SANDRA KAWAMOTO

**SCRUM-DR: UMA EXTENSÃO DO FRAMEWORK SCRUM
ADERENTE AO CMMI-DEV UTILIZANDO TÉCNICAS DE
DESIGN RATIONALE**

São Paulo
2017

SANDRA KAWAMOTO

SCRUM-DR: UMA EXTENSÃO DO FRAMEWORK SCRUM ADERENTE
AO CMMI-DEV UTILIZANDO TÉCNICAS DE DESIGN RATIONALE

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção
do Título de Doutor em Ciências

São Paulo
2017

SANDRA KAWAMOTO

SCRUM-DR: UMA EXTENSÃO DO FRAMEWORK SCRUM ADERENTE
AO CMMI-DEV UTILIZANDO TÉCNICAS DE DESIGN RATIONALE

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção
do Título de Doutor em Ciências

Área de concentração: Engenharia de
Computação

Orientador: Prof. Dr. Jorge Rady de
Almeida Júnior

São Paulo
2017

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catálogo-na-publicação

Kawamoto, Sandra

Scrum-DR: Uma extensão do framework Scrum aderente ao CMMI-DEV utilizando técnicas de Design Rationale / S. Kawamoto – versão corr. – São Paulo, 2017.

174 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Métodos ágeis 2.Qualidade de software 3.Desenvolvimento de software 4.Design Rationale 5.Scrum I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Dedico este trabalho aos meus pais Cecília e Luiz.

AGRADECIMENTOS

Ao Prof. Dr. Jorge Rady pela orientação, sugestões e paciência. Por estar sempre disponível a ajudar e resolver os dilemas que apareceram ao longo desta pesquisa. Pela sua generosidade e serenidade.

Ao Prof. Dr. João José Neto e Profa. Dra. Liria Matsumoto Sato pelas contribuições no exame de qualificação.

Aos colegas da pós-graduação pela convivência e troca de experiências durante as aulas.

Agradecimento especial a Daniela Vieira, Alexandre dos Santos, Leandro Duran, Thays Tagliaferri de Grazia, Adilson Watanabe, Fernando Alves Bernardes e Emanuel Iglesias pelas informações valiosas sobre o desenvolvimento de software nas empresas.

À Ana Maria Badiali pelo auxílio na formatação da tese e instruções sobre pesquisa.

Agradeço ao meu marido Fabiano pelo apoio para que este trabalho fosse concluído, não só pelo incentivo e paciência, mas por todas as ricas discussões sobre o assunto desta pesquisa.

Aos meus filhos Matheus e Luísa por compreenderem a minha ausência e colaborarem para que eu pudesse realizar este doutorado. Minhas riquezas!

Aos meus pais queridos que nunca mediram esforços para facilitar o meu caminho nos estudos. Pelo amor incondicional e incentivo. Por mostrarem com seus próprios exemplos o caminho do bem.

A todos da minha família pela compreensão e incentivo para que este trabalho fosse finalizado.

À USP pela oportunidade e ao CNPq pelo apoio financeiro a esta pesquisa.

A todos que de alguma forma contribuíram com a evolução deste trabalho e colaboraram para a sua conclusão.

Tenho a impressão de ter sido uma criança brincando à beira-mar, divertindo-me em descobrir a pedrinha mais lisa ou uma concha mais bonita que as outras, enquanto o imenso oceano da verdade continua misterioso diante de meus olhos.

(Isaac Newton)

RESUMO

Os métodos ágeis introduziram uma nova abordagem de desenvolvimento de software, enfatizando os indivíduos, o produto e a comunicação. Prometem alta satisfação do cliente, baixo índice de erros, tempo de desenvolvimento mais rápido e uma solução para a rápida mudança de requisitos. Por outro lado, modelos de qualidade de software, como o CMMI, prometem previsibilidade, estabilidade e segurança, melhorando a qualidade dos produtos e dos processos da empresa. O intuito desta pesquisa é unir estas duas abordagens na tentativa de aproveitar as vantagens de ambas. Como resultado, apresenta-se o *framework* Scrum-DR, uma extensão do Scrum, aderente às diretrizes do CMMI-DEV na categoria Suporte. O Scrum-DR possui características do desenvolvimento ágil de software e contempla a utilização de técnicas de *Design Rationale*, apresentando um modelo que soluciona uma importante questão nas metodologias ágeis.

Palavras-chave: Métodos Ágeis. CMMI. Design Rationale. Scrum. Scrum-DR.

ABSTRACT

Agile methods introduced a new approach to software development, emphasizing individuals, product, and communication. They promise high customer satisfaction, low error rates, faster development time and a solution for rapidly changing requirements. On the other hand, software quality models, such as CMMI, promise predictability, stability and security, improving the quality of the company's products and processes. The purpose of this research is to link these two approaches in an attempt to take advantage of both. As a result, the Scrum-DR *framework*, a Scrum extension, adhering to the CMMI-DEV guidelines in the Support category is presented. Scrum-DR has characteristics of agile software development and contemplates the use of Design Rationale techniques, presenting a model that solves an important issue in agile methodologies.

Keywords: Agile Methods. CMMI. Design Rationale. Scrum. Scrum-DR.

LISTA DE ILUSTRAÇÕES

FIGURA 1 – NÚMERO DE CERTIFICAÇÕES CMMI-DEV 1.3 NO BRASIL.....	14
FIGURA 2 – NÚMERO DE CERTIFICAÇÕES CMMI-DEV 1.3 NOS ESTADOS UNIDOS	14
FIGURA 3 - CARACTERÍSTICAS DOS PROCESSOS ÁGEIS.....	22
FIGURA 4 – FEATURE-DRIVEN DEVELOPMENT – DUAS FASES, CINCO PROCESSOS	27
FIGURA 5 – PAPÉIS NO SCRUM E SUAS ATIVIDADES.....	28
FIGURA 6 - FLUXO DO MÉTODO SCRUM	28
FIGURA 7 - REPRESENTAÇÃO DO <i>DR</i> – DE FORMAL A INFORMAL.....	55
FIGURA 8 - IBIS: REPRESENTAÇÃO DOS NÓS	62
FIGURA 9 - MODELO CONCEITUAL DO REMAP.....	65
FIGURA 10 - MODELO INQUIRY CYCLE	66
FIGURA 11 – NÍVEIS DE MATURIDADE DO AGILEQA-RM.....	77
FIGURA 12 – AGILE MATURITY MODEL (AMM)	78
FIGURA 13 – ÁREA DE ATUAÇÃO DOS ENTREVISTADOS.....	92
FIGURA 14 – FASES DO SCRUM-DR.....	103
FIGURA 15 – PAPÉIS NO SCRUM-DR	104
FIGURA 16 – ELEMENTOS DO DOC-DR.....	110
FIGURA 17 – VISÃO GERAL DO DOC-DR	111
FIGURA 18 – SPRINT NO SCRUM-DR	114
FIGURA 19 – REUNIÃO DE DEFINIÇÃO.....	114
FIGURA 20 – REUNIÃO DE RETROSPECTIVA DO PROJETO.....	118
FIGURA 21 – VISÃO GERAL DO SCRUM-DR.....	119
FIGURA 22 – SGH: VISÃO GERAL.....	126
FIGURA 23 – SGH: DR SOBRE PLATAFORMA DE DESENVOLVIMENTO	132
FIGURA 24 – SGH: IMPLEMENTAÇÃO DAS FUNCIONALIDADES PRINCIPAIS	133
FIGURA 25 – SGH: DR SOBRE <i>LOG IN</i> NO SISTEMA	135

LISTA DE TABELAS

TABELA 1 - PRÁTICAS DO XP.....	26
TABELA 2 - ÁREAS DE PROCESSO, SUAS CATEGORIAS E NÍVEIS DE MATURIDADE.....	37
TABELA 3 - QUADRO COMPARATIVO DOS MÉTODOS DE CAPTURA DO <i>DR</i>	59
TABELA 4 - NFR FRAMEWORK – ELEMENTOS PRINCIPAIS.....	71
TABELA 5 – CLASSIFICAÇÃO MÉTODOS ÁGEIS X CMMI.....	79
TABELA 6 - ADERÊNCIA DAS PRÁTICAS ÁGEIS ÀS ÁREAS DE PROCESSO DO CMMI.....	88
TABELA 7 - PLANO DO PROJETO.....	106
TABELA 8 - BACKLOG DO PRODUTO	108
TABELA 9 - EXEMPLO DE UM BACKLOG DO PRODUTO NO SCRUM-DR	108
TABELA 10 - PLANO DE MELHORIA DA SPRINT.....	112
TABELA 11 - PLANO DE MELHORIA	112
TABELA 12 - ADERÊNCIA CMMI X MÉTODOS ÁGEIS X SCRUM-DR	125
TABELA 13 – RESERVAR QUARTO NO SGH.....	128
TABELA 14 – REQUISITOS NÃO FUNCIONAIS DO SGH.....	129
TABELA 15 - PLANO DO PROJETO DO SGH.....	130
TABELA 16 – INICIAÇÃO: BACKLOG DO PRODUTO.....	131
TABELA 17 – ITENS IMPLEMENTADOS NA SPRINT 1.....	133
TABELA 18 - PLANO DE MELHORIA DA SPRINT.....	134
TABELA 19 – SPRINT 2: INCLUSÕES AO BACKLOG DO PRODUTO.....	136
TABELA 20 – ITENS IMPLEMENTADOS NA SPRINT 2.....	137
TABELA 21 – ITENS IMPLEMENTADOS NA SPRINT 3.....	138
TABELA 22 – SPRINT 3: INCLUSÃO AO BACKLOG DO PRODUTO.....	138
TABELA 23 – ITENS IMPLEMENTADOS NA SPRINT 4.....	139
TABELA 24 - PLANO DE MELHORIA	140

LISTA DE ABREVIATURAS E SIGLAS

ASD	<i>Adaptive Software Development</i> (Desenvolvimento de Software Adaptativo)
CAR	<i>Causal Analysis and Resolution</i> (Análise e Resolução de Causas)
CM	<i>Configuration Management</i> (Gestão de Configuração)
CMM	<i>Capability Maturity Model</i> (Modelo de Maturidade e de Capacidade)
CMMI	<i>Capability Maturity Model Integration</i> (Modelo Integrado de Maturidade e de Capacidade)
CMMI-DEV	<i>CMMI for Development</i> (CMMI para Desenvolvimento)
CQ	Controle de qualidade
DAR	<i>Decision Analysis and Resolution</i> (Análise e Tomada de Decisões)
DR	<i>Design Rationale</i>
DRL	<i>Decision Representation Language</i>
FDD	<i>Feature-Driven Development</i>
gIBIS	<i>Graphical Issue Based Information System</i>
GQ	Garantia de qualidade
IBIS	<i>Issue Based Information System</i>
IPM	<i>Integrated Project Management</i> (Gestão Integrada de Projeto)
MA	<i>Measurement and Analysis</i> (Medição e Análise)
OPD	<i>Organizational Process Definition</i> (Definição dos Processos da Organização)
OPF	<i>Organizational Process Focus</i> (Foco nos Processos da Organização)
OPM	<i>Organizational Performance Management</i> (Gestão do Desempenho da Organização)
OPP	<i>Organizational Process Performance</i> (Desempenho dos Processos da Organização)
OT	<i>Organizational Training</i> (Treinamento na Organização)
PI	<i>Product Integration</i> (Integração de Produto)

PMC	<i>Project Monitoring and Control</i> (Monitoramento e Controle de Projeto)
PP	<i>Project Planning</i> (Planejamento de Projeto)
PPQA	<i>Process and Product Quality Assurance</i> (Garantia da Qualidade de Processo e Produto)
QOC	<i>Questions Options and Criteria</i>
QPM	<i>Quantitative Project Management</i> (Gestão Quantitativa de Projeto)
RD	<i>Requirements Development</i> (Desenvolvimento de Requisitos)
REMAP	<i>REpresentation and MAintenance of Process knowledge</i>
REQM	<i>Requirements Management</i> (Gestão de Requisitos)
RSKM	<i>Risk Management</i> (Gestão de Riscos)
RUP	<i>Rational Unified Process</i>
SAM	<i>Supplier Agreement Management</i> (Gestão de Contrato com Fornecedores)
SCRAM	<i>Scenario Based Requirements Analysis Method</i>
SEI	<i>Software Engineering Institute</i>
SEURAT	<i>Software Engineering Using RATionale</i>
TDD	<i>Test-Driven Development</i>
TS	<i>Technical Solution</i> (Solução Técnica)
VAL	<i>Validation</i> (Validação)
VER	<i>Verification</i> (Verificação)
XP	<i>Extreme Programming</i> (Programação Extrema)

SUMÁRIO

1. INTRODUÇÃO.....	13
1.1. Motivação e justificativa.....	13
1.2. Objetivo	16
1.3. Metodologia	16
1.4. Organização do trabalho	17
2. CONTEXTO DO ESTUDO	19
2.1. Métodos Ágeis	19
2.1.1. Manifesto Ágil	20
2.1.2. Características Principais	21
2.1.3. Principais Métodos Ágeis	23
2.1.3.1. Desenvolvimento de Software Adaptativo	24
2.1.3.2. Crystal	24
2.1.3.3. Programação Extrema (XP)	25
2.1.3.4. Feature-Driven Development (FDD)	26
2.1.3.5. Scrum	27
2.1.4. Desafios e Limitações	29
2.1.5. Considerações finais	30
2.2. Qualidade de software - CMMI	31
2.2.1. Definições	31
2.2.2. CMMI	34
2.2.3. Áreas de Processo CMMI	35
2.2.4. Desafios e Limitações	49
2.2.5. Considerações finais	50
2.3. Design Rationale (DR)	50
2.3.1. Definição	51
2.3.2. Formas de obtenção do DR	52
2.3.3. Utilidade	53
2.3.4. Representação das informações	55
2.3.4.1. Representação Formal	55
2.3.4.2. Representação Informal	56
2.3.4.3. Representação Semiformal	56
2.3.5. Captura das informações	57
2.3.6. Uso das informações	59
2.3.7. Ferramentas de suporte	60
2.3.8. Abordagens	61
2.3.9. Modelos de representação	63
2.3.9.1. REMAP	64
2.3.9.2. Inquiry Cycle	65
2.3.9.3. SCRAM	67
2.3.9.4. SEURAT	68
2.3.9.5. Sysiphus	69
2.3.9.6. NFR Framework	70
2.3.10. Desafios e limitações	72
2.3.11. Considerações finais	73
3. MÉTODOS ÁGEIS E CMMI: ANÁLISE	75
3.1. Trabalhos correlatos	75
3.2. Compatibilidade dos Métodos Ágeis com CMMI	79
3.2.1. Gerenciamento de Processo	80
3.2.2. Gerenciamento de Projeto	81

3.2.3.	Engenharia	84
3.2.4.	Suporte.....	86
3.2.5.	Conclusões	87
3.3.	Entrevista.....	90
3.3.1.	Metodologia	91
3.3.2.	Informações coletadas	92
3.3.3.	Análise dos resultados	98
3.4.	Considerações finais.....	99
4.	SCRUM-DR: UMA EXTENSÃO DO FRAMEWORK SCRUM ADERENTE AO CMMI-DEV UTILIZANDO TÉCNICAS DE DESIGN RATIONALE	101
4.1.	Framework Scrum-DR	101
4.1.1.	Fases	102
4.1.2.	Papéis.....	104
4.1.3.	Artefatos	105
4.1.3.1.	Plano do Projeto	106
4.1.3.2.	Backlog do Produto.....	107
4.1.3.3.	Backlog da Sprint.....	109
4.1.3.4.	DOC-DR	109
4.1.3.5.	Incremento	111
4.1.3.6.	Plano de Melhoria da Sprint	111
4.1.3.7.	Plano de Melhoria	112
4.1.4.	Eventos	113
4.1.4.1.	Reunião de Definição	114
4.1.4.2.	Reunião de Planejamento da Sprint.....	115
4.1.4.3.	Implementação e Reunião Diária	115
4.1.4.4.	Revisão da Sprint	116
4.1.4.5.	Retrospectiva da Sprint	117
4.1.4.6.	Retrospectiva do Projeto	117
4.1.5.	Visão geral	119
4.2.	Análise e Aplicação do Scrum-DR	120
4.2.1.	Scrum-DR X CMMI-DEV	121
4.2.2.	Estudo de caso.....	126
4.2.2.1.	Iniciação	127
4.2.2.2.	Desenvolvimento	131
4.2.2.3.	Finalização	139
4.3.	Considerações finais.....	140
5.	CONCLUSÕES.....	142
5.1.	Principais contribuições.....	143
5.2.	Dificuldades e limitações.....	144
5.3.	Trabalhos Futuros	144
	REFERÊNCIAS	146
	APÊNDICE A – Questionário utilizado na entrevista	160
	APÊNDICE B – Plano de Projeto do SGH utilizando Scrum-DR.....	161
	APÊNDICE C – Backlog do Produto do SGH utilizando Scrum-DR.....	163
	APÊNDICE D – DOC-DR do Projeto do SGH utilizando Scrum-DR.....	164
	APÊNDICE E – Backlogs da Sprint do Projeto do SGH utilizando Scrum-DR	165
	ANEXO – Metas e práticas específicas das áreas de processo do CMMI-DEV 1.3	168

1. INTRODUÇÃO

Indicadores de maturidade organizacional, como o *Capability Maturity Model Integration* (CMMI), têm se tornado cada vez mais importantes para o desenvolvimento de software (FRITZSCHE; KEIL, 2007). Clientes e organizações que desenvolvem projetos distribuídos geralmente levam esses indicadores em consideração na escolha de fornecedores, pois estas avaliações e auditorias servem como um sinal de maturidade do processo.

Ao mesmo tempo, a utilização de métodos ágeis continua ganhando espaço. Sua utilização é também encontrada em projetos grandes (COCKBURN; HIGHSMITH, 2001) e também projetos terceirizados e distribuídos (FOWLER, 2003; SANGWAN; MASTICOLA, 2004).

O grande desafio é descobrir se essas ideias, a princípio antagônicas, podem conviver harmoniosamente e quais as lacunas as separam. Além disso, qual a melhor maneira de propor um *framework* unindo ideias dos métodos ágeis e do CMMI de maneira a se beneficiar das vantagens de ambas.

Este capítulo apresenta os motivos e justificativas de escolha do assunto de pesquisa, quais os objetivos deste trabalho, a metodologia utilizada e sua estrutura.

1.1. *Motivação e justificativa*

Segundo CMMI-DEV V1.3 (2010), um processo de software bem estabelecido, compreendido e controlado auxilia os desenvolvedores a obter um produto final com qualidade. Com o objetivo de melhorar tanto a qualidade de seus produtos de software quanto à produtividade do desenvolvimento de software, cada vez mais as empresas estão percebendo a importância da existência de um processo de software formalizado na empresa e investindo na melhoria deste processo (PINO et al., 2008).

Em 2015, a adoção do CMMI aumentou 17% globalmente e 28% só nos Estados Unidos (CMMI INSTITUTE, 2017). Aproximadamente 70% das organizações que obtiveram a certificação CMMI nesse mesmo ano reportaram utilizar uma ou mais práticas ágeis.

A Figura 1 e a Figura 2 mostram, respectivamente, o número de certificações CMMI for Development (CMMI-DEV) versão 1.3 no Brasil e nos Estados Unidos nos últimos anos. Pode-se verificar que no Brasil os números praticamente se mantiveram nos últimos anos e nos Estados Unidos, o número de empresas certificadas cresceu a cada ano. Portanto, pesquisa sobre as diretrizes CMMI continuam importantes.

Figura 1 – Número de certificações CMMI-DEV 1.3 no Brasil

Fonte: Adaptado de CMMI INSTITUTE (2017)

Figura 2 – Número de certificações CMMI-DEV 1.3 nos Estados Unidos

Fonte: Adaptado de CMMI INSTITUTE (2017)

Metodologias ágeis prometem alta satisfação do cliente, baixo índice de erros, tempo de desenvolvimento mais rápido e uma solução para a rápida mudança de requisitos. Métodos baseados no Modelo de Maturidade e de Capacidade (CMM) prometem previsibilidade, estabilidade e segurança e é mais voltado a metodologias tradicionais de desenvolvimento de software. Entretanto, tanto as abordagens ágeis como as planejadas possuem deficiências dependentes da situação, que se não tratadas, podem levar ao fracasso do projeto (BOEHM; TURNER, 2003(a)).

Verifica-se que tanto as metodologias ágeis como as mais tradicionais fornecem diretrizes úteis para o desenvolvimento de software. Silva et al. (2015) defende o uso combinado de práticas provenientes da teoria do gerenciamento ágil e tradicional, em um mesmo projeto, com o objetivo de melhorar o desempenho em coesão, comunicação, satisfação do cliente, tempo e flexibilidade, mantendo, porém, um grau mais elevado de controle do projeto.

De maneira semelhante, o uso simultâneo de metodologias ágeis e CMMI, quando realizado de maneira adequada e de acordo com as características dos projetos, equipe e organização, pode ser muito útil para o desenvolvimento de software. Através da análise de suas características e lacunas, é possível conciliar as duas abordagens, obtendo proveito de ambas.

A importância de *Design Rationale* (DR) tem sido reconhecida desde a década de 1980 (PARNAS; CLEMENTS, 1986; POTTS; BRUNS, 1988). A partir dessa época, muitas pesquisas começaram a ser realizadas com o objetivo de aplicar a técnica em diversos ambientes e tipos de projetos (FISHER et al., 1995; PAIVA, 2007; KAWAMOTO; ALMEIDA, 2009). Entretanto, na prática não houve uma grande adesão (BOSCH, 2004). É difícil definir abordagens e sistemas que possam ser utilizados com sucesso em projetos reais (DUTOIT; PAECH, 2001). Uma abordagem tem mais chances de ser aplicada se possuir a definição de seu processo de utilização, apresentar ferramentas que a suportem e casos de uso de sucesso. A introdução do DR nas metodologias ágeis supre importantes deficiências diminuindo a lacuna entre metodologias ágeis e CMMI.

Existem alguns trabalhos que englobam o assunto metodologia ágil e CMMI. A maior parte tem o foco na categoria Processo do CMMI-DEV (MARÇAL, 2009; LINA; DAN, 2012).

Por ser uma categoria pouco explorada e também por verificar que há uma deficiência nas metodologias ágeis englobando essas áreas, optou-se por focar nas áreas de processo da

categoria Suporte do CMMI-DEV 1.3. É claro que por causa da intensa ligação entre as diversas categorias, é realizado um estudo nas diretrizes CMMI como um todo.

Esta pesquisa é relevante por englobar assuntos importantes e diversos dentro da engenharia de software que se tratados juntos, podem trazer ainda mais benefícios. Nesse sentido, as diretrizes do CMMI e as técnicas de DR têm muito a contribuir para a metodologia ágil de desenvolvimento de software, o que motivou a esta pesquisa.

1.2. Objetivo

Primeiramente, o objetivo deste trabalho é realizar uma análise de aderência dos modelos ágeis às práticas do modelo CMMI-DEV. O intuito é identificar as lacunas existentes entre as duas abordagens. Para cada área de processo do CMMI-DEV é verificado se o modelo de desenvolvimento ágil apresenta práticas ou diretrizes para a sua aplicação.

Definidos os pontos não suportados ou parcialmente suportados pela metodologia ágil, é então definido um *framework* de desenvolvimento ágil aderente às práticas de qualidade do CMMI. O modelo proposto utiliza técnicas de DR, agregando valor à proposta e preenchendo uma lacuna importante não tratada pelas metodologias ágeis. A definição deste *framework* é o objetivo principal deste trabalho.

O *framework* proposto define um modelo de desenvolvimento ágil mais completo, incluindo técnicas que auxiliam a documentação, tomada de decisão e disseminação do conhecimento. Pode ser útil para equipes ágeis que desejam obter a certificação CMMI ou que simplesmente desejam ter melhoria do processo e da qualidade do produto. De maneira semelhante, o *framework* pode auxiliar equipes com certificação CMMI a migrar para um desenvolvimento ágil de software.

1.3. Metodologia

Inicialmente, foi realizada uma pesquisa bibliográfica sobre *rationale*, focando nos trabalhos mais recentes, pois este assunto já havia sido pesquisado anteriormente. Passou-se então ao

estudo das metodologias de desenvolvimento de software, em especial, as metodologias ágeis, suas características e limitações. Finalmente, o foco foi qualidade de software, enfatizando o modelo CMMI. Foram descritos os principais objetivos das áreas de processo do CMMI.

Em seguida, foi realizada uma análise de aderência dos métodos ágeis segundo as diretrizes do modelo de qualidade CMMI, definindo quais as áreas de processo não eram atendidas. Este estudo foi realizado baseado na pesquisa bibliográfica e em trabalhos anteriores.

A fim de obter um panorama do desenvolvimento ágil na prática, foi realizada uma pesquisa em campo através de uma entrevista em formato semiestruturado. A pesquisa foi direcionada a profissionais de empresas de software que utilizam desenvolvimento ágil.

Com este embasamento teórico e prático, passou-se à definição do *framework* de desenvolvimento de software que satisfizesse as áreas de processo do CMMI na categoria Suporte, mas que mantivesse as características de um modelo ágil. Este *framework* é uma extensão do *framework* Scrum.

Finalmente, foi realizado um estudo de caso a fim de ilustrar a aplicação do *framework* proposto.

1.4. Organização do trabalho

Esta tese é composta por 5 capítulos.

O capítulo 2 apresenta os principais conceitos relacionados aos métodos ágeis, qualidade de software e *Design Rationale*. Estes são os assuntos principais tratados neste trabalho.

O capítulo 3 apresenta uma análise sobre a utilização conjunta de métodos ágeis e CMMI. Inicialmente é feita uma pesquisa bibliográfica sobre trabalhos corretos. Em seguida, analisa-se a aderência dos métodos ágeis com o CMMI, identificando as lacunas. Finalmente, descreve-se uma entrevista realizada com profissionais de empresas.

O capítulo 4 contém a contribuição principal deste trabalho. Nele é descrito o *framework* Scrum-DR, com características ágeis e aderentes ao CMMI. É realizada também uma análise

de aderência do Scrum-DR com o CMMI e um estudo de caso aplicando o *framework* proposto.

Finalmente, o capítulo 5 apresenta as conclusões finais desta tese, incluindo as principais contribuições, as dificuldades e limitações da pesquisa, bem como possíveis trabalhos a serem desenvolvidos no futuro, como consequência dos estudos aqui realizados.

2. CONTEXTO DO ESTUDO

Neste capítulo, são apresentados os assuntos principais pesquisados neste trabalho. Inicialmente são abordados os Métodos Ágeis, seus princípios e características. Em seguida, o foco é qualidade de software, em especial, o modelo CMMI. Finalmente, apresentam-se os conceitos relacionados a *Design Rationale* (DR), suas representações e abordagens.

2.1. Métodos Ágeis

A abordagem ágil para desenvolvimento de software tem despertado muito interesse em grandes organizações de todo o mundo. Há uma tendência para o desenvolvimento ágil de aplicações, devido ao ritmo acelerado de mudanças na tecnologia da informação, pressões por constantes inovações e concorrência acirrada no mundo dos negócios (BOEHM, 2006). Segundo Soares e Meira (2015) as três razões que motivaram o aumento do desenvolvimento ágil foram a habilidade de gerenciar mudanças, produtividade e visibilidade do projeto.

Não se deve confundir agilidade com desorganização ou falta de profissionalismo. Na verdade, ser ágil, ao contrário do que parece, exige um alto grau de disciplina e organização. O termo agilidade quer dizer a habilidade de criar e responder às mudanças, buscando a obtenção de lucro em um ambiente de negócios turbulento (HIGHSMITH, 2009). De maneira semelhante, Meso e Jain (2006) afirmam que apesar de aparentemente as metodologias ágeis parecerem que seguem práticas de desenvolvimento sem planejamento e sem disciplina, estas metodologias enfatizam uma mínima, mas suficiente, quantidade de planejamento dentro de cada iteração.

As ideias do desenvolvimento de software ágil surgiram na década de 60 e até mesmo antes (LARMAN; BASILI, 2003). Muitas das ideias não são novas, mas nunca foram levadas a sério e nunca tinham sido apresentadas como uma metodologia completa (ABBAS, 2009).

Com o objetivo de melhorar o desenvolvimento de software e divulgar os chamados Métodos Ágeis de Desenvolvimento de Software, em 2001, Kent Beck, Mike Beedle, John Kern, Robert C. Martin, dentre outros, criaram o Manifesto para Desenvolvimento Ágil de Software, tendo como prioridade, “satisfazer o cliente através da entrega contínua e adiantada de

software com valor agregado”. Este documento serviu de base para que outros especialistas criassem métodos considerados ágeis, voltados para o desenvolvimento de software, tais como: *Extreme Programming (XP)*, *Scrum*, *Crystal Methods*, *Adaptive Software Development (ASD)* e *Feature-Driven Development (FDD)*.

2.1.1. Manifesto Ágil

Os principais proprietários de métodos ágeis se encontraram em Utah (EUA) em 2001 para discutir ideias e procurar uma alternativa aos processos burocráticos e às práticas adotadas nas abordagens tradicionais da Engenharia de Software. Dessa reunião, surgiu o Manifesto Ágil, definindo quatro valores principais (BECK, 2001):

- Indivíduos e interações, mais que processos e ferramentas;
- Software em funcionamento, mais que documentação abrangente;
- Colaboração com o cliente, mais que negociação de contratos;
- Responder a mudanças, mais que seguir um plano.

Mesmo havendo valor nos itens à direita, valoriza-se mais os itens à esquerda.

Os valores podem ser traduzidos nos doze princípios do Manifesto Ágil (BECK, 2001):

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.

7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade, a arte de maximizar a quantidade de trabalho não realizado, é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Essas características trazem dinamismo para o desenvolvimento, motivação para a equipe e informações mais precisas sobre a verdadeira situação do projeto para o cliente.

O manifesto reuniu pessoas que propunham alternativas baseado em suas próprias experiências (FOWLER, 2005). Kent Beck, o idealizador do XP havia sido contratado para contribuir em um projeto de sistema de folha de pagamento. Ken Schwaber, um dos criadores do Scrum era dono de uma empresa e tinha dificuldades em lidar com requisitos que mudavam o tempo todo. As experiências desses profissionais fez com que buscassem maneiras alternativas de se controlar o processo de desenvolvimento de software (ABBAS, 2009).

Foram ao todo dezessete pessoas que contribuíram para este documento: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland e Dave Thomas.

2.1.2. Características Principais

Os processos ágeis caracterizam-se por serem incrementais, cooperativos, diretos e flexíveis (ABRAHAMSSON et al., 2003). Incremental se refere às pequenas versões e ciclos rápidos de desenvolvimento. Cooperativo quer dizer próximo ao cliente e interação entre os desenvolvedores. Direto significa que o método por si só é fácil de aprender e de modificar e

que é suficientemente documentado. Finalmente, flexível refere-se à habilidade de acomodar mudanças de última hora no projeto. A Figura 3 mostra as principais características dos processos ágeis.

Figura 3 - Características dos Processos Ágeis

Fonte: Adaptado de Abrahamsson et al.(2003)

As metodologias ágeis colocam ênfase na entrega de código funcionando ou produto, ao mesmo tempo em que há menor importância em processos formais e documentação muito abrangente. Em consequência, espera-se que os processos de desenvolvimento de software possam se adaptar e reagir prontamente a mudanças e demandas impostas por um ambiente de desenvolvimento volátil (KRUTCHEN, 2001).

O gerenciamento ágil de projetos segue algumas premissas: uso de ciclos iterativos e curtos; presença ativa do cliente para definir, priorizar e verificar os requisitos do produto, desenvolvimento incremental; auto-organização e autodisciplina (BOEHM; TURNER, 2003(b)).

Fogelström et al. (2010) citam um importante princípio do desenvolvimento ágil: a simplificação. Os requisitos devem ser priorizados com o intuito de produzir versões e o desenvolvimento pode ser iniciado mesmo que todos os requisitos não tenham sido devidamente especificados, iniciando por maior prioridade, consistindo no planejamento da versão e seus refinamentos.

Segundo os princípios e características das metodologias ágeis, sua aplicação é voltada para a busca da satisfação do cliente, com flexibilidade para a incorporação de mudanças de requisitos ao longo do desenvolvimento, menor número de defeitos em função da prática constante de testes e validações junto ao cliente, tempos de desenvolvimento mais curtos com entregas frequentes e incrementais.

A equipe de desenvolvimento de projetos ágeis é um ponto chave para o sucesso. As principais características de equipes ágeis são competência, capacidade de solucionar problemas, tomada de decisão, colaboração, confiança, respeito e auto-organização (COCKBURN; HIGHSMITH, 2001).

É importante reforçar a ideia de que o movimento ágil não é contra a metodologia. É apoiada a utilização de modelagens, mas não a elaboração de diagramas que nunca serão utilizados. A documentação é importante, entretanto não se devem criar centenas de páginas que raramente serão lidas e mantidas. É necessário que haja um planejamento, porém que considere um ambiente suscetível a mudanças (BOEHM; TURNER, 2003(b); LARMAN, 2004).

2.1.3. Principais Métodos Ágeis

No final da década de 90, houve o surgimento dos métodos ágeis como o Desenvolvimento de Software Adaptativo, Crystal, Desenvolvimento de Software Dinâmico, Programação Extrema (XP), *Feature-Driven Development* e Scrum (BOEHM, 2006).

Estas abordagens são caracterizadas por (BECK, 2001):

- Envolvimento direto do cliente no desenvolvimento;
- Uso de múltiplas iterações de desenvolvimento para aprender e evoluir o produto;
- Cliente disposto a compartilhar a responsabilidade das decisões e os riscos.

A seguir, serão descritas as características das principais abordagens ágeis.

2.1.3.1. *Desenvolvimento de Software Adaptativo*

Em inglês, *Adaptive Software Development* (ASD). Proposto por Highsmith (1997), apresenta uma nova maneira de se ver o desenvolvimento de software em uma organização, promovendo um modelo adaptativo. A ênfase é na produção de resultados de alto valor procedentes de uma rápida adaptação a eventos externos e internos, em vez da otimização de técnicas de melhoria de processo. Oferece soluções para o desenvolvimento de sistemas grandes e complexos (HIGHSMITH, 1999). Encoraja o desenvolvimento incremental e iterativo, com prototipação constante.

As principais fases dos modelos de processos adaptativos incluem: comunicação e planejamento, análise, projeto e desenvolvimento, teste e implantação (SADIQ; HASSAN, 2014).

2.1.3.2. *Crystal*

Alistair Cockburn (2004) propõe uma família de métodos por acreditar que diferentes abordagens são necessárias para equipes de tamanhos diferentes. É dividido em Crystal Clear, Crystal Yellow, Crystal Orange e Crystal Red de acordo com o número de participantes e a importância do projeto (ZHONG et al., 2011).

Entretanto, todos os métodos desta família compartilham propriedades como: entrega frequente e incremental, participação direta do usuário, teste de regressão automática e comunicação.

Um conceito importante nos métodos Crystal é o que Cockburn chama de habitabilidade (*habitability*): o mínimo de processo necessário para que a equipe consiga ter sucesso.

