências e das arquiteturas concretas. As informações de teste derivadas para as aplicações são uniformes.

1.2 Objetivo

O objetivo geral deste projeto de mestrado é inserir informações de teste de software em arquiteturas de referência, com a extensão do [RAModel](#) para antecipar e facilitar o planejamento de teste de software em processos de desenvolvimento de software baseados em arquitetura de software. Como objetivos específicos, têm-se: identificar elementos de teste de software, estender o [RAModel](#) com elementos de teste de software, realizar estudo de viabilidade com o [RAModel_{TI}](#).

O grupo de pesquisa em que este projeto está inserido tem por objetivo estender o ProSA-RA e o RAModel com elementos de teste de software. Como já mencionado, o ProSA-RA foi estendido com elemento de teste de software, definindo-se o [ProSA-RA_T](#). O RAModel foi estendido neste trabalho, resultando no [RAModel_{TI}](#). Os dois são complementares e fazem referência um ao outro no processo de estabelecimento de uma arquitetura de referência, por isso, faz-se necessário a extensão de ambos. Os passos, atividades e tarefas do [ProSA-RA_T](#) são conduzidos observando os elementos do [RAModel_{TI}](#).

1.3 Métodos

Para realizar este projeto de mestrado, quatro grandes passos foram adotados: i) levantamento bibliográfico; ii) mapeamento sistemático; iii) extensão do [RAModel](#) com a definição do [RAModel_{TI}](#); iv) e avaliação do [RAModel_{TI}](#). Uma revisão da literatura foi conduzida para identificar os principais conceitos e investigar trabalhos correlatos a este. Foram identificados conceitos, classificações e aplicações de arquiteturas de software, arquiteturas de referência e engenharia de arquiteturas de referência. De modo semelhante, foram investigados os principais conceitos, classificações e aplicações de teste de software, tais como critérios de teste, técnicas de teste, modelos de maturidade e processos de teste.

Para caracterizar o estado da arte da relação entre arquitetura de software e teste de software, foi conduzido um mapeamento sistemático da literatura ([SOUZA et al., 2016](#); [SOUZA et al., 2017b](#); [SOUZA et al., 2017a](#)). Nesse, foi evidenciado que o teste baseado em arquitetura é a principal abordagem para relacionar arquitetura de software e teste de software. No entanto, não foram identificados trabalhos que incorporassem informações de teste em arquiteturas. Na fase final deste projeto, em março de 2017, o mapeamento foi revisitado com vistas a verificar se

novos trabalhos foram publicados. Uma síntese dos resultados é apresentada no Apêndice A, para ter acesso completo aos resultados deve-se acessar os relatórios técnicos Souza *et al.* (2017b), Souza *et al.* (2017a).

A extensão do RAModel foi realizada em três passos, seguindo o mesmo método adotado por Nakagawa, Oquendo e Becker (2012). Passo 1 - Identificar Fontes de Informação: foram considerados como fontes de informação o TMMi e a família de normas ISO/IEC/IEEE 29119. Passo 2 - Identificar Elementos de Teste: as fontes foram analisadas procurando identificar as informações que os documentos definiram como imprescindível ao planejamento de teste. Passo 3 - Projetar o RAModel_{TI}: os elementos identificados foram classificados em três grupos de teste e oito subgrupos de teste, cada grupo de teste foi incorporado ao grupo Elementos Transversais. Como apoio para representar as informações de teste de software indicadas pelos elementos de teste, foram definidos três visões de teste utilizando diagramas baseados na UML Testing Profile (UTP) (OMG, 2012).

A avaliação desse projeto foi realizada seguindo a metodologia proposta por Shull, Carver e Travassos (2001) e estendida por Mafra, Barcelos e Travassos (2006). Essa metodologia foi definida para a avaliação de novas tecnologias, que é apresentada na Figura 2. Inicialmente, estudos secundários são realizados para identificar evidências sobre o tema tratado. Depois, no estudo de viabilidade, o objeto em estudo é avaliado em relação aos objetivos pelos quais ele foi criado (SHULL; CARVER; TRAVASSOS, 2001). Assim, nesse tipo de estudo não se busca evidências a respeito de uma tecnologia, mas a criação de um corpo de conhecimento sobre o objeto em estudo, ou seja, se a nova tecnologia atende de forma razoável os objetivos previamente definidos (MAFRA; BARCELOS; TRAVASSOS, 2006). Os dois primeiros níveis da metodologia foram alcançados por meio de um mapeamento sistemático e um estudo de viabilidade.

Figura 2 – Metodologia Experimental para Introdução de Tecnologias.

Fonte: Adaptada de Mafra, Barcelos e Travassos (2006).

1.4 Organização

Esta dissertação está organizada da seguinte forma. No Capítulo 2, são apresentados os conceitos básicos sobre arquitetura de software, arquitetura de referência, [RAModel](#) e teste de software, incluindo um processo de teste e o [TMMi](#). No Capítulo 3, apresentam-se o procedimento adotado para estender o [RAModel](#) com elementos de teste de software, a definição e a descrição do [RAModel_{TI}](#). No Capítulo 4, apresenta-se um exemplo de aplicação de uso do [RAModel_{TI}](#). No Capítulo 5, apresenta-se um estudo de viabilidade do [RAModel_{TI}](#). Por fim, no Capítulo 6, as conclusões obtidas durante a condução deste projeto, os desdobramentos e trabalhos futuros são apresentados.

No Apêndice A é apresentada uma síntese dos resultados do mapeamento sistemático ([SOUZA et al., 2017b](#)) e de sua atualização ([SOUZA et al., 2017a](#)). O formulário de consentimento para participação dos sujeitos é apresentado no Apêndice B. O formulário de caracterização de perfil dos sujeitos do estudo de viabilidade é apresentado no Apêndice C. No Apêndice D é descrita a síntese do [RAModel_{TI}](#) que foi fornecida aos sujeitos do estudo de viabilidade. No Apêndice E é apresentado o formulário respondido pelos sujeitos no estudo de viabilidade.

CONCEITOS BÁSICOS

2.1 Considerações Iniciais

Sistemas de software têm se tornado cada vez maiores e mais complexos. Para o desenvolvimento de um sistema de qualidade (BASS; CLEMENTS; KAZMAN, 2012), atividades relacionadas ao projeto de arquitetura são consideradas fundamentais. Com o início da Engenharia de Software, em 1968, os desenvolvedores começaram a criar diagramas que representavam programas. Tinham como objetivo facilitar a comunicação e promover um melhor entendimento do sistema a ser desenvolvido. Segundo Kruchten, Obbink e Stafford (2006), o termo “arquitetura de software” foi utilizado pela primeira vez no ano de 1969, em uma conferência organizada pela NATO¹, a respeito de técnicas de engenharia de software. À medida que os sistemas de software se tornavam maiores, a representação de sistemas de forma estruturada e com vários níveis de abstração se tornava cada vez mais necessária (BARROCA; HALL; HALL, 2000). Segundo esses autores, até o fim dos anos 80, o termo arquitetura era utilizado para referenciar a arquitetura de um sistema/computador.

O desenvolvimento de software é uma atividade propensa a erros. De acordo com a ISO/IEC/IEEE (2013a), não é possível desenvolver um sistema de software perfeito e, por isso, é necessário testá-lo antes que seja entregue aos usuários. O teste de software é uma atividade dinâmica, ou seja, é realizada em tempo de execução, em que o comportamento do sistema é verificado comparando a saída obtida com a saída esperada. Por meio do teste de software, evidenciam-se a qualidade² e a confiabilidade³. A partir disso, essa atividade deve garantir que o sistema de software se comporte somente como o esperado (MYERS; SANDLER; BADGETT,

¹ <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF>

² No contexto deste trabalho, qualidade é considerada como o grau em que o sistema de software atende as necessidades, expectativas ou requisitos do usuário (ISO/IEC/IEEE, 2010).

³ No contexto deste trabalho, confiabilidade é a capacidade que um sistema possui de desempenhar as funções requeridas sob condições pré-estabelecidas em um determinado período de tempo (ISO/IEC/IEEE, 2010).

2011).

Este capítulo está organizado da seguinte forma. Na Seção 2.2 conceitos relacionados a arquitetura de software e arquitetura de referência são descritos. Na Seção 2.3, apresentam-se conceitos sobre teste de software que são fundamentais para o entendimento deste trabalho. Por fim, na Seção 2.4, são apresentadas as considerações finais sobre estes capítulo.

2.2 Conceitos básicos sobre Arquitetura de Software

Nesta seção, são apresentados conceitos básicos sobre arquitetura de software e arquitetura de referência. Por meio da arquitetura de software, a fase do planejamento de software e o estabelecimento de seus atributos de qualidade são iniciados. A arquitetura permite analisar a viabilidade do projeto de software e da qualidade do produto a ser desenvolvido ainda na fase de planejamento (BASS; CLEMENTS; KAZMAN, 2012). Segundo Clements *et al.* (2010), por meio do projeto da arquitetura de software, os elementos do sistema se relacionam de forma coerente e se comportam de maneira adequada. A arquitetura de software é composta pelas estruturas do sistema que contemplam elementos, suas propriedades externas e o relacionamento entre eles (BASS; CLEMENTS; KAZMAN, 2012). De forma complementar, ela também é reconhecida como um conjunto de decisões estratégicas realizadas na fase de planejamento de software (HEESCH; AVGERIOU; HILLIARD, 2012). Com o objetivo de um melhor entendimento da área de arquitetura de software e de seus termos, a seguir serão apresentados os conceitos básicos que servirão como fundamentação para este projeto de mestrado:

Arquitetura concreta: é um termo sinônimo para a instância da arquitetura de software (ANGELOV; GREFFEN; GREEFHORST, 2012) de um dado sistema de software. Esse conceito geralmente é utilizado em estudos relacionados à arquitetura de referência;

Interesse: necessidade que um ou mais *stakeholders* tenha no sistema que será desenvolvido (ISO/IEC/IEEE-42010, 2011);

Stakeholders: são todos aqueles que têm algum interesse no sistema de software, tais como desenvolvedores ou usuários (ISO/IEC/IEEE, 2010);

Trade-off: são ações em que são selecionadas soluções entre algumas alternativas com base em um benefício para as partes interessadas (ISO/IEC/IEEE, 2010);

Decisões arquiteturais: são decisões estratégicas tomadas durante a etapa de projeto e desenvolvimento do software bem como a razão pela qual tomá-las (TYREE; AKERMAN, 2005; HEESCH; AVGERIOU; HILLIARD, 2012). Durante a definição de uma arquitetura, equipes de projeto devem tomar decisões que adequarão o sistema em desenvolvimento às necessidades de usuário. Segundo Fairbanks (2010), as decisões tomadas em nível arquitetural implicam diretamente na qualidade do código e do sistema;

Padrão arquitetural: expressa a organização estrutural fundamental de um sistema ou conjunto de sistemas (CLEMENTS *et al.*, 2010). Em um padrão arquitetural, são descritos os elementos e suas respectivas relações, as restrições cabíveis a ele, o contexto em que ele se insere e as consequências de sua utilização (CLEMENTS *et al.*, 2010). Segundo Eeles (2008), um padrão arquitetural define uma solução de determinado problema e a condição em que esse problema pode ocorrer. Um exemplo de padrão arquitetural é a Arquitetura em Camadas (BUSCHMANN *et al.*, 1996);

Estilo arquitetural: define como serão dispostos os componentes e seus relacionamentos, bem como o conjunto de restrições sobre como eles devem ser utilizados. Em sistemas que possuem o mesmo estilo arquitetural é possível aplicar conhecimento de projetos, ferramentas, implementações e análises para uma classe em particular de projetos, visto que, possuem similaridades estruturais. Por exemplo, a arquitetura orientada a serviço (SOA, do inglês *Service Oriented Architecture*) e a cliente-servidor são estilos arquiteturais conhecidos (CLEMENTS *et al.*, 2010). Em SOA funcionalidades são apresentadas como serviços por meio de interfaces bem definidas que podem ser utilizadas por várias aplicações e no cliente-servidor componentes interagem requerendo serviços a outros componentes, em que os que requerem são chamados clientes e os provedores chamados servidores (SERRANO; HERNANTES; GALLARDO, 2014; MURER; HAGEN, 2014);

Requisitos funcionais: descrevem as funcionalidades que o sistema deve desempenhar (ISO/IEC/IEEE, 2010);

Requisitos não funcionais: descrevem o modo como os elementos do sistema devem se relacionar entre si e o como devem se relacionar com o seu ambiente. Eles também são chamados de **atributos de qualidade** (PRESSMAN, 2010). Por exemplo: confiabilidade, escalabilidade e segurança;

Descrição arquitetural: expressa a arquitetura do sistema para que os *stakeholders* possam compreendê-la. Descrever bem a arquitetura é importante, visto que a arquitetura serve como um esqueleto para o desenvolvimento e como um poderoso meio de comunicação entre os diversos *stakeholders*, os quais a entendem de forma específica (CLEMENTS *et al.*, 2010; BASS; CLEMENTS; KAZMAN, 2012). A descrição arquitetural deve conter todas as informações referentes à arquitetura, as características individuais de cada componente, bem como a forma que eles interagem. A descrição da arquitetura contém várias visões, uma vez que, uma só visão não contempla as informações suficientes para representar o sistema;

Visão arquitetural: é a representação de todo o sistema concebida de forma coerente seguindo um ponto de vista específico (CLEMENTS *et al.*, 2010);

Ponto de vista: estabelece convenções para a construção, interpretação e o uso de visões arquiteturais para um interesse específico (ISO/IEC/IEEE-42010, 2011); e

Avaliação arquitetural: é a atividade em que é verificado se uma arquitetura satisfaz critérios e requisitos não funcionais. Assim, pode ser verificada a correspondência entre a arquitetura e os requisitos dos *stakeholders* (CLEMENTS *et al.*, 2010). São exemplos de métodos de avaliação arquitetural: *Architecture Trade-off Analysis Method (ATAM)* (KAZMAN *et al.*, 1998) e o *Scenario-based Architecture Analysis Method (SAAM)* (KAZMAN *et al.*, 1994).

A apresentação desses termos é importante para evitar ambiguidade, uma vez que alguns deles podem ser mal compreendidos ou considerados sinônimos. Além da compreensão desses termos, para projetar uma arquitetura de software com qualidade existem métodos, modelos e linguagens que serão apresentados na próxima seção.

2.2.1 Projeto de Arquitetura de Software

Ao longo dos anos, diversos métodos têm sido propostos com o objetivo de contribuir para o projeto de uma arquitetura de software (HOFMEISTER *et al.*, 2007). Por exemplo, têm-se os estudos conduzidos por Jacobson, Booch e Rumbaugh (1999), Buschmann *et al.* (1996), Bass, Clements e Kazman (2012), Hofmeister *et al.* (2007), Clements *et al.* (2010) Heesch, Avgeriou e Hilliard (2012) e Falessi *et al.* (2011). Esses métodos envolvem a proposição de visões arquiteturais (KRUCHTEN, 1995), *frameworks* para decisões arquiteturais (HEESCH; AVGERIOU; HILLIARD, 2012; FALESSI *et al.*, 2011) e projeto da arquitetura (HOFMEISTER *et al.*, 2007; BASS; CLEMENTS; KAZMAN, 2012; BUSCHMANN *et al.*, 1996).

Hofmeister *et al.* (2007) definiram um modelo geral para o projeto de arquiteturas de software. Esse modelo foi concebido com base em cinco métodos de projeto arquitetural utilizados na indústria, a saber: *Attribute-Driven Design* (BASS; CLEMENTS; KAZMAN, 2012), *Siemens' 4 Views (S4V)* (HOFMEISTER; NORD; SONI, 2005), *Rational Unified Process*[®] (RUP[®]) (KRUCHTEN, 2003), *Business architecture process and organization* (MULLER, 2005), e *Architectural separation of concerns* (RAN, 2000). Na Figura 3, é apresentada uma visão geral do modelo de Hofmeister *et al.* (2007). Os conceitos relacionados ao modelo são:

- **Análise arquitetural:** é a atividade para definir quais problemas devem ser resolvidos com a arquitetura. Essa atividade possui dois artefatos de entrada: (i) **Interesses arquiteturais** que são os interesses relacionados ao desenvolvimento do sistema, ao funcionamento, e às características mais relevantes do sistema de interesse; e (ii) **Contexto** que é constituído de: ambiente, operações, políticas, regras de negócio, características da organização e circunstâncias em que serão desenvolvidos o sistema. Essa atividade tem como saída os **requisitos relevantes para a arquitetura** que são os requisitos que influenciam o projeto da arquitetura;

- **Síntese arquitetural** são definidas as possíveis soluções arquiteturais que atendam aos requisitos identificados na análise arquitetural. Essa atividade recebe como entrada os requisitos relevantes para a arquitetura. A saída dessa atividade é uma **solução arquitetural candidata** que contém soluções alternativas que refletem a estrutura do sistema e as decisões tomadas que diferenciam tais soluções; e
- **Avaliação arquitetural** é a atividade que garante que as decisões realizadas no projeto da arquitetura foram corretas. Essa atividade recebe como entrada: (i) **solução de arquitetura candidata**; e os (ii) **requisitos relevantes para a arquitetura**. A saída dessa atividade é a **arquitetura validada**, que é a arquitetura em que suas soluções foram avaliadas em relação aos seus requisitos.

Figura 3 – Atividades de projeto arquitetural.

Fonte: Adaptada de Hofmeister *et al.* (2007).

Para representar a arquitetura de software, uma abordagem bastante conhecida é a “4 + 1” definida por (KRUCHTEN, 1995). Nessa abordagem, é proposto que a arquitetura seja descrita nas visões lógica, de desenvolvimento, de processo e física. Por fim, todas essas visões são representadas juntas por meio de cenários (ilustração do comportamento do sistema em sua arquitetura representada nas quatro visões). Adicionalmente, como meio para dar suporte ao projeto de arquiteturas com qualidade, considerando reúso, padronização e projeto para evolução de arquiteturas, tem-se utilizado de arquiteturas de referência que serão apresentadas na próxima seção.

2.2.2 Arquitetura de Referência

Arquitetura de referência é um tipo especial de arquitetura que contempla a abstração de conhecimento coletado de diversas arquiteturas concretas de um domínio, ou classe de aplicação (NAKAGAWA; OQUENDO; MALDONADO, 2014; ANGELOV; GREFFEN; GREEFHORST, 2012). Bass, Clements e Kazman (2012) definem arquitetura de referência como um modelo de referência mapeado em termos de elementos de software. Esses autores definem um modelo de referência como a decomposição de um problema em elementos que se relacionam seguindo

um fluxo de atividades e, coletivamente, resolvem esse problema. Na Figura 4, é apresentada a relação entre modelo de referência, padrão arquitetural, arquitetura de referência e arquitetura de software, segundo Bass, Clements e Kazman (2012). Galster (2015) apresenta o contexto conceitual em que se insere arquiteturas de referência e modelos de referência com base na norma (ISO/IEC/IEEE-42010, 2011), como pode ser observado na Figura 5. Modelos de referência, padrões arquiteturais e arquiteturas de referência são artefatos que precedem e podem dar suporte ao projeto de arquiteturas concretas. Considerada a relevância de arquiteturas de referência, algumas têm sido propostas, tais como: domínio automotivo (AUTOSAR, 2015); ferramentas de teste de software (OLIVEIRA; NAKAGAWA, 2011); computação ubíqua (MACHADO *et al.*, 2014); segurança para sistemas em nuvem (FERNANDEZ; MONGE; HASHIZUME, 2015); e segurança e privacidade na internet das coisas (ADDIO *et al.*, 2014).

Figura 4 – Relação entre modelo de referência, padrão arquitetural, arquitetura de referência e arquitetura de software.

Fonte: Adaptada de Bass, Clements e Kazman (2012).

Figura 5 – Contexto conceitual das arquiteturas de referência.

Fonte: Adaptada de Galster (2015).

De acordo com [Guessi, Oquendo e Nakagawa \(2014\)](#), arquiteturas de referência guiam e facilitam o desenvolvimento, a padronização e a evolução de múltiplos projetos de sistemas de software referentes a um mesmo domínio de aplicação. Para que isso seja possível, na arquitetura de referência são encapsuladas diversas informações relativas ao domínio de aplicação, por exemplo: regras de negócios, estilos e padrões arquiteturais, boas práticas, restrições, legislação, entre outros ([NAKAGAWA; OQUENDO; MALDONADO, 2014](#)). Além disso, as arquiteturas de referências podem conter informações gerenciais sobre uma ou mais empresas e do ambiente em que ela foi estabelecida. Segundo [Cloutier et al. \(2009\)](#), a arquitetura de referência está diretamente relacionada à missão, visão e a estratégia da empresa (ou múltiplas empresas) que a utiliza. Ao reunir todas essas informações, ela torna-se uma ferramenta de reúso, pois facilita o compartilhamento de conhecimento sobre vários produtos, direcionamentos para o futuro, organizações e conteúdos relacionados à arquitetura atual dos sistemas de software. Dadas as definições, é de comum acordo entre os autores citados que as arquiteturas de referência são um meio para reutilizar conhecimento que pode ser utilizado para guiar o projeto, padronizar e evoluir arquiteturas concretas.

Na Figura 6 é representada uma visão geral do papel da arquitetura de referência no desenvolvimento de sistemas de software. Pode-se notar que uma mesma arquitetura de referência pode ser utilizada no estabelecimento de diversas arquiteturas concretas e, conseqüentemente, diversos sistemas de software. Arquitetura de referência deve ser genérica a ponto de contemplar características similares a cada sistema ([ANGELOV; TRIENEKENS; GREFEN, 2008](#)), além disso, deve conter as variabilidades que cada um deles pode assumir ([GUESSI; OQUENDO; NAKAGAWA, 2014](#)).

2.2.3 Classificação de Arquiteturas de Referência

Arquiteturas de referência têm sido desenvolvidas de várias maneiras e com diferentes propósitos. Para facilitar um melhor entendimento dessas arquiteturas, principalmente em relação ao contexto em que foram estabelecidas, [Angelov, Grefen e Greefhorst \(2012\)](#) propuseram um *framework*. Esse *framework* foi definido em três camadas, a saber, contexto, objetivo e projeto. Ele pode ser utilizado para análise e classificação de arquiteturas de referência. A descrição de cada uma das camadas é dada a seguir:

- **Contexto:** arquiteturas de referência podem ser projetadas para uma única organização ou várias organizações, desde que compartilhem algum interesse. Além disso, arquiteturas de referência podem ser definidas como clássicas, quando definidas em um contexto já estabelecido, ou como preliminares, quando a arquitetura é estabelecida previamente à tecnologia a que ela se refere;
- **Objetivo:** foram identificados dois objetivos para arquiteturas de referências: padronização e facilitação. A padronização tem como objetivo prover interoperabilidade entre sistemas,

Figura 6 – Representação do papel da arquitetura de referência no desenvolvimento de software.

Fonte: Adaptada de [Angelov, Trienekens e Grefen \(2008\)](#).

enquanto a facilitação propicia um guia para o projeto de arquiteturas concretas;

- **Projeto:** elementos das arquiteturas de referência podem ser descritos nos seguintes níveis de abstração: detalhado, semi-detalhado e agregado; bem como nos seguintes níveis de formalidade: informal, semi-formal e formal.

[Angelov, Grefen e Greefhorst \(2012\)](#) apresentam cinco classificações para arquiteturas de referência que podem ser divididas por seu propósito. As cinco classificações são:

Tipo 1 (Padronização, Múltiplas organizações, Clássica): Arquiteturas nessa classificação têm como objetivo promover uma padronização de arquiteturas concretas entre várias organizações;

Tipo 2 (Padronização, Única organização, Clássica): Essas arquiteturas são projetadas para servir como uma ferramenta para padronização e de arquiteturas concretas em uma organização;

Tipo 3 (Facilitação, Múltiplas organizações, Clássica): Essas arquiteturas têm como objetivo facilitar o desenvolvimento de produtos de software em múltiplas organizações;

Tipo 4 (Facilitação, Única organização, Clássica): Essas arquiteturas são projetadas para guiar o desenvolvimento de produtos de software em uma única empresa e facilitar o projeto de arquiteturas na organização em que estão inseridas; e

Tipo 5 (Facilitação, Múltiplas organizações, Preliminar): Arquiteturas nessa classificação são projetadas para múltiplas organizações (ou centros de pesquisa). Diferentemente dos demais tipos, essas arquiteturas são preliminares, ou seja, elas são definidas antes que existam experiência no domínio de interesse.

O estudo de [Martínez-Fernández et al. \(2017\)](#) aponta e compara vantagens e desvantagens decorrentes da implantação e do uso de arquiteturas de referência em uma organização. Entre os resultados desses autores, têm-se que as arquiteturas de referência contribuem positivamente para a interoperabilidade de sistemas de software, redução de custos e melhoria na comunicação entre os *stakeholders*. Por outro lado, caracterizando desvantagens, esses autores identificaram a acentuação da curva de aprendizado, alto nível de abstração, entre outros. [Nakagawa, Oquendo e Maldonado \(2014\)](#) referem-se a área de pesquisa de arquitetura de referência como relativamente nova, embora contribuições tenham sido realizadas na última década. Além disso, segundo eles, ainda há muito a explorar, principalmente, em tópicos emergentes, tais como: sistemas-de-sistemas, arquiteturas orientadas a serviço e múltiplas linhas de produto.

2.2.4 Engenharia de Arquiteturas de Referência

De um modo geral, o projeto de arquiteturas de referência tem sido conduzido de maneira *ad-hoc*, ou seja, não tem seguido um processo sistematizado ([NAKAGAWA et al., 2014](#)). No entanto, a sistematização desse processo pode contribuir para o projeto de arquiteturas de referência mais coerentes com o seu contexto. Algumas iniciativas têm sido conduzidas nesse sentido, como [ProSA-RA \(NAKAGAWA; OQUENDO; MALDONADO, 2014\)](#), [RAModel\(NAKAGAWA; OQUENDO; MALDONADO, 2014\)](#), *frameworks* para análise e classificação de arquiteturas, entre outros. Algumas dessas iniciativas serão apresentadas a seguir.

[Galster e Avgeriou \(2011\)](#) propuseram uma abordagem em seis passos para a definição de uma arquitetura de referência. Eles propuseram que o projeto de arquiteturas seja baseado em conhecimento obtido a partir de projetos existentes. O [ProSA-RA](#) é um processo que tem o objetivo de sistematizar o estabelecimento de arquiteturas em quatro passos. Ele apresenta fontes em que podem ser adquiridas informações para basear o desenvolvimento dessas arquiteturas, além de apresentar fases do projeto e avaliação de tais arquiteturas ([NAKAGAWA et al., 2014](#)). Na Figura 7, o [ProSA-RA](#) é apresentado.

Para analisar e classificar arquiteturas de referência, o *framework* proposto por [Angelov, Grefen e Greefhorst \(2012\)](#) apresenta cinco tipos de arquiteturas com as variações apresentadas na Figura 2.2.3. Esse *framework*, segundo os autores, pode ser utilizado para analisar arquiteturas de referência existentes, guiar o estabelecimento de novas arquiteturas e facilitar a evolução dessas.

Figura 7 – Visão geral do ProSA-RA

Fonte: Adaptada de Nakagawa *et al.* (2014).

Martínez-Fernández *et al.* (2011) propuseram um *framework* para análise de custo-benefício na utilização de arquiteturas de referência. Esse *framework*, *A Reuse-Based Economic Model for Software Reference Architectures* (REARM), definido com base em modelos e métricas econômicas de outras áreas, tem o propósito de facilitar a avaliação do impacto econômico em adotar uma arquitetura de referência.

Muller (2008) discute o nível de detalhes das descrições de arquiteturas de referência. De acordo com os autores, definir o nível de abstração ideal para as informações contidas na arquitetura de referência é uma tarefa complexa, pois se deve abstrair informações, tais como: técnicas sobre sistemas, técnicas empresariais e técnicas de negócios. Além disso, o estudo apresenta algumas observações sobre quais informações devem estar contidas em arquiteturas de referência, tais como: visões e seus elementos essenciais. Esses elementos podem ser considerados como: padrões arquiteturais, conceitos baseados em lições aprendidas e problemas conhecidos. Além disso, também devem estar presentes nas arquiteturas de referência necessidades de usuário, necessidades de negócio, futuros requisitos e um portfólio de produtos.

Dobrica e Niemelä (2008) descrevem uma abordagem para estabelecimento de arquiteturas de referência no contexto de sistemas embarcados orientados a serviços. Para eles, a arquitetura de referência deve conter os principais serviços e características, de maneira abstrata, dos sistemas resultantes. Acrescentam ainda, que o estilo arquitetural na arquitetura de referência, deve ser apresentado por meio de um conhecimento base, no formato de uma taxonomia. Afirmam que o principal interesse sobre atributos de qualidade em arquiteturas de referência é como esses atributos interagem com restrições, bem como *tradeoffs* que se aplicam a classe de

sistemas que embarcados.

