



UNIVERSIDADE DO SUL DE SANTA CATARINA
LUIZ EDUARDO PEREIRA DA MATA

**MELHORIA DE UM PROCESSO ESPECIALIZADO EM CUSTOMIZAÇÃO E
MANUTENÇÃO DE SOFTWARE LEGADO**

Palhoça, Universidade do Sul de Santa Catarina

2015

LUIZ EDUARDO PEREIRA DA MATA

**MELHORIA DE UM PROCESSO ESPECIALIZADO EM CUSTOMIZAÇÃO E
MANUTENÇÃO DE SOFTWARE LEGADO**

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação

Orientador: Prof. Maria Inés Castiñeira

Palhoça, Universidade do Sul de Santa Catarina

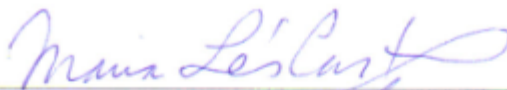
2015

LUIZ EDUARDO PEREIRA DA MATA

**MELHORIA DE UM PROCESSO ESPECIALIZADO EM CUSTOMIZAÇÃO E
MANUTENÇÃO DE SOFTWARE LEGADO**

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Bacharel em Sistemas de Informação e aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina.

Palhoça, 19 de novembro de 2015.



Professor e orientador Maria Inês Castiņeira, Dr.
Universidade do Sul de Santa Catarina



Prof. Roberto Fabiano, Me.
Universidade do Sul de Santa Catarina



Alessandra Mohr, Esp.
Universidade do Sul de Santa Catarina

Dedico este trabalho a minha mãe que sempre esteve ao meu lado com o apoio necessário a minha formação pessoal e profissional, me possibilitando seguir confiante em minha vida independente dos obstáculos que encontrei.

AGRADECIMENTOS

A minha mãe pelo carinho, incentivo nas horas difíceis de desânimo e cansaço e toda sua contribuição na minha educação, além de seu apoio incondicional.

Aos meus familiares pela força e incentivo para a conclusão da graduação.

A minha namorada por estar ao meu lado nas horas difíceis e me incentivar na conclusão do curso.

Em especial a minha orientadora Maria Inés Castiñeira, pelos ensinamentos, orientação e pelo apoio na elaboração deste trabalho de conclusão de curso.

Agradeço a todos os professores durante a trajetória no meu curso pelo grande aprendizado e experiência passada, suas dedicações e apoio e por me proporcionar o conhecimento racional e caráter de afetividade da educação no processo de formação do profissional.

Agradeço a empresa Dígitro Tecnologia, pela oportunidade de trabalho e por todo aprendizado desde o ano 2008, em acreditar no meu potencial e pelo apoio prestado durante o meu curso.

Aos meus colegas de trabalho do setor de desenvolvimento e do suporte da Dígitro Tecnologia, que direta ou indiretamente me ajudarem neste trabalho.

Enfim, a todos que tornaram os momentos da minha vida universitária especial e inesquecível.

“Não se deve aceitar jamais como verdadeira alguma coisa de que não se conheça, a evidência como tal, isto é, evitar cuidadosamente a precipitação e a prevenção, incluindo apenas, nos juízos, aquilo que se mostrar de modo tão claro e distinto ao espírito que não subsista dúvida alguma.” (RENÉ DESCARTES)

RESUMO

Este trabalho foi desenvolvido na empresa Dígitro Tecnologia da região da Grande Florianópolis, Santa Catarina. O principal objetivo foi compreender e propor melhorias no fenômeno da manutenção e customização de software legados e as dificuldades para executar tarefas ligadas à manutenção de software e manter a qualidade do software legado. Visando entender melhor as dificuldades relacionadas a estas atividades foi feito um estudo de caso de uma área do setor de desenvolvimento de software da empresa, responsável pelas atividades de customização e manutenção de software de legado. A partir disso é proposto um modelo de processo de manutenção e customização de software, com foco na qualidade dos softwares legados, podendo assim ser posto em prática garantindo sua aplicação de maneira contínua. Como embasamento para esta proposta foi estudado os processos de desenvolvimento de software, a manutenção de software, as metodologias tradicionais, ágeis e as específicas para manutenção e customizações, além de ferramentas e modelos de qualidade disponíveis no mercado. A proposta consiste em um novo processo para as atividades de customização e manutenção de software da empresa utilizando novos métodos, procedimentos e ferramentas de forma a ser contemplado pelo modelo de qualidade do MPS.BR. Esta proposta pode servir como um modelo de processo para as atividades de manutenção e customização de software que podem ser utilizadas por demais empresas. Apesar desta proposta não ter sido aplicado na empresa ela proporcionou diversas facilidades na resolução dos problemas relacionados às atividades de customização e manutenção de software legado, além de qualificar estas atividades em conformidade com o modelo de qualidade MPS.BR em seu nível G. Além disso, com base no trabalho realizado foi possível destacar os problemas relacionados às atividades de customização e manutenção de software, e principalmente, o problema da empresa em manter os softwares legados. Também se verificou uma carência de metodologias e procedimentos, especializados na evolução do software, assim como da área acadêmica para tratar o assunto da manutenção de software.

Palavras-chave: Engenharia de Software. Manutenção de Software. Customização de Software. Software Legado. Qualidade de Software. Melhoria do Processo de Software Brasileiro (MPS.BR).

LISTA DE ILUSTRAÇÕES

Figura 1 - As camadas da engenharia de software.	8
Figura 2 - Um modelo geral do processo de software.	14
Figura 3 - Estágio de teste.	15
Figura 4 - Estratégia de teste.	16
Figura 5 - Evolução do sistema.	17
Figura 6 - O modelo cascata.	21
Figura 7 - O modelo incremental.	23
Figura 8 - O modelo espiral.	25
Figura 9 - Paradigma da prototipação.	27
Figura 10 - Estrutura do RUP - Rational Unified Process.	29
Figura 11 - As quatro dimensões do XP.	38
Figura 12 - O processo Extreme Programming e suas fases.	39
Figura 13 - Fluxo geral do processo Scrum.	43
Figura 14 - Fatores de qualidade de software.	47
Figura 15 - Processos do ciclo de vida da norma ISO/IEC 12207.	53
Figura 16 - Níveis de maturidade do CMMI.	56
Figura 17 - Fluxo dos envolvidos na manutenção de software.	60
Figura 18 - Atividades do processo de manutenção de software e sistema (ISO/IEC 12207).	69
Figura 19 - Avaliação de um Software Legado.	76
Figura 20 - O processo de reengenharia de software.	78
Figura 21 - Estrutura do MPS.BR.	81
Figura 22 - Produtos da solução PABX.	96
Figura 23 - Produtos da solução Easy Call.	97
Figura 24 - Produtos da solução de URA.	98
Figura 25 - Produtos da solução Gravação.	98
Figura 26 - Produtos da solução Contact Center.	99
Figura 27 - Plataformas NGC.	100
Figura 28 - Organograma Dígitro Tecnologia.	101
Figura 29 - Estrutura das equipes do Desenvolvimento.	103
Figura 30 - Abordagem de processos na Dígitro Tecnologia.	105
Figura 31 - Organograma dos setores participantes do processo atual.	106
Figura 32 - Atores participantes do processo de customização de software legado.	107
Figura 33 - O processo de customização do software legado “As Is”.	108
Figura 34 - O processo de especificação técnica da customização do software legado.	110
Figura 35 - O processo de desenvolvimento da customização do software legado.	110
Figura 36 - O processo de testes da customização do software legado.	111
Figura 37 - Atores participantes do processo de manutenção de software legado.	115
Figura 38 - O processo de manutenção de software legado “As Is”.	116
Figura 39 - O processo de análise do problema informado pelo cliente.	117
Figura 40 - O processo de manutenção do software legado.	118
Figura 41 - O processo de testes do software legado.	118
Figura 42 - O novo modelo de processo.	123
Figura 43 - Ciclo desenvolvimento do TDD.	124
Figura 44 - A Gherkin language do BDD.	126
Figura 45 - Template do CRS - Customization Requirements Specification.	127

Figura 46 - Template do MRS – Maintenance Requirements Specification.....	128
Figura 47 - Exemplo de um arquivo feature.....	130
Figura 48 - Abordagem de processos do novo processo.	132
Figura 49 – Atores participantes do processo de customização de software legado " <i>To Be</i> ".	133
Figura 50 - O processo de customização de software legado " <i>To Be</i> ".	134
Figura 51- O processo de especificação técnica da customização do software legado " <i>To Be</i> ".	136
Figura 52 - O processo de desenvolvimento da customização do software legado " <i>To Be</i> ".	137
Figura 53 - O processo de testes da customização do software legado " <i>To Be</i> ".	137
Figura 54 - O processo de Validação do CRS com os testes do software legado " <i>To Be</i> ".	138
Figura 55 - O processo de validação do CRS com o desenvolvimento do software legado " <i>To Be</i> ".	138
Figura 56 - O processo de manutenção de software legado " <i>To Be</i> ".	142
Figura 57 – O processo de análise do problema informado pelo cliente " <i>To Be</i> ".	144
Figura 58 – O processo de especificação da manutenção do software " <i>To Be</i> ".	144
Figura 59 – O processo de manutenção do software " <i>To Be</i> ".	145
Figura 60 – O processo de testes da manutenção do software " <i>To Be</i> ".	145
Figura 61 – O processo de validação do MRS " <i>To Be</i> ".	146

LISTA DE QUADROS

Quadro 1 - As oito leis da evolução de software.....	18
Quadro 2 - Virtudes e problemas do modelo cascata.....	22
Quadro 3 - Virtudes e problemas do modelo incremental.....	24
Quadro 4 - Virtudes e problemas do modelo espiral.....	26
Quadro 5 - Virtudes e problemas da prototipação.....	28
Quadro 6 - As práticas do Extreme Programming.....	42
Quadro 7 - Áreas de conhecimento do SWBOK.....	46
Quadro 8 - Dimensões de qualidade.....	46
Quadro 9 - Fatores de qualidade de software.....	48
Quadro 10 - Principais características da garantia e controle de qualidade.....	50
Quadro 11 - Normas de Software.....	51
Quadro 12 - Áreas de processo necessárias para atingir um nível de maturidade.....	57
Quadro 13 - Classificação de softwares na manutenção de software.....	59
Quadro 14 - Pesquisa das características das atividades de manutenção de software.....	61
Quadro 15 - Situações estereotipadas.....	64
Quadro 16- Custo proporcional de manutenção do Software Legado.....	73
Quadro 17 - Diferencias do MPS.BR.....	82
Quadro 18 - Níveis de maturidade do modelo MR-MPS-SW do MPS.BR.....	83
Quadro 19 - Timeline Dígito Tecnologia.....	94
Quadro 20 - Atendimento do processo atual de customização de software legado "As Is" aos requisitos da gerência de projetos do nível G do MPS.BR.....	112
Quadro 21 - Atendimento do processo de customização de software legado "As Is" aos requisitos gerência de requisitos do nível G do MPS.BR.....	114
Quadro 22 - Atendimento do processo de manutenção de software legado "As Is" aos requisitos gerência de projetos do nível G do MPS.BR.....	119
Quadro 23 – Atendimento do processo de manutenção de software legado "As Is" aos requisitos gerência de requisitos do nível G do MPS.BR.....	121
Quadro 24 - Atendimento do processo de customização de software legado "To Be" pelos resultados esperados da gerência de requisitos do nível G do MPS.BR.....	140
Quadro 25 - Atendimento do processo de manutenção do software legado "To Be" pelos resultados esperados da gerência de requisitos do nível G do MPS.BR.....	147

LISTA DE GRÁFICOS

Gráfico 1 - Distribuição do esforço de manutenção.....	65
--	----

SUMÁRIO

1. INTRODUÇÃO	1
1.1 TEMA E PROBLEMA	2
1.2 OBJETIVOS	3
1.2.1 OBJETIVO GERAL	3
1.2.2 OBJETIVOS ESPECÍFICOS.....	3
1.3 JUSTIFICATIVA	4
1.4 ESTRUTURA DA MONOGRAFIA	5
2. ENGENHARIA DE SOFTWARE	7
2.1 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE.....	9
2.2 FASES DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE	10
2.2.1 Especificação de Software	11
2.2.1.1 Análise de requisitos funcionais.....	12
2.2.1.2 Análise de requisitos não funcionais	12
2.2.2 Projeto e implementação	13
2.2.3 Validação do software.....	15
2.2.4 Evolução do software.....	17
2.3 ATIVIDADES DE APOIO	19
2.4 MODELOS DE PROCESSO DE SOFTWARE	20
2.4.1 Cascata	21
2.4.2 Incremental.....	23
2.4.3 Espiral.....	24
2.4.4 Prototipação	26
2.5 PROCESSO UNIFICADO (RUP)	28
2.5.1 Práticas	29
2.5.2 Fases	30
2.5.2.1 Concepção	31
2.5.2.2 Elaboração.....	31
2.5.2.3 Construção.....	31
2.5.2.4 Transição	32
2.5.3 Atividades	32
2.5.3.1 Modelagem de negócios.....	32
2.5.3.2 Requisitos.....	33
2.5.3.3 Análise e Projeto	33
2.5.3.4 Implementação	33
2.5.3.5 Teste	34
2.5.3.6 Implantação	34
2.5.3.7 Gerência de configuração e alteração.....	34
2.5.3.8 Gerência de projeto	34
2.5.3.9 Ambiente	35
2.6 MODELOS AGÉIS	35
2.6.1 Manifesto Ágil	36
2.6.2 Princípios	37
2.6.3 Extreme Programming (XP)	38
2.6.3.1 Fases.....	39
2.6.3.1.1 Planejamento.....	40
2.6.3.1.2 Projeto.....	40
2.6.3.1.3 Codificação	40
2.6.3.1.4 Testes.....	41

2.6.3.2	Princípios.....	41
2.6.3.3	Práticas	42
2.6.4	Scrum	43
2.7	QUALIDADE DE SOFTWARE	44
2.7.1	Dimensões de qualidade	46
2.7.2	Fatores de qualidade.....	47
2.7.3	Gerenciamento de qualidade	48
2.7.4	Garantia de qualidade	49
2.7.5	Controle de qualidade	50
2.7.6	Normas.....	50
2.7.6.1	ISO 9001	51
2.7.6.2	ISO/IEC 12207.....	52
2.8	MODELOS DE QUALIDADE DE SOFTWARE.....	54
2.8.1	CMMI	54
2.8.1.1	Níveis de maturidade.....	55
3.	MANUTENÇÃO DE SOFTWARE.....	58
3.1	DEFINIÇÕES	58
3.2	ASPECTOS HISTÓRICOS	60
3.3	TIPOS DE MANUTENÇÃO DE SOFTWARE.....	62
3.3.1	Manutenção corretiva.....	62
3.3.2	Manutenção adaptativa.....	63
3.3.3	Manutenção evolutiva.....	63
3.3.4	Manutenção preventiva.....	63
3.4	CUSTOS	64
3.5	GERENCIAMENTO DE MANUTENÇÃO DE SOFTWARE.....	66
3.6	NORMA ISO/IEC 12207.....	69
3.6.1	Implantação do processo	70
3.6.2	Análise do problema e da modificação.....	70
3.6.3	Implantação da modificação	70
3.6.4	Revisão, aceitação da modificação.....	71
3.6.5	Migração	71
3.6.6	Descontinuação do software.....	71
3.7	SOFTWARE LEGADO	71
3.7.1	Definição	72
3.7.2	Crise do legado e evolução	73
3.7.3	Gerenciamento de software legado.....	75
3.8	REENGENHARIA DE SOFTWARE	77
4.	MPS.BR – MELHORIA DE PROCESSO DE SOFTWARE BRASILEIRO.....	80
4.1	ESTRUTURA.....	80
4.1.1	MR-MPS-SW – Modelo de Referência de Melhoria de Processo de Software.....	82
4.1.2	Níveis de Maturidade.....	83
4.1.2.1	Nível G	84
4.1.2.1.1	Gerência de Projetos.....	85
4.1.2.1.2	Gerência de Requisitos.....	86
4.1.3	MR-MPS-SV – Modelo de Referência de Melhoria de Processo de Serviços.....	86
4.1.4	MR-MPS-RH – Modelo de Referência de Melhoria de Processo para Gestão de Pessoas ..	86
4.1.5	MA-MPS – Método de Avaliação	87
4.1.6	MA-MPS – Modelo de Negócio.....	87
5.	METODOLOGIA	88
5.1	CLASSIFICAÇÃO DA PESQUISA	88
5.2	PROCEDIMENTOS METODOLÓGICOS.....	89
5.3	DELIMITAÇÕES.....	92

6. ESTUDO DE CASO.....	93
6.1 DESCRIÇÃO DA EMPRESA.....	93
6.1.1 História	94
6.1.2 Soluções.....	95
6.1.2.1 Pabx.....	96
6.1.2.2 Easy Call	97
6.1.2.3 Ura.....	97
6.1.2.4 Gravação	98
6.1.2.5 Contact Center.....	99
6.1.3 Plataformas.....	99
6.1.4 Estrutura organizacional.....	100
6.1.4.1 Diretoria de Desenvolvimento	102
6.1.4.1.1 Arquitetura	102
6.1.4.1.2 Projetos	102
6.1.4.1.3 Desenvolvimento	102
6.1.5 Modelos de Qualidade	103
6.1.5.1 TL 9000.....	104
6.1.5.2 ISO 9000	104
6.2 MODELAGEM DE PROCESSOS.....	104
6.2.1 Modelo atual do processo de customização do software legado, “As is”.....	105
6.2.1.1 Atores do processo “As Is”	106
6.2.1.2 O processo “As Is”	107
6.2.1.2.1 O processo de customização de software “As Is”	109
6.2.1.3 Comparativo do processo “As Is” ao modelo MPS.BR nível G.....	111
6.2.1.3.1 Comparativo do processo “As Is” ao processo gerência de projetos	111
6.2.1.3.2 Comparativo do processo “As Is” ao processo gerência de requisitos.....	113
6.2.2 Modelo atual do processo de manutenção do software legado, “As is”.....	114
6.2.2.1 Atores do processo “As Is”	114
6.2.2.2 O processo “As Is”	115
6.2.2.2.1 O processo de manutenção de software “As Is”	117
6.2.2.2.2 Comparativo do processo atual “As Is” ao processo gerência de projetos	119
6.2.2.2.3 Comparativo do processo “As Is” ao processo gerência de requisitos.....	121
6.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO	121
7. A PROPOSTA	122
7.1 DESCRIÇÃO DA PROPOSTA.....	122
7.1.1 Métodos.....	123
7.1.1.1 TDD – Test-Drive Development.....	124
7.1.1.1.1 Benefícios	124
7.1.1.2 BDD – Behaviour-Drive Development.....	125
7.1.1.2.1 The Gherkin language	126
7.1.1.2.2 Benefícios	126
7.1.2 Procedimentos	126
7.1.2.1 CRS – Customization Requirements Specification.....	127
7.1.2.2 MRS – Maintenance Requirements Specification.....	128
7.1.3 Ferramentas	128
7.1.3.1 CMake	129
7.1.3.1.1 Benefícios	129
7.1.3.2 Behave.....	129
7.1.3.2.1 Benefícios	130
7.1.3.3 Jenkins.....	130
7.1.3.3.1 Benefícios	131
7.1.3.4 Google Test.....	131
7.1.3.4.1 Benefícios	131
7.2 A MODELAGEM.....	132

7.2.1 O novo processo de customização do software legado, “To be”	132
7.2.1.1 Atores do processo “To Be”	132
7.2.1.2 O processo “To Be”	133
7.2.1.3 O processo de customização de software “To Be”	136
7.2.1.4 Comparativo do processo “To Be” ao modelo MPS.BR nível G	139
7.2.1.4.1 Comparativo do processo “As Is” ao processo gerência de projetos	139
7.2.1.4.2 Comparativo do processo “To Be” ao processo gerência de requisitos	139
7.2.2 Modelo atual do processo de manutenção do software legado, “To Be”	141
7.2.2.1 Atores do processo “To Be”	141
7.2.2.2 O processo “To be”	141
7.2.2.3 O processo de manutenção de software “To Be”	143
7.2.2.4 Comparativo do processo “To Be” ao modelo MPS.BR nível G	146
7.2.2.4.1 Comparativo do processo “As Is” ao processo gerência de projetos	146
7.2.2.4.2 Comparativo do processo “To Be” ao processo gerência de requisitos	147
7.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO	148
8. CONCLUSÃO	149
8.1 CONSIDERAÇÕES GERAIS	149
8.2 CONTRIBUIÇÕES	150
8.3 TRABALHOS FUTUROS	151
REFERÊNCIAS	152
APÊNDICE A	155
APÊNDICE B	156
APÊNDICE C	157

1. INTRODUÇÃO

O desenvolvimento de software é um processo muito complexo, pois envolve vários fatores que podem impactar em custos, prazos, cronogramas e na qualidade do software. Uma dessas complexidades é gerenciar grandes equipes que participam do processo de desenvolvimento do software. Esta pode ser uma tarefa difícil, requisitando muitas competências do gestor, a fim de cumprir prazos e cronogramas. Esta complexidade envolve fatores de gestão de pessoas e fatores estratégicos de processos. (PRESSMAN, 2011)

Segundo Soares (2004), Paduelli (2007) e Audy (2008) muitas organizações possuem estes problemas e acabam desenvolvendo software sem nenhuma gestão de processo o que acaba impactando no cronograma, custos e principalmente na qualidade do software. Por essas razões, para evitar esse tipo de problemas comuns nas empresas de tecnologias da informação, é fundamental adotar ou adaptar um modelo de processo de desenvolvimento de software, metodologias e práticas adequadas. Procurando reduzir essas falhas no desenvolvimento de software foi estabelecida a *Engenharia de Software*, que define um conjunto de disciplinas que incluem especificação, o desenvolvimento, o gerenciamento e a evolução dos sistemas de software. (AUDY, 2008).

Desde a sua criação a engenharia de software passou a criar e aperfeiçoar continuamente métodos, procedimentos e ferramentas para tornar a atividade de desenvolvimento de software uma tarefa que pudesse ser medida, controlada e avaliada. De um modo geral a engenharia de software forneceu um amadurecimento do conceito de desenvolvimento de software e de suas características e processos. (PADUELLI, 2007)

Entretanto, algumas organizações não possuem um processo de desenvolvimento de software bem definido e que atenda às questões de qualidade no desenvolvimento, assim acarretando com problemas em manter um software já entregue e em funcionamento. Devido a isto surge o aumento da atenção à manutenibilidade do software, que começa a ganhar importância nas organizações, pois muitas delas desenvolveram softwares que contam cinco, dez, ou até mesmo vinte anos de vida no mercado e por representarem investimentos significativos precisam continuar em funcionamento, momento no qual surge a necessidade da manutenção de software. (PADUELLI, 2007)

Quando um software é vital para a empresa ele irá exigir a atividade da manutenção de software de modo a ser aprimorado continuamente para atender constantes mudanças, em

termos de necessidade do cliente ou falhas residuais, que continuam a aparecer mesmo após o software ter sido entregue e alterado durante vários anos. (SCHACH, 2010).

Considerando essa importância da manutenção de software é um fato que as organizações possuem dificuldades no entendimento da atividade, assim como é evidenciado a falta da mão de obra qualificada de um profissional para tratar eficientemente os problemas oriundos da manutenção de software, sendo que cada vez mais softwares são construídos com técnicas não muito recomendadas. (PADUELLI, 2007)

Partindo deste problema, este trabalho visa entender melhor as dificuldades de uma das atividades do ciclo de desenvolvimento do software, que é a manutenção ou customização do software. Etapa esta que ocorre no software pós-entrega, portanto o enfoque do trabalho se justifica pela carência de informações para se ter uma base sólida na atividade do processo de customização e manutenção de software, tanto no meio acadêmico como profissional.

O trabalho irá apresentar um estudo de caso de uma área do setor de desenvolvimento de software da empresa Dígitro Tecnologia da região da Grande Florianópolis, Santa Catarina. E a partir disso será proposto um modelo de processo de manutenção e customização de software com foco na qualidade dos softwares, podendo assim ser posto em prática garantindo sua aplicação de maneira contínua.

1.1 TEMA E PROBLEMA

No setor de desenvolvimento as equipes trabalham com manutenção e customização dos softwares legados e não possuem um processo específico bem definido, voltado à atividade da manutenção e customização de software. E por não possuir esse processo, as customizações e as manutenções dos softwares são complexas, faltam documentações e principalmente qualidade no processo.

Com base nesta deficiência que não é somente da empresa em questão, mas do mercado em si que não possui fácil acesso a metodologias, modelos e processos para customização e manutenção de softwares legados, será realizado um estudo e proposto uma melhoria nos processos da empresa *case*, enfatizando a customização e a manutenção do software.

A literatura apresenta diversos modelos de qualidade de software, como o CMMI, MPS.BR, ou diversas normas ISO, entre outros (SOMMERVILLE, 2011; SOFTEX, 2015). Neste trabalho será utilizado o modelo brasileiro de qualidade, MPS.BR, desenvolvido pela Associação para Promoção da Excelência do Software Brasileiro (SOFTEX), como referência para as melhorias do processo estudado. Esse modelo consiste em sete níveis de qualidade, que iniciam com o nível G, primeiro estágio a ser atingido pela organização, e finalizam com o nível A.

1.2 OBJETIVOS

A seguir serão apresentados os objetivos gerais e específicos.

1.2.1 OBJETIVO GERAL

Realizar um estudo de caso no processo de desenvolvimento de uma empresa que realiza customização e manutenção de softwares legados e propor um processo para a customização e manutenção do software utilizando novas metodologias, ferramentas e procedimentos atendendo o modelo de qualidade MPS-BR nível G.

1.2.2 OBJETIVOS ESPECÍFICOS

- ✓ Estudar os processos de desenvolvimento de software, a manutenção de software, as metodologias tradicionais, ágeis e as específicas para manutenção e customizações.
- ✓ Analisar as metodologias e os processos de qualidade e sua aplicabilidade em manutenção e customização de software legados, realizando um estudo ou adaptação

para a manutenção e customização de softwares legados em empresas que não são uma fábrica de software.

- ✓ Descrever a organização da empresa abordada no *case*.
- ✓ Identificar e descrever os processos de customização e manutenção de software utilizado pela empresa abordada no *case*.
- ✓ Estudar as tecnologias existentes de apoio ao processo de customização e manutenção de software.
- ✓ Escolher e descrever as ferramentas, metodologias e procedimentos de apoio ao processo de customização e manutenção de software.
- ✓ Definir e propor, para a empresa um processo para a manutenção e customização para softwares legado, adequado às perspectivas do modelo de qualidade MPS-BR nível G.

1.3 JUSTIFICATIVA

As metodologias para desenvolvimento de software apontam conjuntos de práticas e métodos para aperfeiçoar a construção e o desenvolvimento de novos sistemas, porém existe uma grande demanda das empresas e do mercado para a manutenção de softwares existentes. (SOMMERVILE, 2011).

As empresas têm constantes dificuldades no processo de desenvolvimento de softwares, existem problemas desde a especificação do software até a validação do software ocorrendo grandes problemas em sua evolução ao longo dos anos. Devido a este problema as empresas tentam adotar algumas metodologias de desenvolvimento para solucionar os problemas nas diversas fases de desenvolvimento, porém muitas acabam não adotando nenhuma metodologia por não disporem de recursos suficientes para implantá-las ou simplesmente por que as metodologias tradicionais não são adequadas às realidades da empresa. (SOARES, 2004).

Por estas razões, segundo Soares (2004), muitas empresas acabam não utilizando nenhuma metodologia no processo de desenvolvimento. Como consequência o resultado desta falta de sistematização na produção de software é a baixa qualidade do produto final, além de dificuldade de cumprir prazos e custos pré-definidos e de inviabilizar a evolução do software.

Este cenário é muito comum em empresas que não são fábricas de software, ou seja, que realizam manutenção e customização de seus softwares legados. Devido ao processo de desenvolvimento de software não ser o produto fim para estas empresas a adoção de um processo de desenvolvimento bem definido é imprescindível para realizar manutenção e melhorias para que a empresa não perca lucro e competitividade no mercado.

Segundo Pressman (2011) conforme as fases do desenvolvimento vão passando, os custos com as correções aumentam substancialmente, já o prejuízo a imagem do produto é incomensurável se essas correções não forem feitas.

Uma empresa que não investe em qualidade do software tender a perder clientes e ter dificuldades para prospectar novos negócios, em um mercado cada vez mais competitivo a imagem do software e da empresa é uma confiabilidade que se transmite a seus clientes e isto é algo imprescindível para as vendas e ganho de novos mercados.

1.4 ESTRUTURA DA MONOGRAFIA

Este trabalho será organizado como descrito a seguir.

No capítulo um foi abordado o tema e problemas, os objetivos gerais e específicos e a justificativa do trabalho.

No capítulo dois será apresentado uma revisão da literatura sobre os seguintes assuntos: engenharia de software, fases do processo de desenvolvimento de software, atividades de apoio, modelos de processo de software, processo unificado, modelos ágeis, qualidade de software e modelos de qualidade de software.

No capítulo três será apresentado a revisão da literatura sobre o tema foco do trabalho que são os assuntos: manutenção de software, definições, aspectos históricos, tipos de manutenção de software, custos, gerenciamento de manutenção de software, norma ISO/IEC 12207 e software legado.

No capítulo quatro será apresentado a revisão da literatura sobre a qualidade de software abrangendo os assuntos: MPS.BR - Melhoria do Processo de Software Brasileiro, estrutura do MPS-BR, modelos de referência de processo de software, serviço e gestão de pessoas (MR-MPS-SW, MR-MPS-SV e MR-MPS-RH), método de avaliação (MA-MPS) e modelo de negócio (MA-MPS).

No capítulo cinco é apresentada a metodologia do trabalho que compreende: método da pesquisa, classificação da pesquisa, procedimentos metodológicos e delimitações.

O capítulo seis descreve o estudo de caso, que compreende: a descrição da empresa, seu histórico, as soluções comercializadas, sua estrutura organizacional e a modelagem do processo de customização e manutenção de software “*As Is*” e seu comparativo ao modelo de qualidade do MPS.BR nível G.

No capítulo sete descreve-se a proposta de um novo modelo de processo de customização e manutenção de software que atende aos critérios do nível G do modelo de qualidade MPS.BR.

No capítulo 8 é apresentada, pelo autor, a conclusão do trabalho, suas contribuições e seus trabalhos futuros.

2. ENGENHARIA DE SOFTWARE

A engenharia de software é uma disciplina das áreas das engenharias que se preocupa com todos os aspectos da produção de software, desde os estágios iniciais como o da especificação do software até a manutenção do software. Essa disciplina também pode incluir o gerenciamento do projeto de software e desenvolvimento de ferramentas, métodos e teorias para apoiar a produção do software. (SOMMERVILLE, 2011)

Pode ser caracterizada pela aplicação de princípios científicos, métodos, modelos, padrões e teorias que possibilitam gerenciar, planejar, modelar, projetar, implementar, medir, analisar, manter e aprimorar sistemas de software. Seu principal desafio é encontrar formas de construir um software conceitualmente utilizando-se de técnicas como, especificar, projetar, construir e testar um sistema de software visando software de qualidade. (PETERS, 2001).

O conceito da engenharia de software é o desenvolvimento e a manutenção de sistemas modulares e suas características são: adequação aos requisitos funcionais, efetivação de padrões de qualidade, fundamentação na *Tecnologia da Informação* viável e planejamento e gestão das atividades, recursos, custos e datas. (REZENDE, 2005).

Tem como objetivo utilizar os sólidos princípios de engenharia a fim de obter o software de maneira econômica e confiável se concentrando nos aspectos práticos do desenvolvimento de software. Seu próprio significado envolve conceitos de criação, construção, análise, desenvolvimento e manutenção. (ENGHLOM, 2010)

O objetivo final é a produção do software isento de falhas, entregue dentro de prazo e orçamento previstos e que atendas as necessidades do usuário final. (SCHACH, 2010).

O padrão IEE 610.12 define de forma mais abrangente a engenharia de software como: (SCHACH, 2010)

A Engenharia de Software é a aplicação de um ambiente sistemático, disciplinado e quantificável para o desenvolvimento, operacionalização e manutenção do software, ou seja, a aplicação da engenharia de software.

Outras definições da engenharia de software podem ser definidas pela maior ênfase dos aspectos gerenciais do desenvolvimento de software. Aplicando o uso de sólidos princípios da engenharia para que se possa obter um software de maneira econômica, confiável e eficiente. (PRESSMAN, 2011).

Tendo como base essas definições, Pressman (2011) entende que software em todas suas formas e seus campos de aplicações deve passar pelos processos de engenharia e que a engenharia de software é uma tecnologia em camadas e que qualquer abordagem a ela deve estar fundamenta no comprometimento organizacional com a qualidade. Essa visão do autor é apresentada na figura a seguir.

Figura 1 - As camadas da engenharia de software.



Fonte: Pressman (2011).

Rezende (2005) tem como conclusão que a engenharia de software é uma metodologia para desenvolvimento de software com os objetivos primários o aprimoramento da qualidade dos produtos de software visando sistematizar a produção, a manutenção e a evolução de modo que ocorra dentro de prazos e custos estimados utilizando princípios, métodos tecnologias e processos contínuos para seu aprimoramento.

Sommerville (2011) caracteriza-a como uma disciplina da engenharia e possuindo todos os aspectos da produção de software denotando aos *engenheiros de software* que apliquem teorias, métodos e ferramentas de forma seletiva e apropriada no desenvolvimento de um software dentro dos aspectos técnicos, organizacionais e contextuais.

Ainda que muitas dessas definições abrangentes tenham sido propostas todas elas convergem no sentido de apontar as necessidades para um maior rigor no desenvolvimento de software. (AUDY, 2008).

2.1 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

O processo de desenvolvimento de software é a maneira pela qual produzimos o software abrangendo desde a metodologia com seu modelo de ciclo de vida do software, técnicas adjacentes, ferramentas utilizadas e acima de tudo a equipe de desenvolvimento que está criando o software. (SCHACH, 2010).