2.1.3.3. *Programação Extrema (XP)*

A Programação Extrema (XP) foi concebida e desenvolvida para tratar as necessidades do desenvolvimento de software conduzido por equipes pequenas em situações em que os requisitos são vagos e sofrem mudanças (BECK, 1999). O método XP é tipicamente atribuído a Kent Beck, Ron Jeffries e Ward Cunningham.

O termo “extremo” surgiu porque XP utiliza práticas e princípios comuns a outros métodos, mas em níveis extremos. Por exemplo, se revisão de código é útil, XP revisa código o tempo todo (programação em pares). Se teste é bom, no XP todos testam a todo o momento (teste unitário), incluindo os clientes (testes funcionais). Os testes de integração são realizados várias vezes por dia (integração contínua). As iterações são realmente curtas, podendo ser segundos, minutos e horas.

O ciclo de vida do XP tem quatro atividades básicas: codificação, teste, escuta e projeto. O dinamismo é demonstrado através de cinco valores fundamentais (BECK; ANDRES, 2004):

- 1- **Comunicação** contínua com o cliente e dentro da equipe; o trabalho é feito conjuntamente com todas as pessoas interessadas desde os requisitos até o código.
- 2- **Simplicidade**, atingida por um foco constante em soluções simples; será feito o que é necessário e solicitado, mas nada mais; passos pequenos e simples;
- 3- **Feedback** rápido através de entrega de software funcionando a cada iteração;
- 4- **Coragem** de tratar os problemas proativamente e de apresentar progressos e estimativas verdadeiras;
- 5- **Respeito**: todos os membros da equipe são importantes e devem oferecer e receber respeito dos demais; gerentes, clientes e desenvolvedores são parceiros em um grupo colaborativo.

XP baseia-se em treze práticas primárias e onze práticas adicionais. Estas práticas são derivadas dos cinco valores. Recomenda-se implementar primeiro as práticas primárias e só passar para as adicionais quando tiver adquirido experiência suficiente com as práticas primárias. A Tabela 1 mostra uma lista das práticas do XP.

Tabela 1 - Práticas do XP

Categoria	Práticas Primárias	Práticas Adicionais
Comunicação na equipe	Estórias, espaço de trabalho informativo	Coesão da equipe, equipes reduzidas
Fatores humanos e colaboração na equipe	Toda equipe, sentar junto, programação em pares, trabalho energizado	Envolvimento real do cliente, contrato de escopo negociado, pagar para usar
Gerenciamento de projeto e código	Ciclo semanal, ciclo trimestral, recesso, construção de dez minutos, integração contínua, programação com teste primeiro, projeto incremental	Análise de causa-raiz, codificar e testar, código compartilhado, base de código simples, implantação diária, implantação incremental

Fonte: Abdullah e Abdelsatir (2013)

XP busca reduzir os riscos do projeto, melhorar a resposta a mudanças no negócio e melhorar a produtividade.

2.1.3.4. *Feature-Driven Development (FDD)*

Desenvolvida por Peter Coad e Jeff de Luca no final da década de 90, a FDD combina os benefícios chaves de outras abordagens ágeis com técnicas centradas em modelo. FDD facilmente escala para equipes e projetos bem maiores do que os recomendados para outras abordagens ágeis.

A FDD define duas fases compostas por cinco processos bem definidos e integrados: a fase de concepção e planejamento, composta por “desenvolver um modelo abrangente”, “construir uma lista de funcionalidades” e “planejar por funcionalidade”; e a fase iterativa de construção, composta por “detalhar por funcionalidade” e “construir por funcionalidade” (PALMER; FELSING, 2001). Veja a Figura 4.

Foca na produção de resultados a cada duas semanas e em facilitar inspeções. Os gerentes sabem o que planejar e como estabelecer marcos significativos (PALMER; FELSING, 2001).

Reduz riscos enfatizando a entrega frequente de resultados de trabalho tangíveis.

Figura 4 – Feature-Driven Development – duas fases, cinco processos

Fonte: Adaptado de Palmer e Felsing (2001)

2.1.3.5. Scrum

Scrum é um modelo iterativo e incremental desenvolvido nas décadas de 80 e 90 por Ken Schwaber, Jeff Sutherland e Mike Beedle (SCHWABER; BEEDLE, 2001). Propõe iterações de duas a quatro semanas, chamadas Sprints, com acompanhamento diário por meio das reuniões em pé (*stand-up meetings*).

Segundo Schwaber e Sutherland (2016), Scrum é um *framework* dentro do qual pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível.

O Scrum não define uma técnica específica para o desenvolvimento de software durante a etapa de implementação. Ele se concentra em descrever como os membros das equipes devem trabalhar para produzir um sistema flexível, num ambiente de mudanças constantes.

Scrum define três papéis principais que possuem tarefas e propósitos diferentes durante o processo e suas práticas: Product Owner, Time de Desenvolvimento e o Scrum Master (SCHWABER, 2004). O Product Owner representa os interesses de todas as pessoas envolvidas. Mantém uma lista priorizada de requisitos do projeto (*Product Backlog*) com estimativas de tempo para a sua implementação. O Time de Desenvolvimento é responsável por desenvolver as funcionalidades do produto. Tem autoridade para realizar a sua organização e gerenciamento. Os membros de um Time de Desenvolvimento são responsáveis coletivamente pelo sucesso de cada iteração e pelo projeto como um todo. O Scrum Master é responsável por gerenciar o processo Scrum, ou seja, ensina Scrum a todas as pessoas envolvidas no projeto, implementa o Scrum de maneira a se adaptar à cultura da organização

e assegura que todas as pessoas seguem as regras e práticas do Scrum. Interage tanto com a Equipe como com o cliente e garante que as pessoas trabalhem de forma mais produtiva possível. A Figura 5 mostra um resumo dos papéis no Scrum e suas atividades principais.

Figura 5 – Papéis no Scrum e suas atividades

Fonte: Adaptado de Schwaber (2004)

O fluxo do Scrum é mostrado na Figura 6. Os projetos são divididos em ciclos chamados de *Sprints*. As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *Product backlog*. As tarefas alocadas em um *Sprint* são transferidas do *Product backlog* para o *Sprint backlog*. A cada dia de uma *Sprint*, a equipe faz uma breve reunião com o objetivo de disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia. Ao final de um *Sprint*, a equipe apresenta as funcionalidades implementadas e parte-se para o planejamento do próximo *Sprint*, reiniciando o ciclo.

Figura 6 - Fluxo do método Scrum

Fonte: Adaptado de Schwaber (2004)

2.1.4. Desafios e Limitações

Um dos desafios do desenvolvimento ágil de software refere-se a projetos distribuídos. Os princípios de comunicação informal são conflitantes em razão da dificuldade de compartilhar informações (LEE; DELONE; ESPINOSA, 2006). Pode haver também diferenças culturais entre os membros do projeto. A comunicação frente-a-frente entre equipes e clientes não é possível, pois em ambientes distribuídos eles não estão todos no mesmo local, limitando as ações de suporte dessa abordagem (TURK; RUMPE, 2009). A prática de reuniões diárias torna-se uma atividade complexa, podendo ser em alguns casos impraticável, em razão das diferenças de fuso horário e limitações de comunicação e compartilhamento de informações.

Um ponto importante para o sucesso de projetos ágeis refere-se às pessoas envolvidas, que devem ser, preferencialmente, competentes, confiáveis e colaboradoras. O funcionamento ágil pode ser comprometido se a equipe não possui estas características ou se as pessoas são pouco experientes (COCKBURN; HIGHSMITH, 2001).

Para Farhan et al. (2009), um dos desafios dos métodos ágeis é a falta de um método apropriado de avaliação da arquitetura de software para modelos de processos ágeis. Sem avaliação, os modelos ágeis podem sofrer de problemas severos de qualidade e de manutenção.

Kajko-Mattsson et al. (2006) ressaltam que as metodologias ágeis de software representam poucos detalhes de implementação do processo de desenvolvimento do produto e, por causa desta abordagem superficial, eles podem construir software que satisfaz as necessidades individuais dos projetos em curto prazo, mas não necessariamente satisfaz as necessidades em longo prazo da organização.

Lycett et al. (2003) destacam que a abordagem ágil é fortemente a favor da comunicação humana e colaboração ao invés de atividades repetidas e definidas como mecanismo para desenvolver software de qualidade. A conclusão que chegam é que, na prática, a dificuldade que muitas organizações enfrentam com a abordagem ágil reside na mudança cultural necessária para convencer a gerência dos benefícios e da viabilidade no contexto de pressão de mercado global e práticas regulatórias.

Telles (2004) salienta que as metodologias ágeis necessitam superar alguns desafios que impedem a obtenção de melhores resultados e a sua ampla utilização em projetos. Estes

desafios envolvem mudanças na maneira como o software é comercializado, na postura do cliente em contratar uma equipe para o desenvolvimento de um sistema, na organização desenvolvedora de software, na forma como o projeto é gerenciado e na equipe de desenvolvedores. Por exemplo, contratos com escopo variável são mais interessantes para metodologias ágeis do que contratos de escopo fechado, por permitirem mudanças nos projetos.

A realização de manutenção do software após a finalização do projeto de desenvolvimento, sobretudo em sistemas que lidam com um domínio pouco conhecido, torna-se difícil. Possíveis mudanças nos membros que compõe a equipe de desenvolvimento agravam ainda mais essa atividade (SILVA, 2009). As metodologias ágeis demandam uma maneira efetiva de representar o conhecimento vivenciado por ocasião do desenvolvimento do sistema, para que outros desenvolvedores possam absorvê-lo e contribuir com modificações posteriores à entrega do software sem, no entanto, sobrecarregar o processo a ponto de torná-lo burocrático.

Abdullah e Abdelsatir (2013) fazem uma boa síntese das limitações dos desenvolvimentos ágeis. As metodologias ágeis são criticadas por serem informais e confiar nos fatores humanos e suas interações em vez de seguir abordagens disciplinadas de desenvolvimento de software. Além disso, a criticidade e o tamanho do projeto são considerados fatores impeditivos para a adaptação das metodologias ágeis. Finalmente, sistemas distribuídos de larga escala merecem atenção, pois envolvem um grau de complexidade por natureza para o gerenciamento de recursos distribuídos bem como a comunicação entre as pessoas envolvidas no processo de desenvolvimento.

2.1.5. Considerações finais

É possível verificar que embora haja uma tendência à utilização dos métodos ágeis, eles podem não ser adequados para todos os tipos de organizações ou projetos.

Um estudo de Misra et al. (2009) determina os fatores que levaram ao sucesso de projetos de desenvolvimento de software que adotaram as práticas do gerenciamento ágil: colaboração, comprometimento do cliente, tempo da tomada de decisão, cultura corporativa, controle, características pessoais, cultura social, treinamento, aprendizagem e satisfação do cliente.

2.2. Qualidade de software - CMMI

A habilidade de entregar projetos de tecnologia da informação bem sucedidos tem se tornado uma necessidade crítica e estratégica para as organizações contemporâneas (OWENS; KHAZANCHI, 2009). Qualidade de software é um critério utilizado para medir o sucesso de um projeto de desenvolvimento de software.

A Engenharia de Software tem, entre seus grandes objetivos, aumentar a qualidade do software, investigando aspectos de qualidade de produtos de software e dos processos de desenvolvimento dos mesmos (ROCHA et al., 2001).

Vários modelos de melhoria de qualidade de software são utilizados, como por exemplo, ISO 9000-3 (ABNT, 1993), ISO/IEC TR 15504 (ISO/IEC TR 15504, 2008) e CMMI (CMMI-DEV V1.3, 2010). Sheard (2001) descreve resumidamente outros modelos, como o *Team Software Process* (TSP), IEE/EIA 12207 e *Systems Engineering Capability Assessment Model* (SECAM).

Os modelos estabelecem formas para garantia e gerenciamento dos produtos e processo de software, prevendo adaptações a características específicas das organizações.

Para esta pesquisa, o foco é no modelo de maturidade CMMI, desenvolvido pelo *Software Engineering Institute* (SEI).

2.2.1. Definições

Existem diversas definições de qualidade, que se complementam em aspectos diferentes, alguns com mais foco no produto e outros no processo de desenvolvimento. Jin e Zeng (2011) salientam que qualidade não é um conceito específico, mas sim uma medida abstrata. Mas, a maioria relaciona qualidade à conformidade com os requisitos e satisfação do cliente.

Para Feigenbaum (1991), a qualidade é determinada pelo consumidor e não pelo engenheiro, marketing ou administração. Ela é baseada na experiência atual do cliente com o produto ou serviço e medida com base em seus requisitos explicitados ou não, conscientes ou não, tecnicamente operacional ou totalmente subjetivo. De maneira semelhante, Crosby (1979)

define qualidade com a plena e perfeita conformidade com os requisitos do cliente. É o defensor do princípio “faça certo na primeira vez”. Ele salienta que a qualidade é gratuita. O que é custoso é a não qualidade, ou seja, todas as atividades resultantes de não fazer o trabalho correto da primeira vez.

Ishikawa (1985) afirma que qualidade interpretada de uma forma restrita significa qualidade do produto e interpretada de uma forma ampla, significa qualidade do produto, serviço, informação, pessoas, processos, sistemas, etc.

Há dois aspectos de qualidade. Um deles está ligado com a consideração de qualidade de algo como uma realidade objetiva independente da existência humana. O outro está ligado ao que pensamos, sentimos ou percebemos como resultado de uma realidade objetiva. Em outras palavras, há um lado subjetivo da qualidade (SHEWART, 1980).

Joseph M. Juran, um dos precursores no assunto sobre gerenciamento da qualidade, é conhecido por introduzir o elemento humano na qualidade, estendendo o que antes era uma metodologia estatística (BAILEY, 2007). Para Juran (1988), há um custo ótimo de qualidade onde um esforço de qualidade adicional pode ser mais custoso que tolerar um nível de defeitos.

Gillies (2002) afirma que o termo qualidade é particularmente problemático quando trata o assunto desenvolvimento de software. Isso ocorre porque um software não possui existência física, os clientes não sabem exatamente quais são as suas necessidades, os requisitos não são imutáveis e a evolução de hardware e software é extremamente rápida. Além disso, a expectativa dos consumidores é sempre alta e pouco gerenciada. Por isso, é necessária a definição de um modelo de qualidade com métricas tangíveis que reflitam uma forma de avaliar os produtos de software.

Para Khalane e Tanner (2013), qualidade refere-se à habilidade de um projeto de software em satisfazer os requisitos do negócio e agregar valor ao usuário.

Pressman (2010) foca em três pontos quando se refere à qualidade de software:

- 1- Requisitos de software são a base para se realizar medidas de qualidade. A falta de conformidade com as especificações é um problema de falta de qualidade;

- 2- Padrões especificados definem um conjunto de critérios de desenvolvimento que guiam a forma como o software será desenvolvido. A não conformidade com as definições desses padrões resultará em falta de qualidade;
- 3- O software deve obedecer a requisitos que muitas vezes não estão explícitos, mas são esperados de um software profissional, como, por exemplo, a manutenibilidade. A ausência desses itens torna a qualidade de um software contestável.

Controle de qualidade (CQ) abrange uma série de inspeções, revisões e testes utilizados durante o processo de desenvolvimento de software, na tentativa de garantir que o produto final cumpra com os requisitos especificados (ISHIKAWA, 1985). As atividades de CQ podem ser totalmente automatizadas, manuais ou uma combinação das duas.

Garantia de qualidade (GQ) é um termo mais abrangente do que CQ e consiste em fornecer a gestão de um projeto dados sobre a qualidade de um produto. GQ é um conjunto de ações necessárias para alcançar qualidade de software, incluindo qualidade do produto de software e do processo de desenvolvimento. Deming (1986), um especialista em gerenciamento de qualidade, enfatiza a necessidade de distinguir qualidade de garantia da qualidade. Assegurar que todos façam o seu melhor é o objetivo da garantia da qualidade e a base da boa qualidade é que todos saibam e entendam o que deve ser feito.

As atividades de GQ devem ser integradas com o plano de projeto que implementa um ou mais modelos de desenvolvimento de software. Além disso, para cada uma das atividades deve ser determinado tempo, tipo, responsável e recursos (GALIN, 2003).

Segundo Pressman (2010), é necessário um controle do processo de desenvolvimento para que até as tarefas mais simples sejam executadas de maneira padronizada, reduzindo a distância entre um produto e outro e facilitando o controle de variáveis como taxa de defeitos, horas de desenvolvimento e recursos financeiros.

Com o objetivo de fornecer melhoria do processo e da qualidade do produto foram desenvolvidos modelos para o processo de desenvolvimento de produtos como o CMMI que é detalhado a seguir.

2.2.2. CMMI

Os modelos CMMI (*Capability Maturity Model Integration* – Modelo Integrado de Maturidade e de Capacidade) são coleções de melhores práticas que auxiliam as organizações a melhorar os seus processos (CMMI-DEV V1.3, 2010).. Estes modelos são desenvolvidos por equipes com membros da indústria, governo e o *Software Engineering Institute* (SEI).

O CMMI para Desenvolvimento (CMMI-DEV) fornece um guia para aplicar as melhores práticas do CMMI em uma organização de desenvolvimento (CMMI-DEV V1.3, 2010). É destinado ao desenvolvimento de produtos e serviços, e composto pelas melhores práticas associadas a atividades de desenvolvimento e de manutenção que cobre o ciclo de vida do produto desde a concepção até a entrega e manutenção.

O público alvo do CMMI-DEV inclui qualquer pessoa interessada em melhorar o processo em um ambiente de desenvolvimento.

No modelo CMMI, utilizam-se níveis para descrever um caminho evolutivo recomendado para uma organização que deseja melhorar os processos utilizados para desenvolver e manter seus produtos e serviços (CMMI-DEV V1.3, 2010). Os seis níveis de capacidade, numerados de 0 a 5 são:

0. Incompleto: processo não é executado ou é executado parcialmente;
1. Executado: processo satisfaz às metas específicas da área de processo, apoiando e viabilizando o trabalho necessário para produzir os produtos de trabalho;
2. Gerenciado: processo executado que dispõe de infraestrutura adequada para apoiar o processo; é monitorado, controlado e revisado;
3. Definido: é processo gerenciado adaptado a partir do conjunto de processos-padrão da organização de acordo com as diretrizes para adaptação da organização e contribui com produtos de trabalho, medidas e outras informações de melhoria de processo;
4. Gerenciado quantitativamente: processo definido que é controlado por meio de técnicas estatísticas e outras técnicas quantitativas;
5. Em otimização: processo gerenciado quantitativamente que é melhorado com base no entendimento das causas comuns de variação inerentes ao processo.

Os níveis de capacidade de uma área de processo são alcançados por meio da aplicação de práticas genéricas ou práticas alternativas adequadas aos processos associados àquela área de processo.

No CMMI, definem-se também níveis de maturidade que representam o amadurecimento de um importante subconjunto dos processos da organização (CMMI-DEV V1.3, 2010). Existem cinco níveis de maturidade:

1. Inicial;
2. Gerenciado;
3. Definido;
4. Gerenciado quantitativamente;
5. Em otimização.

Intencionalmente, os níveis de maturidade de 2 a 5 utilizam os mesmos termos em relação aos níveis de capacidade de 2 a 5, uma vez que os conceitos são complementares.

As organizações podem alcançar melhorias progressivas em sua maturidade organizacional, conseguindo primeiro o controle no âmbito do projeto até chegar à melhoria contínua de processo no contexto da organização, utilizando tanto dados quantitativos quanto dados qualitativos para a tomada de decisão.

2.2.3. Áreas de Processo CMMI

Uma área de processo é um conjunto de práticas relacionadas a uma área que, quando implementado, satisfaz a um conjunto de metas consideradas importantes para realizar melhorias naquela área (CMMI-DEV V1.3, 2010).

Segue a lista das 22 áreas de processo do CMMI versão 1.3, apresentadas em ordem alfabética de sua nomenclatura em inglês (CMMI-DEV V1.3, 2010). A nomenclatura para o português foi obtida através do manual traduzido (CMMI-DEV V1.2, 2006), com exceção do item 8 “Gestão do Desempenho da Organização” que por não constar na versão 1.2, a tradução foi da autora.

Áreas de processo do CMMI-DEV V1.3 (2010):

1. Análise e Resolução de Causas (CAR-*Causal Analysis and Resolution*)
2. Gestão de Configuração (CM-*Configuration Management*)
3. Análise e Tomada de Decisões (DAR-*Decision Analysis and Resolution*)
4. Gestão Integrada de Projeto (IPM-*Integrated Project Management*)
5. Medição e Análise (MA-*Measurement and Analysis*)
6. Definição dos Processos da Organização (OPD-*Organizational Process Definition*)
7. Foco nos Processos da Organização (OPF-*Organizational Process Focus*)
8. Gestão do Desempenho da Organização (OPM-*Organizational Performance Management*)
9. Desempenho dos Processos da Organização (OPP-*Organizational Process Performance*)
10. Treinamento na Organização (OT-*Organizational Training*)
11. Integração de Produto (PI-*Product Integration*)
12. Monitoramento e Controle de Projeto (PMC-*Project Monitoring and Control*)
13. Planejamento de Projeto (PP-*Project Planning*)
14. Garantia da Qualidade de Processo e Produto (PPQA-*Process and Product Quality Assurance*)
15. Gestão Quantitativa de Projeto (QPM-*Quantitative Project Management*)
16. Desenvolvimento de Requisitos (RD-*Requirements Development*)
17. Gestão de Requisitos (REQM-*Requirements Management*)
18. Gestão de Riscos (RSKM-*Risk Management*)
19. Gestão de Contrato com Fornecedores (SAM-*Supplier Agreement Management*)
20. Solução Técnica (TS-*Technical Solution*)
21. Validação (VAL-*Validation*)
22. Verificação (VER-*Verification*)

Essas áreas de processo são organizadas em quatro categorias: Gerenciamento de Processos, Gerenciamento de Projeto, Engenharia e Suporte. Essas categorias enfatizam como as áreas existentes se relacionam. Uma vez selecionadas as áreas de processos, deve-se selecionar também quanto se deseja que os processos associados àquelas áreas amadureçam, isto é, selecionar o nível de capacidade apropriado (CMMI-DEV V1.3, 2010).

A Tabela 2 apresenta uma lista de todas as áreas de processo, suas categorias e níveis de maturidade.

Tabela 2 - Áreas de Processo, suas Categorias e Níveis de Maturidade

Categoria	Área de Processo	Nível de Maturidade
Gerenciamento de Processo	Definição dos Processos da Organização (OPD)	3
	Foco nos Processos da Organização (OPF)	3
	Gestão do Desempenho da Organização (OPM)	5
	Desempenho dos Processos da Organização (OPP)	4
	Treinamento na Organização (OT)	3
Gerenciamento de Projeto	Gestão Integrada de Projeto (IPM)	3
	Monitoramento e Controle de Projeto (PMC)	2
	Planejamento de Projeto (PP)	2
	Gestão Quantitativa de Projeto (QPM)	4
	Gestão de Requisitos (REQM)	2
	Gestão de Riscos (RSKM)	3
	Gestão de Contrato com Fornecedores (SAM)	2
Engenharia	Integração de Produto (PI)	3
	Desenvolvimento de Requisitos (RD)	3
	Solução Técnica (TS)	3
	Validação (VAL)	3
	Verificação (VER)	3
Suporte	Análise e Resolução de Causas (CAR)	5
	Gestão de Configuração (CM)	2
	Análise e Tomada de Decisões (DAR)	3
	Medição e Análise (MA)	2
	Garantia da Qualidade de Processo e Produto (PPQA)	2

Fonte: CMMI-DEV V1.3 (2010)

A seguir, é apresentada uma descrição de cada uma dessas áreas de processo. Para visualizar a lista completa das metas e práticas específicas de cada área de processo, veja o ANEXO – Metas e práticas específicas das áreas de processo do CMMI-DEV 1.3.

Análise e Resolução de Causas (CAR)

Identifica causas de defeitos e de outros problemas, realizando as ações necessárias.

O objetivo da área é fornecer subsídios para identificar causas de defeitos e de determinados problemas e implementar ações para melhorar o desempenho do processo (CMMI-DEV V1.3,

2010). Melhora a qualidade e a produtividade, uma vez que promove a prevenção de defeitos em um produto.

A área de processo CAR tem como práticas específicas selecionar defeitos e outros problemas para análise; realizar a análise de causas de defeitos e de outros problemas selecionados e propor ações para tratá-las; implementar propostas de ação selecionadas que foram desenvolvidas durante a análise de causa; avaliar os efeitos das mudanças no desempenho do processo; e registrar dados de análise e resolução de causas para uso no projeto e na organização (CMMI-DEV V1.3, 2010).

Pode ser vista como um mecanismo de comunicação de lições aprendidas entre projetos, uma vez que defeitos e problemas podem ter sido previamente detectados em outros projetos ou em fases ou tarefas iniciais do projeto atual.

Gestão de Configuração (CM)

O propósito da Gestão de Configuração é estabelecer e manter a integridade de produtos de trabalho, utilizando identificação de configuração, controle de configuração, balanço das atividades de configuração e auditorias de configuração (CMMI-DEV V1.3, 2010).

A área de processo CM assegura que a integridade de todos os produtos de trabalho e todos os itens utilizados para criar qualquer produto de trabalho está estabelecida e efetivamente mantida (LAND; WALZ, 2006).

“Identificar Itens de Configuração” e “Criar ou Liberar *Baselines*” estão entre as práticas específicas da GC. Um “item de configuração” é uma entidade selecionada para a gestão de configuração, podendo ser composta por diversos produtos de trabalho relacionados que formam um *baseline*. Um *baseline* de software pode ser um conjunto formado por requisito, *design*, arquivos de código fonte e código executável associado, arquivos de *build* e documentação de usuário ao qual tenha sido atribuído um identificador único.

A área de processo Gestão de Configuração envolve (CMMI-DEV V1.3, 2010):

- Identificação da configuração de produtos de trabalho selecionados que compõem os *baselines* em determinados instantes;

- Controle de mudanças nos itens de configuração;
- Construção de produtos de trabalho a partir do sistema de gestão de configuração ou fornecimento de especificação para fazê-lo;
- Manutenção da integridade dos *baselines*;
- Disponibilização do status e dos dados atuais de configuração para desenvolvedores, usuários finais e clientes.

Análise e Tomada de Decisões (DAR)

Analisa decisões possíveis, avaliando alternativas para critérios estabelecidos.

O objetivo da área de processo Análise e Tomada de Decisões é fornecer subsídios para tomar decisões com base em um processo formal para avaliação de alternativas identificadas em relação a critérios estabelecidos (CMMI-DEV V1.3, 2010).

As práticas específicas desta área de processo incluem o estabelecimento e manutenção de diretrizes para determinar quais questões críticas estão sujeitas a um processo formal para avaliação de alternativas; o estabelecimento e manutenção de critérios para avaliar as alternativas e para classificá-los de forma relativa; a identificação de soluções alternativas para tratar questões críticas; a seleção de métodos de avaliação; a avaliação de soluções alternativas utilizando os critérios e os métodos estabelecidos; e a seleção de soluções dentre as alternativas, com base nos critérios de avaliação (CMMI-DEV V1.3, 2010).

Gestão Integrada de Projeto (IPM)

Estabelece e gere o projeto e o envolvimento de *stakeholders* relevantes de acordo com o processo definido.

A Gestão Integrada de Projeto envolve (CMMI-DEV V1.3, 2010):

- Estabelecer o processo definido para o projeto no início do projeto a partir da adaptação do conjunto de processos-padrão da organização;
- Gerenciar o projeto utilizando o processo definido para o projeto;

- Estabelecer o ambiente de trabalho para o projeto com base nos padrões de ambiente de trabalho da organização;
- Utilizar os ativos de processo da organização e contribuir para sua melhoria;
- Permitir que as preocupações relevantes das partes interessadas sejam identificadas, consideradas e, quando apropriado, tratadas durante o desenvolvimento do produto;
- Assegurar que as partes interessadas relevantes executem suas tarefas de forma coordenada e em tempo hábil (1) para tratar de requisitos de produto e requisitos de componente de produto, planos, objetivos, problemas e riscos; (2) para cumprir seus compromissos; e (3) para identificar, acompanhar e solucionar questões críticas sobre coordenação.

Definindo-se o processo para cada projeto a partir de um conjunto de processos-padrão da organização, a variabilidade entre projetos tende a se reduzir e os projetos podem compartilhar mais facilmente os arquivos de processo, os dados e as lições aprendidas (CMMI-DEV V1.3, 2010).

Medição e Análise (MA)

Desenvolve e mantém uma capacidade de medição para dar suporte às necessidades de informação de gerenciamento.

A área de processo Medição e Análise envolve (CMMI-DEV V1.3, 2010):

- Especificar os objetivos de medição e análise, de forma que estejam alinhados com as necessidades de informação e objetivos identificados;
- Especificar medidas, técnicas de análise e mecanismos para coleta e armazenamento de dados, relatórios e *feedback*;
- Implementar técnicas de análise e mecanismos para coleta de dados e geração de relatórios;
- Fornecer resultados objetivos que possam ser utilizados na tomada de decisões bem fundamentadas e na implementação de ações corretivas apropriadas.

O foco inicial das atividades de medição está no projeto. Entretanto, a capacidade de medição pode se mostrar útil para tratar as necessidades de informação de toda a organização ou corporação.

Land e Walz (2006) alertam que a coleta de métricas é uma atividade que consome recursos do projeto e, por isso, sugere que se inicie com pequena abrangência e se concentre em métricas que tenham maior probabilidade de promover mudanças.

Definição dos Processos da Organização (OPD)

Estabelece e mantém um conjunto utilizável de recursos de processos da organização.

As metas e práticas específicas da OPD (CMMI-DEV V1.3, 2010) são:

1. Estabelecer e manter um conjunto de processos-padrão da organização, que geralmente inclui processos técnicos, gerenciais, administrativos, de suporte e organizacionais;
2. Estabelecer e manter as descrições dos modelos de ciclo de vida aprovados para uso na organização;
3. Estabelecer e manter os critérios e as diretrizes para a adaptação do conjunto de processos-padrão da organização;
4. Estabelecer e manter o repositório de medições de produto e processo da organização;
5. Estabelecer e manter a biblioteca de ativos de processo da organização;
6. Estabelecer e manter padrões de ambiente de trabalho, permitindo que a organização e os projetos sejam beneficiados pelo uso comum de ferramentas, treinamentos e manutenção;
7. Estabelecer e manter regras e diretrizes para a estrutura, formação e operação das equipes.

Foco nos Processos da Organização (OPF)

Planeja, implementa e instala melhorias de processo organizacional baseado em pontos fortes e fracos dos processos e dos ativos de processo da organização (CMMI-DEV V1.3, 2010).

Um planejamento cuidadoso é necessário para assegurar que os esforços de melhoria de processo na organização sejam adequadamente gerenciados e implementados.

Gestão do Desempenho da Organização (OPM)

Gerencia, de maneira proativa, o desempenho da organização para suprir os objetivos do negócio. Analisa iterativamente dados agregados de projetos, identificando lacunas no desempenho contra os objetivos do negócio.

Os objetivos de negócio que esta área de processo deve tratar incluem (CMMI-DEV V1.3, 2010):

- Melhora da qualidade do produto
- Aumento da produtividade
- Aumento da eficiência e eficácia do processo
- Aumento da consistência em satisfazer orçamento e prazo
- Aumento da satisfação do cliente e usuário final
- Menor tempo de produção ou desenvolvimento para alterar uma funcionalidade, adicionar novas características ou adaptar-se a novas tecnologias
- Melhora do uso de recursos na organização

As práticas desta área de processo têm um impacto maior em organizações cujos projetos são gerenciados quantitativamente.

Desempenho dos Processos da Organização (OPP)

Estabelece e mantém um entendimento quantitativo do desempenho do conjunto de processos-padrão da organização no apoio aos objetivos para qualidade e para o desempenho de processo (CMMI-DEV V1.3, 2010).

Segundo CMMI-DEV V1.3 (2010), organizações que possuem medidas, dados e técnicas analíticas de características críticas de processo são capazes de determinar se os processos são

previsíveis, identificar processos que demonstrem comportamentos atípicos e identificar aspectos que possam ser melhorados.

Treinamento na Organização (OT)

Desenvolve habilidades e conhecimento das pessoas de modo que possam desempenhar suas tarefas efetiva e eficientemente. Inclui treinamentos para apoiar os objetivos estratégicos da organização e para satisfazer às necessidades táticas de treinamento que são comuns aos projetos e aos grupos de suporte (CMMI-DEV V1.3, 2010).

Certas competências podem ser transmitidas de forma eficiente e eficaz por outros meios que não envolvam treinamento presencial em sala de aula (por exemplo, *mentoring* informal).

Integração de Produto (PI)

Fornecer subsídios para montar o produto a partir de componentes de produto, assegurando que o produto integrado execute as funções de forma apropriada, e entregar o produto. O objetivo é conseguir a completa integração do produto por meio da montagem progressiva dos seus componentes. Este processo pode se iniciar com análises e simulações (uso de protótipos, por exemplo) e evoluir constante e gradativamente até o produto final (CMMI-DEV V1.3, 2010).

Monitoramento e Controle de Projeto (PMC)

Proporciona um entendimento do progresso do projeto de maneira que ações corretivas possam ser tomadas quando o desempenho do projeto desviar significativamente do plano (CMMI-DEV V1.3, 2010).

Os objetivos são monitorar o desempenho e o progresso do projeto em relação ao plano de projeto e gerenciar ações corretivas quando o desempenho ou os resultados do projeto tiverem um desvio considerável em relação ao plano.

Revisões técnicas, revisões gerenciais e auditorias são atividades chaves da PMC (LAND; WALZ, 2006).

Planejamento de Projeto (PP)

Estabelece e mantém o plano de projeto, elaborando o plano, interagindo com as partes interessadas, obtendo comprometimento com o plano e realizando a sua manutenção (CMMI-DEV V1.3, 2010).

O planejamento tem início com os requisitos que caracterizam o produto e o projeto.

Para Land e Walz (2006), PP é a base que pode determinar o sucesso ou falha de um projeto de software. É crucial que atenção apropriada seja dada a esta área de processo para possibilitar que os gerentes de projeto tenham acesso à informação útil para tomar decisões e atingir os objetivos de cronograma e orçamento.

Garantia da Qualidade de Processo e Produto (PPQA)

Fornecer visibilidade para a equipe e gerência sobre os processos e produtos de trabalho associados.

A área de processo PPQA envolve (CMMI-DEV V1.3, 2010):

- Avaliar objetivamente os processos executados, produtos de trabalho e serviços em relação às descrições de processo, padrões e procedimentos aplicáveis;
- Identificar e documentar as não conformidades;
- Fornecer *feedback* à equipe do projeto e aos gerentes sobre os resultados das atividades de garantia da qualidade;
- Assegurar que as não conformidades sejam tratadas.

A área de processo PPQA apoia a entrega de produtos e serviços de alta qualidade.

Gestão Quantitativa de Projeto (QPM)

Gerencia o processo definido do projeto de uma maneira quantitativa visando alcançar os objetivos para qualidade e para desempenho de processos estabelecidos para o projeto (CMMI-DEV V1.3, 2010).