Cloutier *et al.* (2009) sugerem que arquiteturas de referência devem conter: i) informações técnicas sobre arquiteturas concretas; ii) arquitetura sobre negócios; iii) informações sobre o contexto do cliente. Nakagawa, Oquendo e Becker (2012), por sua vez, propõem o **RAModel** para apresentar as informações essenciais de arquitetura de referência. A título de exemplo, funcionalidades e restrições dos sistemas resultantes da arquitetura de referência, a infraestrutura de implantação dos sistemas, as particularidades do domínio. Esse modelo será descrito em mais detalhes na próxima seção.

2.2.5 **RAModel: Modelo para Arquiteturas de Referência**

As abordagens que têm sido utilizadas para estabelecer arquiteturas de referência não seguem uma maneira sistemática (NAKAGAWA *et al.*, 2014). Em consequência da falta de sistematização, o estabelecimento de arquiteturas de referência incompletas ou inadequadas com seu contexto pode ser recorrente (NAKAGAWA; OQUENDO; BECKER, 2012). Procurando sistematizar o projeto de arquiteturas de referência, Nakagawa, Oquendo e Becker (2012) definiram um modelo de referência com o objetivo de apoiar o estabelecimento de arquiteturas de referência. Segundo Kirchmer (2011), modelos de referência se baseiam em um conjunto pequeno de conceitos que juntos agregam práticas para um domínio particular. Esses modelos definem uma linguagem comum que eleva a qualidade do projeto e podem ser utilizados por diversas empresas, pois representam boas práticas. Nesse contexto, o **RAModel** insere-se como um modelo genérico que possui os elementos essenciais para guiar o estabelecimento de arquiteturas de referência de qualidade (NAKAGAWA; OQUENDO; BECKER, 2012). Na Figura 8, é apresentada a estrutura geral do **RAModel**.

De acordo com Nakagawa, Oquendo e Maldonado (2014), a utilização desse modelo pode ser relevante para: i) avaliação de uma arquitetura de referência; ii) análise comparativa entre duas ou mais arquiteturas de referência; iii) fornecer bases para o projeto de uma Linha de Produto de Software; e iv) prover informações básicas para estabelecer uma arquitetura de referência. O **RAModel** possui uma estrutura de elementos que propicia a compreensão, definição e avaliação de arquiteturas de referência. Esse modelo fornece os elementos básicos para o estabelecimento de uma arquitetura de referência. Os elementos foram organizados nos seguintes grupos:

- **Domínio:** Os elementos desse grupo contêm informações sobre o ambiente de aplicação da arquiteturas de referência, por exemplo: legislação, padrões, atributos de qualidade e certificações. No Quadro 1, apresentam-se detalhes sobre os elementos referentes ao grupo Domínio;
- **Infraestrutura:** Os elementos desse grupo serão utilizados para basear a arquitetura de referência, por exemplo: boas práticas, elementos de software e hardware. No Quadro 2,

Figura 8 – Representação da estrutura do RAModel

Fonte: Adaptada de Nakagawa, Oquendo e Becker (2012).

apresenta-se a descrição do grupo de elementos Infraestrutura;

- **Aplicação:** Os elementos desse grupo possuem informações relativas às regras de negócios, restrições, riscos e funcionalidades que devem estar presentes em sistemas que serão derivados da arquitetura de referência. No Quadro 3, são descritos os elementos que pertencem ao grupo Aplicação;
- **Elementos transversais:** Esses elementos são relacionados aos demais grupos (domínio, aplicação e infraestrutura), por exemplo, decisões arquiteturais e comunicação. No Quadro 4, são descritos os elementos transversais aos demais grupos.

Quadro 1 – Descrição dos elementos do grupo Domínio. Adaptado de Nakagawa, Oquendo e Becker (2012).

Elemento	Descrição
Legislações, normas e regulamentos	Leis, normas e regulamentos existentes no domínio que devem estar presentes nos sistemas resultante da arquitetura de referência
Atributos de qualidade	Atributos de qualidade, por exemplo: manutenibilidade, portabilidade e escalabilidade que são desejáveis aos sistemas resultantes da arquitetura de referência. Por vezes, esses atributos estão implícitos ou entrelaçados na estrutura da arquitetura. Porém, é importante explicitar essas informações.
Compatibilidade do sistema	Nesse elemento, verifica-se se o sistema resultante da arquitetura segue as leis, normas e regulamentos. Caso esses forem considerados, a arquitetura de referência terá mais chances de estar em conformidade com as exigências de seu domínio no mundo real.

Quadro 2 – Descrição dos elementos do grupo Infraestrutura. Adaptado de Nakagawa, Oquendo e Becker (2012).

Elemento	Descrição
Boas práticas e diretrizes	Práticas bem estabelecidas e, comprovadamente, eficientes no desenvolvimento de sistemas de software no domínio. Por exemplo, padrões de projeto. Essas práticas podem ser acompanhadas por diretrizes que descrevem o modo de utilizá-las.
Estrutura Geral	Estrutura geral da arquitetura de referência, que pode ser representada pelo uso de estilos arquiteturais ou padrões arquiteturais. A estrutura também se refere aos elementos e os relacionamentos existentes entre eles.
Elementos de Hardware	É constituído dos elementos de hardware, como servidores ou dispositivos (por exemplo, sensores ou atuadores). Esses elementos comporão um ou mais sistemas resultantes da arquitetura de referência.
Elementos de Software	Constituem-se dos elementos de software, por exemplo classes, métodos ou componentes. Esses elementos deverão compor um ou mais sistemas resultantes da arquitetura de referência.

Quadro 3 – Descrição dos elementos do grupo Aplicação. Adaptado de Nakagawa, Oquendo e Becker (2012).

Elemento	Descrição
Restrições	Restrições aplicáveis à arquitetura de referência ou em uma parte específica dela.
Dados de domínio	Dados comuns ao domínio de aplicação do sistema. Geralmente, esses elementos são descritos em alto nível de abstração; contudo, isso pode variar a depender do objetivo da arquitetura de referência
Requisitos Funcionais	Conjunto de requisitos funcionais presentes na arquitetura de referência que são comuns aos sistemas desenvolvidos utilizando esta arquitetura.
Objetivos e Necessidades	Para que destina-se a arquitetura de referência e as necessidades e objetivos que ela deve cobrir.
Limitações	Limitações cabíveis à arquitetura de referência ou por em parte específica dela.
Riscos	Riscos gerados a partir do uso da arquitetura de referência ou por em parte específica dela. Com essa informação, o usuário pode preparar-se para gerenciar os riscos.
Escopo	Escopo que a arquitetura de referência abrange, isto é o conjunto de sistemas que podem ser desenvolvidos baseando-se na arquitetura de referência.

Quadro 4 – Descrição dos elementos do grupo Elementos Transversais. Adaptado de Nakagawa, Oquendo e Becker (2012).

Elemento	Descrição
Decisões	Durante o estabelecimento da arquitetura de referência, as decisões tomadas devem ser documentadas. Na documentação das decisões incluem-se: a descrição da decisão, alternativas à decisão, o motivo por qual tomá-la e o <i>tradeoff</i> da decisão.
Terminologia de domínio	Conjunto de termos utilizados para o estabelecimento da arquitetura de referência que são amplamente aceitos no domínio em ela se insere.
Comunicação externa	Informações sobre a comunicação de um sistema resultante da arquitetura de referência e o seu ambiente. Por exemplo, portas, interfaces, protocolos, entre outros. Esses elementos são importantes para prover a interoperabilidade entre sistemas.
Comunicação interna	Informações sobre a comunicação dos elementos de um sistema resultante da arquitetura de referência. Por exemplo, portas, interfaces, protocolos, entre outros. Esses elementos são importantes para prover a interoperabilidade entre os diversos componentes ou módulos do sistema.

2.2.6 Exemplos de Arquitetura de Referência

Nesta seção, serão apresentados dois exemplos de arquiteturas de referência. A primeira, chamada *Automotive Open System Architecture (AUTOSAR)* (AUTOSAR, 2015), é uma arquitetura de referência amplamente conhecida e estabelecida para o domínio automotivo. Também

será descrita a RefSORS (OLIVEIRA, 2015), uma arquitetura de referência para sistemas robóticos orientados a serviço. Além desses exemplos, há novas perspectivas para a aplicação de arquiteturas de referência como em veículos aéreos não tripulados, computação móvel e casas inteligentes, sistemas de sistemas, entre outros (GALSTER, 2015; NAKAGAWA; OQUENDO; MALDONADO, 2014).

Arquitetura para Sistemas Automotivos Abertos

A AUTOSAR é uma arquitetura de referência para o domínio automotivo. Ela tem o propósito de estabelecer arquiteturas de software padronizadas entre fornecedores e fabricantes da indústria automotiva (AUTOSAR, 2015). Segundo Wiesbaden (2013), ela foi iniciada há 10 anos, está bem estabelecida e é bem aceita na indústria.

Dado que um automóvel contém diversos equipamentos que utilizam sistemas de software, por exemplo microcontroladores, sensores e atuadores, a AUTOSAR provê uma estrutura para o projeto de arquiteturas de software para esses equipamentos. A AUTOSAR é dividida em três camadas principais de software, a saber (AUTOSAR, 2015): camada de aplicação, ambiente em tempo de execução e software básico. Cada componente de software é descrito como uma caixa preta. Não sendo descrito como um componente, deve ser implementado, apenas como as interfaces devem ser dispostas. Fernandez, Monge e Hashizume (2015) fizeram uma análise das vantagens e desvantagem no uso da AUTOSAR. Esses autores identificaram como vantagens: padronização; reúso; interoperabilidade; melhoria da comunicação; e repositório de conhecimento. As seguintes desvantagens foram identificadas: complexidade; investimento inicial; curva de aprendizado; confusão de termos; e nível de abstração muito alto.

Arquitetura de Referência para Sistemas Robóticos Orientados a Serviço

A RefSORS, definida por Oliveira (2015), é uma arquitetura de referência para apoiar o desenvolvimento de sistemas robóticos orientados a serviços. Para isso, foram utilizadas as seguintes fontes de informação: sistemas robóticos identificados na literatura; diretrizes para o desenvolvimento de sistemas orientados a serviço; arquiteturas de referência para sistemas robóticos; e arquiteturas concretas de controle do domínio de robótica. Ela foi projetada com descrição informal e semi-formal por meio das linguagens SoaML⁴ e Unified Modeling Language (UML). Além disso, a RefSORS foi descrita utilizando as seguintes visões: visão conceitual, visão de capacidade, visão de interface de serviço e contratos, visão de serviços e visão de implantação.

A RefSORS foi estabelecida utilizando uma taxonomia de serviços, o processo ProSARA e o modelo de referência RAModel. Entre as contribuições dessa arquitetura, destacam-se um melhor entendimento da área de sistemas robóticos orientados a serviço, bem como reunir um conjunto de conceitos com vistas a prover uma terminologia comum e apoiar o projeto de

⁴ <http://www.omg.org/spec/SoaML/>

arquiteturas concretas. A RefSORS foi avaliada por meio de estudo de caso, no qual os autores concluíram que o uso dessa arquitetura influenciou positivamente no reúso, modularidade e no acoplamento de uma arquitetura de software para sistemas robóticos orientados a serviço.

2.3 Conceitos básicos sobre Teste de Software

Nessa seção, serão apresentados os conceitos básicos sobre teste de software. Desenvolver sistemas de software com qualidade é um dos principais objetivos da Engenharia de Software (PRESSMAN, 2010), nesse sentido, as atividades de Verificação, Validação e Teste de Software são consideradas fundamentais (DELAMARO; MALDONADO; JINO, 2016a). A verificação é constituída de atividades que têm o objetivo de analisar se uma função implementada está correta, enquanto a validação objetiva garantir que a função implementada é compatível com seus requisitos (PRESSMAN, 2010). Por sua vez, o teste de software é um processo em que o programa em teste é executado com o objetivo de encontrar erros (MYERS; SANDLER; BADGETT, 2011), que ao serem encontrados e removidos, elevam a confiabilidade do sistema. É importante destacar a relação entre engano, defeito, erro e falha proposto pela norma ISO/IEC/IEEE (2010): i) **Engano (*mistake*)**: ação humana que produz um resultado incorreto; ii) **Defeito (*fail*)**: processo ou definição de dados incorretos no programa; iii) **Erro (*error*)**: estado inconsistente ou inesperado do programa; e iv) **Falha (*failure*)**: evento no qual um sistema ou componente do sistema não consegue realizar uma função requerida nos limites especificados. Ou seja, o programa produz um resultado inesperado em relação à especificação. Visando a sistematizar o teste de software, esse foi dividido em algumas fases, uma vez que as principais dessas são (MYERS; SANDLER; BADGETT, 2011; DELAMARO; MALDONADO; JINO, 2016a):

- **Teste de Unidade**: visa a explorar individualmente as menores unidades computáveis ou módulos do sistema. Desse modo, são priorizadas as falhas relacionadas aos defeitos lógicos referentes unicamente à unidade em teste. Essa fase pode ter início antes mesmo do sistema completo estar pronto, visto que as unidades podem ser, por exemplo, componentes, métodos, classes ou funções;
- **Teste de Integração**: tem como objetivo complementar o teste de unidade, investigando a relação entre os módulos testados individualmente. Nessa fase, assume-se que as unidades do sistema funcionam corretamente e, geralmente, o teste de integração é realizado antes que o sistema seja finalizado;
- **Teste de Sistema**: visa a verificar a funcionalidade do sistema como um todo, envolvendo as unidades e suas interfaces. Nessa fase, o funcionamento do sistema completo é testado. Além disso, é desejável que o teste de sistema ocorra na mesma condição e no mesmo

ambiente em que o sistema será implantado. No teste de sistema, características ligadas aos requisitos não-funcionais como segurança e performance também são testadas; e

- **Teste de Regressão:** visa a testar funcionalidades do sistema após ele sofrer modificações ou adições. Nessa fase, o sistema modificado deve ser testado com o intuito de assegurar que as funções que o sistema desempenhava corretamente permanecem funcionais, ou seja, as modificações não causaram problemas à versão implantada anteriormente.

Em cada uma dessas fases, o teste de software pode ser explorado utilizando diferentes técnicas que variam pela fonte de informação utilizada para derivar os requisitos de teste, conforme detalhado na próxima seção.

2.3.1 Técnicas de Teste de Software

Testar um sistema utilizando todo o domínio de entrada, geralmente, é inviável. Utilizar apenas um conjunto de dados de teste pode tornar-se viável, desde que ele tenha alta probabilidade de revelar defeitos (DELAMARO; MALDONADO; JINO, 2016a). Há duas formas para selecionar subconjuntos de casos de teste (DELAMARO; MALDONADO; JINO, 2016a; ARCURI; IQBAL; BRIAND, 2011): teste randômico e teste de subdomínios. A técnica mais simples para selecionar casos de teste é o teste randômico. No teste randômico casos de teste são selecionados aleatoriamente a partir do domínio de entrada, baseando-se em alguma distribuição (ARCURI; IQBAL; BRIAND, 2011). Por outro lado, no teste de subdomínios, o domínio de entrada é particionado e casos de teste são selecionados a partir de cada partição. No teste de subdomínios, deve-se selecionar no mínimo um caso de teste para representar todo subdomínio. A divisão dos subdomínios segue regras que podem ser baseadas em funcionalidades definidas a partir dos requisitos ou critérios estruturais de cobertura de código (ARCURI; IQBAL; BRIAND, 2011). Essas regras são denominadas critérios de teste e, por meio delas, são selecionados os casos de teste para cada subdomínio (DELAMARO; MALDONADO; JINO, 2016a).

Na literatura, existem diversos critérios de teste de software, eles podem basicamente ser classificados em três técnicas: funcional, estrutural e baseado em defeitos (FABBRI *et al.*, 2016). O que distingue cada técnica é a fonte de informação utilizada para obter os requisitos de teste (DELAMARO; MALDONADO; JINO, 2016b). Os critérios de teste da técnica funcional baseiam-se na especificação do sistema de software para obter os requisitos de teste. Ou seja, a especificação do programa é utilizada com o objetivo de selecionar os casos de teste para cada subconjunto do domínio de entrada. Nessa técnica, o programa é visto como uma “caixa preta” e nenhum detalhe de implementação é fornecido ao testador. Essa técnica depende da qualidade dos casos de teste selecionados, bem como da qualidade da especificação do sistema em teste (FABBRI *et al.*, 2016). A técnica de teste estrutural, também conhecida como “teste caixa branca”, utiliza a estrutura lógica do programa como recurso para derivar os requisitos de teste. Por meio da aplicação dos critérios dessa técnica, é possível testar os caminhos lógicos

do programa. Os principais critérios dessa técnica são baseados na complexidade, no fluxo de controle e no fluxo de dados do programa em teste (BARBOSA *et al.*, 2016). Os critérios da técnica baseada em defeitos são caracterizados pela utilização dos defeitos recorrentes no processo de desenvolvimento de software para derivar requisitos de teste. O teste de mutação, também chamado de análise de mutantes, é o critério mais conhecido da técnica baseada em defeitos (DELAMARO *et al.*, 2016). No teste de mutação, diversos programas alternativos são criados a partir da modificação do programa original por meio da inserção de pequenos desvios sintáticos.

2.3.1.1 Teste de Especificação

A especificação de um sistema define o comportamento e as características que as implementações devem assumir (MILLER; STROOPER, 2012; SIMÃO, 2016). Uma especificação pode ser descrita em linguagem natural, como português ou inglês. No entanto, esse tipo de linguagem é geralmente ambígua e imprecisa. Como alternativa, têm-se as especificações formais que, uma vez bem definidas, permitem aumentar a compreensão do sistema e uma análise sistemática de suas propriedades (MILLER; STROOPER, 2003; SIMÃO, 2016). Segundo Miller e Strooper (2003), testar uma especificação consiste em interpretá-la ou executá-la com o objetivo de encontrar defeitos. Uma característica do teste de especificação é a identificação de *sequências de teste* que exercitem transições entre estados de um modelo, por exemplo, de uma *Máquina de Estados Finitos* (MEF) (MILLER; STROOPER, 2012). Segundo Simão (2016), as principais sequências de teste são: sequências de sincronização, conjunto de caracterização, sequências de distinção e sequência única de entrada e saída. A partir da especificação testada, o modelo da especificação pode ser utilizado como o oráculo para o teste de programa. Esse tipo de teste é chamado de **teste baseado em modelos**, no qual artefatos podem ser derivados automaticamente para o teste de programa, por exemplo, casos de teste (SIMÃO, 2016; TRETMANS, 2008; AARTS *et al.*, 2014).

2.3.2 Processo de Teste de Software

Para apresentar o processo de teste de software, será considerado o modelo em níveis definido em ISO/IEC/IEEE (2013b). Nele o processo de teste é dividido nos níveis organizacional, de gerenciamento e dinâmico, como apresentado na Figura 9.

Figura 9 – Processo de teste em níveis.

Fonte: Adaptada de [ISO/IEC/IEEE \(2013b\)](#).

No Processo de Teste de Software em Nível Organizacional, são definidas a criação e a manutenção das especificações de teste em nível organizacional, tais como, políticas, estratégias, processos e procedimentos que serão utilizados em uma organização e considerado em todos os projetos dela ([ISO/IEC/IEEE, 2013b](#)). Em nível de gerenciamento, toda a atividade de teste é projetada, por exemplo, as fases de teste, tipos de teste a serem conduzidos. Nesse nível, têm-se os subprocessos de planejamento, monitoramento e controle e conclusão da atividade de teste ([ISO/IEC/IEEE, 2013b](#)). No terceiro nível, o dinâmico, é realizada uma ou mais fases de teste (por exemplo, integração ou sistema) utilizando as técnicas (por exemplo, funcional ou estrutural) de teste. Fazem parte desse nível, projeto de implementação, configuração do ambiente e manutenção do processo, execução do processo e notificação de incidentes de teste.

2.3.2.1 Processo de Teste em Nível Organizacional

Especificações de teste em nível organizacional contém informações que não dependem de um projeto específico, ou seja, aplicam-se a um ou mais projetos dentro de uma mesma organização ou consórcio ([VEENENDAAL, 2012](#); [ISO/IEC/IEEE, 2013c](#)). Dois exemplos de artefatos de teste nesse nível são a política de teste e a estratégia de teste organizacional, descritos a seguir ([ISO/IEC/IEEE, 2013c](#)):

- A política de teste define os objetivos e os princípios relacionados a teste de software que deverão ser aplicados a toda organização; contudo, não são apresentados detalhes técnicos de como a atividade de teste deverá ser aplicada. Em uma especificação de política de

teste são descritos o escopo, referências internas e externas ao documento, um glossário, o processo de teste a ser seguido, a estrutura da organização de teste (tais como, papéis, treinamento, código de ética a ser seguido pelos profissionais de teste), normas, o valor da atividade de teste (considerando o retorno do investimento) e uma especificação para melhoria contínua do processo de teste.

- Estratégia de teste em nível organizacional define, estabelecendo diretrizes, como o teste de software deve ser conduzido em uma organização, por exemplo, como os objetivos definidos na política de teste serão aplicados. Apesar da estratégia ser um documento mais técnico que a política de teste, ele é transversal a um ou mais projetos de teste, no entanto, uma mesma organização pode ter uma ou mais estratégias de teste. Nessa estratégia, são definidos processos e subprocessos de teste, critério de priorização, a forma que a atividade de teste deve ser reportada, automatização e ferramentas, gerenciamento de configuração, produtos de trabalho, gerenciamento de incidentes, critérios de entrada e saída.

2.3.2.2 Processo de Teste em nível Gerencial

Especificações do gerenciamento de processo de teste são constituídas de três artefatos, a saber, plano de teste, relatório de status de teste, relatório de conclusão de teste, como descrito a seguir ([ISO/IEC/IEEE, 2013c](#)):

- O Plano de Teste descreve as decisões realizadas durante o planejamento da atividade de teste e as adaptações realizadas nesse durante a condução da atividade. Planos de teste podem estar relacionados a um ou mais projetos de teste de maneira específica (por exemplo, plano para o teste de sistema, plano para o teste funcional) ou um único plano mestre. Um plano de teste constitui-se de informações relacionadas a escopo, referências internas e externas, glossário, projeto, processo e subprocesso, itens a serem testados, escopo de teste, restrições, envolvidos, comunicação de teste, riscos de produto e de projeto, estratégia de teste, entregáveis (“*deliverables*”), técnicas de teste, critérios de conclusão, métricas a serem coletadas, requisitos de ambiente, teste de regressão, critérios de suspensão e reinício, equipe de teste, papéis, atividades e responsabilidades, necessidades de treinamento, calendário.

A ISO/IEC/IEEE (2013b) define o processo retratado na Figura 10 para o planejamento de teste. Esse processo parte de um entendimento adequado do contexto, passa pela identificação de riscos e sua mitigação, elaboração do plano de teste até a aprovação e disponibilização do plano. Segundo a norma, esse processo está estruturado sequencialmente, mas pode tornar-se iterativo a partir do andamento e das necessidades do projeto.

Figura 10 – Processo de Planejamento de Teste.

Fonte: Adaptada de [ISO/IEC/IEEE \(2013b\)](#).

- Relatório de Status de Teste provê informações sobre o status do teste de software em um período de tempo específico. Para descrever o relatório de status, são descritos escopo, glossário, status de teste, período de tempo, progresso da atividade de teste em comparação com o plano de teste, fatores que estão bloqueando o teste, medidas de teste, riscos não previstos e o que deve ser feito no próximo período de teste.
- Relatório de Conclusão de Teste sumariza o que foi realizado no teste de software. As informações contidas no relatório de conclusão são escopo, referências internas e externas, glossário, resumo da atividade de teste realizada, desvios do plano de teste, avaliação do que foi completado, fatores que bloquearam o progresso, medidas de teste, riscos que não foram tratados, lista de todos produtos entregados, artefatos reutilizáveis, lições aprendidas.

2.3.2.3 Processo de Teste Dinâmico

As informações de teste de software em nível dinâmico são divididas nas especificações de projeto de teste (*test design*), caso de teste, procedimento de teste, requisitos de dados de

teste, requisitos de ambiente de teste, relatório de preparação de dados, relatório de preparação de ambiente, documentação de execução de teste (divido em resultados atuais, resultados de teste, *log* de execução, relatório de incidentes) (ISO/IEC/IEEE, 2013c). A descrição dos itens é (ISO/IEC/IEEE, 2013c):

- Projeto de teste identifica recursos a serem testados e as condições derivadas da base de teste para cada um dos recursos, constituindo o primeiro passo para definição de casos de teste.
- Especificação de casos de teste identifica os itens a serem cobertos e os respectivos casos de teste, considerando a priorização dos itens e dos casos de teste para a execução. Casos de teste devem ser descritos por objetivo, prioridade, rastreabilidade, precondições, entrada, saída esperada, resultados atuais e resultado de teste.
- Especificação de procedimento de teste descreve casos de teste listados em ordem de execução em conjuntos de teste, juntamente com as ações necessárias para definir pré-condições e pós-condições. Essa especificação também contém prioridades, ações para preparar a execução de casos de teste, relacionamento com outros procedimentos, ações a serem encerradas após a execução dos procedimentos de teste.
- Requisitos de dados de teste descreve os dados necessários para executar os procedimentos de teste, em que são retratados valores, responsáveis por tornar os dados disponíveis, período de tempo em que o dado deverá estar disponível, os dados que devem ser redefinidos durante o teste, dados de teste que deverão ser arquivados ou descartados. O relatório de preparação de dados de teste descreve o cumprimento de cada requisito de dados de teste, em que também é descrito o status dos dados requeridos.
- Requisitos de ambiente define as propriedades do ambiente necessárias para executar os procedimentos de teste, como software, hardware, ferramentas, segurança, acessórios. Além disso, deve ser descrito o responsável em tornar disponível o ambiente de teste. O relatório de preparação de ambiente descreve o cumprimento de cada requisito de ambiente.
- Nos resultados atuais são armazenados os resultados da execução dos casos de teste em um procedimento. O resultado de teste é armazenado se os resultados obtidos correspondem aos resultados esperados, quais desvios foram observados ou se o houveram impedimentos para planejamento de casos de teste.
- No *log* de execução, são descritos os eventos durante a execução dos procedimentos de teste e seus respectivos impactos. O relatório de incidente contempla quaisquer problemas identificados durante a atividade de teste que requerem atenção. Para relatar o incidente devem ser armazenadas informações quando ocorreram, quem identificou o incidente,

o contexto em que o incidente foi observado, descrição do incidente, risco relacionado, severidade, prioridade, status (aberto, foi solucionado) do incidente.

2.3.3 Modelo de Maturidade em Teste de Software

Para suprir as necessidades do desenvolvimento de software com qualidade, diversos modelos de maturidade para o processo de desenvolvimento de software foram propostos, como Modelo de Maturidade de Capacidade - Integração (CMMi, do inglês, *Capability Maturity Model - Integration*) (CMMI Product Team, 2010) e Melhoria de Processo do Software Brasileiro (MPS-BR) (SOFTEX, 2012). De forma semelhante, foi proposto o Modelo de Maturidade em Teste de Software (TMMi, do inglês, *Test Maturity Model Integration*) (VEENENDAAL, 2012). Esse foi estabelecido como diretriz e framework de referência para a melhoria de processos de teste de software e atualmente é mantido pela TMMi Foundation⁵. O modelo é baseado no CMMi, que é um modelo de melhoria de processo para propósito geral no de desenvolvimento de software e no *Test Maturity Model* (TMM) (BURNSTEIN *et al.*, 1998). O TMMi e o CMMi contribuem de forma diferente e complementar no processo de desenvolvimento (KIM; KIM, 2014; VEENENDAAL, 2012). A estrutura do TMMi é apresentada na Figura 11.

Figura 11 – Estrutura do TMMi

Fonte: Adaptada de Veenendaal (2012).

O TMMi possui uma arquitetura em cinco níveis de maturidade divididos em 16 áreas de processo, 50 objetivos específicos e 188 práticas específicas. À medida que uma organização

⁵ <<http://www.tmmi.org/>>

consegue cobrir às áreas de processos, ela pode alcançar os níveis superiores gradualmente e, assim, partir de um processo de teste sem qualquer definição formal para chegar em um processo gerenciado, definido, mensurado e otimizado. As áreas de processos possuem objetivos gerais que são requisitos para que uma empresa atingir o nível especificado. Cada objetivo geral possui objetivos específicos e práticas específicas que são exemplos a serem realizados para alcançar o propósito da área de processo. Por exemplo, o Nível 3 - Definido - tem cinco áreas de processo, uma delas é a Organização de Teste que contém cinco objetivos específicos. Um dos objetivos específicos é estabelecer papéis de teste para especialistas de teste. Para isso, são estabelecidas descrições de trabalho e atribuições de membros da equipe como práticas específicas.