Este processo é um conjunto de atividades que buscam criar um produto de software para automatizar, melhorar ou inovar algum processo feito manualmente no mundo real. Dentre as várias atividades que compõem esse processo podem ser mencionadas desde a análise de requisitos até a codificação e testes. Embora existam vários modelos de processo de desenvolvimento de software que definem a ordem, sequência e intensidade dessas atividades, existem atividades que são comuns a todos esses modelos e a qualquer projeto de desenvolvimento de software. Entre essas atividades podem ser mencionadas as seguintes. (SOMMERVILLE, 2011).

Especificação de Software: Definição das funcionalidades, requisitos, restrições e de qual a finalidade do software. Geralmente nesta fase do processo, o desenvolvedor interage mais com o cliente a fim de definir as características do software a ser construído.

Projeto e Implementação: O software é projetado e produzido de acordo com as especificações construídas na etapa de anterior, nesta fase são feitos modelos e diagramas que serão implementados em alguma linguagem de programação para a construção do software.

Validação de Software: O software construído é validado com o usuário final a fim de garantir que todos os itens da especificação foram atendidos.

Evolução do Software: O software precisa evoluir, para se útil e customizável para o cliente, sendo assim pode ter novas funcionalidades e com isto o software pode evoluir para algo mais robusto.

O processo de desenvolvimento de software é a chave para a construção de um software de qualidade, ele define as bases para gerenciar o desenvolvimento como um guia do uso de técnicas e métodos de engenharia. Quando uma modelo (ou metodologia) de

desenvolvimento é adotada ela auxilia diretamente na sincronização, fornecendo a todos os membros da equipe de desenvolvimento uma nomenclatura comum de tarefas e atividades. (AUDY, 2008).

A partir destas definições podemos considerar que de forma geral um processo de software padrão pode ser visto como um conjunto de atividades, métodos, ferramentas e práticas que são utilizadas para construção de um software. Na definição de um processo de software devem ser consideradas as seguintes informações: atividades a serem realizadas, recursos necessários, artefatos requeridos e produzidos, procedimentos adotados e o modelo de ciclo de vida utilizado (FALBO, 1998).

Um projeto de software possui fases identificáveis ao longo de seu desenvolvimento até que surja uma aplicação final, ou seja, existem várias maneiras de progredir diante dessas fases e cada maneira é chamada de processo de software.

2.2 FASES DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

As fases do processo de desenvolvimento de software é a definição dos passos que transformam aquela ideia no produto acabado, estas fases desempenham um papel essencial no processo de desenvolvimento de software, possibilitando ao gerente de projetos controlarem o processo de desenvolvimento. (PETERS, 2001).

Peters (2001) define uma visão tradicional do processo de software que é vista como uma sequência linear de atividades, onde existem seis atividades, engenharia de sistema, análise de requisitos, projeto, codificação, testes e manutenção.

Rezende (2005) define que o processo de software é um ciclo natural de vida e que abrange as seguintes fases: concepção, construção, implantação, implementação, maturidade e utilização plena, declínio, manutenção e morte.

Schach (2010) define que as seis fases do processo de desenvolvimento são: levantamento de necessidades, análise e especificação, projeto, implementação, manutenção e entrega.

Para Braude (2004) as principais fases de um processo de software são: análise de requisitos, projeto, implementação, testes e manutenção.

Pressman (2011) entende como sendo cinco atividades genéricas no processo de desenvolvimento de software, comunicação, planejamento, modelagem, construção e emprego.

Os diversos autores concordam com as atividades essenciais do processo de desenvolvimento de software embora agrupem essas atividades em diversas quantidades de fases: Sommerville (2011) define quatro fases, Braude (2004) e Pressman (2011) definem cinco fases, Schach (2010) e Peters (2001) definem seis fases e Rezendes (2005) define sete fases.

A seguir serão apresentadas, com maiores detalhes, as atividades de cada uma dessas fases, segundo a classificação de Sommerville (2011).

2.2.1 Especificação de Software

Os requisitos de um software são funções ou atividades que o software executa ou atende, quando pronto ou em fase de desenvolvimento. Eles devem ser definidos claramente e relatados explicitamente, ou seja, são condições ou capacitações que devem ser contempladas pelo software, solicitada pelo cliente ou por usuário para resolver um problema ou alcançar um objetivo (SOMMERVILLE, 2011).

Define-se como um processo que descreve a engenharia dos objetos, funções, estados, prioridades, necessidades, interface, modelos de fluxo de dados, análise detalhada de riscos e planos de teste de um sistema de software. (PETERS, 2001).

A análise de requisitos é uma tarefa da engenharia de software que efetua a ligação entre a alocação do software em nível de sistema, o projeto de software e anseios do cliente ou usuário. A partir do relato das necessidades do usuário possibilita o engenheiro de software especificar funções ou atividades visando o pleno atendimento das necessidades. O produto desta etapa são os requisitos funcionais e não funcionais. (PRESSMAN, 2011).

Ainda segundo Pressman, (2011) “A análise de requisitos resulta na especificação de características operacionais do software, indica a interface do software com outros elementos do sistema e estabelece restrições que o software deve atender”.

Enghlom (2010) define que esta fase no ciclo de desenvolvimento de software é necessária para entender e documentar as necessidades dos usuários do sistema.

2.2.1.1 Análise de requisitos funcionais

Os requisitos funcionais são fundamentais para elaborar um software que atenda a satisfação plenamente dos usuários envolvidos (desenvolvedor, clientes e usuários), quando bem definidos e relatados formalmente evitam custos de manutenção do software, eles devem ser desenvolvidos em concordância entre desenvolvedores, clientes e usuários. Eles são mais específicos e direcionados à solução de problemas, seu objetivo está mais relacionado com o resultado geral do produto do sistema. (SOMMERVILLE, 2011).

São requisitos que definem as funções ou ações que o usuário pode utilizar, fornecidas pelo sistema, regras de negócio e interfaces. (ENGHLOM, 2010).

Com base na descrição das necessidades do cliente ou usuário a equipe de projeto pode definir os requisitos funcionais do sistema, podendo assim especificar detalhadamente as funções, desempenho, interfaces e restrições do software. (PRESSMAN, 2011).

2.2.1.2 Análise de requisitos não funcionais

Os requisitos não funcionais são aqueles que não estão diretamente relacionados à funcionalidade de um sistema, sendo de suma importância durante o desenvolvimento de um sistema, podendo ser usados como critérios de seleção na escolha de alternativas de projeto, estilo arquitetural e forma de implementação. Muitas vezes, os requisitos não funcionais acabam gerando restrições aos funcionais. (SOMMERVILLE, 2011)

Eles estão relacionados ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, disponibilidade, suportabilidade, restrições de projeto, requisitos de implementação, requisitos de interface, requisitos físicos, segurança e tecnologias envolvidas. (ENGHLOM, 2010).

2.2.2 Projeto e implementação

O projeto de software é uma descrição da estrutura do software a ser desenvolvida, dos modelos, estrutura de dados, interfaces, componentes, plataforma do software e interação com outros sistemas, ou seja, todo o ambiente do software. (SOMMERVILLE, 2011).

Esta etapa expressa como a aplicação deve ser construída, as partes que estão envolvidas dentro do projeto e como elas devem ser montadas em um contexto de software. Isto consiste em um conjunto de documentos em geral, produzido a partir da análise de requisitos. (BRAUDE, 2004).

Schach (2010) define que o processo de projeto é o documento de especificação, uma descrição do que o produto como software deve fazer. Esta fase consiste em três atividades: projeto de arquitetura, no qual o produto como um todo é subdividido em componentes, denominados módulos. Em seguida estes módulos são projetados, esse processo é denominado o projeto detalhado e por último o projeto de documentos que descreve como o produto realiza aquilo que se deseja.

A elaboração do projeto é parte que envolve muitos passos que se concentram em quatro atributos: estrutura de dados, arquitetura de software, detalhes de procedimentos e caracterização da interface. Sua etapa final consiste no que é conhecido no processo de implementação. (PETERS, 2001).

As atividades do processo de projeto podem variar dependendo do tipo de software que esta sendo desenvolvido. A figura 2 mostra as quatro atividades que podem ser parte do processo de projeto de um software. (SOMMERVILLE, 2011)

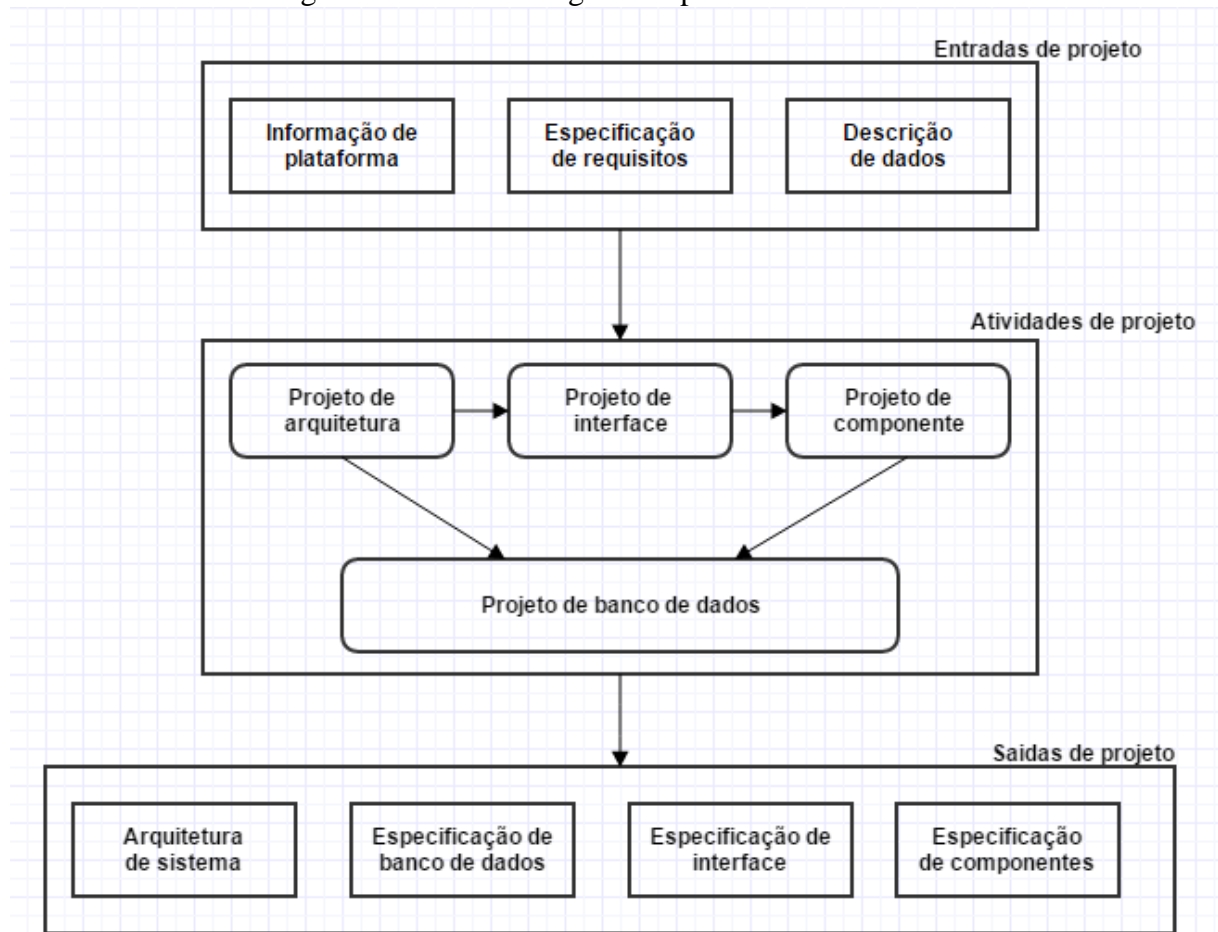
Projeto de arquitetura: No qual pode se identificar a estrutura geral do sistema, os componentes principais (módulos ou subsistemas), seus relacionamentos e como eles são distribuídos.

Projeto de interface: Define as interfaces entre os componentes do sistema.

Projeto de componente: No qual pega cada componente do sistema e projeta seu funcionamento.

Projeto de banco de dados: Projeta as estruturas de dados do software e como eles devem ser representados em um banco de dados.

Figura 2 - Um modelo geral do processo de software.



Fonte: Sommerville (2011).

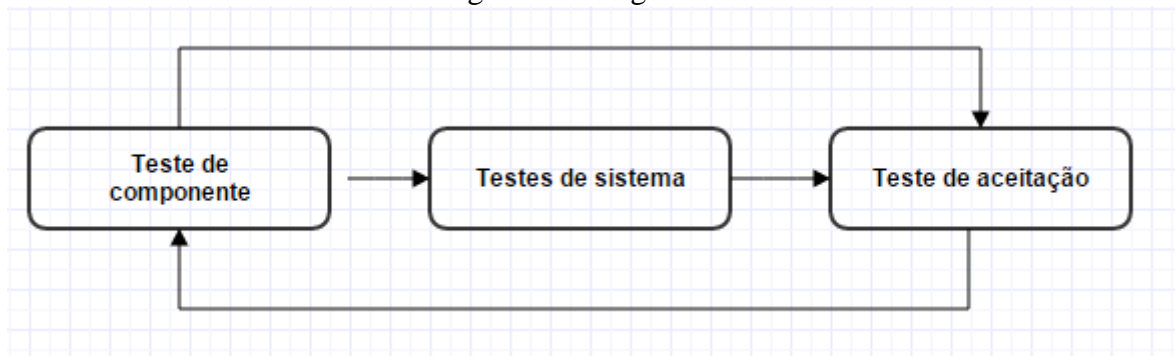
A implementação do desenvolvimento de software é um processo de conversão do que foi especificado do projeto em um sistema executável, envolvendo processo de projeto e programação de software podendo haver um refinamento da especificação do software. (SOMMERVILLE, 2011).

Segundo Peters (2001) a implementação é o processo de converter ou traduzir de forma legível o projeto em uma linguagem de computador.

2.2.3 Validação do software

A validação do software é a etapa que mostra se o software se adequa a sua especificação e satisfaz as especificações do cliente ou sistema. Nesta etapa são executados técnicas para validar o software que são, testes simulados no sistema, processos de verificação, inspeções e revisões, podendo essas técnicas ser utilizadas desde a especificação até a implementação. (SOMMERVILLE, 2011).

Figura 3 - Estágio de teste.



Fonte: Sommerville (2011)

Testes de componente: Os componentes do sistema são testados, onde cada componente é testado de forma independente separado dos outros.

Testes de sistema: Componentes do sistema são integrados para criar um sistema completo.

Teste de aceitação: Estágio final do processo de testes, antes que o software seja aceito para uso operacional. Nesta etapa o software é testado com dados fornecidos pelo cliente.

A fase de testes consiste em fornecer dados a entrada do sistema e comparar a saída com o que foi determinado na especificação dos requisitos do software. Os testes podem ser classificados segundo diferentes critérios, por exemplo, segundo a forma de funcionamento podem ser classificados como sendo de caixa preta ou caixa branca. (PRESSMAN, 2011)

Segundo a fase do processo de software, Pressman (2011), os classifica em: teste de unidade, teste de integração, teste de sistema e teste de aceitação. Os testes de unidade testam partes do software (módulos, métodos, classes, etc.), os testes de sistema verificam o software

inteiro, os testes de integração sistematizam a integração dos módulos e os testes de aceitação são para verificar a consistência entre o sistema implementado e os requisitos, ou seja, se os requisitos são atendidos.

Figura 4 - Estratégia de teste.



Fonte: Pressman (2011).

Estas etapas são indispensáveis no desenvolvimento de um software, pois ajudam a descobrir erros (*bugs*) e outros defeitos no software. Porém mesmo que o sistema tenha passado por um rigoroso conjunto de testes nunca se pode ter total confiança, pois alguns erros podem permanecer presentes mesmo em software completamente testados e validados. (BRAUDE, 2004).

Alguns autores definem outros tipos de testes, Sommerville (2011), por exemplo, define três tipos de testes, testes de desenvolvimento, quando os componentes são testados pelas pessoas que desenvolveram, testes de sistemas, se os componentes são integrados para criar um sistema completo e testes de aceitação, quando o sistema é testado pelos dados do cliente e não dados simulados.

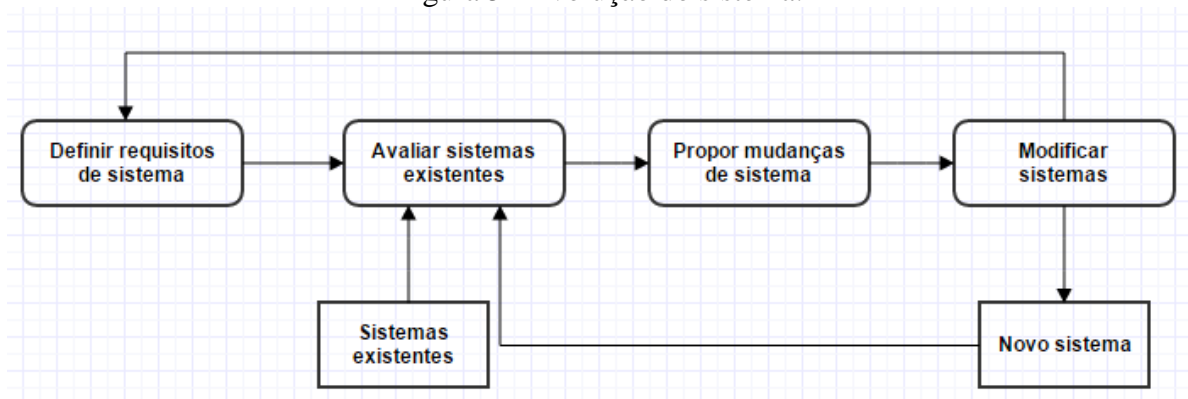
Todo software desenvolvido deve ser validado a procura de defeitos, ao longo do tempo esta atividade tem ganhado cada vez mais importância na engenharia de software, isto se deve ao fato do produto de software ter se tornado cada vez mais crítico.

2.2.4 Evolução do software

A evolução de software ocorre quando se alteram os atuais sistemas de software para atender aos novos requisitos, regras de negócios ou processos gerenciais. Dessa forma as mudanças são contínuas e o software deve se adaptar para continuar sendo útil. Sendo assim é correto pensar que a engenharia de software define o software como um processo evolutivo, no qual ele é constantemente alterado durante seu ciclo de vida para atender as mudanças de requisitos e necessidades do cliente. Como consequência disto a distinção entre desenvolvimento e manutenção é cada vez mais irrelevante, pois faz mais sentido o desenvolvimento de software e a manutenção serem considerados processos contínuos. (SOMMERVILLE, 2011)

Essas mudanças propostas podem vir de requisitos já existentes que não tenham sido implementados, novos requisitos, correções de *bugs* e novas melhorias do software vindas da própria equipe de desenvolvimento. O processo de mudanças e de evolução do sistema é cíclico e continua durante toda vida do sistema. (SOMMERVILLE, 2011)

Figura 5 - Evolução do sistema.



Fonte: Sommerville (2011).

Visando entender melhor as características da evolução de software, Manny Lehman publicou oito leis da evolução do software durante seus 30 anos de análise detalhada dos processos de software. Essas leis consideram não apenas o software em si, mas as características da organização que o desenvolve e mantém. (PRESSMAN, 2011).

A tabela abaixo mostra as oito leis da evolução do software segundo Pressman (2011).

Quadro 1 - As oito leis da evolução de software.

Mudança Contínua	Softwares foram desenvolvidos em um contexto de computação do mundo real e, portanto, irão evoluir com o tempo devendo ser adaptados continuamente ou se tornarão menos satisfatórios.
Complexidade Crescente	À medida que o software evolui sua complexidade aumenta a menos que se desenvolva um trabalho para mantê-la ou reduzi-la.
Autorregulação	O processo de evolução do software é autoregulado com a distribuição do produto e medidas de processo próximo do normal.
Conservação da Estabilidade Organizacional:	A taxa de atividade efetiva média de um software em evolução é invariante durante o seu tempo de vida.
Conservação da Familiaridade	Conforme o software evolui tudo que está associado a ele na organização deve manter o domínio do seu conteúdo e seu comportamento para uma evolução satisfatória.
Crescimento Contínuo	O conteúdo funcional do software deve ser continuamente ampliado para manter a satisfação do usuário.
Qualidade em Declínio	A qualidade do software parecerá que está diminuindo a menos que sejam mantidas e adaptadas as mudanças do ambiente operacional.
Sistema de Realimentação	Processos em evolução constituem em softwares de realimentação multinível, multilaço, multiagente e devem ser tratados como tais para que se obtenha melhora significativa em qualquer base razoável.

Fonte: Pressman (2011).

De uma maneira geral as leis de Lehman abordam características aplicáveis a grandes organizações que desenvolvem softwares com longo ciclo de vida, porém o que se tem é um panorama geral da evolução do software apresentando essas leis como necessidade de intervenção no software para que problemas possam ser evitados.

2.3 ATIVIDADES DE APOIO

A maioria dos projetos de software pode ter essas fases genéricas identificáveis no processo de desenvolvimento de software, sejam projetos simples, pequenos, grandes ou complexos. No entanto as atividades metodologias permanecerão as mesmas, que são aplicadas ao longo de um projeto e ajudam a equipe no gerenciamento, controle de progresso, qualidade, mudanças e risco. (PRESSMAN, 2011)

A seguir abordaremos essas atividades de apoio que têm como objetivo auxiliar outros processos visando principalmente à qualidade do software de acordo com Pressman (2011).

Gerência de configuração de software: Define as atividades para a aplicação de procedimentos administrativos e técnicos em todo o ciclo do software destinado a gerenciar os efeitos das mudanças.

Garantia de qualidade: Define as atividades para fornecer a garantia adequada de que o software durante o seu ciclo de vida esteja em conformidade com os requisitos especificados e estes sejam aderentes aos planos estabelecidos garantindo a qualidade do software.

Medição: Define as atividades para a validação do projeto de software determinando se o software atende ao uso específico proposto.

Documentação: Define as atividades para registrar informações produzidas, engloba criar artefatos, modelos, documentos, logs, formulários e etc.

Administração de riscos: Avalia os riscos que podem afetar a qualidade do software.

Revisões Técnicas: Avaliam artefatos para identificar e eliminar erros antes que sejam propagados.

Controle e acompanhamento: Avalia o progresso em relação ao plano do projeto para que sejam tomadas medidas necessárias para cumprir o cronograma do projeto

Reusabilidade: Definem quais são os critérios para reuso e estabelece mecanismos para obtenção de componentes reutilizáveis.

2.4 MODELOS DE PROCESSO DE SOFTWARE

O modelo do processo de software é uma representação simplificada de um processo de software, cada modelo possui uma perspectiva particular de um processo e, portanto fornece informações parciais sobre ele. (SOMMERVILLE, 2011)

Apesar da considerável contribuição que os modelos tradicionais proporcionaram a engenharia de software, fornecendo um roteiro razoavelmente eficaz no desenvolvimento de software ainda permaneceram barreiras e caos nos softwares produzidos. Para trazer ordem ao caos no desenvolvimento de software foram propostos modelos perspectivos que prescrevem um conjunto de elementos de processo, atividade metodológicas, ações da engenharia de software, garantia de qualidade, e mecanismos de controle de mudanças para o projeto de software, ou seja, cada modelo prescreve um fluxo de processo na qual os elementos do processo estão inter-relacionados. (PRESSMAN, 2011)

Sendo assim um processo de desenvolvimento de software é representado por um modelo, enquanto o modelo é operacionalizado por meio de uma metodologia que estabelece basicamente a seqüência das atividades e como elas se relacionam entre si, identificando o momento em que os métodos e as ferramentas são utilizados. (AUDY, 2008)

Todo processo de software deve ser executado de acordo com um modelo previamente definido, seguindo uma metodologia, método e ferramentas, tendo sempre o foco em que a camada básica é na qualidade do software. (PRESSMAN, 2011).

Deve-se salientar que não existe um modelo de processo ideal, sua escolha depende do tamanho da organização, da experiência da equipe de desenvolvimento de software, da natureza e complexidade do software, do prazo de entrega, entre outros fatores. Esses modelos não são descrições definitivas dos processos de software, são abstrações que podem ser usadas para explicar diferentes abordagens no desenvolvimento de software buscando adequar a um cenário específico, sendo que este modelo deve estar fortemente conectado ao processo desenvolvimento de software. (AUDY, 2008)

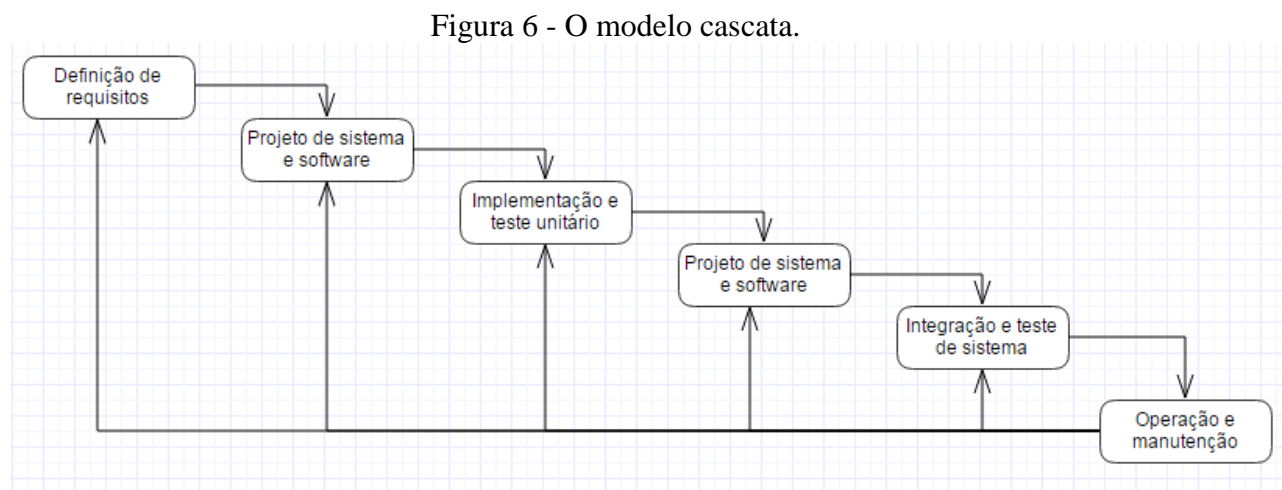
A seguir abordaremos os seguintes modelos do processo de software, *cascata*, *incremental*, *espiral* e *prototipação*.

2.4.1 Cascata

O modelo cascata considera as atividades fundamentais do processo de desenvolvimento de software, representando cada uma delas como fases distintas do processo, tais como, *especificação de requisitos*, *projeto de software*, *implementação*, *testes e manutenção*. (SOMMERVILLE, 2011).

Este modelo algumas vezes chamado de modelo clássico, sugere uma abordagem sequencial e sistemática no desenvolvimento de software, começando pelo levantamento de necessidades do cliente avançando pelas fases de planejamento, modelagem, construção, emprego e culminando no suporte contínuo do software. (PRESSMAN, 2011).

Sommerville (2011) e Peters (2001) definem o que as principais fases do modelo em cascata refletem diretamente nas atividades fundamentais do processo de desenvolvimento de software, são elas:



Fonte: Sommerville (2011).

Definição de requisitos: Os dados como serviços, restrições, metas, funcionalidades e recursos do sistema são coletadas com o cliente, quanto mais dados coletados menor a probabilidade de haver defeitos e manutenções no software futuramente. Esta etapa é

basicamente definida em especificação do sistema e os requisitos do software junto ao cliente final.

Projeto de sistema e software: Este processo envolve uma arquitetura geral do sistema que são estrutura de dados, arquitetura do software, detalhes de procedimentos, caracterização da interface e descrição das abstrações fundamentais do sistema de software.

Implementação e testes unitários: Nesta etapa o projeto de software é desenvolvido e o teste envolve a verificação de cada unidade ou módulo que atenda as especificações que foram feitas do projeto de software.

Integração e testes do sistema: É feita uma integração do sistema como um todo onde ele é testado a fim de assegurar que os requisitos do software tenham sido atendidos.

Operação e manutenção: Considerada a fase mais longa do ciclo de vida do modelo cascata, onde o sistema esta em operação e sendo utilizado pelo usuário que pode identificar erros que não foram descobertos em estágios iniciais do ciclo de vida requerendo demanda para a correção destes erros.

Sendo o ciclo de vida mais utilizada no desenvolvimento de software, este modelo apresenta algumas virtudes e problemas que podem ser observados na sua prática no quadro a seguir. (AUDY, 2008)

Quadro 2 - Virtudes e problemas do modelo cascata.

VIRTUDES	PROBLEMAS
Caracterizam fases das quais podem ser descritas técnicas para o seu desenvolvimento de forma clara.	Os projetos na vida real raramente seguem o fluxo seqüencial que o modelo propõe.
É o ciclo de vida amplamente utilizado na engenharia de software.	O cliente não saberá declarar todas suas exigências e demandas no inicio do projeto.
	O cliente deve ter paciência, pois qualquer erro detectado após a revisão do software pode ser um trabalho desastroso.

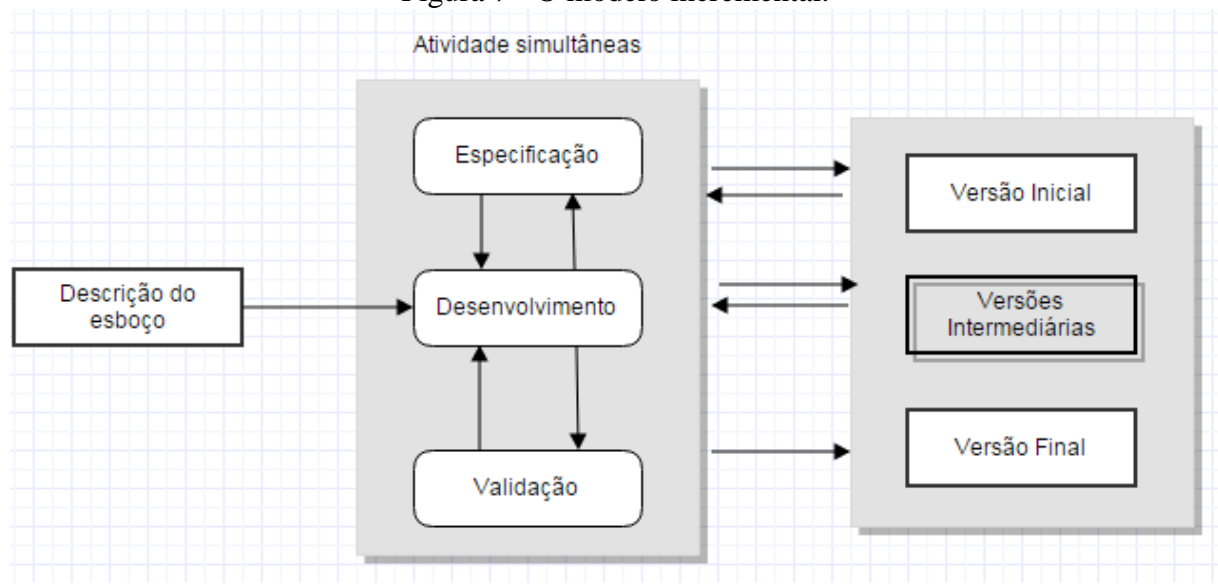
Fonte: Audy(2008).

2.4.2 Incremental

O modelo incremental tem como base desenvolver uma implementação ou projeto inicial a fim de expô-la ao usuário e continuar por meio de criação de várias versões até um sistema adequado e de acordo com o usuário. Cada versão incorpora alguma funcionalidade necessária e com urgência para o usuário, significando que ele poderá avaliar o software em um estágio relativamente inicial do desenvolvimento possibilitando verificar se o software esta atendendo os requisitos. (SOMMERVILLE, 2011)

O desenvolvimento incremental tem seu foco voltado para a entrega de um produto operacional a cada incremento, ou seja, as versões são incrementos do produto final e possuem capacidade de atender as demandas do usuário possuindo uma plataforma de avaliação do mesmo. (PRESSMAN, 2011)

Figura 7 - O modelo incremental.



Fonte: Sommerville (2011).

Este tipo de modelo possui uma abordagem que intercala as atividades de especificação, desenvolvimento e validação, o software é desenvolvido como uma série de versões, onde cada versão é incrementada com novas funcionalidades. (SOMMERVILLE, 2011)

Segundo Sommerville (2011) este modelo apresenta algumas virtudes importantes quando comparadas ao modelo cascata e problemas do ponto de vista de gerenciamento do projeto de software que são mostrados na tabela a seguir.

Quadro 3 - Virtudes e problemas do modelo incremental.

VIRTUDES	PROBLEMAS
O custo de acomodar as mudanças nos requisitos do cliente é reduzido, assim como a quantidade de análise e documentação a ser refeita é muito menor que o modelo cascata.	O processo não é visível, precisa de entregas rápidas para mensurar o progresso do projeto e caso forem desenvolvidos com rapidez não é economicamente viável produzir documentos das versões entregues.
É mais fácil obter feedbacks do cliente em relação ao desenvolvimento realizado, assim como ver o quanto foi desenvolvido, pois o cliente possui dificuldades em avaliar a evolução por meio de documentos no projeto de software.	A estrutura do software tende a se degradar com a adição de novos incrementos e funcionalidades desperdiçando tempo com refatoração para a melhoria do software e o tornando cada vez mais difícil e oneroso.
Possibilidade de implementação e entrega rápida de um software útil ao cliente, mesmo que não possua todas as funcionalidades, sendo assim o cliente pode obter ganhos a partir do software inicial.	

Fonte: Sommerville (2011).