Um elemento essencial da gestão quantitativa é a confiança nas estimativas, isto é, a capacidade de prever o quanto o projeto é capaz de alcançar os seus objetivos para qualidade e para desempenho de processo (CMMI-DEV V1.3, 2010).

Desenvolvimento de Requisitos (RD)

Fornece subsídios para produzir e analisar os requisitos de cliente, de produto e de componente de produto. Juntos, esses três tipos de requisitos tratam das necessidades das partes interessadas relevantes (CMMI-DEV V1.3, 2010).

Em outras palavras, a área de processo RD identifica as necessidades do cliente traduzindo-as em requisitos de produto. Este conjunto de requisitos de produto é analisado para produzir uma solução conceitual de alto nível e é alocado a um conjunto de requisitos de componente de produto. Este conjunto de requisitos de produto e de componente de produto descrevem claramente as características, desempenho e características chaves de *design* do produto, de modo que os desenvolvedores e testadores possam entender e utilizar (LAND; WALZ, 2006).

O desenvolvimento de requisitos inclui as seguintes atividades (CMMI-DEV V1.3, 2010):

- Levantamento, análise, validação e comunicação de necessidades, expectativas e restrições do cliente;
- Coleta e conciliação das necessidades das partes interessadas;
- Desenvolvimento dos requisitos de ciclo de vida do produto;
- Estabelecimento dos requisitos do cliente;
- Estabelecimento de requisitos iniciais do produto e dos componentes de produto, compatíveis com os requisitos do cliente.

Decisões de *design*, ações corretivas subsequentes e *feedback* durante cada fase do ciclo de vida do produto são analisados quanto ao impacto nos requisitos derivados e alocados.

As metas e práticas específicas do RD compreendem três metas principais: definição de um conjunto de requisitos do cliente, definição de um conjunto de requisitos de produto ou de componente de produto e a análise dos requisitos do cliente, do produto e dos requisitos dos componentes de produto. As práticas da terceira meta têm como objetivo apoiar as práticas das duas primeiras metas (CMMI-DEV V1.3, 2010).

Gestão de Requisitos (REQM)

Gerencia os requisitos dos produtos do projeto e os componentes de produto do projeto e assegura um alinhamento entre esses requisitos e os planos do projeto e produtos de trabalho (CMMI-DEV V1.3, 2010).

A REQM gerencia todos os requisitos recebidos ou gerados pelo projeto, incluindo tanto requisitos técnicos como não técnicos, assim como aqueles requisitos impostos ao projeto pela organização.

Parte da gestão de requisitos consiste na documentação de mudanças de requisitos e de sua motivação, mantendo a rastreabilidade bidirecional entre os requisitos originais e todos os requisitos de produto e de componentes de produto (CMMI-DEV V1.3, 2010).

Gestão de Riscos (RSKM)

Identifica problemas potenciais antes de sua ocorrência, de forma que atividades de tratamento de riscos possam ser planejadas e colocadas em prática quando necessário (CMMI-DEV V1.3, 2010).

As metas e práticas da RSKM podem ser divididas em três partes: definição de uma estratégia para a gestão de riscos; identificação e análise de riscos; e tratamento de riscos identificados, incluindo a implementação de planos de mitigação de riscos quando necessário.

Uma gestão de riscos efetiva inclui a identificação abrangente e antecipada de riscos por meio da colaboração e do envolvimento das partes interessadas relevantes (CMMI-DEV V1.3, 2010).

Gestão de Contrato com Fornecedores (SAM)

Gerencia a aquisição de produtos de fornecedores.

A área de processo SAM envolve a determinação do tipo de aquisição, a seleção de fornecedores, o estabelecimento e manutenção de contrato de fornecedores, a execução dos contratos, a aceitação da entrega de produtos adquiridos e a transferência dos produtos adquiridos para o projeto (CMMI-DEV V1.3, 2010).

Nesta área de processo, os contratos com os fornecedores são estabelecidos, mantidos e cumpridos pelo projeto e pelo fornecedor.

Um contrato formal é estabelecido para gerenciar o relacionamento entre a organização e o fornecedor, podendo ser um contrato, uma licença, um acordo de nível de serviço ou memorando de acordo entre as partes (CMMI-DEV V1.3, 2010).

Solução Técnica (TS)

Projeta, desenvolve e implementa soluções para os requisitos.

A área de processo TS converte os requisitos na arquitetura de produto, projeto de componentes de produto e no próprio componente de produto (LAND; WALZ, 2006).

A área de processo TS é responsável por (LAND; WALZ, 2006):

- Avaliar e selecionar soluções que potencialmente satisfaçam um conjunto apropriado de requisitos;
- Desenvolver projetos detalhados para as soluções selecionadas;

- Implementar os projetos transformando-os em código como um produto ou componente de produto.

Segundo CMMI-DEV V1.3 (2010), a área de processo TS é aplicável a qualquer nível da arquitetura de produto e a todos os produtos, componentes de produto (incluindo serviços e seus componentes) e processo do ciclo de vida relacionados ao produto.

Validação (VAL)

Demonstra que um produto ou componente de produto satisfaz o uso planejado.

A prática “Selecionar Produtos para Validação” permite a identificação do produto ou componente de produto a ser validado e dos métodos a serem utilizados para executar a validação. A prática específica “Estabelecer Ambiente de Validação” permite a determinação do ambiente a ser utilizado para realizar a validação. A prática “Estabelecer Procedimentos e Critérios de Validação” permite a elaboração dos procedimentos e critérios de validação alinhados às características dos produtos selecionados, restrições do cliente, métodos e ambiente de validação. Finalmente, as práticas específicas “Realizar Validação” e “Analisar Resultados de Validação” permitem a execução da validação de acordo com métodos, procedimentos e critérios e a análise dos resultados (CMMI-DEV V1.3, 2010).

Verificação (VER)

Assegura que os produtos de trabalho selecionados satisfaçam os requisitos especificados (CMMI-DEV V1.3, 2010).

A área de processo VER envolve a preparação e execução da verificação, e identificação de ação corretiva.

A validação demonstra que o produto, tal como fornecido, atenderá ao seu uso pretendido, enquanto que a verificação examina se o produto de trabalho reflete adequadamente os requisitos especificados (CMMI-DEV V1.3, 2010).

A verificação dos produtos de trabalho aumenta consideravelmente a probabilidade de que sejam satisfeitos os requisitos do cliente, do produto e dos componentes de produto.

2.2.4. Desafios e Limitações

Ishikawa (1985) afirma que não é suficiente que os produtos atendam a padrões e especificações de qualidade. Os padrões não são perfeitos e não podem acompanhar as mudanças dos requisitos do cliente.

Charette et al. (2004) encontraram alguns problemas na aplicação do CMMI como falta de capacidade, baixo desempenho e falta de aderência aos processos.

Para Omran (2008), os principais tópicos que podem ser obstáculos às equipes de desenvolvimento em pequenas e médias empresas são:

- CMMI foca em gerenciamento de projeto e não em desenvolvimento de produto;
- CMMI ignora o uso de algumas tecnologias como a prototipação rápida;
- CMMI não incorpora a análise de risco como uma área de processo chave;
- CMMI é caro para ser utilizado;

Muitas empresas encontram problemas durante a implementação do CMMI em níveis altos de maturidade por causa de práticas complexas como medição e gerenciamento quantitativo ou o uso de modelos de desempenho efetivos para prever o rumo futuro de processos controlados (TAKARA, 2007).

Leeson (2009) indica alguns problemas na implementação do CMMI: 1) gerência sênior não envolvida no estabelecimento de objetivos, políticas e na necessidade de processos; 2) patrocinador não cumpre o seu papel, delegando autoridade; 3) grupo de desempenho da engenharia de software não gerenciado; 4) organização tem como foco atingir o nível de maturidade em vez da melhoria da qualidade dos seus produtos e serviços. Sobre as dificuldades encontradas pelas organizações durante a interpretação do CMMI, o autor cita a falta de uma visão global do modelo e o não entendimento da relação entre medição e monitoramento de projeto ou a diferença entre capacidade e níveis de maturidade.

Margarido et al. (2012) conduziram estudos de casos para verificar problemas que uma implementação do CMMI pode apresentar. Em um dos casos, os problemas foram decorrentes da evolução rápida para nível de maturidade 5 sem haver tempo suficiente para se ter ferramentas, processos e comportamento humano estáveis. Em outro caso, vários problemas foram relacionados a métricas e a resistência a mudanças. Nos três casos avaliados, o tempo para implementar um nível alto de maturidade do CMMI foi subestimado e o processo levou mais tempo para ser finalizado. Um problema recorrente foi o entendimento da natureza estatística do CMMI nível 4.

Os problemas relacionados à implantação do CMMI são resultados de uma implantação deficiente do mesmo e eles se tornam mais sérios se forem detectados tardiamente.

2.2.5. Considerações finais

É possível identificar alguns casos de fracasso na implantação do CMMI, geralmente nos níveis mais altos de maturidade. Entretanto, a grande maioria dos casos relatados na literatura é de sucesso e benefícios para a organização, resultando em uma melhora na performance de desenvolvimento de software.

2.3. *Design Rationale (DR)*

Muitos processos de desenvolvimento de software utilizados hoje em dia são orientados a artefatos, isto é, a ênfase está na geração e rastreamento dos artefatos de projeto intermediários (requisitos, especificações, protótipos e documentação) que conduzirão ao sistema final propriamente dito. Mas, o processo de desenvolvimento destes artefatos fica implícito, escondido em reuniões, cadernos de anotações, e-mails ou na memória dos programadores (SHUM; HAMMOND, 1994). Consequentemente, estas informações não ficam disponíveis e o que é pior, sua recuperação pode nem ser possível no momento que for necessário.

Geralmente, apenas se registram as informações relativas às decisões finais de determinada fase, pois os analistas consideram onerosa a tarefa de documentar cada uma das alternativas

investigadas. Entretanto, registrando-se e recuperando-se estas informações de maneira adequada, podem-se obter várias vantagens ao projeto: melhor conhecimento do projeto, maior facilidade de manutenção e comunicação da equipe de projeto, melhor possibilidade de reuso, maior facilidade de integração de um novo membro à equipe, bem como diminuir a possibilidade de se fazer uma tentativa de opção já descartada no passado.

Como se pode ver, as vantagens são inúmeras. A grande questão é como capturar estas informações com o mínimo de interferência às atividades do processo usual e também como recuperar, de maneira eficiente, tais informações. Neste ponto é que entra o estudo do DR, cuja ideia básica é a captura, representação e recuperação desse tipo de informação.

Neste capítulo, são descritos os principais conceitos relacionados ao DR. Inicia-se com a apresentação da própria definição do termo, seguida da descrição das formas de obtenção do DR, sua utilidade e formas de representação, captação e uso das informações. Ao final do capítulo, são apresentadas algumas ferramentas de suporte à utilização do DR.

2.3.1. Definição

Há vários trabalhos relacionados à análise de projeto e gerenciamento de decisão que citam o termo *Design Rationale* para descrever os registros que informam a razão dos analistas terem feito decisões específicas durante o processo de desenvolvimento. Entretanto, esta abordagem não é limitada à fase de projeto, e sim, é aplicada a todas as fases do desenvolvimento de software. Segundo Sauer (2003), como muitas pesquisas sobre *rationale* tiveram, no passado, seu foco em atividades de projeto, o termo *Design Rationale* é o que predominou e também é o que se utiliza mais frequentemente.

DR refere-se à documentação das alternativas, escolhas e decisões realizadas durante o processo de desenvolvimento de um projeto, bem como das razões de se ter tomado determinado rumo. Vários pesquisadores já apresentaram suas definições. A seguir, são mostradas algumas delas.

De acordo com Gruber e Russel (1991), *rationales* representam explicações dos relacionamentos entre estrutura, comportamento e funcionalidade de artefatos, podendo-se citar como uma determinada estrutura implementa uma funcionalidade, ou como um

determinado comportamento se forma através de uma estrutura. Eles também explicam o processo de tomada de decisão.

Para Souza et al. (1998), DR é uma representação da documentação que contém as razões e argumentos que fazem sentido a um artefato específico. Inclui tanto as razões de uma decisão de projeto, como a justificativa para elas, as alternativas consideradas e que foram descartadas, alterações avaliadas e os argumentos que levaram à decisão.

DR é a informação que explica por que um artefato é estruturado da maneira que é e tem o comportamento que tem (CONKLIN; BURGESS- YAKEMOVIC, 1995).

Segundo Hu et al. (2000), DR é a explicação da razão de um artefato ou alguma parte dele ter sido projetado de determinada maneira. Fazem parte as considerações, raciocínios, alterações e tomadas de decisão do processo de projeto de um artefato. Estas informações podem ser valiosas e até críticas a várias pessoas que lidam com o artefato.

Em (BURGE; BROWN, 2000), DR é definido como sendo as decisões tomadas durante o processo de projeto e as razões que levaram a tais decisões.

Todas as definições parecem concordar que o DR está relacionado às informações sobre a razão, considerações e justificativas de uma tomada de decisão, bem como as alternativas consideradas.

2.3.2. Formas de obtenção do DR

A obtenção de informações, visando à técnica do DR, ou seja, à captura e recuperação das informações necessárias pode ser realizada de várias maneiras. A utilização de uma determinada forma não exclui a existência de outras formas, ou seja, podem-se utilizar, em um mesmo sistema, alternativas distintas. As cinco principais formas existentes são:

- **Baseada em argumentação:** consiste no registro dos argumentos e questões que tenham surgido durante um projeto (CONKLIN; BEGEMAN, 1988); englobam questões levantadas, respostas alternativas a estas questões e argumentos contra e a favor a cada alternativa.

- **Baseada em histórico:** consiste no registro da sequência de eventos que ocorreram durante o projeto. Estas informações podem ser armazenadas como entradas do documento de projeto ou como um arquivo de mensagens de e-mails, entre outros (BURGE; BROWN, 1998).
- **Baseada em modelo de dispositivo:** contempla o registro do modelo de comportamento do dispositivo; um modelo do próprio dispositivo é utilizado para gerar explicações textuais e gráficas sobre como e porque um dispositivo funciona (GRUBER, 1990); é mais poderoso e específico ao domínio que o baseado em argumentação e histórico e é mais utilizado para diagnóstico (GARCIA; HOWARD, STEFIK, 1993).
- **Baseada em processo:** o registro do DR é integrado ao próprio processo de desenvolvimento de software que guia o formato do *rationale*.
- **Baseada em documentos ativos:** o DR é gerado previamente com base no conhecimento armazenado sobre o sistema. Para cada decisão que é tomada, o sistema compara a decisão tomada pelo usuário com a decisão que ele teria tomado baseado no conhecimento existente. Se as ações do usuário forem conflitantes com as recomendações do sistema, ele é avisado e poderá alterar sua decisão ao modificar algum critério da base de conhecimento (BURGE; BROWN, 1998).

Independentemente da forma escolhida, o DR pode auxiliar diversas áreas da engenharia de software. Algumas delas são discutidas no próximo item.

2.3.3. Utilidade

O DR consiste em informações variadas sobre o desenvolvimento de um determinado sistema. Estas informações podem auxiliar diversas atividades relacionadas direta ou indiretamente ao projeto (LEE; LAI, 1991):

- **Verificação:** o DR pode ser utilizado para verificar se o projeto satisfaz os requisitos do sistema e as intenções do projetista.
- **Avaliação:** o DR pode ser útil para avaliar projetos (e partes de projetos) e escolhas realizadas a fim de detectar inconsistências que podem afetar a qualidade do projeto. Pode fornecer auxílio durante todo o processo de projeto do software. É possível

verificar e avaliar decisões de projeto no momento que são tomadas, evitando a violação de uma restrição ou efeitos colaterais não permitidos.

- **Manutenção:** o *DR* pode ser utilizado para determinar quais escolhas foram feitas durante as atividades do projeto com a finalidade de localizar fontes de problemas ou indicar que mudanças devem ser feitas para modificar o projeto; além disso, rastreando as alternativas que já foram rejeitadas, o projetista pode evitar que se faça uma escolha já rejeitada anteriormente.
- **Reuso:** o *DR* pode ser útil na definição das partes do projeto que podem ser reutilizados e, em alguns casos, sugerir onde e como deve ser modificado para se adequar ao novo conjunto de requisitos. É especialmente importante que o projetista saiba por que as decisões foram tomadas.
- **Ensino:** o *DR* pode auxiliar no ensino de um projeto a novos integrantes, mostrando a razão de cada decisão tomada. Alguns sistemas de *DR* possibilitam ao usuário fazer perguntas sobre o projeto, o que é geralmente mais rápido e mais fácil do que analisar grande quantidade de documentação de projeto.
- **Comunicação:** o *DR* pode melhorar a disseminação de conhecimento sobre um projeto, tanto durante como depois do projeto inicial. Pode também detectar conflitos em ambientes com vários projetistas e diminuir o ciclo de revisão.
- **Documentação:** o *DR* pode auxiliar na documentação do projeto, complementando com informações importantes. Se estas informações estiverem guardadas em formato que o computador entende, podem ser utilizadas como parte de um sistema de geração de documentos personalizados, através da escolha de diferentes perspectivas. Por exemplo, o cliente deve precisar de um nível de detalhamento diferente de um projetista. Além disso, alguns sistemas permitem que o usuário faça perguntas sobre o projeto.

Apesar do grande número de vantagens que a utilização do *DR* pode oferecer, existem poucos exemplos de seu uso nas empresas.

A maior parte das pesquisas em *DR* tem como foco a sua representação, captura e uso. Este parece ser o ponto chave para determinar a sua utilização nas empresas, visto que estas

características estão diretamente relacionadas à determinação do custo-benefício em se utilizar esta tecnologia.

2.3.4. Representação das informações

A representação do *DR* varia de formal a informal, conforme mostra a Figura 7. Uma abordagem formal possibilita a utilização dos dados automaticamente pelo computador, mas nem sempre apresenta as informações em uma forma que as pessoas possam entender. Além disso, requer que os dados inseridos no sistema tenham um formato mais rigoroso. Uma abordagem informal utiliza dados em formatos que são facilmente gerados e entendidos pelas pessoas, mas não podem ser facilmente utilizados pelo computador.

Figura 7 - Representação do *DR* – de formal a informal

Fonte: Adaptado de Burge (2005)

Nos itens a seguir, estas formas de representação do *DR* são descritas com mais detalhes.

2.3.4.1. Representação Formal

Representações formais contêm informação em formato entendido pela máquina, tornando fácil a manipulação e a interpretação da informação, utilizando um computador. Entretanto, é

mais difícil de ser entendido pelas pessoas. O tipo de representação formal dependerá das operações que serão efetuadas.

Entre os exemplos de representações formais estão as regras embutidas em um sistema especialista e um sistema de aprendizado de máquina que utiliza rastreamentos de solução de problemas passados para solucionar problemas futuros.

Quanto mais formal for a representação dos *rationales*, mais serviços o sistema poderá oferecer. Entretanto, formalizar o conhecimento não é uma tarefa fácil. Lee (1997) sugere a formalização incremental: a ideia principal é transformar uma representação semiformal em uma formal. Consequentemente, *rationales* podem ser capturados sem muita sobrecarga ao sistema usual (pois eles são capturados de maneira semiformal), mas assim que são transformados em formais, podem ser utilizados para dar suporte a mais serviços computacionais.

2.3.4.2. Representação Informal

Representações são classificadas como informais quando elas capturam informações na forma gerada pelo projetista durante o projeto em vez de exigir que uma nova estrutura seja utilizada (BURGE, 2005). Os *rationales* são capturados em uma forma não estruturada.

Existem vários exemplos de representações informais, entre eles, fitas de áudio ou de vídeo, anotações, relatórios, e-mails, descrição de experimentos, etc.

2.3.4.3. Representação Semiformal

Representações semiformais são tipicamente na forma de argumentação.

Abordagens semiformais tentam se beneficiarem das vantagens das outras duas abordagens: oferecer certo poder computacional, mas em um formato que é entendido pelas pessoas.

2.3.5. Captura das informações

Uma das dificuldades em utilizar a tecnologia de *DR* é a captura das informações. O registro de todas as decisões tomadas bem como as rejeitadas, podem consumir muito tempo e ter um alto custo. No geral, quanto mais o processo de captura for intrusivo, maior a resistência encontrada. Além do tempo extra que será necessário a esta atividade, ela pode desviar o foco do projetista de sua tarefa usual de projeto. Por outro lado, não se pode esperar que o projetista saiba extrair estas informações do processo de projeto de software sem o auxílio de uma ferramenta que o guie, pois é uma tarefa cognitiva difícil (GRUBER; RUSSEL, 1991).

A captura refere-se à obtenção e registro do *DR*, tanto durante como após o processo de projeto. A quantidade de dados que é capturada varia muito. Alguns sistemas optam por registrar o máximo de informações possível, enquanto outros utilizam uma abordagem mais focada.

Há vários métodos propostos, incluindo o registro de reuniões e discussões do projeto em fita de vídeo até métodos em que os projetistas devem documentar manualmente cada decisão feita. Os seguintes métodos para captura do *rationale* têm sido estudados:

- **Reconstrução:** as informações são capturadas fora do processo de projeto do software (geralmente depois). A captura pode ser feita através da análise de seu próprio conhecimento, através de entrevistas com as pessoas envolvidas no projeto ou capturando o raciocínio em formato não processado, como vídeo e então transformando estes dados em um formato mais estruturado. Outra maneira é realizar a engenharia reversa com o intuito de inferir um plano sobre o projeto do artefato a um conhecimento geral de funções e comportamentos de dispositivo (LEE, 1997). A vantagem é ser uma abordagem não intrusiva e a desvantagem é que a captura do *rationale* pode não ser precisa ou completa.
- **Subproduto metodológico** (*methodological byproduct*): segundo Lee (1997), a informação aparece naturalmente durante o processo de projeto do software, através da utilização de uma metodologia adequada. Esta abordagem é atrativa porque o usuário se beneficia do método e o *rationale* é obtido a um custo relativamente baixo. Para Burge (2005), a definição de Lee não é tão clara: alguns processos de projeto de software baseados em ferramentas que automaticamente realizam a captura do *DR* podem ser considerados como produtores de *rationale* como um subproduto metodológico. Outra maneira seria utilizar uma metodologia que force a captura do *DR*.

- **Aprendiz:** o sistema acompanha as ações tomadas pelo projetista e faz perguntas quando não entende ou não concorda com uma ação. O *rationale* é, de certa maneira, pré-gerado. Se as ações do projetista forem de acordo com o que o sistema esperava, o raciocínio gerado pelo sistema é salvo. Segundo Lee (1997), a vantagem desta abordagem é que esta interação traz benefícios tanto para o usuário que está inserindo a informação como para o sistema. Além disso, as informações são capturadas no momento que estão sendo utilizadas diminuindo a probabilidade de obter informações imprecisas ou incompletas.
- **Geração Automática:** em sua forma mais simples, as informações são geradas a partir de um histórico de execuções (LEE, 1997). Esta abordagem tem a vantagem de obter as informações a um baixo custo e de ter a possibilidade de manter uma base de conhecimento consistente e atualizada. Entretanto, há um custo inicial alto para analisar as informações relevantes e construir os *rationales*. Além disso, analisando o método mais profundamente, percebe-se que muitas questões ligadas à pesquisa sobre aprendizado de máquina devem ser resolvidas: que partes do rastreamento da solução do problema devem ser guardadas, como salvar, como inferir *rationales* a partir destas informações, como determinar sua importância e como adaptar estas informações à situação atual.
- **Historiador:** pessoa ou programa realiza um rastreamento de todas as ações feitas durante o processo de projeto do software. Este método é semelhante ao aprendiz, com exceção de que o sistema não faz sugestões. É semelhante à geração automática, exceto que o raciocínio é especificamente registrado durante o projeto, e não gerado posteriormente (CHEN et al., 1990).

A Tabela 3 mostra um resumo destas técnicas de captura levando-se em conta o momento em que a informação é capturada e o nível de interação com o usuário. Métodos que interagem muito com o usuário tendem a ser mais intrusivo do que os que não interagem. É claro que isso não necessariamente significa uma desvantagem da técnica caso esta interação seja vista como útil ou se é consistente com o processo normal do projeto do software.

A captura do *DR* não tem utilidade se não houver um modo de acessá-lo. Este é o assunto do próximo item.

Tabela 3 - Quadro comparativo dos métodos de captura do *DR*

	Momento da Captura		Interação com usuário	
	Depois do projeto	Durante o projeto	Baixa	Alta
Reconstrução	✓	✓	✓	
Subproduto Metodológico		✓		✓
Aprendiz		✓		✓
Geração Automática	✓		✓	
Historiador		✓	✓	

Fonte: Autora

2.3.6. Uso das informações

O *DR* pode ser utilizado de diversas maneiras. Alguns sistemas somente suportam a captura das informações. O modo como é utilizado após esta captura é de responsabilidade do projetista. Outros sistemas suportam a recuperação da informação e também oferecem a possibilidade de verificar consistência e completude no *rationale* ou no projeto. Estes sistemas podem ser classificados como “iniciativa do usuário” ou “iniciativa do sistema”, dependendo em quem inicia o processo. Em um sistema de “iniciativa do usuário”, o usuário decide quais as partes do *DR* deseja examinar e quando e quem irá acessar. Em um sistema de “iniciativa do sistema”, por outro lado, o sistema que decide quando e como apresentar determinadas partes do *DR*. Estes sistemas precisam ter conhecimento suficiente para realizar decisões inteligentes, apresentando as informações de maneira não intrusiva ao usuário (LEE, 1997).

Uma questão crítica associada ao acesso das informações é a indexação inteligente dos dados. Os projetistas se sentirão interrompidos se a resposta do sistema for lenta, não importando o quanto este tempo é apropriado ao tamanho e complexidade da base de conhecimento.

Outra característica muito útil oferecida por alguns sistemas é a habilidade de fazer questionamentos sobre o projeto ou sobre o *rationale*. É possível, por exemplo, pedir explicações sobre a razão do valor de um parâmetro ter sido alterado.

Outro uso do *DR* é o suporte à colaboração, usando o *rationale* como um meio de comunicação entre diferentes pessoas ou equipes de um projeto.

2.3.7. Ferramentas de suporte

Há uma tendência à integração das ferramentas de *DR* com outras representações de projeto, considerando o sistema de *DR* como sendo uma extensão do sistema de projeto de software (HU et al., 2000). Esta incorporação ao processo já em uso tem o intuito de facilitar o aprendizado do desenvolvedor de como acessar e usar a ferramenta.

Segundo Burge (2005), uma ferramenta desenvolvida para dar suporte ao uso do *DR* deve ter as seguintes características:

1. Integração com o ambiente de desenvolvimento. Os projetistas não devem ser obrigados a abrir uma ferramenta adicional somente para criar e acessar os *rationales*. As informações precisam estar disponíveis nas ferramentas já em uso.
2. Associação do *rationale* com artefatos de desenvolvimento. Deve haver uma maneira de associar explicitamente o *rationale*, particularmente as alternativas, com o código que os implementa.
3. Apresentação “automática” do *rationale*. O usuário deve receber uma mensagem avisando que existe *DR* disponível quando estiver trabalhando com um código que possui esta informação. Não se deve ter a necessidade de deliberadamente verificar para saber se existe ou não *rationale* associado com o código que eles estão trabalhando.
4. Visualização dos argumentos do *rationale*. O *DR* deve ser exibido em um formato hierárquico, tirando proveito da estrutura de argumentação.
5. Verificação automática do *rationale*. O status do *rationale* (erros e avisos gerados durante inferências) deve ser atualizado automaticamente quando o *rationale* é adicionado, removido ou modificado.
6. Interface de consulta do *rationale*. Deve haver uma maneira de procurar facilmente por uma informação específica.

A existência de uma ferramenta de boa usabilidade para captura, representação e recuperação do *rationale* é um motivador para a utilização das técnicas de *DR*.

2.3.8. Abordagens

Uma grande variedade de abordagens ao DR tem sido proposta. Segundo Stumpf (1997), o foco está em três modelos: Questions Options and Criteria, Graphical Issue Based Information System e Decision Representation Language. Estes modelos são descritos a seguir.

- ***Questions Options and Criteria (QOC)***

Desenvolvido por MacLean et al. (1991) em “Questions, Options and Criteria: Elements of Design Space Analysis”, este modelo destaca a exploração de um espaço de alternativas do projeto, o espaço do projeto e as escolhas realizadas. O método desenvolvido inclui três tipos de nós: “Questões, Opções e Critério”. As questões são os problemas chave a resolver, as opções são as alternativas levantadas para resolver os problemas identificados, os critérios justificam as opções existentes, e finalmente, os argumentos são usados para conduzir a discussão sobre os demais elementos.

Apresenta uma estrutura baseada no processo como as alternativas são geradas e avaliadas. Esta representação possibilita a representação explícita de um espaço estruturado de alternativas de projeto e as considerações que levaram a sua escolha. Dentro das opções de espaço de projeto há possíveis respostas às perguntas (JARCZYK et al., 1992).

- **Graphical Issue Based Information System (gIBIS)**

Descrito em “gBIS: A Hypertext Tool for Exploratory Policy Discussion” (CONKLIN; BEGEMAN, 1988). É uma adaptação do IBIS - Issue Based Information System (KUNS; RITTEL, 1970) para argumentação de projeto, estendendo o vocabulário e adicionando uma representação gráfica.

gIBIS é utilizado para registrar ideias e relacionamentos durante discussões de projeto enquanto elas estão ocorrendo. Apresenta uma estrutura sobre como as questões são discutidas, mas não como o problema é resolvido, como alternativas são extraídas e avaliadas

ou como se chega a um consenso. Então, diferentemente do QOC, gIBIS coloca mais ênfase na representação do processo pelo qual as decisões são feitas.

Sua grande vantagem é que o *DR* pode ser capturado de maneira informal e em um estágio inicial do processo, assegurando que as questões do projeto estão entendidas. Entretanto, pelo formato de sua representação, IBIS não pode fazer uso de uma exploração estruturada dos assuntos. O controle fica muito mais nas mãos do projetista e demais pessoas envolvidas que devem investigar maneiras factíveis de resolver cada problema.

A estrutura do método consiste de três tipos de nós e oito tipos de conexões possíveis, formando uma rede semiestruturada de raciocínio das decisões de projeto.

Nós:

- Questões: problemas em discussão;
- Ideias: possíveis soluções para os problemas;
- Argumentos: opiniões favoráveis ou não às soluções levantadas.

Conexões:

- Suporta, opõe-se a, substitui, responde a, generaliza, especializa, pergunta, proposto por.

A Figura 8 mostra a representação dos nós que fazem parte do método.

Figura 8 - IBIS: Representação dos nós

Fonte: Adaptado de Kuns e Rittel (1970)

- **Decision Representation Language (DRL)**

O foco deste trabalho é para o desenvolvimento de uma ferramenta baseada em conhecimento chamada SIBYL para representar decisões de *rationale*.

É um modelo de argumentação bem mais completo, envolvendo sete tipos de nós principais (alternativa, objetivo, reclamação, questão, procedimento, grupo e ponto de vista) de uma hierarquia com 26 tipos de nós e mais de 20 relacionamentos.

Apesar de ter também uma notação semiformal, em comparação com outras notações, ela suporta mais informação na parte formal da descrição da informação.

Lee e Lai (1991) defendem que uma representação deve suportar uma variedade de tarefas de projeto, tais como responder questões sobre o progresso do projeto, as alternativas geradas, as avaliações que levaram à escolha de determinadas alternativas e a possível transferência de conhecimento a projetos futuros ou outras pessoas. DRL foi desenvolvido para suportar todas estas questões.

Sua ênfase é gerenciar os elementos qualitativos das tomadas de decisão e do gerenciamento de suas dependências. Portanto, é mais um sistema de gerenciamento de *DR*.

Estes modelos são baseados em argumentação e possuem notação semiformal. Eles diferem na extensão de características que almejam capturar e nas funções que suportam pelo modo como estes modelos são utilizados em uma representação. A melhor escolha dependerá da aplicação e objetivos específicos do ambiente analisado.

2.3.9. Modelos de representação

Neste item, apresentam-se alguns modelos de representação do DR.

2.3.9.1. REMAP

O REMAP (REpresentation and MAintenance of Process knowledge) é um modelo conceitual que estende o IBIS para incluir elementos de requisitos e de projeto para processar conhecimento durante a engenharia de requisitos (RAMESH; DHAR, 1992). Tem o objetivo de melhorar a qualidade por meio da propagação de restrições e validações de hipóteses através do diagrama, ilustrando os benefícios do rastreamento dos requisitos pelos elementos de projeto e *rationale*.

A Figura 9 mostra a base do modelo conceitual do REMAP. A parte argumentativa referente às definições de projeto pode ser representada utilizando-se as primitivas originais do IBIS (porção dentro do retângulo pontilhado). A **decisão** resolve uma ou mais **questões** através da seleção de uma **posição** dentro do conjunto de **posições** possíveis. A **decisão** sobre uma questão durante o processo de projeto pode também levar a alterações nos requisitos, caracterizando a natureza evolutiva do processo de projeto (RAMESH; DHAR, 1992).

Neste modelo, as **hipóteses** associadas a argumentos são representadas explicitamente, possibilitando a existência de mecanismos para avaliar a aplicabilidade dos argumentos em uma dada situação e, portanto, auxiliando na avaliação das posições.

Um protótipo do REMAP foi construído para demonstrar como requisitos, elementos de projeto, decisões de projeto, restrições e argumentos são capturados em um gráfico (DUTOIT et al., 2006). A construção do protótipo foi baseada em ConceptBASE, uma implementação de alto nível da linguagem de modelagem Telos.

Segundo Ang et al. (1998), uma limitação do modelo REMAP é que o suporte à fase de engenharia de requisitos é para um momento em que os requisitos já estão definidos e as consequências de diferentes projetos já foram analisadas. Entretanto, geralmente este não é o cenário real. Os analistas normalmente precisam extrair os requisitos dos usuários.

Figura 9 - Modelo Conceitual do REMAP

Fonte: Adaptado de Ramesh e Dhar (1992)

2.3.9.2. Inquiry Cycle

Inquiry Cycle é um modelo cíclico para análise de requisitos que consiste de três atividades principais, expressão, discussão e comprometimento, que são repetidos em sequência. Na atividade de expressão, as informações relacionadas aos requisitos são preparadas, incluindo não só os documentos de requisitos, mas também informação específica do domínio, cenários e objetivos da corporação. Durante a discussão, requisitos são examinados em reuniões formais e são realizadas anotações individuais sobre os requisitos. A atividade de comprometimento inclui a tomada de decisões baseadas nas discussões, como requisições de mudança, acordos sobre terminologia e busca de informações desconhecidas (TAKAHASHI et al., 1996). É um modelo hipertexto, pois as informações registradas e compartilhadas nas

discussões de requisitos são primariamente textuais e, portanto, informais, embora apresente inter-relacionamentos bem definidos (POTTS et al., 1994).

A ênfase do modelo é em investigação e interrogatório durante o processo de produção, principalmente sobre a origem desta informação e quando ela é necessária (ANG et al., 1998).

A Figura 10 mostra o modelo Inquiry Cycle. A **documentação dos requisitos**, consistindo de requisitos, cenários e outras informações é discutida através de **questões, respostas e razões**. Escolhas podem levar à requisição de **mudanças**, modificando, então, a documentação dos requisitos.