Através dos níveis do TMMi todo o processo de teste, desde o planejamento em nível organizacional até o teste de regressão e avaliação de resultados, é influenciado e pode ser avaliado. As características de cada nível são:

- **Nível 1 - Inicial:** não há organização na execução dos testes e eles são conduzidos de forma *ad-hoc*. A qualidade da atividade de teste nesse nível depende inteiramente do empenho da equipe de teste. O objetivo do teste nesse nível é mostrar que o software funciona sem grandes falhas;
- **Nível 2 - Gerenciado:** a organização possui estratégias e há um planejamento para que as atividades de teste sejam conduzidas em vários níveis (por exemplo, unidade, integração, sistema). As áreas de processo desse nível são: i) Política e Estratégia de Teste; ii) Planejamento de Teste; iii) Monitoramento e Controle de Teste; iv) Projeto e Execução de Teste; v) Ambiente de Teste. O objetivo do teste nesse nível é verificar se o software satisfaz aos requisitos do usuário;
- **Nível 3 - Integração:** o teste está integrado no ciclo de vida do software. Nesse nível, o planejamento de teste é iniciado nas primeiras etapas do desenvolvimento do software, é realizada a revisão por pares e o teste não funcional é realizado. As áreas de processo desse nível são: i) Organização de Teste; ii) Programa de Treinamento; iii) Ciclo de Vida e Integração de Teste; iv) Teste Não-Funcional; v) Revisão em pares. No Nível 3, os profissionais de teste são treinados e há regulamentação do testador como profissão. Além disso, os processos de teste são descritos e documentados de forma mais rigorosa que nos níveis anteriores;
- **Nível 4 - Gestão e Medição:** o processo de teste está bem definido, fundamentado e pode ser medido. As áreas de processo desse nível são: i) Medição de Teste; ii) Avaliação de Qualidade de Produtos; iii) Revisão em Pares Avançado. Nesse nível os critérios de qualidade são utilizados para avaliar os produtos desenvolvidos e as medições são incorporadas no repositório da organização, contribuindo, assim, para dar suporte a tomada de decisão baseada em evidências; e

- **Nível 5 - Otimizado:** há um controle rígido do processo de teste, em que ele é otimizado e efetivo, sendo que os métodos e técnicas utilizados estão em constante evolução. Nesse nível, a prevenção de defeitos é realizada com o objetivo de encontrar a principal causa dos defeitos no ciclo de vida do desenvolvimento. As áreas de processo nesse nível são: i) Prevenção de Defeitos; ii) Controle de Qualidade; iii) Otimização de Processo de Teste. Além disso, o reuso de artefatos produzidos durante a atividade de teste é projetado.

2.3.4 UTP - Um perfil de Teste para a UML

Nesta seção, será descrita a extensão da **UML** para modelar informações relevantes para a atividade de teste, denominada **UTP**. Considerando a complexidade atual dos sistemas de software e o teste de alto nível a ser realizado, o teste baseado em modelos foi introduzido (BAKER *et al.*, 2007). A **UTP** é um perfil (“*profile*”) para a **UML** que foi desenvolvido para potencializar a descrição de conceitos de teste de software por meio de modelos (OMG, 2012). Esse perfil adiciona à **UML** estereótipos, atributos e restrições relacionados à atividade de teste. De acordo com Baker *et al.* (2007), os conceitos de teste na **UTP** ajudam a estruturar os testes, definir procedimentos e os dados relacionados a eles, contribuindo, assim, para combinar e integrar o desenvolvimento de sistemas e teste via **UTP**.

A **UTP** é uma linguagem padronizada baseada na **UML** para projetar, visualizar, especificar, analisar, construir e documentar artefatos utilizados e requeridos em teste de software (OMG, 2012). Os conceitos contidos na **UTP** são divididos em (OMG, 2012): arquitetura de teste, dados de teste, comportamento dos testes e gerenciamento de teste. Na arquitetura são descritos conceitos sobre o ambiente de teste a configuração de teste. Em dados de teste, têm-se estruturas para descrição, armazenamento e seleção de dados de teste. Em comportamento de teste, o comportamento dinâmico de casos de teste e procedimentos de teste. No gerenciamento de teste, são descritos conceitos para o gerenciamento de atividades de teste específicas. Além disso, há extensões a **UTP** como Costa *et al.* (2013) para coordenação e Ali *et al.* (2014) para riscos.

2.4 Considerações Finais

Para desenvolver sistemas de software com qualidade, é necessário que algumas atividades sejam rigorosamente definidas, projetadas e executadas, dentre elas estão o teste de software e o projeto de arquitetura de software. Em vista disso, neste capítulo, foram explicitados os conceitos acerca de arquitetura de software teste de software.

Conceitos sobre arquitetura de software foram explanados, mas, principalmente, no tocante a arquiteturas de referência, que tem uma relação mais estreita com este trabalho. Foi explorada a importância de um projeto arquitetural, visto que a arquitetura influencia o desenvolvimento de um software com qualidade. De forma complementar, arquiteturas de

referência apresentam-se como um meio para elevar a qualidade de projetos de arquiteturas de software em termos de reúso, padronização, evolução. Para elucidar esses conceitos, foram apresentados exemplos de arquiteturas de referência e iniciativas para tornar o estabelecimento de arquiteturas de referência sistemático e eficiente, dentre elas, foi destacado o [RAModel](#).

Para desenvolver um sistema software com qualidade, é necessário também que existam atividades para evidenciar que ele se comporta como o esperado pelo usuário. Nessa perspectiva, o teste de software é reconhecido como uma importante atividade no processo de desenvolvimento de software. Em virtude da importância do teste, diversas técnicas, critérios e ferramentas foram propostas para apoiá-lo. Em particular, foram destacados o processo e a documentação de teste apresentado pela norma ISO/IEC/IEEE 29119 e o [TMMi](#). No próximo capítulo, será apresentada a extensão e a descrição do [RAModel](#).

RAMODEL_T: UMA EXTENSÃO DO RAMODEL COM INFORMAÇÕES DE TESTE

3.1 Considerações Iniciais

Segundo [Andrade et al. \(2013\)](#), propor ferramentas ou técnicas para o teste de software não é o bastante. É importante que existam meios eficientes para que o conhecimento produzido durante a atividade de teste seja reutilizado. Esses autores afirmam que o teste de software é uma atividade intensiva em conhecimento, pois, ao executá-la, são definidos ([ANDRADE et al., 2013](#)): (i) planos de teste; (ii) estratégias de teste; (iii) casos de teste; (iv) programas ou partes deles são priorizados; (v) teste de regressão; e (vi) técnicas de teste. Reutilizar esse conjunto de conhecimento contribui para: prover continuidade entre projetos de teste, padronizar o processo de teste e adaptar *stakeholders* ao processo de teste.

Dentre os meios que promovem reuso, têm-se as arquiteturas de referência. [RAModel](#) é um modelo utilizado para facilitar e sistematizar o estabelecimento desses tipos de arquiteturas. Assim, esse modelo contém os elementos essenciais que as arquiteturas de referência devem contemplar. No entanto, foi identificado que o [RAModel](#) ([NAKAGAWA; OQUENDO; BECKER, 2012](#); [NAKAGAWA; OQUENDO; MALDONADO, 2014](#)) não possui informações sobre teste de software. Essa constatação foi realizada após uma análise do modelo e de reuniões com seus autores. Assim, neste projeto de mestrado o [RAModel](#) é estendido com elementos de teste de software para que arquiteturas de referência com informações de teste sejam projetadas, e essas informações sejam planejadas nas fases iniciais do processo de desenvolvimento de software. Com esse tipo de informação, a arquitetura de referência poderá contribuir para o planejamento de teste.

O [RAModel](#) estendido com elementos de teste de software recebeu o nome de [RAModel_T](#). Os elementos de teste foram obtidos por meio de documentos importantes da área de teste de

software, que são o **TMMi** e a norma ISO/IEC/IEEE 29119. A extensão do modelo foi realizada em três passos, seguindo um procedimento semelhante ao adotado por Nakagawa, Oquendo e Becker (2012). Foram adicionados ao **RAModel** 23 elementos de teste organizados em três grupos de teste e oito subgrupos de teste. Para descrever como as informações de teste podem ser representadas, utilizou-se a estrutura para *viewpoints* da norma ISO/IEC/IEEE-42010 (2011). A estrutura é: i) nome; ii) visão geral; iii) interesses; iv) *stakeholders*; e v) tipos de modelos. Foi definido um *viewpoint* para cada grupo de teste, sendo que para dois deles foram apresentados exemplos de diagramas utilizando a UTP.

Este capítulo está organizado da seguinte forma. Na Seção 3.2, o procedimento adotado para definir e estender o **RAModel** é descrito. Na Seção 3.3, o **RAModel_{TI}** é descrito em detalhes, abordando cada elemento de teste, grupo de teste e subgrupo de teste. Uma comparação entre o **TMMi** e a extensão do **RAModel** é apresentada na Seção 3.4. Por fim, na Seção 3.5, apresenta-se as considerações finais.

3.2 Definição do **RAModel_{TI}**

A extensão do **RAModel** foi realizada em três passos, seguindo um procedimento semelhante ao adotado por Nakagawa, Oquendo e Becker (2012). Passo 1 - Identificar Fontes de Informação: foram considerados como fonte de informação o **TMMi** e a família de normas ISO/IEC/IEEE 29119. Passo 2 - Identificar Elementos de Teste: as fontes de informação foram analisadas procurando o que os documentos definiram como imprescindível ao planejamento de teste. Os elementos de teste foram classificados em três grupos de teste e oito subgrupos de teste. Passo 3 - Projetar o **RAModel_{TI}**: foi incorporado a um pacote denominado Teste de Software no grupo Elementos Transversais do **RAModel**. O procedimento adotado para estender o **RAModel** é ilustrado na Figura 12.

Figura 12 – Procedimento para definir o **RAModel_{TI}**

Fonte: Elaborada pelo autor.

Na Figura 13, é retratada uma visão geral do **RAModel_{TI}**. Nela, a extensão é destacada por um retângulo de bordas tracejadas. Além disso, foi adicionado o elemento Gerenciamento

de Configuração, que, segundo Nakagawa (2006), também é transversal no desenvolvimento de software. Contudo, esse elemento não foi explorado em detalhes neste trabalho.

Figura 13 – Visão geral do *RAModel_{TI}* (extensão do (NAKAGAWA; OQUENDO; BECKER, 2012)).

Fonte: Elaborada pelo autor.

Para organizar os elementos de teste no modelo, a seguinte nomenclatura foi adotada:

- **Grupo de Teste:** agrupamento de subgrupos que possuem um objetivo em comum, nesse caso, informações gerenciais, informações de infraestrutura e projeto de teste;
- **Subgrupo de Teste:** estrutura de organização de elementos de teste em um mesmo grupo;
- **Elemento de Teste:** tópico da área de teste de software que é relacionado ao planejamento de teste e que, a partir dele, poderão ser derivadas informações de teste;
- **Descrição:** uma descrição textual do elemento em si; e
- **Atributo:** item ou conjunto de itens que devem estar incluídos na documentação das informações de teste.

Na Figura 14 é apresentada como as nomenclaturas são estruturadas na organização dos elementos do *RAModel_{TI}*.

Figura 14 – Estrutura dos elementos de teste no RAModel_{TI}

Fonte: Elaborada pelo autor.

3.3 Descrição do RAModel_{TI}

Uma visão geral dos elementos de teste é ilustrada na Figura 15, em que os grupos de teste, subgrupos de teste e elementos de teste são mapeados. Nas próximas seções, cada um deles serão descritos.

Figura 15 – Estrutura geral do RAModel_{TI}

Fonte: Elaborada pelo autor.

3.3.1 Gerenciamento de Teste

Neste grupo de teste estão os elementos de teste de nível gerencial. Os elementos desse grupo guiam a definição das informações dos demais subgrupos, estabelecendo a política, o processo de teste a ser seguido, as diretrizes do domínio, o modo como poderá ser verificada se a atividade de teste está sendo conduzida com a qualidade esperada e quais os fundamentos para uma melhoria programada dessa atividade. Esse grupo é composto pelos subgrupos de teste Política de Teste, Performance de Teste e Melhoria de Teste. Os subgrupos de teste são descritos nos Quadros 5 – 7. Cada elemento desse grupo de teste recebeu um identificador com EG (Elemento de Gerenciamento) e um número inteiro, por exemplo “EG.1”.

3.3.1.1 Política de Teste

Neste subgrupo de teste a política de teste é descrita. Ela é composta pelos elementos de teste: objetivo de teste, processos de teste e normas regulamentares. Esse subgrupo é importante para definir o escopo em que a atividade de teste será conduzida, considerando o objetivo geral da organização. Esse subgrupo de teste é apresentado no Quadro 5. No *TMMi*, a política de teste é definida no nível de maturidade 2.

Quadro 5 – Descrição do subgrupo Política de Teste

EG.1 Objetivo de Teste
<p>Descrição Descreve a finalidade, as metas e o escopo geral dos testes dentro da organização. Indica a posição da organização sobre o fato de os testes serem realizados e o que eles procuram alcançar.</p> <p>Atributos - Objetivo Geral - Objetivos Específicos</p>
EG.2 Processos de Teste
<p>Descrição Apresenta um ou mais processos de teste a serem executados na condução da atividade de teste.</p> <p>Atributos - Processo Padrão de Teste - Sub-processo(s) de Teste</p>
EG.3 Normas Regulamentares
<p>Descrição Apresenta as normas de teste que são mandatórias e as normas que são opcionais para que se cumpra o objetivo definido em EG.1 no domínio do sistema em teste.</p> <p>Atributos - Norma(s) mandatória(s) - Norma(s) opcional(is)</p>

3.3.1.2 Performance de Teste

Neste subgrupo de teste a performance de teste é descrita. Ela é composta pelos elementos de teste: objetivos para performance, critérios de aceitação e indicadores para performance. Por meio desse subgrupo de teste, a performance da atividade de teste poderá ser medida, aprovada ou rejeitada. Assim, o processo de teste poderá ser avaliado considerando as informações relevantes para o domínio. O subgrupo de teste é ilustrado no Quadro 6. No TMMi, a performance de teste é definida no nível de maturidade 2.

Quadro 6 – Descrição do subgrupo Performance de Teste

EG.4 Objetivos para Performance
<p>Descrição Estabelece uma diretriz clara do que será medido e o que será comunicado em relação ao desempenho da atividade de teste que está sendo conduzida e da planejada. Esse elemento é baseado nas informações descritas em EG.1, EG.2 e EG.3.</p> <p>Atributos - Conjunto de objetivo(s) para performance</p>
EG.5 Critérios de Aceitação
<p>Descrição Estabelece critérios para aceitar ou recusar se a performance apresentada pela atividade teste atingiu os objetivos apresentados em EG.4.</p> <p>Atributos - Conjunto do(s) critério(s)</p>
EG.6 Indicadores de Performance
<p>Descrição Define os meios a serem utilizados para coletar e medir se o processo de teste está adequado em relação a EG.1 e EG.4. Indicadores podem ser definidos com base nas informações descritas em EG.1, EG.2, EG.3, EG.7 e EG.9.</p> <p>Atributos - Conjunto de Indicador(es)</p>

3.3.1.3 Melhoria de Teste

Neste subgrupo de teste, a melhoria de teste é descrita. Ela é composta pelos elementos de teste: pontos fortes, pontos fracos e conjunto de melhorias. Esse subgrupo de teste relaciona informações relevantes para uma melhoria programada do processo de teste, proporcionando meios para ajustar o processo de teste. Assim, verifica-se se existem vulnerabilidades a serem tratadas e pontos fortes do processo a serem mantidos. No Quadro 7, são descritos elementos do subgrupo de melhoria de teste. No TMMi, a melhoria de teste é definida no nível de maturidade 2.

Quadro 7 – Descrição do subgrupo Melhoria de Teste

EG.7 Pontos Fortes e Pontos Fracos
<p>Descrição Apresenta um conjunto de pontos fortes e de pontos fracos do processo de teste (definido no elemento EG.2) em execução.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto de Ponto(s) Forte(s) - Conjunto de Ponto(s) Fraco(s)
EG.8 Conjunto de Melhorias
<p>Descrição Define um conjunto de melhorias que devem ser aplicadas ao processo de teste de software. Essas melhorias são relacionadas à maturidade do processo de teste definido em EG.2, em EG.7, e à política de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Melhoria(s) no processo de teste - Ponto(s) forte(s) relacionado(s) - Ponto(s) fraco(s) relacionado(s)
EG.9 Plano de Melhoria
<p>Descrição Estabelece o método para garantir a melhoria contínua do(s) processo(s) de teste. Esse elemento é definido com base nos elementos EG.1, EG.7, EG.8 e na experiência dos <i>stakeholders</i> acerca do sistema em teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto de melhorias - Critério(s) de priorização - Cronograma - Envolvido(s) - Ação(ões) - Risco(s) associado(s)

3.3.1.4 Modo de Descrição

A forma sugerida de descrição das informações relacionadas ao grupo de teste é apresentada no Quadro 8.

Quadro 8 – *Viewpoint* para descrição de Gerenciamento de Teste

Nome	<i>Viewpoint</i> de Gerenciamento de Teste
Visão Geral	Nesse <i>viewpoint</i> , são descritas informações de teste relacionadas ao grupo de elementos de teste. A descrição dessas informações poderá ser feita por meio de linguagem natural, seguindo a ordem em que os subgrupos de teste foram apresentados nesta seção.
Interesses	<ul style="list-style-type: none"> • Descrever informações relacionadas aos elementos do subgrupo Política de Teste. • Descrever informações relacionadas aos elementos do subgrupo Performance de teste. • Descrever informações relacionadas aos elementos do subgrupo Melhoria de Teste.
Envolvidos	Desenvolvedores, Gerentes de Qualidade, Testadores, Arquitetos de teste, Arquitetos de Software, Gerentes do Sistema e Usuários do Sistema.
Tipos de Modelo	Nesse <i>viewpoint</i> , as descrições serão feitas por meio de texto em linguagem natural.

Nele, apresenta-se o *Viewpoint* Gerenciamento de Teste, no qual o interesse é descrever as informações relacionadas aos elementos de teste. Por isso, na linha Interesses do quadro foram listados os subgrupos de teste de Gerenciamento de Teste. Os envolvidos na descrição são, em resumo, os profissionais da área gerencial, da área de teste, da área de arquitetura e usuários (ou pelo menos alguém que faça o papel de usuário final). Nesse *Viewpoint*, apenas a linguagem natural foi sugerida para descrever as informações de teste.

3.3.2 *Infraestrutura de Teste*

Este grupo de teste é composto por dois subgrupos de teste para descrever a infraestrutura necessária para condução da atividade de teste. A infraestrutura, nesse grupo de teste, constitui-se de itens fundamentais para a condução da atividade de teste, tais como simuladores, software ou hardware, com vistas a descrever um ambiente adequado para condução da atividade. A infraestrutura como uma organização dos elementos de software e de hardware do sistema em teste/desenvolvimento foi definido no *RAModel* original. Dada a importância da automatização na atividade de teste, um subgrupo de teste elenca as ferramentas de teste necessárias para definir um plano de automatização. Os subgrupos de teste que compõem esse grupo são Ambiente de Teste e Automatização de Teste. Os subgrupos de teste são descritos nos Quadros 9 e 10. Cada elemento desse grupo de teste recebeu um identificador com EI (Elemento de Infraestrutura) e um número inteiro, por exemplo “EI.1”.

3.3.2.1 *Ambiente de Teste*

Neste subgrupo de teste o ambiente de teste é descrito. Ele é composto pelos elementos de teste requisitos de ambiente e requisitos de profissionais. Esse subgrupo de teste descreve itens

requeridos para que a atividade de teste seja conduzida com qualidade, de forma gerenciável e repetível. Nesses itens são considerados hardware, software e qualificação pessoal necessária dos envolvidos. No Quadro 9, são descritos os elementos desse subgrupo de teste. No *TMMi*, o ambiente de teste é definido no nível de maturidade 2.

Quadro 9 – Descrição do subgrupo Ambiente de Teste

EL.1 Requisitos de Ambiente
Descrição Define itens e condições para preparar e adequar o ambiente em que a atividade de teste será conduzida. Esses itens podem ser software, hardware, simuladores, definidos com base nas informações descritas em EG.2, EG.3.
Atributos <ul style="list-style-type: none">- Documentos relacionados- Itens mandatórios- Itens opcionais- Riscos associados- Requisitos de sistema associados- Requisitos de teste associados
EL.2 Requisitos de Profissionais
Descrição Estabelece quais profissionais podem compor o time de testadores para o sistema em teste. As habilidades, competências e formações também podem ser apresentadas. Essas informações são baseadas em EG.1, EG.2, EG.3.
Atributos <ul style="list-style-type: none">- Profissionais necessários- Certificações/formações necessárias- Papeis dos testadores

3.3.2.2 Automatização de Teste

Neste subgrupo de teste a automatização de teste é descrita. Ela é composta pelos elementos de teste ferramentas de teste e abordagem para automatização. Esse subgrupo de teste contempla elementos de teste que contribuem para definir como a atividade de teste pode ser automatizada. Os elementos são descritos no Quadro 10. No *TMMi*, a automatização de teste é definida no nível de maturidade 2. O atributo integração de ferramentas de teste foi adicionado em decorrência do resultado do estudo de viabilidade.

Quadro 10 – Descrição do subgrupo Automação de Teste

EI.3 Ferramentas de Teste
<p>Descrição Define ferramentas a serem utilizadas para gerenciar, executar e reportar a atividade de teste. Essas ferramentas são baseadas no conhecimento a respeito do sistema em teste e nas informações descritas em EG.1, EG.2, EG.3 e EG.4.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Ferramentas - Requisitos de teste associados - Condições de teste associadas
EI.4 Abordagem para Automação
<p>Descrição Define como as ferramentas descritas em EI.3 podem ser utilizadas e relacionadas às informações dos elementos EP.7, EP.8, EP.9, EP.10.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Procedimentos de teste a serem automatizados - Fases de teste - Cronograma - Envolvidos - Ferramentas de teste relacionadas - Requisitos de teste - Integração de ferramentas de teste

3.3.2.3 Modo de Descrição

A forma sugerida de descrição das informações relacionadas ao grupo de teste é apresentada no Quadro 11, que ilustra o *Viewpoint* Infraestrutura de Teste. O interesse dele é descrever as informações relacionadas aos elementos da infraestrutura de teste. Por isso, na linha Interesses do quadro foram listados os subgrupos de teste do grupo Infraestrutura de Teste. Os envolvidos na descrição são profissionais das áreas de teste de software, arquitetura de software e gerencial. As notações sugeridas são a linguagem natural e UTP. Um exemplo de diagrama representado com base *Viewpoint Infraestrutura de Teste* é apresentado na Figura 16.

Quadro 11 – *Viewpoint* para descrição de Gerenciamento de Teste

Nome	<i>Viewpoint</i> de Infraestrutura de Teste
Visão Geral	Nesse <i>viewpoint</i> é descrito como apresentar as informações de teste descritas nesse grupo de elementos.
Interesses	<ul style="list-style-type: none"> • Descrever informações relacionadas aos elementos do subgrupo Ambiente de Teste. • Descrever informações relacionadas aos elementos do subgrupo Automação de teste.
Envolvidos	Desenvolvedores, Gerentes de Qualidade, Testadores, Arquitetos de teste, Arquiteto de Software e Gerentes do Sistema.
Tipos de Modelo	Nesse <i>viewpoint</i> as descrições serão feitas por meio de texto em linguagem natural, diagrama de implantação e UTP.

Sistema em Teste contém um conjunto de sistemas, sistema ou, pelo menos, partes dele que devem ser testados. Nesse caso, quatro elementos de software foram utilizados como exemplo. Esses podem ser quaisquer elementos testáveis do sistema, tais como componentes, funções, serviços ou modelos executáveis. Para caracterizar esse pacote, utiliza-se o estereótipo «SUT» do UTP. Ambiente de Teste é o pacote em que o sistema em teste é relacionado com ferramentas, componentes de teste, envolvidos e níveis de teste. Por exemplo, para testar um elemento de software 3 utiliza-se uma ferramenta na fase de teste de unidade. Foram utilizados os estereótipos «TextContext» para caracterizar cada contexto. Gestão de Conhecimento contém informações sobre boas práticas e conhecimento que podem contribuir para a definição do ambiente de teste. Esse pacote contempla informações sobre a própria arquitetura como também informações de fontes externas. Artefatos contempla os artefatos que poderão ser utilizados durante a atividade de teste, tais como as ferramentas de automatização, funções, casos de teste, *scripts* de teste, documentação. Quanto às classes com o estereótipo «TestContext» em uso nessa descrição, são originais da UTP: nível de teste e versão compatível com o sistema. Os seguintes atributos foram adicionados às classes com o estereótipo «TestContext» para atender o *RAModel_{TI}*, que são ferramentas, componentes de teste, requisitos de teste relacionados, requisitos de sistema relacionados, riscos relacionados ao contexto de teste, envolvidos.

Figura 16 – Diagrama que compõe a Visão de Ambiente de Teste

Fonte: Elaborada pelo autor.

3.3.3 Projeto de Teste

A partir da definição das informações gerenciais de teste e da infraestrutura necessária para que a atividade seja conduzida, informações concretas para projetos de teste específicos são definidas. As informações nesse grupo são definidas a partir de um gerenciamento dos defeitos no domínio e no contexto em que o sistema está sendo implementado. Riscos de projeto e de produto são listados com base em defeitos, riscos conhecidos, experiência de usuários e da literatura, buscando uma caracterização dos riscos associados a cada elemento do software que será desenvolvido. Por fim, o grupo Estratégia de Teste é definido, com requisitos e condições de teste, técnicas de teste, casos de teste ordenados em procedimentos de teste. Esse grupo reúne os subgrupos de teste Gerenciamento de Defeitos, Análise de Riscos e Estratégia de Teste. Os subgrupos de teste são descritos nos Quadros 12 – 17. Cada elemento desse grupo de teste recebeu um identificador com EP (Elemento de Projeto) e um número inteiro, por exemplo “EP.1”.

3.3.3.1 Gerenciamento de Defeitos

O gerenciamento de defeitos é descrito neste subgrupo de teste. Ele é composto pelos elementos de teste conjunto de defeitos, critérios de priorização e propostas de solução. Esse subgrupo de teste contém elementos que fornecem bases para compreender e prevenir defeitos para sistemas de software no domínio de aplicação. Os elementos desse subgrupo de teste estão no Quadro 12. No TMMi, o gerenciamento de defeitos é definido no nível de maturidade 5. O atributo testes relacionados foi adicionado em decorrência do resultado do estudo de viabilidade.

Quadro 12 – Descrição do subgrupo de teste Gerenciamento de Defeitos

EP.1 Conjunto de Defeitos
<p>Descrição</p> <p>Conjunto dos principais defeitos relacionados ao sistema em teste. O conjunto dos defeitos é obtido por meio de defeitos comuns no desenvolvimento de software, defeitos comuns no domínio, defeitos de versões anteriores do sistema, defeitos obtidos da literatura.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Defeitos - Prioridade - Provável causa - Requisitos relacionados - Identificador
EP.2 Critérios de Priorização
<p>Descrição</p> <p>Define critérios para selecionar os principais defeitos para análise. Esses critérios podem ser definidos a partir de taxonomias, de normas (EG.3), da literatura, do objetivo (EG.1) de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição dos critérios - Defeitos associados

Quadro 13 – Descrição do subgrupo de teste Gerenciamento de Defeitos (Continuação do Quadro 12).

EP.3 Proposta de Solução
<p>Descrição</p> <p>Soluções são propostas para os defeitos listados. Essas soluções podem ser obtidas por meio de padrões (“<i>patterns</i>”), reuso de conhecimento e boas práticas.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Proposta de solução - Defeitos associados - Testes relacionados

3.3.3.2 Análise de Riscos

Neste subgrupo de teste a análise de riscos é descrita. Ela é composta pelos elementos lista de riscos, critérios de priorização e plano de mitigação. Esse subgrupo de teste relaciona riscos que podem afetar sistemas instanciados a partir de uma arquitetura de referência, incluindo riscos de produto, domínio e projeto. Os elementos desse subgrupo de teste são apresentados no Quadro 14. No *TMMi*, a análise de riscos é definida no nível de maturidade 2. O atributo testes relacionados foi adicionado em decorrência do resultado do estudo de viabilidade.

Quadro 14 – Descrição do subgrupo de teste Análise de Riscos

EP.4 Lista de Riscos
<p>Descrição</p> <p>Lista os riscos de sistema e de arquitetura que podem estar relacionados ao sistema em teste. Os riscos podem ser obtidos por meio de taxonomias, ontologias, normas (descritas em EG.3), revisões e mapeamentos sistemáticos.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Riscos - Descrição - Prioridade - Impacto - Probabilidade - Identificador
EP.5 Critérios de Priorização
<p>Descrição</p> <p>Define critérios a serem utilizados para priorizar os riscos apresentados em EP.1. Essa informação é obtida a partir dos objetivos apresentados em EG.1, das normas (EG.3), do conhecimento de domínio e das experiência dos profissionais da equipe de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição dos critérios - Riscos associados

Quadro 15 – Descrição do subgrupo de teste Análise de Riscos (Continuação do Quadro 14.)