2.4.3 Espiral

O modelo espiral é um gerador de modelos de processos dirigidos a risco e é utilizado para guiar a engenharia de sistemas intensivos de software que ocorre de forma concorrente e tem múltiplos envolvidos. Suas características principais são a abordagem cíclica para ampliar incrementalmente o grau de definição e implementação do software enquanto diminui o grau de risco do mesmo e a consistência de uma série de pontos de controle para assegurar o comprometimento dos interessados a busca de soluções de sistema que sejam amplamente satisfatórias e praticáveis. (PRESSMAN, 2011)

Este modelo combina prevenção e tolerância a mudanças e assume que mudanças são resultados de riscos de projeto e inclui atividades de gerenciamento de riscos para sua redução, sendo sua principal diferença o reconhecimento do risco. (SOMMERVILLE, 2011)

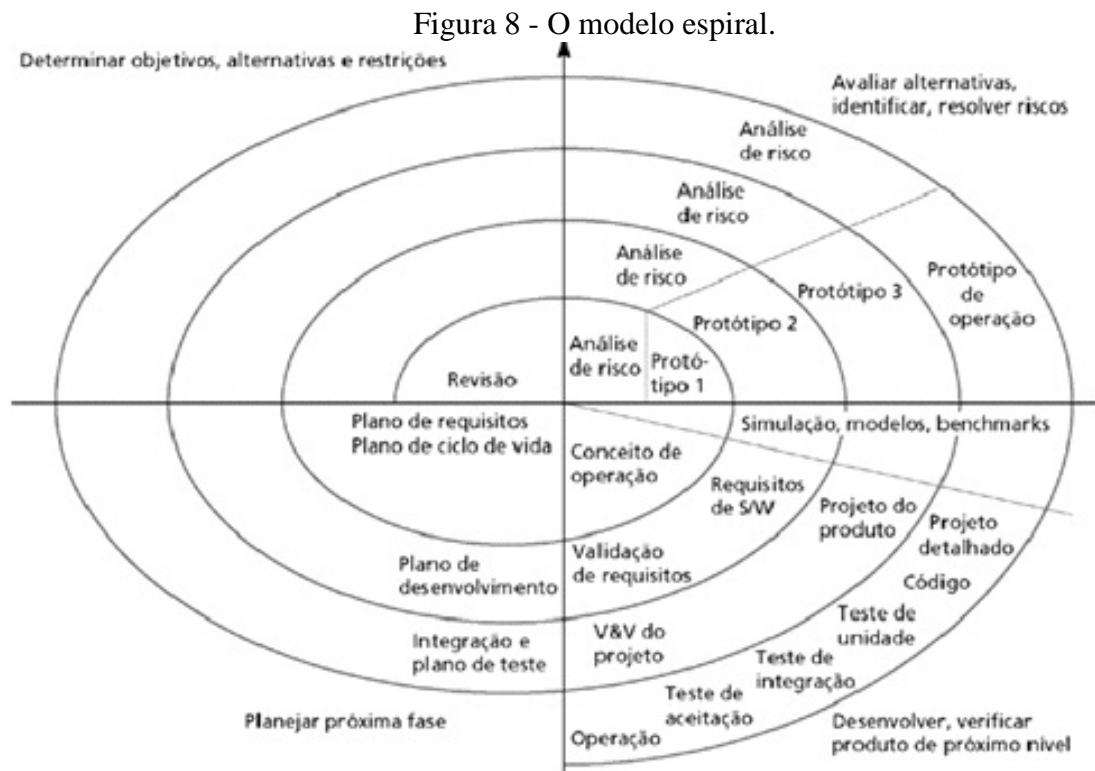
Sommerville (2011) define que a volta espiral é dividida em quatro setores:

Definição de objetivos: São definidos objetivos específicos nesta etapa do projeto, tais como restrições ao processo, plano de gerenciamento detalhado e riscos do projeto.

Avaliação e redução de risco: Para cada risco identificado é feita uma análise detalhada e medidas para redução deles são tomadas.

Desenvolvimento de validação: Após a avaliação dos riscos é selecionado um modelo de desenvolvimento do software.

Planejamento: O projeto é revisado e decidido a respeito da continuidade do modelo com mais uma volta espiral.



Fonte: Sommerville (2011).

O modelo espiral é uma abordagem realista para o desenvolvimento de software pelo fato de evoluir à medida que o processo avança, o desenvolvedor e o cliente compreendem e reagem melhor aos riscos em cada nível evolucionário, diferentemente de outros modelos que o software é entregue, este modelo pode ser adaptado para ser aplicado ao longo da vida do software. (PRESSMAN, 2011)

Este ciclo de vida é atualmente a abordagem mais realista para o desenvolvimento de software, sendo relativamente novo comparado aos outros dois modelos apresentados e suas principais virtudes e problemas são identificáveis na tabela a seguir. (AUDY, 2008)

Quadro 4 - Virtudes e problemas do modelo espiral.

VIRTUDES	PROBLEMAS
Permite o desenvolvimento evolutivo do software.	Pode ser difícil convencer grandes clientes de que a abordagem evolutiva é controlável.
Permite interação com o usuário.	Se um grande risco não for descoberto poderão ocorrer problemas.
Os requisitos não precisam ser todos definidos no começo.	É mais complexo e possui um custo mais alto que os demais modelos.
É iterativo, com um marco para avaliação ao final de cada iteração.	Pode não convergir para uma solução.
Busca integração do desenvolvimento com a prototipação.	Dependendo do projeto a relação custo benefício pode ser duvidosa.
Introduz a análise de risco.	

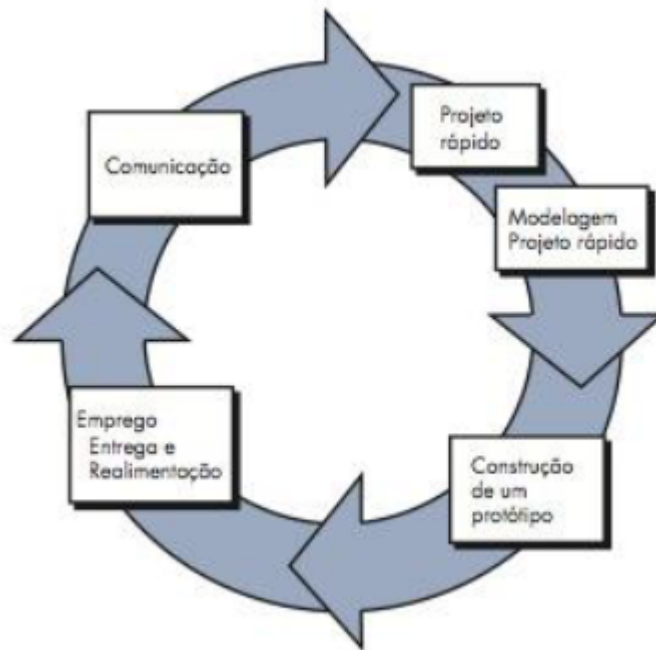
Fonte: Audy (2008).

2.4.4 Prototipação

Um protótipo é uma versão inicial do sistema de software usado para demonstrar conceitos, opções de projeto, identificar os problemas e suas possíveis soluções. Este modelo possui como objetivo *prototipar* um sistema tais como, prototipar uma interface, requisitos funcionais e projeto, para que possam ser experimentados logo no início do processo de desenvolvimento de software, podendo assim demonstrar a viabilidade técnica e funcional do sistema aos envolvidos. (SOMMERVILLE, 2011)

O paradigma deste modelo é a construção de um projeto rápido, ou seja, um protótipo que é desenvolvido a fim de receber um retorno do usuário para aprimorar os requisitos do sistema, possibilitando uma melhor compreensão e visando atender as necessidades do usuário. (PRESSMAN, 2011)

Figura 9 - Paradigma da prototipação.



Fonte: Pressman (2011).

Embora seja utilizada como um modelo de processo é comumente utilizado como uma técnica passível a ser implementada em outros modelos, independente da forma como será aplicada tem como objetivo auxiliar os envolvidos a compreender melhor o software que esta para ser desenvolvido. (PRESSMAN, 2011)

Um protótipo de software pode ser utilizado em um processo de desenvolvimento de software como no levantamento e validação de requisitos, no estudo de soluções específicas e projeto de interface, podendo assim ajudar a antecipar as mudanças que podem ser requisitadas. (SOMMERVILLE, 2011)

Da mesma forma que os modelos anteriores, a prototipação também apresenta algumas virtudes e problemas que podem ser observados em sua prática, conforme a tabela a seguir. (AUDY, 2008)

Quadro 5 - Virtudes e problemas da prototipação.

VIRTUDES	PROBLEMAS
Permite que o usuário interaja de forma mais ativa na modelagem do software.	O cliente quer resultados e muitas vezes não sabe ou entende que um protótipo está longe do software ideal que ele sequer imagina como será.
Facilita a identificação dos requisitos do software.	Na pressa para colocar um protótipo em funcionamento, pode ser escolhido um sistema operacional ou linguagem de programação com maior familiaridade pela equipe de desenvolvimento, podendo levar a soluções ineficientes caso não sejam bem planejadas.
Acelera o desenvolvimento do software.	

Fonte: Audy (2008)

2.5 PROCESSO UNIFICADO (RUP)

O processo unificado mais comumente conhecido como *Rational Unified Process* (RUP) é um modelo de processo de desenvolvimento de software moderno, considerado um processo híbrido, ele reúne elementos de todos os modelos de processo genéricos ilustrando boas práticas na especificação e projeto do software apoiando o modelo incremental e prototipação. (SOMMERVILLE, 2011)

Ele fornece uma abordagem disciplinada para determinar tarefas e responsabilidades dentro de uma organização assegurando a produção de software de alta qualidade suprimindo as necessidades dos usuários com orçamentos, agenda e prazos previsíveis. Para isto ele oferece *guidelines*, *templates* e guias de ferramentas para atividades críticas do desenvolvimento de software. (AUDY, 2008)

O processo unificado é uma tentativa de aproveitar as melhores características dos modelos tradicionais do processo de desenvolvimento, mas implementando muito dos princípios dos modelos de desenvolvimento ágil de software. (PRESSMAN, 2011)

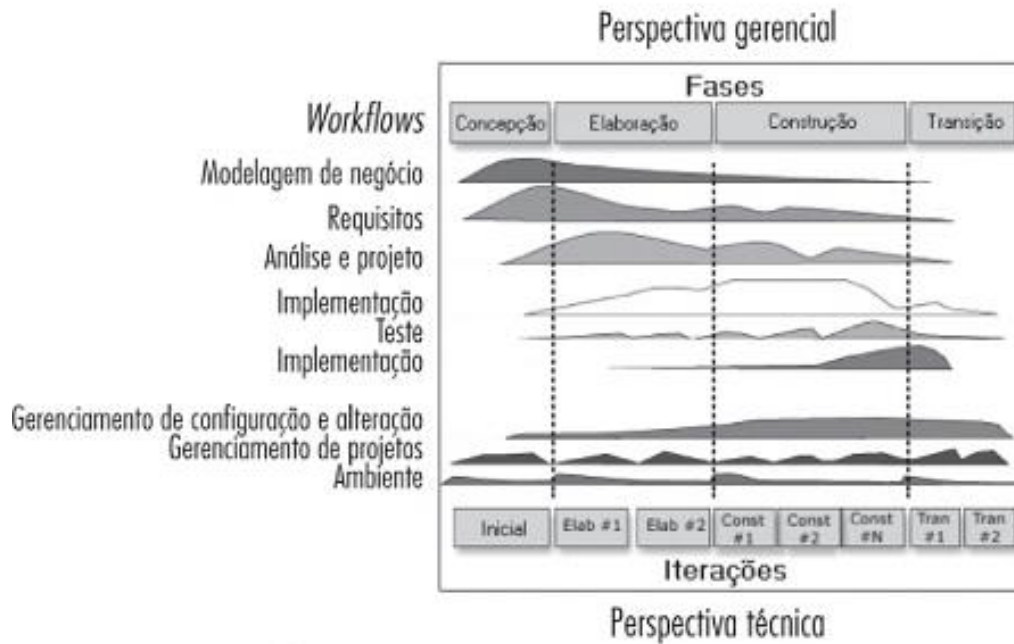
Segundo Sommerville (2011) é normalmente descrito em três perspectivas.

1. Uma perspectiva dinâmica que mostra as fases do modelo ao longo do tempo.
2. Uma perspectiva estática que mostra as atividades realizadas no processo.

3. Uma perspectiva prática que sugerem boas práticas a serem usadas durante o processo.

Sua estrutura de projeto pode ser vista em duas dimensões (Figura 6) na horizontal, a representação de tempo e os aspectos do processo, na vertical as disciplinas que agrupam atividades por natureza. (AUDY, 2008)

Figura 10 - Estrutura do RUP - Rational Unified Process.



Fonte: Audy (2008)

Um dos principais conceitos é a noção de desenvolvimento iterativo, organizando projetos em termos de disciplinas e fases, cada uma consistindo em uma ou mais iterações. (AUDY, 2008)

2.5.1 Práticas

A perspectiva prática do RUP descreve seis boas práticas da engenharia de software que são recomendadas para o desenvolvimento de software, podendo ser chamadas como *templates*, elas ajudam na avaliação do progresso do projeto e ajudam a equipe de desenvolvimento a manterem-se concentradas no projeto. (SOMMERVILLE, 2011). São elas:

Desenvolver software iterativamente: Planejar os incrementos do software com base nas prioridades do cliente e desenvolver os recursos de alta prioridade no início do processo de desenvolvimento.

Gerenciar os requisitos: Documentar os requisitos do cliente e acompanhar as mudanças, analisando o impacto delas antes de aceitá-las.

Uso de arquitetura baseada em componentes: Estruturar a arquitetura do sistema em componentes.

Modelar o software visualmente: Usar modelos gráficos da UML apresentando visões estáticas e dinâmicas do software.

Verificação da qualidade do software: Assegurar que o software atende os padrões de qualidade da organização.

Controle de mudanças do software: Gerenciar as mudanças do software, usando um sistema de gerenciamento de mudanças, procedimentos e ferramentas de gerenciamento de configurações.

2.5.2 Fases

O RUP é um modelo constituído por quatro fases distintas no processo de desenvolvimento de software, diferentemente dos outros modelos estas fases estão diretamente relacionadas ao modelo de negócio do software. (SOMMERVILLE, 2011)

2.5.2.1 Concepção

A fase de concepção tem como objetivo estabelecer um *bussiness case* para o sistema, definem-se todas as entidades (pessoas e sistemas) que vão interagir com o sistema e quais serão suas interações, avaliando essas informações sob a perspectiva de contribuição relacionada ao negócio do sistema. (SOMMERVILLE, 2011)

Nesta fase são identificadas as necessidades de negócio do software, propondo uma arquitetura rudimentar para o software desenvolvendo um planejamento para a natureza iterativa e incremental do projeto. (PRESSMAN, 2011)

2.5.2.2 Elaboração

A fase de elaboração possui ênfase na arquitetura do software tendo como meta desenvolver uma compreensão do problema, estabelecendo um *framework* para a arquitetura do software, identificando o plano e os riscos do projeto. (SOMMERVILLE, 2011)

Esta fase envolve atividades de comunicação e modelagem do modelo de processo, ampliando a representação da arquitetura do software, incluindo cinco visões diferentes do software: modelo de caso prático, modelo de requisitos, modelo de projeto, modelo de implementação e modelo de emprego. (PRESSMAN, 2011)

2.5.2.3 Construção

A fase de construção envolve o projeto, programação e teste do software, ou seja, as partes do software são desenvolvidas em paralelo e integradas tendo no final desta etapa um software funcional bem como sua documentação associada e pronta para ser entregue ao usuário final. (SOMMERVILLE, 2011)

2.5.2.4 Transição

A fase de transição é ultima fase do RUP implica na transferência do software para o usuário final em seu funcionamento no ambiente operacional real, considerado uma das etapas mais caras e problemáticas. É nesta fase que o software deve estar documentado e funcionando corretamente no ambiente operacional do usuário. (SOMMERVILLE, 2011)

Durante esta fase monitora-se o uso do software pelo usuário disponibilizando suporte para o ambiente operacional podendo realizar e avaliar relatórios de defeitos (*bugs*) e solicitações de mudanças e melhorias no software. (PRESSMAN, 2011)

2.5.3 Atividades

Existem nove atividades no RUP, agrupadas em *workflows*, e elas representam uma divisão das atividades em grupos lógicos. A seguir são detalhados os *workflows*.

2.5.3.1 Modelagem de negócios

A modelagem de negócios tem como objetivo entender a estrutura, a dinâmica e os problemas da organização que fará o desenvolvimento do software, tendo como premissa a identificação de potenciais melhorias no processo para um melhor entendimento das necessidades e problemas que devem ser resolvidos pelo software. (AUDY, 2008)

Nesta etapa os processos de negócio são modelados por casos de uso de negócios, outros artefatos como o modelo de objeto de negócio, glossário de negócios, documento da arquitetura são criados nesta fase. (AUDY, 2008)

2.5.3.2 Requisitos

Os requisitos têm como meta estabelecer um acordo entre o cliente e o *stakeholders* em relação ao que o software deve fazer e por qual motivo, fornecendo a equipe de desenvolvedores um melhor entendimento dos requisitos do software, podendo definir um escopo do software e bases para estimativas de prazos e custos do software. (AUDY, 2008)

Nesta etapa os atores que interagem com o software são identificados e os casos de uso são desenvolvidos para a modelagem dos requisitos dos sistemas, outros artefatos são criados como a especificação dos requisitos do software, visão do software, e protótipos como o de interface. (AUDY, 2008)

2.5.3.3 Análise e Projeto

A análise e o projeto visa transformar os requisitos em uma especificação que descreve como o software deverá ser implementado, estabelecendo uma arquitetura e adaptação do projeto ao ambiente de desenvolvimento seguindo critérios de desempenho, escalabilidade e testabilidade. (AUDY, 2010)

Um modelo de projeto é criado e documentado com os modelos de arquitetura, modelo de componentes, modelo de objetos, modelo de sequência e modelo de dados.

2.5.3.4 Implementação

A implementação é o desenvolvimento do software em um ambiente definido, visa implementar o código em termos de subsistemas, organizado em camadas, implementar as classes em objetos em termos de componentes e testes de componentes individuais. (AUDY, 2010)

2.5.3.5 Teste

O teste tem como objetivo garantir que o software final é aquele solicitado pelos *stakeholders*, nesta etapa são verificadas as interações e integrações entre os componentes desenvolvidos a fim de identificar e assegurar que todos defeitos descobertos (*bugs*) foram corrigidos. (AUDY, 2008)

2.5.3.6 Implantação

A implantação tem como objetivo disponibilizar para o usuário final o produto final de software, distribuído e instalado, além de oferecer treinamento do software em seu ambiente final de operação, assim como migração de softwares ou bases de dados. (AUDY, 2008)

2.5.3.7 Gerência de configuração e alteração

Tem com meta estabelecer o controle das modificações e a manutenção da integridade do projeto, apoiando a gerencia de mudanças do software. (AUDY, 2008)

2.5.3.8 Gerência de projeto

Visa estabelecer objetivos, riscos e restrições para a entrega do software de acordo com as necessidades dos *stakeholders*, tendo como propósito o apoio ao gerenciamento do desenvolvimento do software fornecendo um conjunto de atividades para a gerência de projetos. (AUDY, 2008)

2.5.3.9 Ambiente

O ambiente tem como objetivo fornecer processos e ferramentas de aquisição, instalação, configuração de ferramentas, configuração e melhorias de processo para a equipe de desenvolvimento. (AUDY, 2008)

2.6 MODELOS ÁGEIS

Os modelos ágeis são métodos de desenvolvimento incremental nos quais os incrementos do software são pequenos e normalmente novas versões são entregues ao cliente rapidamente, envolvendo o cliente no processo de desenvolvimento a fim de obter feedbacks rápidos sobre a evolução dos requisitos. (SOMMERVILLE, 2011)

Na essência os modelos ágeis se desenvolveram para sanar fraquezas reais e perceptíveis da engenharia de software convencional, oferecendo benefícios importantes como sua habilidade em reduzir custos de mudanças ao longo de todo o processo de desenvolvimento de software. (PRESSMAN, 2011)

O desenvolvimento ágil é concebido para produzir rapidamente softwares úteis, sendo desenvolvido com uma série de incrementos onde cada incremento inclui uma nova funcionalidade do software. (SOMMERVILLE, 2011)

Ainda segundo Sommerville (2011), embora existam muitas abordagens para o desenvolvimento ágil, elas compartilham as mesmas características fundamentais:

- ✓ As etapas de especificação, projeto e implementação são intercaladas, não existindo uma especificação detalhada do software e sua documentação é mínima e a documentação de requisitos apenas define as características importantes do software.
- ✓ O software é desenvolvido em uma série de versões, onde os usuários finais e os *stakeholders* são envolvidos na especificação e avaliação de cada versão podendo propor alterações e novos requisitos no software que serão disponibilizadas em novas versões.

- ✓ As interfaces dos usuários são desenvolvidas de forma interativa permitindo criação rápida do projeto de interface por meio de desenhos e posicionamentos de ícones na interface.

Esses modelos são mais adequados ao desenvolvimento de softwares dos quais os requisitos mudam rapidamente durante o processo de desenvolvimento, destinando-se a entregar um software em funcionamento rapidamente ao cliente que em seguida pode propor alterações e novos requisitos a serem incluídos em novas versões, podendo assim diminuir a burocracia do processo de desenvolvimento. (SOMMERVILLE, 2011)

2.6.1 Manifesto Ágil

Em 2001 foi criada a *Aliança Ágil* onde assinara o *Manifesto Ágil* que tem como conceitos chaves os seguintes itens:

Indivíduos e interações acima de processos e ferramentas.

Software operacional acima de documentação completa.

Colaboração dos clientes acima de negociação contratual.

Respostas a mudanças acima de seguir um plano.

O *Manifesto Ágil* não rejeita os processos, ferramentas, documentação, contratos ou planejamento, mas ele simplesmente demonstra que eles possuem uma importância secundária quando comparadas com os indivíduos e interações, com o software operacional, com a colaboração dos clientes e as respostas rápidas a mudanças no contexto do processo de desenvolvimento de software. (PRESSMAN, 2011)

2.6.2 Princípios

A *Aliança Ágil* estabelece doze princípios para quem ter agilidade no desenvolvimento de software, são eles: (PRESSMAN, 2011)

1. A prioridade é satisfazer o cliente por meio de entrega adianta e continua do software.
2. As alterações no software são acolhidas e aproveitadas pelos processos ágeis como uma vantagem competitiva perante o cliente.
3. Entregar o software em funcionamento frequentemente em intervalos curtos.
4. As equipes da área comercial e do desenvolvimento devem trabalhar em conjunto ao longo de todo projeto.
5. Deve-se construir o projeto em torno de indivíduos motivados fornecendo ambiente e apoio necessário e confiando no trabalho feito por eles.
6. O método mais eficiente para transmitir informações para a equipe de desenvolvimento é uma conversa aberta e presencial.
7. Software em funcionamento é a principal medida de progresso.
8. Os processos ágeis promovem um desenvolvimento sustentável, portanto a equipe do desenvolvimento e o usuário devem estar capacitados para o ritmo constante indefinidamente.
9. Atenção continua na excelência técnica e no projeto aumenta a agilidade.
10. Simplicidade.
11. Melhores arquiteturas, requisitos e projetos emergem de equipes que tem auto-organização.
12. A equipe se avalia em intervalos regulares verificando como se tornar mais eficiente e sintonizando e ajustando seu comportamento para isto.

Nem todo modelo de processo ágil aplica esses doze princípios, alguns ignoram ou relevam a importância de um ou mais desses princípios. Entretanto os princípios definem um espírito ágil mantido em cada um dos modelos de processo. (PRESSMAN, 2011)

A seguir serão abordados dois modelos ágeis o *Extreme Programming (XP)* e o *Scrum*, que foram utilizados na empresa *case*.

2.6.3 Extreme Programming (XP)

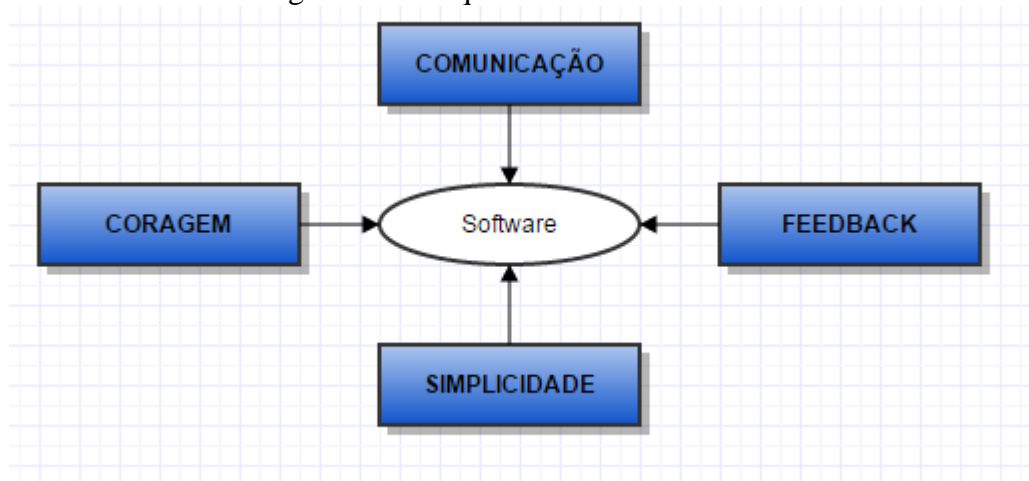
O *Extreme Programming* (XP) é uma abordagem amplamente utilizada para o desenvolvimento de software ágil que tem como objetivo impulsionar práticas reconhecidamente boas no processo de desenvolvimento de software como o desenvolvimento iterativo a níveis extremos. (SOMMERVILLE, 2011)

Assim, o *Extreme Programming* é uma proposta disciplinada para o desenvolvimento de software, visando entregar o software que o cliente deseja e quando ele desejar. Enfatizando o trabalho em uma única equipe formada por gerente, cliente e desenvolvedores que estão dedicados a entregar um software de qualidade. (AUDY, 2008)

Sendo assim o XP surgiu baseado em quatro dimensões onde o software podia melhorar (Figura 8), onde é necessário sempre pensar em melhorar a comunicação, a simplicidade sempre deve ser vista em primeiro plano, é necessário sempre obter feedback de como as atividades estão sendo executadas e sempre é necessário ter coragem. (AUDY, 2008).

Segundo Audy (2008), o *Extreme Programming* dá ênfase a quatro fatores; comunicação, simplicidade, *feedback* (melhorias) e coragem, conforme figura a seguir.

Figura 11 - As quatro dimensões do XP.



Fonte: Audy (2008).

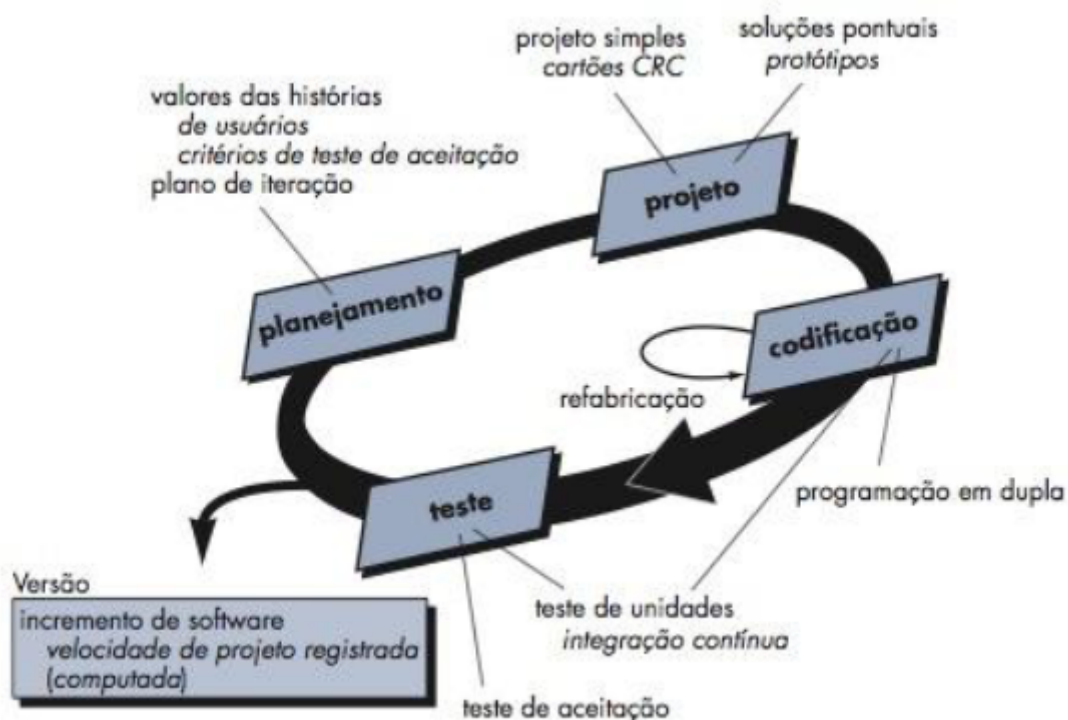
A dimensão da comunicação ocorre de forma contínua entre o cliente e a equipe de desenvolvimento, podendo assim o cliente está sempre acompanhando de perto o progresso do projeto. Com relação à simplicidade é produzido o menor numero de artefatos possíveis

que não estejam no código, mantendo sempre um código revisado e corrigido. Os feedbacks são recebidos com as avaliações das pequenas versões liberadas (*releases*) e a coragem significa que os desenvolvedores devem ser honestos sabendo o que pode ser feito e o que não pode ser feito. (AUDY, 2008)

2.6.3.1 Fases

A *Extreme Programming* envolve um conjunto de regras e práticas constantes no contexto de quatro fases ou atividades metodológicas (Figura 9), são elas planejamento, projeto, codificação e testes. (PRESSMAN, 2011)

Figura 12 - O processo Extreme Programming e suas fases.



Fonte: Pressman (2011)

A figura acima ilustra o processo *Extreme Programming* e suas fases pelo autor Pressman (2011), destacando alguns conceitos e tarefas-chaves associadas a cada uma das atividades metodológicas, a seguir veremos cada uma dessas fases.

2.6.3.1.1 *Planejamento*

A atividade de planejamento se inicia em levantar os requisitos para capacitar à equipe de desenvolvimento do em entender o ambiente de negócios do software e possibilitar que se consiga ter uma percepção ampla sobre os resultados solicitados, fatores principais e funcionalidade. (PRESSMAN, 2011)

2.6.3.1.2 *Projeto*

O Projeto *Extreme Programming* segue rigorosamente a simplicidade, sendo assim é preferível ter sempre um projeto simples do que uma representação completa, então o projeto oferece um guia de implementação para uma estória a medida que ela é desenvolvida. (PRESSMAN, 2011)

2.6.3.1.3 *Codificação*

Esta etapa ocorre depois de desenvolvida às estórias e o processo de elaboração do projeto ter sido concluído, a equipe passa então a desenvolver uma série de testes de unidades que exercitarão cada uma das estórias a serem inclusas em uma nova entrega (ou *release*).

O conceito-chave desta atividade é a programação em dupla. Duas pessoas trabalham juntas em uma mesma estação de trabalho fornecendo um mecanismo para resolução de problemas em tempo real, já que duas cabeças funcionam melhor do que uma, sendo assim o código é revisado à medida que for sendo criado oferecendo uma qualidade em tempo real. (PRESSMAN, 2011)

2.6.3.1.4 Testes

O *Extreme Programming* aborda dois testes.

1. Os testes de unidade, que são iniciados antes de começar a codificação e que devem ser implementados usando uma metodologia de automatização.
2. Os testes de aceitação, que são especificados pelo cliente e mantém o foco nas características e nas funcionalidades do software, sendo estas visíveis e revistas pelo cliente.

2.6.3.2 Princípios

Sommerville (2011) define cinco práticas do XP relacionadas às práticas dos modelos ágeis. São elas:

- ✓ O desenvolvimento incremental é sustentado por meio de frequentes *releases* do sistema, seus requisitos são baseados em histórias de clientes e usados para decidir a funcionalidade que deve ser incluída em um *release* do sistema.
- ✓ O envolvimento do cliente é sustentado por meio do engajamento contínuo com a equipe de desenvolvimento, participando do desenvolvimento e dos testes de aceitação do sistema.
- ✓ Pessoas e não processos, isto é sustentado pela programação em pares, propriedade coletiva do código do software e um processo de desenvolvimento sustentável que não envolve horas excessivas de trabalho.
- ✓ As mudanças são aceitas por meio de *releases* contínuos para o cliente.
- ✓ A manutenção da simplicidade é feita por meio de refatoração constante que melhora a qualidade do código.

2.6.3.3 Práticas

Segundo Sommerville (2011), o XP envolve uma série de práticas que refletem os princípios dos métodos ágeis, que são:

Quadro 6 - As práticas do Extreme Programming.

PRÁTICAS	DESCRIÇÃO
Planejamento incremental	Os requisitos são definidos em histórias que são incluídas em <i>releases</i> que são determinadas pelo tempo disponíveis e sua relativa prioridade.
<i>Releases</i>	É desenvolvido um conjunto mínimo funcionalidades úteis que fornecem um valor de negócio, as <i>releases</i> são frequentes e gradualmente adicionam funcionalidades à primeira <i>release</i> .
Projeto simples	Cada projeto é realizado para atender as necessidades atuais e nada mais.
Desenvolvimento <i>test-first</i>	Um <i>framework</i> de testes iniciais automatizados é usado para elaborar os testes para cada nova funcionalidade antes de ela ser implementada.
Refatoração	Todos os desenvolvedores devem refatorar o código continuamente assim que encontrar melhorias, assim mantendo o código simples e manutenível.
Programação em pares	Os desenvolvedores trabalham em pares, verificando o trabalho dos outros e prestando apoio para um bom trabalho simples.
Propriedade coletiva	Os pares desenvolvedores trabalham em todas as áreas do software, todos detêm o mesmo conhecimento do software e assumem responsabilidade por todo o código.
Integração contínua	Assim que uma tarefa é concluída ela é integrada ao sistema como um todo e após esta integração, todos os testes de unidade do sistema devem passar.
Ritmo sustentável	Grande quantidade de horas extras não é aceitável, pois o resultado muitas vezes é redução da qualidade do código e da produtividade em médio prazo.
Cliente no local	Um representante do cliente deve estar disponível todo o tempo para a equipe de desenvolvimento, sendo ele um membro da equipe de desenvolvimento e responsável por levar a ela os requisitos de sistema para a implementação.