Figura 10 - Modelo Inquiry Cycle

Fonte: Adaptado de Potts et al. (1994)

O modelo tem três fases (POTTS et al., 1994):

- Documentação dos requisitos – requisitos propostos são anotados em nós separados do hipertexto;
- Discussão dos requisitos – requisitos propostos são desafiados vinculando anotações a eles;

- Evolução dos requisitos – se uma requisição de mudança for aprovada, os requisitos são refinados.

Ferramentas que suportam vínculo de hipertexto são adequadas para suportar o modelo. O protótipo de uma ferramenta hipertexto, chamada Tuiqiao, foi desenvolvido por Potts e Takahashi (1993). É uma ferramenta monousuária para anotação de requisitos implementada em Hypercard.

Robinson e Pawlowski (1998) desenvolveram uma ferramenta chamada DealScribe que suporta a troca de ideias sobre requisitos. Ela fornece uma interface web simples para registro dos requisitos, um formalismo para definição de produtos de requisitos e processos orientados a objetivos e suporte automatizado para análise e monitoramento de requisitos. DealScribe pode rastrear requisitos digitados e também a situação de análises associadas.

Uma das limitações do modelo Inquiry Cycle é que ele se concentra na análise de requisitos, mas não mostra como a informação originada do modelo pode ser integrada com os tradicionais documentos e diagramas CASE (Computer-aided Software Engineering) (ANG et al., 1998).

2.3.9.3. SCRAM

Scenario Based Requirements Analysis Method (SCRAM) utiliza *rationale* para melhorar a extração e validação de requisitos com os usuários. O foco é melhorar a qualidade dos requisitos extraindo mais informações e mais tipos de informações e tornando o DR de requisitos visível aos usuários (SUTCLIFFE; RYAN, 1998).

Em um trabalho anterior, Sutcliffe (1995) observou que a disponibilidade do DR levava os usuários a fazer mais questionamentos durante as sessões de elicitação de requisitos.

O método SCRAM é baseado em quatro técnicas para captura e validação de requisitos (SUTCLIFFE; RYAN, 1998):

- 1- Uso de protótipos ou demonstradores de conceito: um conceito chave é produzir um artefato com que os usuários possam reagir.

- 2- Cenários: o artefato está situado em um contexto de uso, auxiliando os usuários a relacionarem o projeto ao seu contexto de trabalho ou tarefa.
- 3- *Design rationale*: o raciocínio do projetista é intencionalmente exposto aos usuários para encorajar sua participação no processo de decisão. Utiliza-se a notação QOC – Question, Options and Criteria (MACLEAN et al., 1991).
- 4- Resumo: os requisitos são resumidos em um quadro para identificar dependências e prioridades.

Em um dos experimentos, Sutcliffe e Ryan (1998) observaram que os usuários acharam a notação QOC difícil de utilizar, embora dessem um retorno positivo sobre o uso de diagramas de *DR*.

2.3.9.4. SEURAT

O sistema Software Engineering Using RAtionale (SEURAT) foi desenvolvido para explorar o uso de *rationale* para dar suporte ao desenvolvimento de software e, em particular, manutenção de software (BURGE; BROWN, 2004).

O sistema SEURAT se integra com o ambiente de desenvolvimento interativo Eclipse dando suporte à captura e ao uso de *DR* (BURGE, BROWN, 2008). A integração torna a captura e o uso melhor, pois não força o desenvolvedor a sair de seu ambiente usual para executar uma ferramenta adicional.

Os dados são armazenados em um banco de dados relacional. O *DR* é registrado utilizando-se uma interface de edição ou pode ser importado de documentos Microsoft Word cujos comentários foram utilizados para definição dos *rationales*.

Além de apresentar a informação quando necessária, suporta também o rastreamento de requisitos e avaliação de impacto (BURGE, BROWN, 2008). SEURAT suporta tanto a exibição como a inferência do *DR* para apontar questões não resolvidas e inconsistências (BURGE; BROWN, 2004).

Bracewell et al. (2009) salientam que a ferramenta SEURAT possibilita o gerenciamento de uma grande quantidade de *rationales* em um único gráfico. O usuário pode escolher o que

mostrar em um dado momento, expandindo ou colapsando determinado nível. Adicionalmente, é possível obter uma cópia abrangendo todos os dados, imprimindo a estrutura hierárquica expandida. Entretanto, estas facilidades provocam dois problemas. Primeiramente, pode ser muito inadequado representar espaços de decisão ou de projeto que não possuem uma estrutura em árvore. Outro problema é a falta de clareza na apresentação dos dados, pois é difícil determinar que níveis podem ser fechados e quais podem ser expandidos para avaliar com precisão uma questão em particular. É fácil perder um argumento crítico simplesmente porque ficou escondido em um ramo colapsado ou rolou para fora da tela na expansão de ramos menos importantes.

2.3.9.5. *Sysiphus*

Sysiphus é um ambiente para modelagem e colaboração distribuída que fornece um *framework* uniforme para modelos de sistema, artefatos de colaboração e modelos organizacionais. Encoraja os participantes a fazer uma comunicação explícita e a se atentar aos *stakeholders* relevantes (BRUEGGE et al., 2006). A parte principal do Sysiphus é um software de código aberto (THURIMELLA; BRUEGGE, 2007).

Modelos de sistema, de colaboração e organizacionais são representados como um grafo simples guardado em um repositório compartilhado. Os elementos de *rationale* são considerados elementos de primeira classe e são acessados da mesma forma que os elementos de modelo do sistema. Alterações realizadas por usuários são propagados sincronamente para outros usuários que estejam trabalhando no mesmo projeto, possibilitando a colaboração síncrona (DUTOIT et al., 2006).

Sysiphus fornece modelos de sistema para várias atividades de desenvolvimento, incluindo análise de requisitos, projeto de sistemas, implementação, teste e manutenção. Os modelos incluem requisitos funcionais e não funcionais, casos de uso, diagramas de componentes e de classe, diagramas de sequência, diagrama de estados e casos de teste (BRUEGGE et al., 2006).

Sysiphus adota uma abordagem semelhante à SEURAT para diminuir o limiar da captura de DR e aumentar o estímulo em curto prazo do desenvolvedor. Entretanto, Sysiphus foca no ambiente de modelagem e colaboração enquanto o SEURAT foca no ambiente de desenvolvimento (DUTOIT et al., 2006).

2.3.9.6. NFR Framework

O NFR Framework é um método para rastrear os requisitos não funcionais relevantes para cada decisão, alternativas avaliadas e interação entre requisitos não funcionais (CHUNG et al., 1996). Requisitos não funcionais são tratados como objetivos a serem atingidos. Esses objetivos são refinados e esclarecidos decompondo-se em sub-objetivos. Objetivos e sub-objetivos são representados como nós em um grafo de objetivos.

NFR Framework fornece dois tipos de decomposições (DUTOIT et al., 2006):

- Decomposição AND: Um objetivo pode ser decomposto em sub-objetivos e todos precisam ser atingidos para que o objetivo pai seja alcançado.
- Decomposição OR: Um objetivo pode ser decomposto em sub-objetivos alternativos e se qualquer um deles for satisfeito, o objetivo pai também é alcançado.

O NFR Framework possui como principal representação, o grafo de interdependência de *softgoals* ou SIG (*Softgoal Interdependency Graph*). Não existe uma tradução consolidada para *softgoal*. Seu significado está relacionado ao termo *satisfied* em inglês, que corresponde à satisfação em um determinado nível (CHUNG et al., 2000).

Os *softgoals* são representados graficamente como nuvens e conectados entre si através de relações de interdependência. Um *softgoal* pode contribuir positivamente ou negativamente para o atendimento de outro *softgoal*. Esta contribuição pode ser de quatro tipos: MAKE, HELP, HURT e BREAK. A ordem de classificação dessas categorias é a seguinte: MAKE > HELP > HURT > BREAK.

Os *softgoals* podem ser relacionados um ao outro por três contribuições: AND (representado por um arco simples), OR (representado por um arco duplo) e um refinamento (somente um filho). Além disso, um *softgoal* pode ser considerado crítico, sendo acompanhado então pela seguinte marcação: !. A

Tabela 4 mostra os elementos principais do NFR Framework.

Tabela 4 - NFR Framework – elementos principais

Representação	Nome	Descrição
Softgoal	Representação de um <i>softgoal</i> . Representa algo incerto ou nebuloso.
Operationalizing Softgoal	Operacionalização – uma possível solução para satisfazer um <i>softgoal</i>
Claim Softgoal	Observação ou argumentação – auxilia revisões futuras
MAKE Contribution	Totalmente positiva
HELP Contribution	Parcialmente positiva
HURT Contribution	Parcialmente negativa
BREAK Contribution	Totalmente negativa
AND Contribution	E
OR Contribution	Ou
Criticality	Elemento crítico

Fonte: Adaptado de CHUNG et al. (1996)

A NFR Assistant é uma ferramenta de suporte ao NFR Framework que auxilia o desenvolvedor a navegar, consultar, associar e estender catálogos e construir o histórico de informações (CHUNG et al., 1996).

O NFR Framework preenche a lacuna entre requisitos e projeto em relação aos requisitos não funcionais. Entretanto, é deficiente de uma forma de suporte quantitativa que preserve os ideais do *framework* (AFFLECK; KRISHNA, 2012).

2.3.10.Desafios e limitações

Embora o uso do DR seja reconhecido como um dos passos mais promissores para avançar o estado da arte da manutenção e projeto de arquitetura de software (BOSCH, 2004), não se observa a sua utilização em larga escala na indústria. O livro de DUTOIT et al. (2006) apresenta diversos trabalhos no assunto. Entretanto, há muita resistência sobre a utilização do DR na prática. Para LIANG et al. (2010), os obstáculos que impedem os usuários de utilizar DR são basicamente dois: o custo-benefício do gerenciamento do DR e a preocupação de se perder propriedades intelectuais através da documentação explícita de um conhecimento.

LIANG et al. (2010) lembram que o conhecimento racional de um projeto geralmente fica apenas na cabeça dos desenvolvedores e as técnicas de gerenciamento deste conhecimento fornecem pouco suporte para manipular este conhecimento implícito e complexo. Por isso, a primeira questão que surge é como capturar este tipo de conhecimento. Além disso, a dificuldade aumenta para projetos grandes e complexos.

A captura do DR através de documentos textos lineares é um desafio por causa da natureza complexa e rápida das atividades do projeto (AURISICCHIO; BRACEWELL, 2013). Diagramas facilitam o processo cognitivo humano, oferecendo uma estruturação aos pensamentos dos desenvolvedores.

Falessi (2013) aponta algumas razões da baixa adesão ao uso de DR na prática:

- Tempo e benefício tardio: o período em que as decisões de projeto são realizadas geralmente é crítico para o sucesso do projeto como um todo. É um período de muita pressão e, por isso, a documentação do *rationale* é visto como menos importante.
- Previsão da informação: as pessoas que produzem o DR geralmente não são as mesmas que o utilizam. Por isso, quem captura o DR precisa prever que tipo de informação será útil para outras pessoas.
- Sobrecarga: apesar de já existirem várias técnicas de DR, elas geralmente focam em maximizar os benefícios de quem a utiliza em vez de minimizar o esforço de quem produz a informação.

- Benefícios incertos: tomadores de decisão nem sempre têm certeza de como o DR dará suporte às suas atividades.
- Falta de motivação: consequência da ausência de benefícios diretos à pessoa que captura a informação ou a falta de interesse pessoal. Especialistas podem não estar interessados em deixar disponível de maneira explícita o seu conhecimento.
- Falta de maturidade nas ferramentas: poucas ferramentas disponíveis e a maioria ainda são imaturas.
- Inconsistências potenciais: o documento de DR e o projeto precisam estar atualizados e alinhados para evitar potenciais inconsistências quando o projeto é modificado ou quando as decisões mudam.

Horner e Attwood (2006) pesquisaram sobre as barreiras do uso do DR e as dividiram em quatro tipos de limitações: cognitiva, captura, recuperação e uso. As limitações cognitivas originam-se nas limitações de processamento de informação humana que tornam praticamente impossível a captura do DR de forma exaustiva. As limitações de captura envolvem a necessidade de coletar o *rationale* com o seu contexto, falta de incentivo para coletar o DR, o custo da captura contra os benefícios possíveis e se o DR coletado pode ser um risco para o indivíduo (expondo uma decisão errada) ou para a corporação (possivelmente questões de responsabilidade). Limitações de recuperação referem-se a determinar qual o conteúdo deve ser recuperado e qual o método de recuperação. Finalmente, as limitações de uso referem-se à deficiência dos métodos para avaliar quão efetivo o DR se apresenta.

2.3.11.Considerações finais

Há muitas pesquisas sobre DR e uma lista grande de possíveis benefícios para a sua utilização. Entretanto, ainda há muita resistência para a sua utilização na prática.

Para definição das melhores práticas de *DR*, deve-se procurar seguir os seguintes princípios:

- Encontrar maneira de capturar informações de maneira menos intrusiva possível.

- Procurar modelos cujos benefícios possam ser percebidos durante o processo de projeto do software ou logo após o seu término, aproximando as pessoas envolvidas (equipe que gera as informações e equipe que se beneficia das mesmas).
- Definir modelos que auxiliem a resolução do problema em questão, auxiliando o raciocínio, gerando benefícios também para quem captura a informação.
- Utilizar ferramentas adequadas que suportem o processo.

Além disso, é importante ter uma maneira de incentivar e lembrar os desenvolvedores sobre o seu uso.

3. MÉTODOS ÁGEIS E CMMI: ANÁLISE

Sutherland, um dos criadores do Scrum, afirma que Scrum e CMMI juntos produzem uma combinação mais poderosa de adaptabilidade e previsibilidade se comparado a cada um isoladamente (SUTHERLAND et al., 2007). CMMI pode ajudar empresas ágeis a institucionalizar métodos ágeis de maneira mais consistente e também a entender que processos endereçar.

Neste capítulo, é realizada uma análise sobre os Métodos Ágeis e o CMMI-DEV. No item 3.1, é realizada uma breve descrição de trabalhos semelhantes, enfatizando a diferença com esta pesquisa. No item 3.2, foi realizado um estudo sobre a aderência das práticas ágeis a cada uma das áreas de processo do CMMI-DEV versão 1.3. No item 3.3 apresenta-se uma entrevista sobre métodos ágeis e qualidade de software, realizada com profissionais de empresas brasileiras.

As informações apresentadas neste capítulo serão de fundamental importância para a proposta que é apresentada no Capítulo 4.

3.1. Trabalhos correlatos

Silva et al. (2015) realizou uma revisão sistemática sobre a combinação do uso de CMMI com desenvolvimento de software ágil e verificou que a discussão e pesquisa sobre o tema é relevante e atual, devido ao número de estudos encontrados. A possibilidade de usar CMMI com desenvolvimento ágil é manifestada dos dois lados.

A última versão do CMMI-DEV (CMMI-DEV V1.3, 2010) incorpora algumas sugestões para aplicar o modelo em ambientes com metodologia ágil. Do lado ágil, o grande número de artigos publicados em eventos de desenvolvimento ágil também sinaliza que a comunidade ágil está aberta à aproximação com CMMI.

Vários trabalhos realizaram a análise sobre compatibilidade entre CMMI e métodos ágeis e a grande maioria acredita que os conceitos, a princípio antagônicos, podem conviver harmoniosamente (GLAZER et al., 2008; GANDOMANI; HAZURAZULZALIL, 2013; TURNER; JAIN, 2002; FRITZSCHE; KEIL, 2007; MARÇAL et al., 2007). A grande maioria

desses trabalhos tem o foco na categoria Gerenciamento de Projetos do CMMI. Uma das conclusões de SILVA et al. (2015) é que as metodologias ágeis sozinhas não foram suficientes para se obter o nível de qualidade desejado. Foi necessária a utilização de práticas adicionais. Salientam também a importância da documentação e evidências, necessárias para o modelo formal de melhoria do processo.

Marçal (2009) analisa a aderência do Scrum em relação ao CMMI, especificamente no que diz respeito aos processos de gerenciamento de projetos. Na parte da pesquisa em campo, pode verificar que a utilização destas duas abordagens é uma realidade nas empresas brasileiras. Em seguida, define um processo de gestão ágil de projetos, chamado Scrummi, construído a partir de extensões do Scrum para ficar aderente às áreas de processo de gerenciamento de projeto do CMMI. Em seu trabalho, não considerou as demais áreas de processo.

O trabalho de Lina e Dan (2012) também tem o foco no aspecto de gestão de projeto. Apresentam o relacionamento entre métodos ágeis e CMMI e os benefícios de se utilizar as duas abordagens para melhorar a eficiência em pequenas e médias empresas. O foco é definir um *framework* de gestão de projeto.

Zanatta (2004) propõe uma extensão de um método ágil para a gerência e desenvolvimento de requisitos visando adequação ao CMMI. O modelo, chamado xScrum, é uma extensão do método Scrum para adequação ao modelo CMMI nas área de processo REQM e RD.

Diaz et al. (2009) fornecem um modelo e descrevem um estudo de caso de maneira detalhada, mas concentram somente nas diretrizes do CMMI nível 2. Além disso, apresentam poucas práticas adicionais para dar suporte a todos os objetivos do CMMI nível 2 de maneira apropriada.

Fritzsche e Keil (2007) apresentam um modelo mais completo, explorando todos os níveis de maturidade do CMMI, utilizando práticas do Scrum e do XP. Entretanto, apenas mostram quais práticas e objetivos estão em conflito, mas não apresentam uma solução de como complementar a abordagem ágil.

Lukasiewicz e Miler (2012) desenvolveram um modelo de referência chamado “C-S model” com o objetivo de mapear práticas específicas dos níveis 2 e 3 de maturidade do CMMI com as atividades descritas na metodologia Scrum. Segundo os autores, Scrum cobre apenas parte

das práticas dos níveis 2 e 3. Não demonstram uma cobertura abrangente de todas as práticas dos níveis citados.

O Testing Maturity Model Integration (TMMI), proposto pela TMMi Foundation (2012), é um modelo complementar ao CMMI nas áreas de Verificação e Validação, relacionado a testes.

O AgileQA-RM foi construído tendo como base os modelos de maturidade CMMI e o MPS.BR e metodologias ágeis (SILVA et al., 2014). Consiste de 5 níveis de maturidade, 18 áreas de processo, 43 resultados esperados e 31 produtos de trabalho. A Figura 12 ilustra os níveis de maturidade do modelo.

Figura 11 – Níveis de maturidade do AgileQA-RM

Fonte: Adaptado de Silva et al. (2014)

Patel e Ramachandran (2009) propuseram um modelo de maturidade composto por cinco níveis: inicial, explorado, definido, melhorado e suportado. A Figura 12 mostra uma visão de alto nível do modelo. O modelo tem o objetivo de melhorar a metodologia de desenvolvimento de software ágil e impulsionar os princípios e objetivos ágeis tais como diminuir custos e melhorar a satisfação do cliente e a qualidade do software.

Figura 12 – Agile Maturity Model (AMM)

Fonte: Adaptado de Patel e Ramachandran (2009)

É possível verificar que há várias pesquisas envolvendo métodos ágeis e CMMI. Entretanto, não se observa uma grande adesão aos modelos propostos, nem na academia e nem na indústria. Para Engdashed *et al.* (2016), o problema principal é que os modelos apresentados tiveram o objetivo de ser dominante sobre os demais em vez de colaborarem uns com os outros para um melhor resultado.

Além disso, quase todos os trabalhos não tiveram continuidade. Os autores trabalharam em outros assuntos após a publicação destas pesquisas. Por isso, a utilização de metodologias ágeis com as diretrizes do CMMI-DEV ainda merece estudos.

Finalmente, em nenhum deles foi encontrado o foco nas áreas de processo da categoria Suporte do CMMI-DEV.

3.2. Compatibilidade dos Métodos Ágeis com CMMI

Boehm e Turner (2003(b)) acreditam que projetos bem-sucedidos necessitam tanto de agilidade como de disciplina. Disciplina sem agilidade leva a burocracia e estagnação. Agilidade sem disciplina leva a um entusiasmo sem controle e sem resultados.

Segundo Anderson (2005), é possível desenvolver um processo completo de ciclo de vida verdadeiramente ágil que satisfaça os requisitos para todos os 5 níveis do modelo CMMI. Outros pesquisadores acreditam que há pouca aderência dos métodos ágeis para o nível 5 do CMMI. Organizações podem melhorar a produtividade, qualidade e previsibilidade à medida que atingem os níveis do CMMI-DEV (GARZÁS; PAULK, 2013).

Visando ao estudo da compatibilidade e integração dos métodos ágeis às áreas de processo do CMMI, este item tem o objetivo de verificar quais áreas de processo do CMMI terão um esforço maior de adaptação. A Tabela 5 mostra a simbologia e classificação utilizada.

Tabela 5 – Classificação Métodos Ágeis x CMMI

Símbolo	Significado
[-]	Conflitante
[0]	Não suportado
[+]	Parcialmente suportado
[++]	Suportado
[+++]	Totalmente suportado

Fonte: Autora

“Totalmente suportado” indica que as práticas dos métodos ágeis satisfazem a maior parte das práticas genéricas da área de processo do CMMI. “Suportado” e “Parcialmente suportado”

estão ligados a uma cobertura parcial. No “Suportado”, pouco esforço é necessário para se obter a cobertura dos objetivos da área de processo, enquanto no “Parcialmente suportado”, um esforço maior deve ser realizado. “Não suportado” indica que não há cobertura. “Conflitante” indica que os objetivos da área de processo não podem ser alcançados com as práticas ágeis e um grande esforço seria necessário para alcançá-lo, possivelmente indo contra os princípios dos métodos ágeis.

A análise é realizada por categorias do CMMI-DEV V1.3 (2010) - Gerenciamento de Processo, Gerenciamento de Projeto, Engenharia e Suporte.

3.2.1. Gerenciamento de Processo

1. Definição dos Processos da Organização (OPD)

No desenvolvimento ágil há regras e diretrizes para a equipe, definição do ciclo de vida e critérios para a forma de trabalho (RESS, 2013). Entretanto, estas regras geralmente têm o foco nos projetos e não na organização como um todo.

Conclusão: Não suportado [0].

2. Foco nos Processos da Organização (OPF)

Os princípios e métodos ágeis têm como foco o produto e não o processo. Além disso, o conhecimento tácito e a troca de informações apenas no projeto corrente, sem a documentação escrita, dificulta a disseminação deste conhecimento para a organização.

Para Fritzsche e Keil (2007), esta área de processo é conflitante com os métodos ágeis, pois, apesar de melhorias serem realizadas, elas são limitadas ao projeto corrente e não são documentadas. O conhecimento sobre estas melhorias está ligado às pessoas.

Conclusão: Conflitante [-].

3. Gestão do Desempenho da Organização (OPM)

Nos métodos ágeis, há um gerenciamento do desempenho dos projetos de maneira proativa para suprir os objetivos do negócio e do cliente. Entretanto, não há diretrizes para um gerenciamento do desempenho da organização.

Conclusão: Não suportado [0].

4. Desempenho dos Processos da Organização (OPP)

Os métodos ágeis não oferecem uma avaliação quantitativa do desempenho dos processos da organização.

A ideia de medir um processo e manter modelos conflita com o Manifesto Ágil (TURNER; JAIN, 2002).

Conclusão: Conflitante [-].

5. Treinamento na Organização (OT)

O plano e treinamento ocorrem nos projetos sem avaliações e métricas para que sejam expandidos para a organização.

Fritzsche e Keil (2007) consideram que técnicas como programação em pares e *coaching* podem ser consideradas como treinamento, melhorando a capacidade de treinamento da organização. Entretanto, eles mesmos reconhecem que há deficiências em relação ao estabelecimento de registros e à avaliação da efetividade do treinamento.

Conclusão: Não suportado [0].

3.2.2. Gerenciamento de Projeto

1. Gestão Integrada de Projeto (IPM)

Na metodologia ágil, o gerenciamento do projeto permite que as preocupações das partes interessadas sejam identificadas e tratadas e que todos tenham uma visão compartilhada do projeto. Assegura também que a equipe execute suas tarefas de

forma coordenada e em tempo hábil, por meio das reuniões semanais, da intensa comunicação e proximidade com o cliente.

Entretanto, não satisfaz a prática de utilizar ativos de processo da organização e contribuir para sua melhoria.

Conclusão: Parcialmente suportado [+].

2. Monitoramento e Controle de Projeto (PMC)

O monitoramento do projeto é realizado diariamente. O cronograma e as estimativas são efetuados por iterações, e por isso, o monitoramento é efetuado por ciclo de desenvolvimento e não para o projeto todo. A intensa comunicação entre o cliente e os membros da equipe ajuda a difundir estas informações (RESS, 2013).

Ocorrências que demandam ações corretivas são coletadas informalmente e analisadas (FRITZSCHE; KEIL, 2007). Novas iterações sempre oferecem boas oportunidades para realizar ajustes.

Marçal (2009) considera a atividade Monitorar o Gerenciamento de Dados não satisfeita, já que no Scrum não há procedimento e planejamento formal para a gestão dos dados do projeto. Mas concorda que no geral, esta área de processo é atendida.

Conclusão: Suportado [++].

3. Planejamento de Projeto (PP)

As estimativas do projeto são efetuadas por iterações e podem ser corrigidas durante o desenvolvimento do projeto. As responsabilidades das pessoas interessadas, incluindo cliente, são também estabelecidas por iterações.

Fritzsche e Keil (2007) salientam que a precisão das estimativas é aumentada por causa das curtas iterações e o comprometimento com as entregas e planos é obtido através do envolvimento de todas as pessoas interessadas.

Marçal (2009), embora concorde que no geral esta área de processo seja atendida pelas práticas ágeis, evidencia alguns pontos em que não são totalmente atendidas. Ela

considera que a estimativa de atributos de produtos de trabalho e tarefas não seja atendida, pois não faz menção a um método explícito para orientar a estimativa tal como WideBand Delphi, Análise de Pontos de Função, Use Case Points ou Story Points. Também considera que não há menção explícita para o estabelecimento de custos do projeto. Também considera que a atividade Planejar o Gerenciamento de Dados não seja satisfeita, pois não existe um plano de dados que formalize a coleta, consolidação e divulgação das informações e dados do projeto.

Conclusão: Suportado [++].

4. Gestão Quantitativa de Projeto (QPM)

Nos métodos ágeis, o foco está nas pessoas e não no processo e cada projeto tende a ser único na forma de condução. Não há foco nas estatísticas do projeto.

Conclusão: Conflitante [-].

5. Gestão de Requisitos (REQM)

Os métodos ágeis fornecem diretrizes para uma efetiva gestão de requisitos através da intensa comunicação entre os integrantes da equipe de desenvolvimento e o cliente.

Entretanto, geralmente não atende a parte da documentação de mudanças de requisitos e de sua motivação a fim de se manter uma rastreabilidade bidirecional entre os requisitos.

Fritzsche e Keil (2007) consideram que apesar da rastreabilidade de requisitos não ser um objetivo explícito dos métodos ágeis, ela é suportada através de histórias, tarefas, testes funcionais e testes unitários, que detectam inconsistências entre o projeto e os requisitos. Entretanto, salientam que a prática XP de jogar fora cartas de história que já foram implementadas pode ser problemática.

Conclusão: Suportado [++].

6. Gestão de Riscos (RSKM)

Não há uma forma ou recomendação explícita de gestão de riscos para abordagem ágil de como identificar, analisar, priorizar ou mesmo mitigar (RESS, 2013). Geralmente esta atividade fica sob os critérios do líder de projeto. Um ponto positivo é que as iterações curtas permitem que a identificação, análise e mitigação sejam aplicadas rapidamente.

Conclusão: Não suportado [0].

7. Gestão de Contrato com Fornecedores (SAM)

A abordagem ágil não trata sobre contrato com fornecedores. A entrega de produtos por uma equipe terceirizada pode ser crítica, pois as iterações são curtas. Há casos em que componentes de fornecedores são necessários para obter o software funcionando ao final de uma iteração (FRITZSCHE; KEIL, 2007).

Conclusão: Não suportado [0].

3.2.3. Engenharia

1. Integração de Produto (PI)

As atividades dos métodos ágeis suportam a integração do produto por meio da montagem progressiva dos seus componentes. A cada iteração, partes do produto são desenvolvidas, integradas ao produto final e avaliadas pelo cliente, assegurando compatibilidade com versões anteriores e componentes já desenvolvidos.

Conclusão: Totalmente suportado [+++].

2. Desenvolvimento de Requisitos (RD)

Os métodos ágeis tratam do levantamento, análise, validação e comunicação das necessidades e expectativas do cliente através de técnicas como proximidade com o

cliente, programação em pares e entregas rápidas. A cada interação define-se um conjunto de requisitos a ser desenvolvido.

Entretanto, a especificação de requisitos muitas vezes permanece vaga (FRITZSCHE; KEIL, 2007). Geralmente é necessário discutir os detalhes diretamente com o cliente durante o desenvolvimento.

Conclusão: Totalmente suportado [+++].

3. Solução Técnica (TS)

Os métodos ágeis apresentam diretrizes para o projeto, desenvolvimento e implementação de soluções para os requisitos. Através de iterações curtas, prioriza e desenvolve um conjunto de requisitos.

Soluções alternativas são exploradas no início do projeto através de protótipos e mais tarde através da refatoração e desenvolvimento iterativo (FRITZSCHE; KEIL, 2007).

Conclusão: Totalmente suportado [+++].

4. Validação (VAL)

A validação de um produto ou componente de produto é realizada pelo cliente que realiza o aceite do componente desenvolvido em cada iteração. A proximidade do cliente é um ponto chave na validação.

Conclusão: Totalmente suportado [+++].

5. Verificação (VER)

As técnicas dos métodos ágeis fornecem suporte para a preparação e execução da verificação, através da programação e revisão em pares e também do *Test-Driven Development* – TDD (primeiro escreve-se o teste e depois o código fonte). As iterações curtas também favorecem uma melhor verificação com componentes desenvolvidos.

Conclusão: Totalmente suportado [+++].

3.2.4. Suporte

1. Análise e Resolução de Causas (CAR)

Os métodos ágeis não oferecem diretrizes para a identificação de causas de defeitos e a implementação de ações para melhorar o desempenho do processo. O foco é o produto e o projeto atual.

Conclusão: Conflitante [-].

2. Gestão de Configuração (CM)

A integridade de produtos de trabalho é alcançada através da intensa comunicação da equipe de desenvolvimento e de práticas como programação em pares e *Test-Driven Development*. O cliente próximo também contribui muito para auxiliar o entendimento e acompanhamento das solicitações de mudanças. As mudanças são bem aceitas nos métodos de desenvolvimento ágil, sendo rapidamente absorvidas e integradas. Em alguns casos, há deficiência na documentação dessas mudanças, tendendo a ficar na informalidade.

Conclusão: Suportado [++].

3. Análise e Tomada de Decisões (DAR)

O objetivo desta área de processo é fornecer subsídios para tomar decisões com base em um processo formal para avaliação de alternativas (CMMI-DEV V1.3, 2010). A metodologia ágil não oferece diretrizes para análise e tomada de decisões, principalmente de maneira formal. Os ciclos curtos dificultam a coleta e análise deste tipo de informação.

Turner (2002) salienta que os métodos ágeis preferem a habilidade de se adaptar rapidamente a novas situações a um processo formal de avaliação.

Conclusão: Conflitante [-].

4. Medição e Análise (MA)

Nos métodos ágeis, há a foco no produto e entrega e, por isso, as coletas e análise de dados geralmente ficam em segundo plano. Além disso, as medições de progresso são por ciclo e não para o projeto todo, o que nem sempre é suficiente para fornecer dados objetivos que possam ser utilizados na tomada de decisões e na implementação de ações corretivas apropriadas. Fritzsche e Keil (2007) complementam afirmando que embora existam dados de medida frutos da intensa comunicação entre as equipes, estes dados geralmente não são guardados.

Conclusão: Conflitante [-].

5. Garantia da Qualidade de Processo e Produto (PPQA)

Embora haja técnicas para avaliação objetiva dos produtos de trabalho, os métodos ágeis não focam o processo. Por isso, nem sempre é possível fornecer um *feedback* completo à equipe do projeto e aos gerentes sobre os resultados das atividades de garantia da qualidade.

Conclusão: Parcialmente suportado [+].

3.2.5. Conclusões

A

Tabela 6 mostra um resumo da análise realizada neste item. A coluna “Suporte” indica se as práticas ágeis oferecem suporte à área de processo, seguindo a simbologia da Tabela 5. Na coluna “Detalhes”, apresentam-se as justificativas para esta classificação.

Na categoria Gerenciamento de Processo observa-se que a maioria das áreas de processo não são suportadas e duas são conflitantes. Na verdade, há diretrizes e práticas relacionadas a estas áreas de processo, mas elas são voltadas ao produto e ao projeto e não para a organização.

Tabela 6 - Aderência das práticas ágeis às áreas de processo do CMMI

Categoria	Área de Processo	Suporte	Detalhes
Gerenciamento de Processo	Definição dos Processos da Organização (OPD)	[0]	Há definição dos processos, mas com foco nos projetos e não na organização.
	Foco nos Processos da Organização (OPF)	[-]	O foco é no produto.
	Gestão do Desempenho da Organização (OPM)	[0]	Há apenas gerenciamento do desempenho dos projetos.
	Desempenho dos Processos da Organização (OPP)	[-]	Não há uma avaliação quantitativa do desempenho dos processos da organização.
	Treinamento na Organização (OT)	[0]	O treinamento ocorre apenas nos projetos.
Gerenciamento de Projeto	Gestão Integrada de Projeto (IPM)	[+]	A gestão é realizada por meio das reuniões semanais, da intensa comunicação e proximidade com o cliente. Mas, não utiliza ativos para melhorar a organização.
	Monitoramento e Controle de Projeto (PMC)	[++]	O monitoramento é realizado diariamente e é efetuado por ciclos de desenvolvimento. Mas não há procedimento e planejamento formal para a gestão dos dados do projeto.
	Planejamento de Projeto (PP)	[++]	O planejamento do projeto é efetuado por iterações. Falta estimativa formal de tarefas e custos e não existe um plano formal de dados de informação do projeto.
	Gestão Quantitativa de Projeto (QPM)	[-]	Não há foco nas estatísticas do projeto.
	Gestão de Requisitos (REQM)	[++]	Realizada através de intensa comunicação entre os integrantes da equipe e o cliente. Entretanto, não oferece práticas para rastreabilidade bidirecional dos requisitos.
	Gestão de Riscos (RSKM)	[0]	Não há recomendação explícita de como realizar a gestão de riscos.
	Gestão de Contrato com Fornecedores (SAM)	[0]	Não aborda o contrato com fornecedores.