EP.6 Plano de Mitigação
<p>Descrição Define um plano, no qual decisões serão tomadas para proteger, evitar, ou minimizar riscos em níveis especificados. Essa informação é baseada nos riscos elencados (EP.4), na literatura, na política de teste e nos critérios de priorização de riscos (EP.5).</p> <p>Atributos</p> <ul style="list-style-type: none"> - Risco - Cronograma - Ações - Requisitos de teste relacionados (Descritos em EP.7) - Testes relacionados

3.3.3.3 Estratégia de Teste

Neste subgrupo de teste a estratégia de teste é definida. Ela é composta pelos elementos de teste: requisitos de teste, condições de teste, técnicas de teste e procedimento de teste. Nesse subgrupo de teste, as informações de teste tornam-se específicas para projetos de software, envolvendo identificação de requisitos, seleção de técnicas de teste, projeto de casos de teste e execução. No TMMi, a estratégia de teste é definida no nível de maturidade 2. O subgrupo de teste é apresentado no Quadro 17.

Quadro 16 – Descrição do subgrupo de teste Estratégia de Teste

EP.7 Requisitos de Teste
<p>Descrição Define itens ou eventos de um componente ou sistema que devem ser testados. Esses requisitos são elencados a partir da análise do sistema, da política de teste, da análise de risco, da gestão de defeitos e de técnicas de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Identificador - Requisitos de teste - Defeitos associados - Riscos associados - Requisitos de sistema associados
EP.8 Condições de Teste
<p>Descrição Define aspectos testáveis de um componente ou sistema, como uma função, transação, característica, como base para o teste. Essas condições podem ser obtidas a partir de requisitos de teste (EP.7), requisitos de sistema e capacidades do sistema.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Itens de teste - Condições - Identificador

Quadro 17 – Descrição do subgrupo de teste Estratégia de Teste

EP.9 Técnicas de Teste
<p>Descrição São elencados técnicas e critérios de teste a serem utilizados para gerar casos de teste aplicáveis a cada item das condições de teste. Essas técnicas devem estar alinhadas com a política de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Técnicas de teste - Requisitos associados - Condições associadas - Identificador
EP.10 Procedimento de Teste
<p>Descrição Define a execução de um ou mais casos de teste em ordem pré-definida e quaisquer ações associadas de pré e pós execução.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição - Identificador - Caso de teste - Requisitos relacionados (Descritos em EP.7) - Condições relacionadas (Descritos em EP.8) - Ordem de execução

3.3.3.4 *Modo de Descrição*

A forma sugerida de descrição das informações relacionadas ao grupo de teste é apresentada no Quadro 18, que ilustra o *Viewpoint* Projeto de Teste. O interesse dele é descrever informações de teste descritas no grupo projeto de teste. Por isso, na linha Interesses do quadro foram listados os subgrupos de teste do Projeto de Teste. Os *stakeholders* na descrição são profissionais das áreas de arquitetura de software, teste de software e desenvolvimento. As notações sugeridas são a linguagem natural e a UTP. Para representar esse *viewpoint*, três diferentes diagramas são apresentados nas Figuras 17 - 19.

Quadro 18 – *Viewpoint* para descrição de Projeto de Teste

Nome	<i>Viewpoint</i> de Projeto de Teste
Visão Geral	Apresentam-se as informações de teste que foram levantados pelo grupo Projeto de Teste. Desse modo, deve-se descrever riscos, defeitos e procedimento de teste.
Interesses	<ul style="list-style-type: none"> • Descrever informações relacionadas aos elementos do grupo Gerenciamento de Defeitos. • Descrever informações relacionadas aos elementos do grupo Análise de Riscos. • Descrever informações relacionadas aos elementos do grupo Estratégia de Teste.
Stakeholders	Desenvolvedores, Testadores, Arquitetos de teste, Arquiteto de Software.
Tipos de Modelo	Linguagem natural e UTP.

Figura 17 – Diagrama que compõe a Visão de Riscos

Fonte: Elaborada pelo autor.

No diagrama de descrição dos riscos, utilizou-se a notação definida por [Ali et al. \(2014\)](#), que estende o UTP e é baseada na norma [ISO-31000 \(2009\)](#). Na descrição dos riscos devem ser descritas a probabilidade, o impacto e a prioridade para cada um dos riscos. Deve-se, também, relacionar os riscos aos elementos da arquitetura que podem ser impactados por eles. O diagrama é composto por três pacotes principais, que são Sistema em Teste, Riscos e Conhecimento de Apoio. Sistema em Teste contém os elementos que devem ser testados no sistema. Riscos contém todos os riscos com probabilidade, impacto e prioridade. O pacote conhecimento de apoio contém três listas das quais os valores foram retirados da norma [ISO-31000 \(2009\)](#) para caracterizar riscos.

Figura 18 – Diagrama que compõe a Visão de Defeito

Fonte: Elaborada pelo autor.

O diagrama é dividido em três pacotes apresentados na Figura 18, que são Sistema em Teste, Defeitos e Conhecimento de Apoio. Sistema em Teste contém os elementos de software que possuem algum defeito relacionado. O estereótipo «SUT» é utilizado para caracterizar o sistema em teste. Defeitos contém o conjunto de defeitos que estão relacionados aos elementos do Sistema em Teste. O estereótipo «Defect» foi criado para caracterizar cada defeito. Conhecimento de Apoio contém informações importantes para gestão dos defeitos.

Na Figura 19, é apresentado o diagrama em que são relacionados os itens de sistema a serem testados, requisitos de teste, casos de teste, *stakeholders* e riscos. O diagrama é composto por quatro pacotes, a saber, Sistema em Teste, Estratégia de Teste, Conhecimento de Apoio

e Artefatos. O pacote Sistema em Teste já foi explicado anteriormente. No pacote Estratégia de Teste, cada contexto de teste é identificado pelo estereótipo «TestContext». Nele, foram adicionados atributos que não existem no UTP original para relacionar requisitos, riscos e *stakeholders*.

Figura 19 – Diagrama Projeto de Teste

Fonte: Elaborada pelo autor.

3.4 Elementos de teste e o TMMi

Como mencionado anteriormente, o TMMi foi utilizado como base para identificar os elementos de teste do RAModel_T. No Quadro 19, são relacionados os níveis do TMMi (linhas) e os elementos de teste (colunas). Pode-se observar que mais de um nível de maturidade foi considerado para definir um mesmo subgrupo de teste. Isso ocorre pois temas semelhantes podem ser abordados em mais de um nível de maturidade, variando restrições e a quantidade de detalhes.

Por exemplo, o subgrupo de teste Ambiente de Teste possui elementos que foram retirados dos níveis de maturidade dois e três. A maioria dos elementos foram retirados dos dois primeiros níveis de teste, devido a eles serem elementos básicos do processo de teste.

Quadro 19 – Relacionamento entre os subgrupos do [RAModel_{TI}](#) e o TMMi

	Gerenciamento de Teste									Infra. de Teste				Projeto de Teste									
	EG.1	EG.2	EG.3	EG.4	EG.5	EG.6	EG.7	EG.8	EG.9	EI.1	EI.2	EI.3	EI.4	EP.1	EP.2	EP.3	EP.4	EP.5	EP.6	EP.7	EP.8	EP.9	EP.10
Nível 2	✓		✓	✓		✓				✓		✓	✓							✓	✓	✓	✓
Nível 3		✓					✓		✓		✓						✓	✓	✓				
Nível 4								✓															
Nível 5					✓									✓	✓	✓							

3.5 Considerações Finais

Neste capítulo, a definição e a descrição do [RAModel_{TI}](#) foram apresentadas. O [RA-Model](#), inicialmente proposto para apoiar o estabelecimento de arquiteturas de referência sem informações de teste de software, foi estendido com elementos relevantes para o planejamento de teste. Ao modelo original, foram adicionados elementos extraídos de dois importantes documentos da área de teste de software, a saber, [TMMi](#) e [ISO/IEC/IEEE \(2013b\)](#). Cada documento foi analisado e, a partir deles, foram retirados 23 elementos de teste. Esses elementos foram agrupados em oito subgrupos de teste e inseridos em três grupos.

Para contribuir na descrição das informações correspondentes aos elementos de teste na documentação arquitetural, três *viewpoints* foram definidos. Neles foram definidos o que deve ser descrito, o tipo de modelo a ser utilizado e os profissionais envolvidos. Os *viewpoints* são Gerenciamento de Teste, Infraestrutura de Teste e Projeto de Teste, um para cada grupo de teste do [RAModel_{TI}](#). A avaliação do [RAModel_{TI}](#) foi realizada por meio de um estudo de viabilidade apresentado no próximo capítulo. No próximo capítulo, um exemplo de aplicação do [RAModel_{TI}](#) em conjunto com o [ProSA-RA_T](#) é apresentado.

EXEMPLO DE APLICAÇÃO DO RAMODEL_T

4.1 Considerações Iniciais

Neste capítulo é apresentado um exemplo de aplicação do [RAModel_{TI}](#) e do [ProSA-RA_T](#) por meio da extensão da arquitetura de referência RefSORS ([OLIVEIRA, 2015](#)). A RefSORS é uma arquitetura de referência para o domínio da robótica projetada com o base nos elementos do RAModel original e nas atividades do ProSA-RA original. Dessa forma, a partir dos elementos presentes na RefSORS serão executados as atividades do [ProSA-RA_T](#) para que os elementos do [RAModel_{TI}](#) possam ser caracterizados na arquitetura de referência. De acordo com [Oliveira \(2015\)](#), um entendimento comum de como estruturar sistemas robóticos orientados a serviço é fundamental para que esses sistemas sejam desenvolvidos com qualidade. De modo semelhante, um entendimento comum de como testar esses sistemas é fundamental para dar suporte aos meios que asseguram a qualidade desejada. Quanto a robótica, serão apresentados os principais tópicos relevantes para testar sistemas robóticos na visão da robótica industrial e suas principais fontes de erros, por fim, serão apresentados requisitos relacionados ao teste em robótica. Quanto a SOA, serão apresentados requisitos de teste relacionados de aplicações SOA. Além disso, deve-se ressaltar que o trabalho apresentado nesse capítulo foi desenvolvido em parceria com [Simão \(2017\)](#), que definiu o [ProSA-RA_T](#).

Para ilustrar os *viewpoints* definidos no Capítulo 3, os serviços de localização dos agentes robóticos serão utilizados. No decorrer do capítulo, os requisitos de sistema referenciados são os requisitos robóticos e de SOA definidos por [Oliveira \(2015\)](#).

Este capítulo está organizado da seguinte forma. Na Seção 4.2 é apresentada a extensão da [RefSORS](#) com informações de teste de software. Na Seção 4.3, é apresentado um exemplo de uma visão de riscos. Uma visão de ambiente é apresentado na Seção 4.4. Na Seção 4.5, é apresentada uma visão de estratégia. Por fim, as considerações iniciais são apresentadas na Seção 4.6.

4.2 Extensão da RefSORS

Para ilustrar uma das partes utilizadas para descrever a RefSORS, na Figura 20 é apresentada uma das visões que compõem a RefSORS. Por meio dela, pode-se ver as principais funcionalidades definidas nessa arquitetura de referência. A RefSORS foi definida para agentes robóticos móveis, orientados a serviço, terrestres e para uso interno (“*indoor*”). Dessa forma, as capacidades referem-se, por exemplo, a localização, navegação, planejamento rotas, percorrer trajetórias, manipular objetos e receber tarefas e as executar. Para definir essa arquitetura, seus autores executaram as quatro fases do [ProSA-RA](#), ou seja, foram levantadas informações relevantes, a arquitetura foi sintetizada e avaliada.

Figura 20 – Visão de Capacidade da RefSORS

Fonte: Adaptada de [Oliveira \(2015\)](#).

Para estender a RefSORS foi preciso revisitar a RA-1 para realizar a Investigação de Fontes de informação relacionadas com a atividade de teste. Após o levantamento das informações sobre teste de software, as atividades da Fase RA-T foram iniciadas. A seguir é apresentada a execução da extensão, seguindo os passos da Fase RA-T indicados pelo [ProSA-RA_T](#).

4.2.1 RA-1: Investigação de Fontes de Informação

A partir do resultado dessa fase, foi possível identificar um conjunto de Documentos Relevantes disponíveis na literatura relacionada ao domínio de robótica e de sistemas SOA. Tais documentos são normas, artigos, relatórios, e planos de teste e uma arquitetura de referência. Nessa fase, os documentos relevantes são levantados, mas nenhuma informação é extraída. A classificação do que é mandatório e o que é opcional será feito durante a extração das informações. Por limitações de tempo, não foi possível levantar fontes de informação por meio de revisões/mapeamentos sistemáticos da literatura, o que seria um dos melhores métodos a serem seguidos. Contudo, as informações levantadas foram consideradas suficientes para ilustrar o uso do modelo e do processo estendidos.

- **Normas:**

- *ISO/TS 15066: Robots and Robotic Devices - Collaborative Robots.*
- *ISO 10218-1: Safety Requirements for Industrial Robots.*
- *ISO 18646-1: Performance criteria and related test methods for service robots.*
- *ISO/IEC 17998: SOA Governance Framework.*
- *ISO/IEC 18384: Reference Architecture for Service Oriented Architecture.*
- *IEEE Std 1872: Std. Ontologies for Robotics and Automation.*
- *IEEE 1873: Std. for Robot Map Data Representation for Navigation.*
- *ISO 9283: Manipulating Industrial Robots – Performance Criteria and Related Test Methods*

- **Artigos:**

- *Robot Accuracy* ([GREENWAY, 2000](#)).
- *Improving Robotic Machining Accuracy Through Experimental Error Investigation and Modular Compensation* ([SCHNEIDER et al., 2014](#)).

- **Arquitetura de Referência:**

- *Reference Architecture Foundation for Service Oriented Architecture* ([OASIS, 2012](#)).

- **Relatórios/Planos de Teste:**

- *Testing Plan for HAM Robot System.*
- *Robot Drive Motor Characterization Test Plan.*
- *Test Plan Modeling and Control of an Industrial Robot.*
- *Robot Platform 1kg - Test Plan.*
- *Communication Model for Cooperative Robotics Simulator - Test Plan.*

4.2.2 RA-T.1: Levantar Conhecimento de Teste

Esta atividade consiste no levantamento de lições aprendidas e elementos de teste adquiridos de outros processos de teste. Esse conhecimento deve abranger os elementos apresentados no [RAModel₇₁](#) e ser adquirido por meio dos *Stakeholders* do projeto e da Biblioteca de Documentos

Relevantes. Para isso, foram definidas duas tarefas para levantar as informações que compõem o Documento de Conhecimento de Teste. A primeira tarefa do ProSA-RA_T consiste em levantar conhecimento tácito e, a segunda, artefatos de teste que possam ser reutilizáveis para o(s) sistema(s) em teste. Durante a condução deste exemplo, não foi possível encontrar indivíduos com experiência em teste de software no domínio de robótica para que o conhecimento tácito fosse obtido. Com relação aos artefatos de teste reutilizáveis, foram considerados os relatórios e planos de teste identificados e apresentados na subseção anterior.

4.2.3 RA-T.2: Definir Contexto Organizacional de Teste

Essa atividade consiste na especificação do Documento Organizacional de Teste. Esse documento deve determinar o escopo em que se inserem os elementos de teste nas atividades seguintes dentro do processo de desenvolvimento. O contexto organizacional de teste deve estar alinhado com o contexto do projeto da Arquitetura de Referência. A segunda atividade do ProSA-RA_T consiste na definição do contexto organizacional de teste. Para isso, os elementos EG.1 a EG.6 do RAModel_{TI} foram considerados. O contexto para a RefSORS é composto pela Política de Teste e Performance de Teste apresentados a seguir:

- **Política de Teste:**

- **Objetivo de teste:** Facilitar o processo de teste dos sistemas robóticos considerando os requisitos de performance para robôs industriais apresentados na ISO 9283:1998.

- **Processo de Teste:** o processo de teste definido na ISO/IEC/IEEE (2013b) deverá ser utilizado como base para realizar o teste dos softwares implementados a partir da Arquitetura de Referência.

- **Normas Regulamentadoras:** Como obrigatória, será considerada a ISO 9283:1998 *Manipulating Industrial Robots – Performance Criteria and Related Test Methods*. As outras normas apresentadas foram consideradas opcionais, uma vez que o contexto da aplicação possa variar.

- **Performance de Teste:**

- **Objetivos do Desempenho:** garantir que o sistema robótico será operacional para as atividades que foi projetado nas condições devidas.

- **Indicadores:** (i) cobertura das condições definidas na ISO 9382; (ii) cobertura de código; (iii) riscos mitigados; (iv) número de defeitos revelados; (v) incerteza de medição.

4.2.4 RA-T.3: Investigar Riscos do Domínio

Esta atividade estabelece uma gestão de riscos do domínio da Arquitetura de Referência. Nela, os principais defeitos e os riscos relacionados a arquitetura de referência são identificados.

Em seguida, os riscos são analisados e os mais críticos são priorizados. Por fim, um plano de mitigação de riscos é definido. Nesta etapa, foram investigados riscos de sistemas do domínio de robótica e os riscos relacionados à SOA. No Quadro 20, são apresentados os riscos relacionados a sistemas robóticos. No Quadro 21 são apresentados os riscos relacionados à SOA. Cada risco recebeu um identificador, possui uma descrição e recebeu uma análise de prioridade, impacto e probabilidade.

Quadro 20 – Exemplos de riscos para área de robótica

ID	Risco	Descrição	Análise
R-R [1]	Erros humanos	Erro humano ocorre na atividade do dia-a-dia e isso não é diferente em relação a uma célula de trabalho robótica.	Raro
			Insignificante
			Moderada
R-R [2]	Erros de controle	Erros no software e no hardware de controles podem levar a riscos dentro de uma célula de trabalho robótica.	Improvável
			Moderado
			Moderada
R-R [3]	Acesso não autorizado	Acesso por um operador não autorizado a uma célula de trabalho robótica protegida.	Provável
			Alto
			Alta
R-R [4]	Falhas mecânicas	Durante os estágios de design e programação, a falha mecânica parcial nem sempre é levada em consideração.	Raro
			Catastrófico
			Moderada
R-R [5]	Fontes ambientais	Fatores externos e interferências de comunicação podem criar um efeito indesejável em uma célula de trabalho robótica.	Provável
			Alto
			Alta
R-R [6]	Sistemas de energia	As fontes de energia que têm comunicação com a célula robótica podem ser interrompidas e levar a ações indesejadas.	Quase Certo
			Alto
			Altíssima
R-R [7]	Instalação inadequada	Sempre que um robô industrial é instalado, é vital para o sucesso do projeto e segurança dos operadores que o sistema está instalado corretamente antes de estar totalmente operacional.	Possível
			Moderado
			Altíssima

Quadro 21 – Exemplos de riscos para SOA

ID	Risco	Descrição	Análise
R-S [1]	Alteração de mensagem	Se um invasor for capaz de modificar o conteúdo das mensagens que são trocadas sem que os participantes tenham conhecimento disso, o invasor comprometeu com sucesso a segurança do sistema.	Raro
			Insignificante
			Moderada
R-S [2]	Interceptação de mensagens	Se um invasor é capaz de interceptar e entender as mensagens trocadas entre os participantes, o atacante poderá fazer uso indevido das informações.	Improvável
			Moderado
			Moderada
R-S [3]	Homem no meio	O atacante tenta convencer cada participante de que é seu correspondente, quando na verdade ele não é.	Provável
			Alto
			Alta
R-S [4]	Ataque de negação de serviço	Um ataque de Negação de Serviço (DoS) é um ataque à disponibilidade e desempenho de um serviço ou conjunto de serviços. Em um ataque DoS, o atacante tenta evitar o uso legítimo do serviço.	Raro
			Catastrófico
			Moderada
R-S [5]	Ataque de repetição	O invasor captura o tráfego de mensagens de uma interação legítima e reproduz parte dele para o alvo. O alvo é persuadido de que uma interação semelhante à anterior está sendo repetida e responde como se fosse uma interação legítima.	Provável
			Alto
			Alta
R-S [6]	Falso repúdio	Em falso repúdio, uma interação normal é completada, mas um ator mais tarde tenta negar que a interação ocorreu.	Quase Certo
			Alto
			Altíssima

A partir dos riscos arquiteturais, algumas medidas para mitigação e ações preventivas foram levantadas. No Quadro 22 são apresentadas exemplos de medidas para mitigação de riscos.

Quadro 22 – Exemplo de mitigações para riscos identificados

ID	Mitigação	Ações	Risco (s)
M-S [1]	Proteção de Integridade	Criptografia da informação e estabelecer políticas de segurança.	R-S [1]
M-S [2]	Aplicação de Privacidade	Adicionar assinatura digital do código <i>hash</i> de uma mensagem.	R-S [2] e R-S [3]
M-S [4]	Engajamento graduado	Sistemas detectores de intrusão (por exemplo, roteadores, <i>firewalls</i>), o provedor não deve fornecer serviços além dos implicados pelo estado atual das interações.	R-S [4]
M-S [5]	Proteção de Repetição de Mensagens	Adicionar informações que podem ser usadas para detectar mensagens repetidas.	R-S [5]
M-S [6]	Auditoria e registro	Assinatura digital, registro de interação e auditoria de registros	R-S [6]

4.2.5 RA-T.4: Estabelecer Estratégia de Teste

Nessa atividade, uma estratégia de teste a partir do Documento de Gestão de Riscos é definida, estabelecido na atividade anterior. O Documento de Estratégia de Teste deve fornecer diretrizes para a execução dos processos de teste. O documento deve abordar os itens a serem testados, as condições em que o teste deve ser executado, as técnicas que serão utilizadas e os procedimentos que deverão ser executados.

4.2.5.1 Requisitos de Teste

Os Requisitos de Teste a respeito de Robótica foram obtidos com vistas a representar os principais informações de teste de software que a arquitetura de referência deve considerar. Além disso, são apresentados os requisitos de teste para SOA que foram derivados da arquitetura de referência OASIS.

RTR-1 O robô deve estar montado de acordo com as recomendações do fabricante e com todas as partes funcionais.

RTR-2 Todas as operações de nivelamento, procedimentos de alinhamento e teste funcionais devem estar satisfatoriamente testados.

RTR-3 Antes do início da atividade de teste, o robô deve passar por uma operação de aquecimento adequada.

- RTR-4** A temperatura do ambiente de teste deve ser de 20 °C, sendo que outras temperaturas ambientes devem ser indicadas e explicadas no relatório de teste.
- RTR-5** Para testar características de trajetória, ultrapassagem e medições de estabilização, as características dinâmicas do equipamento de aquisição de dados (por exemplo, taxa de amostragem) devem ser suficientemente elevadas para garantir que seja obtida uma representação adequada das características a serem medidas.
- RTR-6** Os instrumentos utilizados para os testes devem ser calibrados e a incerteza de medição (levando em consideração, erros de instrumentação, erros relacionados ao método adotado, erros de cálculo) deve ser estimada e indicada no relatório de teste.
- RTR-7** A incerteza total na medição não deve exceder 25%.
- RTR-8** Todos os testes devem ser executados com uma carga de teste igual a 100% das condições carregadas.
- RTR-9** Podem ser feitos testes adicionais com carga reduzida em 10%.
- RTR-10** Todas as características de posicionamento devem ser testadas à velocidade máxima nas posições especificadas.
- RTR-11** Podem ser realizados teste de posicionamento adicionais com 50% e 10% da velocidade máxima.
- RTR-12** Todas as juntas do robô devem ser exercitadas em movimentos considerando todas posturas que o robô pode assumir.
- RTR-13** Deve ser testado o tempo de posicionamento mínimo, considerando o tempo que o robô gasta para estabilizar-se na posição.
- RTR-14** Deve ser considerada a conformidade estática, considerando a quantidade máxima de deslocamento por unidade de carga aplicada. As forças devem ser aplicadas em três direções diferentes.
- RTR-15** Com o passar do tempo, testes de desempenho devem ser realizados para assegurar que o funcionamento do robô continua adequado para realizar determinada tarefa.
- RTR-16** Recomenda-se o teste de tempo de estabilização seja realizado antes do teste de repetitividade de postura.
- RTR-17** As características de postura devem ser testadas postura-a-postura ou em uma trajetória continua.
- RTR-18** Podem ser testados simultaneamente exatidão/repetitividade de trajeto e características de velocidade.

RTR-19 Podem ser testadas simultaneamente o desvio de curvas e erros de arredondamento.

RTR-20 Erros de precisão e repetitividade de postura podem ser causadas por: definição de controle interno, erros de transformação de coordenadas, diferenças entre a estrutura de articulação do robô e a implementada no controle do sistema, defeitos mecânicos ou efeitos externos.

Os Requisitos de Teste sobre SOA foram obtidos com objetivo de definir as principais informações de teste de software que a arquitetura SOA deve considerar. Os RTS são:

RTS-1 Conformidade entre a arquitetura instanciada e a arquitetura de referência deve ser testada, isto é, verificar se ocorreu uma aderência dos princípios descritos na arquitetura de referência na arquitetura concreta.

RTS-2 Serviços devem ser testáveis no ambiente e nas condições que podem ser encontrados operacionalmente.

RTS-3 Protocolos de teste devem reconhecer e acomodar mudanças nas atividades dentro do ecossistema¹ SOA.

RTS-4 Um conjunto de serviços padrão de monitoramento deve ser definido, desenvolvido e mantido.

RTS-5 Serviços devem fornecer interfaces que suportem o acesso em modo de teste.

RTS-6 Recursos de teste devem ser descritos e suas condições devem ser catalogadas de forma a permitir seu acesso e descoberta.

RTS-7 Deve-se considerar características de segurança distribuída e melhores práticas para garantir recursos seguros.

RTS-8 Teste de unidade, utilizando *SANDBOX*, que inclui teste de funcionalidades, interfaces, e se o serviço tem trabalhado satisfatoriamente com recursos internos e externos.

RTS-9 Após testes unitários terem demonstrado um nível adequado de confiança, o teste deve ser realizado em um ecossistema SOA que se assemelha ao que o serviço será utilizado.

RTS-10 Serviços podem apresentar interfaces de teste.

RTS-11 Detalhes de teste legado e casos de teste já executados devem estar disponíveis para que possam ser novamente executados.

¹ Uma visão de ecossistema SOA envolve as capacidades desenvolvidas para atender necessidades, ele é um espaço em que pessoas, processos e máquinas interagem para fornecer essas capacidades como serviços (OASIS, 2012).

- RTS-12** Deve existir um meio para testar novas políticas ou contratos que surgirem depois que o serviço já está sendo executado.
- RTS-13** O teste deve garantir a funcionalidade dos serviços.
- RTS-14** O teste deve garantir as políticas e restrições associadas aos serviços.
- RTS-15** O teste deve garantir que as interfaces dos serviços estejam adequadas para o uso.
- RTS-16** O teste deve garantir a disponibilidade de um serviços (por exemplo, por meio de monitoramento).
- RTS-17** O teste deve garantir que as métricas indicadas na descrição dos serviços estão precisas (por exemplo, tempo médio de resposta), conforme acordado em contrato.
- RTS-18** Para gerar as condições de teste, elas devem ser concebidas para abranger: i) condições nominais; ii) condições limites esperadas; iii) condições limites em que o funcionamento do serviço ficou comprometido; iv) condições aleatórias; v) condições de erros.
- RTS-19** Pessoas que realizam os testes devem estar familiarizadas com as funcionalidades e com o ecossistema SOA que o serviço deverá operar.
- RTS-20** Em casos que há a necessidade, apenas algumas pessoas ou organizações têm autoridade para realizar atividades de teste em um serviço.
- RTS-21** Informações de teste e certificações realizadas no serviços devem sempre estar disponíveis (incluindo teste realizado por terceiros).

4.2.5.2 *Técnicas de Teste*

A técnica de teste a ser utilizada para esta arquitetura de referência é a técnica funcional.

4.2.6 ***RA-T.5: Estabelecer Infraestrutura de Teste***

Nessa atividade, a infraestrutura de teste necessária para suportar a estratégia de teste é definida. A infraestrutura deve prover um ambiente de teste adequado para condução e gerenciamento dos processos de teste. Para isso foram estabelecidas duas tarefas. Essas tarefas têm por objetivo investigar como pode ser realizada a automatização de teste e os requisitos de ambiente de teste.

4.2.6.1 *Ferramentas de Teste*

Nessa tarefa, o analista de teste analisa os procedimentos de teste para identificar os recursos necessários para condução da atividade de teste. Com base nesta análise, deve-se procurar por relatórios, estudos e ferramentas mais adequadas ao domínio da arquitetura de

referência. Para isso, foram listadas ferramentas que contribuem para gestão da atividade de teste (relevantes para distribuição de tarefas a serem realizadas para *stakeholders*), ferramentas para execução da atividade de teste (que podem ser utilizadas para descrever casos/procedimentos de teste e viabilizam a execução da atividade) e um simulador. Ferramentas para a gestão da atividade de teste:

- JIRA²
- *Robot Framework*³
- *Active Colab*⁴
- *Test Rail*⁵

Ferramentas para execução e descrição da atividade de teste:

- GTEST⁶
- CUnit⁷
- JUnit⁸

Ferramentas para simulação de agente robóticos, tem-se:

- Gazebo⁹.