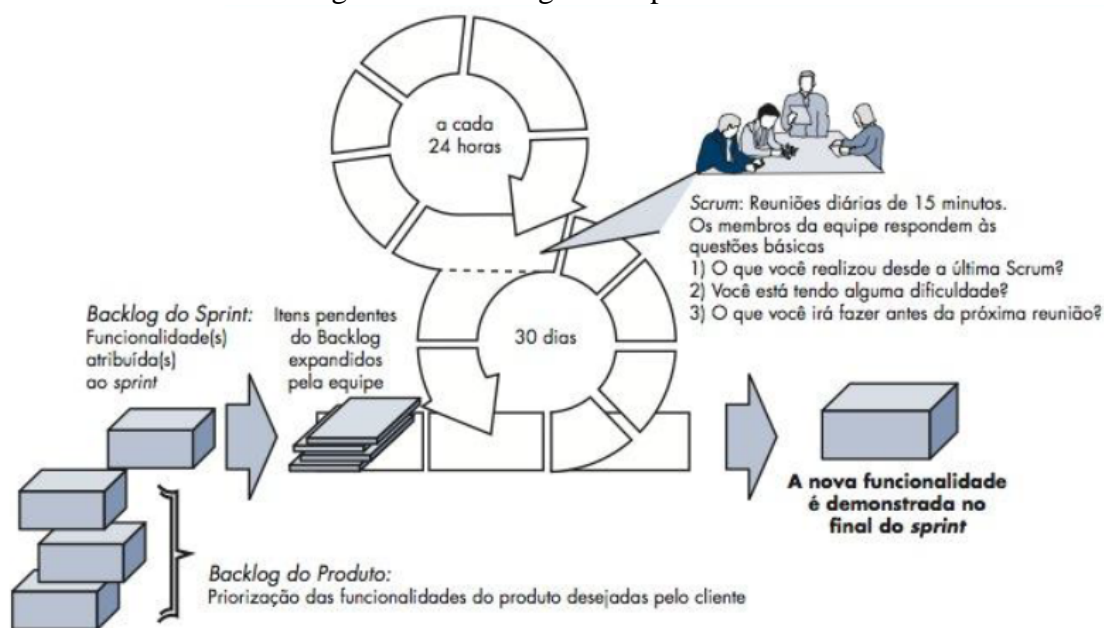
Fonte: Sommerville (2011).

2.6.4 Scrum

O Scrum é um método de desenvolvimento ágil que engloba um conjunto de padrões de processos enfatizando prioridades de projeto, unidades de trabalho compartmentalizadas comunicação e feedback frequente dos clientes. Seus princípios são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento de um processo que incorpora os requisitos, análise, projeto, evolução e entrega. (PRESSMAN, 2011)

Em cada atividade metodológica, ocorrem tarefas a realizar dentro de um padrão de processo chamado *sprint*, sua quantidade varia de acordo com a complexidade do software ele é adaptado e definido ao problema em questão e modificado em tempo real pela equipe do Scrum. (PRESSMAN, 2011)

Figura 13 - Fluxo geral do processo Scrum.



Fonte: Pressman (2011).

Segundo Pressman (2011), Schawaber(2013) e Sutherland (2013) o Scrum enfatiza o uso de um conjunto de padrões de processo de software que são eficazes a projetos com prazos de entrega apertados, requisitos mutáveis e críticos de negócio. Cada um desses padrões define um conjunto de ações de desenvolvimento, são eles:

Backlog: Uma lista com a prioridade dos requisitos ou funcionalidades do projeto que fornecem um valor comercial ao cliente.

Sprints: Consistem em unidades de trabalho solicitadas para atingir um requisito estabelecido no *backlog* e que precisa ser ajustado dentro de um prazo já fechado.

Reuniões Scrum: São reuniões curtas realizadas diariamente pela equipe, onde são feitas perguntas sobre o que foi feito desde a última reunião, se encontrou obstáculos e o que planeja fazer até a próxima reunião. Essas perguntas são feitas e respondidas pelos membros da equipe de desenvolvimento, geralmente quem conduz a reunião é o líder da equipe chamado de *Scrum master*, que avalia as respostas de cada integrante.

Demo: Entregar de uma versão do software ao cliente para que a funcionalidade desenvolvida possa ser demonstrada e avaliada pelo cliente.

Estes padrões de processos do Scrum capacitam uma equipe de desenvolvimento de software a trabalhar com sucesso em um mundo onde a eliminação da incerteza é impossível ajudando a revelar problemas em potencial mais cedo e entregando o software dentro do prazo estipulado. (PRESSMAN, 2011)

2.7 QUALIDADE DE SOFTWARE

A qualidade de software está relacionada a entregar ao cliente um software que satisfaça suas expectativas dentro daquilo que foi acordado nos requisitos do projeto, neste contexto a qualidade de software é uma área da engenharia de software que objetiva a garantir esta qualidade pelas definições do processo de desenvolvimento do software. (ENGHOLM, 2010)

No sentido mais geral podemos definir a qualidade de software como uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam. (PRESSMAN, 2011)

O software possui qualidade quando este está adequado à empresa, ao cliente, atendendo padrões de qualidade predefinidos e quando o software estiver pronto o mesmo terá

qualidade se gerar informações com qualidade, adequada, útil, precisa, confiável, clara e oportuna. (REZENDE, 2005)

A qualidade de software contempla uma série de objetivos no desenvolvimento de um software, tais como, requisitos não funcionais, extensibilidade, capacidade de manutenção, reutilização de código, estabilidade, desempenho, escalabilidade usabilidade e confiabilidade. (ENGHOLM, 2010)

Pressman (2011) define três pontos chaves na qualidade de software, são eles:

1. Uma gestão de qualidade efetiva estabelece a infraestrutura que fornece suporte a qualquer construção de um produto de software de alta qualidade. Aspectos administrativos do processo criam mecanismos de controle evitando o caos nos projetos, práticas da engenharia de software permitem desenvolver uma solução de software consistente.
2. Um produto útil fornece conteúdo, funções, confiabilidade, isenção de erros e funcionalidades e recursos que o cliente deseja, sempre satisfazendo as exigências definidas explicitamente pelos interessados no software. Isto é o que se espera de todo o software de alta qualidade.
3. Ao agregar valor tanto para o fabricante como para o cliente do software, um software de alta qualidade gera benefícios para a empresa de software bem como para a comunidade e usuário final. A empresa em específico, ganha um alto valor agregado exigindo menos manutenção, menos correções de erros e menos suporte ao cliente, permitindo que seus desenvolvedores despendam mais tempo criando novas soluções de software.

Ainda Pressman (2011), define que a qualidade de software é atingida pela aplicação de métodos de engenharia de software, práticas administrativas consistentes e controle de qualidade, todos estes, suportados por uma infraestrutura de garantia de qualidade de software.

Segundo a norma ISO 9000 e citada por Engholm (2010), qualidade pode ser definida como:

Qualidade é o grau em que um conjunto de características inerentes a um produto, processo ou sistema cumpre os requisitos inicialmente estipulados para estes, ponde ser vista como a conformidade aos requisitos do projeto.

O *SWBOK (Software Engennering Body of Knowledge)* apresenta em uma de suas áreas de conhecimento a qualidade sendo obtida por meio dos seguintes itens. (ENGHOLM, 2010)

Quadro 7 - Áreas de conhecimento do SWBOK.

FUNDAMENTOS	PROCESSOS	CONSIDERAÇÕES PRÁTICAS
Cultura, ética, valores, custos, modelos e características da qualidade.	Garantia de qualidade, verificação, validação, revisões e auditorias.	Aplicação dos requisitos de qualidade, caracterização de defeitos, técnicas de gerenciamento e medidas da qualidade de software.

Fonte: Engholm (2010).

2.7.1 Dimensões de qualidade

Pressman (2011) entende que as oito dimensões de Garvin (1987) podem ser aplicadas quando se considera qualidade de software.

Quadro 8 - Dimensões de qualidade.

DIMENSÃO	DESCRIÇÃO
Qualidade do desempenho	O software fornece todo conteúdo, funções e recursos que são especificados como parte do modelo de requisitos do software.
Qualidade dos recursos	O software fornece recursos que surpreendam o usuário final que o utilizam pela primeira vez.
Confiabilidade	O software fornece todos os recursos e capacidade sem falhas e esta disponível quando necessário.
Conformidade	O software está de acordo com os padrões de software locais e externos relacionados com a aplicação, seguindo convenções de projeto e codificação.
Durabilidade	O software pode ser mantido (modificado) ou corrigido (depurado) sem a geração involuntária de efeitos colaterais indesejáveis.
Facilidade de Manutenção	O software pode ser mantido (modificado) ou corrigido (depurado) em um período de tempo aceitável e curto. O pessoal de suporte obtém todas as informações necessárias para realizar alterações e correções de erros.
Percepção	Situações que pode ter algum preconceito ao software influenciando a percepção de qualidade.

Fonte: Pressman (2011)

Pressman (2011) entende que muitas dessas dimensões podem ser consideradas apenas subjetivamente, por tal razão também precisamos de um conjunto de fatores de qualidade subdividido em duas categorias; fatores medidos diretamente e indiretamente.

2.7.2 Fatores de qualidade

Os fatores de qualidade de software criados por McCall, Richards e Walter, citados por Pressman (2011) focam em três aspectos de um software, as características operacionais, habilidade de suportar mudanças e a adaptabilidade a novos ambientes.

Figura 14 - Fatores de qualidade de software.



Fonte: Pressman (2011)

Referindo-se aos fatores citados na Figura 14, veremos as descrições de cada um desses fatores apontadas por Pressman (2011).

Quadro 9 - Fatores de qualidade de software.

FATORES	DESCRIÇÃO
Correção	O quanto o software satisfaz sua especificação e atende aos objetivos do cliente.
Confiabilidade	O quanto se pode esperar que o software realize a função pretendida com precisão exigida.
Eficiência	A quantidade de recursos computacionais e códigos exigidos para o software desempenhar sua função
Integridade	O quanto o acesso ao software ou dados por pessoas não autorizadas pode ser controlado.
Usabilidade	Esforço necessário para aprender a operar o software.
Facilidade de Manutenção	Esforço necessário para localizar e corrigir um erro no software.
Flexibilidade	Esforço necessário para modificar o software em operação
Testabilidade	Esforço necessário para testar um software a modo de garantir que ele desempenhe sua função destinada.
Portabilidade	Esforço necessário para migrar o software de um ambiente de hardware ou software para outro.
Reusabilidade	O quanto o software pode ser reutilizado em outras aplicações.
Interoperabilidade	Esforço necessário para integrar o software a outro sistema.

Fonte: Pressman (2011)

2.7.3 Gerenciamento de qualidade

Com a finalidade de desenvolver um software de alta qualidade precisa-se definir uma série de modelos, padrões, técnicas e procedimentos que contribuam para atingir a este objetivo durante o processo de desenvolvimento de software. (ENGHOLM, 2010)

Os processos de gerencia de qualidade abrangem todos os aspectos do desenvolvimento de software, tais como ferramentas, controle de versão, linguagens, metodologias de revisão, técnicas organizacionais e administração de pessoas etc. (KOSCIANSKI, 2007; SOARES, 2007)

Sommerville (2011) defende que o gerenciamento de qualidade de software possui três principais preocupações:

- ✓ No nível organizacional, o gerenciamento de qualidade esta preocupado com estabelecimento de um *framework* de processos organizacionais e padrões que levem a um software de alta qualidade. Levando equipes de qualidade a definirem processos de desenvolvimento de software e padrões a serem utilizados no software.
- ✓ No nível de projeto, o gerenciamento da qualidade envolve a aplicação de processos de qualidade verificando que os processos planejados foram seguidos e que as saídas do projeto estejam em conformidade com os padrões aplicados no projeto.
- ✓ Ainda no nível de projeto, o gerenciamento de qualidade se preocupa com o estabelecimento de um plano de qualidade, este que deve definir as metas de qualidade, os processos e padrões que devem ser utilizados.

Este gerenciamento visa assegurar que o software possua o menor número possível de defeitos e que se adeque aos atributos e fatores de qualidade incluindo a definições de padrões para o processo de desenvolvimento do software o estabelecimento de processos para verificar se tais padrões estão sendo seguidos. (SOMMERVILLE, 2011)

2.7.4 Garantia de qualidade

A garantia de qualidade estabelece uma infraestrutura para suporta métodos da engenharia de software, gerenciamento de projeto, ações de controle de qualidade e processos de auditoria e relatórios para uma avaliação do controle de qualidade, todos estes fundamentais para o processo de desenvolvimento de software. (PRESSMAN, 2011)

Esta etapa tem como propósito de assegurar que os objetivos planejados no início do projeto serão cumpridos, estabelecendo sistema de controle para todo o processo de desenvolvimento de software visando garantir que o software fabricado fará aquilo que se espera. (KOSCIANSKI, 2007; SOARES, 2007)

A garantia de qualidade de software (*Software Quality Assurance – SQA*) tem como objetivo atingir padrões de qualidade aceitáveis e deve ser considerada em todo ciclo de desenvolvimento do software. Possui a finalidade de prevenir os defeitos de software, certificar que eles foram encontrados nos planos de testes e devidamente corrigidos além de auditar os trabalhos com procedimentos previamente estabelecidos. (ENGHOLM, 2010)

2.7.5 Controle de qualidade

O controle de qualidade possui um conjunto de ações da engenharia de software de tal maneira que visa garantir que o software produzido atinja as metas de qualidade. (PRESSMAN, 2011)

É muito comum existir confusão entre os conceitos relacionados a garantia de qualidade e controle de qualidade, com a finalidade de esclarecer isto, Engholm (2010) relaciona as características das duas disciplinas e suas diferenças no quadro 10.

Quadro 10 - Principais características da garantia e controle de qualidade.

CONTROLE DE QUALIDADE	GARANTIA DE QUALIDADE
Visa garantir que os resultados estão em conformidade com as especificações.	Visa garantir que existe processo definido e corretamente utilizado.
É focado na descoberta de defeitos em itens de configuração de software.	É focado em monitoração e melhoria de processo.
É orientado a produto.	Utiliza metodologia e padrões de desenvolvimento.
É orientada a detecção de erros.	É orientado a processos.
Garante atendimento aos requisitos.	É orientada a prevenção.
É executado no final das fases do ciclo de desenvolvimento.	É executada no início das fases do ciclo vida de desenvolvimento de software.

Fonte: Engholm (2010).

2.7.6 Normas

As normas internacionais de qualidade são criadas com base no trabalho de especialista do mundo todo, elas tornam-se base para especificar produtos, serviços ou mesmo para elaborar legislação em vários países. (KOSCIANSKI, 2007; SOARES, 2007)

Ainda segundo Koscianski (2007) e Soares (2007), as normas em sua imensa maioria as normas tem um caráter mais informativo do que regulatório, enquanto existem normas a serem respeitadas sob pena de lei para instalação elétrica ou hidráulica em um prédio isso não acontece com o software.

A seguir no quadro 11 são apresentadas algumas das normas mais conhecidas.

Quadro 11 - Normas de Software.

NORMAS	PROPÓSITO
ISO 12207	Processos de ciclo de vida de software.
ISO/IEC 12119:1994	Pacotes de software – Requisitos de qualidade e testes.
ISO/IE 14598-1:1999	Avaliação de qualidade de produtos de software.
ISO/IEC 25000:2005	Modelo de qualidade de software, nova versão.
ISO 9241:1998	Ergonomia de software.
ISO/IEC 20926:2003	Medida de software por pontos de função.
ISO 90000-3:2004	Diretivas para aplicação da ISO 9001 ao software.
ISSO 9001:2000	Requisitos para sistemas de gerenciamento de qualidade (aplicável a qualquer empresa de software ou não).

Fonte: Koscianski (2007) e Soares (2007).

Existem umas séries de normas que podem ser utilizadas no desenvolvimento de sistema de gerenciamento de qualidade em todos os setores das organizações, porém a ISO 9001, a mais geral desses padrões aplica-se a organizações que projetam, desenvolvem e mantém produtos incluindo o software. (SOMMERVILLE, 2011)

2.7.6.1 ISO 9001

A norma não é propriamente um padrão para o desenvolvimento de software, mas um framework para o desenvolvimento de padrões de software, definindo os princípios gerais de qualidade, descrevendo os processos gerais de qualidade e estabelecendo padrões organizacionais e procedimentos que devem ser definidos. (SOMMERVILLE, 2007)

As necessidades abordadas pela norma são responsabilidade administrativa, um sistema de qualidade, revisão contratada, controle de projeto, controle de dados e documentações, identificação e rastreabilidade de produtos, controle de processos, inspeções e testes, ações preventivas e corretivas, registros de controle de qualidade, auditorias de qualidade internas, treinamento, manutenção e técnicas estatísticas. (PRESSMAN, 2011)

Ainda segundo Pressman (2011) para a organização de software seja certificada da ISO 9001 tem de estabelecer políticas e procedimentos para atender a cada uma dessas

necessidades e depois ser capaz de provar que tais políticas e procedimentos estão sendo seguidos.

2.7.6.2 ISO/IEC 12207

A norma cobre todo o ciclo de vida do software de requisitos até a manutenção e retirada de uso de um software, provendo uma estrutura para que a organização defina seus processos oferecendo como propósito a definição de um linguajar comum aos métodos, técnicas, modelos e normas que tratam a qualidade. (KOSCIANSKI, 2007; SOARES, 2007)

Segundo Koscianski (2007) e Soares (2007) os processos da norma são classificados em três categorias:

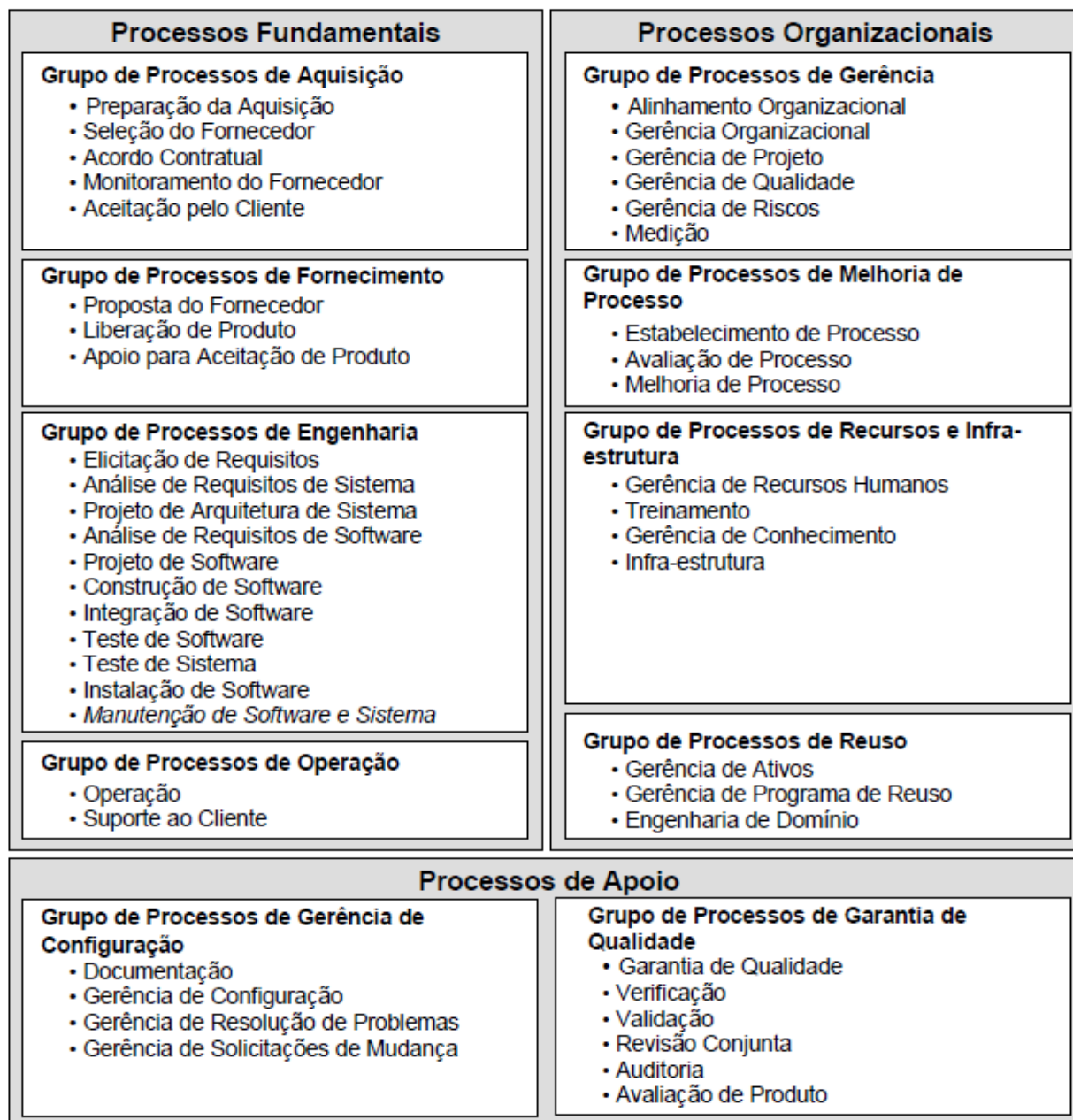
Primários: São os processos básicos que se relacionam aos produtos de softwares.

De apoio: O processo desta categoria tem lugar depois que um processo primário é iniciado.

Organizacional: São os processos relacionados à operação da organização em si.

A norma provê um conjunto de processos da engenharia de software que a organização deve utilizar para o desenvolvimento de software, porém ela não representa um modelo fixo ao qual a organização que adote se submete, mas funciona mais como uma estrutura de apoio onde a organização que adotar procede com as adaptações nas recomendações para sua realidade. (PADUELLI, 2007)

Figura 15 - Processos do ciclo de vida da norma ISO/IEC 12207.



Fonte: Paduelli (2007).

2.8 MODELOS DE QUALIDADE DE SOFTWARE

Um modelo de qualidade de software proporciona uma indicação geral de qualidade de processos exibida pela organização de software, proporcionando uma sensação de qualidade sobre ao processo de desenvolvimento de software. (PRESSMAN, 2011)

Este modelo serve como uma estrutura para melhorar o processo de software definindo as características que devem estar presentes para se atingir um processo eficaz e um método de avaliação que ajude a determinar se aquelas características estão presentes na organização de software na implementação destas características de processo. (PRESSMAN, 2011)

Atualmente na área de desenvolvimento de software tem se destacado em muito a adoção de modelos de qualidade de software internacionalmente aceitos, tais como a ISO ou CMMI chamando a atenção para a adoção de padrões visando reduzir os problemas existentes na área de desenvolvimento de software. (AUDY, 2008)

Muitas das organizações têm buscado modelos de verificação e reconhecimentos de níveis de maturidade nos seus processos de desenvolvimento de software, isso se deve a necessidade de elas ter um mínimo de garantia sobre a qualidade dos seus processos. (AUDY, 2008)

Entre esses modelos pode-se citar o CMMI, que será apresentado a seguir, ou o MPS.BR (Koscianski, 2007; Soares, 2007), descrito em um outro capítulo.

2.8.1 CMMI

A CMMI (*Capability Maturity Model Integration*) foi criada pelo SEI (*Software Engeneering Institute*) representa um metamodelo de processo abrangente, qualificado em uma série de capacidades de sistemas e engenharia de software que devem estar presentes a medida que a organização alcança os diferentes níveis de capacidade e maturidade do processo. (PRESSMAN, 2011)

Seu objetivo é servir como guia para a melhoria nos processos de uma organização e também na habilidade dos profissionais em gerenciar o desenvolvimento, aquisição e manutenção de produtos, softwares ou serviços. (KOSCIANSKI, 2007; SOARES, 2007)

Koscianski (2007) e Soares (2007) definem alguns termos importantes do CMMI.

Áreas de processo: É um conjunto de práticas que quando executadas coletivamente satisfazem um conjunto de objetivos que é importante para obter uma melhora significativa desta área.

Objetivos específicos: Identificam características únicas, descrevendo o que deve ser implementado para ser aplicado em uma área satisfazendo-a.

Práticas específicas: São atividades importantes para atingir um determinado objetivo específico, sendo essas práticas associada a um nível de maturidade.

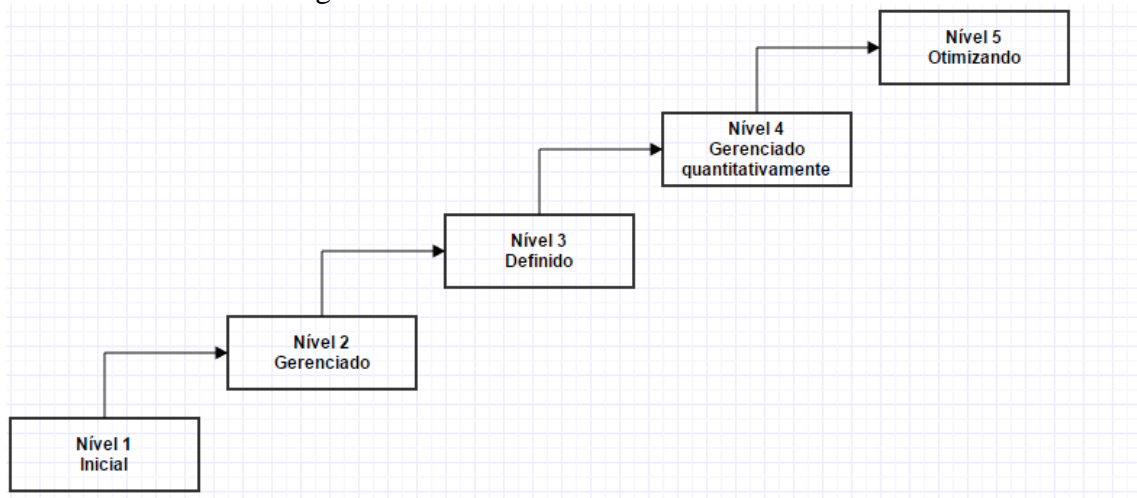
Objetivos genéricos: Cada nível de maturidade possui apenas um objetivo genérico que descreve o que uma organização deve fazer para atingir um nível determinado.

Práticas genéricas: Asseguram que os processos associados com as áreas de processo serão efetivos e repetíveis.

2.8.1.1 Níveis de maturidade

O modelo CMMI envolve examinar os processos de uma determinada organização e classificar esses processos ou áreas em uma escala de seis níveis relacionada ao nível de capacidade de cada área ou processo. (SOMMERVILLE, 2011)

Figura 16 - Níveis de maturidade do CMMI.



Fonte: Sommerville (2007).

Sommerville (2011) define que a idéia inicial é quanto mais maduro for o processo melhor, sendo esta escala de seis pontos atribui níveis de capacidade para cada área do processo da seguinte forma:

1. **Incompleto:** Pelo menos uma das metas específicas associadas com a área do processo não está satisfeita.
2. **Executado:** As metas associadas com a área do processo são satisfeitas e para todos os processos o escopo do trabalho a ser realizado é explicitamente definido.
3. **Gerenciado:** Nesse nível as metas associadas com a área de processo são cumpridas e as políticas organizacionais definem quando cada processo deve ser usado.
4. **Definido:** Esse nível concentra-se na padronização e implementação de processos organizacionais.
5. **Gerenciado quantitativamente:** Nesse nível, existe uma responsabilidade organizacional de usar os métodos estatísticos e outros métodos quantitativos para controlar os subprocessos.
6. **Otimizando:** Nesse nível mais alto, a organização deve usar as medições de processo e produto para dirigir a melhoria de processos.

A CMMI define área de processos em termos de metas específicas e as práticas específicas necessárias para atingir as metas. Estas metas estabelecem características que devem existir nas atividades envolvidas por uma área de processos e as práticas refinam uma

meta transformado-a em uma série de atividades relacionadas ao processo. Então para atingir um nível de maturidade as metas específicas e as práticas associadas com uma série de áreas de processo devem ser atingidas. (PRESSMAN, 2011)

Quadro 12 - Áreas de processo necessárias para atingir um nível de maturidade.

Nível	Foco	Áreas de Processo
Otimizando	Melhoria contínua do processo	Inovação organizacional e distribuição (deployment) Análise causal e resolução
Gerenciado quantitativamente	Gerenciamento quantitativo	Desempenho de processo organizacional Gerenciamento quantitativo de projeto
Definido	Padronização de processo	Desenvolvimento de requisitos Solução técnica Integração de produto Verificação Validação Foco no processo organizacional Definição de processo organizacional Treinamento organizacional Gerenciamento de projeto integrado Gerenciamento de fornecimento integrado Análise de decisão e resolução Ambiente organizacional para integração Equipe integrada
Gerenciado	Gerenciamento básico de projeto	Gerenciamento de requisitos Planejamento de projetos Monitoração e controle de projeto Gerenciamento de acordo com fornecedor Medição e análise Garantia de qualidade de processo e produto Gerenciamento de configuração
Executado		

Fonte: Pressman (2011).

3. MANUTENÇÃO DE SOFTWARE

A engenharia de software surgiu para resolver os problemas no processo de desenvolvimento de software com uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento e manutenção de software. Porém de fato muitos dos problemas ligados ao desenvolvimento e manutenção de software continuam sem solução, apesar dos conceitos relacionados à engenharia de software terem evoluído nos últimos tempos. (PADUELLI, 2007)

De um modo geral, a engenharia de software oferece algumas soluções para os desafios da manutenção de software, muita pesquisa tem sido feita atualmente para tentar entender os defeitos da má manutenção de software e a busca para soluções compatíveis para essa crescente demanda de projetos de manutenção de software. (AUDY, 2008)

Entender a manutenção de software, sua abrangência e significado são um passo fundamental para o aprofundamento das soluções para os problemas que são oriundos desta atividade. (PADUELLI, 2007)

Neste capítulo abordaremos as definições de manutenção de software, seus aspectos históricos, custos, os tipos de manutenção de software, software legado e o gerenciamento da manutenção de software.

3.1 DEFINIÇÕES

A manutenção de software é um processo geral de mudanças em um sistema que já foi entregue e liberado para uso, essas alterações feitas no software podem ser mudanças para correção de erros de codificação (*bugs*), correções de erros de projeto, melhorias significativas para corrigir erros de especificação ou atender a novos requisitos. Estas mudanças podem ser feitas nos componentes já existentes do sistema ou por meio da adição de novos componentes. (SOMMERVILLE, 2011).

O conceito de manutenção de software é antigo e extremamente importante e que muitas vezes não tem sua devida importância reconhecida, sua definição é muitos mais que consertar defeitos de software. (AUDY, 2008)

Todavia Braude (2004) entende que a manutenção se caracteriza por um trabalho realizado no software que ocorre depois de ter sido entregue, podendo ser correção de defeitos (*bugs*) que são inconsistências com os documentos de requisitos, erros de lógica ou de projeto e melhorias como novas capacidades ou aprimoramentos no software.

As atividades de manutenção são caracterizadas por intervenções e alterações nos software a fim de evitar sua deterioração, pois o software não se desgasta como o hardware ou equipamentos de informática, mas sim no sentido dos seus objetivos e funcionalidades cada vez menos se adequando ao ambiente externo e ao usuário final. (PRESSMAN, 2011).

Segundo Pfleeger (2001) citado por Paduelli (2007), embora essas definições tratem genericamente qualquer software de forma geral, existem diferenças entre a manutenção de software com propósitos distintos que podem ser estabelecidas em três categorias.

Quadro 13 - Classificação de softwares na manutenção de software.

CLASSIFICAÇÃO	DESCRIÇÃO
Primeira	Representa aqueles softwares construídos com base em uma especificação rígida e bem definida cujos resultados esperados são bem conhecidos.
Segunda	Os softwares que constituem implementações de soluções aproximadas do mundo real. Sua técnica consiste em desenvolver um tipo de solução que se baseia em descrever o problema de forma abstrata e então definir os seus requisitos a partir desta abstração.
Terceira	Consideram-se mudanças no ambiente de software onde ele vai ser utilizado, algumas características não existentes nas categorias anteriores.

Fonte: Paduelli (2007).

Mesmo considerando esta classificação dos softwares, o processo de manutenção durante sua execução possui diferentes partes que precisam interagir de forma que os objetivos da manutenção sejam entendidos e os resultados esperados alcançados. (PADUELLI, 2007)

De acordo com Paduelli (2007) essas partes são representadas como:

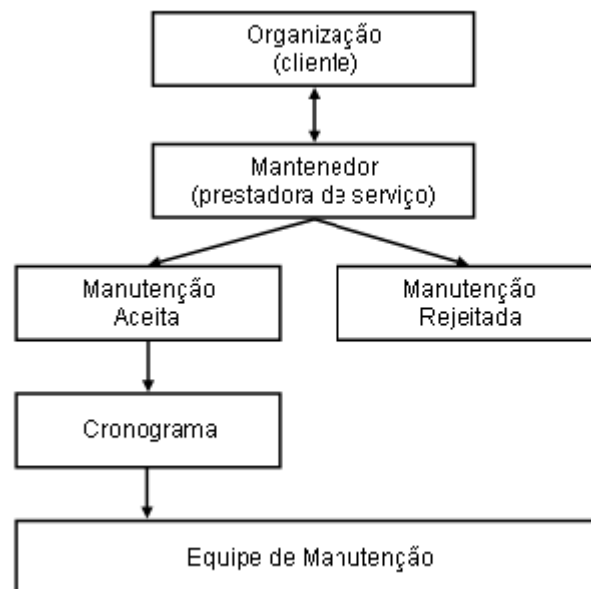
Organização Cliente: Essa organização corresponde ao adquirente de software, conforme definido na norma ISO/IEC 12207, ou seja, a organização possui o software e requisita um serviço de manutenção.

Mantenedor: Trata-se da organização que irá fazer o serviço de manutenção de software.

Usuário: Representa a organização que utiliza o software em questão.