Continua

Continuação

Categoria	Área de Processo	Suporte	Detalhes
Engenharia	Integração de Produto (PI)	[+++]	Integração através da montagem progressiva dos seus componentes.
	Desenvolvimento de Requisitos (RD)	[+++]	Realizado através de técnicas como proximidade com o cliente, programação em pares e entregas rápidas.
	Solução Técnica (TS)	[+++]	Há diretrizes para o projeto, desenvolvimento e implementação de soluções para os requisitos.
	Validação (VAL)	[+++]	Realizada pelo cliente a cada iteração.
	Verificação (VER)	[+++]	Programação e revisão em pares e <i>Test-Driven Development</i> em iterações curtas.
Suporte	Análise e Resolução de Causas (CAR)	[-]	Foco é no produto e projeto atual.
	Gestão de Configuração (CM)	[++]	Intensa comunicação da equipe de desenvolvimento e proximidade com o cliente. Falta de documentação das mudanças.
	Análise e Tomada de Decisões (DAR)	[-]	Não há diretrizes para a análise e tomada de decisões de maneira formal.
	Medição e Análise (MA)	[-]	O foco é no produto e na entrega.
	Garantia da Qualidade de Processo e Produto (PPQA)	[+]	Há técnicas para avaliação dos produtos de trabalho, mas sem foco nos processos.

Fonte: Autora

As áreas de processo da categoria Gerenciamento de Projeto são quase todas atendidas, através de práticas como reuniões semanais, proximidade do cliente e projeto e desenvolvimento por iterações curtas. As lacunas aparecem por conta da falta de foco nas estatísticas do projeto, recomendações formais de gestão de riscos e contrato com fornecedores.

Na categoria Engenharia, todas as áreas de processo são atendidas. O desenvolvimento, validação, verificação e integração do produto são realizados em iterações curtas, com

participação ativa do cliente e utilizando técnicas como programação em pares, entregas rápidas e *Test-Driven Development*.

Finalmente, na categoria Suporte, encontra-se o maior número de áreas conflitantes. As áreas de processo Análise e Resolução de Causas, Análise e Tomada de Decisões e Medição e Análise são conflitantes, novamente pelo foco no produto e projeto atuais e entregas rápidas. A área Garantia da Qualidade é parcialmente suportada, pois apesar de haver técnicas para avaliação dos produtos de trabalho, não há foco nos processos. A área Gestão de Configuração é atendida pela intensa comunicação da equipe de desenvolvimento e o cliente, mas muitas vezes carece de uma documentação das mudanças.

A análise das áreas de processo do CMMI não atendidas pelos métodos ágeis mostra as lacunas entre eles. É possível verificar quais as áreas demandariam maior esforço para atenderem as diretrizes do CMM. A categoria Suporte apresenta o maior número de áreas de processo conflitantes. As maiores lacunas aparecem quando o CMMI enfatiza a noção de organização. É exatamente nestas lacunas existentes nos métodos ágeis que este pesquisa tem o foco.

3.3. Entrevista

A fim de se investigar sobre a utilização de métodos ágeis no desenvolvimento de software, foi realizada uma entrevista com profissionais da área. Estas entrevistas foram realizadas com pessoas ligadas ao desenvolvimento de software que utilizam métodos ágeis. Os principais objetivos destas entrevistas foram:

- 1- Identificar as metodologias ágeis de desenvolvimento de software utilizadas nas empresas;
- 2- Verificar qual a importância de certificações de qualidade de software para estas empresas;
- 3- Identificar que tipo de documentação é feita nas empresas;
- 4- Verificar se há mecanismos de aprendizado entre um projeto e outro.

Esta visão do desenvolvimento de software nas empresas contribui para o entendimento da prática da engenharia de software e constitui uma parte importante da análise realizada neste trabalho.

3.3.1. Metodologia

As entrevistas foram realizadas por telefone seguindo um roteiro semiestruturado em forma de questionário. Uma entrevista pode ser entendida como uma conversa guiada (CHARMAZ, 2006). As perguntas foram elaboradas de acordo com o referencial teórico e a revisão da literatura e os entrevistados puderam discursar livremente, com a menor interferência possível do entrevistador. As conversas foram gravadas e posteriormente transcritas sem a identificação dos participantes. As entrevistas foram realizadas entre outubro de 2016 e abril de 2017.

O roteiro em forma de questionário foi composto por questões abertas e dirigidas a pessoas ligadas ao ciclo de desenvolvimento de software das empresas. Inicialmente foi feito um contato com as empresas por e-mail e telefone e após a sinalização positiva do aceite de participação no projeto de pesquisa, marcou-se a reunião para a realização da entrevista. O roteiro da entrevista encontra-se no APÊNDICE A – Questionário utilizado na entrevista.

Foram entrevistadas sete pessoas ligadas ao ciclo de desenvolvimento de software no Brasil, nomeadas E1, E2, E3, E4, E5, E6 e E7. Dois entrevistados atuam na área de testes e garantia da qualidade (E2 e E6). Um entrevistado é desenvolvedor de software (E7) e os demais atuam na coordenação, gerência ou diretoria de desenvolvimento de software (E1, E3, E4 e E5). A Figura 13 mostra a distribuição da área de atuação dos entrevistados.

Figura 13 – Área de atuação dos entrevistados

Fonte: Autora

3.3.2. Informações coletadas

Todas as pessoas entrevistadas estão diretamente envolvidas no ciclo de desenvolvimento de software e, por isso, o entendimento sobre as perguntas foi muito fácil e rápido.

Metodologia de desenvolvimento

Duas empresas seguem a metodologia ágil Scrum com poucas alterações. As demais seguem uma metodologia ágil própria baseado no Scrum, com algumas adaptações. Desses, alguns são baseados em metodologia enxuta. No geral, todas as equipes da empresa seguem a metodologia ágil, mas cada uma a sua maneira.

Embora o entrevistado E3 considere que o modelo de desenvolvimento de software do departamento seja ágil, ele apresenta características de um modelo em cascata, com definição do projeto e fechamento de escopo no início do desenvolvimento. Isso provavelmente seja consequência do modelo comercial utilizado na empresa. Contudo, por utilizar algumas práticas ágeis e pela longa experiência com certificações de qualidade, decidiu-se manter as informações desta entrevista neste trabalho.

O entrevistado E1 comentou que antes utilizavam metodologia bem próxima do RUP (*Rational Unified Process*), com vários artefatos, documento de visão, casos de uso,

diagramas de classe, etc. Depois da mudança para métodos ágeis, houve uma diminuição drástica das entregas que eram desfeitas e também uma diminuição considerável dos erros em produção.

Outro entrevistado, E4, também utilizava RUP antes de mudarem para metodologia ágil. *“Funcionava bem, principalmente para segurança de que aquilo que se definiu foi entregue, mas não dava segurança de que aquilo que estava entregando ao cliente realmente agregava valor”*, afirmou o entrevistado. E concluiu dizendo: *“E esses modelos anteriores eram contra as mudanças. E percebemos que era necessário saber mudar.”*. Está bem satisfeito com a metodologia ágil.

Certificação de qualidade

Das empresas entrevistadas, apenas duas possuem certificação de qualidade de software voltada ao desenvolvimento de software. A empresa do entrevistado E6 possui certificação CMMI-DEV 1.3 e do entrevistado E3 tem MPS-BR. O entrevistado E6 acredita que a certificação CMMI-DEV não seja tão valorizada no Brasil, mas no exterior sim. Para os projetos internacionais, ter a certificação é um diferencial.

O entrevistado E3 informou que a empresa teve certificação CMMI por vários anos e as razões para a mudança para MPS-BR foram custo e simplicidade da certificação. Entrevistado E1 afirmou que os clientes não cobram uma certificação, mas acredita que possa ser importante em outros tipos de sistemas, como os críticos, e também em licitações (que a empresa não participa). Finalizou dizendo que não sabe afirmar se com uma certificação o processo de desenvolvimento de software seria melhor. As empresas grandes que eles se espelharam não tinham a certificação. O entrevistado E2 disse que a empresa não tem certificação de qualidade. Acredita que as empresas que solicitam certificações de seus fornecedores e parceiros sejam apenas as bem grandes. É uma maneira de fazer uma primeira filtragem.

A empresa do entrevistado E4 tinha certificação CMMI, mas acabaram abandonando a certificação, pois entenderam que os problemas que a certificação endereçava não eram os problemas que realmente a indústria de software enfrentava. Consequentemente, acabava atrapalhando a entrega de valor efetivo ao cliente. Afirmou que os clientes em geral não

cobram nenhum certificado de qualidade de desenvolvimento de software. A empresa não participa de licitações e projetos críticos. Entretanto, reconhecem que no passado, o processo de certificação auxiliou na melhoria do processo de desenvolvimento de software e até hoje alguns princípios do CMMI são utilizados. Na época, a busca por uma certificação de qualidade teve como objetivo melhorar os processos de desenvolvimento de software e aumentar a capacidade de atuação no mercado americano.

O entrevistado E2 é a favor da certificação de qualidade, pois acredita que o desenvolvimento tradicional só ficou mais maduro e consistente por causa da certificação de software, que exigiu processos e documentações. Disse não conhecer diretrizes do CMMI para metodologias ágeis.

Dificuldades no desenvolvimento de software

Indagados sobre as dificuldades encontradas durante o desenvolvimento de software, em especial relacionadas à metodologia ágil, a resposta que mais apareceu foi a dificuldade de elicitar requisitos e definir escopo (E3, E4, E6). Além disso, outras dificuldades são:

- Imaturidade de algumas pessoas, achando que na metodologia ágil pode fazer o que quiser; outras acham que não precisa documentar nada. (E1 e E2);
- Projetos que, por suas características, dificultam o trabalho com a metodologia ágil, como por exemplo, desenvolvimento de plataforma e aplicativos de baixo nível (E1);
- Projetos com participação de várias equipes em uma entrega (E2);
- Falta de documentação em equipes com alta rotatividade (E2);
- Empresas com várias equipes, com gerências distintas, acabam trabalhando de maneira muito diferente uma das outras (E2);
- Falta de planejamento, projeto e pesquisa, logo entrando na fase de desenvolvimento (E5);
- Dificuldade de realizar treinamentos; prazos muito apertados (E5);
- Resistência a mudanças em equipes mais velhas ou antigas (E5);
- Conhecer e trabalhar com ferramentas e negócio específicos da empresa (E6).

Alguns entrevistados citaram dificuldades encontradas durante a transição para modelos ágeis:

- Alteração do espaço físico, incluindo o mobiliário (bairas para mesas grandes, por exemplo) (E1);
- Dificuldade para áreas que não são de tecnologia (como Jurídico, Financeiro e RH) entender a mudança (E1);
- Adaptação de algumas pessoas da equipe à mudança (E1, E4);
- Encaixar este modo de trabalho nos modelos comerciais (E4).

O entrevistado E1, apesar de mencionar algumas dificuldades durante a transição para métodos ágeis, não acredita que a metodologia seja responsável pelos problemas encontrados. Na sua visão, a metodologia ágil na verdade revela mais fácil essas fraquezas e mostra os gargalos.

O entrevistado E2 acredita que o grande problema dos métodos ágeis é deixar muito “solto” e consequentemente, cada equipe trabalha de uma maneira. Acredita que deve haver um padrão para que todos os participantes trabalhem de maneira semelhante. *“É possível verificar esta fragilidade em empresas com muitas equipes, com lideranças diferentes”*, complementa.

O entrevistado E4, apesar de citar a definição de escopo como um ponto de atenção, explica que o problema é minimizado pelo modelo comercial de seus produtos. A empresa não vende projeto fechado, pois não acredita que seja possível definir todos os requisitos antes de ver o trabalho acontecendo. É fornecida uma ordem de grandeza, mas sabe-se que os requisitos irão mudar a cada iteração. A cada produto entregue, é esperado que o cliente interaja com ele e proponha alterações e novas funcionalidades. Eles acreditam que ninguém consegue dizer exatamente o que quer só pensando. Se o cliente for obrigado a definir todos os requisitos no início, ele vai pedir uma quantidade grande de funcionalidades, mesmo sem saber se precisa, porque só tem aquele momento para pedir. O modelo de gestão da empresa se preocupa com o valor que gera para o cliente.

O entrevistado E4 finaliza dizendo que a grande maioria dos analistas apoiou a mudança para o modelo ágil, pois eles passaram a ser mais integrados ao processo de concepção do software, participando das decisões sobre o mesmo, não apenas recebendo uma especificação pronta do que fazer.

Documentação

No geral, as empresas entrevistadas documentam as informações mais relevantes, ou que geram impacto no projeto. A maioria utiliza ferramentas *web* nas quais as equipes podem livremente documentar o que achar importante. Alguns exemplos de documentação realizada:

- Requisitos e testes (E1, E2, E3, E7);
- Atividades que estão sendo realizadas por cada integrante da equipe (E1)
- Regras de negócio, maneira de testar o sistema, como o sistema foi construído (E1);
- Proposta comercial (escopo) (E3);
- Especificação funcional (detalhamento do escopo) (E2, E3, E5, E6, E7);
- Documentos de monitoramento e controle do status do projeto; atas de reunião; matriz de riscos (E3);
- Guias de boas práticas (E4);
- Documentação no código (E5, E6);

O entrevistado E2 complementou dizendo que não há uma regra para a documentação. Algumas equipes não documentam quase nada. Em outras, a documentação faz parte da meta da equipe. Considera esta falta de padronização um ponto de atenção.

O entrevistado E4 afirma que a documentação é a mínima possível. Aboliram-se documentações que eram criadas apenas para comunicar algo a uma pessoa.

Para o entrevistado E5, a documentação é bem deficiente, faltando uma visão macro do projeto e suas funcionalidades. Há documentação focada no usuário final, mas não pensando no entendimento de uma nova pessoa na equipe ou uma documentação suficientemente detalhada para se fazer uma análise de segurança, por exemplo. Internamente, a documentação que é feita é sobre novas funcionalidades e nos testes unitários. Alteração em código gera pelo menos nota de *release*.

Na empresa do entrevistado E6, a documentação é relativamente extensa, com descrição funcional e especificação técnica. Acredita que isso seja necessária porque o desenvolvimento é distribuído, com integrantes de diversos países.

Design Rationale

Para a questão sobre o DR, o entrevistador explicou de forma sucinta do que se trata antes de realizar o questionamento. As informações sobre *DR* estão no item 2.4.

Dos sete entrevistados, apenas um afirmou que este tipo de informação é quase toda documentada, incluindo questionamentos e havendo a possibilidade de rastrear as alterações (E6). Três entrevistados disseram que este tipo de informação não é documentado (E2, E5 e E7). O entrevistado E2 comentou que este tipo de informação fica apenas nas reuniões do grupo. E até citou um exemplo de decisão que causou problemas, por não ter sido compartilhada com as demais equipes. O entrevistado E5 reconhece que é um risco não ter este tipo de informação documentada. Fica apenas na cabeça dos integrantes das equipes. Se um dia ele sair, nunca recuperará as razões das decisões realizadas. E corre-se o risco de voltar um erro já corrigido ou tentar uma solução que já foi descartada. A inclusão de novo integrante na equipe também é dificultada por esta falta de documentação.

Entrevistados E1, E3 e E4 relataram que parte do DR é capturado. E1 afirma que em geral só se documenta decisões que gerarão regras que todos precisam ter conhecimento para testar. Não se recorda de haver registro de alternativas consideradas ou soluções descartadas. O entrevistado E3 relatou que apenas são registradas as decisões que geram impacto em custo, escopo, prazo ou recurso. E, portanto, alternativas descartadas não são registradas.

Gestão do conhecimento

Em geral, não há um processo formal de aprendizado entre um projeto e outro (E2, E4, E5 e E6). O entrevistado E4 relata que as informações são trocadas informalmente dentro do projeto e compartilhadas entre os líderes, mas não há um processo formal para registro de informações importantes. Não há registro de informações que poderiam ser úteis para outras equipes ou outros projetos. Entrevistado E2 diz ter reuniões de retrospectiva por ciclos e acha importante a reunião de retrospectiva sobre o projeto todo, mas não é uma prática realizada atualmente.

Na empresa do entrevistado E1, há encontros mensais ou bimestrais para falar sobre conhecimentos técnicos mais específicos. Todos os técnicos recebem *email* com a convocação

e os tópicos que serão abordados e então podem se inscrever. A reunião de retrospecto é só por *Sprint*. Na empresa do entrevistado E3, há reuniões de lições aprendidas no encerramento do projeto, da qual participam coordenadores de projetos. Na empresa do E7, o aprendizado se dá através de cursos e palestras com membros de outros projetos, sobre ferramentas e funcionalidades em comum. Não há reunião *post-mortem*.

3.3.3. Análise dos resultados

As empresas estão muito satisfeitas com a metodologia ágil e verificaram na prática que ela é a mais adequada para suprir as necessidades do mercado, entregando mais rapidamente um produto ao cliente, com menos defeitos e maior valor agregado. Acreditam que seguir uma metodologia que dê suporte a mudanças de requisitos seja algo essencial.

A metodologia Scrum se sobressaiu entre as empresas entrevistadas. Corbucci et al. (2011) realizaram uma pesquisa em empresas que utilizam métodos ágeis no Brasil e verificaram que mais de 50% seguem a metodologia Scrum no país.

Embora a maioria das empresas não possua certificação de qualidade e acredite não ser vantajoso para o nicho de atuação da empresa, quase todos acreditam que ela possa ser útil em determinados casos, como em sistemas críticos, licitações e grandes projetos. A grande maioria das empresas entrevistadas tem o foco quase todo no mercado nacional. Isso também parece ser uma das razões de não se preocupar em ter uma certificação de qualidade.

A maioria das empresas, apesar de não ter a certificação de qualidade atualmente, já a tiveram no passado. Ou seja, já tiveram uma estruturação dos processos e uma melhoria no ciclo de desenvolvimento de software.

Sobre as dificuldades encontradas no desenvolvimento de software com metodologias ágeis, a mais lembrada foi a elicitação de requisitos e definição de escopo. Relataram também faltar maturidade para entender e aplicar a metodologia. Há problema de não padronização do modo de trabalho em equipes com gerências diferentes.

Na parte de documentação, capturam-se apenas as informações mais importantes ou que geram impacto no projeto. A parte de DR é semelhante, somente documenta-se se houve um impacto direto no projeto.

O aprendizado entre projetos se dá de forma informal ou através de reuniões ao final do projeto. Mas pouca informação é documentada.

Por se tratar de uma investigação qualitativa, os resultados inferidos nesta entrevista não podem ser generalizados para todas as equipes de desenvolvimento ágil de software. Entretanto, foi possível obter uma boa visão sobre alguns pontos do desenvolvimento ágil nas empresas.

3.4. Considerações finais

Shutherland et al. (2007) referem-se à combinação de Scrum com CMMI como uma “poção mágica”. Scrum assegura que os processos são implementados de maneira eficiente ao mesmo tempo em que aceita mudanças e o CMMI assegura que todos os processos relevantes são considerados.

Observa-se que há um grande número de trabalhos científicos sobre métodos ágeis e CMMI, embora nenhum desses trabalhos se sobressaia ou tenha se tornado um padrão. A maioria foca em assuntos pontuais sobre a utilização desses dois temas. Quase todos eles realizam a aderência na categoria Gerenciamento de Projeto do CMMI-DEV.

A utilização desta abordagem híbrida é uma realidade em várias empresas brasileiras. Isso foi constatado tanto através da revisão bibliográfica como pelas entrevistas.

As entrevistas realizadas em empresas de desenvolvimento de software revelaram como são realizadas as atividades ágeis na prática. Realmente a maioria não realiza a captura de DR.

A análise de aderência das práticas ágeis com as diretrizes do CMMI-DEV mostrou as áreas de processo do CMMI que são atendidas pelos métodos ágeis. Foi possível verificar as lacunas entre eles e quais áreas demandariam maior esforço para atenderem as diretrizes do CMMI. Em geral, estas áreas são as que enfatizam a noção de organização.

A análise realizada neste capítulo reforça a importância do tema de estudo e verifica-se que ainda há lacunas a ser pesquisadas.

4. SCRUM-DR: UMA EXTENSÃO DO FRAMEWORK SCRUM ADERENTE AO CMMI-DEV UTILIZANDO TÉCNICAS DE DESIGN RATIONALE

Este capítulo apresenta a contribuição principal deste trabalho. A partir da análise realizada no Capítulo 3, construiu-se um *framework* de desenvolvimento de software ágil aderente às diretrizes do modelo de qualidade CMMI-DEV versão 1.3 na categoria Suporte, contendo práticas, modelo de documentos e representações.

Um *framework* geralmente proporciona uma abstração da solução para um número de problemas que têm semelhanças. Além disso, usualmente esboça os passos ou fases que devem ser seguidos para implementar uma solução sem entrar em detalhes de quais atividades serão feitas em cada fase (MNKANDLA, 2009).

4.1. Framework Scrum-DR

O *framework* proposto Scrum-DR foi construído tendo como base a satisfação das diretrizes da categoria Suporte do modelo de qualidade CMMI-DEV 1.3. Consequentemente, levaram-se em consideração as práticas e diretrizes das áreas de processo Análise e Resolução de Causas (CAR), Gestão de Configuração (CM), Análise e Tomada de Decisão (DAR), Medição e Análise (MA) e Garantia da Qualidade de Processo e Produto (PPQA). É claro que, pela intensa ligação com as demais áreas de processo do modelo de qualidade, a proposta contribui para diversas outras áreas de processo.

Além disso, Scrum-DR é baseado nos valores, nas práticas e nos princípios do desenvolvimento ágil de software. Suas diretrizes seguem as características dos modelos ágeis descrito por Abrahamsson et al. (2003):

- Cooperativo: próximo ao cliente; forte interação entre os desenvolvedores;
- Direto: fácil de aprender e modificar e suficientemente documentado;
- Incremental: feito em pequenas versões com ciclos rápidos;
- Flexível: habilidade de acomodar mudanças.

Essas características são atendidas pelo conjunto de elementos básicos da proposta:

- Simples, direto e fácil de aprender;
- Aceita mudanças;
- Modelo de documentação único para captura do racional;
- Incentivo à melhoria contínua de produto e processo;
- Intensa comunicação entre os envolvidos;
- Desenvolvimento incremental em todas as fases.

O Scrum-DR é formado pela definição de suas fases, papéis, artefatos e eventos. Especifica também modelos de documentação para facilitar a sua aplicação.

O *framework* foi construído tendo o objetivo de ser executado com o mínimo de alocação de recursos e sem perder a agilidade das atividades diárias do desenvolvimento de software. Sempre que possível, procurou-se manter a nomenclatura dos artefatos e papéis para facilitar o entendimento do *framework* proposto e facilitar uma migração do Scrum para Scrum-DR.

4.1.1. Fases

O Scrum-DR possui 3 fases distintas: Iniciação, Desenvolvimento e Finalização. A Figura 14 mostra uma visão geral das fases do *framework* Scrum-DR.

Na fase Iniciação, primeiramente estabelece-se uma visão geral do projeto e principais funcionalidades. Em seguida, através de iterações sucessivas, refinam-se os requisitos até se obter uma lista com todos os requisitos desejados (Backlog do Produto). Nesta fase, definem-se também riscos ao projeto, estimativas de esforço, padrões, tecnologia, arquitetura e recursos. Os papéis são definidos e documentados, deixando claro quem é o Scrum Master, Product Owner, Time de desenvolvimento e Cliente (Vide item 4.1.2).

Ao final desta fase, obtém-se também o Backlog do Produto, que é uma lista ordenada com todas as funcionalidades desejadas no produto. Esta é uma versão inicial do documento e ela poderá sofrer atualizações ao longo do desenvolvimento.

Figura 14 – Fases do Scrum-DR

Fonte: Autora

Na fase Desenvolvimento, realizam-se várias iterações chamadas Sprints, com o objetivo de definir uma lista de funcionalidades a ser implementada nas próximas semanas, realizando a análise, projeto, implementação e testes.

As principais decisões de projeto (*rationales*) são capturadas pelo Product Owner, gerando o documento DOC-DR, conforme modelo proposto no item 4.1.3.4.

Ao final desta fase, obtém-se um incremento de produto, que é a soma de todos os itens do Backlog do Produto completados durante a Sprint.

Na última fase, chamada de Finalização, é realizada a reunião Retrospectiva do Projeto, com o objetivo de sugerir melhorias para os processos da empresa visando à melhoria da qualidade de produto e processos da organização.

4.1.2. Papéis

No Scrum, o Time Scrum é composto pelo Product Owner, o Time de Desenvolvimento e o Scrum Master (SCHWABER; SUTHERLAND, 2016). Os Times Scrum são auto-organizáveis e multifuncionais. O modelo de equipe no Scrum é projetado para aperfeiçoar a flexibilidade, criatividade e produtividade.

Nesta proposta, os papéis se mantêm, com algumas atribuições extras. Além disso, foi acrescentado o papel do Cliente, devido à sua importância e constante participação durante o desenvolvimento do produto. A Figura 15 mostra os papéis do Scrum-DR e um resumo de suas atribuições.

Figura 15 – Papéis no Scrum-DR

Fonte: Autora

O Product Owner é o responsável por maximizar o valor do produto e do trabalho do Time de Desenvolvimento. É responsável por gerenciar o Backlog do Produto, no seu conteúdo e

priorização. Na fase Iniciação, é o responsável pela captura e recuperação das decisões do projeto durante a criação do Backlog do Produto.

O Time de Desenvolvimento é formado por profissionais que realizam o trabalho de entregar uma versão utilizável que potencialmente incrementa o produto ao final de cada Sprint (SCHWABER; SUTHERLAND, 2016). São estruturados e autorizados pela organização para gerenciar seu próprio trabalho.

No Scrum-DR são também estimulados a informar e consultar sobre decisões importantes do desenvolvimento do projeto e sugerir melhorias ao processo de desenvolvimento. Além disso, realizam a captura de medidas sobre o projeto, como o número de ocorrências de erros. Em casos em que o número de erros de uma Sprint ultrapasse um valor limite, a equipe deve diminuir a quantidade de funcionalidades a ser entregue e verificar como pode melhorar.

O Scrum Master é o responsável por garantir que o *framework* seja entendido e aplicado. No Scrum-DR, recebe uma atribuição muito importante e que é o ponto chave para o sucesso da utilização do DR: é o responsável por garantir que todos do Time de Desenvolvimento e também o Product Owner entendam as características e importância do DR. Além disso, é o responsável por estimular a sua captura e recuperação, questionando o Time de Desenvolvimento durante a reunião de Revisão da Sprint e Reunião de Retrospectiva.

O Cliente participa ativamente do desenvolvimento do software, tanto na definição dos requisitos e priorização como na validação e testes. Tem papel fundamental para que o modelo funcione, colaborando significativamente para que o produto entregue realmente agregue valor à empresa. Preferencialmente fica fisicamente próximo à equipe de desenvolvimento, facilitando a comunicação e o entendimento do projeto.

4.1.3. Artefatos

Os artefatos definidos para o Scrum-DR são projetados para maximizar a transparência das informações entre todos os envolvimento e também fornecer informações importantes para a tomada de decisão e melhoria da qualidade de produtos e processos.

Os principais artefatos são o Plano de Projeto, Backlog do Produto, Backlog da Sprint, DOC-DR, Incremento, Plano de Melhoria da Sprint e Plano de Melhoria.

4.1.3.1. Plano do Projeto

O Plano do Projeto contém as informações gerais sobre o projeto, indicando seus objetivos e características principais. Inclui informações sobre os papéis, previsão de recursos e definições das reuniões (local, horário).

A Tabela 7 mostra o modelo com as informações do Plano do Projeto.

Tabela 7 - Plano do Projeto

Plano do Projeto	
Campos	Descrição
Nome	Descrição resumida
Descrição	Descrição mais detalhada do projeto
Data Inicial	Data de início do projeto
Data de Término	Previsão de finalização do projeto
Objetivos	Objetivos e justificativas do projeto
Restrições	Restrições de escopo, prazo ou custo, se houver
Riscos	Riscos previamente identificados
Reuniões	Local, data e horário das próximas reuniões
Medições	Definição das medições do projeto
Product Owner	Nome da pessoa com papel de Product Owner
Scrum Master	Nome da pessoa com papel de Scrum Master
Time de Desenvolvimento	Definição das pessoas que farão parte do Time de Desenvolvimento
Cliente	Nome da(s) pessoa(s) com papel de Cliente
Stakeholders	Demais pessoas, não listadas acima, mas que tem interesse no projeto

Fonte: Autora

A “Data de Término” é apenas uma previsão da data de finalização do projeto. A data real de término será melhor estimada quando os requisitos forem melhor detalhados.

As reuniões são preferencialmente no mesmo local. As reuniões diárias devem ser sempre no mesmo horário, a fim de que sejam incorporadas ao cotidiano dos envolvidos.

No campo “Medições”, descrevem-se as medições de desempenho que serão realizadas no projeto. Podem incluir ações que serão tomadas caso o desempenho não seja satisfatório. Por exemplo, enviar um relatório ao Scrum Master se o número de bugs da Sprint ultrapassar um valor.

É importante ter a definição clara das pessoas envolvidas e de seus papéis.

4.1.3.2. Backlog do Produto

O Backlog do Produto é uma lista ordenada de tudo que deve ser necessário no produto. Mudanças nos requisitos de negócio, condições de mercado ou tecnologia podem causar mudanças no Backlog do Produto. Contém também informações sobre mudanças de requisitos.

A Tabela 8 apresenta o modelo do Backlog do Produto. São informações diretas sobre as funcionalidades que se deseja no produto. O campo “Sprint” armazena o número da Sprint que a funcionalidade foi implementada. Inicialmente contém valor 0. O campo “Prioridade” indica a ordem que as funcionalidades devem ser implementadas. O valor 1 indica prioridade alta e o requisito deve ser implementado nas próximas Sprints. Elementos com prioridade mais alta devem ter um detalhamento maior, pois serão os primeiros a serem implementados.

O campo “Agrupamento” reúne funcionalidades semelhantes. Exemplos de agrupamento podem ser: cadastro, venda, processamento off-line, relatório, etc. Tem o objetivo de facilitar uma busca.

Quando um requisito é alterado, o campo “Alteração” é preenchido, incluindo data da alteração, solicitante e razão da mudança.

Tabela 8 - Backlog do Produto

Backlog do Produto	
Campos	Descrição
ID	Identificador
Sprint	Número da Sprint que o requisito foi implementado
Nome	Descrição resumida
Descrição	Descrição Detalhada
Tipo	Tipo do requisito: funcional (F) ou não funcional (NF)
Prioridade	Valor de 1 a 10 indicando a urgência de implementação do requisito. 1 indica prioridade mais alta.
Grupo	Agrupamento
Alteração	Data e razão de mudança, se houver
Obs	Informações adicionais sobre o requisito

Fonte: Autora

A Tabela 9 ilustra um exemplo de Backlog do Produto de um sistema que realiza o aluguel de carros.

Tabela 9 - Exemplo de um Backlog do Produto no Scrum-DR

Sprint	1	1	2
Nome	Cadastro Cliente	Cadastro de Carro	Tempo para buscar carros disponíveis
Descrição	Cadastro de Cliente – informações Vide CAD004	Cadastro de carros disponíveis para locação – informações CAD010	O tempo para realizar a busca por carros disponíveis não deve ultrapassar 5 segundos
Tipo	F	F	NF
Prioridade	1	1	8
Grupo	Cadastro	Cadastro	Desempenho
Alteração			
Obs			

Fonte: Autora

Com a finalidade de se ter a informação sobre dependência de requisitos, adiciona-se ao Backlog do Produto uma matriz de dependências. Através dela, é possível verificar quais requisitos são afetados se houver alteração em determinado requisito.

O Backlog do Produto é dinâmico. Ocorrem alterações tanto para refinar uma funcionalidade como para excluir ou incluir novos itens. Elementos que estão mais próximos de serem implementados devem estar mais detalhados.

4.1.3.3. Backlog da Sprint

O Backlog da Sprint contém informações necessárias para a implementação da próxima Sprint. É formada pelo conjunto de itens do Backlog do Produto selecionados para a Sprint mais o plano de entrega. No plano de entrega o Time de Desenvolvimento define resumidamente como irá construir as funcionalidades selecionadas.

4.1.3.4. DOC-DR

O DOC-DR é uma documentação em formato gráfico que contém informações sobre decisões de projeto e também alternativas consideradas em uma solução. O modelo foi definido incluindo elementos dos modelos gIBIS (CONKLIN; BEGEMAN, 1988) e NFR-Framework (CHUNG et al., 1996). A escolha por utilizar elementos destes dois modelos foi porque eles contêm uma representação gráfica muito intuitiva e também pelo grande número de pesquisas realizadas utilizando-se estes modelos.

O objetivo foi definir um modelo gráfico simples e intuitivo, que pudesse ser utilizado tanto durante o raciocínio para se chegar à resposta a um problema como também como documentação de uma decisão tomada.

Os principais elementos deste documento são (Veja Figura 16):

- Questão: problema em discussão;
- Alternativa: possível solução para os problemas;
- Argumento: requisitos e restrições que as soluções podem satisfazer.

Ao elemento Argumento foi acrescentado um valor numérico de 1 a 5 chamado Criticidade, que indica a importância desta característica, sendo valor 5 pouco importante e 1 crítico. Neste último caso, o Argumento fica marcado com um sinal de exclamação (!), alertando sobre a sua criticidade.

Figura 16 – Elementos do DOC-DR

Fonte: Autora

Finalmente, temos um retângulo que é colocado em volta da alternativa que foi escolhida como solução do problema.

A Figura 17 mostra uma visão geral destes elementos e como podem se relacionar. Entre Alternativa e Argumento, utilizou-se um conector semelhante ao existente no modelo NFR. Os valores das setas indicam que a Alternativa satisfaz o Argumento de maneira:

- + Parcialmente positiva
- ++ Totalmente positiva
- - Parcialmente negativa
- -- Totalmente negativa

Viana, Conte e Marczak (2015) procuram entender como o conhecimento é formado e perdido nas organizações. Eles entendem que as empresas de software praticam um conjunto de atividades que apoiam essa criação de conhecimento, porém a falta de documentação impede a retenção de um conhecimento preciso durante o desenvolvimento de software. O modelo de representação do DR proposto neste item tem o objetivo também de capturar este conhecimento.

Figura 17 – Visão geral do DOC-DR

Fonte: Autora

4.1.3.5. Incremento

O Incremento é a soma de todos os itens do Backlog do Produto completados durante a Sprint.

Ao final da Sprint, um novo incremento deve estar pronto, ou seja, deve estar na condição utilizável.

4.1.3.6. Plano de Melhoria da Sprint

A Tabela 10 mostra o modelo proposto para o Plano de Melhoria da Sprint. Se ainda não houver uma sugestão de solução para o problema, o campo “Solução” deve ficar em branco.

Tendo uma solução do problema, define-se a data em que ela será implantada. Geralmente isso ocorre já na próxima Sprint, a fim de se melhorar o processo corrente e evitar a recorrência do problema.

Além de problemas relatados por participantes da reunião, devem ser incluídos no Plano de Melhoria da Sprint casos de desvio de desempenho apontados pelas medições realizadas.

O campo “Organização” indica se a melhoria é candidata a se tornar um padrão para a organização, ou seja, não só o projeto atual pode se beneficiar com ela, mas outros projetos

também. Nestes casos, a melhoria pode ser levada a discussão na reunião de Retrospectiva do Projeto.