4.2.6.2 Ambiente de Teste

Nessa tarefa, o analista de teste deve definir os itens necessários (tais como, software, hardware e simuladores) para a execução dos processos de teste. A Automação de Teste e o Documento de Estratégia de Teste devem ser considerados. A partir da arquitetura de referência RefSORS, a aplicação RobAFIS foi instanciada. Em resumo, na aplicação um sistema robótico foi simulado. O ambiente para executar a aplicação robótica é:

- Sistema Operacional Ubuntu¹⁰ 14.04;
- Sistema Operacional ROS¹¹ (*Robotic Operating System*).

4.3 Visão de Riscos

Na Figura 21, o diagrama que compõe a visão de riscos é apresentada. Esse diagrama é uma extensão da UTP para riscos. Na visão de riscos, um subconjunto dos serviços relacionados

² <<https://br.atlassian.com/software/jira>>

³ <<http://robotframework.org/>>

⁴ <<https://activecollab.com/>>

⁵ <<http://www.gurock.com/testrail/>>

⁶ <<http://wiki.ros.org/gtest>>

⁷ <<http://cunit.sourceforge.net/>>

⁸ <<http://junit.org/junit4/>>

⁹ <<http://gazebo.org/>>

¹⁰ <<https://www.ubuntu.com/>>

¹¹ <www.ros.org>

a localização, navegação, mapeamento, foi utilizado como exemplo. Esses serviços foram adicionados no pacote “Sistema em Teste” com o estereótipo «SUT».

Figura 21 – Diagrama de Visão de Riscos

Fonte: Elaborada pelo autor.

Os serviços seleccionados nessa visão foram relacionados a cinco riscos levantados na subseção anterior. As setas tracejadas indicam quando um risco está relacionado a um serviço, sistema ou a outro risco. Por exemplo, O risco “Acesso não Autorizado” é relacionado ao risco “Falhas Mecânicas” e aos serviços “Localização”, “Planejamento de Trajetória” e “Navegação”.

4.4 Visão de Infraestrutura

A visão de ambiente foi definida utilizando os itens de infraestrutura e de alguns serviços relacionados ao planejamento. Foi utilizado o estereótipo «TestContext» adaptado para definir o ambiente necessário de teste seja para o sistema ou a serviços específicos. Essa visão relaciona ferramentas, o sistema em teste, riscos, envolvidos e requisitos de teste e sistema.

Figura 22 – Diagrama de Visão de Infraestrutura

Fonte: Elaborada pelo autor.

4.5 Visão de Estratégia

Na visão de estratégia, foi apresentado como relacionar os itens de teste a casos de teste, riscos relacionados, requisitos de teste e requisitos do sistema. Casos de teste e outros artefatos podem ser definidos durante o estabelecimento da arquitetura de referência ou obtidos nos artefatos reutilizáveis da fase RA-T.1.

Figura 23 – Diagrama de Estratégia de Teste

Fonte: Elaborada pelo autor.

4.6 Considerações Finais

Neste capítulo, exemplos que ilustram a aplicação do RAModel_T e ProSA-RA_T foram apresentados. Os exemplos indicaram de um modo inicial a usabilidade do processo e do modelo bem como ilustram visões de teste para arquiteturas de referência. No trabalho, a RefSORS foi estendida com elementos de teste de software com vistas a estruturar como atividades de teste poderiam ser incorporadas RefSORS e a satisfazer requisitos e as restrições do domínio. Para representar as informações foram utilizadas as visões conceitual, de capacidade e de implantação. À luz de informações de teste coletadas de uma norma para robótica industrial, estudos sobre erros comuns em robótica e em teste para sistemas baseados em SOA, as três visões arquiteturais foram exemplificadas. Dessa forma, foram exploradas novas perspectivas de representação de informações em arquiteturas de referência. No próximo capítulo, apresenta-se a avaliação do

[RAModel_{7I}](#) por meio de um estudo de viabilidade.

AVALIAÇÃO DO RAMODEL_T

5.1 Considerações Iniciais

Neste capítulo, é apresentada a avaliação do RAModel_{TI} por meio de um estudo de viabilidade. De acordo com [Travassos, Gurov e Amaral \(2002\)](#), nenhum método, técnica ou linguagem deveriam ser sugeridos sem antes passar por um processo de experimentação e validação. Para isso, foi adotada a metodologia definida por [Shull, Carver e Travassos \(2001\)](#) e estendida por [Mafra, Barcelos e Travassos \(2006\)](#). Segundo [Mafra, Barcelos e Travassos \(2006\)](#), a proposição de novas tecnologias é dividida em Definição Inicial da Tecnologia e Refinamento da Tecnologia. Neste trabalho, a Definição Inicial da Tecnologia foi realizada em um mapeamento sistemático da literatura. Ele foi realizado para identificar evidências sobre a relação entre arquitetura e teste de software ([SOUZA *et al.*, 2017b](#); [SOUZA *et al.*, 2017a](#)). Depois, para o Refinamento da Tecnologia, foi realizado um estudo de viabilidade com especialistas em arquitetura de software e teste de software.

Na introdução deste trabalho, foi exposto um breve histórico acerca das áreas de arquitetura de software e teste de software, destacando desde quando data a literatura sobre elas. Por meio de um mapeamento sistemático ([SOUZA *et al.*, 2017b](#)), identificaram-se estudos que relacionam essas áreas, os quais têm sido publicados desde 1996, abordando principalmente o teste baseado em arquitetura. Para o desenvolvimento de sistemas de software com qualidade, têm-se modelos para melhoria de processos como o CMMi e o TMMi. No CMMi, desde o nível de maturidade três, o estabelecimento de arquiteturas de software é requerido. No TMMi, o nível dois requer o planejamento de teste. No nível três, o planejamento de teste é requerido desde os níveis iniciais do desenvolvimento de software.

Este capítulo está organizado da seguinte forma. Na Seção [5.2](#), é fornecida uma introdução a estudos de viabilidade. Na Seção [5.3](#), é apresentado o planejamento do estudo de viabilidade. Na Seção [5.4](#) é apresentada a execução do estudo. A análise e discussão dos resulta-

dos são apresentados na Seção 5.5. Na Seção 5.6, é realizada a discussão dos resultados obtidos. As ameaças a validade são apresentadas na Seção 5.7. As considerações finais são apresentadas na Seção 5.8.

5.2 Estudo de Viabilidade

Um *survey* é um sistema para coleta de informações de/ou sobre pessoas para descrever, comparar ou explicar seus conhecimentos, atitudes e comportamento (WOHLIN *et al.*, 2012). *Surveys* podem ser realizados quando seus objetos de estudos já foram implantados ou antes mesmo de serem introduzidos (WOHLIN *et al.*, 2012). Para isso, o principal meio de coleta de dados é o questionário, a partir da análise desses, conclusões e explicações sobre o objeto em estudo são obtidas (TRAVASSOS; GUROV; AMARAL, 2002; WOHLIN *et al.*, 2012). Além disso, há três tipos de *surveys*, que, em suma, são (WOHLIN *et al.*, 2012): i) descritivo - são conduzidos para confirmar afirmações sobre uma população e determinar a distribuição de atributos ou características; ii) explanatório - tem o objetivo de realizar declarações explicativas sobre uma população, por exemplo, porque alunos escolhem uma ou outra técnica para estudar determinado assunto; iii) exploratório - é utilizado como pré-estudo para uma investigação mais aprofundada para assegurar que temas importantes não serão esquecidos em um estudo mais completo. Para execução desta avaliação, serão combinados o estudo de viabilidade projetado no formato de estudo empírico *survey* explanatório. O planejamento, a execução e a análise de dados da avaliação são descritos nas seções subsequentes.

5.3 Planejamento do Estudo de Viabilidade

O sujeitos que participaram do estudo de viabilidade são especialistas em arquitetura de software e/ou em teste de software. Eles foram questionados sobre os subgrupos de teste e os elementos de teste. O planejamento desse estudo de viabilidade é composto por questões de pesquisa, objetivo, seleção de sujeitos e instrumentação, que são apresentados a seguir.

5.3.1 Questões de Pesquisa

As questões de pesquisa foram definidas para verificar se: i) o conjunto de elementos de cada subgrupo do $RAModel_{TI}$ está completo; ii) existem elementos de teste no $RAModel_{TI}$ que não são necessários; iii) a descrição de cada um dos elementos do $RAModel_{TI}$ está correta; e iv) o conjunto de atributos de cada elemento do $RAModel_{TI}$ está bem definido. Dessa forma, para cada subgrupo do $RAModel_{TI}$ foi questionado:

- O conjunto de elementos de teste do subgrupo está completo?
 - Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

- Existem elementos de teste do subgrupo que não são necessários?
 - Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.
- A descrição de cada um dos elementos de teste do subgrupo está correta?
 - Se não, por favor, sugira definições que poderiam ser incluídas ou excluídas.
- O conjunto de atributos de cada elemento de teste do subgrupo está bem definido?
 - Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

5.3.2 Objetivo

O objetivo desse estudo de viabilidade foi definido com base no modelo GQM ([BASILI et al., 2007](#)):

Analisar o $RAModel_{TI}$

Com propósito de avaliá-lo

Com respeito a problemas de ambiguidade, consistência e completude nos elementos inseridos no $RAModel_{TI}$

Do ponto de vista de especialistas

No contexto de professores especialistas e pós-graduandos em engenharia de software, arquitetura de software e teste de software.

5.3.3 Seleção de Sujeitos

Para realizar esse estudo, foram convidados cinco especialistas em arquitetura de software e/ou teste de software. Os sujeitos do estudo são doutorandos do Laboratório de Engenharia de Software (LabES) do Instituto de Ciências Matemáticas e de Computação (ICMC) e professores do ICMC/USP e do Departamento de Computação/UFSCAR. Deve-se ressaltar que o ICMC possui importantes grupos de pesquisa em arquitetura de referência e em teste de software. A caracterização da experiência dos sujeitos seguiu a classificação: i) experiência em teste de software; ii) experiência em arquitetura de software; iii) experiência em arquitetura de referência; iv) experiência com o $RAModel$. Todos os sujeitos responderam os formulários de avaliação, consentimento e caracterização.

5.3.4 Instrumentação

Apresenta-se, a seguir, uma lista que resume os materiais utilizados para conduzir o estudo de viabilidade.

- **Formulário de Consentimento:** nesse formulário os sujeitos responderam um questionário concordando com os objetivos da pesquisa e declarando o seu interesse em participar de forma voluntária do estudo. O formulário de consentimento é apresentado no Apêndice [B](#);

- **Formulário de Caracterização de Perfil:** nesse formulário os sujeitos declararam o nível de familiaridade/conhecimento sobre teste de software e arquitetura de referência, e os tópicos de pesquisa abordados na execução deste estudo, como o TMMi, planejamento de teste e arquiteturas de referência. O formulário de caracterização é apresentado no Apêndice C;
- **Síntese do RAModel_{TI}:** o RAModel_{TI} foi apresentado aos sujeitos do estudo de viabilidade na síntese descrita no Apêndice D; e
- **Formulário de Avaliação:** os sujeitos responderam ao questionário para avaliar o RAModel_{TI}. Esse formulário contém perguntas para responder as questões de pesquisa apresentadas na Subseção 5.3.1. O formulário é apresentado no Apêndice E.

5.4 Execução do Estudo de Viabilidade

Após o planejamento, a fase de avaliação do estudo de viabilidade foi realizada. A execução aconteceu em julho de 2017 por meio de um formulário online. Os sujeitos foram convidados por um e-mail, recebendo um texto de convite e o *link* para o formulário¹. Nesse formulário, os sujeitos declararam concordar em participar do estudo, seu nível de escolaridade, sua experiência nas áreas correlatas ao estudo de viabilidade e a opinião sobre o RAModel_{TI}.

Participaram do estudo cinco sujeitos, sendo um professor, três alunos de pós-graduação e um doutor recém titulado. A maioria (três) dos participantes declaram ser do sexo feminino. A escolaridade de cada um deles é apresentada na Figura 24. Para participar do estudo de viabilidade, os sujeitos declararam participar de forma voluntária. A eles, também, foi apresentada a possibilidade de desistência a qualquer momento.

Figura 24 – Nível de escolaridade dos sujeitos

Fonte: Elaborada pelo autor.

¹ <<https://goo.gl/forms/MHxfC6JSJrjoJUsN2>>

A caracterização de perfil foi dividida em três partes. A primeira relacionada a teste de software. A segunda relacionada a arquitetura de software. A terceira relacionada a arquitetura de referência e o RAModel. A caracterização em relação ao teste de software é apresentada na Figura 25 e em mais detalhes: i) um sujeito se declarou especialista (nível 5), dois se declararam com nível intermediário (nível 3) e dois iniciantes (nível 1); ii) dois sujeitos declararam ter mais de dez anos de experiência na academia, um declarou dois anos e dois declararam um ano; iii) apenas um sujeito declarou possuir experiência na indústria (um ano); iv) três sujeitos conhecem o TMMi e dois conhecem a norma IEEE/IEC/ISO 29119; v) apenas um sujeito não realizou uma disciplina de teste de software.

Figura 25 – Perfil dos usuários em teste de software

Fonte: Elaborada pelo autor.

A caracterização em relação à arquitetura de software é apresentada na Figura 26 e em mais detalhes: i) um sujeito se declarou especialista (nível 5) e quatro se declararam em nível intermediário (nível 3); ii) um sujeito possui um ano de experiência na academia, um possui dois anos, um possui três anos, um possui quatro anos e um possui oito anos; iii) apenas um sujeito possui experiência na indústria (três anos); iv) três sujeitos fizeram uma disciplina sobre arquitetura de software; v) quatro participaram de um projeto de arquitetura de software.

Figura 26 – Perfil dos usuários em arquitetura de software

Fonte: Elaborada pelo autor.

A caracterização em relação à arquitetura de referência é apresentada na Figura 27 e em mais detalhes: i) dois sujeitos se declararam iniciantes (nível 1), dois declararam nível intermediário (nível 3), um se declarou especialista (nível 5); ii) dois sujeitos não possuem experiência na academia, um possui dois anos, um possui quatro anos e um possui cinco anos; iii) apenas um possui um ano de experiência na indústria; iv) nenhum dos sujeitos cursou disciplina sobre arquitetura de referência; v) três participaram de projetos de arquitetura de referência; vi) dois sujeitos não conhecem o RAModel e três conhecem; vii) dois sujeitos já utilizaram o RAModel; viii) um sujeito participou de um projeto com o RAModel e um sujeito participou de dois projetos com o modelo. A Figura 28 contém os níveis de experiência reportadas pelos participantes do estudo acerca das áreas.

Figura 27 – Perfil dos usuários em arquitetura de referência

Fonte: Elaborada pelo autor.

Figura 28 – Experiência dos sujeitos nas áreas do estudo de viabilidade

Fonte: Elaborada pelo autor.

5.5 Análise dos Resultados

Para cada subgrupo, foram respondidas quatro questões objetivas. Para cada questão, existia uma subquestão em que era possível justificar a resposta dada. Dessa forma, os resultados obtidos das questões objetivas são apresentados nas Tabelas 1 - 3. As justificativas das respostas objetivas são discutidas no decorrer do texto.

Na Tabela 1, são apresentados os resultados sobre o grupo Gerenciamento de Teste. O subgrupo Política de Teste foi considerado completo por todos os todos os participantes do estudo. A maioria (80%) considerou que todos os elementos desse subgrupo são necessários, apenas um alegou que “Objetivos de Teste” não seria necessário. Todos os sujeitos afirmaram que os elementos estão bem descritos. Todos os sujeitos apresentaram que o conjunto de atributos estão completos. No subgrupo Performance de Teste, a maioria (80%) dos sujeitos consideraram o subgrupo completo. Eles, porém, apresentaram que “Desempenho” seria um termo mais adequado para o subgrupo. Todos sujeitos consideraram os elementos necessários. Todos consideraram os elementos bem definidos, porém, um sujeito indicou a necessidade de serem incluídos exemplos em todos os elementos de teste. O subgrupo Melhoria de Teste foi considerado completo por 80% dos sujeitos, porém um sujeito considerou que esse subgrupo já poderia ter sugestões de melhoria. Todos os elementos do subgrupo Melhoria de Teste foram considerados necessários e a descrição dos elementos foi considerada completa. A maioria (80%) considerou que os atributos estão bem definidos, no entanto, um apresentou que deve ser explicitado mais claramente as relações entre os elementos.

Tabela 1 – Resultado da avaliação para o grupo Gerenciamento de Teste

	Política de Teste		Performance de Teste		Melhoria de Teste	
	Sim	Não	Sim	Não	Sim	Não
Subgrupo completo	100%	0	80%	20%	80%	20%
Elementos não necessários	20%	80%	0	100%	0	100%
Descrição dos elementos	100%	0	100%	0	100%	0
Conjunto de atributos	100%	0	100%	0	80%	20%

Fonte: Elaborada pelo autor.

Na Tabela 2, o resultado da avaliação para o grupo Infraestrutura de Teste é apresentado. No subgrupo de ambiente de teste, os sujeitos declararam que todos os elementos são necessários e a descrição está correta. Um sujeito considerou que o conjunto de atributos não está completo, mas não apresentou justificativa. No subgrupo automatização de teste, os sujeitos consideraram o subgrupo completo e todos os elementos apresentados no modelo são necessários. Quanto aos atributos da automatização, dois sujeitos sugeriram modificações no modelo. A primeira é a inserção de informações sobre a viabilidade da integração de ferramentas e as formas de integração das ferramentas. A segunda se refere a padronização de termos no modelo.

Tabela 2 – Resultado da avaliação para o grupo Infraestrutura de Teste

	Ambiente de Teste		Automatização de Teste	
	Sim	Não	Sim	Não
Subgrupo completo	100%	0	100%	0
Elementos não necessários	0	100%	0	100%
Descrição dos elementos	100%	0	100%	0
Conjunto de atributos	80%	20%	60%	40%

Fonte: Elaborada pelo autor.

O resultado da avaliação dos elementos do grupo Estratégia de Teste é apresentado na Tabela 3. Na avaliação do subgrupo Gerenciamento de Defeitos, um dos sujeitos apontou que a criticalidade dos defeitos pode ser incluída no subgrupo. Além disso, testes relacionados aos defeitos podem ser melhor explorados para fim de rastreabilidade. A maioria (80%) dos sujeitos indicaram que todos os elementos de teste são necessários. A maioria (80%) dos sujeitos indicou que a descrição está correta, um sujeito indicou que são necessárias melhores explicações sobre os itens da descrição. Na avaliação da Análise de Riscos, foi indicado por 80% dos sujeitos que o subgrupo está completo. Todos os elementos foram avaliados como necessários. A descrição dos elementos e o conjunto de atributos receberam 60% de avaliação positiva. Os sujeitos, nessas questões, requisitaram melhores explicações nas descrições do elemento e dos atributos. Na estratégia de teste, o subgrupo foi considerado completo por 80% dos participantes e todos os elementos foram considerados necessários. A descrição dos elementos foi considerada completa por todos os sujeitos e 80% deles avaliaram o conjunto de atributos como bem definido, o sujeito que avaliou como incompleto não justificou a resposta.

Tabela 3 – Resultado da avaliação para o grupo Estratégia de Teste

	Gerenc. de Defeitos		Análise de Riscos		Estratégia de Teste	
	Sim	Não	Sim	Não	Sim	Não
Subgrupo completo	80%	20%	80%	20%	80%	20%
Elementos não necessários	20%	80%	0	100%	0	100%
Descrição dos elementos	80%	20%	40%	60%	100%	0
Conjunto de atributos	60%	40%	40%	60%	80%	20%

Fonte: Elaborada pelo autor.

5.6 Discussão dos Resultados

A partir do resultado obtido no estudo de viabilidade, foi possível identificar como pesquisadores da área de teste de software e arquitetura de software avaliam o RAModel_{TI}. Por meio do estudo, foram identificadas melhorias no modelo em relação a completude dos subgrupos de teste e de uma descrição mais precisa dos elementos de teste. Em suma, o modelo

foi avaliado positivamente pelos sujeitos do estudo de viabilidade. O menor índice de avaliação positiva (60%) foi referente ao subgrupo Análise de Riscos, em que foi indicada a necessidade de uma melhor explicação com exemplos nos elementos e nos atributos. Um resumo dos comentários dos sujeitos é apresentado no Quadro 23

Quadro 23 – Resumo dos comentários dos sujeitos

Sujeito	Alterar Elemento	Alterar Atributo	Incluir Relacionamento	Incluir Exemplos	Comentário
Sujeito 1	✓	✓	✓	✓	"Eu substituiria Performance por Desempenho", "Testes associados para fins de rastreabilidade", "Acredito que um ponto importante na automatização de teste é a viabilidade de integração de ferramentas e quais são essas formas de integração. Esse pode ser um atributo a ser incluído dentre os que foram apresentados."
Sujeito 2	✓		✓	✓	"Exemplo de riscos", "Não estou certo quanto aos objetivos do teste. O objetivo não é sempre o mesmo? Encontrar defeitos?", "incluir exemplos complementando os itens descrição e atributo em todos os EGs.", "Da mesma forma que atributos de EG8 estão relacionados à atributos de EG7, talvez o atributo "Conjunto de melhorias" em EG9 poderia ser uma relação com o atributo "Melhorias no processo de teste" de EG8.", "Poderia deixar claro a relação do atributo "Risco" de EP6 com o atributo "Riscos" de EP4."
Sujeito 3				✓	"Não sei, eu não entendi direito como funcionária aquele patterns."
Sujeito 4					
Sujeito 5			✓	✓	"EG9, melhor trocar envolvidos por stakeholders", "Na EG1, sugiro inserir o termo teste de software, ao invés de deixar somente teste que pode deixar aberto a interpretações, uma vez que usa o termo "teste dentro da organização"."

Alguns termos foram ajustados no RAModel₇₁ de acordo com a sugestão dos sujeitos do estudo de viabilidade, ou seja, o modelo apresentado no Capítulo 3 foi atualizado. Os termos "Performance de Teste" não foi substituído por "Desempenho de Teste", pois acredita-se que o termo performance é adequado para o que se pretende tratar. Apenas o termo *stakeholder* foi utilizado para indicar os envolvidos no processo de desenvolvimento de software, visando a promover uma padronização no texto. A integração de ferramentas para automatização de teste foi adicionado ao subgrupo Automatização de Teste bem como a forma em que a integração

pode ser realizada. Aos elementos do grupo Estratégia de Teste, foram adicionados atributos para relacionar testes realizados e artefatos de teste para aprimorar a rastreabilidade nas informações de teste. Um atributo referente a criticalidade de defeitos foi adicionado ao elemento Conjunto de Defeitos.

A maioria dos sujeitos consideraram todos os elementos inseridos no RAModel_{TI} necessários, devendo levar em consideração que a remoção de nenhum elemento foi indicada. O elemento “objetivo de teste” foi considerado não necessário por um sujeito, com a alegação de que o objetivo seria sempre o mesmo: encontrar defeitos. Porém, nesse caso, possíveis objetivos seriam “Verificar a conformidade com os padrões externos”, “Impedir que os defeitos ocorram em operação”, “Validar produtos para eles estejam ‘adequado para uso’”. Então, o elemento “objetivo de teste” continuou a ser considerado necessário e não foi removido do modelo. Quanto ao pedido de exemplo nos elementos e atributos, acredita-se que parte sejam sanadas ao apresentar o Capítulo 4.

5.7 Ameaças à Validade

Segundo Wohlin *et al.* (2012), a validade dos resultados obtidos em um experimento é uma questão fundamental. Nesse sentido, segundo esses autores, deve-se considerar as ameaças à validade desde o planejamento para que os resultados tenham uma qualidade adequada. Em relação as ameaças à validade para este estudo de viabilidade, têm-se:

- **Validade de conclusão:** refere-se à capacidade de chegar a uma conclusão correta a respeito dos relacionamentos entre o tratamento e o resultado obtido no estudo. Neste estudo, os formulários poderiam ser uma ameaça, tendo em vista que coletar informações incorretas ou não fazer perguntas claras aos sujeitos podem levar a conclusões equivocadas. Para tentar mitigar essa ameaça, os formulários foram baseados em outros já utilizados e foram cuidadosamente revisados;
- **Validade interna:** refere-se às influências que podem afetar a causalidade entre o tratamento e o resultado do estudo. Deve-se, portanto, ter atenção à seleção dos participantes, à divisão nas classes e no modo de aplicação dos tratamentos. Neste estudo, os participantes foram selecionados com base nas competências e em suas especialidades. Além disso, o questionário foi definido com poucas questões a fim de que os sujeitos não ficassem desanimados ou com vontade de terminar muito rapidamente.
- **Validade de construção:** refere-se ao relacionamento entre a teoria e a observação, ou seja, a influência do tratamento no resultado. Poderiam ter sido escolhidos métodos de avaliação que favorecessem o RAModel_{TI} e o levassem a ser facilmente aceito. Porém, estudo de viabilidade que faz parte de uma metodologia consolidada, que possui outros meios de avaliação que serão conduzidos em trabalhos futuros; e

- **Validade externa:** refere-se à habilidade de generalizar os resultados obtidos no estudo. Essa ameaça não se aplica a este estudo, pois o objetivo do estudo de viabilidade refere-se a compreender se o objeto em estudo cumpre seus objetivos. Portanto, não se pretende generalizar os resultados obtidos.

5.8 Considerações Finais

Neste capítulo, um estudo de viabilidade foi apresentado como meio para avaliação do RAModel_{TI}. O estudo foi realizado por meio de um formulário online em que os sujeitos do estudo puderam avaliar cada elemento do modelo. No resultado do estudo de viabilidade, foram identificadas melhorias que foram analisadas e, nos casos pertinentes, foram aplicadas aos elementos de teste. Também foram identificadas melhorias a serem realizadas no questionário de avaliação, em que perguntas devem ser revisitadas para serem mais claras e com mais opções. Como resultado, pode-se dizer que o RAModel_{TI} recebeu uma avaliação positiva e apresentou-se viável para elencar elementos essenciais para que arquiteturas de referência possuam informações de teste de software e, assim, deem suporte ao planejamento de teste. No próximo capítulo são apresentadas as conclusões deste trabalho, as contribuições e as propostas de trabalhos futuros.

CONCLUSÕES E TRABALHOS FUTUROS

6.1 Visão Geral da Pesquisa

Arquitetura de software e teste de software são reconhecidos como duas importantes áreas dentro da Engenharia de Software. Com base nelas, sistemas de software podem ser desenvolvidos com qualidade. A partir de um mapeamento sistemático, identificou-se que poucos trabalhos relacionavam essas duas áreas em conjunto. No escopo dos trabalhos que relacionam as áreas, destaca-se o teste baseado em arquitetura. Essa abordagem tem como base informações arquiteturais para, a partir delas, derivar informações relevantes para a atividade de teste. Não foram identificados estudos que inserissem informações de teste nas arquiteturas. As arquiteturas de referência, nesse sentido, podem contribuir como uma ferramenta para o armazenamento e reúso de informações de teste de software, principalmente para o planejamento de teste. Como meio para estabelecer arquiteturas de referência com esse tipo de informação, um modelo e um processo de engenharia de arquiteturas de referência foram revisitados à luz de documentos de teste consolidados.

Nesse intento, o [RAModel_T](#) foi estendido com elementos de teste de software no presente projeto de mestrado. O procedimento adotado para a extensão do modelo foi semelhante a utilizada por [Nakagawa, Oquendo e Becker \(2012\)](#) na definição do RAModel original. A partir desse procedimento, foram identificados 23 elementos de teste que foram divididos em três grupos de teste. Os elementos de teste foram obtidos do TMMi e da norma ISO/IEC/IEEE 29119. Para avaliar a extensão do modelo, um estudo de viabilidade foi conduzido. Os sujeitos do estudo de viabilidade avaliaram cada um dos 23 elementos de teste do RAModel_T. O estudo foi realizado em um *survey* online. Os resultados obtidos indicaram que o modelo precisava de ajustes, que foram indicados pelos próprios sujeitos. Além disso, a própria avaliação deverá ser complementada por outros tipos de estudo, como comentado nos trabalhos futuros.

6.2 Contribuições e Limitações da Dissertação

As contribuições identificadas na conclusão deste projeto de mestrado são: i) identificação do estado da arte nas pesquisas que relacionam arquitetura de software e teste de software; e ii) desenvolvimento de um modelo com elementos de teste de software para arquiteturas de referência, que contempla também a definição de três *viewpoints* de teste para arquiteturas de referência.

As principais limitações deste projeto de mestrado foram: i) ser um tema novo e, de certa forma, abstrato, que resultou em muito tempo nas etapas iniciais e na consolidação do trabalho; ii) apesar da avaliação do trabalho ter sido baseada na metodologia proposta por [Mafra, Barcelos e Travassos \(2006\)](#), em que a primeira e a segunda etapas foram conduzidas; iii) o exemplo de aplicação foi conduzido sem a execução do código, em que a arquitetura foi analisada à luz dos elementos do [RAModel_T](#).