Paduelli (2007) cita um relacionamento existente entre essas partes e a constituição de um fluxo para os pedidos de manutenção conforme figura abaixo.

Figura 17 - Fluxo dos envolvidos na manutenção de software.



Fonte: Paduelli (2007)

Podemos em geral concluir que todo software sofre manutenções, sejam elas simples ajustes, melhorias ou correções de erros, o ciclo genérico da manutenção de software contempla essas três formas mais utilizadas nas interferências de software. (REZENDES, 2005)

3.2 ASPECTOS HISTÓRICOS

Muitos dos softwares dos quais dependemos hoje tem em média de 10 a 15 anos, mesmo quando esses programas foram criados usando as melhores técnicas de projeto e codificação conhecidas na época ainda são passíveis de terem erros. (PRESSMAN, 2011)

Edward Yourdon (1992) citado por Paduelli (2007) considerou que a manutenção de software seria um dos problemas mais relevantes no futuro, previsões estas, feitas com base nas constatações da época em que as organizações adotavam a políticas de entregar a tempo o software para o cliente não se preocupando com questões de *manutenibilidade* e tendo uma postura de atacar os problemas do software somente quando eles emergissem.

Ainda Yourdon (1992) citado por Paduelli (2007) realizou um estudo na área de manutenção de software que colheu resultados de uma pesquisa desenvolvida em duas fases.

Na primeira fase foram levantadas as características das atividades de manutenção de software realizadas por 69 organizações. Na segunda foi realizado com 487 organizações dentre elas instituições governamentais com o intuito de validar os resultados obtidos na fase anterior buscando uma compreensão mais profunda e abrangente.

Quadro 14 - Pesquisa das características das atividades de manutenção de software.

FASE	CONCLUSÃO
1ª Fase	<p>A manutenção de software existente consome uma média de 48% das horas anuais do pessoal de desenvolvimento.</p> <p>Aproximadamente 60% dos esforços de manutenção são dedicados à manutenção do tipo perfectiva e dentro desse percentual dois terços são referente aos esforços de melhoria de software.</p> <p>Problemas de natureza gerencial em manutenção foram vistos como mais importantes do que aqueles de natureza técnica.</p>
2ª Fase	<p>Mais de um terço dos softwares tinham sua manutenção realizada por apenas uma pessoa.</p> <p>A maioria dos profissionais alocados para manutenção tinha outras tarefas ao mesmo tempo.</p> <p>Do total de manutenções, cerca de 20% representavam manutenções corretivas, 25% adaptativas e 50% as perfectivas.</p> <p>Quanto mais velho era o software, maior era o esforço que deveria ser empregado em sua manutenção e mais sujeita a organização estava em perder o conhecimento em torno deste software devido à rotatividade dos profissionais.</p>

Fonte: Paduelli (2007)

Paduelli (2007) cita que os autores chegaram aos principais problemas de manutenção enfrentados pelas organizações. São eles:

- ✓ Baixa qualidade da documentação dos softwares.
- ✓ Necessidade constante dos usuários por melhorias e novas funcionalidades.
- ✓ Falta de uma equipe de manutenção.
- ✓ Falta de comprometimentos com cronogramas.

- ✓ Treinamento inadequado do pessoal da manutenção.
- ✓ Rotatividade dos profissionais.

Pressman (2011) entende que o problema que resulta na manutenção são softwares mal projetados, mal codificados, mal especificados e mal documentados em relação aos sistemas de software que aplicam as metodologias da engenharia de software.

3.3 TIPOS DE MANUTENÇÃO DE SOFTWARE

As ações ligadas à atividade de manutenção de software são classificadas em *corretiva*, *adaptativas* e *evolutivas*. Essas categorias geralmente são reconhecidas por diversos autores tais como, Sommerville (2011), Rezende (2005), Braude (2004), Schach (2010), Audy (2008) e Pressman (2011).

Entretanto alguns autores como no caso do Pressman (2011) e a Norma ISO/IEC 12207 (IEEE, 1998) defendem uma quarta categoria que se chama *preventiva*. A seguir iremos abordar estas quatro categorias e como esses autores as definem.

Neste trabalho será considerada a classificação de tipos de manutenção defendida por Sommerville (2011), Pressman (2011) e Paduelli (2007).

3.3.1 Manutenção corretiva

Neste tipo de manutenção são feitas no software as correções que visam corrigir defeitos de funcionalidade, erros de projeto, erros de requisitos funcionais que muitas vezes implicam acertos emergenciais para deixar o software operante. (PADUELLI, 2007)

Sommerville (2011) entende que os erros de codificação são relativamente baratos para serem corrigidos, porém, erros de projeto e erros de requisitos são os mais caros e podem implicar em reescrever vários componentes do software, assim como pode ser necessário um reprojeito do software.

3.3.2 Manutenção adaptativa

Este tipo de manutenção é necessário quando algum aspecto do ambiente externo do software precisa sofrer mudanças, como banco de dados, sistemas operacionais e hardware. Neste caso o sistema como um todo deve se adaptar a essas mudanças de ambiente. (SOMMERVILLE, 2011)

3.3.3 Manutenção evolutiva

A manutenção evolutiva tem como objetivo acrescentar novos recursos ao software, tais como, novas funcionalidades ou novos requisitos, geralmente em razão de demanda vindas do usuário final do sistema. (PADUELLI, 2007)

Este tipo de manutenção é necessário quando os requisitos do software mudam em relação às mudanças organizacionais ou as regras de negócios. Geralmente neste caso as mudanças necessárias para o software são frequentemente muito maiores que os outros tipos de manutenção. (SOMMERVILLE, 2011)

3.3.4 Manutenção preventiva

Este tipo de manutenção é feita quando se modifica o software a fim de corrigir falhas latentes, antes que elas se tornem falhas efetivas, ou seja, identificadas pelo seu usuário final. (PADUELLI, 2007)

Sommeville (2011) define a refatoração como um processo de manutenção preventiva, isso significa alterar o software para melhorar sua estrutura para reduzir sua complexidade ou torná-lo mais compreensível.

Fowler e outros (1997) citados por Sommerville (2011) sugerem que existem situações estereotipadas que podem melhorar por meio da refatoração. São elas:

Quadro 15 - Situações estereotipadas.

SITUAÇÕES	DESCRIÇÃO
Código duplicado	O mesmo código ou muito semelhante pode ser incluído em diferentes lugares de um programa, que pode ser removido e implementado como uma única função quando necessário.
Métodos longos	Se um método for muito longo, ele deve ser reprojetoado como uma série de métodos mais curtos.
Declarações switch	Envolvem duplicação em situações em que o switch depende do tipo de algum valor e podem ser espalhadas em torno do software. Muitas vezes em linguagens orientada a objetos pode se utilizar o polimorfismo para conseguir o mesmo resultado.
Aglutinação de dados	Ocorre quando um mesmo grupo de itens de dados (campos ou parâmetros em classes) reincide em vários lugares do software. Podem ser substituídos por um objeto que encapsule todos os dados.
Generalidade especulativa	Ocorre quando desenvolvedores incluem generalidades em um software, pois estas podem ser necessárias no futuro. Muitas vezes elas podem ser removidas.

Fonte: Sommerville (2011)

A refatoração quando utilizada durante o processo de desenvolvimento de software, muitas vezes é uma forma eficaz de diminuir os custos de manutenção em longo prazo do software. (SOMMERVILLE, 2011)

3.4 CUSTOS

A manutenção de software é uma atividade importante, pois consome a maior parte dos recursos envolvidos no ciclo de vida de um software. Embora não exista um valor exato do custo atrelado à atividade de manutenção, pesquisas apontam que na maioria dos casos, sempre são mais de 50% dos custos destinado ao software. (PADUELLI, 2007)

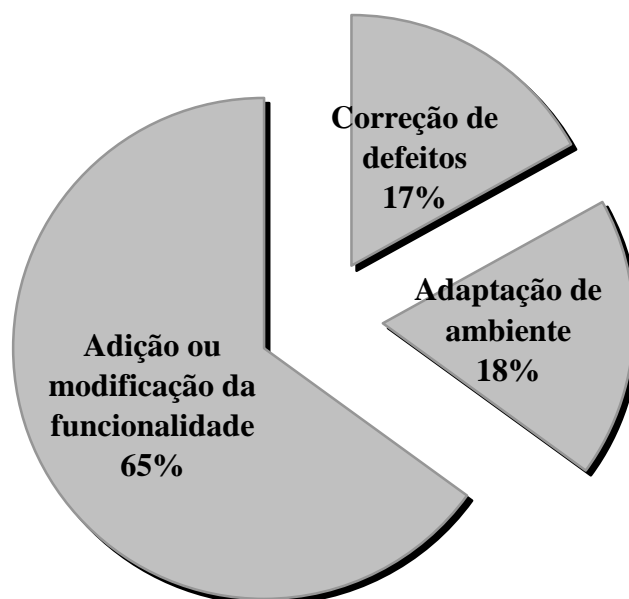
Paduelli (2007) cita que a importância financeira atrelada a atividade de manutenção é agravada ainda mais, quando se leva em consideração os ricos financeiros relacionados as oportunidade de negócio, que podem ser causadas pela falta de gerenciamento ou compreensão da atividade da manutenção de software.

Seneed (2003) citado por Paduelli (2007) afirma que além dos altos custos financeiros, esta atividade é uma das que exige maior esforço dentre as atividades da engenharia de software. Embora existam vários motivos que influenciam os custos da manutenção do software, podemos enumerar alguns pontos citados por Engholm (2010) que estão diretamente relacionados ao custo da manutenção.

- ✓ Documentação inexistente relacionada ao software e a qualidade do mesmo.
- ✓ Qualidade do software.
- ✓ Design do software.
- ✓ Paradigma de programação utilizado no desenvolvimento do software.
- ✓ Tamanho do software, relacionado a quantidade de linhas de código.
- ✓ Qualidade técnica da equipe responsável pela manutenção do software.
- ✓ Regras de negócio embarcadas no software.
- ✓ Plataforma de desenvolvimento do software.

Engholm (2010), afirma que pessoas envolvidas em manutenção gastam em média 50% do esforço tentando entender o código do software, resultado este que geralmente é de um software mal codificado, sem documentação e sem qualidade, podendo ser este o resultado da falta de utilização das técnicas da engenharia de software.

Gráfico 1 - Distribuição do esforço de manutenção.



Fonte: Sommerville (2011).

Os custos relativos de manutenção variam de acordo ao tipo do software, Guimaraes (1983) citado por Sommerville (2011) verificou que os custos de manutenção para softwares são comparáveis aos custos de desenvolvimento de um novo software. Geralmente é mais caro adicionar funcionalidade depois que um software já esta em operação no cliente do que implementar esta mesma funcionalidade durante o processo de desenvolvimento do software.

De acordo com Sommerville (2011), as razões para isto são:

- ✓ Estabilidade da equipe. Não é sempre que as pessoas que desenvolveram o sistema são as mesmas que irão realizar a manutenção, portanto não entende as estruturas do projeto para tomar decisões e antes de implementar as alterações é preciso investir tempo para compreender o software existente.
- ✓ Más práticas de desenvolvimento. O contrato de manutenção pode ser dado a uma empresa ou pessoas diferentes da que realizaram o desenvolvimento do software original, sendo assim essa falta de estabilidade da equipe pode desenvolver um software que não é manutenível, sendo mais difícil de mudá-lo no futuro.
- ✓ Qualificação de pessoal. A equipe de manutenção é inexperiente e não familiarizada com o domínio do software em muitos casos são profissionais jovens e além disso esses softwares podem utilizar linguagens de programação obsoleta, requisitando primeiro que esses profissionais apreendam para depois manter o software.
- ✓ Idade do software e sua estrutura. Com as alterações realizadas no software sua estrutura tende a se degradar, consequentemente com a idade do software os sistemas tornam-se mais difíceis de serem entendidos e alterados.

3.5 GERENCIAMENTO DE MANUTENÇÃO DE SOFTWARE

Devido ao aumento da necessidade de manutenção de software por consequência de software mal desenvolvidos as empresas têm buscado maneiras de lidar com este problema, quando não é possível substituir o software por um novo. Esta busca por alternativas viáveis economicamente e eficazes tecnicamente acabam produzindo diferentes abordagens à resposta da necessidade de manutenção, respostas estas nem sempre as mais corretas e eficazes. (PADUELLI, 2007)

De Lucia et al. (2004) citado por Paduelli (2007) define quatro grupos que são os tomadores de decisão nas questões importantes relacionadas a manutenção de software, são eles:

Programadores: Envolvem questões ligadas à equipe de engenheiros de software, e estão relacionados a assuntos de manutenção de software, e não ao desenvolvimento, sendo necessário avaliar questões técnicas como também a compreensão dos negócios do cliente.

Atitude Gerencial: A postura gerencial é a responsável pelo sucesso ou fracasso no ambiente da manutenção de software. Este tipo de grupo envolve questões relacionadas ao estudo do processo de manutenção de software, que em geral são escassos e possui um relacionamento direto com tempo e dinheiro.

Código-Fonte: É importante definir critérios de qualidade no código-fonte que são questões pertinentes na manutenção de software, pois arquitetura pobre de software, documentação irregular, falta de padrões para atividades de manutenção e ausência de estudos dos impactos de manutenção em outros trechos de código ou módulos do software são considerações pertinentes e devem ser consideradas na atividade de manutenção.

Usuários: Os usuários devem ter participação ao longo de toda atividade de manutenção evitando assim problemas de manutenibilidade no software que possam gerar falta de credibilidade no trabalho do *programador*.

É importante o acompanhamento do resultado das decisões tomadas no âmbito gerencial, efetuando ajustes o tempo inteiro, à medida que falhas forem diagnosticadas, pois são muitos os critérios a serem considerados nas decisões gerenciais. Ao tomador de decisão é fundamental o conhecimento dos objetivos da sua empresa, assim como as características e dificuldades da atividade de manutenção de software possibilitando assim a capacidade de conduzir sua equipe de manutenção. (PADUELLI, 2007)

Neste contexto Basili (1990) citado por Paduelli (2007) propôs três paradigmas para tratar a manutenção de software, são elas:

- ✓ **Conserto Rápido**, que compreende aplicar primeiramente as mudanças necessárias no código-fonte e então depois atualizar a documentação do software a respeito das

mudanças. Este tipo de atitude é a mais comum, normalmente preferida pelo mantenedor, em geral, justificando a urgência em se realizar a manutenção. Entretanto a tarefa da documentação pode ser mal executada ou sequer ser feita devido a pressão pelo tempo do mantenedor, o que pode ocasionar dificuldades em manutenções posteriores.

- ✓ **Melhoria Interativa** propõe inicialmente adequar e atualizar a documentação de alto nível que será afetada, para só então fazer as mudanças necessárias até o nível de código-fonte.
- ✓ **Reuso Total**, consiste em construir um novo software utilizando os componentes do software antigo, isto não é frequentemente utilizado pelas empresas e uma das razões seria a deficiência ainda existente na tecnologia para gerenciar repositório e o reuso de componentes de software.

De acordo com De Lucia et al. (2004) citado por Paduelli (2007) outro desafio gerenciável notável relaciona-se ao esforço de estimar o esforço da atividade de manutenção de software. Tendo uma maneira eficaz de estimar esforço fica mais fácil para os tomadores de decisão argumentar as melhores opções que a empresa possui, contribuindo inclusive para a decisão de terceirizar ou não a atividade de manutenção. Este auxílio na tomada de decisão envolve considerações como:

- ✓ Reavaliação da eficiência dos processos de manutenção.
- ✓ Tratamento da manutenção ou reengenharia.
- ✓ Fundamentar as decisões tomadas.
- ✓ Planejar o orçamento e avaliar a rotatividade da equipe.

A manutenção de software e o desenvolvimento de software apresentam características distintas, portanto nem sempre constitui uma decisão correta aplicar as técnicas, ferramentas e modelos de processos, próprios do desenvolvimento de software de acordo com Polo e outros (2003) citado por Paduelli (2007).

E para isto Polo e outros (2003) citado por Paduelli (2007) afirmam que as razões que justificam essas características distintas são:

- ✓ O esforço despendido em tarefas de manutenção e desenvolvimento não são iguais, constituindo uma prática mais comum nas empresas de gastar mais recursos em teste durante o desenvolvimento do software.
- ✓ Outra diferença importante é o fato da manutenção ter mais similaridade com um serviço do que com o desenvolvimento de software em si, o que acaba implicando em não adotar as mesmas estratégias do desenvolvimento de software.

3.6 NORMA ISO/IEC 12207

Conforme abordado no capítulo 2.8.6.2 a Norma ISO/IEC 12207 prevê um conjunto de processos de engenharia de software que uma organização deve utilizar durante o processo de desenvolvimento de software servindo como um modelo de referencia de processos.

Entretanto na norma a manutenção de software aparece como um dos processos dentro da categoria de processos fundamentais. (PADUELLI, 2007)

Figura 18 - Atividades do processo de manutenção de software e sistema (ISO/IEC 12207).



Fonte: Paduelli (2007).

3.6.1 Implantação do processo

Esta atividade inclui tarefas de desenvolvimento de planos e procedimentos para a manutenção de software, definindo procedimentos para os pedidos de manutenção e estabelecendo uma interface organizacional como o processo de gerenciamento de configuração. (PADUELLI, 2007)

Pigoski (1996) citado por Paduelli (2007) reforça que este processo deve começar cedo no processo de desenvolvimento de software, afirmando que os planos de manutenção devem ser preparados paralelamente com os de desenvolvimento.

3.6.2 Análise do problema e da modificação

Esta atividade tem como objetivo analisar as requisições de manutenção e classificá-las, determinando escopo em termos de custo tamanho, tempo de correção e prioridades. As demais tarefas desta atividade focam no desenvolvimento, documentações de mudança da implementação a ser feita e aprovação das opções adotadas. (PADUELLI, 2007)

3.6.3 Implantação da modificação

Esta etapa engloba a identificação de itens que precisam ser modificados e o que deverá ser implementado, outros requisitos incluem os testes e validação das modificações feitas e se elas estão corretamente implementadas assegurando que não foram afetados os itens que não foram modificados. (PADUELLI, 2007)

3.6.4 Revisão, aceitação da modificação

Nesta atividade as tarefas são dedicadas à confirmação da integridade do software que foi modificado e à consolidação com o cliente, quando ele concorda e aprova a conclusão do requisito de manutenção. Nesta etapa vários processos de apoio podem ser utilizados tais como, garantia da qualidade, verificação, validação e revisão conjunta. (PADUELLI, 2007)

3.6.5 Migração

Etapa que corresponde à atividade quando o software for atualizado no ambiente de operação, aqui será preciso o desenvolvimento de planos de migração, usuários precisarão estar cientes dos requisitos, dos motivos do antigo ambiente não ser mais suportados e terem a sua disposição a descrição e data de disponibilidade do novo ambiente. (PADUELLI, 2007)

3.6.6 Descontinuação do software

Esta atividade consiste na descontinuação do software por meio de uma formalização junto ao usuário final e de um plano de descontinuação. (PADUELLI, 2007)

3.7 SOFTWARE LEGADO

O Software legado são softwares mais antigos e têm sido foco de continua atenção e preocupação desde os anos 1960, pois algumas vezes há uma característica adicional que pode estar presente neste tipo de software que é a *baixa qualidade*. (PRESSMAN, 2011)

Pressman (2011) entende que esses tipos de software algumas vezes têm projetos não expansíveis, código intrincado, documentação inexistente ou pobre, casos de testes e resultados esquecidos, assim como um histórico de modificações mal administrados, porém ainda assim esses softwares são indispensáveis para o negócio de muitas organizações.

Segundo Visaggio (2001) citado por Paduelli (2007) este tipo de sistema normalmente representa um dos bens com maior valor econômico de uma empresa, tendo isto em mente, este tipo de software envolve muitas decisões e variáveis a serem consideradas quando se pensa em manutenção e evolução deste tipo de software. Entre essas considerações algumas delas são:

- ✓ É arriscado substituir este tipo de software se ele já está estável.
- ✓ Seus usuários já estão acostumados com a forma de trabalhar com o software e uma mudança pode não ser bem vinda.
- ✓ A mudança caso exista, vai exigir gastos com treinamento de todos os envolvidos deste tipo de software.
- ✓ A compra ou construção deste um novo software para substituir o legado pode extrapolar previsões de custos e prazos.
- ✓ Um novo software pode possuir menos funcionalidades que o antigo, dificultando as tarefas do usuário que o utilizar.

3.7.1 Definição

Um software desenvolvido no passado e ainda em utilização em função da dependência do sistema em uma determinada organização constitui em um sistema legado. Por causa dessa dependência cada vez maior em relação ao software legado nas organizações e considerando a evolução da tecnologia, tanto de hardware como a de software nos últimos anos, ainda mais com o abandono de linguagens de programação mais antigas e com advento de novas metodologias no desenvolvimento de software, é fortemente esperado que estes softwares tenham sido construídos com alguma tecnologia ou método que hoje é obsoleto. (PADUELLI, 2007)

Esses tipos de softwares constituem mais um desafio para a engenharia de software, especialmente para a manutenção de software, uma vez que dificilmente serão abandonados de imediato, justamente pelo que representam para as organizações. (PADUELLI, 2007)

Dayani Fard citado por Pressman (2011) descrevem software legado da seguinte maneira:

Sistema de software legado.... Foram desenvolvidos décadas atrás e têm sido continuamente modificados para se adequar a mudanças dos requisitos de negócio e as plataformas computacionais. A proliferação de tais sistemas esta causando dores de cabeça para grandes organizações que consideram dispendiosos de manter e arriscados de evolui.

3.7.2 Crise do legado e evolução

De acordo com o *Information Technology Reserach Institute*, citado por Engholm (2010) o custo relacionado a manutenção desse software e o gerenciamento de sua evolução representa cerca de 90% do custo total de uma organização.

Quadro 16- Custo proporcional de manutenção do Software Legado.

ANO	PROPORÇÃO DE CUSTOS	DEFINIÇÃO
2000	> 90%	Custo de software relacionado à manutenção e evolução de sistemas.
1993	0,75	Manutenção de software/orçamento de sistemas em 1000 empresas.
1990	> 90%	Custo de software relacionado a manutenção do sistema.
1990	60-70%	Manutenção de software/orçamento total de operação de sistemas.
1988	60-70%	Manutenção de software/orçamento operacional do total de sistemas de informação gerencial.
1984	65-75%	Esforço gasto na manutenção de softwares/total disponíveis relacionado a engenharia de software.
1981	> 50%	Tempo gasto pela equipe de manutenção/ tempo total.
1979	0,67	Custos de manutenção/custo total do software.

Fonte: Engholm (2010).

Nos anos de 1970 surgiu o termo crise do software, quando não existia ainda a engenharia de software, este termo estava relacionado às dificuldades enfrentadas no

desenvolvimento de software, inerentes ao aumento das demandas e da complexidade delas, aliado a inexistência de técnicas apropriadas para resolver estes desafios. (ENGHOLM, 2010).

Devido a isto surgiu o conceito de crise do software, tendo vários sintomas, tais como:

- ✓ Software de baixa qualidade.
- ✓ Projetos com prazos e custos e maiores que os planejados.
- ✓ Software não atendendo aos requisitos dos clientes.
- ✓ Custos e dificuldades no processo de manutenção.

Sommerville (2003) citado por Paduelli (2007), explica que as razões para explicar a dificuldade de manutenção em softwares legados são:

- ✓ Diferentes partes do software foram desenvolvidas por equipes diferentes implicando em uma não uniformidade no estilo de programação ao longo da vida útil do software.
- ✓ O software ou partes dele podem ter sido desenvolvidos utilizando alguma linguagem de programação obsoleta, levando a dificuldade de se encontrar profissionais com conhecimento e experiência podendo levar a terceirização da manutenção, elevando assim os custos.
- ✓ Um software difícil de ser compreendido, devido aos muitos anos de manutenção nos quais foram alteradas as estruturas do software.
- ✓ Os dados processados pelo software podem estar armazenados em arquivos separados, estruturas distintas ou incompatíveis, podendo ter erros, duplicação de dados, ou até mesmo dados incompletos.

Analisando este contexto e os sintomas expostos, podemos ver que a crise ainda esta presente hoje em dia, mesmo dispondo de técnicas apropriadas na atualidade para resolvê-las, vemos estes tipos de problemas nos softwares legados que ainda são mantidos. (ENGHOLM, 2010)

Pressman (2011) define que se o software legado atende as necessidades de seus usuários e roda de forma confiável ele não está quebrado e, portanto, não precisa ser consertado ou evoluído. Entretanto com o decorrer do tempo pode ser necessário evoluir o software pelas seguintes razões:

- ✓ O software deve ser adaptar para atender as necessidades de novos ambientes ou de novas tecnologias computacionais.
- ✓ O software deve ser aperfeiçoado para implementar novos requisitos de negócio.
- ✓ O software deve ser expandido para torná-lo interoperável com outros bancos de dados.
- ✓ O software deve ser rearquitetado para torná-lo viável dentro de um ambiente de rede.

Quando essas evoluções no software ocorrerem, o software legado deve passar por uma reengenharia de software para permanecer viável no futuro. (PRESSMAN, 2011)

3.7.3 Gerenciamento de software legado

Geralmente as organizações que possuem um software legado possuem um orçamento limitado para a manutenção e modernização destes softwares, portanto elas precisam fazer uma avaliação realista destes sistemas e em seguida adotar uma estratégia mais adequada para a evolução deles. (SOMMERVILLE, 2011)

De acordo com Sommerville (2011) existem quatro opções estratégicas para serem adotadas.

Descartar completamente o sistema: Esta opção ocorre quando o software legado não esta mais contribuindo efetivamente com os processos de negocio, ou seja, já não possui mais um valor econômico para manter este software.

Manutenções regulares: Essa opção é escolhida quando o software ainda é importante no processo de negócio da organização, sendo o software estável e tem poucas mudanças solicitadas pelos usuários do software.

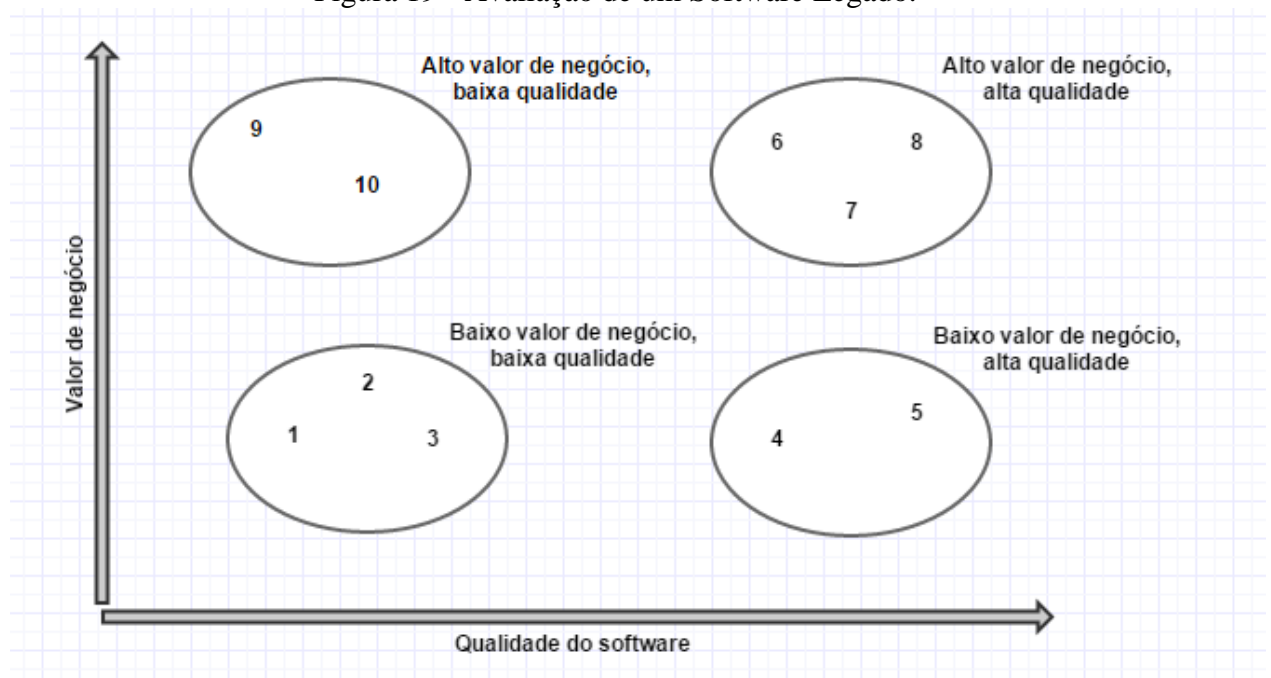
Reestruturação visando melhorar a manutenibilidade: Essa opção é escolhida quando o sistema foi degradado devido às mudanças feitas afetando assim sua estabilidade.

Substituição total ou em parte para um novo software: Essa opção é escolhida quando fatores como hardwares novos significam que o software legado não pode mais continuar em operação ou quando o software legado pode permitir o desenvolvimento de um novo software a um custo razoável.

Naturalmente essas opções não são exclusivas, várias opções podem ser aplicadas dependendo do tipo de software que esta sendo avaliado, porém quando se avalia um software legado precisam-se ter duas perspectivas, a de negócio e a técnica. Na primeira se verifica se o negócio realmente necessita deste software, na perspectiva técnica avalia-se a qualidade do software, o apoio do software e hardware. Em seguida deve-se utilizar a combinação dessas duas perspectivas para tomar uma estratégia do que será feito com o software legado. (SOMMERVILLE, 2011)

Sommerville (2011) define quatro grupos de softwares legados, conforme figura abaixo.

Figura 19 - Avaliação de um Software Legado.



Fonte: Sommerville (2011).

Baixa qualidade, baixo valor de negócio: Manter um software deste tipo é caro e a taxa de retorno para o negócio é bastante reduzida, levando isto em consideração este software deve ser descartado.

Baixa qualidade, alto valor de negócio: Este software tem uma contribuição importante para o negócio da empresa, portanto não pode ser descartado, porém sua baixa qualidade significa altos custos de manutenção então eles devem ser reestruturados para melhorar a qualidade ou substituídos.

Alta qualidade, baixo valor de negócio: Esses softwares não contribuem muito para o negócio e seus custos de manutenção não são altos, sendo assim não vale substituir o software, apenas mantê-lo com manutenção normal desde que elas não fiquem caras.

Alta qualidade, alto valor de negócio: Esses softwares precisam ser mantidos em operação, sua alta qualidade significa que não possui necessidade de investimentos para mudanças ou substituição, portanto manutenção normal do sistema deve ser mantida.

3.8 REENGENHARIA DE SOFTWARE

O processo de manutenção e evolução de um software envolve a compreensão dele e do que tem que ser mudado para em seguida realizar a implementação dessas mudanças, porém muitos softwares em especial os softwares legados são de difíceis de serem compreendidos e mudados. Para esses casos existe a reengenharia visando melhorar a estrutura e inteligibilidade dos softwares legados e a possibilidade deles serem mais fáceis de serem mantidos. (SOMMERVILLE, 2011)

Sommerville (2011) cita que entre as atividades da reengenharia podemos citar.

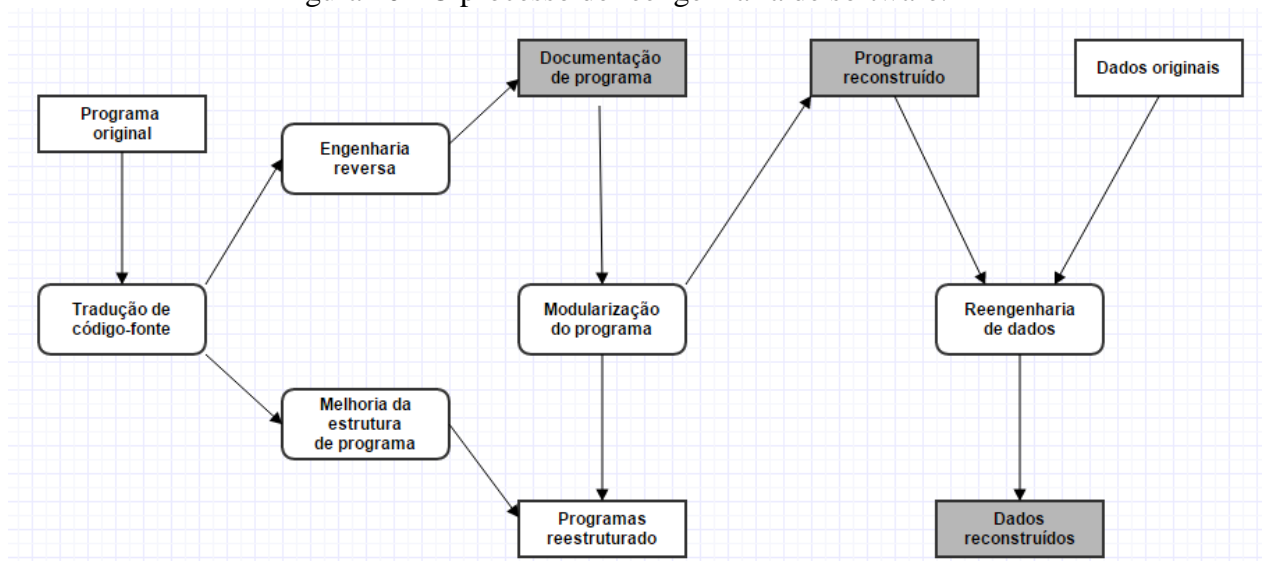
- ✓ Redocumentação do software.
- ✓ Refatoração da arquitetura do software.
- ✓ Mudança de linguagem de programação para uma moderna.
- ✓ Modificações e atualizações da estrutura e dos dados do software.

Apesar dessas modificações a funcionalidade do software não é alterada, portanto devem-se evitar grandes mudanças na arquitetura do software. (SOMMERVILLE, 2011)

Ainda Sommerville (2011) cita que existem dois grandes benefícios importante na reengenharia de software ao invés de substituir totalmente o software, são eles:

1. **Risco Reduzido.** Existe um alto risco em desenvolver um novo software crítico de negócios, podendo ocorrer erros na especificação do software ou no desenvolvimento dele.
2. **Custo reduzido.** O custo de uma reengenharia no software é significativamente menor do que o desenvolvimento de um novo software.