Tabela 10 - Plano de Melhoria da Sprint

Plano de Melhoria da Sprint	
Campos	Descrição
Nome	Descrição resumida
Problema	Descrição do problema encontrado
Solução	Descrição de como resolver e quais os impactos
Data de implantação	Data em que a solução será implantada
Organização	Indica se a melhoria é candidata a se tornar padrão na organização. Valores possíveis: S (Sim), N (Não), T (Talvez)

Fonte: Autora

4.1.3.7. Plano de Melhoria

O Plano de Melhoria contém ações a serem tomadas no nível organizacional. Contém sugestões de melhorias a problemas detectados no projeto corrente, mas que podem ser expandidos para outros projetos também.

No campo “Justificativa”, documentam-se as razões de se implementar a melhoria, incluindo evidências de seu sucesso, caso existam. A Tabela 11 mostra as informações contidas no Plano de Melhoria.

Tabela 11 - Plano de Melhoria

Plano de Melhoria	
Campos	Descrição
Nome	Descrição resumida
Problema	Descrição do problema encontrado
Solução	Descrição de como resolver e quais os impactos
Justificativa	Razões para adoção da melhoria

Fonte: Autora

4.1.4. Eventos

A definição dos eventos no Scrum-DR, assim como no Scrum, tem o objetivo de criar uma rotina e minimizar a necessidade de reuniões não definidas no Scrum. Todos os eventos tem uma duração máxima, a fim de mantê-los produtivos e focados. Além disso, os eventos são projetados para permitir transparência e inspeção criteriosa. É uma oportunidade de inspecionar, adaptar e melhorar produtos e processos.

Inicialmente, ocorrem as Reuniões de Definição, com o objetivo de obter uma visão geral do projeto e definir as informações gerais (papéis, reuniões, medições e lista inicial de requisitos). Realizam-se também definições sobre as configurações das ferramentas de trabalho.

Sprint é o evento principal do *framework*. Tem a duração de no máximo um mês e ao final dele uma versão incremental potencialmente utilizável do produto é criado. Este limite de tempo tem o objetivo de minimizar a complexidade e o risco. As Sprints são compostas por uma Reunião de Planejamento, Implementação, Reuniões Diárias, a Revisão da Sprint e a Retrospectiva da Sprint. A Figura 18 mostra o evento Sprint com todos os seus sub-eventos e também os artefatos de entrada e saída de cada um deles.

Os tempos definidos nos sub-eventos de uma Sprint foram baseados em uma Sprint de um mês e são semelhantes aos tempos definidos no Scrum. Para Sprint menores, esses tempos devem ser proporcionalmente menores.

Finalmente, após as Sprints, ocorre a Retrospectiva do Projeto, com o objetivo de disseminar o conhecimento do projeto corrente para a organização. A seguir, estes eventos serão descritos com mais detalhes.

Figura 18 – Sprint no Scrum-DR

Fonte: Autora

4.1.4.1. Reunião de Definição

Inicialmente são realizadas as reuniões para definição da visão geral do projeto e criação do Plano do Projeto e lista inicial dos requisitos (Backlog do Produto). A Figura 19 ilustra o evento mostrando os artefatos de saída.

Figura 19 – Reunião de Definição

Fonte: Autora

Usualmente mais de uma reunião é necessária até de se obter as informações básicas do projeto. As informações, principalmente os requisitos, são refinadas de forma iterativa.

Nestas reuniões são estabelecidos também os objetivos de medição de dados. Estes objetivos podem variar de um projeto para outro, mas é importante coletar pelo menos as informações básicas de desempenho do projeto, como desvios de tempo de desenvolvimento previsto e o real, número de bugs encontrados, tempo para resolver bugs, etc. Além disso, definem-se critérios para a tomada de ações de melhoria, como por exemplo, número máximo de bugs por Incremento. A coleta destas informações é realizada diretamente na ferramenta de controle de tarefas e bugs. A ferramenta deve estar configurada antes do início das Sprints.

Estas reuniões iniciais não tem um tempo máximo definido, pois varia muito dependendo do tamanho do projeto, conhecimento do negócio e entendimento do cliente. Entretanto, é importante manter o foco, tornando-as as mais objetivas possíveis. O Scrum Master é responsável por assegurar que as reuniões não percam as características ágeis mas tenham o detalhamento necessário para início das Sprints.

4.1.4.2. Reunião de Planejamento da Sprint

Na Reunião de Planejamento da Sprint, define-se o trabalho que será realizado na Sprint. Este planejamento é realizado com a colaboração de todo o Time Scrum e tem a duração máxima de 8 horas. A principal entrada da Reunião de Planejamento é o Backlog do Produto. Como resultado desta reunião, obtém-se o Backlog da Sprint, que é formada pelo conjunto de itens do Backlog do Produto selecionados para a Sprint mais o plano de entrega. Além disso, decisões importantes tomadas nesta reunião são inseridas no DOC-DR.

4.1.4.3. Implementação e Reunião Diária

Terminada a reunião de planejamento da Sprint, o Time de Desenvolvimento passa a implementar as funcionalidades definidas e então, realizam-se reuniões diárias de 15 minutos com o objetivo de inspecionar o trabalho desde a última reunião e prever o trabalho que deverá ser feito antes da próxima reunião diária.

Durante o trabalho de construção do software, o Time de Desenvolvimento utiliza técnicas que auxiliam a agilidade e qualidade do produto:

- Programação em pares: possibilita uma revisão do código desenvolvido além de ser uma maneira de disseminação do conhecimento;
- Testes antecipados: testes automatizados e escritos antes do código;
- Testes de integração constantes: builds automatizados diários, com o intuito de detectar não conformidades rapidamente;
- Técnicas de DR: auxiliando a tomada de decisões e evitando a consideração de alternativas já descartadas anteriormente;
- Intensa comunicação: geralmente face a face, entre todos os envolvidos, facilitando o entendimento;

Durante a Reunião Diária, os membros do Time de Desenvolvimento esclarecem:

1. O que eu fiz ontem que ajudou a atender a meta da Sprint?
2. O que eu farei hoje para ajudar a atender a meta da Sprint?
3. Eu vejo algum obstáculo que impeça a mim ou o Time de Desenvolvimento no atendimento da meta da Sprint?

O Scrum Master assegura que a reunião ocorra e que todos os envolvidos entendem o seu funcionamento. Esta reunião é essencial para o funcionamento do *framework*.

4.1.4.4. Revisão da Sprint

A Revisão da Sprint é executada no final da Sprint para verificar o incremento e, se necessário, atualizar o Backlog do Produto. Esta reunião deve durar no máximo 4 horas. Participam o Time Scrum e as partes interessadas. É uma reunião informal com o objetivo de promover a colaboração.

O Product Owner apresenta os itens do Backlog do Produto que foram prontos. O Time de Desenvolvimento demonstra o trabalho que está pronto e responde as questões sobre o

incremento. Finalmente, o grupo todo colabora sobre o que fazer a seguir, atualizando o Backlog do Produto se necessário.

É uma oportunidade também de expor as principais decisões do projeto realizadas na Sprint e estas informações são capturadas no DOC-DR. O resultado da Reunião de Revisão da Sprint é um Backlog do Produto revisado que será a entrada para a próxima Sprint.

4.1.4.5. Retrospectiva da Sprint

A Retrospectiva da Sprint ocorre depois da Revisão da Sprint e dura até 3 horas. É uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano de melhorias a serem aplicadas na próxima Sprint. O objetivo desta reunião é inspecionar a última Sprint, identificando e ordenando os principais itens que foram bem e as potenciais melhorias e criar um plano para implementar essas melhorias.

Durante esta reunião, o Time de Desenvolvimento discute o que foi bem durante a Sprint, quais problemas ocorreram e como foram solucionados. As medições sobre desempenho da Sprint também são avaliadas. Cria-se, então, o Plano de Melhoria da Sprint no modo que o Time Scrum faz seu trabalho. As melhorias identificadas são implementadas na próxima Sprint.

Algumas ideias surgidas aqui podem ser importantes para melhorar o produto e o processo tanto do projeto corrente como definir melhorias para toda a organização. Neste momento, o foco é o projeto atual, mas deve-se sempre estar atento a melhorias que podem se tornar padrão a todos os processos da empresa.

A Retrospectiva da Sprint é um evento focado na inspeção e adaptação.

4.1.4.6. Retrospectiva do Projeto

Após a sequência de Sprints, quando toda a implementação do projeto é finalizada, ocorre a reunião de Retrospectiva do Projeto. Este é um evento adicionado ao Scrum-DR e é de vital

importância para a melhoria contínua de processos visando à corporação. É uma reunião de no máximo 3 horas.

Participam desta reunião o Time Scrum do projeto e também Scrum Master de outros projetos da empresa. O objetivo é discutir sobre acontecimentos que foram positivos no projeto e também os que foram negativos. Neste último caso, incluem-se sugestões de melhoria se houver. O foco desta reunião é em compartilhar situações que possam ser úteis a outros projetos.

Enquanto na reunião de Retrospectiva da Sprint o foco é no produto e projeto corrente, na reunião de Retrospectiva do Projeto o foco são os produtos e processos da organização. É claro que parte da discussão e documentos gerados na Retrospectiva da Sprint podem ser utilizados na Retrospectiva do Projeto.

A pressão por prazos e ansiedade para iniciar um novo projeto podem acabar negligenciando este importante evento. Ele é responsável por evitar que se cometa um mesmo erro em projetos futuros. É uma forma eficaz de disseminar as lições aprendidas, buscando sempre a melhoria contínua. Por isso, é vital o papel do Scrum Master, garantindo que a reunião ocorra e que todos os envolvidos entendam a sua importância. Do mesmo modo, é necessário que todos da equipe colaborem na reunião, compartilhando suas experiências e opiniões.

Para apoiar a reunião, dois documentos são utilizados: DOC-DR e Plano de Melhoria da Sprint. Eles podem auxiliar a lembrar eventos passados que podem ser discutidos nesta reunião.

A Figura 20 mostra o evento Retrospectiva do Projeto no Scrum-DR, com seus artefatos de entrada e saída.

Fonte: Autora

4.1.5. Visão geral

A Figura 21 mostra o fluxo de atividades do Scrum-DR, destacando os eventos e os principais artefatos gerados. Nesta figura, é possível ter uma visão do *framework* Scrum-DR como um todo.

Figura 21 – Visão geral do Scrum-DR

Fonte: Autora

Foi indicado o local de criação de cada artefato. Entretanto, a consulta e atualização dos documentos podem ocorrer em qualquer momento. É o que ocorre principalmente com o Backlog do Produto e o DOC-DR.

Não houve muita alteração nos eventos e fluxo de atividades do Scrum. A extensão ocorre nos papéis e dentro de cada evento, conforme visto anteriormente. Foi incluído o evento Retrospectiva do Projeto, importante passo para melhoria contínua da qualidade e disseminação de informação sobre lições aprendidas em projetos anteriores. Além disso, definiram-se três fases distintas: Iniciação, Desenvolvimento e Finalização. Esta divisão enfatiza a importância de todos os eventos.

O *framework* todo tem característica de desenvolvimento iterativo, realizando refinamentos a cada iteração. Isso evita que seja realizada uma documentação extensa antes de se ter uma visão clara das funcionalidades desejadas. Ele é simples e ágil, com o mínimo de documentação necessária.

Para o acompanhamento das atividades do Scrum-DR, é importante adotar uma ferramenta de gerenciamento de projeto. A rastreabilidade é uma característica que possibilita uma melhor gestão do projeto, a realização de estimativas e medições, além de fornecer um ótimo auxílio para a documentação do software.

Uma ferramenta que atende muito bem as necessidades do Scrum-DR é o Jira Software (2017). É possível definir o fluxo de trabalho, acompanhar as tarefas e gerar relatórios de uma maneira simples e rápida. E, se necessário, a ferramenta pode ser customizada de diversas formas, inclusive alterando o fluxo de eventos do desenvolvimento.

4.2. Análise e Aplicação do Scrum-DR

O objetivo deste capítulo é analisar e aplicar o *framework* proposto no item 4.1. Primeiramente, realiza-se uma análise do *framework* proposto em relação às lacunas entre métodos ágeis e CMMI na categoria Suporte. Finalmente, descreve-se uma aplicação do *framework* em um projeto fictício a fim de avaliar a sua aplicabilidade. Ajustes ao modelo podem ser sugeridos decorrentes da análise deste estudo de caso.

4.2.1. Scrum-DR X CMMI-DEV

Neste item, analisa-se se o Scrum-DR cobre as lacunas entre métodos ágeis e CMMI-DEV V1.3 na categoria Suporte, apontadas no item 3.2.4.

O foco desta pesquisa é nas cinco áreas de processo da categoria Suporte: Análise e Resolução de Causas (CAR), Gestão de Configuração (CM), Análise e Tomada de Decisão (DAR), Medição e Análise (MA) e Garantia da Qualidade de Processo e Produto (PPQA). Nesta categoria, conforme apresentado no Capítulo 2, há áreas de processo de nível de maturidade 2 (CM, MA e PPQA), 3 (DAR) e 5 (CAR). Segue a análise em cada uma dessas áreas. Para a conclusão, utilizou-se a mesma simbologia e classificação apresentada no item 3.2.

1. Análise e Resolução de Causas (CAR)

O Scrum oferece um mecanismo eficiente para detecção e análise de ocorrências durante o projeto pela intensa comunicação de seus participantes e também através da reunião de Retrospectiva da Sprint. Nesta reunião, identificam-se os problemas ocorridos e ações são tomadas já na próxima Sprint a fim de se melhorar o processo do projeto atual. Entretanto, o foco é no projeto corrente e não havia um mecanismo determinado para que este conhecimento chegasse a outros projetos ou se tornasse um padrão para toda a organização.

O Scrum-DR, através da Retrospectiva do Projeto, oferece uma maneira eficaz de identificar causas de defeitos, implementando melhorias visando tanto o projeto atual como outros projetos. O intuito é que haja uma transferência de conhecimento de um projeto para o outro, sempre contribuindo para a melhoria de produtos e processos.

A utilização do DOC-DR é outra maneira importante de se identificar causas de problemas em produtos e evitar a tentativa de uma alternativa já descartada em outras ocasiões.

Scrum-DR apresenta modelo para gravação destas informações, através do DOC-DR, Plano de Melhoria da Sprint e Plano de melhoria.

Conclusão: Totalmente suportado [+++].

2. Gestão de Configuração (CM)

A Gestão de Configuração é particularmente importante em ambiente ágeis por causa da necessidade de suportar mudanças frequentes, *builds* frequentes (geralmente diários) e múltiplos ambientes de trabalho (individuais, por equipe e até por pares de programação) (CMMI-DEV V1.3, 2010).

A integridade de produtos de trabalho no Scrum-DR é obtida pela proximidade do cliente e intensa comunicação entre os envolvidos. O *framework* está preparado para mudanças nos requisitos e são rapidamente integradas. A documentação destas mudanças é realizada no Backlog do Produto. Além disso, as alterações mais importantes são também documentadas no DOC-DR, incluindo as razões das mesmas.

Conclusão: Totalmente suportado [+++].

3. Análise e Tomada de Decisões (DAR)

Apesar das metodologias ágeis não oferecerem diretrizes para a análise e tomada de decisões, no Scrum-DR estas informações são coletadas através do DOC-DR. O modelo de representação apresentado auxilia no momento em que um problema está sendo discutido, apontando a melhor solução. O modelo apresentado reduz a natureza subjetiva de uma decisão e proporciona uma probabilidade maior de selecionar a melhor solução de um problema.

Além disso, esta informação é capturada de uma forma gráfica e fica disponível para futuras consultas.

O modelo apresentado para representação do DR pode ser utilizado tanto para assuntos técnicos como para não técnicos.

Conclusão: Totalmente suportado [+++].

4. Medição e Análise (MA)

A princípio, a área de processo Medição e Análise não era totalmente suportada no Scrum-DR, por ter o foco no produto e entrega do projeto corrente. Entretanto, após a análise cuidadosa desta área de processo, verificou-se que era possível sim atendê-la através de práticas que não onerariam tanto o processo. Então, foi realizada uma revisão no *framework* Scrum-DR para incorporar estas práticas.

As medições de progresso por ciclo podem ser facilmente obtidas através da utilização da ferramenta de controle de tarefas e bugs. Ferramentas como o JIRA SOFTWARE (2017) oferecem maneiras rápidas de se verificar desempenho real contra projetos e objetivos planejados, tempo para resolver bugs, número de bugs por Sprint ou projeto, etc. Desse modo, há apenas um tempo a mais na fase Iniciação que deve ser utilizado para definição do que será medido e também para configuração da ferramenta. Durante as Sprints, pouco tempo a mais é necessário para se ter esta informação disponível, que pode ser muito útil.

As informações sobre as medições ficam disponíveis na ferramenta e as mais importantes são incorporadas ao Plano de Melhoria da Sprint e são então analisadas na Retrospectiva da Sprint.

De maneira semelhante às outras informações do Plano de Melhoria da Sprint, as informações que podem ser úteis para toda a organização são sinalizadas no campo “Organização” e então são analisadas na Retrospectiva do Projeto. Então, estas informações podem ser utilizadas para a tomada de decisões com impacto para toda a empresa.

Por ser uma solução que impacta pouco o dia a dia dos desenvolvedores, ela mantém o *framework* ágil.

Conclusão: Totalmente suportado [+++].

5. Garantia da Qualidade de Processo e Produto (PPQA)

Existem várias técnicas no *framework* Scrum-DR que auxiliam a avaliação objetiva dos produtos de trabalho e garante qualidade nos processos e produtos.

Para a garantia da qualidade de produto:

- A programação em pares: possibilita uma checagem adicional do código escrito;
- Testes antecipados e testes de integração constantes: detectam rapidamente não conformidades no código que acabou de ser finalizado;
- Técnicas de DR: fornecem suporte para decisões do projeto;
- Intensa comunicação: a comunicação entre os envolvidos é intensa, geralmente face a face, facilitando o entendimento;
- Cliente próximo: cliente participa ativamente de todo o processo e está próximo fisicamente, facilitando a comunicação e aumentando as chances de que o software construído está de acordo com as suas expectativas.

Para a garantia de qualidade dos processos:

- Coleta e análise de medições de desempenho auxiliam a detectar problemas no processo e a melhorá-lo;
- A Retrospectiva da Sprint é um evento totalmente focado na identificação, análise e criação de ações corretivas para ocorrências da Sprint; é uma maneira de melhorar o processo na próxima Sprint;
- A Retrospectiva do Projeto visa à melhoria de processo da organização, disseminando o conhecimento adquirido em um projeto;

- O Plano de Melhoria da Sprint e o Plano de Melhoria registram de maneira formal ocorrências que foram positivas e também negativas, buscando o aprimoramento sempre;
- O uso de ferramenta para controle de tarefas e bugs seguindo o *framework* Scrum-DR facilita e possibilita a identificação de resultados das atividades de garantia da qualidade.

Conclusão: Totalmente suportado [+++].

É possível verificar que o Scrum-DR fornece suporte para todas as áreas de Processo do CMMI-DEV na categoria Suporte. A Tabela 12 mostra uma comparação da aderência dos métodos ágeis e do Scrum-DR com o CMMI-DEV 1.3.

Tabela 12 - Aderência CMMI X Métodos Ágeis X Scrum-DR

Área de Processo CMMI	Métodos Ágeis	Scrum-DR
Análise e Resolução de Causas (CAR)	[-] Conflitante	[+++] Totalmente Suportado
Gestão de Configuração (CM)	[++] Suportado	[+++] Totalmente Suportado
Análise e Tomada de Decisões (DAR)	[-] Conflitante	[+++] Totalmente Suportado
Medição e Análise (MA)	[-] Conflitante	[+++] Totalmente Suportado
Garantia da Qualidade de Processo e Produto (PPQA)	[+] Parcialmente suportado	[+++] Totalmente Suportado

Fonte: Autora

4.2.2. Estudo de caso

Neste item é realizado um estudo de caso de aplicação do *framework* proposto a fim de ilustrar a sua utilização. Esta aplicação do *framework* também tem o propósito de analisar o *framework* e, se necessário, ajustá-lo.

A aplicação escolhida foi o Sistema de Gerenciamento de Hotel (SGH) descrito por Jacobson e Ng (2004). Escolheu-se este sistema por ilustrar uma variedade de sistemas semelhantes, como aluguel de carros, jogos, locadora, etc.

As principais funcionalidades do SGH são a reserva de quarto, *check in* e *check out*. O sistema pode ser utilizado tanto pelos clientes como pelos funcionários do hotel. Para simplificar, somente o cliente pode fazer reserva. A Figura 22 ilustra a visão geral do sistema.

Alguns detalhes serão acrescentados mais para frente, seguindo o modelo iterativo. Optou-se por considerar que nem todas as informações ou detalhes foram definidos no início do projeto, a fim de mostrar a dinâmica das alterações nos requisitos.

Fonte: Jacobson e Ng (2004)

As versões finais dos documentos deste estudo de caso encontram-se no APÊNDICE B – Plano de Projeto do SGH utilizando Scrum-DR, APÊNDICE C – Backlog do Produto do

SGH utilizando Scrum-DR, APÊNDICE D – DOC-DR do Projeto do SGH utilizando Scrum-DR e APÊNDICE E – Backlogs da Sprint do Projeto do SGH utilizando Scrum-DR.

4.2.2.1. Iniciação

Na fase de Iniciação, através das Reuniões de Definição, definem-se as informações gerais do projeto, incluindo papéis e funcionalidades do sistema. Estas reuniões não têm um tempo definido, pois variam muito dependendo do projeto. Participam todos os interessados no projeto.

Logo nas primeiras reuniões, obtém-se um entendimento melhor das principais funcionalidades do sistema:

- Para reservar um quarto, checka-se a disponibilidade do mesmo, e se o tipo de quarto estiver disponível, a reserva é criada.
- Para realizar o *check in* de um cliente, ele deve ser associado a um quarto e então a sua reserva é processada. Ao mesmo tempo, cria-se uma conta inicial para o cliente.
- Para realizar o *check out* de um cliente, o cliente realiza o pagamento da conta e então o cliente é removido do quarto.

A funcionalidade de reserva do quarto é a principal do sistema. A Tabela 13 descreve esta funcionalidade com mais detalhes.

Tabela 13 – Reservar quarto no SGH

Funcionalidade	Reservar quarto (RES001)
Descrição resumida	Reserva de um quarto pelo cliente
Fluxo básico	1- Cliente seleciona a opção de reserva de um quarto 2- O sistema apresenta os tipos de quartos do hotel e seus valores 3- Cliente verifica os valores dos quartos 4- Cliente realiza a reserva para o quarto escolhido 5- O sistema subtrai do banco de dados o número de quartos disponíveis para reserva do tipo selecionado 6- O sistema cria uma nova reserva com os detalhes fornecidos 7- O sistema apresenta o número de confirmação da reserva e as instruções de check-in 8- O evento é finalizado
Fluxo alternativo	5.1- Submissão duplicada: se no passo 5 do fluxo básico ocorrer uma reserva idêntica no sistema (mesmo nome, e-mail, e datas de entrada e saída), o sistema mostra a reserva existente e pergunta ao cliente se ele quer proceder com a nova reserva. 5.1.1- Se o cliente quiser continuar, o sistema executa a reserva 5.1.2- Se o cliente informar que a nova reserva é duplicada, a reserva é cancelada
Subfluxos	3.1- Checar custo do quarto 3.1.1- O cliente seleciona o tipo de quarto desejado e indica o seu período de permanência 3.1.2- O sistema calcula o custo para o período especificado
Pré-condições	O cliente fez <i>log in</i> no sistema
Pós-condições	Se a reserva finalizar com sucesso, um novo registro de reserva é criado e é decrementado o número de quartos disponíveis do mesmo tipo e para as datas especificadas. Se a reserva não for finalizada com sucesso, não há mudança na base de dados.
Requisitos especiais	O sistema deve ser capaz de manipular cinco reservas concorrentes. Cada reserva não deve demorar mais que 20 segundos.

Fonte: Jacobson e Ng (2004)

Os requisitos não funcionais do SGH são apresentados na Tabela 14. Em “Prioridade”, o menor valor indica a maior prioridade. É possível verificar que o requisito de maior prioridade

é a acessibilidade via Internet, pois todas as reservas são executadas pelo cliente, via Internet. Em seguida, a prioridade são os requisitos de desempenho e número de reservas concorrentes.

Os requisitos não funcionais também são adicionados ao Backlog do Produto.

Tabela 14 – Requisitos não funcionais do SGH

Requisito	Prioridade
Acessibilidade do sistema via Internet.	1
O sistema deve armazenar todos os registros em um banco de dados relacional.	1
Somente pessoas autorizadas podem desempenhar as suas funções no sistema.	3
Todas as recuperações de dados não devem exceder 2 segundos.	2
O sistema deve manipular até cinco reservas concorrentes.	2
Cada reserva não deve exceder 20 segundos.	2

Fonte: Jacobson e Ng (2004)

Ao final desta fase, obtém-se o Plano de Projeto e o Backlog do Produto.

A Tabela 15 ilustra o Plano de Projeto do SGH. Estas informações ficam disponíveis a todos. Algumas informações como Data Inicial, papéis, restrições, reuniões e medições são dados fictícios acrescentados para ilustrar o Plano do Projeto.

Tabela 15 - Plano do Projeto do SGH

Plano do Projeto do SGH	
Campos	Descrição
Nome	Sistema de Gerenciamento de Hotel
Descrição	Sistema de reservas de quartos no hotel. Será utilizado por clientes e funcionários. As funcionalidades principais são reserva de quarto, <i>check in</i> e <i>check out</i> .
Data Inicial	15/01/2016
Data de Término	30/03/2016
Objetivos	Facilitar e automatizar a reserva de quartos
Restrições	A data final do projeto não pode ultrapassar 30/04/2016.
Riscos	
Reuniões	- Reunião de planejamento da Sprint: Primeira segunda-feira do mês, 09:00, Sala 10. - Reunião Diária: Segunda a sexta, 14:00, sala 5.
Medições	- Número de bugs encontrado em uma Sprint não pode ultrapassar 10. - Número de bugs abertos não pode ultrapassar 10.
Product Owner	Luísa
Scrum Master	Matheus
Time de Desenvolvimento	Fabiano, Cecília e Luiz
Clientes	Julia e Wal
<i>Stakeholders</i>	Luciane e Reginaldo

Fonte: Autora

A versão final do Plano de Projeto do SGH pode ser encontrada no APÊNDICE B – Plano de Projeto do SGH utilizando Scrum-DR,

A Tabela 16 mostra o Backlog do Produto em sua primeira versão, criado na fase de Iniciação.

Tabela 16 – Iniciação: Backlog do Produto

Backlog do Produto					
Sprint	Nome	Descrição	Tipo	Prioridade	Grupo
	Cadastro de quartos	Vide CAD001	F	1	Cadastro
	Cadastro de Cliente	Vide CAD003	F	1	Cadastro
	Reserva de quarto	Vide RES001	F	2	Reserva
	<i>Check in</i>	Cliente é associado a um quarto e então a sua reserva é processada. Ao mesmo tempo, cria-se uma conta inicial para o cliente.	F	2	Check in/out
	<i>Check out</i>	O cliente realiza o pagamento da conta e então o cliente é removido do quarto.	F	2	Check in/out
	Estabelecer taxas	Gerente do Hotel define custos de cada quarto	F	1	Cadastro
	Internet	Acessibilidade do sistema via Internet.	NF	1	Acessibilidade
	BD Relacional	O sistema deve armazenar todos os registros em um banco de dados relacional.	NF	1	Base de Dados
	Pessoas autorizadas	Somente pessoas autorizadas podem desempenhar as suas funções no sistema	NF	3	Segurança
	Recuperação de dados	Todas as recuperações de dados não devem exceder 2 segundos.	NF	2	Desempenho
	Reservas concorrentes	O sistema deve manipular até cinco reservas concorrentes.	NF	2	Desempenho
	Tempo reserva	Cada reserva não deve exceder 20 segundos.	NF	2	Desempenho

Fonte: Autora

4.2.2.2. Desenvolvimento

Nesta fase, temos a sequência de Sprints, conforme Figura 21.

Sprint 1

Na primeira Reunião de Planejamento da Sprint o Time de Desenvolvimento realiza definições sobre plataforma de desenvolvimento e banco de dados.

A Figura 23 ilustra o DR sobre a questão da plataforma de desenvolvimento. A equipe de desenvolvimento questiona o Product Owner se o sistema será acessado via dispositivos móveis. Ele responde negativamente já que nunca houve uma solicitação de seus clientes. A plataforma Cliente/Servidor também é considerada, mas como a única necessidade do sistema é a acessibilidade pela Internet e como a plataforma Web é uma tecnologia já dominada pela equipe de desenvolvimento, ela é a escolhida.

Figura 23 – SGH: DR sobre plataforma de desenvolvimento

Fonte: Autora

Para o banco de dados, define-se o banco de dados relacional que o cliente já possui. Este foi um requisito não funcional de prioridade máxima.

Para o acompanhamento das tarefas, do projeto e dos bugs, utiliza-se a ferramenta JIRA. Ela possibilita um controle rápido sobre o andamento do projeto e auxilia a coleta das medições.

Nesta reunião, define-se também como serão implementadas as funcionalidades principais do sistema (Vide Figura 24). Esta informação integra o Plano de Entrega.

Figura 24 – SGH: Implementação das funcionalidades principais

	Quarto	Reserva	Pagamento
Reserva de quarto	checarDisponibilidade()	criar()	
Check in do Cliente	associarCliente()	consumir()	criarConta()
Check out do Cliente	removerCliente()		pagarConta()

Fonte: Jacobson e Ng (2004)

Finalmente, determinam-se quais itens do Backlog do Produto serão implementados na Sprint. A escolha é feita pelas prioridades, juntamente com o cliente. As funcionalidades escolhidas estão na Tabela 17.

Tabela 17 – Itens implementados na Sprint 1

Sprint 1				
Nome	Descrição	Tipo	Prioridade	Grupo
Cadastro de quartos	Vide CAD001	F	1	Cadastro
Cadastro de Cliente	Vide CAD003	F	1	Cadastro
Estabelecer taxas	Gerente do Hotel define custos de casa quarto	F	1	Cadastro
Internet	Acessibilidade do sistema via Internet.	NF	1	Acessibilidade
BD Relacional	O sistema deve armazenar todos os registros em um banco de dados relacional.	NF	1	Base de Dados

Fonte: Autora

Em seguida, passa-se à implementação das funcionalidades. Primeiramente são feitas as definições do Banco de Dados e criação das tabelas. Em seguida, cria-se a aplicação principal com a tela principal. Finalmente, passa-se à criação do Cadastro de quartos e Cadastro de Cliente.

A Reunião Diária ocorre todos os dias, no mesmo local e horário. A intensa comunicação entre a equipe de desenvolvimento e todos os interessados facilita a análise e resolução de problemas, mostrando a aderência com a área de processo Análise e Resolução de Causas.

Na Revisão da Sprint, todos os elementos definidos no Backlog da Sprint são entregues e demonstrados ao cliente. Apesar de estar tudo de acordo com o que foi solicitado, percebe-se que ainda há funcionalidades que o cliente deseja, mas que ainda não estão no Backlog do Produto.

Na Reunião de Retrospectiva da Sprint, decide-se diminuir a duração da Sprint para 2 semanas, pois os ciclos menores provavelmente auxiliarão na definição dos requisitos restantes. Esta informação é incluída no Plano de Melhoria da Sprint (Tabela 18).

Tabela 18 - Plano de Melhoria da Sprint

Plano de Melhoria da Sprint	
Campos	Descrição
Nome	Ciclo mais curto da Sprint
Problema	Dificuldade quando o cliente não tem muita ideia das funcionalidades que necessita.
Solução	Ciclos mais curtos da Sprint facilitam a elicitação dos requisitos faltantes.
Data de implantação	Sprint 2
Organização	S (Sim)

Fonte: Autora

Sprint 2

A equipe se depara com uma questão: “Qual o momento que o cliente deve fazer *log in* no sistema”. A dúvida é se deve fazer assim que acessa o sistema ou só no momento de fechar a reserva. A Figura 25 ilustra o DR desta questão, que auxiliou a todos os envolvidos raciocinarem sobre o problema. Assim que ela foi colocada em discussão, o cliente informou que gostaria de rastrear as consultas executadas pelos clientes para que em uma nova busca, fizesse algumas sugestões. Além disso, informou que utilizam um sistema de fidelidade e oferece bônus para as próximas reservas, então, é importante a identificação do cliente no início para que os valores já se apresentem com os descontos. Finalmente, informou que todas as transações devem ser registradas com a finalidade de realizar auditorias.

A Figura 25 irá compor o DOC-DR. O modelo de captura do DR apresentado no Scrum-DR auxiliou a descoberta da solução e também a identificar novas informações que ainda não haviam sido mencionadas. Quatro novos requisitos do sistema foram acrescentados ao Backlog do Produto (Vide Tabela 19).

Figura 25 – SGH: DR sobre *log in* no sistema

Fonte: Autora

É possível verificar que a utilização de técnicas de DR no Scrum-DR fornece subsídios para a tomada de decisões no projeto. Além disso, esta informação é capturada e fica disponível para todos. Esta característica possibilita a aderência do Scrum-DR à área de processo Análise e Tomada de Decisões (DAR).

Tabela 19 – Sprint 2: inclusões ao Backlog do Produto

Backlog do Produto				
Nome	Descrição	Tipo	Prioridade	Grupo
Programa de Fidelidade	Vide FID001	F	4	Reserva
<i>Log in</i> no sistema	Vide LOG001	F	1	Início
Transações registradas	Para fins de auditoria, todas as transações do sistema devem ser registradas.	NF	4	Segurança
Preferências do cliente	O sistema deve rastrear as preferências do cliente e utilizá-las como <i>default</i> .	NF	5	Usabilidade

Fonte: Autora

O *framework* Scrum-DR fornece mecanismos para acomodar mudanças que são rapidamente integradas ao projeto. A documentação destas mudanças é realizada no Backlog do Produto. Além disso, alterações importantes são também documentadas no DOC-DR, explicitando as razões destas mudanças. Portanto, o Scrum-DR é aderente à área de processo Gestão de Configuração.

Os itens selecionados para implementação na Sprint 2 estão na Tabela 20.

Tabela 20 – Itens implementados na Sprint 2

Sprint 2				
Nome	Descrição	Tipo	Prioridade	Grupo
Reserva de quarto	Vide RES001	F	2	Reserva
Log in no sistema	Vide LOG001	F	1	Início
Transações registradas	Para fins de auditoria, todas as transações do sistema devem ser registradas.	NF	4	Segurança
Pessoas autorizadas	Somente pessoas autorizadas podem desempenhar as suas funções no sistema	NF	3	Segurança

Fonte: Autora

Continua o ciclo do Scrum-DR, com as Reuniões Diárias até a entrega do Incremento, Revisão da Sprint e Retrospectiva da Sprint.

Durante a Retrospectiva da Sprint, a ferramenta de desenvolvimento verifica que o número de bugs abertos é maior que 10 e envia um relatório para o *email* do Scrum Master e os desenvolvedores. Analisando a causa, verifica-se que foi devido principalmente à complexidade maior da funcionalidade “Reserva de quarto”. Então, o Time de Desenvolvimento decide dedicar três dias inteiros para a resolução destes bugs.

As medições realizadas pela ferramenta de desenvolvimento podem checar desempenho do projeto e realizar a captura de informações sobre bugs, oferecendo uma maneira rápida de se adequar e melhorar produto e processo. Esta prática é aderente às áreas de processo Medição e Análise e Garantia da Qualidade de Processo e Produto.