6.3 Trabalhos Futuros

Nesta dissertação foi realizado um estudo de viabilidade com especialistas para avaliar o [RAModel_T](#). No estudo, os especialistas apresentaram suas considerações e críticas a respeito do modelo, caracterizado com um estudo de viabilidade. Seguindo metodologia proposta por [Shull, Carver e Travassos \(2001\)](#), a próxima etapa de avaliação empírica será realizar um estudo observacional por meio de um experimento em que os sujeitos serão alunos. Nesse estudo, sujeitos realizarão o planejamento teste para o teste de sistemas robóticos com base na RefSORS. Para isso, parte dos sujeitos realizará o planejamento de forma livre e a outra parte estará amparada pelo [RAModel_T](#) e pelo PROSA-RA_T. A partir disso, os *viewpoints* definidos também serão revisitados para que sejam realizados estudos de caso.

Neste trabalho foi evidenciado o potencial de arquiteturas de referência para dar suporte às atividades que não sejam apenas o projeto de arquiteturas de software, como o planejamento de teste. De forma semelhante, pretende-se também explorar de forma mais ampla as atividades de VV&T no contexto de arquiteturas de referência, principalmente em relação às atividades estáticas. Como última atividade no contexto de trabalhos futuros, pretende-se acompanhar o processo de estabelecimento de arquiteturas de referência para outros domínios, como o de saúde. E, dessa forma, verificar se os elementos de teste inseridos no [RAModel_T](#) compreendem a variabilidade e as características de diversos domínios.

6.4 Produção Científica

Como parte dos trabalhos desenvolvidos neste projeto, os seguintes trabalhos foram publicados:

- Souza, N. M.; Simão, D. D.; Lana, C. A.; Oliveira, L. B. R.; Nakawaga, E. Y.; Maldonado, J. C.. Exploring together Software Architecture and Software Testing: A Systematic Mapping. In: 35th International Conference of the Chilean Computer Science Society (SCCC 2016).
- Lana, C. L.; SOUZA, N. M.; Delamaro, M. E.; NAKAGAWA, E. Y.; MALDONADO, J. C.. System-of-Systems Development: Initiatives, Trends, and Challenges. In: Conferencia Latinoamericana de Informática, 2016, Valparaíso. Latin American Computing Conference (CLEI), 2016.

REFERÊNCIAS

AARTS, F.; KUPPENS, H.; TRETMAANS, J.; VAANDRAGER, F. W.; VERWER, S. Improving active Mealy machine learning for protocol conformance testing. **Machine Learning**, v. 96, n. 1-2, p. 189–224, 2014. Citado na página 50.

ADDU, I. D.; AHAMED, S. I.; YAU, S. S.; BUDURU, A. A Reference Architecture for Improving Security and Privacy in Internet of Things Applications. In: **Proceedings of the IEEE International Conference on Mobile Services (MS 2014)**. Alaska, USA: IEEE, 2014. p. 108–115. Citado na página 38.

AFZAL, W.; ALONE, S.; GLOCKSIEN, K.; TORKAR, R. Software test process improvement approaches: A systematic literature review and an industrial case study. **Journal of Systems and Software**, v. 111, p. 1–33, 2016. Citado na página 27.

ALI, S.; YUE, T.; HOFFMANN, A.; WENDLAND, M. F.; BAGNATO, A.; BROSSE, E.; SCHACHER, M.; DAI, Z. R. How Does the UML Testing Profile Support Risk-Based Testing. In: **Proceedings of the 27th IEEE International Symposium on Software Reliability Engineering (ISSRE 2016)**. Nápolis, Itália: IEEE, 2014. p. 311–316. Citado nas páginas 57 e 74.

ANDRADE, J.; ARES, J.; MARTÍNEZ, M.-A.; PAZOS, J.; RODRÍGUEZ, S.; ROMERA, J.; SUÁREZ, S. An architectural model for software testing lesson learned systems. **Information and Software Technology**, Elsevier B.V., v. 55, n. 1, p. 18–34, 2013. Citado nas páginas 27 e 59.

ANGELOV, S.; GREFFEN, P.; GREEFHORST, D. A framework for analysis and design of software reference architectures. **Information and Software Technology**, Elsevier, v. 54, n. 4, p. 417–431, 2012. Citado nas páginas 25, 34, 37, 39, 40 e 41.

ANGELOV, S.; TRIENEKENS, J.; GREFFEN, P. Towards a Method for the Evaluation of Reference Architectures: Experiences from a Case. In: **Proceedings of the European Conference on Software Architecture (ECSA 2008)**. Paphos, Chipre: Springer Berlin Heidelberg, 2008. p. 225–240. Citado nas páginas 39 e 40.

ARCURI, A.; IQBAL, M. Z.; BRIAND, L. Random Testing: Theoretical Results and Practical Implications. **IEEE Transactions on Software Engineering**, IEEE, v. 38, n. 2, p. 258–277, 2011. Citado na página 49.

AUTOSAR. AUTOSAR (AUTomotive Open System ARchitecture). 2015. <www.autosar.org>. [Online, acessado em: 25/07/2017]. Citado nas páginas 26, 38, 46 e 47.

BAKER, P.; DAI, Z. R.; GRABOWSKI, J.; HAUGEN, O.; SCHIEFERDECKER, I.; WILLIAMS, C. **Model-Driven Testing: Using the UML Testing Profile**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. Citado na página 57.

- BARBOSA, E. F.; CHAIM, M. L.; VINCENZI, A. M. R.; DELAMARO, M. E.; JINO, M.; MALDONADO, J. C. Capítulo 4 - Teste Estrutural. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). **Introdução ao Teste de Software**. 2. ed. Rio de Janeiro: Elsevier, 2016. p. 59–92. Citado na página [50](#).
- BARROCA, L.; HALL, J.; HALL, P. An Introduction and History of Software Architectures, Components, and Reuse. In: **Software Architectures**. [S.l.]: Springer London, 2000. p. 1–11. Citado na página [33](#).
- BASILI, V.; HEIDRICH, J.; LINDVALL, M.; MÜNCH, J.; REGARDIE, M.; ROMBACH, D.; SEAMAN, C.; TRENDOWICZ, A. GQM+Strategies: A Comprehensive Methodology for Aligning Business Strategies with Software Measurement. In: **Proceedings of the DASMA Software Metric Congress (MetriKon 2007)**. Kaiserslautern, Germany: Magdeburger Schriften zum Empirischen Software Engineering, 2007. p. 253–266. Citado na página [97](#).
- BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3. ed. [S.l.]: Addison-Wesley Professional, 2012. 640 p. Citado nas páginas [25](#), [33](#), [34](#), [35](#), [36](#), [37](#) e [38](#).
- BERTOLINO, A.; BUCCHIARONE, A.; GNESI, S.; MUCCINI, H. An architecture-centric approach for producing quality systems. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, Springer, v. 3712, p. 21–37, 2005. Citado na página [122](#).
- BRITO, P.; LEMOS, R. D.; RUBIRA, C.; MARTINS, E. Architecting fault tolerance with exception handling: Verification and validation. **Journal of Computer Science and Technology**, Springer US, v. 24, n. 2, p. 212–237, 2009. Citado na página [122](#).
- BRITTON, T.; JENG, L.; CARVER, G.; CHEAK, P.; KATZENELLENBOGEN, T. **Reversible Debugging Software**. [S.l.], 2013. 482 p. Citado na página [27](#).
- BUCCHIARONE, A.; MUCCINI, H.; PELLICCIONE, P.; PIERINI, P. Model-checking plus testing: from software architecture analysis to code testing. In: **Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics**. Toledo, Espanha: [s.n.], 2004. p. 351–365. Citado na página [122](#).
- BUCHGEHER, G.; KLAMMER, C.; HEIDER, W.; SCHÜETZ, M.; HUBER, H. Improving Testing in an Enterprise SOA with an Architecture-Based Approach. In: **Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016)**. [S.l.]: IEEE, 2016. p. 231–240. Citado na página [123](#).
- BURNSTEIN, I.; HOMYEN, A.; GROM, R.; CARLSON, C. A model to assess testing process maturity. **Crosstalk: The Journal of Defense Software Engineering**, v. 11, p. 26 – 30, 1998. Citado na página [55](#).
- BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **Pattern-oriented Software Architecture: A System of Patterns**. New York, NY, USA: John Wiley & Sons, Inc., 1996. Citado nas páginas [35](#) e [36](#).
- CARACCILO, A. A unified approach to automatic testing of architectural constraints. In: **Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)**. Piscataway, EUA: IEEE, 2015. p. 871–874. Citado nas páginas [123](#) e [128](#).

CLEMENTS, P. C.; BACHMANN, F.; BASS, L.; GARLAN, D.; IVERS, J.; LITTLE, R.; MERSON, P.; NORD, R.; STAFFORD, J. A. **Documenting Software Architectures: Views and Beyond**. 2. ed. [S.l.]: Addison-Wesley Professional, 2010. 592 p. Citado nas páginas 25, 34, 35 e 36.

CLOUTIER, R.; MULLER, G.; VERMA, D.; NILCHIANI, R.; HOLE, E.; BONE, M. The Concept of Reference Architectures. **Systems Engineering**, Wiley Subscription Services, Inc., v. 13, n. 1, p. 14–27, 2009. Citado nas páginas 39 e 43.

CMMI Product Team. **CMMI® for Development, Version 1.3**. Hanscom AFB, USA, 2010. 482 p. Citado nas páginas 27 e 55.

COSTA, A. D. da; LUCENA, C. J. P. de; VENIERIS, R.; CARVALHO, G. sing a new uml profile for modeling software tests. In: **Proceedings of the 8th International Conference on Software Engineering Advances (ISSRE 2016)**. Veneza, Itália: IARIA, 2013. p. 169–177. Citado na página 57.

CÁMARA, J.; LEMOS, R.; VIEIRA, M.; ALMEIDA, R.; VENTURA, R. Architecture-based resilience evaluation for self-adaptive systems. **Computing**, Springer Vienna, v. 95, n. 8, p. 689–722, 2013. Citado na página 25.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Capítulo 1 – Conceitos Básicos. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). **"Introdução Ao Teste de Software"**. 2. ed. Rio de Janeiro: Elsevier, 2016. p. 1–7. Citado nas páginas 26, 48 e 49.

_____. **Introdução ao teste de software**. 2. ed. Rio de Janeiro: Elsevier, 2016. 430 p. Citado nas páginas 26 e 49.

DELAMARO, M. E.; OLIVEIRA, R. A. P.; BARBOSA, E. F.; MALDONADO, J. C. Capítulo 5 - Teste de Mutação. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). **Introdução ao Teste de Software**. 2. ed. Rio de Janeiro: Elsevier, 2016. p. 93–142. Citado na página 50.

DOBRICA, L.; NIEMELÄ, E. An approach to reference architecture design for different domains of embedded systems. In: **Proceedings of the International Conference on Software Engineering Research & Practice (SERP 2008)**. [S.l.]: CSREA Press, 2008. p. 287–293. Citado na página 42.

EELES, P. Understanding Architectural Assets. In: **Proceedings of Working IEEE/IFIP Conference on the Software Architecture (WICSA 2008)**. Vancouver, Cananda: [s.n.], 2008. p. 267–270. Citado na página 35.

ENGLISCH, N.; HÄNCHEN, F.; ULLMANN, F.; MASRUR, A.; HARDT, W. Application-driven evaluation of autosar basic software on modern ecus. In: **Proceedings of the 13th IEEE International Conference on Embedded and Ubiquitous Computing (EUC 2015)**. [S.l.]: IEEE, 2015. p. 60–67. Citado na página 123.

FABBRI, S. C. P. F.; VINCENZI, A. M. R.; FREITAS, E. N. A.; MALDONADO, J. C. Capítulo 2 - Teste Funcional. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). **Introdução ao Teste de Software**. 2. ed. Rio de Janeiro: Elsevier, 2016. p. 9–38. Citado na página 49.

FAIRBANKS, G. **Just Enough Software Architecture: A Risk-Driven Approach**. 1. ed. Boulder: Marshall & Brainerd, 2010. 378 p. Citado na página 34.

FALESSI, D.; CANTONE, G.; KAZMAN, R.; KRUCHTEN, P. Decision-making techniques for software architecture design: A comparative survey. **ACM Computing Surveys**, ACM, New York, NY, USA, v. 43, n. 4, p. 33:1–33:28, out. 2011. Citado na página 36.

FERNANDEZ, E. B.; MONGE, R.; HASHIZUME, K. Building a security reference architecture for cloud systems. **Requirements Engineering**, Springer London, v. 20, Jan 2015. Citado nas páginas 26, 38 e 47.

GALSTER, M. Software reference architectures: related architectural concepts and challenges. In: **Proceedings of 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA 2015)**. [S.l.: s.n.], 2015. p. 1–4. Citado nas páginas 38 e 47.

GALSTER, M.; AVGERIOU, P. Empirically-grounded reference architectures: a proposal. In: **Proceedings of the Joint ACM SIGSOFT (QoSA 2011) Conference and ACM SIGSOFT Symposium (ISARCS 2011)**. New York, USA: ACM, 2011. p. 153–158. Citado na página 41.

GARCIA, C.; DAVILA, A.; PESSOA, M. Test Process Models: Systematic Literature Review. In: **Proceedings of the 14th International Conference on Software Process Improvement and Capability Determination (SPICE 2014)**. Vilnius, Lituânia: Springer International Publishing, 2014. p. 84–93. Citado na página 27.

GAROUSI, V.; FELDERER, M.; HACALOGLU, T. Software test maturity assessment and test process improvement: A multivocal literature review. **Information and Software Technology**, v. 85, p. 16 – 42, 2017. Citado na página 27.

GREENWAY, B. Robot accuracy. **Industrial Robot: An International Journal**, v. 27, n. 4, p. 257–265, 2000. Citado na página 81.

GUESSI, M.; OQUENDO, F.; NAKAGAWA, E. Y. Variability viewpoint to describe reference architectures. In: **Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2014)**. Sydney, Australia: ACM, 2014. p. 14:1–14:6. Citado na página 39.

HAO, W.; ZHANBAO, G.; JINSONG, Y.; LEILI, H. Design of a highly adaptive auto-test software platform based on common data interface. In: **Proceedings of the 6th IEEE Conference on Industrial Electronics and Applications (ICIEA 2011)**. Beijing, China: [s.n.], 2011. p. 254–258. Citado na página 123.

HARROLD, M. J. Architecture-based regression testing of evolving systems. In: **Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (Rosatea 1998)**. Sicília, Itália: [s.n.], 1998. p. 73–77. Citado na página 122.

HEESCH, U. v.; AVGERIOU, P.; HILLIARD, R. A documentation framework for architecture decisions. **Journal of Systems and Software**, Elsevier, v. 85, n. 4, p. 795–820, 2012. Citado nas páginas 34 e 36.

HOFMEISTER, C.; KRUCHTEN, P.; NORD, R. L.; OBBINK, H.; RAN, A.; AMERICA, P. A general model of software architecture design derived from five industrial approaches. **Journal of Systems and Software**, v. 80, n. 1, p. 106–126, 2007. Citado nas páginas 25, 36 e 37.

HOFMEISTER, C.; NORD, R.; SONI, D. Global Analysis: moving from software requirements specification to structural views of the software architecture. **IEE Proceedings–Software**, v. 152, n. 4, p. 187–197, Aug 2005. Citado na página 36.

HäNSEL, J.; VOGEL, T.; GIESE, H. A testing scheme for self-adaptive software systems with architectural runtime models. In: **Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW 2015)**. [S.l.]: IEEE, 2015. p. 134–139. Citado na página 123.

ISO-31000. **Risk management 31000:2009**, International Organization for Standardization (ISO). 2009. 21 p. Citado na página 74.

ISO/IEC/IEEE. **Systems and software engineering – Vocabulary. Standard 24765:2010(E)**, International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE). 2010. 1–418 p. Citado nas páginas 33, 34, 35 e 48.

_____. **Software and systems engineering – Software testing – Part 1: Concepts and definitions. Standard 29119-1:2013(E)**, International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE). 2013. 1–64 p. Citado na página 33.

_____. **Software and systems engineering – Software testing – Part 2: Test processes. Standard 29119-2:2013(E)**, International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE). 2013. 1–68 p. Citado nas páginas 26, 50, 51, 53, 77 e 82.

_____. **Software and systems engineering – Software testing – Part 3: Test Documentation. Standard 29119-3:2013(E)**, International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE). 2013. 1–130 p. Citado nas páginas 51, 52 e 54.

ISO/IEC/IEEE-42010. **Systems and software engineering – Architecture description. Standard 42010:2011(E)**, International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE). 2011. 1–46 p. Citado nas páginas 34, 35, 38 e 60.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. Citado na página 36.

JAVED, A. Z.; STROOPER, P. A.; WATSON, G. N. Automated Generation of Test Cases Using Model-Driven Architecture. In: **Proceedings of the 2nd International Workshop on Automation of Software Test (AST’07)**. Washington, Estados Unidos: [s.n.], 2007. p. 3. Citado na página 122.

Jl, Y.; OFFUTT, J. Deriving tests from software architectures. In: **Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE’01)**. Hong Kong, China: [s.n.], 2001. p. 308–313. Citado na página 122.

KAZMAN, R.; BASS, L.; WEBB, M.; ABOWD, G. SAAM: a method for analyzing the properties of software architectures. In: **Proceedings of the International conference on Software engineering (ICSE 1994)**. Los Alamitos, USA: IEEE, 1994. p. 81–90. Citado na página 36.

KAZMAN, R.; KLEIN, M.; BARBACCI, M.; LONGSTAFF, T.; LIPSON, H.; CARRIERE, J. The architecture tradeoff analysis method. In: **Proceedings of the International Conference on Engineering of Complex Computer Systems (ICECCS 2014)**. Monterey, USA: IEEE, 1998. p. 68–78. Citado na página 36.

- KIM, K.; KIM, R. Y. C. Improving test process for test organization assessed with TMMi based on TPI NEXT. **International Journal of Software Engineering and its Applications**, v. 8, n. 2, p. 59–66, 2014. Citado na página 55.
- KIRCHMER, M. Reference Models to Empower MPE. In: **High Performance Through Process Excellence**. [S.l.]: Springer Berlin Heidelberg, 2011. p. 87–101. Citado na página 43.
- KRUCHTEN, P. The 4+1 view model of architecture. **IEEE Software**, v. 12, n. 6, p. 42–50, Nov 1995. Citado nas páginas 36 e 37.
- _____. **The Rational Unified Process: An Introduction**. 3. ed. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. Citado na página 36.
- KRUCHTEN, P.; OBBINK, H.; STAFFORD, J. The Past, Present, and Future for Software Architecture. **IEEE Software**, IEEE, v. 23, n. 2, p. 22 – 30, 2006. Citado na página 33.
- LEROUX, E.; OQUENDO, F.; XIONG, Q. Architecture-based conformance testing. In: **Proceedings of the 8th International Conference on Software Engineering Advances (ICSEA 2013)**. Veneza, Italia: [s.n.], 2013. p. 55–64. Citado na página 123.
- LI, F.-S.; MA, W.-M.; CHAO, A. Architecture centric approach to enhance software testing management. In: **Proceedings of the 8th International Conference on Intelligent Systems Design and Applications (ISDA'08)**. Taiwan, China: [s.n.], 2008. p. 654–659. Citado na página 122.
- LIU, H.; SPICHKOVA, M.; SCHMIDT, H. W.; ULRICH, A.; SAUER, H.; WIEGHARDT, J. Efficient Testing Based on Logical Architecture. In: **Proceedings of the 24th Australasian Software Engineering Conference (ASWEC 2015)**. Adelaide, Austrália: ACM, 2015. p. 49–53. Citado na página 123.
- LIU, H.; SPICHKOVA, M.; SCHMIDT, H. W.; SELLIS, T.; DUCKHAM, M. Spatio-Temporal Architecture-Based Framework for Testing Services in the Cloud. In: **Proceedings of the 24th Australasian Software Engineering Conference (ASWEC 2015)**. Adelaide, Austrália: ACM, 2015. p. 18–22. Citado na página 123.
- LOCHAU, M.; LITY, S.; LACHMANN, R.; SCHAEFER, I.; GOLTZ, U. Delta-oriented model-based integration testing of large-scale systems. **Journal of Systems and Software**, Elsevier, v. 91, p. 63–84, 2014. Citado nas páginas 123 e 126.
- LUN, L.; DING, X. Analyzing relation between software architecture testing criteria on test sequences. In: **Proceedings of the 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement (QRS 2009)**. Shanghai, China: [s.n.], 2009. p. 181–186. Citado nas páginas 122 e 128.
- MACHADO, C. A.; SILVA, E.; BATISTA, T.; LEITE, J.; NAKAGAWA, E. Y. RA-Ubi: A reference architecture for ubiquitous computing. In: **Proceedings of the European Conference on Software Architecture (ECSA 2014)**. Vienna, Austria: Springer Berlin Heidelberg, 2014. p. 98–105. Citado nas páginas 26 e 38.
- MAFRA, S.; BARCELOS, R.; TRAVASSOS, G. Aplicando uma metodologia baseada em evidência na definição de novas tecnologias de software. In: **20th Brazilian Symposium on Software Engineering**. [S.l.]: Sociedade Brasileira de Computação, 2006. p. 239–254. Citado nas páginas 30, 95, 108 e 149.

MARINESCU, R.; SAADATMAND, M.; BUCAIONI, A.; SECELEANU, C.; PETTERSSON, P. A model-based testing framework for automotive embedded systems. In: **Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2014)**. Verona, Itália: [s.n.], 2014. p. 38–47. Citado na página 123.

MARTÍNEZ-FERNÁNDEZ, S.; AYALA, C. P.; FRANCH, X.; MARQUES, H. M. REARM: A Reuse-Based Economic Model for Software Reference Architectures. In: **Proceedings of the International Conference on Software Reuse (ICSR 2011)**. Pisa, Italy: Springer Berlin Heidelberg, 2011. p. 97–112. Citado na página 42.

MARTÍNEZ-FERNÁNDEZ, S.; AYALA, C. P.; FRANCH, X.; MARQUES, H. M. Benefits and drawbacks of software reference architectures: A case study. **Information and Software Technology**, p. –, 2017. Citado na página 41.

MILLER, T.; STROOPER, P. A framework and tool support for the systematic testing of model-based specifications. **ACM Transactions on Software Engineering and Methodology**, ACM, v. 12, n. 4, p. 409–439, 2003. Citado na página 50.

_____. A case study in model-based testing of specifications and implementations. **Software Testing, Verification and Reliability**, John Wiley & Sons, Ltd, v. 22, n. 1, p. 33–63, 2012. Citado na página 50.

MUCCINI, H. Using model differencing for architecture-level regression testing. In: **Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA'07)**. Luebeck, Alemanha: [s.n.], 2007. p. 59–66. Citado nas páginas 122 e 128.

MUCCINI, H.; BERTOLINO, A.; INVERARDI, P. Using software architecture for code testing. **IEEE Transactions on Software Engineering**, IEEE, v. 30, n. 3, p. 160–171, 2004. Citado na página 25.

MUCCINI, H.; DIAS, M.; RICHARDSON, D. Systematic testing of software architectures in the C2 style. In: **Fundamental Approaches to Software Engineering**. Barcelona, Espanha: [s.n.], 2004. p. 295–309. Citado na página 122.

_____. Software architecture-based regression testing. **Journal of Systems and Software**, v. 10, n. 79, p. 1379–1396, 2006. Citado nas páginas 122 e 128.

MUCCINI, H.; INVERARDI, P.; BERTOLINO, A. Using software architecture for code testing. **IEEE Transactions on Software Engineering**, IEEE, v. 30, n. 3, p. 160–171, 2004. Citado nas páginas 122 e 127.

MULLER, G. **The Gaudi Project Website**. 2005. <www.gaudisite.nl>. [Online, acessado em: 25/07/2017]. Citado na página 36.

_____. Right sizing reference architectures. how to provide specific guidance with limited information. In: . Utrecht, The Netherlands: [s.n.], 2008. v. 18, n. 1, p. 2047–2054. Citado na página 42.

MURER, S.; HAGEN, C. Fifteen Years of Service Oriented Architecture at Credit Suisse. **IEEE Software**, IEEE, v. 31, n. 6, p. 9–15, 2014. Citado na página 35.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. 3. ed. [S.l.]: John Wiley & Sons, Ltd, 2011. 240 p. Citado nas páginas 27, 34 e 48.

- NAKAGAWA, E. Y. **Uma contribuição ao Projeto Arquitetural de Ambientes de Engenharia de Software**. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação/USP, São Carlos, SP, 2006. Citado na página 61.
- 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: **Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2014)**. Sydney, Australia: IEEE, 2014. p. 143–152. Citado nas páginas 25, 26, 41, 42 e 43.
- NAKAGAWA, E. Y.; OQUENDO, F.; BECKER, M. RAModel: A Reference Model for Reference Architectures. In: **Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture (WICSA/ECSA 2012)**. Helsinki, Finland: IEEE, 2012. p. 297–301. Citado nas páginas 15, 25, 26, 30, 43, 44, 45, 46, 59, 60, 61, 107, 139 e 149.
- NAKAGAWA, E. Y.; OQUENDO, F.; MALDONADO, J. C. Reference Architectures. In: **Software Architecture 1**. [S.l.]: John Wiley & Sons, Ltd, 2014. p. 55–82. Citado nas páginas 26, 37, 39, 41, 43, 47 e 59.
- OASIS. **Reference Architecture Foundation for Service Oriented Architecture Version 1.0**. [S.l.], 2012. 118 p. Citado nas páginas 26, 81 e 87.
- OLIVEIRA, L. B. R. **Architectural design of service-oriented robotic systems**. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação/USP, São Carlos, SP, 2015. Citado nas páginas 26, 47, 79 e 80.
- OLIVEIRA, L. B. R.; NAKAGAWA, E. Y. A Service-Oriented Reference Architecture for Software Testing Tools. In: **Proceedings of the European Conference on Software Architecture (ECSA 2011)**. Essen, Germany: Springer Berlin Heidelberg, 2011. p. 405–421. Citado na página 38.
- OMG, O. M. G. **UML Testing Profile (UTP) Version 1.2**. [S.l.], 2012. Disponível em: <<http://www.omg.org/spec/UTP/1.2/>>. Citado nas páginas 30 e 57.
- PARK, G.; KU, D.; LEE, S.; WON, W.-J.; JUNG, W. Test methods of the autosar application software components. In: **International Congress Center - The Society of Instrument and Control Engineers (ICCAS-SICE'09)**. Fukuoka, Japão: [s.n.], 2009. p. 2601–2606. Citado na página 122.
- PAULISCH, F.; ZIMMERER, P. Collaboration of software architect and test architect helps to systematically bridge product lifecycle gap. In: **Proceedings of the 1st IEEE/ACM International Workshop on Bringing Architectural Design Thinking Into Developers' Daily Activities (BRIDGE 2016)**. Austin, EUA: IEEE, 2016. p. 11–13. Citado na página 123.
- PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. 7. ed. [S.l.]: McGraw-Hill Education, 2010. 895 p. Citado nas páginas 25, 35 e 48.
- RAN, A. ARES Conceptual Framework for Software Architecture. In: JAZAYERI, M.; RAN, A.; LINDEN, F. van der (Ed.). **Software Architecture for Product Families Principles and Practice**. Primeira edição. Boston, EUA: Addison-Wesley, 2000. p. 1–29. Citado na página 36.
- REZA, H.; GRANT, E. A method to test concurrent systems using architectural specification. **Journal of Supercomputing**, Kluwer Academic Publishers-Plenum Publishers, v. 39, n. 3, p. 347–357, 2007. Citado na página 122.

REZA, H.; LANDE, S. Model based testing using software architecture. In: **Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG'10)**. Las Vegas, Estados Unidos: [s.n.], 2010. p. 188–193. Citado na página 122.

RICHARDSON, D.; WOLF, A. Software testing at the architectural level. In: **Joint Proceedings of the 2nd International Software Architecture Workshop (ISAW'96)**. San Francisco, Estados Unidos: [s.n.], 1996. p. 68–71. Citado na página 122.

ROSENBLUM, D. S. Challenges in exploiting architectural models for software testing. In: **Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (Rosatea 1998)**. Sicília, Itália: [s.n.], 1998. p. 49–53. Citado na página 122.

ROZANSKI, N.; WOODS, E. **Software systems architecture: Working with stakeholders using viewpoints and perspectives**. 2. ed. [S.l.]: Addison–Wesley Professional, 2011. 704 p. Citado na página 126.

SCHNEIDER, U.; DRUST, M.; ANSALONI, M.; LEHMANN, C.; PELLICCIARI, M.; LEALI, F.; GUNNINK, J. W.; VERL, A. Improving robotic machining accuracy through experimental error investigation and modular compensation. **The International Journal of Advanced Manufacturing Technology**, v. 85, n. 1, p. 3–15, 2014. Citado na página 81.