Figura 20 - O processo de reengenharia de software.



Fonte: Sommerville (2011).

Tradução de código-fonte: Utilizando uma ferramenta de tradução o programa é convertido de uma linguagem antiga para uma mais moderna ou outra diferente.

Engenharia reversa: Um processo totalmente automatizado consiste em extrair as informações do programa analisado podendo ajudar a documentar sua organização e funcionalidade.

Melhoria de estrutura de programa: A estrutura do programa é analisada e modificada para que se torne mais fácil de ser compreendida, podendo ser parcialmente automatizado.

Modularização de programa: As partes relacionadas do programa são agrupados e onde houver redundância é removida, podendo envolver refatoração da arquitetura do programa, sendo esta etapa totalmente manual.

Reengenharia de dados: Os dados processados pelo programa são alterados para refletir as mudanças do programa, podendo significar redefinição dos esquemas de banco de dados, conversão dos dados existente para a nova estrutura.

A reengenharia do software pode não exigir necessariamente todas as etapas da Figura 20, porém os custos da reengenharia, obviamente, dependem da extensão do trabalho e destas etapas, por exemplo, tradução de código-fonte é a opção mais barata, já a migração de arquitetura é a mais cara. (SOMMERVILLE, 2011)

4. MPS.BR – MELHORIA DE PROCESSO DE SOFTWARE BRASILEIRO

O modelo MPS.BR – Melhoria de Processo de Software Brasileiro foi criado por pesquisadores como uma iniciativa para melhorar a capacidade de desenvolvimento de software nas empresas brasileiras, sob coordenação da Softex – Associação para Promoção da Excelência do Software com o apoio do Ministério de Ciência, Tecnologia e Inovação. Neste capítulo apresentaremos este modelo em linhas gerais.

O MPS.BR tem como foco principal as micro, pequenas e médias empresas de software brasileiras que possuem poucos recursos disponível para melhoria de processos, mas que tem necessidades em fazê-lo. Seu objetivo é implantar os princípios da engenharia de software de forma adequada ao contexto das empresas brasileiras, seguindo os padrões internacionais para definição, avaliação e melhoria de processos de software. (KOSCIANSKI, 2007; SOARES, 2007)

A Softex (2015) define o MPS.BR como um projeto de qualidade tendo o seguinte objetivo.

Impulsionar a melhoria da capacidade de desenvolvimento de software e serviços nas empresas brasileiras é uma das metas do programa MPS.BR. Considerado um marco que representa a evolução da qualidade do software desenvolvido no país, ele trouxe ganhos comprovados de competitividade para a indústria nacional.

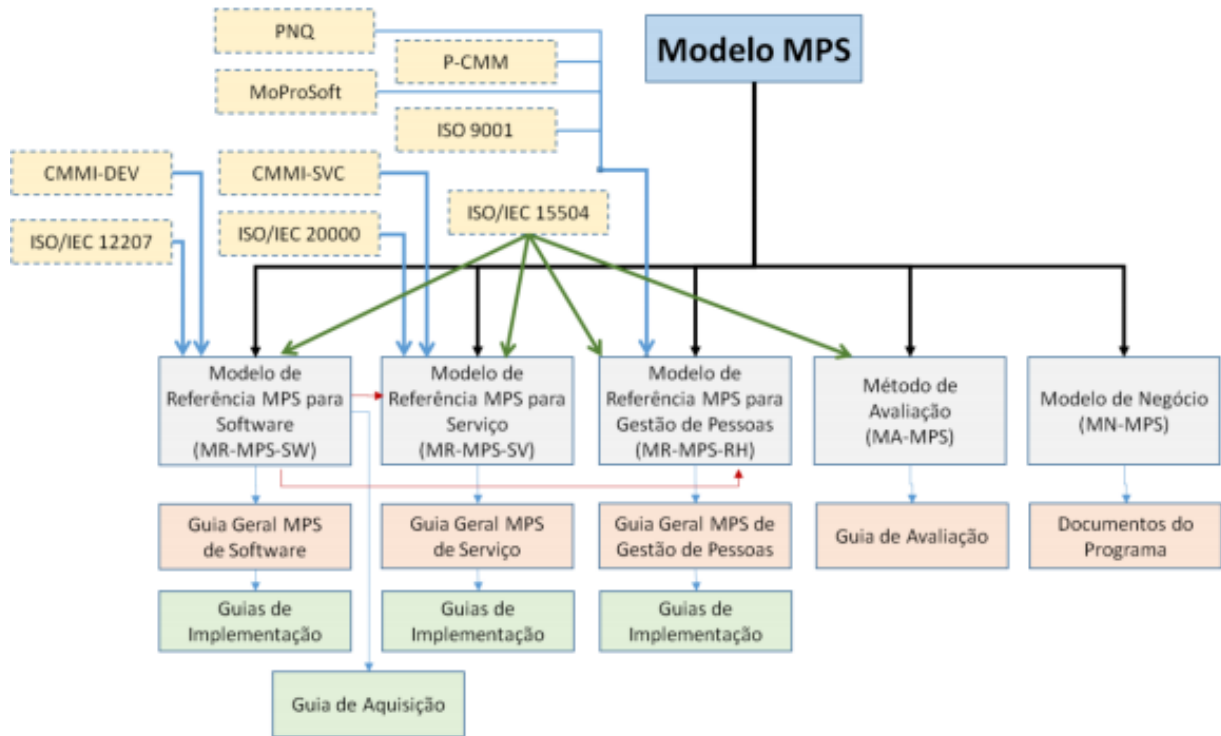
Busca-se que o modelo seja adequado ao perfil de diferentes tipos de empresas, embora com especial atenção as micros, pequenas e médias empresas, sendo que o modelo é compatível com os padrões de qualidade aceitos internacionalmente e que tenha como pressuposto o aproveitamento da competência existente nos padrões e modelos de melhoria de processo já disponíveis. (SOFTEX, 2014)

4.1 ESTRUTURA

O modelo baseia-se nos conceitos de maturidade e capacidade de processo para avaliação e melhoria da qualidade do software e serviços, dentro deste contexto o modelo apresenta cinco componentes (Figura 19): Modelo de Referência MPS para Software (MR-

MPS-SW), Modelo de Referência MPS para Serviços (MR-MPS-SV), Modelo de Referência MPS para Gestão de Pessoas (MR-MPS-RH), Método de Avaliação (MA-MPS) e o Modelo de Negócio para Melhoria de Processo de Software e Serviços (MN-MPS). (SOFTEX, 2014)

Figura 21 - Estrutura do MPS.BR.



Fonte: Softex (2014).

Segundo a Softex (2015) o modelo MPS.BR está descrito por meio de documentos em formato de guias que são:

Guia Geral MPS de Software: Contém a descrição geral do modelo MPS e detalha o Modelo de Referência MPS para Software (MR-MPS-SW), seus componentes e as definições comuns necessárias para seu entendimento e aplicação.

Guia Geral MPS de Serviços: Contém a descrição geral do modelo MPS e detalha o Modelo de Referência MPS para Serviços (MR-MPS-SV), seus componentes e definições comuns necessárias para seu entendimento e aplicação.

Guia Geral MPS de Gestão de Pessoas: Contém a descrição geral do modelo MPS e detalha o Modelo de Referência MPS para Gestão de Pessoas (MR-MPS-RH), seus componentes e as definições comuns necessárias para seu entendimento e aplicação.

Guia de Aquisição: Descreve o processo de aquisição de software e serviços, descrito como forma de apoiar as instituições que queiram adquirir produtos de software e serviços apoiando-se no MR-MPS-SW.

Guia de Avaliação: Descreve o processo e o método de avaliação MA-MPS, os requisitos para avaliadores, líderes, avaliadores adjuntos e instituições avaliadores.

Guia de Implementação: Série de documentos que fornecem orientações para implementar nas organizações os níveis de maturidade do Modelo de Referência MR-MPS-SW.

No quadro a seguir são mostrados alguns diferenciais do MPS.BR de acordo com os autores Koscianski (2007) e Soares (2007).

Quadro 17 - Diferencias do MPS.BR

DIFERENCIAIS
Sete níveis de maturidade, possibilitando implantação mais gradual Compatibilidade com o CMMI Criadas para o Brasil e para micro, pequenas e médias empresas Custo acessível Avaliação bienal das empresas Forte interação entre Universidade e Empresa.

Fonte: Koscianski(2007) e Sores (2007).

4.1.1 MR-MPS-SW – Modelo de Referência de Melhoria de Processo de Software

Este modelo possui as definições dos níveis de maturidade dos processos, sendo baseado nas normas ISSO/IEC 12207 e 15504 e adequado ao CMMI, que contém os requisitos a serem cumpridos pela organização que deseja estar em conformidade com o MR-MPS. (KOSCIANSKI, 2007; SOARES, 2007)

O MR-MPS-SW (*Modelo de Referência de Melhoria de Processo de Software*) tem como base os requisitos de processos definidos nos modelos de melhoria de processo e atende a necessidade de implantar os princípios de engenharia de software de forma adequada ao contexto das empresas, estando em conformidade com as principais abordagens internacionais para definição e avaliação de processos. (SOFTEX, 2012b)

4.1.2 Níveis de Maturidade

Os níveis de maturidade são definidos pelo MR-MPS (*Modelo de Referência*) que são uma combinação de processos e capacitação de processos. Estes objetivos de melhoria foram divididos em sete níveis de maturidade A (alto grau de maturidade) ao G (baixo grau de maturidade). (KOSCIANSKI, 2007; SOARES, 2007)

De acordo com a Softex (2012b) os níveis de maturidade têm como objetivo estabelecer patamares de evolução de processos, caracterizando estágios de melhoria da implementação de processos na organização, sendo que sua divisão em sete estágios tem o objetivo de possibilitar a implementação e avaliação adequada em micro, pequenas e médias empresas.

Segundos os autores do MPS-BR citado pelos autores Koscianski (2007) e Soares (2007) os sete níveis devem permitir uma implantação mais gradual do que o CMMI, facilitando as empresas.

No quadro abaixo são mostrados os sete níveis de maturidade do MPS-BR referente ao modelo de software (MR-MPS-SW) e os processos para capacitação da maturidade.

Quadro 18 - Níveis de maturidade do modelo MR-MPS-SW do MPS.BR.

Nível	Maturidade	Processos
A	Em otimização	Inovação e implantação na organização Análise de causas e resolução
B	Gerenciado quantitativamente	Desempenho do processo organizacional Gerência quantitativa do projeto
C	Definido	Análise de decisão e resolução Gerência de risco
D	Largamente definido	Desenvolvimento de requisitos Solução técnica Integração de produto Instalação de produto Liberação de produto Verificação Validação
E	Parcialmente definido	Treinamento Avaliação e melhoria do processo organizacional Definição do processo organizacional Adaptação do processo para gerência de projeto

F	Gerenciado	Medição Gerência de configuração Aquisição Garantia de qualidade
G	Parcialmente gerenciado	Gerência de requisitos Gerência de projeto

Fonte: Koscianski (2007) e Soares (2007).

Neste trabalho iremos abordar a seguir somente o Nível G do MR-MPS-SW, os itens gerência de requisitos e gerência de projeto.

4.1.2.1 Nível G

É o primeiro nível da maturidade do MR-MPS-SW que visa à implantação de melhoria dos processos de software da organização, sendo que a final desta implantação a organização deve ser capaz de gerenciar parcialmente seus projetos de desenvolvimento de software. (SOFTEX, 2013)

Segundo a Softex (2013) dois pontos são desafiadores no momento da implantação do nível G, são eles:

- ✓ Mudança de cultura organizacional, orientando a definição e melhoria dos processos de desenvolvimento de software.
- ✓ Definição do conceito acerca do que é projeto para a organização.

O Nível G é composto pelos processos *gerência de projetos* e *gerência de requisitos*, a seguir veremos estes processos.

4.1.2.1.1 Gerência de Projetos

Este processo envolve várias atividades, tais como, desenvolver um plano geral de controle do projeto, comprometimento para mantê-lo ao longo de toda a execução, conhecimento do progresso do projeto de tal maneira que as ações corretivas possam ser tomadas quando a execução do projeto desvia do planejado. (SOFTEX, 2013)

O propósito da gerência de projetos é estabelecer e manter planos que definem as atividades, recursos e responsabilidades do projeto, bem como prover informações sobre o andamento do projeto que permitam a realização de correções quando houver desvios significativos no desempenho do projeto. (SOFTEX, 2013)

A IEEE (1990) citada pela Softex (2013), em seu glossário, afirma que a gerência de projetos pode ser definida como a aplicação de planejamento, coordenação, medição, monitoramento, controle de divulgação de relatórios, com o intuito de garantir que o desenvolvimento e a manutenção de softwares sejam sistemáticos, disciplinados e qualificados.

O PMBOK (*Project Management Body of Knowledge*) que é de responsabilidade do PMI (*Project Management Institute*) é um guia de gerência de projetos. Ele agrupa o conhecimento em gerência de projetos que é amplamente reconhecido como boas práticas de gerenciamento de projetos, tendo como visão a aplicação de conhecimento, habilidades, ferramentas e técnicas nas atividades do projeto a fim de atender aos seus requisitos. (SOFTEX, 2013)

Entre os resultados esperados pela execução deste processo estão, por exemplo, o escopo do projeto que é bem definido, as tarefas são corretamente estimadas e alocadas dentro da equipe, sendo que a consequência disto é receber cronogramas mais realísticos e controláveis, assim como os riscos que são bem identificados e analisados para que sejam evitados ou minimizados. (KOSCIANSKI, 2007; SOARES, 2007)

4.1.2.1.2 Gerência de Requisitos

O principal objetivo deste processo é controlar a evolução dos requisitos, gerenciando todos os requisitos recebidos ou gerados pelo projeto, incluindo requisitos funcionais e não funcionais, assim como requisitos impostos no projeto pela organização. (SOFTEX, 2013)

O propósito do processo Gerência de Requisitos é gerenciar os requisitos do produto e dos componentes do produto do projeto e identificar inconsistências entre os requisitos, os planos do projeto e os produtos de trabalho do projeto. (SOFTEX, 2013)

Esta atividade envolve identificar os requisitos do software e dos componentes do projeto, bem como estabelecer e manter um acordo entre o cliente e a equipe de projeto sobre os requisitos, além de controlar e tratar as mudanças dos requisitos ao longo do desenvolvimento do projeto. (SOFTEX, 2013)

4.1.3 MR-MPS-SV – Modelo de Referência de Melhoria de Processo de Serviços

Este modelo possui as definições dos níveis de maturidade dos processos, maturidade e atributos do processo sendo descrito no Guia Geral MPS de Serviços. Contém os requisitos que os processos das unidades organizacionais devem atender para estar em conformidade com o MR-MPS-SV. (SOFTEX, 2012a)

4.1.4 MR-MPS-RH – Modelo de Referência de Melhoria de Processo para Gestão de Pessoas

Este modelo contém os requisitos que os processos das empresas ou suas unidades organizacionais devem atender para estar em conformidade com o MR-MPS-RH, contém as definições dos níveis de maturidade, processos e atributos do processo. (SOFTEX, 2014)

4.1.5 MA-MPS – Método de Avaliação

Este método está descrito de forma detalhada no *Guia de Avaliação*, conforme a figura 17, sendo baseado no documento da ISO 15504. O MA-MPS contém o processo de avaliação, os requisitos para avaliadores e para a averiguação da conformidade ao modelo MR-MPS (*Modelo de Referência de Melhoria de Processo de Software*). (KOSCIANSKI, 2007; SOARES, 2007)

4.1.6 MA-MPS – Modelo de Negócio

Este modelo contém a descrição das regras para a implementação do modelo MR-MPS (*Modelo de Referência de Melhoria de Processo de Software*) pelas empresas de consultoria de software e avaliação. (KOSCIANSKI, 2007; SOARES, 2007)

5. METODOLOGIA

Em um trabalho acadêmico, a pesquisa deve ser realizada de maneira séria, e para tanto, é necessário seguir alguns passos para o seu melhor desenvolvimento e sucesso. Para RUIZ (1996, p.48), a pesquisa é “a realização concreta de uma investigação planejada, desenvolvida e redigida de acordo com as normas da metodologia consagradas pela ciência. É o método de abordagem de um problema em estudo que caracteriza o aspecto científico de uma pesquisa.” Ou ainda, conforme MARCONI e LAKATOS (2001, p.43), a pesquisa científica “pode ser considerada um procedimento formal com método de pensamento reflexivo que requer um tratamento científico e se constitui no caminho para se conhecer a realidade ou descobrir verdades parciais.” No campo dos procedimentos metodológicos, “estão os delineamentos, que possuem um importante papel na pesquisa científica, no sentido de articular planos e estruturas a fim de obter respostas para os problemas de estudo.” (RAUPP; BEUREN, 2004, p.76). Dentre as pesquisas possíveis a serem realizadas, o presente trabalho tem como objetivo explicar uma problemática, com referências teóricas já publicadas em documento.

Minayo (2007, p. 44) define metodologia de forma abrangente e concomitante.

(...) a) Como a discussão epistemológica sobre o “caminho do pensamento” que o tema ou o objeto de investigação requer; b) Como a apresentação adequada e justificada dos métodos, técnicas e dos instrumentos operativos que devem ser utilizados para as buscas relativas às indagações da investigação; c) e como a “criatividade do pesquisador”, ou seja, a sua marca pessoal e específica na forma de articular teoria, métodos, achados experimentais, observacionais ou de qualquer outro tipo específico de resposta às indagações específicas.

5.1 CLASSIFICAÇÃO DA PESQUISA

Para a realização da pesquisa metodológica serão utilizados diversos conceitos que servirão de base para o desenvolvimento e conclusão do referido trabalho, segundo os critérios de classificação de MARCONI e LAKATOS (2001), RUIZ (1996) e MICHEL (2009) quanto a sua natureza, fins e meios.

Quanto à natureza desta pesquisa ela é qualitativa, pois ela se preocupa com aspectos da realidade que não podem ser quantificados, centrando-se na compreensão e explicação da dinâmica das relações sociais aprofundando em uma organização, grupo social, etc. (MINAYO, 2001, p. 14)

As características da pesquisa qualitativa são objetivação do fenômeno, hierarquização das ações de descrever, compreender, explicar, precisão das relações entre o global e o local em determinado fenômeno, observância das diferenças entre o mundo social e o mundo natural.

Quanto aos fins, será do tipo pesquisa aplicada, pois tem como objetivo gerar conhecimento para aplicação prática, geridos a solução de problemas específicos envolvendo verdade e interesses da organização.

Quanto aos meios, classifica-se como pesquisa de estudo de caso, devido ao fato de ser de cunho descritivo e sem intervenção sobre a situação pesquisada.

Um estudo de caso pode ser caracterizado como um estudo de uma entidade bem definida como um programa, uma instituição, um sistema educativo uma pessoa, ou uma unidade social. Visa conhecer em profundidade o como e o porquê de uma determinada situação que se supõe ser única em muitos aspectos, procurando descobrir o que há nela de mais essencial e característico. O pesquisador não pretende intervir sobre o objeto a ser estudado, mas revelá-lo tal como ele o percebe. O estudo de caso pode decorrer de acordo com uma perspectiva interpretativa, que procura compreender como é o mundo do ponto de vista dos participantes, ou uma perspectiva pragmática, que visa simplesmente apresentar uma perspectiva global, tanto quanto possível completa e coerente, do objeto de estudo do ponto de vista do investigador (FONSECA, 2002, p. 33).

Este tipo de pesquisa objetiva a compreensão e o conhecimento da situação a ser estudada retratando a realidade de forma concreta e explícita, demonstrando assim os diferentes pontos de vista do campo em questão avaliado.

5.2 PROCEDIMENTOS METODOLÓGICOS

O presente trabalho consiste em um estudo de caso a ser realizado na empresa Dígitro Tecnologia Ltda. Na cidade de Florianópolis, Santa Catarina.

Dentre os possíveis procedimentos para a realização da pesquisa, coleta e análise dos dados optou-se neste trabalho pela realização de pesquisa bibliográfica para entendimento dos assuntos relacionados à engenharia de software, processo de desenvolvimento e manutenção de software e a qualidade de software em conjunto com o modelo MPS-BR.

Uma das vantagens da pesquisa bibliográfica é que esta permite ao pesquisador cobrir uma gama maior de fontes de consulta se comparado com uma pesquisa realizada de forma direta. (GIL, 2010)

Portanto neste trabalho serão utilizadas diversas referências, tais como livros, artigos, revistas, testes de mestrado e conteúdo diversos encontrados na internet. O objetivo é principalmente entender o atual modelo de processo de desenvolvimento de software no ciclo de manutenção na área da empresa em que o estudo será realizado, a fim de propor um novo modelo de manutenção de software adequado a modelo de qualidade MPS-BR.

Com o objetivo de entender o processo de manutenção e customização de software realizado na área foco do estudo de caso, será estudo o modelo aplicado atualmente na empresa. Para isto serão utilizadas documentações já existentes na empresa, além de entrevistas com os colaboradores que já trabalham em tarefas relacionadas à manutenção de software.

Após entender todo o processo já existente, o objetivo é utilizar os conceitos e melhores práticas no desenvolvimento e manutenção de software com apoio de ferramentas e tecnologias existentes no mercado para propor um modelo de manutenção de software adequado a um modelo de qualidade, para que possa ser aplicado e torne o processo atual de manutenção robusto, eficiente e eficaz.

Mapeado as dificuldades, os pontos críticos e as melhorias necessários no processo de manutenção e customização dos softwares já existente, será proposto um novo modelo para o processo de manutenção e customização de software na área da empresa.

Dentro desse contexto, as atividades que serão realizadas neste trabalho podem ser agrupadas em três macros atividades:

1. Levantamento da fundamentação teórica.
2. Estudo de caso e modelagem do processo atual “*As Is*”.
3. Proposta de um novo processo “*To Be*”.

Cada uma delas é detalhada a seguir.

Fundamentação Teórica.

- ✓ Engenharia de software.
- ✓ Modelos de desenvolvimento de software
- ✓ Modelos Ágeis.
- ✓ Modelos de Qualidade
- ✓ Manutenção de Software
- ✓ MPS.BR

Estudo de caso.

Descrição:

- ✓ Da empresa.
- ✓ Soluções da empresa.
- ✓ Modelos de desenvolvimento de softwares da empresa.
- ✓ Modelos de qualidade da empresa.
- ✓ Normas adotadas pela empresa.

Modelagem:

- ✓ Processo de customização de software.
- ✓ Processo de manutenção de software.

Proposta.

- ✓ Comparação entre os modelos e processo de desenvolvimento de software com a teoria.
- ✓ Proposta de um novo processo de customização e manutenção de software considerando o MPS.BR nível G.
- ✓ Modelagem de um novo processo de manutenção e customização de software.
- ✓ Descrição e adoção de ferramentas e tecnologias de apoio para a automatização do processo de manutenção e customização de software proposto.
- ✓ Comparativo do processo de customização e manutenção de software propostos aos requisitos esperados do modelo de qualidade do MPS.BR nível G.

5.3 DELIMITAÇÕES

Neste trabalho será elaborada uma proposta de um modelo de processo que adota as práticas do MPS.BR para a manutenção e customização de software na empresa do estudo de caso. Essa a proposta será sugerida e apresentada à empresa, mas poderá não ser aplicado devido a isto ser uma decisão da organização estudada neste trabalho. Portanto, considerando isto como uma delimitação deste trabalho.

6. ESTUDO DE CASO

Neste capítulo será apresentado o estudo de caso feito no setor de desenvolvimento de software das soluções corporativas da empresa Dígitro Tecnologia da região da Grande Florianópolis. As principais atividades desse setor são rotineiras como manutenção e customizações de softwares legados, estes com mais de dez anos no mercado e que necessitam estar sendo aperfeiçoados para competir em um mercado cada vez mais acirrado.

6.1 DESCRIÇÃO DA EMPRESA

A Dígitro Tecnologia é uma empresa brasileira, que desenvolve soluções em telecomunicações, tecnologia da informação e inteligência, possuindo softwares e hardwares próprios, ela provê soluções para a comunicação de voz e dados, sistemas de inteligência corporativa e investigativa, serviços de gerenciamento de performance de TI e Telecom, Call Center e URA (Unidade de Resposta Audível). (DÍGITRO TECNOLOGIA, 2010a)

Criada em 1º de setembro de 1977 por iniciativa de jovens engenheiros recém-formados. Em uma fase experimental a organização ainda não atuava no campo de Telecom, somente a partir de 1981, a empresa passou efetivamente a direcionar seu foco para o setor de telecomunicações, desenvolvendo e disponibilizando soluções inovadoras.

A empresa destacou-se como uma empresa inovadora, em constante aprimoramento nas soluções para seus clientes. Por isso estar presente em todo o País, prestando sempre o melhor serviço em qualquer lugar e a qualquer hora faz parte da missão e da conquista no mercado brasileiro. (DIGITRO TECNOLOGIA, 2010a)

Além disso, a Dígitro define sua qualidade em produzir produtos e serviços que atendam as necessidades contratadas pelo cliente, com rentabilidade necessária para o crescimento do negócio, por ser do setor de tecnologia, ela sempre busca melhoria em seus produtos e serviços tendo uma evolução constante do seu sistema de gestão de qualidade, possuindo certificações TL 9000, CMMI e ISO 9001:2008.

6.1.1 História

A Dígito possui mais de trinta anos de história com seu portfólio de produtos e serviços para diversas empresas em segmentos corporativos com operadoras de telefonia, tendo a mente, o coração e o espírito no futuro, ela construiu sua própria trajetória. (DÍGITRO TEDCNOLOGIA, 2015b)

A seguir será apresentado um pouco da sua história.

Quadro 19 - Timeline Dígito Tecnologia.

ANO	MARCO
1977	Fundação da Dígito, em 1º de setembro de 1977, com foco em tecnologia.
1983	Desenvolvimento do sistema de despertador telefônico automático para a TELESC e outros produtos para operadoras
1986	A empresa se firma como desenvolvedora de tecnologia, firmando-se no Complexo Industrial de Informática.
1991	Lançamento do DACT 512T - plataforma que integra em um único sistema: capacidade de mensagens digitais, identificação do assinante por bilhetagem e tarifação diferenciadas.
1993	Desenvolvimento da plataforma digital AXS, voltada para aplicações de grande porte, com conceito multissolução.
1994	A expansão da empresa exige mudança de sede, para comportar uma equipe de desenvolvimento maior.
1996	A 1ª empresa de Florianópolis a conquistar a certificação ISO 9001.
1997	A área de telecomunicações é privatizada e a Dígito parte em busca de novos mercados. Desenvolve o BXS, com um conceito que integra telecomunicações e informática (CTI). O BXS é o antecessor das plataformas de CTI (Computer & Telephone Integrated). Inaugurada a filial Dígito em São Paulo.
1998	Lançamento do Sistema Guardião, que, mais tarde, dará origem a todos os sistemas de Inteligência Investigativa e Estratégica. Lançada a placa BXS-RAS, concentrador de acesso remoto, especialmente voltado para provimento de acesso discado à Internet.
2001	Disponibilizado sistema de URA para serviços em Call Centers, Contact Centers e sistemas de telefonia corporativa.
2002	Primeiro projeto de grande porte com voz sobre IP.
2003	Início de atuação no mercado internacional.
2004	Lançamento da linha de plataformas NGC (Next Generation Communication), baseadas na tecnologia NGN (Next Generation Network).
2005	Lançamento dos terminais de telefonia IP: ATA, IP Phone e FaleWEB.

2006	A Dígitro é a 1ª empresa brasileira a conquistar a certificação TL 9000, norma internacional de gestão da qualidade, específica para a área de telecomunicações.
2007	A Dígitro participa no suporte aos XV Jogos Panamericanos, no Rio de Janeiro, destacando-se como provedora de sistemas de inteligência para segurança em grandes eventos.
2008	Lançamento do SmartCell IP, solução do segmento de Redes Convergentes IP.
2009	Inauguração da nova sede em Capoeiras, construída dentro dos conceitos de sustentabilidade.
2010	Certificação PPB como empresa que atende às condições de bens de informática e automação desenvolvidos no País, reconhecida pelo MCTI.
2011	A Dígitro está pronta para o futuro trazendo uma combinação de soluções junto à sua equipe de colaboradores. A missão continua a ser: prestar serviços de qualidade e superar as expectativas de seus clientes ao redor do mundo.
2012	Inauguração da unidade do Peru. Prêmio Expressão de Ecologia na categoria "Construção Sustentável".
2013	Certificação CMMI. Certificação Empresa Estratégica de Defesa. Prêmio Expressão de Ecologia na categoria "Conservação de Insumos de Produção - Energia".
2014	Certificação CERTICS.

Fonte: Dígitro Tecnologia (2015b)

6.1.2 Soluções

A Dígitro possui um portfólio de soluções voltadas para as áreas de Telecom e TI, estando dividida em dois segmentos, o corporativo e o de segurança pública. Em constante desenvolvimento tecnológico a empresa leva para o mercado soluções e serviços inovadores que alinha inteligência, Telecom e TI.

A seguir serão apresentadas algumas soluções feitas pela empresa, segundo Dígitro Tecnologia (2015d).

6.1.2.1 Pabx

O conceito PABX surgiu na década de noventa quando as empresas de telecomunicações proporem às operadores de telefonia a substituição das centrais telefônicas eletros mecânica pelas centrais digitais, sendo que estas ultimas têm mais facilidade para os usuários e maior valor agregado que as antigas.

O termo PABX significa *Private Automatic Branch* em inglês, ou seja, troca automática de ramais privados, basicamente é uma rede de telefonia privada usada por uma empresa ou até mesmo uma residência, podendo constituir de uma plataforma hardware ou somente o software.

O PABX Dígitro incorpora ramais, celulares ampliando a mobilidade dos colaboradores, sendo que o celular atua como um ramal da central telefônica tendo participação em todas as funcionalidades da plataforma como conferências, consultas e siga-me. (DÍGITRO TECNOLOGIA, 2015e)

A solução da Dígitro é multi-funcional, modular e de fácil operação, possuindo uma interface intuitiva é fácil de configurar funcionalidades e acessos ao administrador da central ou como operador. Possui mais de 50 funcionalidades para facilitar a comunicação, agregar mais valor ao negócio da empresa, acompanhando o crescimento das demandas do mercado e podendo até se transformar em um call center corporativa trazendo resultados palpáveis para as empresas.

Figura 22 - Produtos da solução PABX.



Fonte: Dígitro Tecnologia (2015e)

6.1.2.2 Easy Call

O Easy Call é solução da Dígitro para call centers dos mais variados portes, sua eficiência faz ele a ponte do cliente ao público aumentando a qualidade no atendimento, facilitando o atendimento aos clientes, os operadores atendentes ficam livres para se dedicar exclusivamente aos clientes, processo de discagem automatizado, entre outras funcionalidades aumentando a produtividade da PA perante o cliente. (DIGITRO TECNOLOGIA, 2015f)

Ele é um sistema modular podendo se integrar com a solução de PABX Dígitro, sendo possível customizar para as telefonias fixa móvel ou VOIP, tornando assim uma solução ideal para call centers.

Figura 23 - Produtos da solução Easy Call.



Fonte: Dígitro Tecnologia (2015f)

6.1.2.3 Ura

A URA da Dígitro ou Unidade de Resposta Audível, é um equipamento para call center que provê serviços automáticos para os clientes que ligam para a central telefônica sem a necessidade de um atendente. É a solução ideal para quem busca agilidade e flexibilidade operacional, sua síntese de fala permite que se digite um texto e construa rapidamente a mensagem personalizada do jeito que o cliente deseja. O reconhecimento da fala simplifica a interação da URA da empresa com o cliente, sendo que as mensagens divulgadas pela URA Dígitro podem ser alteradas rapidamente e sem complicações. (DIGITRO TECNOLOGIA, 2015g)

Figura 24 - Produtos da solução de URA.



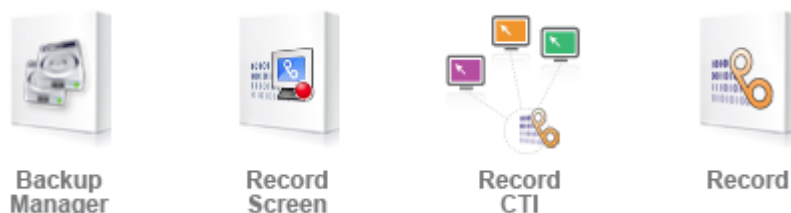
Fonte: Dígitro Tecnologia (2015g)

6.1.2.4 Gravação

O sistema de gravação é uma solução para call centers, fornecendo a possibilidade de registrar as chamadas de uma central telefônica permitindo aprimorar atendimentos e garantindo mais eficiência no processo de um call center, as chamadas que podem ser gravadas são telefonia fixa, móvel ou ip, podendo ser realizada uma pesquisa de dados destes registros de forma inteligente e útil para a administração do call center. (DIGITRO TECNOLOGIA, 2015h)

O Gravador da Dígitro é fácil de implantar e de usar, os dados são rapidamente recuperados de forma amigável e de qualquer lugar, a qualquer hora. Para a gravação automática basta especificar quando e como deseja gravar possuindo um suporte integral e total aos sistemas de call center da Dígitro, tais como URA e o Easy Call, além de ter uma integração total com a central de telefonia da Dígitro o PABX ou de terceiros

Figura 25 - Produtos da solução Gravação



Fonte: Dígitro Tecnologia (2015h)

6.1.2.5 Contact Center

Interact é a solução de contact center da Dígitro que possui uma arquitetura inteligente e total, facilitando a vida de administradores e operadores e fidelizando o cliente. Com o Interact é possível ter um desempenho otimizado, sendo que a produtividade de cada PA (Ponto de Atendimento) é maior, possuindo relatórios detalhados que facilitam os ajustes das estratégias com rapidez, tudo em tempo real de qualquer lugar e qualquer hora. (DIGITRO TECNOLOGIA, 2015i)

O Interact aproxima a empresa do cliente, é uma evolução do contact center sendo uma ferramenta eficiente de gestão, onde o cliente tem um atendimento diferenciado. Além disso, a solução permite a expansão gradativa e programada na medida da sua necessidade permitindo a integração com as soluções de URA, Gravação e PABX. (DIGITRO TECNOLOGIA, 2015i)

Figura 26 - Produtos da solução Contact Center.