Sprint 3

Na Sprint 3, a equipe já possui uma boa comunicação e entendimento do sistema. As funcionalidades escolhidas para a Sprint estão na Tabela 21.

Tabela 21 – Itens implementados na Sprint 3

Sprint 3				
Nome	Descrição	Tipo	Prioridade	Grupo
<i>Check in</i>	Cliente é associado a um quarto e então a sua reserva é processada. Ao mesmo tempo, cria-se uma conta inicial para o cliente.	F	2	Check in/out
<i>Check out</i>	O cliente realiza o pagamento da conta e então o cliente é removido do quarto.	F	2	Check in/out
Programa de Fidelidade	Vide FID001	F	4	Reserva
Preferências do cliente	O sistema deve rastrear as preferências do cliente e utilizá-las como <i>default</i> .	NF	5	Usabilidade

Fonte: Autora

Durante a Revisão da Sprint, o cliente percebe mais uma funcionalidade que necessita para o sistema e que agregará valor ao seu negócio. Se durante uma reserva, o tipo de quarto solicitado não estiver disponível, o cliente vai para uma lista de espera. Caso o quarto seja liberado a tempo, o cliente é contatado. Veja Tabela 22.

Tabela 22 – Sprint 3: inclusão ao Backlog do Produto

Backlog do Produto				
Nome	Descrição	Tipo	Prioridade	Grupo
Manipulação de Lista de Espera	Se não tiver quarto disponível, cliente vai para a lista de espera	F	3	Reserva

Fonte: Autora

Sprint 4

Para a última Sprint, será implementado o controle de reservas concorrentes, a fim de se ter até cinco reservas sendo atendidas ao mesmo tempo. Além disso, nesta Sprint há a verificação

dos requisitos de desempenho do sistema. É claro durante todo o desenvolvimento estes requisitos foram levados em consideração, mas é neste momento que serão oficialmente validados. A

Tabela 23 mostra as funcionalidades implementadas na Sprint 4.

Tabela 23 – Itens implementados na Sprint 4

Sprint 4				
Nome	Descrição	Tipo	Prioridade	Grupo
Recuperação de dados	Todas as recuperações de dados não devem exceder 2 segundos.	NF	2	Desempenho
Reservas concorrentes	O sistema deve manipular até cinco reservas concorrentes.	NF	2	Desempenho
Tempo reserva	Cada reserva não deve exceder 20 segundos.	NF	2	Desempenho

Fonte: Autora

Verificou-se que o ciclo mais curto realmente auxiliou o cliente a fechar as funcionalidades necessárias ao sistema.

Todas as funcionalidades do sistema foram implementadas e o Backlog do Produto final está no APÊNDICE C – Backlog do Produto do SGH utilizando Scrum-DR.

4.2.2.3. Finalização

O projeto terminou dentro do prazo máximo definido pelo cliente. As medições de número de bugs por Sprint e número de bugs abertos se mantiveram dentro dos valores aceitáveis, com exceção de uma ocorrência na Sprint 2. Não houve problema com a restrição de prazo e o projeto foi finalizado antes do dia 30/04/2016.

É realizada a Retrospectiva do Projeto com a finalidade de disseminar o conhecimento do projeto corrente. Scrum Masters de outros projetos também são convidados a participar.

A melhoria sobre ciclos mais curtos da Sprint é levada para a reunião e ela se torna padrão na empresa. A

Tabela 24 ilustra as informações do Plano de Melhoria. Este caso exemplifica uma prática aderente à área de processo Análise e Resolução de Causas (CAR) e Garantia da Qualidade de Processo e Produto do CMMI. O problema foi detectado em uma Sprint, a melhoria foi implantada na Sprint seguinte do projeto e na Retrospectiva do Projeto acabou contribuindo para a melhoria do processo da empresa.

Tabela 24 - Plano de Melhoria

Plano de Melhoria	
Campos	Descrição
Nome	Ciclo mais curto da Sprint
Problema	Dificuldade quando o cliente não tem muita ideia das funcionalidades que necessita.
Solução	Ciclos mais curtos da Sprint facilitam a elicitação dos requisitos faltantes.
Justificativa	Solução foi implantada do projeto SGH e os resultados foram positivos.

Fonte: Autora

4.3. Considerações finais

O *framework* proposto teve o objetivo de apresentar práticas e modelos que suprem uma deficiência encontrada nas metodologias ágeis. A categoria Suporte, foco desta pesquisa, é a que mais apresentou conflito entre métodos ágeis e CMMI-DEV e também não foi encontrada nenhuma pesquisa com foco nesta categoria.

O *framework* apresentado aqui é a versão final. A primeira versão sofreu algumas alterações após a análise de aderência do Scrum-DR com o CMMI-DEV (item 4.2.1) e o estudo de caso (item 4.2.2).

Uma das alterações após o estudo de caso foi sobre a “Críticidade” no Argumento do DOC-DR. Inicialmente foi definido como “Peso”, com valores inversos da prioridade do Backlog

do Produto. A fim de simplificar, optou-se por considerar como criticidade e seguir a mesma definição de valores do Backlog do Produto.

O estudo de caso exemplificou várias práticas do Scrum-DR aderentes às áreas de processo do CMMI-DEV na categoria Suporte.

A utilização de um sistema de conhecimento público no estudo de caso auxiliou a validar vários tópicos do *framework* e a ilustrar a sua aplicação.

5. CONCLUSÕES

A análise das áreas de processo do CMMI não atendidas pelos métodos ágeis mostra as lacunas entre eles. É possível verificar quais as áreas demandariam maior esforço para atenderem às diretrizes do CMMI.

Observa-se uma grande lacuna nas áreas de processos com objetivos voltados à organização. Isso se deve ao foco dos métodos ágeis no produto e projeto atual. Para que seja possível aplicar melhorias no processo voltado à organização, é necessário incluir práticas que suportem estas tomadas de decisão.

Embora sejam encontrados alguns modelos de maturidade para os métodos ágeis, não há evidência de trabalhos que foram amplamente aceitos e consolidados na academia ou nas empresas.

O estudo sobre a utilização de métodos ágeis e modelos de qualidade nas empresas, realizado através de um questionário, mostrou que as empresas que adotaram a metodologia ágil estão muito satisfeitas, embora percebam alguns problemas em sua utilização, principalmente sobre padronização e melhoria de processo. O confronto das informações obtidas na pesquisa bibliográfica com a realidade nas empresas foi muito enriquecedor.

O Scrum-DR foi construído com o objetivo de definir práticas e modelos de documentação para suprir as lacunas entre metodologias ágeis e a categoria Suporte do CMMI-DEV. A inclusão de elementos de DR tornou o modelo ágil mais completo, possibilitando a recuperação de informações importantes sobre o projeto. Observou-se que esta era uma grande deficiência nos métodos ágeis. Esta foi a razão de incluir DR ao nome do *framework* proposto.

Esta pesquisa teve caráter interdisciplinar, analisando e confrontando temas importantes ligados ao desenvolvimento de software. A interdisciplinaridade se caracteriza pela intensidade das trocas entre os especialistas e pelo grau de integração real das disciplinas (PACHECO; TOSTA; FREIRE, 2010). A interdisciplinaridade contribui significativamente para o avanço da ciência, principalmente no estudo de temas complexos. Abordagens interdisciplinares procuram respeitar os paradigmas de diferentes disciplinas construindo novas visões de mundo que retroalimentam paradigmas unitários, de maneira a fortalecer não

somente o espaço em que elas conversam, mas principalmente fazendo evoluir suas próprias certezas.

5.1. Principais contribuições

Uma das contribuições deste trabalho foi a análise de aderência dos métodos ágeis com o CMMI-DEV, em todas as categorias. Embora haja alguns trabalhos sobre o assunto, procurou-se fazer uma análise mais detalhada, considerando os argumentos dos trabalhos anteriores encontrados, pois os trabalhos divergiam nas conclusões. Além disso, a análise foi realizada atribuindo cinco valores possíveis: Conflitante, Não suportado, Parcialmente Suportado, Suportado e Totalmente suportado.

Outra contribuição foi a descrição do panorama do desenvolvimento ágil e certificações de qualidade nas organizações, através de entrevistas com pessoas que trabalham em empresas de desenvolvimento de software. Neste estudo, foi possível verificar as dificuldades de se utilizar a metodologia ágil, a importância dos modelos de qualidade de software e como a documentação é realizada na prática.

Finalmente, a principal contribuição deste trabalho foi a definição de um *framework* de desenvolvimento de software ágil que fosse aderente às diretrizes da categoria Suporte do CMMI-DEV. Denominado Scrum-DR, o *framework* tem características dos métodos ágeis ao mesmo tempo em que preenche importantes lacunas para aumentar a aderência com as diretrizes do CMMI. O *framework* apresentou um modelo de representação do DR e diretrizes para se utilizar a técnica de maneira a não sobrecarregar as atividades diárias da equipe de desenvolvimento. A não documentação do DR em projetos ágeis pode causar a perda de informações importantes do projeto e era um ponto de atenção nos métodos ágeis.

Os resultados intermediários desta pesquisa foram apresentados nos II, III, IV e V Workshop de Pós-Graduação da Engenharia de Computação (Engenharia Elétrica – USP). Além disso, uma visão geral do *framework* proposto Scrum-DR foi apresentada no congresso internacional IEEE ChileCon2017, em Pucón, Chile (KAWAMOTO; ALMEIDA, 2017).

5.2. Dificuldades e limitações

A pesquisa apresenta um escopo amplo e interdisciplinar. As informações abordadas não esgotam a totalidade dos respectivos conteúdos que envolvem cada uma dessas disciplinas. Desenvolvimento de software, métodos ágeis, modelos de qualidade de software e documentação de software são caracterizados por uma grande diversidade conceitual, relacionam-se a diversos atores organizacionais e podem sofrer influências de variáveis contextuais, sociais e individuais. Consequentemente, a fim de realizar uma pesquisa envolvendo todas essas disciplinas, inicialmente houve a necessidade de se estudar os conceitos separadamente e só posteriormente partir para a análise conjunta.

Houve certa dificuldade para se conseguir as respostas das empresas ao questionário. Até pela característica ágil do modo de trabalho dos entrevistados, que estão sempre muito focados no produto e suas entregas, não foi fácil conseguir marcar as reuniões com a maioria das pessoas contatadas. Algumas reuniões foram desmarcadas várias vezes.

Por ser a proposta de algo novo, há resistência das empresas em participar de um experimento para estudo em projetos reais, sem a garantia de sucesso na aplicação e nos resultados. A ideia inicial era a aplicação do *framework* proposto em um projeto real, mas este objetivo não foi alcançado.

5.3. Trabalhos Futuros

Durante o desenvolvimento desta pesquisa, foram identificados alguns aspectos que podem dar origem a novos trabalhos, relacionados com o refinamento desta proposta.

Esta pesquisa contempla as áreas de processos da categoria Suporte do CMMI-DEV. Foi possível verificar que a proposta contribui para a aderência a diversas outras áreas de processo, mas um estudo formal das demais áreas avaliaria objetivamente as lacunas existentes nas demais categorias.

De maneira semelhante ao estudo realizado aqui com o CMMI-DEV, pode-se realizar também com outros modelos de qualidade, principalmente o MPS.BR, muito utilizado no Brasil atualmente.

Um passo importante é a criação de mecanismos ou ferramentas para a implementação automatizada do Scrum-DR, a fim de facilitar ainda mais a sua aplicação e observação da sua utilização. Os modelos de documentos também podem ser criados como *templates*, facilitando a sua edição. Sempre que possível, esta implementação deve estar incorporada às ferramentas já utilizadas pelos integrantes dos projetos.

Finalmente, a aplicação do modelo proposto em um projeto real de desenvolvimento de software agregaria muito à pesquisa realizada, contribuindo para verificar a sua aplicabilidade e resultados. Além disso, críticas e sugestões de pessoas que utilizaram o *framework* na prática contribuiriam para um refinamento da proposta.

REFERÊNCIAS

- ABBAS, N. **Software quality and governance in agile software development**. Thesis (Doctoral) - University of Southampton, School of Electronics and Computer Science. United Kingdom, 2009. 240p.
- ABDULLAH, E.; ABDELSATIR, E. T. B. Extreme programming applied in a large-scale distributed system. In: COMPUTING, ELECTRICAL AND ELECTRONICS ENGINEERING (ICEEEE), 2013 INTERNATIONAL CONFERENCE ON. **Proceedings...**, Khartoum, 2013, p. 442-446.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6633979&isnumber=6633894>.
 Acesso em: out. 2016.
- ABNT (Associação Brasileira de Normas e Técnicas). **NBR ISSO 9000-3 Parte 3: Diretrizes para a aplicação da NBR 19001 ao desenvolvimento, fornecimento e manutenção de software**. São Paulo, 1993.
- ABRAHAMSSON, P. ; WARSTA, J. ; SIPONEN, M. T.; RONKAINEN, J. New directions on agile methods: a comparative analysis. In: 25TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2003. **Proceedings...**, 2003, pp. 244-254. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1201204&isnumber=27042>.
 Acesso: out. 2014.
- AFFLECK, A.; KRISHNA, A. Supporting quantitative reasoning of non-functional requirements: a process-oriented approach. In: SOFTWARE AND SYSTEM PROCESS (ICSSP), 2012 INTERNATIONAL CONFERENCE ON. **Proceedings...**, p.88,92. Zuerich, 2-3 June 2012.
- AMARAL, D.C. et al. **Gerenciamento ágil de projetos - aplicação em produtos inovadores**. São Paulo: Saraiva, 2011. 240p.
- ANDERSON, D. J., Stretching agile to fit CMMI level 3 - the story of creating MSF for CMMI® process improvement at Microsoft corporation. In: AGILE CONFERENCE, 2005. **Proceedings...**, 2005, p. 193-201. DOI: 10.1109/ADC.2005.42.
- ANG, D.; LAI-HUAT LIM; CHAN, C.M.L. Collaborative requirements engineering: an overview and a proposed integrated model, System Sciences, 1998. In: HAWAII INTERNATIONAL CONFERENCE ON, 31. **Proceedings...**, v.5, p.355,364, USA, 6-9 Jan 1998.
- ATTARHA, M.; MODIRI, N. Focusing on the importance and the role of requirement engineering. In: INTERACTION SCIENCES (ICIS), INTERNATIONAL CONFERENCE ON, 4. **Proceedings...** South Korea, p.181,184, Aug. 2011.
- AURISICCHIO, M.; BRACEWELL, R. Capturing an integrated design information space with a diagram-based approach. **Journal of Engineering Design**, v. 24, n.6, p.397-428. United Kingdom, 2013.

BAILEY, J. Profile - People quality guru - Joseph Juran. In: **Engineering Management Journal**, v. 17, n. 4, p. 46-47, Aug.-Sept. 2007. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4402931&isnumber=4400842>

BASS, L. et al. Capturing and Using Rationale for a Software Architecture. In: Dutoit, A.H. et al. (eds.). **Rationale Management in Software Engineering**. Springer, 2006. p. 255-272.

BECK, K. **Extreme Programming Explained: Embrace Change**. Boston, MA: Addison-Wesley, 1999.

BECK, K.; ANDRES, C. **Extreme Programming Explained: Embrace Change**. 2nd ed. Boston, MA: Addison-Wesley, 2004.

BECK, K. et al. **Agile Manifesto, 2001**. Disponível em: <http://agilemanifesto.org/>. Acesso em: dez. 2014.

BEECHAM, S.; HALL, T.; RAINER, A. Software Process Improvement Problems in Twelve Software Companies: an Empirical Analysis. **Empirical Softw. Eng.** v.8, n.1, p. 7-42, March 2003.

BEECHAM, S.; HALL, T.; RAINER, A. **Defining a Requirements Process Improvement Model**. Software Quality, 13, 247-279, 2005.

BOEHM B. **A view of 20th and 21st century software engineering**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE '06), 28. **Proceedings...** New York, NY: ACM, 2006, p. 12-29.

BOEHM, B.; TURNER, R. Using risk to balance agile and plan-driven methods, **Computer**, v. 36, n. 6, p. 57-66, June 2003(a).

BOEHM, B.; TURNER, R. **Balancing Agility and Discipline: a Guide for the Perplexed**. Boston, MA: Addison-Wesley, 2003(b).

BOSCH, J. Software Architecture: the Next Step. In: SOFTWARE ARCHITECTURE: FIRST EUROPEAN WORKSHOP, 1. EWSA 2004. **Proceedings...** St Andrews, UK., p. 194-199, 2004.

BRACEWELL, R. H.; WALLACE, K. M.; MOSS, M.; KNOTT, D. Capturing design rationale. **Computer Aided Design**, v.41, n. 3, p.173-186, 2009.

BRITTON, C. **Object-oriented system development: a gentle introduction**. New York; London: McGraw-Hil, 2000.

BROOKS, F. P. No Silver Bullet Essence and Accidents of Software Engineering, **Computer**, v. 20, n. 4, p. 10-19, Apr. 1987.

BRUEGGE, B.; DUTOIT, A. H.; WOLF, T. **Sisyphus: Enabling informal collaboration in global software development**. In: GLOBAL SOFTWARE ENGINEERING, 2006. ICGSE '06. **International Conference on**, p.139,148, Oct. 2006.

BURGE, J. E. **Software Engineering Using design RATIONale**, PhD Dissertation, CS Dept., WPI, May 2005. Disponível em: <http://www.wpi.edu/Pubs/ETD/Available/etd-050205-085625/>. Acesso em: nov. 2005.

BURGE J. E.; BROWN D. C. **Design Rationale Types and Tools**, AI in Design Group, Computer Science Department, The Worcester Polytechnic Institute, 1998.

BURGE, J. E.; BROWN, D. C. Reasoning with design rationale. In: THE ARTIFICIAL INTELLIGENCE DESIGN CONFERENCE, **Proceedings...** 2000.

BURGE J. E.; BROWN D. C. **An Integrated Approach for Software Design Checking Using Rationale**, Design Computing and Cognition '04, J. Gero (Ed.), Kluwer Academic Publishers, Netherlands, 2004, p. 557-576.

BURGE, J. E.; BROWN, D. C. SEURAT: Integrated Rationale Management. SOFTWARE ENGINEERING ICSE '08. INTERNATIONAL CONFERENCE ON, 30. **Proceedings...** ACM/IEEE, p.835,838, 10-18 May 2008. doi: 10.1145/1368088.1368215.

BURGE J; CARROLL J.; MCCALL R.; MISTRIK I., Rationale and Requirements Engineering, in **Rationale-Based Software Engineering**. Springer, 2008, p. 139–153.

CANFORA, G.; CASAZZA, G.; DE LUCIA, A. A Design Rationale Based Environment for Cooperative Maintenance. **International Journal of Software Engineering and Knowledge Engineering**, v.10, n. 5, p. 627-645, 2000.

CHARETTE, R.; DWINNELL, L.; MCGARRY, J. **Understanding the Roots of Process Performance Failure**. CrossTalk (August 2004) p.18-22

CHARMAZ, K. **Constructing grounded theory**: A practical guide through qualitative analysis. London: Sage Publications. 2006.

CHEN A.; MCGINNIS B; ULLMAN D. G. Design History Knowledge Representation and its basic Computer Implementation. In: ASME INT. CONF. ON DESIGN THEORY AND METHODOLOGY, 2. **Proceedings...** J. Rinderle (Ed.), Chicago, IL, ASME, DE, v. 27, p. 175-184, 1990.

CHRISTEL, M.; KANG, K. **Issues in Requirements Elicitation** (CMU/SEI-92-TR-012). Software Engineering Institute, Carnegie Mellon University, 1992.
<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=12553>.

CHUNG, L.; NIXON, B. A.; YU, E. Dealing with Change: An Approach Using Non-Functional Requirements, **Requirements Engineering Journal**, v.1, n.4, p. 238--259, 1996.

CHUNG, L.; NIXON, B.; YU, E.; MYLOPOULOS, J. **Non-Functional Requirements in Software Engineering**, Netherlands: Kluwer Publishing, Dordrecht, 2000.

CMMI INSTITUTE, 2017. **Published Appraisal Results**. Disponível em: <https://sas.cmmiinstitute.com/pars/pars.aspx>. Acesso em: fev. 2017.

CMMI-DEV V1.2 – **CMMI para Desenvolvimento** - Versão 1.2, CMU/SEI-2006-TR-008 ESC-TR-2006-008. Melhoria de processos visando melhores produtos. Carnegie Mellon University, Software Engineering Institute (SEI), Agosto de 2006. Disponível em: http://www.sei.cmu.edu/library/assets/whitepapers/cmmi-dev_1-2_portuguese.pdf. Acesso em: dez. 2015.

CMMI-DEV V1.3 - **CMMI for Development** - Version 1.3, CMU/SEI-2010-TR-033. Carnegie Mellon University, Software Engineering Institute (SEI), November 2010. Disponível em <http://www.sei.cmu.edu/reports/10tr033.pdf>. Acesso em: dez. 2015.

CORBUCCHI, H. et al. **Genesis and Evolution of the Agile Movement in Brazil -- Perspective from Academia and Industry**. In: *Brazilian Symposium on Software Engineering*, 25. Sao Paulo, 2011, pp. 98-107. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6065151&isnumber=6065120>. Acesso em: abr. 2016.

COCKBURN, A. **Crystal Clear: A Human-Powered Methodology for Small Teams**. New York: Addison Wesley, 2004.

COCKBURN, A.; HIGHSMITH, J. Agile Software Development: The People Factor. **IEEE Computer**, v. 34, n. 11, p.131–133, 2001.

CONKLIN, J.; BEGEMAN, M. L. gIBIS: A hypertext tool for exploratory policy discussion. In: CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE. WORK (CSCW-88), **Proceedings...** ACM, Portland, Oregon, 1988, p. 140-152.

CONKLIN, J.; BURGESS-YAKEMOVIC, K. A Process-Oriented Approach to Design Rationale, In: T. Moran and J. Carroll, (ed). **Design Rationale Concepts, Techniques, and Use**, Mahwah, NJ: Lawrence Erlbaum Associates, 1995, p. 293-428.

CROSBY, P. B. **Quality is free**. New York: McGraw-Hill, 1979.

DEMING, W. E. **Out of the Crisis**. Cambridge, MA: Massachusetts Institute of Technology, Center for Advanced Engineering Study, 1986.

DIAZ, J. et al. Mapping CMMI level 2 to Scrum practices: an experience report. In: EuroSPI 2009. Communications in computer and information science.

DUTOIT, A. H.; MCCALL, R., MISTRIK I., PAECH, B. **Rationale Management in Software Engineering**, Springer-Verlag. New York: Secaucus, NJ, 2006.

DUTOIT, A. H.; PAECH, B. Supporting evolution: Using rationale in use case driven software development. In: DRIVEN SOFTWARE DEVELOPMENT. INTERNATIONAL WORKSHOP ON REQUIREMENTS ENGINEERING: FOUNDATION FOR SOFTWARE QUALITY, 6. **Proceedings...** 2000.

DUTOIT, A. H.; PAECH, B. Rationale Management in Software Engineering. In: **Handbook of Software Engineering and Knowledge Engineering**, p. 787–815. World Scientific Publishing, 2001.

ENG DASHET, T. ; MIDEKSO, D.; MACHADO, R. J. Integrated Framework of Agile and CMMI : An Alternative Path towards Product Focused SPI for Small Companies, **Lecture Notes in Software Engineering**, v. 4, n. 1, p. 1-6, 2016.

FALESSI, D. et al. **The value of design rationale information**, ACM Transactions on Software Engineering and Methodology (TOSEM), v.22 n.3, July 2013 doi:10.1145/2491509.2491515.

FARHAN, S.; TAUSEEF, H.; FAHIEM, M. A. Adding Agility to Architecture Tradeoff Analysis Method for Mapping on Crystal. In: SOFTWARE ENGINEERING, 2009. WCSE '09. WRI WORLD CONGRESS ON, XIAMEN, 2009, p. 121-125.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5319515&isnumber=5319487>

FEIGENBAUM, A. V. **Total quality control**. McGraw-Hill, 1991.

FERREIRA, D.; SILVA, A., An Enhanced Wiki for Requirements Engineering, In: SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS, SEAA '09, 35. *Euromicro Conference on*, p.87,94, 27-29 Aug. 2009.

FISCHER, G.; LEMKE, A.; MCCALL, R.; MORCH, A. Making argumentation serve design. In: Moran, T.; Carroll, J. ed. **Design Rationale Concepts, Techniques, and Use**. Lawrence Erlbaum Associates, 1995. 267-294.

FOGELSTRÖM, N. D.; GORSCHKE T.; SVAHNBERG, M.; OLSSON, P. The impact of agile principles on market-driven software product development. **J. Softw. Maint. E.** DOI=<http://dx.doi.org/10.1002/smr.v22:1>

FOWLER, M. **Using an Agile Software Process with Offshore Development**. 2003. Disponível em: <http://www.martinfowler.com/articles/agileOffshore.html>. Acesso em: jan. 2016.

FOWLER, M. **The New Methodology**. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em: jan. 2014.

FRITZSCHE, M.; KEIL, P. Agile Methods and CMMI: Compatibility or Conflict? **e-Informatica Software Engineering**, v. 1, n. 1, p. 9–26, 2007.

GALIN, D. **Software Quality Assurance: From Theory to Implementation**. Addison Wesley, 2003.

GANDOMANI, T. J.; HAZURAZULZALI, L. **Compatibility of Agile Software Development Methods and CMMI**. Indian Journal of Science and Technology, [S.l.], p. 5089-5094, aug. 2013. ISSN 0974 -5645. Disponível em: <http://www.indjst.org/index.php/indjst/article/view/36349>. Acesso em: fev. 2016.

GARCIA, A.; HOWARD, H.; STEFIK, M. Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design, **Tech. Report 82**, Stanford Univ. Center for Integrated Facility Engineering, Stanford, Calif.

GARZÁS, J.; PAULK, M. C. A case study of software process improvement with CMMI-DEV and Scrum in Spanish companies. **Journal of Software: Evolution and Process**, v.25, p. 1325-1333, 2013.

GILLIES, A. C. **Software Quality: Theory and Management**. London: Chapman & Hall, 2002.

GLAZER, H. et al. **CMMI or Agile: Why Not Embrace Both!**. CMU/SEI-2008-TN-003. Software Engineering Institute, Carnegie Mellon University. 2008. Disponível em: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8533>. Acesso em: mar. 2016.

GRUBER, T. R. Model-based Explanation of Design Rationale. In: AAAI-90 EXPLANATION WORKSHOP, **Proceedings...** Boston, July 30, 1990.

GRUBER, T. R.; RUSSEL, D. M. Design Knowledge and Design Rationale: A framework for representation, capture, and use. **Technical Report KSL 90-45**, Knowledge Systems Laboratory, Stanford, California, 1991. 40p.

HIGHSMITH, J. Messy, Exciting, and Anxiety Ridden: Adaptive Software Development. **American Programmer**, v.10, n.1, 1997.

HIGHSMITH, J. **Adaptive Software Development: A Collaborative Approach to Managing Complex Systems**. New York, NY: Dorset House Publishing, 1999.

HIGHSMITH, J. **Agile Project Management: Creating Innovative Products**. 2nd ed. Addison-Wesley Professional, 2009.

HORNER, J. ; ATTWOOD, M.E. (2006). Effective design rationale: understanding the barriers. **Rationale Management in Software Engineering** (A. Dutoit , R. McCall , I. Mistrík , & B. Paech , Eds.), pp. 73–90. Heidelberg: Springer–Verlag.

HU, X. et al. A survey on design rationale: representation, capture and retrieval. **Engineering with Computers: An Int'l Journal for Simulation-Based Engineering**, v.16, p. 209–235, 2000.

ISHIKAWA, K. **What is total quality control?** The Japanese way. Prentice-Hall, 1985.

ISO/IEC TR 15504. **Software Process Assessment**. The International Organization for Standardization and The International Electrotechnical Commission, 2008.

JACOBSON, I., NG, P-W. **Aspect-Oriented Software Development with Use Cases** (Addison-Wesley Object Technology Series), Addison-Wesley, 2004.

JARCZYK, A.; LOFFLER, P.; SHIPMAN, F. Design Rationale for Software Engineering: A Survey. In: ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 25. **Proceedings...** (8-10 January, 1992), 1992.

JIN H.; ZENG, F. **Research on the definition and model of software testing quality**, *Reliability, Maintainability and Safety (ICRMS), 2011 9th International Conference on*, Guiyang, 2011, pp. 639-644.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5979352&isnumber=5979226>.

JIRA SOFTWARE. Atlassian. Ferramenta para planejar, rastrear e lançar software. Disponível em <https://br.atlassian.com/software/jira>. Acesso em: mar. 2017.

JURAN, J. M. **Quality Control Handbook**, 4th ed. New York: McGraw Hill, 1988.

KAJKO-MATTSSON, M. et al., **Long-term Life Cycle Impact of Agile Methodologies**. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 22. Philadelphia, PA, USA, 2006, p. 422-425. Disponível em:
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4021369&isnumber=4021305>

KAWAMOTO, S.; ALMEIDA JR, J.R. A Representation Model of Design Rationale for the Design of ERP Systems. **INFOCOMP Journal of Computer Science**, v. 8, n. 3, p. 88-100, 2009.

KAWAMOTO, S.; ALMEIDA JR, J. R. Scrum-DR: An Extension of the Scrum Framework Adherent to the Capability Maturity Model Using Design Rationale Techniques. In: IEEE CHILECON 2017, Pucón. **Proceedings...** Pucón- Chile, 2017.

KHALANE, T.; TANNER, M. **Software quality assurance in Scrum: The need for concrete guidance on SQA strategies in meeting user expectations**, 2013 International Conference on Adaptive Science and Technology, Pretoria, 2013, pp. 1-6. Disponível em:
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6707499&isnumber=6707481>

KHATTER, K.; KALIA, A. **Impact of Non-functional Requirements on Requirements Evolution**. In: EMERGING TRENDS IN ENGINEERING AND TECHNOLOGY (ICETET), INTERNATIONAL CONFERENCE ON, 6. p.61,68, 16-18 Dec. 2013. doi: 10.1109/ICETET.2013.15

KOTONYA, G.; SOMMERVILLE, I. **Requirements Engineering: Processes and Techniques**. London: J. Wiley, 1998, ISBN 0471972088.

KRUCHTEN, P. Agility with the RUP. **Cutter IT Journal**. v.14, n. 12, p.27-33, 2001.

KUNZ W.; RITTEL H.J., 1970, **Issues as Elements of Information Systems**, Working Paper No 131, University of California, Berkeley. Reprinted in 1979. Disponível em:
<http://www.cc.gatech.edu/~ellendo/rittel/rittel-issues.pdf>. Acesso em: mar. 2014.

LAND, S. K.; WALZ J. W. **Practical Support for CMMI - SW Software Project Documentation: Using IEEE Software Engineering Standards (Practitioners)**. Wiley-IEEE Computer Society Pr. 2006.

LARMAN, C. **Agile and Iterative Development: A Manager's Guide**. Pearson Education, 2003.

LARMAN C.; BASILI, V. R. Iterative and Incremental Development: A Brief History. **Computer**, v.36, n.6, p. 47-56, (June 2003)
DOI=<http://dx.doi.org/10.1109/MC.2003.1204375>

LAWRENCE B; WIEGERS, K.; EBERT, C. The top risk of requirements engineering, **Software, IEEE** , v.18, n.6, p.62,63, Nov/Dec 2001. doi: 10.1109/52.965804.

LEE, G.; DELONE, W.; ESPINOSA, J. A. Ambidextrous coping strategies in globally distributed software development projects. **Commun. ACM**, v.49, n.10 (October 2006), p.35-40. DOI=<http://dx.doi.org/10.1145/1164394.1164417>

LEE, J. Design rationale Systems: Understanding the Issues. **IEEE Expert**, v. 12, n. 3, p78-85. May/Jun 1997.

LEE J.; LAI K-Y. What's in design rationale? **Hum.-Comput. Interact**, v.6, n. 3 (September 1991), p.251-280.

LEESON, P. **Why CMMI does not work?**, presented at the SEPG Europe, Prague, Czech Republic, 2009.

LIANG P.; AVGERIOU, P.; HE K. Rationale management challenges in requirements engineering. In: MANAGING REQUIREMENTS KNOWLEDGE (MARK), INTERNATIONAL WORKSHOP ON, 30., **Proceedings...** p.16,21, 27-27 Sept. 2010.

LINA, Z.; DAN, S. **Research on Combining Scrum with CMMI in Small and Medium Organizations**, 2012 International Conference on Computer Science and Electronics Engineering, Hangzhou, 2012, p. 554-557. Disponível em <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6187907&isnumber=6187799>. Acesso em mar. 2013.

LIU L.; LI T.; PENG F. Why Requirements Engineering Fails: A Survey Report from China, IN: REQUIREMENTS ENGINEERING CONFERENCE (RE), 18.IEEE INTERNATIONAL, p.317,322, Sept. 27 2010-Oct. 1 2010.

LUKASIEWICZ, K.; MILER, J. **Improving agility and discipline of software development with the Scrum and CMMI**, IET Software, v. 6, n. 5, p. 416-422, October 2012.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6334806&isnumber=6334803>

LYCETT, M. et al. Migrating Agile Methods to Standardized Development Practice. **Computer**, v. 36, n. 6, p. 79-85 (June 2003),
DOI=<http://dx.doi.org/10.1109/MC.2003.1204379>

MAALEJ, W.; THURIMELLA, A.K. Towards a Research Agenda for Recommendation Systems in Requirements Engineering. In: MANAGING REQUIREMENTS KNOWLEDGE (MARK), INTERNATIONAL WORKSHOP ON, 2. p.32,39, 1-1 Sept. 2009 doi: 10.1109/MARK.2009.12

MACLEAN, A.; YOUNG, R. M.; BELLOTTI, V.M.E; MORAN, T.P. Questions, Options and Criteria: Elements of Design Space Analysis. In: Moran; Carroll (ed.). **Design Rationale: Concepts, techniques, and Use**. Lawrence Erlbaum, 1996.

MARÇAL, A. S. **Scrummi**: Um processo de gestão ágil baseado no SCRUM e aderente ao CMMI. Dissertação (Mestrado) - UNIFOR – Fortaleza, Brasil, 2009.

MARÇAL, A. S. C.; SOARES, F. S. F.; BELCHIOR, A. D. **Mapping CMMI Project Management Process Areas to SCRUM Practices**. In: IEEE SOFTWARE ENGINEERING WORKSHOP (SEW '07), 31. **Proceedings...** IEEE Computer Society, Washington, DC, USA, p.13-22.

MARGARIDO, I. L. ; VIDAL, R. M.; VIEIRA, M. Lessons Learnt in the Implementation of CMMI® Maturity Level 5. In: QUALITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY (QUATIC), INTERNATIONAL CONFERENCE ON THE, 8. **Proceedings...** Lisbon, 2012, p. 47-56.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6511781&isnumber=6511765>

MESO, P.; JAIN, R. Agile software development: adaptive systems principles and best practices. **Information Systems Management**, v. 23, p. 19–30, 2006.