SCOLLO, G.; ZECCHINI, S. Architectural Unit Testing. **Electronic Notes in Theoretical Computer Science**, Elsevier, v. 111, p. 27–52, 2005. Citado nas páginas 122 e 128.

SERRANO, N.; HERNANTES, J.; GALLARDO, G. Service-Oriented Architecture and Legacy Systems. **IEEE Software**, IEEE, v. 31, n. 5, p. 15–19, 2014. Citado na página 35.

SHULL, F.; CARVER, J.; TRAVASSOS, G. H. An empirical methodology for introducing software processes. In: **Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering**. New York, USA: ACM, 2001. (ESEC/FSE-9), p. 288–296. Citado nas páginas 30, 95, 108 e 149.

SIMÃO, A. S. Capítulo 3 – Teste Baseado em Modelos. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). **Introdução ao Teste de Software**. 2. ed. Rio de Janeiro: Elsevier, 2016. p. 39–58. Citado na página 50.

SIMÃO, D. D. **Caracterização de Informações de Teste de Software no Processo de Estabelecimento de Arquiteturas de Referência**. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação/USP, São Carlos, SP, 2017. Citado nas páginas 28 e 79.

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro**. [S.l.], 2012. Disponível em: <<http://www.softex.br/mpsbr/>>. Citado na página 55.

SOUZA, N. M. d.; SIMÃO, D. D.; NAKAGAWA, E. Y.; MALDONADO, J. C. **Relação entre arquitetura de software e teste de software: um mapeamento sistemático**. [S.l.], 2017. Citado nas páginas 29, 30, 31 e 95.

SOUZA, N. M. d.; SIMÃO, D. D.; OLIVEIRA, L. B. R. d.; LANA, C. A.; NAKAGAWA, E. Y.; MALDONADO, J. C. **Relação entre Arquitetura de Software e Teste de Software: uma Atualização de Mapeamento Sistemático**. [S.l.], 2017. Citado nas páginas 27, 29, 30, 31 e 95.

- SOUZA, N. M. de; DIAS, D.; LANA, C. A.; OLIVEIRA, L. B. R. de; NAKAGAWA, E. Y.; MALDONADO, J. C. Exploring together Software Architecture and Software Testing: A Systematic Mapping. In: **SCCC 2016**. Valparaíso, Chile: IEEE, 2016. p. X–X. Citado na página 29.
- TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G. do. **Introdução à Engenharia de Software Experimental**. Rio de Janeiro, 2002. 52 p. Citado nas páginas 95 e 96.
- TRETMANS, J. Model Based Testing with Labelled Transition Systems. In: **Proceedings of the Formal Methods and Testing (FORTEST 2008)**. [S.l.]: Springer Berlin Heidelberg, 2008. p. 1–38. Citado na página 50.
- TSAI, W.-T.; GAO, J.; WEI, X.; CHEN, Y. Testability of software in service-oriented architecture. In: **Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)**. Washington, Estados Unidos: [s.n.], 2006. p. 163–168. Citado na página 122.
- TYREE, J.; AKERMAN, A. Architecture decisions: demystifying architecture. **IEEE Software**, IEEE, v. 22, n. 2, p. 19–27, 2005. Citado na página 34.
- ULRICH, A. Sustainable Architectures: A Tester's Perspective. In: **Proceedings of the 2015 European Conference on Software Architecture Workshops (ECSAW 2015)**. Dubrovnik, Croácia: ACM, 2015. p. 61:1–61:2. Citado na página 123.
- VEENENDAAL, E. v. **Test Maturity Model integration**. 2012. <<http://www.tmmi.org/pdf/TMMi.Framework.pdf>>. [Online, acessado em: 08/07/2015]. Citado nas páginas 51 e 55.
- WANG, F.; WANG, S.; JI, Y. An automatic generation method of executable test case using model-driven architecture. In: **Proceedings of the 4th International Conference on Innovative Computing, Information and Control (ICICIC'09)**. Kaohsiung, Taiwan: [s.n.], 2009. p. 389–393. Citado na página 122.
- WIESBADEN, S. A. M. “AUTOSAR has Become Mature and Accepted”. **ATZextra worldwide**, Springer Automotive Media, v. 18, n. 9, p. 13–15, 2013. Citado na página 47.
- WOHLIN, C.; RUNESON, P.; HöST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering**. 2. ed. [S.l.]: Springer Berlin Heidelberg, 2012. 249 p. Citado nas páginas 96 e 104.
- YANG, L.; LI, Y.; WANG, P. Design and implementation of automatic generation of test cases based on model driven architecture. In: **Proceedings of the 2nd International Conference on Information Technology and Computer Science (ITCS'2010)**. Kiev, Ucrânia: [s.n.], 2010. p. 344–347. Citado na página 122.
- YE, J.; DONG, W.; QI, Z. A method to generate embedded real-time system test suites based on software architecture specifications. In: **Proceedings of the 9th International Conference for Young Computer Scientists (ICYCS'08)**. Hunan, China: [s.n.], 2008. p. 2325–2329. Citado na página 122.
- YOUNG, M. Testing complex architectural conformance relations. In: **Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (Rosatea 1998)**. Sicília, Itália: [s.n.], 1998. p. 42–45. Citado na página 122.

RESUMO DO MAPEAMENTO SISTEMÁTICO

A.1 Considerações Iniciais

Neste capítulo é apresentado um resumo dos resultados obtidos em um mapeamento sistemático conduzido em 2016 e atualizado em 2017. O objetivo desse mapeamento é “*caracterizar o estado da arte de como as atividades de projeto de arquitetura de software e teste de software têm sido exploradas de forma complementar no processo de desenvolvimento de software*”. A partir dos resultados, concluiu-se que o uso de arquitetura de software e de teste de software de maneira conjunta encontra-se ainda em um estágio inicial. Ainda existem oportunidades e desafios de pesquisa para a proposição de abordagens que visem a contribuir para a produção de sistemas de software de alta qualidade. Contudo, a área já mobiliza pesquisadores de universidades, indústria e centros de pesquisa distribuídos em 13 países. O ápice de publicações foi no ano de 2015, quando seis de 35 estudos foram publicados. A partir de 2014 as publicações foram realizadas apenas em conferências e *workshops*.

Este capítulo está organizado da seguinte forma. Na Seção [A.2](#) é exposta uma breve descrição sobre a condução dos estudos retornados no mapeamento e as respostas das três questões de pesquisa. Na Seção [A.3](#) as considerações finais do capítulo são apresentadas.

A.2 Resumo dos resultados do mapeamento

No Quadro [4](#) estão listados todos os estudos identificados no mapeamento sistemático. As colunas dos quadros contém um índice para facilitar o acompanhamento da leitura, o título dos estudos, os autores e o ano em que foram publicados e o código utilizado para referenciar cada estudo.

Tabela 4 – Estudos incluídos no mapeamento sistemático

id	Título	Autores	Código
1	Software Testing at the Architectural Level	Richardson e Wolf (1996)	S01
2	Architecture-Based Regression Testing of Evolving Systems	Harrold (1998)	S02
3	Challenges in Exploiting Architectural Models for Software Testing	Rosenblum (1998)	S03
4	Testing Complex Architectural Conformance Relations	Young (1998)	S04
5	Deriving Tests from Software Architectures	Ji e Offutt (2001)	S05
6	Model-Checking Plus Testing: from Software Architecture Analysis to Code Testing	Bucchiarone <i>et al.</i> (2004)	S06
7	Systematic Testing of Software Architectures in the C2 Style	Muccini, Dias e Richardson (2004)	S07
8	Using Software Architecture for Code Testing	Muccini, Inverardi e Bertolino (2004)	S08
9	An Architecture-Centric Approach for Producing Quality Systems	Bertolino <i>et al.</i> (2005)	S09
10	Architectural Unit Testing	Scollo e Zecchini (2005)	S10
11	Software Architecture-based Regression Testing	Muccini, Dias e Richardson (2006)	S11
12	Testability of Software in Service-oriented Architecture	Tsai <i>et al.</i> (2006)	S12
13	Automated Generation of Test Cases Using Model-Driven Architecture	Javed, Strooper e Watson (2007)	S13
14	Using Model Differencing for Architecture-Level Regression Testing	Muccini (2007)	S14
15	A Method to Test Concurrent Systems Using Architectural Specification	Reza e Grant (2007)	S15
16	Architecture Centric Approach to Enhance Software Testing Management	Li, Ma e Chao (2008)	S16
17	A Method to Generate Embedded Real-Time System Test Suites Based on Software Architecture Specifications	Ye, Dong e Qi (2008)	S17
18	Architecting Fault Tolerance with Exception Handling: Verification and Validation	Brito <i>et al.</i> (2009)	S18
19	Analyzing Relation between Software Architecture Testing Criteria on Test Sequences	Lun e Ding (2009)	S19
20	Test Methods of the AUTOSAR Application Software Components	Park <i>et al.</i> (2009)	S20
21	An Automatic Generation Method of Executable Test Case Using Model-driven Architecture	Wang, Wang e Ji (2009)	S21
22	Model Based Testing Using Software Architecture	Reza e Lande (2010)	S22
23	Design and Implementation of Automatic Generation of Test Cases Based on Model Driven Architecture	Yang, Li e Wang (2010)	S23

Tabela 4 – Continua: Estudos incluídos no mapeamento sistemático

id	Título	Autores	Código
24	Design of a Highly Adaptive Auto-test Software Platform Based on Common Data Interface	Hao <i>et al.</i> (2011)	S24
25	Architecture-Based Conformance Testing	Leroux, Oquendo e Xiong (2013)	S25
26	Delta-Oriented Model-based Integration Testing of Large-Scale Systems	Lochau <i>et al.</i> (2014)	S26
27	A Model-Based Testing Framework for Automotive Embedded Systems	Marinescu <i>et al.</i> (2014)	S27
28	Collaboration of Software Architect and Test Architect Helps to Systematically Bridge Product Lifecycle Gap	Paulisch e Zimmerer (2016)	S28
29	Sustainable Architectures: A Tester's Perspective	Ulrich (2015)	S29
30	Spatio-Temporal Architecture-Based Framework for Testing Services in the Cloud	Liu <i>et al.</i> (2015b)	S30
31	Efficient Testing Based on Logical Architectures	Liu <i>et al.</i> (2015a)	S31
32	Application-Driven Evaluation of AUTOSAR Basic Software on Modern ECUs	Englisch <i>et al.</i> (2015)	S32
33	Improving Testing in an Enterprise SOA with an Architecture-Based Approach	Buchgeher <i>et al.</i> (2016)	S33
34	A Unified Approach to Automatic Testing of Architectural Constraints	Caracciolo (2015)	S34
35	A Testing Scheme for Self-Adaptive Software Systems with Architectural Runtime Model-based	Hänsel, Vogel e Giese (2015)	S35

RQ-1 - Quais abordagens têm explorado arquitetura de software e teste de software em conjunto no processo de desenvolvimento de software?

O objetivo dessa questão é identificar como arquitetura de software e teste de software têm sido relacionados na literatura. Como principal resultado, foi identificado que os estudos são relacionados por meio do teste baseado em arquitetura. No qual, a arquitetura, por meio de suas descrições e artefatos, é utilizada como guia para a atividade de teste. Dos 35 estudos selecionados, 32 abordam esse tema. Os estudos foram publicados entre os anos de 1996 até 2016, sendo que no ano de 2015 houve o maior número de publicações.

O Quadro 24 será utilizado para apresentar os resultados obtidos, uma vez que sumariza todos os estudos com suas informações e classificações. As classificações definidas para os dados extraídos foram “Atividade de teste”, “Fases”, “Técnica”, “Objetivo”, “Outros”. Atividade de teste: conjunto dos artefatos que são utilizados durante a condução da atividade de teste. Fases:

conjunto das fases de teste em que os estudos foram conduzidos. Técnicas: conjunto das técnicas de teste que foram utilizadas ou indicadas pelos estudos. Objetivo: o objetivo da abordagem que relaciona arquitetura de software e teste de software. Outros: informação a respeito de teste que não se enquadram nas outras categorias anteriores.

Quadro 24 – Quadro comparativo entre estudos com relação ao teste de software

Tópicos de teste	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S32	S33	S34	S35
Atividade de teste	Casos de teste	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓				
	Critério de teste	✓	✓		✓			✓				✓							✓		✓				✓	✓			✓	✓					
	Critério de adequação			✓				✓																											
	Sequências de teste				✓			✓							✓			✓	✓																
	Gerenciar atividade de teste															✓																			
	Gerar oráculo de teste			✓																															
	Gerar plano de teste	✓						✓														✓													
Fases de teste	Aceitação																																	✓	
	Carga																																	✓	
	Fumaça																																	✓	
	Integração	✓			✓		✓	✓			✓				✓			✓				✓		✓	✓	✓			✓			✓	✓		
	Regressão	✓	✓	✓							✓			✓																					
	Sistema																													✓			✓		
	Unidade	✓								✓		✓						✓									✓			✓			✓		
Técnica	"Grey-Box"																																		✓
	Funcional	✓				✓	✓	✓	✓	✓	✓			✓	✓			✓	✓	✓		✓			✓	✓	✓			✓	✓	✓	✓	✓	
	Estrutural																			✓															
Objetivo	Teste baseado em arquitetura	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	
	Validação de arquitetura	✓				✓			✓									✓																	
Outros	Testabilidade	✓	✓							✓		✓				✓				✓	✓								✓						
	Estilo arquitetural						✓					✓																					✓		
	Teste de conformidade	✓			✓	✓	✓	✓	✓	✓	✓			✓				✓		✓		✓			✓	✓								✓	
	Fluxo de dados				✓																									✓					
	Fluxo de controle																													✓	✓				
	Papel do Testador																											✓							
	Viewpoint																																✓		

Pelo resultado obtido no mapeamento, pôde-se classificar o teste baseado em arquitetura em: i) teste baseado em modelos; ii) estilo arquitetural (nesse caso, SOA); iii) transformações MDA. Dados os estudos que não partiram da validação arquitetural, os autores consideraram que as arquiteturas estavam adequadas para dar o suporte a atividade de teste. Apenas quatro estudos, considerando os 35, submeteram suas arquiteturas a um processo de validação arquitetural como atividade prévia ao teste baseado em arquitetura. Os artefatos relevantes para a atividade de teste produzidos nos estudos são apresentados no Quadro 25.

Quadro 25 – Informações e artefatos de teste gerados nos estudos

Artefato	Descrição	QE	PO
Casos de teste	Casos de teste são selecionados ou derivados a partir da arquitetura do sistema em teste.	23	65%
Critério de teste	São definidos critérios de cobertura e funcionalidades para partes específicas de uma arquitetura.	10	29%
Critério de adequação	Crítérios são definidos para verificar se um conjunto de casos de teste é adequado para um sistema.	2	6%
Sequências de teste	Modelos da arquitetura são utilizados para derivar sequências de teste que poderão ser utilizadas em modelos.	5	14%
Gerenciar atividade de teste	A arquitetura de software é utilizada para contribuir para o gerenciamento da atividade de teste.	1	3%
Oráculo de teste	A arquitetura de software é utilizada para derivar oráculos para a atividade de teste.	1	3%
Plano de teste	A arquitetura é utilizada como apoio ao desenvolvimento de um plano de teste.	3	9%

Foram identificadas informações arquiteturais que nortearam a atividade de teste nas abordagens identificadas, conforme Quadro 26. Essas informações retratam o modo como o teste de software foi relacionado à arquitetura de software. Dentre os trabalhos, 23% abordaram aspectos de testabilidade de sistemas de software. O teste de conformidade foi utilizado em 43% dos estudos e o teste de fluxo de dados e fluxo de controle da arquitetura foram considerados em 12% dos estudos. Os demais estudos, ou seja, 31%, não se encaixaram em nenhuma das características identificadas.

Quadro 26 – Características gerais de teste

Categoria	Descrição	QE	PO
Testabilidade	A testabilidade de sistemas de software dada sua arquitetura.	9	23%
Estilo arquitetural	O estilo arquitetural é utilizado como fonte para derivar requisitos de teste.	3	6%
Teste de conformidade	No estudo, tem-se o objetivo de verificar a conformidade entre a arquitetura e o sistema implementado	15	43%
Fluxo de dados	O estudo utiliza o fluxo de dados entre elementos da arquitetura para derivar requisitos de teste.	4	12%
Fluxo de controle	O estudo utiliza o fluxo de controle entre elementos da arquitetura para derivar requisitos de teste.	4	12%
<i>Viewpoint</i>	Foi definido um <i>viewpoint</i> dentro da arquitetura destinada a informações que serão úteis a atividade de teste, em que são destacadas as vulnerabilidades do sistema em teste.	1	3%
Papel do Testador	Foi discutido o papel do testador enquanto um arquiteto de teste.	1	3%
Não especificaram	O estudo não se enquadrava nas categorias identificadas.	11	31%

RQ-2 - Quais informações arquiteturais têm sido consideradas para apoiar a atividade de teste?

Na segunda questão de pesquisa, buscou-se caracterizar como as informações arquiteturais foram obtidas e descritas no escopo das abordagens identificadas na RQ-1. As análises estática e dinâmica, transformações de modelos MDA e o gerenciamento de configuração foram os meios utilizados para obter as informações das arquiteturas. Também foi identificado que diversas linguagens foram utilizadas para descrever as arquiteturas e apenas duas linguagens de teste foram mencionadas.

A análise estática foi utilizada por 70% dos estudos e refere-se à análise de elementos em tempo de projeto e sua organização (ROZANSKI; WOODS, 2011), por exemplo, a utilização de modelos arquiteturais como base para planejar o teste de unidade e de integração de grandes sistemas (LOCHAU *et al.*, 2014). A estrutura e a descrição desses elementos são utilizadas como fontes de informação para a atividade de teste. Na análise dinâmica, as relações dos elementos e suas interações são consideradas como fontes de informação para a atividade de teste. Por exemplo, a troca de mensagens entre elementos da arquitetura ou, até mesmo, a especificação das interfaces. O método de transformação de modelos de MDA foi adotado por 11% dos estudos para definir casos de teste abstratos e, posteriormente, refinados para casos de teste concretos por meio de transformações entre o PIM e o PSM. Além disso, em 11% dos estudos, o gerenciamento de configuração da documentação arquitetural atua como apoio para selecionar casos de teste para o teste de regressão.

No Quadro 27 são apresentadas as linguagens utilizadas para descrever e representar às arquiteturas. A maioria das linguagens, 21 delas, foram utilizadas para descrever as arquiteturas. As linguagens TTNC-3 e TDL foram indicadas como meios potenciais para descrever informações ou visões de teste de software. Apesar de não terem sido apresentados exemplos ou estudos de caso com as linguagens para teste, o fato delas existirem e serem relacionadas as arquiteturas de software indica um possível interesse em representar teste de software em arquiteturas de software.

Quadro 27 – Linguagens utilizadas para representar a arquitetura

Linguagem	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S32	S33	S34	S35		
CHAM	✓	✓	✓																✓																		
ADL Wright					✓																																
LTS							✓	✓																													
π -ADL																									✓												
HPrTNs										✓	✓			✓		✓			✓			✓		✓													
UML									✓	✓		✓		✓				✓			✓		✓														
B-METHOD																		✓																			
EAST-ADL																																					
MEF				✓		✓			✓		✓															✓		✓									
IOLTS										✓															✓												
Basic LO-TOS										✓																											
EMF												✓																									
DRTSADL																		✓																			
TIOA																		✓																			
ACME ADL																						✓															
<i>Model Delta</i>																																					
ADL																										✓											
Dicto																																			✓		
FSA																																					
<i>Petri Nets</i>																																✓					
RDAL																																✓					
TDL																																✓					
TTCN-3																													✓								
Não especificaram																												✓					✓	✓		✓	

Legenda: CHAM - Chemical Abstract Machine; LTS - Labeled Transition System; HPrTNS - Hierarchical Predicate Transition Nets; UML - Unified Modeling Language; DRTSADL - Architecture description language of distributed/embedded real-time system; IOLTS - Input/Output Labelled Transition Systems; EMF - Eclipse Modeling Framework; TIOA - Timed Input/Output Automaton; ADL - Architecture Description Language; MEF - Máquina de Estados Finita; RDAL - Rich Architecture Description Language; TDL - Test Description Language; FSA - Finite State Automata; TTNC3- Testing and Test Control Notation Version 3.

A linguagem mais utilizada nos estudos foi a UML, adotada por nove estudos. Os estudos S9, S10, S12, S17, S18, S22, S25 e S26 adotaram mais de uma linguagem para representar a arquitetura. Um exemplo é o estudo S09 que considera as fontes de informações estática e dinâmica e, em função disso, utiliza a UML e MEF para representar a arquitetura.

RQ-3 - Existem ferramentas que ofereçam suporte às abordagens identificadas?

Dada a importância da automatização no desenvolvimento de sistemas de software com qualidade, essa questão de pesquisa tem o objetivo de identificar ferramentas relacionadas às abordagens da RQ-1. Dos estudos selecionados para extração de dados, sete utilizaram ferramentas para auxiliar a automatização de suas abordagens.

A abordagem proposta em S08 consiste em explorar a dinâmica de uma arquitetura de software para identificar interações entre os componentes do sistema e criar classes de testes para os comportamentos arquiteturais mais relevantes (MUCCINI; INVERARDI; BERTOLINO, 2004). Seus autores propõem modelar o comportamento arquitetural na linguagem LTS¹ (*Labeled Transition Systems*) e, a partir disso, derivar um conjunto adequado de abstrações do modelo

¹ <http://www.mcrl2.org/dev/user_manual/articles/lts.html>

LTS denominadas ALTSS (*Abstract Labeled Transition Systems*). No estudo de caso apresentado, os autores utilizam uma ferramenta denominada LTSA² (*Labeled Transition System Analyzer*) para derivar estados em linguagem LTS recebendo como entrada uma especificação em FSP³ (*Finite State Process*).

A abordagem proposta por S10 consiste em modelar unidades arquiteturais em diagramas de estado traduzidas para linguagem *Basic LOTOS* (*Language of Temporal Ordering Specification*) (SCOLLO; ZECCHINI, 2005). A partir de uma especificação descrita na linguagem *Basic LOTOS*, os autores utilizam a ferramenta TGV (*Test Generation using Verification techniques*) para gerar casos de teste e verificar a conformidade de uma unidade de arquitetura com as funcionalidades necessárias.

Na abordagem proposta por S11, os autores exploram como o teste de regressão pode ser aplicado em nível arquitetural. Os autores verificam se uma implementação ligeiramente modificada está em conformidade com a arquitetura inicial (MUCCINI; DIAS; RICHARDSON, 2006). No estudo de caso, um conjunto de ferramentas foi adotado em diferentes etapas. Inicialmente, os autores especificaram a arquitetura no estilo arquitetural C2 e, a partir da especificação, conduziram dois experimentos (MUCCINI; DIAS; RICHARDSON, 2006): um utilizando a ferramenta *FC2 Tools* para determinar um ATS (*Abstract Transition System*) e outro a ferramenta LTSA para derivar estados na linguagem LTS.

Na abordagem definida em S14, é analisado como uma especificação arquitetural baseada em modelos pode ser utilizada para implementar teste de regressão em nível arquitetural (MUCCINI, 2007). Para selecionar um conjunto de casos de teste a partir de uma especificação arquitetural, os autores utilizam um *plugin* para a ferramenta CHARMY⁴ (*CHecking ARchitectural Model consistencY*). Esse *plugin*, a partir de um diagrama de estado de componente, implementa a cobertura de vários critérios (por exemplo, todas arestas, todos componentes, todos estados e McCabe).

A abordagem proposta em S19 consiste em derivar uma sequência de teste a partir de uma arquitetura descrita em CHAM. Nela, os autores aplicam a ferramenta RDG (*Rule Dependence Graph*) para criar um grafo do sistema em teste. A partir desse grafo, um conjunto de regras de cobertura é gerado para determinar uma sequência de teste para ser aplicada no programa (LUN; DING, 2009).

A ferramenta Probo (CARACCILO, 2015) foi definida e utilizada no estudo S34 para permitir que usuários verifiquem regras personalizadas usando ferramentas de terceiros. Essa foi a única ferramenta utilizada nos estudos. Para viabilizar a condução do estudo S32 foram

² <<http://www.doc.ic.ac.uk/ltsa/>>

³ <<http://www.doc.ic.ac.uk/~jnm/LTSdocumentation/FSP-notation.html>>

⁴ <<http://www.di.univaq.it/charm/>>

utilizadas as ferramentas SystemDesk⁵, ECU-TEST⁶, DiabCompiler⁷, EB tresos Studio⁸.

A.3 Considerações Finais

Neste capítulo foi apresentado um resumo dos resultados obtidos de um mapeamento sistemático e de sua atualização. Os dois mapeamentos resultaram em 35 artigos selecionados para extração de dados. Os artigos foram publicados nos anos de 1996 até 2016, contabilizando vinte anos de pesquisas em que arquitetura de software e teste de software são relacionados. Apesar disso, ainda foi identificada a necessidade de amadurecimento da área, validações empíricas e ferramentas para automatização.

A arquitetura de software e o teste software têm sido relacionados por meio do teste baseado em arquitetura. As técnicas de teste utilizadas foram funcional, estrutural e “grey box”. As fases de teste identificadas foram aceitação, teste de carga, integração, regressão, sistema, unidade. Os artefatos de teste gerados a partir das abordagens foram, majoritariamente, casos de teste (65%), seguido por critérios de teste (29%), sendo que oráculos foram considerado apenas por um estudo. As linguagens referenciadas pelos estudos, em sua maioria, foram designadas para representar a arquitetura e apenas duas específicas para teste de software foram identificadas, que são a TDL e a TTCN-3.

⁵ <https://www.dspace.com/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm>

⁶ <<https://www.tracetronic.com/products/ecu-test/>>

⁷ <https://www.windriver.com/evaluations/diab_compiler/>

⁸ <<https://www.elektrobit.com/products/ecu/eb-tresos/studio/>>

FORMULÁRIO DE CONSENTIMENTO

B.1 Termo de Consentimento

Sua participação não é obrigatória e a qualquer momento você poderá desistir de participar e retirar seu consentimento. Toda informação coletada neste estudo é confidencial e seu nome não será identificado em momento algum. Da mesma forma, você se compromete a não comunicar os seus resultados enquanto não terminar o estudo, bem como manter sigilo dos documentos apresentados e que fazem parte do estudo.

Os benefícios que você receberá neste estudo são limitados ao aprendizado do material que é distribuído. Você é livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à sua pessoa não seja incluída no estudo. A participação nesse estudo de viabilidade deve ser de livre e espontânea vontade com o único intuito de contribuir para o avanço da pesquisa na área de Engenharia de Software.

Qualquer dúvida a respeito da pesquisa, você poderá entrar em contato com:

Nilton Mendes de Souza

Email: niltonmendes@usp.br

Concordo com tudo que foi citado anteriormente e livremente dou meu consentimento ao enviar este formulário preenchido.

Concorda com os termos?

☐ Sim

☐ Não

FORMULÁRIO DE CARACTERIZAÇÃO DE PERFIL

Este formulário tem como objetivo obter informações sobre o conhecimento de cada participante sobre os temas abordados.

C.1 Dados Pessoais

Nome completo:

Data de nascimento:

Sexo:

- ☐ Masculino
☐ Feminino

Qual seu nível de escolaridade?

- ☐ Graduação
☐ Mestrado Incompleto

- ☐ Mestrado Completo
- ☐ Doutorado Incompleto
- ☐ Doutorado Completo
- ☐ Pós-doutorado Incompleto
- ☐ Pós-doutorado Completo

Qual instituição em que você trabalha/estuda?

Em qual cargo você atua?

C.2 Conhecimento em Teste de Software

Como você considera a sua experiência em teste de software?

Iniciante 1 2 3 4 5 Especialista
☐ ☐ ☐ ☐ ☐

Há quanto tempo trabalha com teste de software na academia? (anos)

Há quanto tempo trabalha com teste de software na indústria? (anos)

Cursou alguma disciplina relacionada a teste de software?

- ☐ Sim
- ☐ Não

Você já teve algum tipo de experiência com o TMMi?

☐ Sim

☐ Não

Você já teve algum tipo de experiência com a norma IEEE/IEC/ISO 29119?

☐ Sim

☐ Não

Você já participou de um planejamento de teste?

☐ Sim

☐ Não

C.3 Conhecimento em Arquitetura de Software

Como você considera a sua experiência em teste de software?

	1	2	3	4	5	
Iniciante	☐	☐	☐	☐	☐	Especialista

Há quanto tempo trabalha com arquiteturas de software na academia? (anos)

Há quanto tempo trabalha com arquiteturas de software na indústria? (anos)

Cursou alguma disciplina relacionada a arquitetura de software?

☐ Sim

☐ Não

Você já participou de um projeto de arquitetura?

☐ Sim

☐ Não

C.4 Conhecimento em Arquitetura de Referência

Como você considera a sua experiência em arquitetura de referência?