Fonte: Dígitro Tecnologia (2015i)

6.1.3 Plataformas

A Dígitro possui uma plataforma base utilizada para suas múltiplas soluções é a NGC, Next Generation Communication, que este presente nas soluções PABX, Call Center, Contact Center, Ura, Gravador e outras. (DIGITRO TECNOLOGIA, 2015j)

A principal característica da plataforma NGC é a modularidade que permite atender as necessidades de empresas e instituições dos mais diferentes portes, proporcionando uma expansão programada para adequação conforme a demanda da empresa. Possui uma arquitetura aberta que garante plena conectividade com os mais diversos ambientes

computacionais e softwares, além de integração com ferramentas externas como, por exemplo, CRMs (Customer Relationship Management). (DIGITRO TECNOLOGIA, 2015j)

Figura 27 - Plataformas NGC.

	<p>NGC Evolution Solução para empresas com grandes demandas, que exigem uma plataforma mais robusta para atender as suas múltiplas necessidades de roteamento de dados, telefonia e conectividade.</p>
	<p>NGC Corporate Indicado para empresas que necessitam rotear dados e apresentam um uso mais intensivo de telefonia e conectividade.</p>
	<p>NGC Office Ideal para empresas de pequeno e médio porte, com sistemas menos complexos.</p>
	<p>NGC Office Lite Adequado para pequenas e médias empresas e filiais de grandes companhias.</p>

Fonte: Dígitro Tecnologia (2015j)

6.1.4 Estrutura organizacional

A seguir é apresentada a estrutura organizacional da empresa.

Figura 28 - Organograma Dígito Tecnologia



Fonte: Autor (2015)

A organização possui uma diretoria de desenvolvimento e tecnologia (Diretoria), que contém setores responsáveis pelo desenvolvimento e manutenção de software, arquitetura das soluções de hardware e software, portfólio de produtos e projetos. A seguir serão descritos apenas os setores que pertencem à área da Tecnologia, por serem estes os que estão relacionados ao escopo deste trabalho.

6.1.4.1 Diretoria de Desenvolvimento

Esta área concentra a diretoria responsável pelas tomadas de decisões estratégicas referente aos produtos, softwares e projetos, tendo contato direto com as demais áreas de marketing e suporte ao cliente visando atender as necessidades de negócio da empresa.

6.1.4.1.1 Arquitetura

Esta área é responsável pela definição e especificação em termos de como será feito a arquitetura de software e hardware dos novos produtos ou projetos que serão desenvolvidos para atender a demanda do mercado. Além disso, fornecem suporte com novas tecnologias, metodologias e opções em termos de hardware e software para todos os produtos da empresa.

6.1.4.1.2 Projetos

Esta área é responsável por fazer a gestão dos projetos em andamento pela empresa.

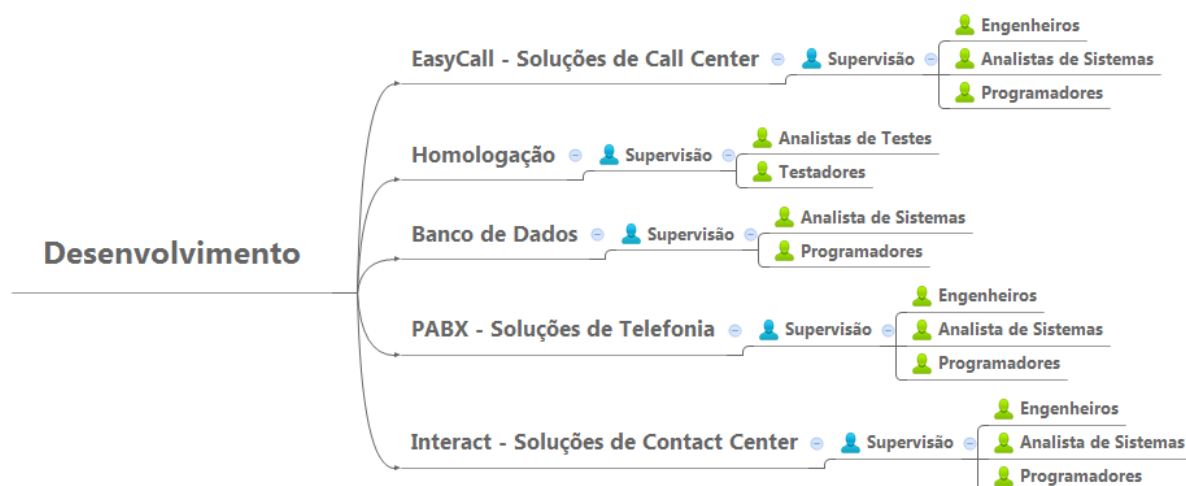
6.1.4.1.3 Desenvolvimento

A área possui um total de aproximadamente setenta colaboradores executando os mais variados tipos de atividades, sendo esta área responsável pelas soluções do mercado corporativo, tais como soluções de call center, contact center e pabx.

As equipes são divididas por subáreas que compõem os produtos dos segmentos corporativos, banco de dados e homologação. Seus principais colaboradores são engenheiros,

analistas de sistemas, analistas de testes, programadores e testadores que possuem um cargo acima que é o de supervisão e um com maior hierarquia que é a gerência, conforme figura abaixo.

Figura 29 - Estrutura das equipes do Desenvolvimento.



Fonte: Autor (2015)

As principais atividades da área são o desenvolvimento, customizações e manutenção das soluções do segmento corporativo existentes na organização, entretanto determinadas subáreas tem atividades específicas que ajudam no processo das soluções do segmento corporativo.

6.1.5 Modelos de Qualidade

As soluções Dígito de TI, Telecom e Inteligência são atestadas e utilizadas por mais de 2500 clientes que compõem a carteira de negócios da empresa, porém além desta credibilidade a empresa possui certificações TL 9000, sendo a primeira empresa brasileira a conquistar este tipo de certificação que se somou à certificação ISO 9001 conquistada também pela empresa. Entretanto não são todos os setores que possuem estas certificações, podendo ter setores específicos que possuem uma ou mais certificações.

6.1.5.1 TL 9000

A certificação TL 9000 é a principal e mais importante certificação de qualidade do setor de telecomunicações, contendo critérios específicos e rigorosos que definem padrões internacionais de qualidade na área de telecomunicações. (DIGITRO TECNOLOGIA, 2015k)

A partir de 2006 a empresa conquistou a certificação e ingressou em seletor e restrito grupo de empresas mundiais que se caracterizam por apresentarem as melhores práticas em gestão de telecomunicações. (DIGITRO TECNOLOGIA, 2015k)

6.1.5.2 ISO 9000

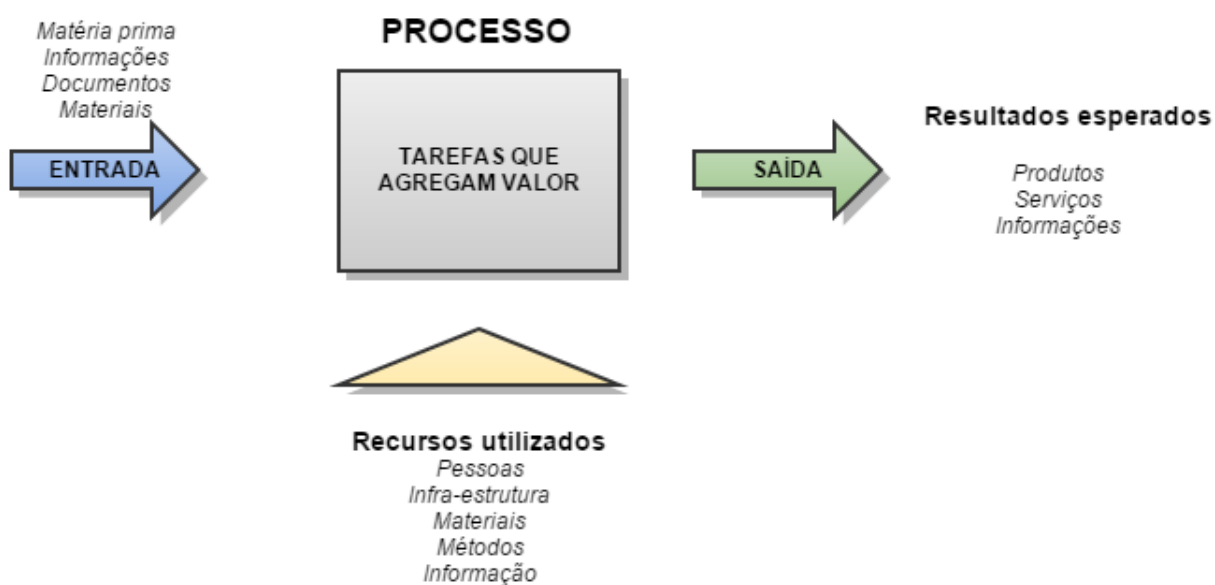
A partir de 1996 que a política de qualidade da empresa é reconhecida pelos órgãos certificados da ISO, são feitas auditorias periódicas que incentivam o ciclo de melhorias continuadas, com o objetivo de atingir um grau de excelência cada vez maior. (DIGITRO TECNOLOGIA, 2015k)

6.2 MODELAGEM DE PROCESSOS

Por ser uma empresa de tecnologia a Dígitro (2015c) necessita estar sempre em constante aperfeiçoamento para atender as demandas dos clientes e mercado, e para isto, necessita ter processos bem estruturados e definidos.

A figura abaixo ilustra como é feito de maneira geral, a abordagem aos processos internos da empresa.

Figura 30 - Abordagem de processos na Dígitro Tecnologia.



Fonte: Dígitro Tecnologia (2015c)

A seguir veremos os processos de desenvolvimento de software, customização e manutenção de software legado, no seu modelo *As Is*, suas devidas atividades e os atores envolvidos em cada um dos processos.

6.2.1 Modelo atual do processo de customização do software legado, “*As is*”

A seguir será exibido o processo no seu atual momento, do desenvolvimento deste trabalho, conforme literatura o processo “*As is*”.

6.2.1.1 Atores do processo “As Is”

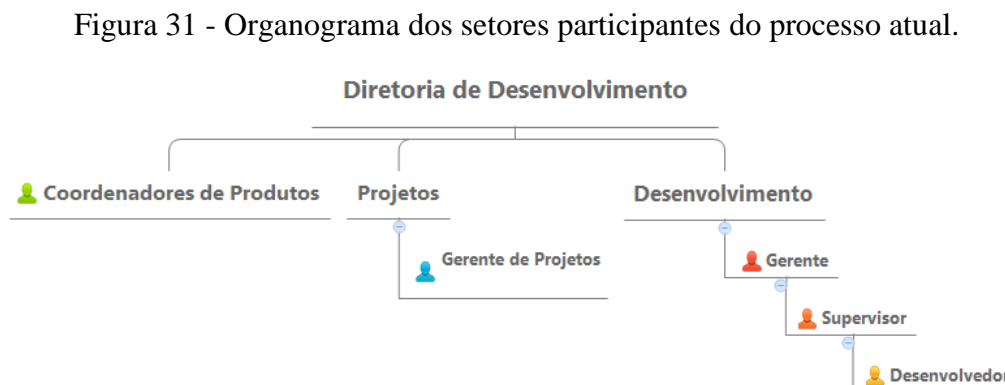
Para o entendimento do processo de customização do software legado da empresa é fundamental entender a hierarquia e os autores dentro da área de desenvolvimento de software da empresa.

Os autores que participam dos processos no momento “As is” são:

- Da Diretoria
 - Coordenador do Produto.
- Do Projeto
 - Gerente de projetos.
- Da Arquitetura
 - Arquiteto.
- Do Desenvolvimento
 - Gerente.
 - Supervisor.
 - Desenvolvedor.

Cada produto, seja ele Pabx, Easy Call ou Contact Center, possui um coordenador de produto responsável por gerir este produto com novas *features*, estes coordenadores estão subjugados a área de Tecnologia. Todo o produto possui um gerente de projetos subjugado na área de Projetos e uma equipe de desenvolvedores pertencentes à área de Desenvolvimento, que possuem uma supervisão e gerência diretamente ligada a eles.

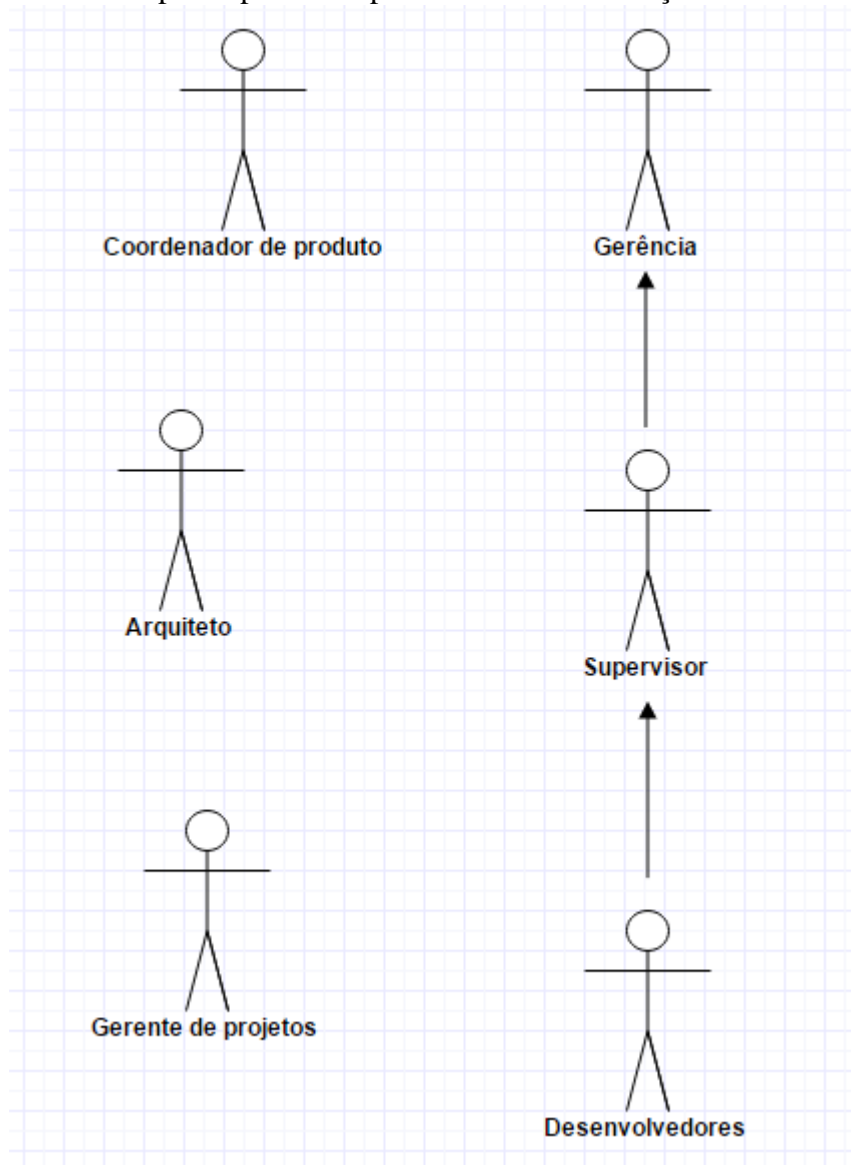
A figura abaixo ilustra o organograma dos atores principais no processo de desenvolvimento.



Fonte: Autor (2015).

A figura abaixo ilustra todos os participantes deste processo.

Figura 32 - Atores participantes do processo de customização de software legado.

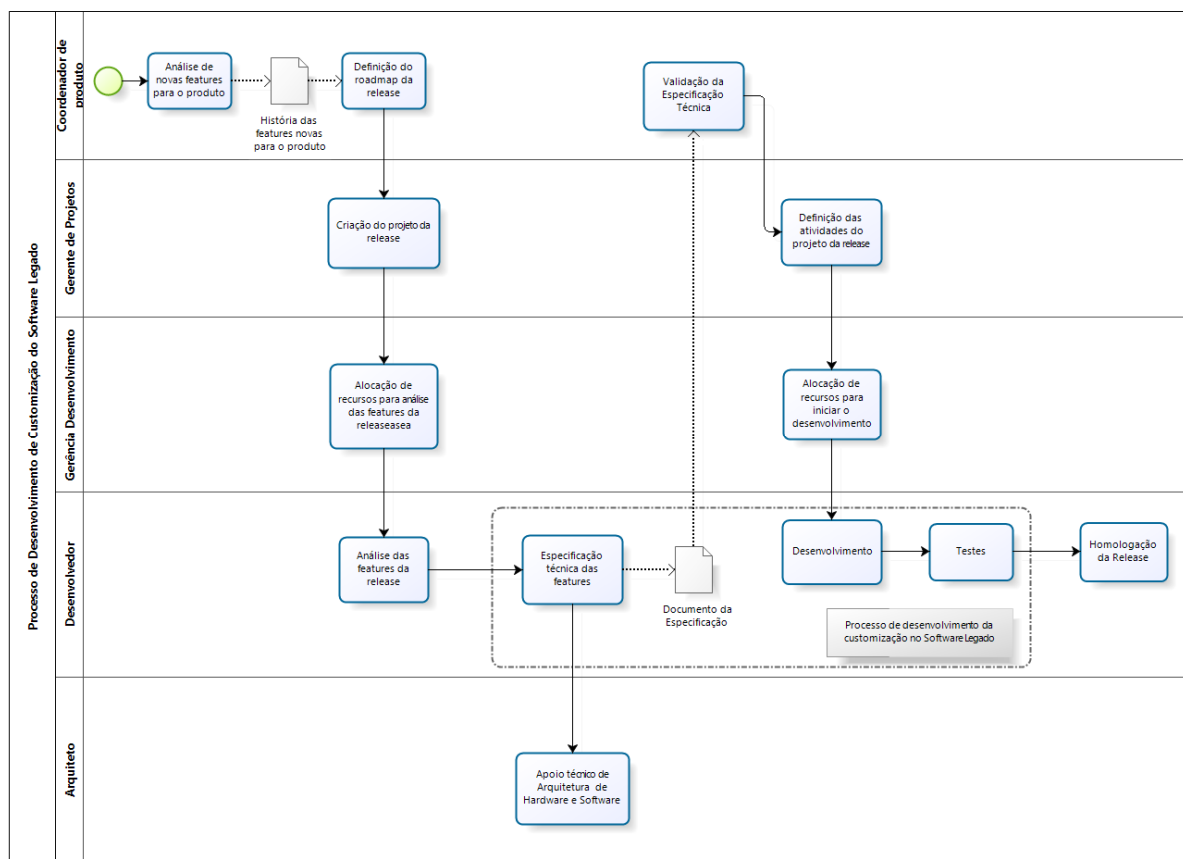


Fonte: Autor (2015).

6.2.1.2 O processo “As Is”

A figura a seguir ilustra as atividades do processo de customização de software legado.

Figura 33 - O processo de customização do software legado “As Is”.



Fonte: Autor (2015)

Conforme a figura, ou o modelo “As is”, as atividades do processo de customização de software legado são detalhadas a seguir.

O primeiro passo do processo de desenvolvimento da customização do software legado começa com a figura do coordenador de produto, responsável por uma solução ou produto, como por exemplo, o Easy Call, solução de *call center* da empresa.

A origem de um novo desenvolvimento seja ele um novo projeto, uma nova customização ou um novo produto se dá pela visão do coordenador do produto em ver uma nova oportunidade de negócio para a empresa ou uma demanda de um cliente que já utiliza uma das soluções.

A partir desta visão de mercado, surge do coordenador de produto uma história ou requisito do que será feito, seja ela uma nova solução, customização ou simplesmente uma manutenção como melhoria no produto já existente.

Após a criação deste novo requisito o coordenador de produto determina em qual *release* deve entrar esta nova *feature* e qual o escopo terá esta *release*.

O segundo passo é o gerente de projetos criar o projeto do *roadmap* da *release*, ele contém todas as informações das histórias que a *release* possui assim como suas atividades, tarefas, marcos e etc.

O terceiro é a gerência do desenvolvimento em conjunto com a supervisão avaliar a disponibilidade de recursos para desenvolver a história e alocar os devidos recursos para especificar a história definindo a especificação técnica e custos.

O quarto passo seria a especificação técnica da história que o desenvolvedor realizar com apoio do arquiteto, nesta etapa cada desenvolvedor cria um tipo de especificação definindo o que será desenvolvido e o custo do desenvolvimento.

O quinto passo é a validação da especificação técnica junto com o coordenador de produto, nesta etapa é verificado se aquilo que foi especificado está de acordo com o que foi pedido pelo coordenador de produto.

O sexto passo ocorre após a validação da especificação técnica, ele consiste na criação das atividades de desenvolvimento, relacionadas à aquela história no projeto do *roadmap* da *release*.

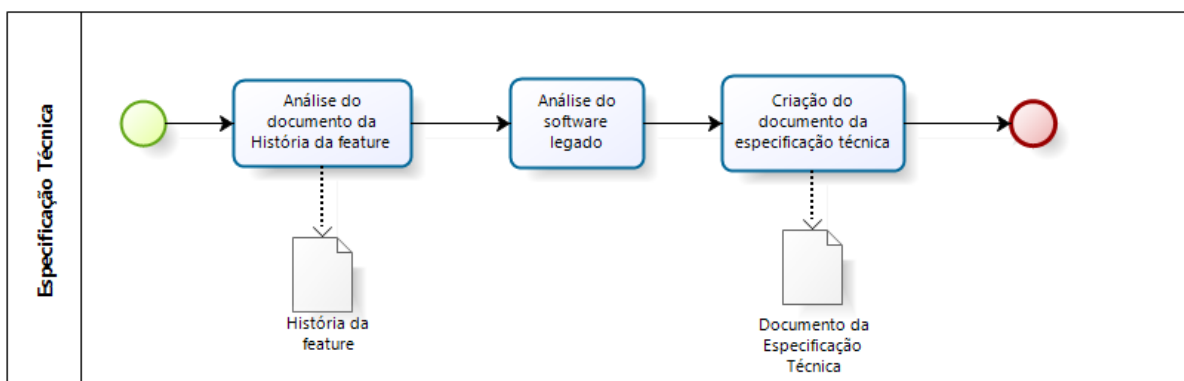
O sétimo passo é o desenvolvimento daquilo que foi especificado e posteriormente a entrega para equipe de homologação poder homologar o produto.

6.2.1.2.1 O processo de customização de software “As Is”

Conforme figura 33 do processo de customização do software legado, o subprocesso especificação técnica das *features*, desenvolvimento e testes estão contornados por um quadro destacando que estes são os processos de desenvolvimento cruciais dentro do processo de desenvolvimento da customização do software legado.

A figura abaixo ilustra o subprocesso de especificação técnica no seu modelo atual “As Is”.

Figura 34 – O processo de especificação técnica da customização do software legado.



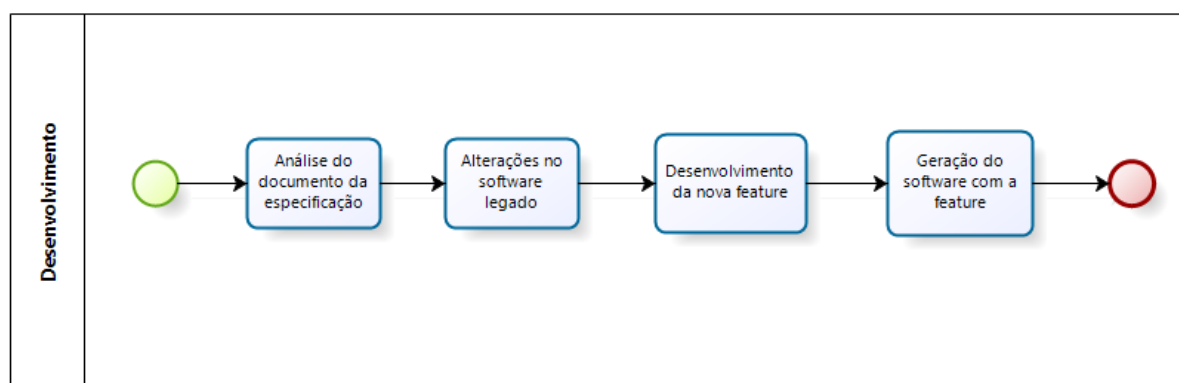
Fonte: Autor (2015)

Nesta etapa o desenvolvedor recebe a história da *feature* do coordenador de produto e a partir dela cria um documento da especificação contendo todas as informações necessárias para o desenvolvimento da solução requisitada pelo coordenador de produto.

Este documento contém uma descrição técnica do que deverá ser feito, quais alterações serão feitas no software e o custo delas para ser desenvolvido a *feature* requisitada pelo coordenador de produto.

Depois da validação deste documento com o coordenador de produto, o desenvolvedor começa o desenvolvimento da *feature* no software legado, seguindo as especificações técnicas descrita no documento da especificação da *feature*, conforme ilustrado na figura abaixo.

Figura 35 – O processo de desenvolvimento da customização do software legado.



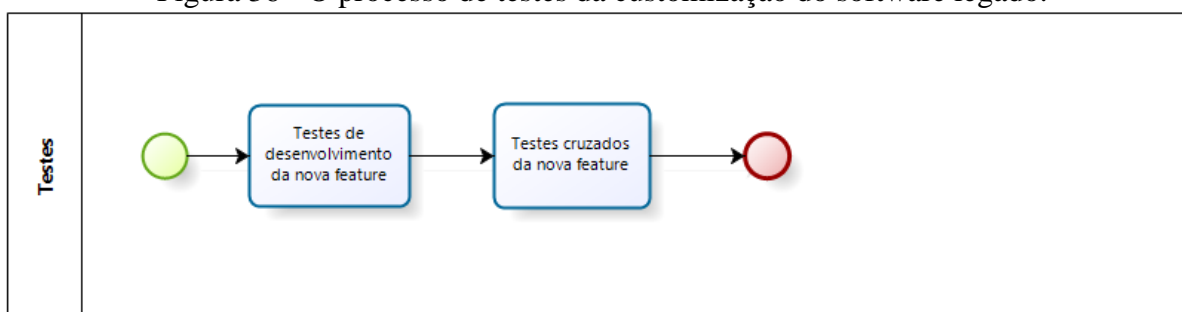
Fonte: Autor (2015)

Após o desenvolvimento da *feature* no software legado existe uma etapa de testes no software legado a fim de assegurar a qualidade do desenvolvimento feito. Estes testes são

manuals e cruzados feitos por desenvolvedores da área, depois de feito estes testes o software segue para outra equipe responsável pela homologação do produto.

A figura abaixo ilustra a etapa de testes da *feature* que foi desenvolvida no software legado.

Figura 36 - O processo de testes da customização do software legado.



Fonte: Autor (2015)

6.2.1.3 Comparativo do processo “As Is” ao modelo MPS.BR nível G

A seguir será apresentado se o processo atual no modelo “As Is” atende os requisitos do nível G do modelo de qualidade MPS.BR, que é composto pelos processos de gerência de projetos e gerência de requisitos.

6.2.1.3.1 Comparativo do processo “As Is” ao processo gerência de projetos

O quadro abaixo ilustra um quadro mostrando se a empresa atende aos resultados esperados dos requisitos do nível G da gerência de projetos do modelo de qualidade MPS.BR.

Quadro 20 - Atendimento do processo atual de customização de software legado "As Is" aos requisitos da gerência de projetos do nível G do MPS.BR

REQUISITO	RESULTADO ESPERADO	ATENDE	PARCIALMENTE	NÃO ATENDE
GPR 1	O escopo do trabalho para o projeto é definido.	X		
GPR 2	As tarefas e os produtos de trabalho do projeto são dimensionados utilizando métodos apropriados;	X		
GPR 3	O modelo e as fases do ciclo de vida do projeto são definidos;	X		
GPR 4	O esforço e o custo para a execução das tarefas e dos produtos de trabalho são estimados com base em dados históricos ou referências técnicas;	X		
GPR 6	O orçamento e o cronograma do projeto, incluindo a definição de marcos e pontos de controle, são estabelecidos e mantidos;	X		
GPR 7	Os riscos do projeto são identificados e o seu impacto, probabilidade de ocorrência e prioridade de tratamento são determinados e documentados;	X		
GPR 8	Os recursos humanos para o projeto são planejados considerando o perfil e o conhecimento necessários para executá-lo;	X		
GPR 9	(Até o nível F) Os recursos e o ambiente de trabalho necessários para executar o projeto são planejados;	X		
GPR 11	Os dados relevantes do projeto são identificados e planejados quanto à forma de coleta, armazenamento e distribuição. Um mecanismo é estabelecido para acessá-los, incluindo, se pertinente, questões de privacidade e segurança;	X		
GPR 12	Um plano geral para a execução do projeto é estabelecido com a integração de planos específicos;	X		
GPR 15	A viabilidade de atingir as metas do projeto é explicitamente avaliada considerando	X		

	restrições e recursos disponíveis. Se necessário, ajustes são realizados;			
GPR 16	O Plano do Projeto é revisado com todos os interessados e o compromisso com ele é obtido e mantido;	X		
GPR 17	O escopo, as tarefas, as estimativas, o orçamento e o cronograma do projeto são monitorados em relação ao planejado;	X		
GPR 18	Os recursos materiais e humanos bem como os dados relevantes do projeto são monitorados em relação ao planejado;	X		
GPR 19	Os riscos são monitorados em relação ao planejado;	X		
GPR 20	O envolvimento das partes interessadas no projeto é planejado, monitorado e mantido;	X		
GPR 21	Revisões são realizadas em marcos do projeto e conforme estabelecido no planejamento;	X		
GPR 22	Registros de problemas identificados e o resultado da análise de questões pertinentes, incluindo dependências críticas, são estabelecidos e tratados com as partes interessadas;	X		

Fonte: Autor (2015)

6.2.1.3.2 Comparativo do processo “As Is” ao processo gerência de requisitos

O quadro a seguir ilustra se os requisitos da gerência de requisitos do nível G do modelo de qualidade MPS.BR são atendidos pela empresa case.

Quadro 21 - Atendimento do processo de customização de software legado "As Is" aos requisitos gerência de requisitos do nível G do MPS.BR

REQUISITO	RESULTADO ESPERADO	ATENDE	PARCIALMENTE	NÃO ATENDE
GRE 1	O entendimento dos requisitos é obtido junto aos fornecedores de requisitos;			X
GRE 2	Os requisitos são avaliados com base em critérios objetivos e um comprometimento da equipe técnica com estes requisitos é obtido;		X	
GRE 3	A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida;	X		
GRE 4	Revisões em planos e produtos de trabalho do projeto são realizadas visando identificar e corrigir inconsistências em relação aos requisitos;	X.		
GRE 5	Mudanças nos requisitos são gerenciadas ao longo do projeto.	X.		

Fonte: Autor (2015)

6.2.2 Modelo atual do processo de manutenção do software legado, “As is”

A seguir será exibido o processo no seu atual momento, conforme literatura “As is”.

6.2.2.1 Atores do processo “As Is”

Para o entendimento do processo de manutenção do software legado da empresa é fundamental entender a hierarquia e os autores dentro da área de manutenção de software da empresa.

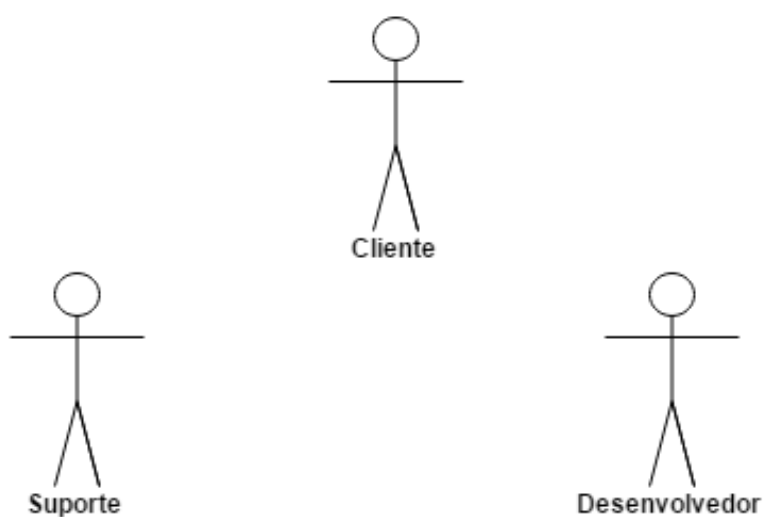
Os atores que participam dos processos no momento “As is” são:

- Cliente

- Do Suporte Técnico
 - Analista de Suporte
- Do Desenvolvimento
 - Desenvolvedor.

A figura abaixo ilustra os participantes envolvidos no processo de manutenção do software legado.

Figura 37 – Atores participantes do processo de manutenção de software legado.