MISRA S. C.; KUMAR V.; KUMAR U. Identifying some important success factors in adopting agile software development practices. **J. Syst. Softw.** v.82, n. 11, p. 1869-1890, 2009. DOI=<http://dx.doi.org/10.1016/j.jss.2009.05.052>

MNKANDLA, E. About software engineering frameworks and methodologies, In: **AFRICON AFRICON '09**. p.1,5, 23-25 Sept. 2009.

MYERS, K. L.; ZUMEL N.B.; GARCIA P. Automated Capture of Rationale for the Detailed Design Process, In: INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE (IAAI-99), **Proceedings...** AAAI Press: Menlo Park, CA, USA.1999.

OMRAN, A. **AGILE CMMI from SMEs perspective**. *Information and Communication Technologies: From Theory to Applications*, 2008. *ICTTA 2008. 3rd International Conference on*, Damascus, 2008, pp. 1-8. Disponível em:
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4530352&isnumber=4529902>.

OWENS, D. M.; KHAZANCHI, D. Software Quality Assurance. In: I.S. Reference (Ed.), **Handbook of Research on Technology Project Management, Planning, and Operations**, p. 245–263, 2009.

PACHECO, R. C. D. S.; TOSTA, K. C. B. T.; FREIRE, P. D. S. Interdisciplinaridade vista como um processo complexo de construção do conhecimento: uma análise do Programa de Pós-graduação do EGC / UFSC. **R.B.P. G**, v. 7, n. 12, p. 136–159, 2010.

PAIVA, D. M. B. **Um processo envolvendo elementos de *design rationale* para projetos de pesquisa em software**. 2007 254p. Tese (Doutorado) - Instituto de Ciências Matemáticas e de Computação – ICMC-USP, 2007.

PALMER, S. R.; FELSING, M. **A Practical Guide to Feature-Driven Development**. Pearson Education, 2001.

PARNAS, D.; CLEMENTS, P. A Rationale Design Process: How and Why to Fake it. **IEEE Transactions of Software Engineering**, v. 12, n.2, p. 251-257, 1986.

PATEL C, RAMACHANDRAN M. Agile maturity model (AMM): A Software Process Improvement framework for agile software development practices. **International Journal of Software Engineering, IJSE**. v.2, n. 1, p. 3–28, 2009.

PINO, F. J.; GARCÍA, F.; PIATTINI, M. **Software process improvement in small and medium software enterprises: a systematic review**. *Software Quality Journal* 16, 2 (June 2008), 237-261. DOI=<http://dx.doi.org/10.1007/s11219-007-9038-z>.

POTTS, C.; BRUNS, G. Recording the Reasons for Design Decisions. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Kyoto, Japão. IEEE CS Press, 1988, p. 418-427.

POTTS, C.; TAKAHASHI, K., An active hypertext model for system requirements. In: SOFTWARE SPECIFICATION AND DESIGN. INTERNATIONAL WORKSHOP ON, 7. **Proceedings...** p.62,68, 6-7 Dec 1993.

POTTS, C.; TAKAHASHI, K.; ANTON, A.I. **Inquiry-based requirements analysis**, *Software, IEEE* , v.11, n.2, p.21,32, March 1994. doi: 10.1109/52.268952.

PRESSMAN R. S.; **Engenharia de Software**. 6. ed. Sao Paulo: McGraw-Hill, 2006.

PRESSMAN, R. S. **Software Engineering – A Practitioner’s Approach**. 5. ed. São Paulo: McGraw-Hill, 2010.

RAATIKAINEN, M.; MANNISTO, T.; TOMMILA, T.; VALKONEN, J. Challenges of requirements engineering - A case study in nuclear energy domain. In: REQUIREMENTS ENGINEERING CONFERENCE (RE), IEEE INTERNATIONAL, 11. **Proceedings...** p.253,258, Aug. 29 2011-Sept. 2 2011. doi: 10.1109/RE.2011.6051629

RAMESH, B.; DHAR, V. Supporting systems development by capturing deliberations during requirements engineering, **Software Engineering, IEEE Transactions on** , v.18, n.6, p.498,510, Jun 1992. doi: 10.1109/32.142872

RESS, A. P. P. M. **Um estudo sobre a integração CMMI e desenvolvimento ágil**. Dissertação (Mestrado em Engenharia de Produção) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2013. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/3/3136/tde-14102014-151014/>>. Acesso em: ago. 2015.

ROBINSON, W.N.; PAWLOWSKI, S. Surfacing root requirements interactions from inquiry cycle requirements documents. In: REQUIREMENTS ENGINEERING, 1998. PROCEEDINGS. INTERNATIONAL CONFERENCE ON, 3. **Proceedings...** p.82,89, 6-10 1998. doi: 10.1109/ICRE.1998.667812

ROCHA, A.R.C.; MALDONADO, J. C.; WEBER, K.C. **Qualidade de Software: Teoria e Prática**. São Paulo: Prentice-Hall, 2001.

SADIQ, M.; HASSAN, T. **An extended adaptive software development process model**, *Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014 *International Conference on*, Ghaziabad, 2014, pp. 552-558. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6781341&isnumber=6781240>

SALVATORE, F.J.; ALAMEDA, T. Daily challenges in requirements engineering. In: REQUIREMENTS ENGINEERING CONFERENCE. IEEE INTERNATIONAL, 11. **Proceedings...** p.297, 8-12 Sept. 2003 DOI: 10.1109/ICRE.2003.1232769.

SANGWAN, R. S.; MASTICOLA, S. P. Model-Driven Rapid Application Development: A Framework for Agile Development in Outsourced Environments. **Technical report**, Siemens Corporate Research, 2004.

SAUER, T. Using Design Rationales for Agile Documentation. In: IEEE INTERNATIONAL WORKSHOPS ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES 2003 (WETICE'03). **Proceedings ...**

SCHWABER, K. **Agile Project Management with Scrum**. Redmond, WA: Microsoft Press, 2004.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. New Jersey: Prentice Hall PTR, 2001.

SCHWABER K.; SUTHERLAND, J. **Guia do Scrum. Um guia definitivo para o Scrum: As Regras do Jogo**. Julho de 2016. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Portuguese-Brazilian.pdf>. Acesso em: jan. 2017.

SHEARD, S. A. Evolution of the Frameworks Quagmire, **IEEE Computer**, July 2001, p. 96-98.

SHEWART, W. A. **Economic control of quality of manufactured product**. ASQ Quality Press, 1980.

SHUM, S. B.; HAMMOND, N. Argumentation-Based Design Rationale: What Use at What Cost? **International Journal of Human-Computer Studies**, v. 40, n.4, p. 603-652, 1994.

SILVA, F. B. et al. **Proposta e avaliação de um procedimento de planejamento de tempo combinado ágil e tradicional**. Universidade de São Paulo, 2015.

SILVA, F. S. **Uso de Representação de Conhecimento para Documentação de Metodologias Ágeis**. Dissertação de Mestrado, PUCRS, 2009, 152p.

SILVA, F. S. et al. A Reference Model for Agile Quality Assurance: Combining Agile Methodologies and Maturity Models. In: QUALITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY (QUATIC), INTERNATIONAL CONFERENCE ON THE, 9.Guimaraes, 2014, pp. 139-144.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6984105&isnumber=6984078>

SILVA, F. S. et al. Using CMMI together with agile software development: A systematic review. **Information and Software Technology**, v. 58, p. 20-43, February 2015, ISSN 0950-5849.

SOARES, F. S. F.; MEIRA, S. R. L. **An agile strategy for implementing CMMI project management practices in software organizations**, In: Iberian Conference on Information Systems and Technologies (CISTI), 10. Aveiro, 2015, p. 1-4. doi: 10.1109/CISTI.2015.7170402
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7170402&isnumber=7170350>.

SOLEMON, B.; SAHIBUDDIN, S.; GHANI A. A. A. Requirements engineering problems in 63 software companies in Malaysia. In: INFORMATION TECHNOLOGY, 2008. ITSIM 2008. INTERNATIONAL SYMPOSIUM ON , v.4, p.1,6, 26-28 Aug. 2008.

SOLEMON, B.; SHAHIBUDDIN, S.; GHANI, A. Re-defining the Requirements Engineering Process Improvement Model, In: SOFTWARE ENGINEERING CONFERENCE, 2009. APSEC '09. ASIA-PACIFIC, **Proceedings...**, p.87,92, 1-3 Dec. 2009.

SOMMERVILLE, I. Integrated Requirements Engineering: a Tutorial. **IEEE software**, p. 16–23, 2005.

SOMMERVILLE, I. **Engenharia de Software**, 8. ed. Pearson Addison-Wesley, 2007.

SOUZA, C. R. B.; SANTOS, D. B.; WAINER, J.; DIAS, K. L. A model tool for semi-automatic recording of design rationale in software diagrams. In STRING PROCESSING AND INFORMATION RETRIEVAL SYMPOSIUM. **Proceedings...** p. 306–313, 1998.

STANDISH GROUP. **The Chaos Report**. 1995. Disponível em <http://www.standishgroup.com>. Acesso em: dez. 2013.

STUMPF S.C. **Argumentation-based design rationale - the sharpest tools in the box**. Research Note RN/98/103, Computer Science Department, University College London, University of London, U.K. 1997.

SUTCLIFFE, A. **Requirements rationales: integrating approaches to requirement analysis**. In: CONFERENCE ON DESIGNING INTERACTIVE SYSTEMS: PROCESSES, PRACTICES, METHODS, & TECHNIQUES (DIS '95), 1. **Proceedings...** Olson, G.M. and Schuon, S. (eds.). ACM, New York, NY, p.33-42, 1995.

SUTCLIFFE, A.G.; RYAN, M., Experience with SCRAM, a SScenario Requirements Analysis Method. In: REQUIREMENTS ENGINEERING, 1998. INTERNATIONAL CONFERENCE ON, 3., **Proceedings...** p.164,171, 1998. doi: 10.1109/ICRE.1998.667822.

SUTHERLAND, J.; JAKOBSEN, C. R.; JOHNSON, K. **Scrum and CMMI Level 5: The Magic Potion for Code Warriors**. In: AGILE 2007 (AGILE 2007), Washington, DC, 2007, p. 272-278. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4293608&isnumber=4293563>

SYDENHAM, P. H. **System Approach to Engineering Design**. Norwood: Artech House Publishers, 2003, 333p. ISBN 1580534791.

TAKAHASHI, K. et al. Hypermedia support for collaboration in requirements analysis. In: REQUIREMENTS ENGINEERING, 1996, INTERNATIONAL CONFERENCE ON, 2. **Proceedings...**, p.31,40, 15-18 Apr 1996.

TAKARA, A.; BETTIN, A. X.; TOLEDO, C. M. T. **Problems and Pitfalls in a CMMI level 3 to level 4 Migration Process.** *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, Lisbon, 2007, pp. 91-99.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4335237&isnumber=4335220>

TELLES, V. M. **Extreme Programming:** Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec, 2004. 320 p.

THAYER R. H.; DORFMAN M. **System and Software Requirements Engineering.** Los Alamitos, CA: IEEE Computer Society Press, 1990.

THURIMELLA, A.K.; BRUEGGE, B. Evolution in Product Line Requirements Engineering: A Rationale Management Approach. In: REQUIREMENTS ENGINEERING CONFERENCE, RE '07. IEEE INTERNATIONAL, 15. **Proceedings...**, p.254,257, 15-19 Oct. 2007.

TMMI FOUNDATION. **Test Maturity Model Integration.** Release 1.0. Ireland, 2012. 219 p.

TURK, D., FRANCE, R., RUMPE, B. Limitations of agile software processes. In: INTERNATIONAL CONFERENCE ON EXTREME PROGRAMMING AND AGILE PROCESSES IN SOFTWARE ENGINEERING, 3. **Proceedings...** Sardinia, Italy.2002.

TURNER, R. 2002. Agile Development: Good Process or Bad Attitude? In: PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON PRODUCT FOCUSED SOFTWARE PROCESS IMPROVEMENT (PROFES '02), 4. **Proceedings...** Springer-Verlag, London, UK, UK, 134-144.

TURNER, R.; JAIN, A. **Agile Meets CMMI: Culture Clash or Common Cause?** In: PROCEEDINGS OF THE SECOND XP UNIVERSE AND FIRST AGILE UNIVERSE CONFERENCE ON EXTREME PROGRAMMING AND AGILE METHODS - XP/AGILE UNIVERSE 2002, **Proceedings ...** Springer-Verlag, London, UK, UK, 153-165.

ULLAH, S.; IQBAL, M.; KHAN, A.M. A survey on issues in non-functional requirements elicitation. In: COMPUTER NETWORKS AND INFORMATION TECHNOLOGY (ICCNIT), 2011 INTERNATIONAL CONFERENCE ON, **Proceedings...** p.333,340, 11-13 July 2011. doi: 10.1109/ICCNIT.2011.6020890.

VIANA, D., CONTE, T., MARCZAK, S. **Knowledge Creation and Loss within a Software Organization: An Exploratory Case Study.** In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS 2015), 48., **Proceedings...** p. 3980-3989, Hawaii, 2015.

YIN, R. K. The Case study crisis: some answers. In: **Administrative Science Quarterly**, Cornell University, v.26, mar. 1981.

WASSON, C. **System Analysis, design, and development: concepts, principles, and practices**. New Jersey: John Wiley & Sons, 2006, 818p. ISBN: 9780471393337.

ZANATTA, A. L. **XScrum: uma proposta de extensão de um Método Ágil para Gerência e Desenvolvimento de Requisitos visando adequação ao CMMI**. 2004. 180 f. Dissertação (Mestrado) - Universidade Federal De Santa Catarina, Florianópolis, 2004.

ZHONG, S.; LIPING, C. ; TIAN-EN, C. Agile planning and development methods. In: COMPUTER RESEARCH AND DEVELOPMENT (ICCRD), INTERNATIONAL CONFERENCE ON, 3. Shanghai, 2011, pp. 488-491, 2011.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5764064&isnumber=5763958>

APÊNDICE A – Questionário utilizado na entrevista

1. Qual a sua área de atuação na empresa?
2. A empresa segue uma metodologia ágil específica?
3. É comum a todos os projetos?
4. Empresa tem certificação CMMI ou outra de qualidade? Clientes cobram a existência de certificação de qualidade?
5. Quais as principais dificuldades dos projetos, principalmente relacionados à metodologia ágil?
6. Quanto de documentação é realizada? Cite alguns exemplos.
7. Há captura de *Design Rationale*?
8. Há aprendizado entre um projeto e outro? Quais as práticas realizadas para a gestão do conhecimento?

APÊNDICE B – Plano de Projeto do SGH utilizando Scrum-DR

Informações gerais do projeto SGH:

Campos	Descrição
Nome	Sistema de Gerenciamento de Hotel
Descrição	Sistema de reservas de quartos no hotel. Será utilizado por clientes e funcionários. As funcionalidades principais são reserva de quarto, <i>check in</i> e <i>check out</i> .
Data Inicial	15/01/2016
Data de Término	30/03/2016
Objetivos	Facilitar e automatizar a reserva de quartos
Restrições	A data final do projeto não pode ultrapassar 30/04/2016.
Riscos	
Reuniões	Reunião de planejamento da Sprint: Primeira e terceira segunda-feira do mês, 09:00, Sala 10. Reunião Diária: Segunda a sexta, 14:00, sala 5.
Medições	<ul style="list-style-type: none"> Número de bugs encontrado em uma Sprint não pode ultrapassar 10 Número de bugs abertos não pode ultrapassar 10.
Product Owner	Luísa
Scrum Master	Matheus
Time de Desenvolvimento	Fabiano, Cecília e Luiz
Cliente	Julia e Wal
Stakeholders	Luciane e Reginaldo

Descrição da reserva de quarto:

Funcionalidade	Reservar quarto (RES001)
Descrição resumida	Reserva de um quarto pelo cliente
Fluxo básico	<p>9- Cliente seleciona a opção de reserva de um quarto</p> <p>10- O sistema apresenta os tipos de quartos do hotel e seus valores</p> <p>11- Cliente verifica os valores dos quartos</p> <p>12- Cliente realiza a reserva para o quarto escolhido</p> <p>13- O sistema subtrai do banco de dados o número de quartos disponíveis para reserva do tipo selecionado</p> <p>14- O sistema cria uma nova reserva com os detalhes fornecidos</p> <p>15- O sistema apresenta o número de confirmação da reserva e as instruções de check-in</p> <p>16- O evento é finalizado</p>
Fluxo alternativo	<p>5.2- Submissão duplicada: se no passo 5 do fluxo básico ocorrer uma reserva idêntica no sistema (mesmo nome, e-mail, e datas de entrada e saída), o sistema mostra a reserva existente e pergunta ao cliente se ele quer proceder com a nova reserva.</p> <p>5.1.1- Se o cliente quiser continuar, o sistema executa a reserva</p> <p>5.1.2- Se o cliente informar que a nova reserva é duplicada, a reserva é cancelada</p>
Subfluxos	<p>3.1- Checar custo do quarto</p> <p>3.1.1- O cliente seleciona o tipo de quarto desejado e indica o seu período de permanência</p> <p>3.1.2- O sistema calcula o custo para o período especificado</p>
Pré-condições	O cliente fez <i>log in</i> no sistema
Pós-condições	Se a reserva finalizar com sucesso, um novo registro de reserva é criado e é decrementado o número de quartos disponíveis do mesmo tipo e para as datas especificadas. Se a reserva não for finalizada com sucesso, não há mudança na base de dados.
Requisitos especiais	O sistema deve ser capaz de manipular cinco reservas concorrentes. Cada reserva não deve demorar mais que 20 segundos.

APÊNDICE C – Backlog do Produto do SGH utilizando Scrum-DR

Versão final do Backlog do Produto:

Sprint	Nome	Descrição	Tipo	Prioridade	Grupo
1	Cadastro de quartos	Vide CAD001	F	1	Cadastro
1	Cadastro de Cliente	Vide CAD003	F	1	Cadastro
2	Reserva de quarto	Vide RES001	F	2	Reserva
	Manipulação de Lista de Espera	Se não tiver quarto disponível, cliente vai para a lista de espera	F	3	Reserva
3	<i>Check in</i>	Cliente é associado a um quarto e então a sua reserva é processada. Ao mesmo tempo, cria-se uma conta inicial para o cliente.	F	2	Check in/out
3	<i>Check out</i>	O cliente realiza o pagamento da conta e então o cliente é removido do quarto.	F	2	Check in/out
3	Programa de Fidelidade	Vide FID001	F	4	Reserva
2	<i>Log in</i> no sistema	Vide LOG001	F	1	Início
1	Estabelecer taxas	Gerente do Hotel define custos de cada quarto	F	1	Cadastro
1	Internet	Acessibilidade do sistema via Internet.	NF	1	Acessibilidade
1	BD Relacional	O sistema deve armazenar todos os registros em um banco de dados relacional.	NF	1	Base de Dados
2	Transações registradas	Para fins de auditoria, todas as transações do sistema devem ser registradas.	NF	4	Segurança
2	Pessoas autorizadas	Somente pessoas autorizadas podem desempenhar as suas funções no sistema	NF	3	Segurança
3	Preferências do cliente	O sistema deve rastrear as preferências do cliente e utilizá-las como <i>default</i> .	NF	5	Usabilidade
4	Recuperação de dados	Todas as recuperações de dados não devem exceder 2 segundos.	NF	2	Desempenho
4	Reservas concorrentes	O sistema deve manipular até cinco reservas concorrentes.	NF	2	Desempenho
4	Tempo reserva	Cada reserva não deve exceder 20 segundos.	NF	2	Desempenho

APÊNDICE D – DOC-DR do Projeto do SGH utilizando Scrum-DR

Plataforma de Desenvolvimento:

Log in no sistema:

APÊNDICE E – Backlogs da Sprint do Projeto do SGH utilizando Scrum-DR

Sprint 1:

Funcionalidades que serão implementadas:

Sprint 1				
Nome	Descrição	Tipo	Prioridade	Grupo
Cadastro de quartos	Vide CAD001	F	1	Cadastro
Cadastro de Cliente	Vide CAD003	F	1	Cadastro
Estabelecer taxas	Gerente do Hotel define custos de casa quarto	F	1	Cadastro
Internet	Acessibilidade do sistema via Internet.	NF	1	Acessibilidade
BD Relacional	O sistema deve armazenar todos os registros em um banco de dados relacional.	NF	1	Base de Dados

Implementação das funcionalidades principais:

	Quarto	Reserva	Pagamento
Reserva de quarto	checarDisponibilidade()	criar()	
Check in do Cliente	associarCliente()	consumir()	criarConta()
Check out do Cliente	removerCliente()		pagarConta()

Sprint 2:

Funcionalidades que serão implementadas:

Sprint 2				
Nome	Descrição	Tipo	Prioridade	Grupo
Reserva de quarto	Vide RES001	F	2	Reserva
<i>Log in</i> no sistema	Vide LOG001	F	1	Início
Transações registradas	Para fins de auditoria, todas as transações do sistema devem ser registradas.	NF	4	Segurança
Pessoas autorizadas	Somente pessoas autorizadas podem desempenhar as suas funções no sistema	NF	3	Segurança

Sprint 3:

Funcionalidades que serão implementadas:

Sprint 3				
Nome	Descrição	Tipo	Prioridade	Grupo
<i>Check in</i>	Cliente é associado a um quarto e então a sua reserva é processada. Ao mesmo tempo, cria-se uma conta inicial para o cliente.	F	2	Check in/out
<i>Check out</i>	O cliente realiza o pagamento da conta e então o cliente é removido do quarto.	F	2	Check in/out
Programa de Fidelidade	Vide FID001	F	4	Reserva
Preferências do cliente	O sistema deve rastrear as preferências do cliente e utilizá-las como <i>default</i> .	NF	5	Usabilidade

Sprint 4:

Funcionalidades que serão implementadas:

Sprint 4				
Nome	Descrição	Tipo	Prioridade	Grupo
Recuperação de dados	Todas as recuperações de dados não devem exceder 2 segundos.	NF	2	Desempenho
Reservas concorrentes	O sistema deve manipular até cinco reservas concorrentes.	NF	2	Desempenho
Tempo reserva	Cada reserva não deve exceder 20 segundos.	NF	2	Desempenho

ANEXO – Metas e práticas específicas das áreas de processo do CMMI-DEV 1.3

Categoria Gerenciamento de Processo:

Área de Processo	Metas Específicas	Práticas Específicas
Definição dos Processos da Organização	1 Estabelecer Ativos de Processo da Organização	1.1 Estabelecer Processos-Padrão
		1.2 Estabelecer Descrições de Modelos de Ciclo de Vida
		1.3 Estabelecer Critérios e Diretrizes para Adaptação
		1.4 Estabelecer o Repositório de Medições da Organização
		1.5 Estabelecer a Biblioteca de Ativos de Processo da Organização
		1.6 Estabelecer Padrões de Ambiente de Trabalho
		1.7 Estabelecer Regras e Diretrizes para as Equipes
Foco nos Processos da Organização	1 Determinar Oportunidades de Melhoria de Processo	1.1 Estabelecer Necessidades de Processo da Organização
		1.2 Avaliar os Processos da Organização
		1.3 Identificar Melhorias para os Processos da Organização
	2 Planejar e Implementar Melhorias de Processo	2.1 Estabelecer Planos de Ação de Processo
		2.2 Implementar Planos de Ação de Processo
	3 Implantar os Ativos de Processo da Organização e Incorporar Lições Aprendidas	3.1 Implantar Ativos de Processo da Organização
		3.2 Implantar Processos-padrão
Gestão do Desempenho da Organização	1 Gerenciar Desempenho do Negócio	3.3 Monitorar Implementação
		3.4 Incorporar Experiências Relacionadas a Processos nos Ativos de Processo da Organização
	2 Selecionar Melhorias	1.1 Manter os Objetivos do Negócio
		1.2 Analisar os Dados de Desempenho de Processos
		1.3 Identificar Áreas Potenciais de Melhoria
		2.1 Extrair as Melhorias Sugeridas
		2.2 Analisar as Melhorias Sugeridas
		2.3 Validar as Melhorias
		2.4 Selecionar e Implementar as Melhorias
	3 Implantar Melhorias	3.1 Planejar a Implantação
		3.2 Gerenciar a Implantação
		3.3 Avaliar os Efeitos da Melhoria

Continua

Continuação

Área de Processo	Metas Específicas	Práticas Específicas
Desempenho dos Processos da Organização	1 Estabelecer <i>Baselines</i> e Modelos de Desempenho	1.1 Estabelecer Objetivos para Qualidade e para Desempenho de Processo
		1.2 Selecionar Processos
		1.3 Estabelecer Medidas de Desempenho de Processo
		1.4 Analisar Desempenho de Processo e Estabelecer <i>Baselines</i> de Desempenho de Processo
		1.5 Estabelecer Modelos de Desempenho de Processo
Treinamento na Organização	1 Estabelecer uma Capacidade de Treinamento na Organização	1.1 Estabelecer Necessidades Estratégicas de Treinamento
		1.2 Identificar as Necessidades de Treinamento sob Responsabilidade da Organização
		1.3 Estabelecer um Plano Tático de Treinamento na Organização
		1.4 Estabelecer Capacidade de Treinamento
	2 Proporcionar Treinamento Necessário	2.1 Fornecer Treinamentos
		2.2 Estabelecer Registros de Treinamento
		2.3 Avaliar a Eficácia dos Treinamentos

Fonte: CMMI-DEV V1.3 (2010)

Categoria Gerenciamento de Projeto:

Área de Processo	Metas Específicas	Práticas Específicas
Gestão Integrada de Projeto	1 Utilizar o Processo Definido para o Projeto	1.1 Estabelecer o Processo Definido para o Projeto
		1.2 Utilizar os Ativos de Processo da Organização para Planejar as Atividades do Projeto
		1.3 Estabelecer o Ambiente de Trabalho do Projeto
		1.4 Integrar Planos
		1.5 Gerenciar o Projeto Utilizando Planos Integrados
		1.6 Estabelecer Equipes
		1.7 Contribuir para os Ativos de Processo da Organização
	2 Coordenar e Colaborar com as Partes Interessadas Relevantes	2.1 Gerenciar o Envolvimento das Partes Interessadas
		2.2 Gerenciar Dependências
		2.3 Solucionar Questões Críticas de Coordenação
Monitoramento e Controle de Projeto	1 Monitorar o Projeto em Relação ao Plano	1.1 Monitorar os Parâmetros de Planejamento do Projeto
		1.2 Monitorar Compromissos
		1.3 Monitorar Riscos do Projeto
		1.4 Monitorar a Gestão de Dados
		1.5 Monitorar o Envolvimento das Partes Interessadas
		1.6 Conduzir Revisões de Progresso
		1.7 Conduzir Revisões de Marco
	2 Gerenciar Ações Corretivas até a sua Conclusão	2.1 Analisar Questões Críticas
		2.2 Implementar Ações Corretivas
		2.3 Gerenciar Ações Corretivas
Planejamento de Projeto	1 Estabelecer Estimativas	1.1 Estimar o Escopo do Projeto
		1.2 Estabelecer Estimativas para Atributos de Produtos de Trabalho e de Tarefas
		1.3 Definir Ciclo de Vida do Projeto
		1.4 Determinar Estimativas de Esforço e Custo
	2 Elaborar um Plano de Projeto	2.1 Estabelecer Orçamento e Cronograma
		2.2 Identificar Riscos do Projeto
		2.3 Planejar Gestão de Dados
		2.4 Planejar Recursos do Projeto
		2.5 Planejar Habilidades e Conhecimento Necessários
		2.6 Planejar o Envolvimento das Partes Interessadas
		2.7 Estabelecer o Plano do Projeto
	3 Obter Comprometimento com o Plano	3.1 Revisar Planos que Afetam o Projeto
		3.2 Conciliar Carga de Trabalho e Recursos
		3.3 Obter Comprometimento com o Plano

Continua

Área de Processo	Metas Específicas	Práticas Específicas
Gestão Quantitativa de Projeto	1 Preparar-se para o Gerenciamento Quantitativo	1.1 Estabelecer os Objetivos do Projeto
		1.2 Compor o Processo Definido
		1.3 Selecionar os Subprocessos e Atributos
		1.4 Selecionar Medidas e Técnicas Analíticas
	2 Gerenciar Estatisticamente o Desempenho de Subprocessos	2.1 Monitorar o Desempenho dos Subprocessos Selecionados
		2.2 Gerenciar o Desempenho do Projeto
		2.3 Desempenhar Análise de Causa-raiz
Gestão de Requisitos	1 Gerenciar Requisitos	1.1 Entender os Requisitos
		1.2 Obter Comprometimento com os Requisitos
		1.3 Gerenciar Mudanças nos Requisitos
		1.4 Manter Rastreabilidade Bidirecional dos Requisitos
		1.5 Assegurar o alinhamento entre Produtos de Trabalho, Planos de Projeto e Requisitos
Gestão de Riscos	1 Preparar-se para Gestão de Riscos	1.1 Determinar Fontes e Categorias de Riscos
		1.2 Definir Parâmetros para Riscos
		1.3 Estabelecer uma Estratégia para Gestão de Riscos
	2 Identificar e Analisar Riscos	2.1 Identificar Riscos
		2.2 Avaliar, Categorizar e Priorizar Riscos
	3 Mitigar Riscos	3.1 Elaborar Planos de Mitigação de Riscos
		3.2 Executar Planos de Mitigação de Riscos
Gestão de Contrato com Fornecedores	1 Estabelecer Contratos com Fornecedores	1.1 Determinar Tipo de Aquisição
		1.2 Selecionar Fornecedores
		1.3 Estabelecer Contratos com Fornecedores
	2 Cumprir Contratos com Fornecedor	2.1 Executar Contrato com Fornecedor
		2.2 Aceitar Produto Adquirido
		2.3 Assegurar Transferência de produtos

Fonte: CMMI-DEV V1.3 (2010)

Categoria Engenharia:

Área de Processo	Metas Específicas	Práticas Específicas
Integração de Produto	1 Preparação para Integração de Produto	1.1 Determinar Sequência de Integração
		1.2 Estabelecer Ambiente de Integração do Produto
		1.3 Estabelecer Procedimentos e Critérios para Integração do Produto
	2 Assegurar Compatibilidade das Interfaces	2.1 Revisar Descrições de Interfaces para Assegurar Completude
		2.2 Gerenciar Interfaces
	3 Montar Componentes do Produto e Entregar Produto	3.1 Confirmar se os Componentes do Produto estão Prontos para Serem Integrados
		3.2 Montar Componentes do Produto
		3.3 Avaliar Componentes de Produto Montados
		3.4 Empacotar e Entregar Produto ou Componente de Produto
Desenvolvimento de Requisitos	1 Desenvolver Requisitos de Cliente	1.1 Levantar Necessidades
		1.2 Transformar as Necessidades das Partes Interessadas em Requisitos de Cliente
	2 Desenvolver Requisitos de Produto	2.1 Estabelecer Requisitos de Produto e de Componente de Produto
		2.2 Alocar Requisitos de Componente de Produto
		2.3 Identificar Requisitos de Interface
	3 Analisar e Validar Requisitos	3.1 Estabelecer Conceitos Operacionais e Cenários
		3.2 Estabelecer uma Definição da Funcionalidade Requerida e dos Atributos de Qualidade
		3.3 Analisar Requisitos
		3.4 Analisar Requisitos Visando ao Balanceamento
		3.5 Validar Requisitos
Solução Técnica	1 Selecionar Soluções de Componentes de Produto	1.1 Desenvolver Soluções Alternativas e Critérios de Seleção
		1.2 Selecionar Soluções de Componentes de Produto
	2 Desenvolver <i>Design</i>	2.1 Desenvolver o <i>Design</i> do Produto ou dos Componentes de Produto
		2.2 Estabelecer Pacote de Dados Técnicos
		2.3 Projetar Interfaces Utilizando Critérios
		2.4 Analisar Alternativas: Desenvolver, Comprar ou Reusar
	3 Implementar <i>Design</i> do Produto	3.1 Implementar <i>Design</i>
		3.2 Elaborar Documentação de Suporte ao Produto

Continua

Continuação

Área de Processo	Metas Específicas	Práticas Específicas
Validação	1 Preparar-se para Validação	1.1 Selecionar Produtos para Validação
		1.2 Estabelecer Ambiente de Validação
		1.3 Estabelecer Procedimentos e Critérios de Validação
	2 Validar Produto ou Componentes de Produto	2.1 Realizar Validação
		2.2 Analisar Resultados de Validação
Verificação	1 Preparar-se para Verificação	1.1 Selecionar Produtos de Trabalho para Verificação
		1.2 Estabelecer Ambiente de Verificação
		1.3 Estabelecer Procedimentos e Critérios de Verificação
	2 Realizar Revisão por Pares	2.1 Preparar-se para Revisão por Pares
		2.2 Conduzir Revisão por Pares
		2.3 Analisar Dados de Revisão por Pares
	3 Verificar Produtos de Trabalho Selecionados	3.1 Realizar Verificação
		3.2 Analisar Resultados da Verificação

Fonte: CMMI-DEV V1.3 (2010)

Categoria Suporte:

Área de Processo	Metas Específicas	Práticas Específicas
Análise e Resolução de Causas	1 Determinar Causas de Defeitos	1.1 Selecionar Dados de Defeitos para Análise
		1.2 Analisar Causas
	2 Tratar Causas de Defeitos	2.1 Implementar Propostas de Ação
		2.2 Avaliar Efeitos de Mudanças
		2.3 Registrar Dados
Gestão de Configuração	1 Estabelecer <i>Baselines</i>	1.1 Identificar Itens de Configuração
		1.2 Estabelecer um Sistema de Gestão de Configuração
		1.3 Criar ou Liberar <i>Baselines</i>
	2 Acompanhar e Controlar Mudanças	2.1 Acompanhar Solicitações de Mudança
		2.2 Controlar Itens de Configuração
	3 Estabelecer Integridade	3.1 Estabelecer Registros de Gestão de Configuração
		3.2 Executar Auditorias de Configuração
Análise e Tomada de Decisões	1 Avaliar Alternativas	1.1 Estabelecer Diretrizes para Análise e Decisão
		1.2 Estabelecer Critérios de Avaliação
		1.3 Identificar Soluções Alternativas
		1.4 Selecionar Métodos de Avaliação
		1.5 Avaliar Alternativas
		1.6 Selecionar Soluções
Medição e Análise	1 Alinhar Atividades de Medição e Análise	1.1 Estabelecer Objetivos de Medição
		1.2 Especificar Medidas
		1.3 Especificar Procedimentos de Coleta e Armazenamento de Dados
		1.4 Especificar Procedimento de Análise
	2 Fornecer Resultados de Medição	2.1 Coletar Dados Resultantes de Medição
		2.2 Analisar Dados Resultantes de Medição
		2.3 Armazenar Dados e Resultados
		2.4 Comunicar Resultados
Garantia da Qualidade de Processo e Produto	1 Avaliar Objetivamente Processos e Produtos de Trabalho	1.1 Avaliar Objetivamente os Processos
		1.2 Avaliar Objetivamente Produtos de Trabalho e Serviços
	2 Fornecer Visibilidade	2.1 Comunicar e Assegurar a Solução de não Conformidades
		2.2 Estabelecer Registros

Fonte: CMMI-DEV V1.3 (2010)