Iniciante 1 2 3 4 5 Especialista
○ ○ ○ ○ ○

Há quanto tempo trabalha com arquiteturas de referência na academia? (anos)

Há quanto tempo trabalha com arquiteturas de referência na indústria? (anos)

Cursou alguma disciplina relacionada a arquitetura de referência?

- ☐ Sim
☐ Não

Você já participou do projeto de uma arquitetura de referência?

- ☐ Sim
☐ Não

C.5 Conhecimento sobre o RAModel

Você conhece o RAModel?

- ☐ Sim
☐ Não

Você já utilizou o RAModel?

- ☐ Sim
☐ Não

Você já utilizou o RAModel em quantos projetos?

- ☐ 0
☐ 1

- ☐ 2
- ☐ 3
- ☐ 4 ou mais

SÍNTESE DO RAMODEL_T

D.1 Introdução ao RAModel_{TI}

O RAModel é um modelo de referência para arquiteturas de referência que descreve elementos que devem estar contidos nesse tipo de arquitetura (NAKAGAWA; OQUENDO; BECKER, 2012). Esses elementos foram organizados nos grupos Domínio, Aplicação, Infraestrutura e Elementos Transversais. A partir desse modelo, foi proposta uma extensão com elementos de teste de software que deu origem ao RAModel_{TI} (*Reference Architecture Model with Testing Information*). O modelo estendido é apresentado na Figura 29, com as modificações indicadas por um retângulo tracejado.

Figura 29 – Visão geral do RAModel_{TI}.

Fonte: Elaborada pelo autor.

O procedimento para extensão teve como base as informações contidas no TMMi e na norma IEEE/IEC/ISO 29119. Esses documentos guiaram a identificação de elementos de teste e a organização dos elementos no modelo. A estrutura geral da organização dos elementos de teste no RAModel_{TI}, com grupos de teste, subgrupos de teste e elementos de teste, é apresentada na Figura 30.

Figura 30 – Estrutura dos elementos de teste no RAModel_{TI}

Fonte: Elaborada pelo autor.

A definição da nomenclatura adotada na organização dos elementos é apresentada a seguir:

- **Grupo de Teste:** agrupamento de subgrupos que possuem um objetivo em comum. Nesse caso, informações gerenciais, informações de infraestrutura necessária e projeto de teste;
- **Subgrupo de Teste:** estrutura de organização de elementos em um mesmo grupo;
- **Elemento de Teste:** tópico da área de teste de software que é relacionado ao planejamento de teste. A partir dele, poderão ser derivadas informações de teste;
- **Descrição:** uma descrição textual do elemento em si; e
- **Atributo:** item ou conjunto de itens que devem estar incluídos na documentação das informações de teste.

Os elementos importantes para o planejamento de teste de software que poderiam ser relacionados a arquiteturas de referência foram incluídos no RAModel_{TI}, como se pode observar na Figura 31. Nesta figura, uma visão geral dos elementos é apresentada.

Figura 31 – Estrutura geral do RAModel_{TI}.

Fonte: Elaborada pelo autor.

D.2 Elementos do RAModel_{TI}

Nesta seção, são apresentados os elementos de teste inseridos no RAModel. Esses elementos são apresentados nos grupos de teste de Gerenciamento de Teste, Infraestrutura de Teste e Projeto de Teste.

D.2.1 Grupo Gerenciamento de Teste

Nesse grupo de teste, estão os elementos de teste de nível gerencial. Dessa forma, as informações de teste de uma mesma organização ou múltiplas organizações que possuem os mesmos princípios são descritas. Os elementos desse grupo guiam a definição das informações dos demais, estabelecendo a política, o processo de teste a ser seguido, as diretrizes do domínio, o modo como poderá ser verificada se a atividade de teste está sendo conduzida com a qualidade esperada e quais os fundamentos para uma melhoria programada dessa atividade. Esse grupo é composto pelos subgrupos Política de Teste, Performance de Teste e Melhoria de Teste. Os subgrupos são descritos nos Quadros 28 – 31. Cada elemento recebeu um identificador com EG (Elemento de Gerenciamento) e um número inteiro, por exemplo “EG.1”.

Quadro 28 – Descrição do subgrupo de teste Política de Teste

EG.1 Objetivo de Teste
<p>Descrição Descreve a finalidade, as metas e o escopo geral dos testes dentro da organização. Indica a posição da organização sobre o fato de os testes serem realizados e o que eles procuram alcançar.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Objetivo Geral - Objetivos Específicos
EG.2 Processos de Teste
<p>Descrição Apresenta um ou mais processos de teste a serem executados na condução da atividade de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Processo Padrão de Teste - Subprocesso(s) de Teste
EG.3 Normas Regulamentares
<p>Descrição Apresenta as normas de teste que são mandatórias e as normas que são opcionais para que se cumpra o objetivo definido em EG.1 no domínio do sistema em teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Norma(s) mandatória(s) - Norma(s) opcional(is)

Quadro 29 – Descrição do subgrupo de teste Performance de Teste

EG.4 Objetivos para Performance
<p>Descrição Estabelece uma diretriz clara do que será medido e o que será comunicado em relação ao desempenho da atividade de teste que está sendo conduzida e da planejada. Esse elemento é baseado nas informações descritas em EG.1, EG.2 e EG.3.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto de objetivo(s) para performance

Quadro 30 – Descrição do subgrupo de teste Performance de Teste (Continuação do Quadro 29).

EG.5 Critérios de Aceitação
<p>Descrição Estabelece critérios para aceitar ou recusar se a performance apresentada pela atividade teste atingiu os objetivos apresentados em EG.4.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto do(s) critério(s)
EG.6 Indicadores de Performance
<p>Descrição Define os meios a serem utilizados para coletar e medir se o processo de teste está adequado em relação a EG.1 e EG.4. Indicadores podem ser definidos com base nas informações descritas em EG.1, EG.2, EG.3, EG.7 e EG.9.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto de Indicador(es)

Quadro 31 – Descrição do subgrupo de teste Melhoria de Teste

EG.7 Pontos Fortes e Pontos Fracos
<p>Descrição Apresenta um conjunto de pontos fortes e de pontos fracos do processo de teste (definido no elemento EG.2) em execução.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Ponto(s) Forte(s) - Ponto(s) Fraco(s)
EG.8 Conjunto de Melhorias
<p>Descrição Define um conjunto de melhorias que devem ser aplicadas ao processo de teste de software. Essas melhorias são relacionadas à maturidade do processo de teste definido em EG.2, em EG.7, e à política de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Melhoria(s) no processo de teste - Ponto(s) forte(s) relacionado(s) - Ponto(s) fraco(s) relacionado(s)
EG.9 Plano de Melhoria
<p>Descrição Estabelece o método para garantir a melhoria contínua do(s) processo(s) de teste. Esse elemento é definido com base nos elementos EG.1, EG.7, EG.8, e na experiência dos <i>stakeholders</i> acerca do sistema em teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto de melhorias - Critério(s) de priorização - Cronograma - Envolvido(s) - Ação(ões) - Risco(s) associado(s)

D.2.2 Grupo Infraestrutura de Teste

Esse grupo de teste é composto por dois subgrupos de teste para descrever a infraestrutura necessária para condução da atividade de teste. A infraestrutura, nesse grupo, constitui-se de itens fundamentais para a condução da atividade de teste, tais como simuladores, software ou hardware. Isso tem em vista descrever um ambiente adequado para condução da atividade. Dada a importância da automação na atividade de teste, um subgrupo de teste elenca as ferramentas de teste necessárias para definir um plano de automação. Os subgrupos de teste que compõem esse grupo são Ambiente de Teste e Automação de Teste. Eles são descritos nos Quadros 32 – 33, sendo que cada um dos elementos foi identificado com Elemento de Infraestrutura (EI).

Quadro 32 – Descrição do subgrupo de teste Ambiente de Teste.

El.1 Requisitos de Ambiente
<p>Descrição Define itens e condições para preparar e adequar o ambiente em que a atividade de teste será conduzida. Esses itens podem ser software, hardware, simuladores, definidos com base nas informações descritas em EG.2, EG.3.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Documentos relacionados - Itens mandatórios - Itens opcionais - Riscos associados - Requisitos de sistema associados - Requisitos de teste associados
El.2 Requisitos de Profissionais
<p>Descrição Estabelece quais profissionais podem compor o time de testadores para o sistema em teste. As habilidades, competências e formações também podem ser apresentadas. Essas informações são baseadas em EG.1, EG.2, EG.3.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Profissionais necessários - Certificações/formações necessárias - Papeis dos testadores

Quadro 33 – Descrição do subgrupo de teste Automatização de Teste.

El.3 Ferramentas de Teste
<p>Descrição Define ferramentas a serem utilizadas para gerenciar, executar e reportar a atividade de teste. Essas ferramentas são baseadas no conhecimento a respeito do sistema em teste e nas informações descritas em EG.1, EG.2, EG.3 e EG.4.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Ferramentas - Requisitos de teste associados - Condições de teste associadas

Quadro 34 – Descrição do subgrupo de teste Automatização de Teste (Continuação do Quadro 33).

EL.4 Abordagem para Automatização
<p>Descrição Define como as ferramentas descritas em EL.3 podem ser utilizadas e relacionadas às informações dos elementos EP.7, EP.8, EP.9, EP.10.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Procedimentos de teste a serem automatizados - Fases de teste - Cronograma - Envolvidos - Ferramentas de teste relacionadas - Requisitos de teste

D.2.3 Grupo Projeto de Teste

A partir da definição das informações gerenciais de teste e da infraestrutura necessária para que a atividade seja conduzida, definem-se informações concretas para projetos de teste específicos. As informações nesse grupo são definidas a partir de um gerenciamento dos defeitos no domínio e no contexto em que o sistema está sendo implementado. Riscos de projeto e de produto são listados com base em defeitos, riscos conhecidos, experiência de usuários e da literatura, buscando uma caracterização dos riscos associados a cada elemento do software que será desenvolvido. Por fim, o grupo Estratégia de Teste é definido com requisitos e condições de teste, técnicas de teste e casos de teste ordenados em procedimentos de teste. Esse grupo reúne os subgrupos Gerenciamento de Defeitos, Análise de Riscos e Estratégia de Teste. Eles são descritos nos Quadros 35 – 38, sendo que cada um dos elementos foi identificado com Elemento de Projeto de Teste (EP).

Quadro 35 – Descrição do subgrupo de teste Gerenciamento de Defeitos.

EP.1 Conjunto de Defeitos
<p>Descrição Conjunto dos principais defeitos relacionados ao sistema em teste. O conjunto dos defeitos é obtido por meio de defeitos comuns no desenvolvimento de software, defeitos comuns no domínio, defeitos de versões anteriores do sistema, defeitos obtidos da literatura.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Defeitos - Prioridade - Provável causa - Requisitos relacionados - Identificador

Quadro 36 – Descrição do subgrupo de teste Gerenciamento de Defeitos (Continuação do Quadro 35).

EP.2 Critérios de Priorização
<p>Descrição Define critérios para selecionar os principais defeitos para análise. Esses critérios podem ser definidos a partir de taxonomias, de normas (EG.3), da literatura, do objetivo (EG.1) de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição dos critérios - Defeitos associados
EP.3 Proposta de Solução
<p>Descrição Soluções são propostas para os defeitos listados. Essas soluções podem ser obtidas por meio de padrões (“<i>patterns</i>”), reúso de conhecimento e boas práticas.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Proposta de solução - Defeitos associados

Quadro 37 – Descrição do subgrupo de teste Análise de Riscos

EP.4 Lista de Riscos
<p>Descrição Lista os riscos de sistema e de arquitetura que podem estar relacionados ao sistema em teste. Os riscos podem ser obtidos por meio de taxonomias, ontologias, normas (descritas em EG.3), revisões e mapeamentos sistemáticos.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Riscos - Descrição - Prioridade - Impacto - Probabilidade - Identificador
EP.5 Critérios de Priorização
<p>Descrição Define critérios a serem utilizados para priorizar os riscos apresentados em EP.1. Essa informação é obtida a partir dos objetivos apresentados em EG.1, das normas (EG.3), do conhecimento de domínio e das experiência dos profissionais da equipe de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição dos critérios - Riscos associados
EP.6 Plano de Mitigação
<p>Descrição Define um plano no qual decisões serão tomadas para proteger, evitar, ou minimizar riscos em níveis especificados. Essa informação é baseada nos riscos elencados (EP.4), na literatura, na política de teste e nos critérios de priorização de riscos (EP.5).</p> <p>Atributos</p> <ul style="list-style-type: none"> - Risco - Cronograma - Ações - Requisitos de teste relacionados (Descritos em EP.7)

Quadro 38 – Descrição do subgrupo de teste Estratégia de Teste

EP.7 Requisitos de Teste
Descrição Define itens ou eventos de um componente ou sistema que devem ser testados. Esses requisitos são elencados a partir da análise do sistema, da política de teste, da análise de risco, gestão de defeitos e das técnicas.
Atributos <ul style="list-style-type: none">- Requisitos de teste- Defeitos associados- Riscos associados- Requisitos de sistema associados
EP.8 Condições de Teste
Descrição Define aspectos testáveis de um componente ou sistema, como uma função, transação, característica, como base para o teste. Essas condições podem ser obtidas a partir de requisitos de teste (EP.7), requisitos de sistema e capacidades do sistema.
Atributos <ul style="list-style-type: none">- Itens de teste- Condições
EP.9 Técnicas de Teste
Descrição São elencados técnicas e critérios de teste a serem utilizados para gerar casos de teste aplicáveis a cada item das condições de teste. Essas técnicas devem estar alinhadas com a política de teste.
Atributos <ul style="list-style-type: none">- Técnicas de teste- Requisitos associados- Condições associadas
EP.10 Procedimento de Teste
Descrição Define a execução de um ou mais casos de teste em ordem pré-definida e quaisquer ações associadas de pré e pós execução.
Atributos <ul style="list-style-type: none">- Descrição- Identificador- Caso de teste- Requisitos relacionados (Descritos em EP.7)- Condições relacionadas (Descritos em EP.8)- Ordem de execução

FORMULÁRIO DE AVALIAÇÃO

E.1 Objetivo do Estudo

O objetivo deste estudo é avaliar o $RAModel_{TI}$, uma extensão do $RAModel$ (NAKAGAWA; OQUENDO; BECKER, 2012) com elementos de teste de software por meio de um estudo de viabilidade. Uma síntese do $RAModel_{TI}$ é apresentado no Apêndice D. Nesse estudo de viabilidade, avalia-se se o objeto em estudo cumpre o objetivo para o qual ele foi criado (SHULL; CARVER; TRAVASSOS, 2001; MAFRA; BARCELOS; TRAVASSOS, 2006). As questões foram definidas para verificar se: i) o conjunto de elementos está completo; ii) existem elementos de teste que não são necessários; iii) a descrição de cada um dos elementos está correta; e iv) o conjunto de atributos está bem definido.

E.2 Grupo: Gerenciamento de Teste

E.2.1 Subgrupo: Política de Teste

Quadro 39 – Descrição do subgrupo Política de Teste

EG.1 Objetivos de Teste
Descrição Descreve a finalidade, as metas e o escopo geral dos testes dentro da organização. Indica a posição da organização sobre o fato de os testes serem realizados e o que eles procuram alcançar.
Atributos <ul style="list-style-type: none">- Objetivo Geral- Objetivos Específicos
EG.2 Processos de Teste
Descrição Apresenta um ou mais processos de teste a serem executados na condução da atividade de teste.
Atributos <ul style="list-style-type: none">- Processo Padrão de Teste- Sub-processo(s) de Teste
EG.3 Normas Regulamentares
Descrição Apresenta as normas de teste que são mandatórias e as normas que são opcionais para que se cumpra o objetivo definido em EG.1 no domínio do sistema em teste.
Atributos <ul style="list-style-type: none">- Norma(s) mandatória(s)- Norma(s) opcional(is)

1. O conjunto de elementos de teste do subgrupo Política de Teste está completo?

☐ Sim

☐ Não

1.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

--

2. Existem elementos de teste do subgrupo Política de Teste que não são necessários?

☐ Sim

☐ Não

2.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

3. A descrição de cada um dos elementos de teste do subgrupo Política de Teste está correta?

- ☐ Sim
- ☐ Não

3.1. Se não, por favor, sugira definições que poderiam ser incluídas ou excluídas.

4. O conjunto de atributos de cada elemento de teste do subgrupo Política de Teste está bem definido?

- ☐ Sim
- ☐ Não

4.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

E.2.2 Subgrupo: Performance de Teste

Quadro 40 – Descrição do subgrupo Performance de Teste

EG.4 Objetivos para Performance
<p>Descrição Estabelece uma diretriz clara do que será medido e o que será comunicado em relação ao desempenho da atividade de teste que está sendo conduzida e da planejada. Esse elemento é baseado nas informações descritas em EG.1, EG.2 e EG.3.</p> <p>Atributos - Conjunto de objetivo(s) para performance</p>
EG.5 Critérios de Aceitação
<p>Descrição Estabelece critérios para aceitar ou recusar se a performance apresentada pela atividade teste atingiu os objetivos apresentados em EG.4.</p> <p>Atributos - Conjunto do(s) critério(s)</p>
EG.6 Indicadores de Performance
<p>Descrição Define os meios a serem utilizados para coletar e medir se o processo de teste está adequado em relação a EG.1 e EG.4. Indicadores podem ser definidos com base nas informações descritas em EG.1, EG.2, EG.3, EG.7 e EG.9.</p> <p>Atributos - Conjunto de Indicador(es)</p>

5. O conjunto de elementos de teste do subgrupo Performance de Teste está completo?

☐ Sim

☐ Não

5.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

6. Existem elementos de teste do subgrupo Performance de Teste que não são necessários?

☐ Sim

☐ Não

6.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

7. A descrição de cada um dos elementos de teste do subgrupo Performance de Teste está correta?

- ☐ Sim
☐ Não

7.1. Se não, por favor, sugira melhorias que poderiam ser incluídas na(s) descrição(ões).

8. O conjunto de atributos de cada elemento de teste do subgrupo Performance de Teste está bem definido?

- ☐ Sim
☐ Não

8.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

E.2.3 Subgrupo: Melhoria de Teste

Quadro 41 – Descrição do subgrupo Melhoria de Teste

EG.7 Pontos Fortes e Pontos Fracos
<p>Descrição Apresenta um conjunto de pontos fortes e de pontos fracos do processo de teste (definido no elemento EG.2) em execução.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto de Ponto(s) Forte(s) - Conjunto de Ponto(s) Fraco(s)
EG.8 Conjunto de Melhorias
<p>Descrição Define um conjunto de melhorias que devem ser aplicadas ao processo de teste de software. Essas melhorias são relacionadas à maturidade do processo de teste definido em EG.2, em EG.7, e à política de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Melhoria(s) no processo de teste - Ponto(s) forte(s) relacionado(s) - Ponto(s) fraco(s) relacionado(s)
EG.9 Plano de Melhoria
<p>Descrição Estabelece o método para garantir a melhoria contínua do(s) processo(s) de teste. Esse elemento é definido com base nos elementos EG.1, EG.7, EG.8, e na experiência dos <i>stakeholders</i> acerca do sistema em teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Conjunto de melhorias - Critério(s) de priorização - Cronograma - Envolvido(s) - Ação(ões) - Risco(s) associado(s)

9. O conjunto de elementos de teste do subgrupo Melhoria de Teste está completo?

- ☐ Sim
- ☐ Não

9.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

10. Existem elementos de teste do subgrupo Melhoria de Teste que não são necessários?

- ☐ Sim
- ☐ Não

10.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

11. A descrição de cada um dos elementos de teste do subgrupo Melhoria de Teste está correta?

- ☐ Sim
☐ Não

11.1. Se não, por favor, sugira melhorias que poderiam ser incluídas na(s) descrição(ões).

12. O conjunto de atributos de cada elemento de teste do subgrupo Melhoria de Teste está bem definido?

- ☐ Sim
☐ Não

12.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

E.3 Grupo: Infraestrutura de Teste

E.3.1 Subgrupo: Ambiente de Teste

Quadro 42 – Descrição do subgrupo Ambiente de Teste.

EI.1 Requisitos de Ambiente
Descrição Define itens e condições para preparar e adequar o ambiente em que a atividade de teste será conduzida. Esses itens podem ser software, hardware, simuladores, definidos com base nas informações descritas em EG.2, EG.3.
Atributos <ul style="list-style-type: none">- Documentos relacionados- Itens mandatórios- Itens opcionais- Riscos associados- Requisitos de sistema associados- Requisitos de teste associados
EI.2 Requisitos de Profissionais
Descrição Estabelece quais profissionais podem compor o time de testadores para o sistema em teste. As habilidades, competências e formações também podem ser apresentadas. Essas informações são baseadas em EG.1, EG.2, EG.3.
Atributos <ul style="list-style-type: none">- Profissionais necessários- Certificações/formações necessárias- Papeis dos testadores

13. O conjunto de elementos de teste do subgrupo Ambiente de Teste está completo?

- ☐ Sim
☐ Não

13.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

14. Existem elementos de teste do subgrupo Ambiente de Teste que não são necessários?

- ☐ Sim
☐ Não

14.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

15. A descrição de cada um dos elementos de teste do subgrupo Ambiente de Teste está correta?

☐ Sim

☐ Não

15.1. Se não, por favor, sugira melhorias que poderiam ser incluídas na(s) descrição(ões).

16. O conjunto de atributos de cada elemento de teste do subgrupo Ambiente de Teste está bem definido?

☐ Sim

☐ Não

16.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

E.3.2 Subgrupo Automatização de Teste

Quadro 43 – Descrição do subgrupo Automatização de Teste.

EI.3 Ferramentas de Teste
Descrição Define ferramentas a serem utilizadas para gerenciar, executar e reportar a atividade de teste. Essas ferramentas são baseadas no conhecimento a respeito do sistema em teste e nas informações descritas em EG.1, EG.2, EG.3 e EG.4.
Atributos <ul style="list-style-type: none">- Ferramentas- Requisitos de teste associados- Condições de teste associadas
EI.4 Abordagem para Automatização
Descrição Define como as ferramentas descritas em EI.3 podem ser utilizadas e relacionadas às informações dos elementos EP.7, EP.8, EP.9, EP.10.
Atributos <ul style="list-style-type: none">- Procedimentos de teste a serem automatizados- Fases de teste- Cronograma- Envolvidos- Ferramentas de teste relacionadas- Requisitos de teste

17. O conjunto de elementos de teste do subgrupo Automatização de Teste está completo?

- ☐ Sim
☐ Não

17.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

18. Existem elementos de teste do subgrupo Automatização de Teste que não são necessários?

- ☐ Sim
☐ Não

18.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

19. A descrição de cada um dos elementos de teste do subgrupo Automaatização de Teste está correta?

- ☐ Sim
☐ Não

19.1. Se não, por favor, sugira melhorias que poderiam ser incluídas na(s) descrição(ões).

20. O conjunto de atributos de cada elemento de teste do subgrupo Automaatização de Teste está bem definido?

- ☐ Sim
☐ Não

20.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

E.4 Grupo: Projeto de Teste

E.4.1 Subgrupo: Gerenciamento de Defeitos

Quadro 44 – Descrição do subgrupo Gerenciamento de Defeitos.

EP.1 Conjunto de Defeitos
<p>Descrição Conjunto dos principais defeitos relacionados ao sistema em teste. O conjunto dos defeitos é obtido por meio de defeitos comuns no desenvolvimento de software, defeitos comuns no domínio, defeitos de versões anteriores do sistema, defeitos obtidos da literatura.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Defeitos - Prioridade - Provável causa - Requisitos relacionados - Identificador
EP.2 Critérios de Priorização
<p>Descrição Define critérios para selecionar os principais defeitos para análise. Esses critérios podem ser definidos a partir de taxonomias, de normas (EG.3), da literatura, do objetivo (EG.1) de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição dos critérios - Defeitos associados
EP.3 Proposta de Solução
<p>Descrição Soluções são propostas para os defeitos listados. Essas soluções podem ser obtidas por meio de padrões (“<i>patterns</i>”), reuso de conhecimento e boas práticas.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Proposta de solução - Defeitos associados

21. O conjunto de elementos de teste do subgrupo Gerenciamento de Defeitos está completo?

- ☐ Sim
- ☐ Não

21.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

22. Existem elementos de teste do subgrupo Gerenciamento de Defeitos que não são necessários?

- ☐ Sim
- ☐ Não

22.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

--

23. A descrição de cada um dos elementos de teste do subgrupo Gerenciamento de Defeitos está correta?

- ☐ Sim
☐ Não

23.1. Se não, por favor, sugira melhorias que poderiam ser incluídas na(s) descrição(ões).

--

24. O conjunto de atributos de cada elemento de teste do subgrupo Gerenciamento de Defeitos está bem definido?

- ☐ Sim
☐ Não

24.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

--

E.4.2 Subgrupo: Análise de Riscos

Quadro 45 – Descrição do subgrupo Análise de Riscos

EP.4 Lista de Riscos
<p>Descrição Lista os riscos de sistema e de arquitetura que podem estar relacionados ao sistema em teste. Os riscos podem ser obtidos por meio de taxonomias, ontologias, normas (descritas em EG.3), revisões e mapeamentos sistemáticos.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Riscos - Descrição - Prioridade - Impacto - Probabilidade - Identificador
EP.5 Critérios de Priorização
<p>Descrição Define critérios a serem utilizados para priorizar os riscos apresentados em EP.1. Essa informação é obtida a partir dos objetivos apresentados em EG.1, das normas (EG.3), do conhecimento de domínio e das experiência dos profissionais da equipe de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição dos critérios - Riscos associados
EP.6 Plano de Mitigação
<p>Descrição Define um plano no qual decisões serão tomadas para proteger, evitar, ou minimizar riscos em níveis especificados. Essa informação é baseada nos riscos elencados (EP.4), na literatura, na política de teste e nos critérios de priorização de riscos (EP.5).</p> <p>Atributos</p> <ul style="list-style-type: none"> - Risco - Cronograma - Ações - Requisitos de teste relacionados (Descritos em EP.7)

25. O conjunto de elementos de teste do subgrupo Análise de Riscos está completo?

- ☐ Sim
- ☐ Não

25.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

26. Existem elementos de teste do subgrupo Análise de Riscos que não são necessários?

- ☐ Sim

☐ Não

26.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

27. A descrição de cada um dos elementos de teste do subgrupo Análise de Riscos está correta?

☐ Sim

☐ Não

27.1. Se não, por favor, sugira melhorias que poderiam ser incluídas na(s) descrição(ões).

28. O conjunto de atributos de cada elemento de teste do subgrupo Análise de Riscos está bem definido?

☐ Sim

☐ Não

28.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.

E.4.3 Subgrupo: Estratégia de Teste

Quadro 46 – Descrição do subgrupo Estratégia de Teste

EP.7 Requisitos de Teste
<p>Descrição Define aspectos testáveis de um componente ou sistema, como uma função, transação, característica, como base para o teste. Essas condições podem ser obtidas a partir de requisitos de teste (EP.7), requisitos de sistema e capacidades do sistema.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Requisitos de teste - Defeitos associados - Riscos associados - Requisitos de sistema associados
EP.8 Condições de Teste
<p>Descrição Define aspectos testáveis de um componente ou sistema, como uma função, transação, característica, como base para o teste. Essas condições podem ser obtidas a partir de requisitos de teste (EP.7) e de requisitos e capacidades do sistema.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Itens de teste - Condições
EP.9 Técnicas de Teste
<p>Descrição São elencados técnicas e critérios de teste a serem utilizados utilizadas para gerar casos de teste aplicáveis a cada item das condições de teste. Essas técnicas devem estar alinhadas com a política de teste.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Técnicas de teste - Requisitos associados - Condições associadas
EP.10 Procedimento de Teste
<p>Descrição Define a execução de um ou mais casos de teste em ordem pré-definida e quaisquer ações associadas de pré e pós execução.</p> <p>Atributos</p> <ul style="list-style-type: none"> - Descrição - Identificador - Caso de teste - Requisitos relacionados (Descritos em EP.7) - Condições relacionadas (Descritos em EP.8) - Ordem de execução

29. O conjunto de elementos de teste do subgrupo Estratégia de Teste está completo?

- ☐ Sim
- ☐ Não

29.1. Se não, por favor, descreva os elementos que poderiam ser inseridos nesse subgrupo.

30. Existem elementos de teste do subgrupo Estratégia de Teste que não são necessários?

☐ Sim

☐ Não

30.1. Se sim, por favor, descreva os elementos que poderiam ser excluídos desse subgrupo.

31. A descrição de cada um dos elementos de teste do subgrupo Estratégia de Teste está correta?

☐ Sim

☐ Não

31.1. Se não, por favor, sugira melhorias que poderiam ser incluídas na(s) descrição(ões).

32. O conjunto de atributos de cada elemento de teste do subgrupo Estratégia de Teste está bem definido?

☐ Sim

☐ Não

32.1. Se não, por favor, sugira atributos que poderiam ser incluídos ou excluídos.