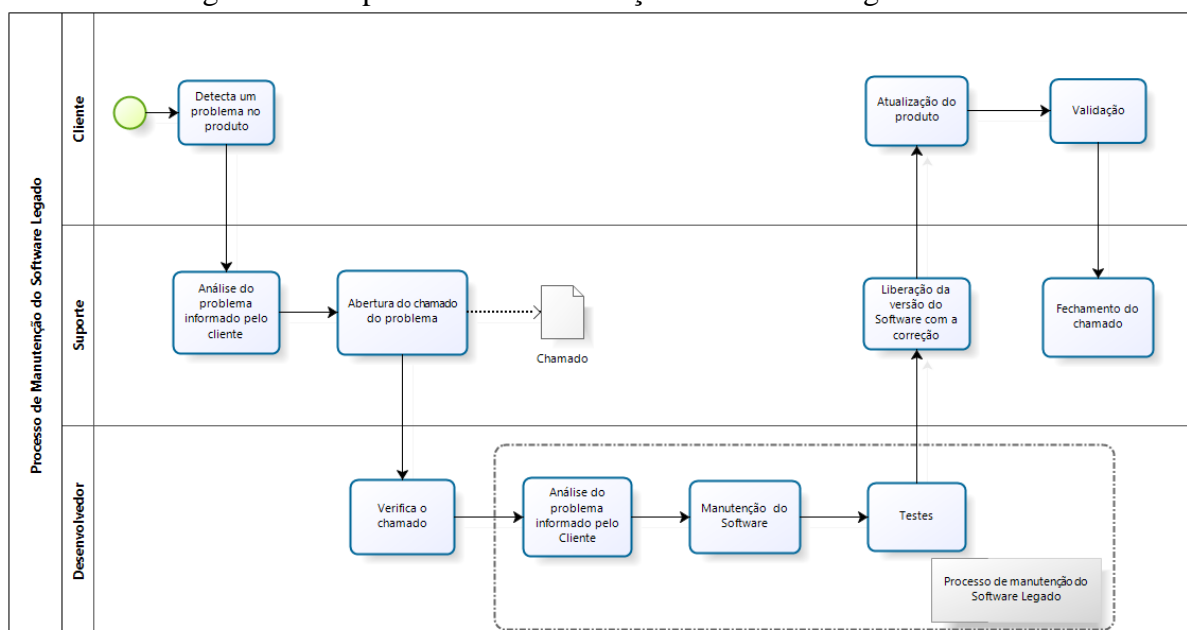


Fonte: Autor (2015).

6.2.2.2 O processo “As Is”

A figura na continuação ilustra as atividades do processo de customização de software legado.

Figura 38 – O processo de manutenção de software legado “As Is”.



Fonte: Autor (2015)

Conforme a figura acima, ou o modelo “As is”, as atividades do processo de manutenção de software legado são:

O primeiro passo do processo de manutenção do software legado começa com a figura do cliente que adquiriu um software e está com problemas na utilização do mesmo, a figura cliente entra em contato com o suporte da empresa informando a ocorrência do problema buscando uma solução para o seu software.

O suporte, ao identificar a ocorrência de problema em conjunto com o cliente abre um chamado para o desenvolvedor. Neste chamado, que é um protocolo, consta toda a descrição do problema encontrado pelo cliente, qual a versão do software e a arquitetura da solução existente no cliente.

O segundo passo é o desenvolvedor ao receber o chamado, analisar a solução e arquitetura do cliente e fazer uma identificação do problema ocorrido verificando a existência de *BUGs*.

O terceiro passo, caso tenha sido constatada a existência de *BUGs*, é o registro do problema encontrado no cliente e a realização da manutenção no software legado, corrigindo o problema identificado no cliente.

O quarto passo é a validação da correção do problema encontrado no cliente, são feitos testes manuais, tentando simular o problema encontrado no cliente em máquinas de testes da empresa.

Após a validação dos testes internos o desenvolvedor libera uma versão do software legado contendo a correção para o suporte da empresa que entra em contato com o cliente agendando uma atualização do produto a fim de solucionar o problema ocorrido.

O quinto passo é a validação da correção em campo, ou seja, o suporte faz um check-list geral da solução atualizada no cliente e em conjunto com este faz testes a fim de validar a manutenção realizada.

Após a validação da manutenção é feito o *feedback* do cliente para o suporte que registra no chamado que a atualização foi feita com sucesso e a correção do problema no cliente foi solucionada.

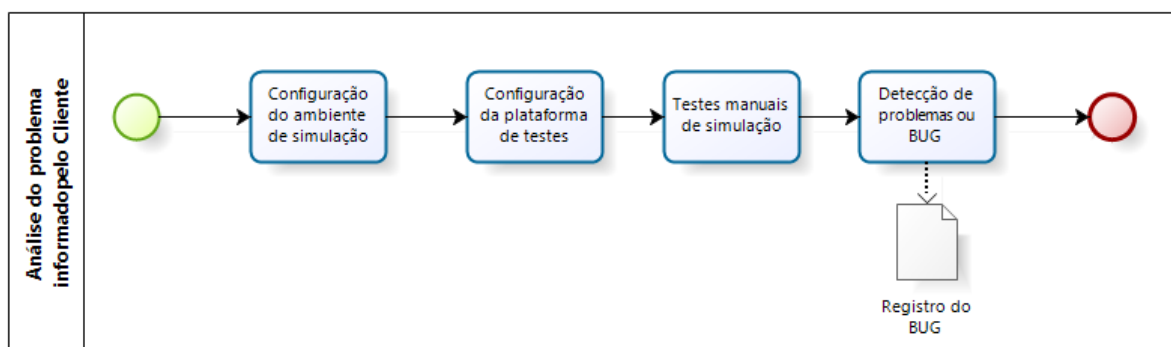
Por fim o desenvolvedor fecha a providência complementar interna assim como o *BUG* registrando esta correção em releases futuras do produto.

6.2.2.2.1 O processo de manutenção de software “As Is”

Conforme figura 38 do processo de manutenção de software legado, o subprocesso análise do problema informado pelo cliente, manutenção do software e testes estão contornado por um quadro destacando que este é o processo de desenvolvimento crucial dentro do processo de desenvolvimento da customização do software legado.

A figura abaixo ilustra o processo de análise do problema relatado pelo cliente no seu modelo atual “As Is”.

Figura 39 - O processo de análise do problema informado pelo cliente.



Fonte: Autor (2015)

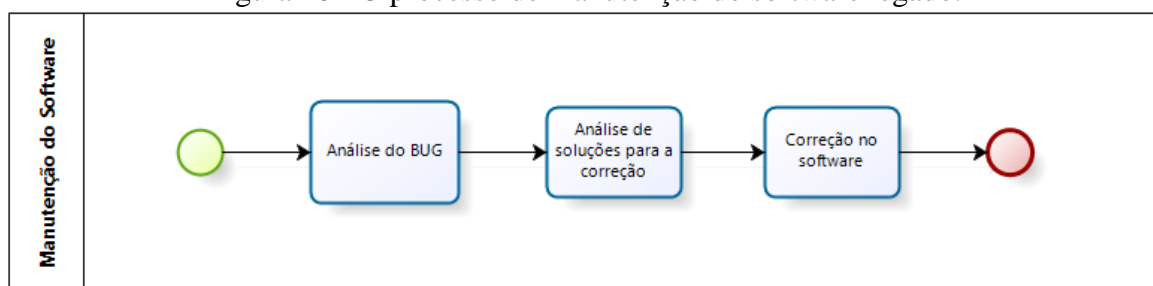
Nesta etapa o desenvolvedor recebe o problema relatado pelo cliente pelo suporte da empresa, ao receber o problema o desenvolvedor realiza uma configuração de um ambiente para a simulação do problema a fim de identificar se realmente existe um problema de software, ou seja, um *BUG*.

Caso seja detectado um problema de software é registrado um *bug* em uma ferramenta de registro de *bugs*, após a detecção do problema é relatado para o suporte que de fato o problema existe e então é estimado um prazo para a manutenção do problema encontrado.

Depois de registrado o *bug* começa a manutenção no software legado a fim de corrigi-lo, então é feito uma análise do problema e das possíveis soluções para as correções. Após a escolha da solução é feito a correção no software legado.

A figura abaixo ilustra o processo de manutenção de software no seu modelo atual “As Is”.

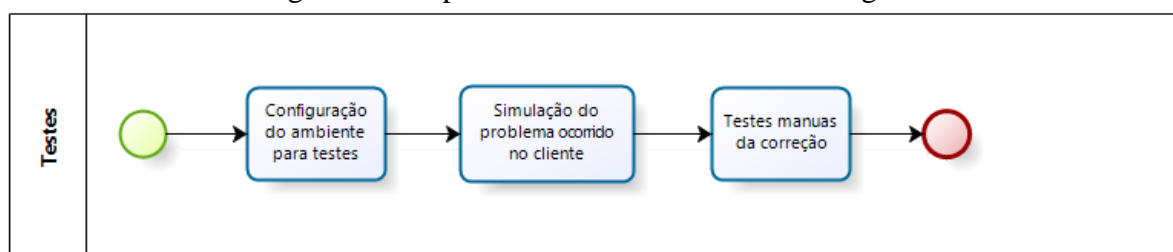
Figura 40 - O processo de manutenção do software legado.



Fonte: Autor (2015)

Depois de corrigido o problema é feito testes manuais para verificar se a correção esta funcionando, etapa esta ilustrada na figura abaixo que representa o processo de testes no seu modelo atual “As Is”.

Figura 41 - O processo de testes do software legado.



Fonte: Autor (2015)

Em certos casos dependendo do *BUG* encontrado, e se a solução for custosa em termos de manutenção ou necessitar grandes alterações no software, o *bug* pode entrar como

um requisito em uma história para ser feito em um release posterior. Neste caso o *bug* entraria como uma *feature* em um *roadmap* e seguiria o fluxo da customização do software legado.

6.2.2.2.2 Comparativo do processo atual “As Is” ao processo gerência de projetos

O quadro abaixo ilustra um atendimento dos resultados esperados dos requisitos do nível G da gerência de projetos do MPS.BR pela empresa case.

Quadro 22 - Atendimento do processo de manutenção de software legado "As Is" aos requisitos gerência de projetos do nível G do MPS.BR

REQUISITO	RESULTADO ESPERADO	ATENDE	PARCIALMENTE	NÃO ATENDE
GPR 1	O escopo do trabalho para o projeto é definido.	X		
GPR 2	As tarefas e os produtos de trabalho do projeto são dimensionados utilizando métodos apropriados;	X		
GPR 3	O modelo e as fases do ciclo de vida do projeto são definidos;	X		
GPR 4	O esforço e o custo para a execução das tarefas e dos produtos de trabalho são estimados com base em dados históricos ou referências técnicas;	X		
GPR 6	O orçamento e o cronograma do projeto, incluindo a definição de marcos e pontos de controle, são estabelecidos e mantidos;	X		
GPR 7	Os riscos do projeto são identificados e o seu impacto, probabilidade de ocorrência e prioridade de tratamento são determinados e documentados;	X		
GPR 8	Os recursos humanos para o projeto são planejados considerando o perfil e o conhecimento necessários para executá-lo;	X		
GPR 9	(Até o nível F) Os recursos e o ambiente	X		

	de trabalho necessários para executar o projeto são planejados;			
GPR 11	Os dados relevantes do projeto são identificados e planejados quanto à forma de coleta, armazenamento e distribuição. Um mecanismo é estabelecido para acessá-los, incluindo, se pertinente, questões de privacidade e segurança;	X		
GPR 12	Um plano geral para a execução do projeto é estabelecido com a integração de planos específicos;	X		
GPR 15	A viabilidade de atingir as metas do projeto é explicitamente avaliada considerando restrições e recursos disponíveis. Se necessário, ajustes são realizados;	X		
GPR 16	O Plano do Projeto é revisado com todos os interessados e o compromisso com ele é obtido e mantido;	X		
GPR 17	O escopo, as tarefas, as estimativas, o orçamento e o cronograma do projeto são monitorados em relação ao planejado;	X		
GPR 18	Os recursos materiais e humanos bem como os dados relevantes do projeto são monitorados em relação ao planejado;	X		
GPR 19	Os riscos são monitorados em relação ao planejado;	X		
GPR 20	O envolvimento das partes interessadas no projeto é planejado, monitorado e mantido;	X		
GPR 21	Revisões são realizadas em marcos do projeto e conforme estabelecido no planejamento;	X		
GPR 22	Registros de problemas identificados e o resultado da análise de questões pertinentes, incluindo dependências críticas, são estabelecidos e tratados com as partes interessadas;	X		

Fonte: Autor (2015)

6.2.2.2.3 Comparativo do processo “As Is” ao processo gerência de requisitos

O quadro abaixo ilustra os requisitos da gerência de requisitos do nível G do modelo de qualidade MPS.BR, verificando se são atendidos pela empresa case.

Quadro 23 – Atendimento do processo de manutenção de software legado "As Is" aos requisitos gerência de requisitos do nível G do MPS.BR

REQUISITO	RESULTADO ESPERADO	ATENDE	PARCIALMENTE	NÃO ATENDE
GRE 1	O entendimento dos requisitos é obtido junto aos fornecedores de requisitos;			X
GRE 2	Os requisitos são avaliados com base em critérios objetivos e um comprometimento da equipe técnica com estes requisitos é obtido;			X
GRE 3	A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida;			X
GRE 4	Revisões em planos e produtos de trabalho do projeto são realizadas visando identificar e corrigir inconsistências em relação aos requisitos;			X
GRE 5	Mudanças nos requisitos são gerenciadas ao longo do projeto.			X

Fonte: Autor (2015)

6.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foi apresentado o estudo de caso sobre a empresa Dígitro Tecnologia, descrevendo seu histórico, soluções e seus processos de customização e manutenção de software no seu modelo *As Is*. O próximo capítulo descreve a proposta de melhoria para os processos de customização e manutenção de software, no seu modelo *To Be*.

7. A PROPOSTA

Neste capítulo será apresentada uma proposta de processo para customização e manutenção de software idealizado pelo Autor. A partir dessa proposta são induzidas melhorias nos processos de customização e manutenção de software da empresa, referenciando-se no embasamento teórico apresentado e utilizando uma metodologia.

7.1 DESCRIÇÃO DA PROPOSTA

Visando o aperfeiçoamento dos processos de manutenção e customização de software, o autor propõe um novo processo a ser utilizado. Serão adotados métodos, procedimentos e ferramentas para atender ao nível G do modelo de qualidade do MPS.BR, que servirá como base para os novos processos propostos pelo autor.

É detalhado como a proposta atende ao requisito de gerência de requisitos do MPS.BR nos processos de customização e manutenção de software, pois a empresa case já atende todos os resultados esperados do processo de gerência de projetos em seu ambiente organizacional.

A seguir será descrito os métodos, ferramentas e procedimentos adotados conforme figura 42.

Métodos.

- *TDD – Test-Drive Development.*
- *BDD – Behaviour-Driven Development.*

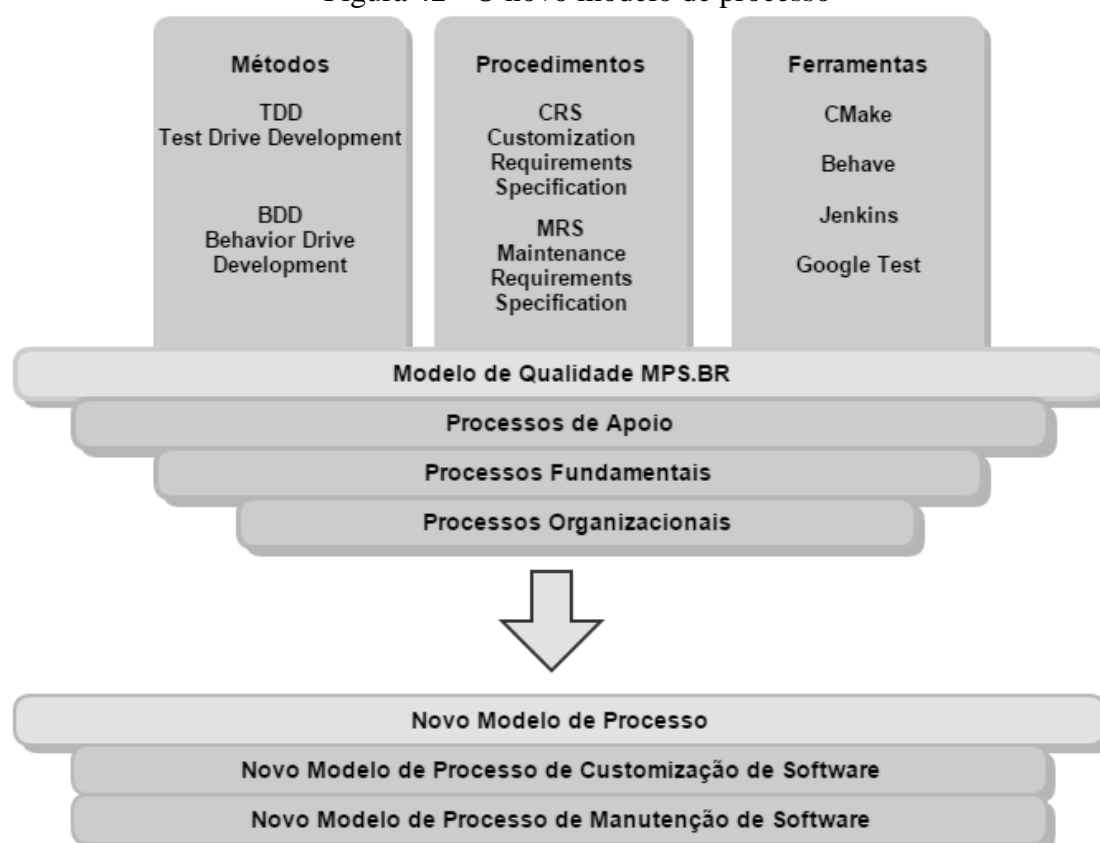
Procedimentos.

- *Documento CRS – Customization Requirements Specification.*
- *Documento MRS – Maintenance Requirements Specification.*

Ferramentas.

- CMake, Behave, Jenkins, Google Test.

Figura 42 - O novo modelo de processo



Fonte: Autor (2015)

O novo processo de software utilizara novos métodos, procedimentos e ferramentas de forma a ser contemplado pelo modelo de qualidade do MPS.BR, gerando assim um novo processo geral que será composto pelos novos processos de customização de software e manutenção de software.

A seguir são descritos detalhadamente os métodos, procedimentos, ferramentas e por fim a modelagem dos novos processos de customização e manutenção de software proposto pelo o autor.

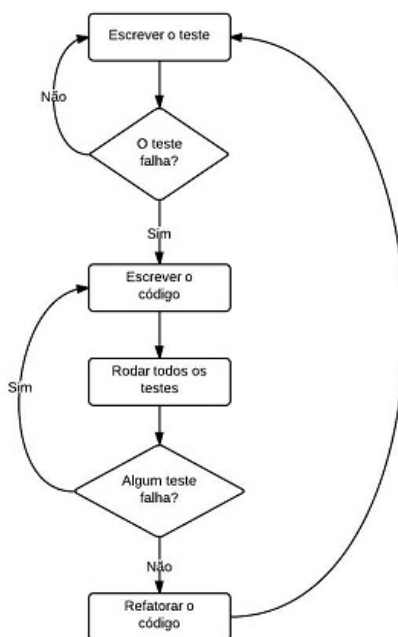
7.1.1 Métodos

Os métodos utilizados no novo modelo de processo são os modelos *TDD* (*Test-Drive Development*) e uma variação dele o *BDD* (*Behavior-Driven Development*) ambos servem de apoio para os testes de software.

7.1.1.1 TDD – Test-Drive Development

É uma técnica de desenvolvimento de software em que ao contrário do que habitualmente o desenvolvedor este acostumado, primeiro são desenvolvido os testes do software para só depois produzir o código em si. Ele se baseia em ciclos de repetição, nos quais para cada funcionalidade do sistema um teste é criado, como começamos desenvolvendo o teste primeiro ele acaba falhando, porém ao começarmos implementando a funcionalidade o teste acaba passando. (ANICHE, 2014)

Figura 43 - Ciclo desenvolvimento do TDD.



Fonte: O Autor (2015)

7.1.1.1.1 Benefícios

A seguir são apresentados os benefícios em se utilizar o TDD na proposta do autor.

- Rápido feedback nos testes da nova funcionalidade sendo desenvolvida e das existentes.
- Código limpo e menos acoplado.
- Facilidade na manutenção de código, como a refatoração e correção de bugs
- Simplificação na integração de módulos.
- Servir como documentação, dando detalhes da funcionalidade implementada.

7.1.1.2 BDD – Behaviour-Drive Development

O *Behavior-Drive Development* é uma metodologia ágil de desenvolvimento de software que encoraja a colaboração entre os desenvolvedores e os *stakeholders*, criada por Dan North em 2003 inclui as práticas do *TDD*.

Dan North (BEHAVIOR, tradução nossa, 2015) define metodologia como.

Behavior-Drive Development é a segunda geração com várias entradas e saídas, múltiplos stakeholders, múltiplas escalas e alta automatização e uma metodologia ágil. É descrito um ciclo de interação com saídas bem definidas resultando na entrega de um software bem trabalhado e testado.

O BDD combina as principais técnicas do TDD com idéias de domínio e análise de orientação a objetos provendo um desenvolvimento de software e gerenciamento de equipe com um processo e ferramentas que colabora com o processo de desenvolvimento do software. (BEHAVIOUR, 2015)

O BDD foca em obter um entendimento claro do software que esta sendo desenvolvido através de uma discussão com os *stakeholders*. É estendido ao TDD escrevendo casos de testes em linguagem natural que mesmo quem não é programador possa entender. (BEHAVIOR, 2015)

7.1.1.2.1 The Gherkin language

BDD é largamente facilitada utilizando uma simples *DSL*, *Domain specific language*, ou seja, uma linguagem de domínio específica chamada *Gherkin Language*.

Figura 44 – A Gherkin language do BDD.

```
Scenario: New lists are empty
  Given a new list
    then the list should be empty.

Scenario: Lists with things in them are not empty.
  Given a new list
    when we add an object
    then the list should not be empty.
```

Fonte: Behavior (2015)

7.1.1.2.2 Benefícios

A seguir são apresentados os benefícios em se utilizar o BDD na proposta do autor.

- Estabelecer metas de diferentes *stakeholders* para uma visão implementada.
- Envolver *stakeholders* durante o processo de implementação.
- Facilitar testes de funcionalidade, unitários e de integração do software.
- Mantém a documentação viva.

7.1.2 Procedimentos

A seguir serão apresentados os procedimentos adotados para o novo processo, estes procedimentos visam adequar a qualidade dos processos.

7.1.2.1 CRS – Customization Requirements Specification

O autor propõe a adoção do *CRS* (*Customization Requirements Specification* ou Especificação de requisitos de customização). Este é um documento que visa descrever a customização ou a *feature* que será realizada no software. Este documento idealizado pelo autor serve como base para o desenvolvedor poder realizar o seu trabalho e atende aos resultados esperados na gerência de requisitos do nível G do modelo de qualidade do MPS.BR.

Este documento se encontra no apêndice B deste trabalho e é descrito na imagem a seguir.

Figura 45 - Template do CRS - Customization Requirements Specification.

CRS – Customization Requirements Software	
• NOME DA CUSTOMIZAÇÃO	INTRODUÇÃO DA CUSTOMIZAÇÃO.
• Motivação	Motivação da customização.
• Descrição	Descritivo da nova funcionalidade
• Requisitos	Requisitos da nova funcionalidade
• Funcionais	Requisitos funcionais.
• Não Funcionais	Requisitos não funcionais
• Regras de negócio	Regras de negócio.
• Testes	Descritivos dos testes e sua abrangência.
• Testes unitários	Testes unitários das classes novas.
• Testes de funcionalidade	Testes de funcionalidade da nova customização.
• Testes de integração	Testes de integração com os demais módulos do software.
• Diagramas	Diagramas, classe, casos de uso, sequencia, etc.
• Correlatos	Impactos em outras funcionalidades.

Fonte: Autor (2015)

7.1.2.2 MRS – Maintenance Requirements Specification

O autor propõe a adoção do *MRS (Maintenance Requirements Specification* ou Especificação de requisitos de manutenção). Este é um documento que visa descrever a manutenção que será realizada no software. Esse documento, idealizado pelo autor serve como base para o desenvolvedor poder realizar o seu trabalho e atende ao resultados esperados na gerência de requisitos do nível G do modelo de qualidade do MPS.BR.

Este documento se encontra no apêndice C deste trabalho e é descrito na imagem a seguir.

Figura 46 - Template do MRS – Maintenance Requirements Specification.

MRS – Maintenance Requirements Specification	
•NOME DA MANUTENÇÃO OU IDENTIFICAÇÃO DO BUG	INTRODUÇÃO DO BUG.
•Descrição do problema	Motivação da manutenção.
•Descrição da solução	Descritivo da solução.
•Testes	Descritivos dos testes e sua abrangência.
•Testes unitários	Testes unitários das classes.
•Testes de integração	Testes de integração com os demais módulos do software.
•Testes específicos da correção	Testes específicos para validação da correção.
•Correlatos	Impacto em outras funcionalidades.

Fonte: Autor (2015)

7.1.3 Ferramentas

A seguir serão apresentadas as ferramentas utilizadas no novo modelo de processo, estas ferramentas servem como apoio para os processos de manutenção e customização de software.

7.1.3.1 CMake

O CMake, é uma ferramenta *open-source* que gerencia o processo de *build* em sistemas operacionais e compiladores de maneira independente, é um sistema *cross-plataform*, ou seja, permite a automação de um processo de *build* em diferentes sistemas operacionais e compiladores. (CMAKE, 2015)

Esta ferramenta foi designada para ser usado em junção com o ambiente de *build* possuindo arquivos de configurações simples. Ele pode gerar um ambiente de desenvolvimento nativo que irá compilar o código fonte, criar as bibliotecas e gerar os executáveis. (CMAKE, 2015)

7.1.3.1.1 Benefícios

A seguir são apresentados os benefícios em se utilizar o CMake na proposta definida pelo autor.

- Controlar a compilação do software, utilizando plataforma e compiladores diferentes.
- Gera automaticamente *makefiles* nativos e *workspaces*.
- Permite a automatização entre diferentes sistemas operacionais e compiladores.
- Facilita a automatização de todo o processo de *build* da aplicação.

7.1.3.2 Behave

O Behave é ferramenta de linha de execução em *Python* que utiliza a metodologia *BDD*, *Behaviour-Driven Development*, utilizando linguagem natural definida pela *BDD* a *Gherkin Language* para fazer os cenários de testes.

Esta ferramenta baseada em uma *API* em Python serve para a execução dos casos de testes contendo o Behave, os arquivos de *steps* e *features*. O arquivo *features* contém os casos de testes que serão executados pelo Behave, e são em linguagem *Gherkin* conforme figura abaixo.

Figura 47 - Exemplo de um arquivo feature.

```
Scenario: test something
  Given some known state
  then some observed outcome.
```

Fonte: Behavior (2015)

7.1.3.2.1 Benefícios

A seguir são apresentados os benefícios em se utilizar o CMake na proposta.

- É uma API em Python que é executada em linha de comando, facilitando o desenvolvimento de casos de testes rápidos que exigem integração com diferentes tipos de aplicação.
- Vantagem em utilizar a linguagem de programação em Python para utilizar a metodologia BDD.
- Uma ferramenta enxuta e rápida para executar a metodologia BDD para testes.

7.1.3.3 Jenkins

O Jenkins é uma ferramenta *open source* de integração contínua desenvolvida na linguagem de programação Java. Usando jenkins é possível montar e testar os projetos de software continuamente, facilitando aos desenvolvedores a integração das mudanças do projeto e a obtenção de uma nova cópia atualizada do software. (JENKINS, 2015)

7.1.3.3.1 *Benefícios*

A seguir são apresentados os benefícios em se utilizar o Jenkins na proposta definida pelo autor.

- Fácil instalação, Jenkins roda em Java em um container servlet.
- Fácil configuração, a configuração toda pode ser feita via interface.
- Rico em plugins para customização e necessidades dos projetos dos desenvolvedores.

7.1.3.4 Google Test

O Google Test C/C++ Test Framework, é uma biblioteca para realizar testes unitários em projetos desenvolvidos na linguagem de programação C/C++ com intuito de validar o que foi propriamente desenvolvido. (SEN, 2010)

No processo proposto pelo autor são aplicados testes unitários e como o processo é para softwares legados, que na empresa case em grande parte são desenvolvidos na linguagem de programação C/C++, então foi adotado o Google C++ Test framework, como ferramenta para desenvolver os testes unitários no processo.

7.1.3.4.1 *Benefícios*

A seguir são apresentados os benefícios em se utilizar o Google Test na proposta definida pelo autor.

- Simples maneira de rodar os testes unitários.
- Possibilidade de gerar XML com resultados de maneira simples.

7.2 A MODELAGEM

A seguir são descritos os processos propostos para a customização e manutenção de software legado, na sua forma *To be*, com as suas devidas atividades e os atores e ferramentas envolvidas em cada um dos processos.

Figura 48 - Abordagem de processos do novo processo.



Fonte: Autor (2015)

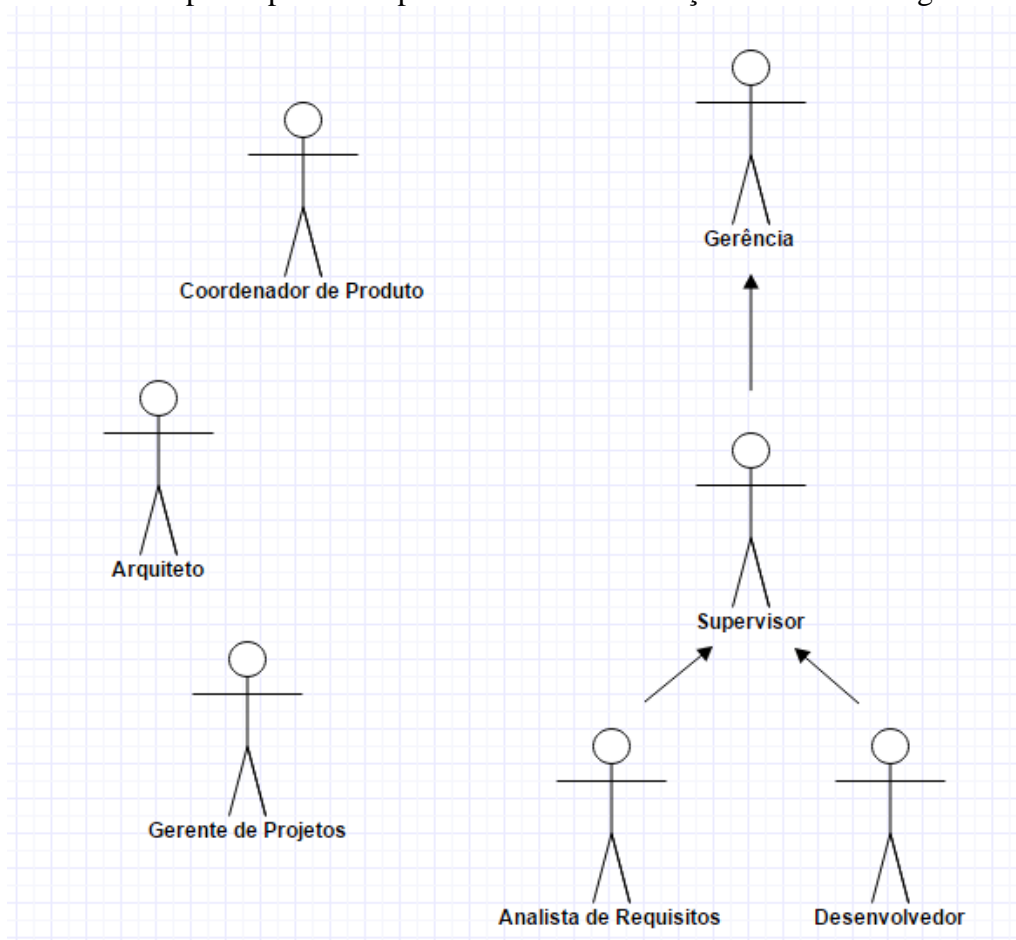
7.2.1 O novo processo de customização do software legado, “*To be*”

A seguir será exibido o novo processo de customização de software conforme o proposto pelo autor, o processo “*To be*”.

7.2.1.1 Atores do processo “To Be”

Os atores que farão parte deste novo processo são os mesmos existentes no modelo “*As Is*”, vistos no capítulo anterior do estudo de caso da empresa. Entretanto surge uma nova figura chamada de *Analista de Requisitos* que pode ou não também ser um desenvolvedor.

Figura 49 – Atores participantes do processo de customização de software legado "*To Be*".



Fonte: Autor (2015)

7.2.1.2 O processo "To Be"

A figura a seguir ilustra as atividades do novo processo de customização de software legado aplicando o modelo proposto pelo autor.

Conforme a figura 50, ou o modelo “To be”, as atividades são as mesmas do modelo “As Is”, porém com mudanças a partir do quarto passo. A seguir são detalhadas as atividades de todo o processo e as mudanças.

O primeiro passo do processo da customização do software legado começa com a figura do coordenador de produto.

A origem de um novo desenvolvimento seja um novo projeto, uma nova customização ou um novo produto se inicia pela visão do coordenador do produto em ver uma nova oportunidade de negócio para a empresa ou uma demanda de um cliente que já utiliza uma das soluções.

A partir desta visão de mercado, surge do coordenador de produto uma história ou requisito do que será feito, uma nova solução, customização ou simplesmente uma manutenção como melhoria no produto já existente.

Após a criação deste novo requisito o coordenador de produto determina em qual *release* deve entrar esta nova *feature* e qual o escopo terá esta *release*.

O segundo passo é o gerente de projetos criar o projeto do *roadmap* da *release*, ele contém todas as informações das histórias que a *release* possui assim como suas atividades, tarefas, marcos e etc.

O terceiro é a gerência do desenvolvimento em conjunto com a supervisão avaliar a disponibilidade de recursos para desenvolver a história e alocar os devidos recursos para especificar a história definindo a especificação técnica e custos.

O quarto passo seria a especificação técnica da história, o analista de requisitos faz a especificação técnica com apoio do arquiteto, nesta etapa o analista de requisito cria o documento *CRS*, que tem toda a especificação técnica da customização, diagramas, casos de testes e etc.

O quinto passo é a validação do *CRS* com o coordenador de produto, esta é a etapa no qual é verificado se aquilo que foi descrito no documento está de acordo com o que foi pedido pelo coordenador de produto.

O sexto passo ocorre após a validação do *CRS*, ela consiste na criação das atividades de desenvolvimento, relacionadas à aquela história no projeto do *roadmap* da *release*.

O sétimo passo é o desenvolvimento daquilo que está no *CRS* e posteriormente a entrega para equipe de homologação poder homologar o produto.

No oitavo passo, após o desenvolvimento, ocorreria uma validação do que foi desenvolvido relacionado ao *CRS* em conjunto com o analista de requisitos tendo em vista a aplicação da metodologia do *BDD*.

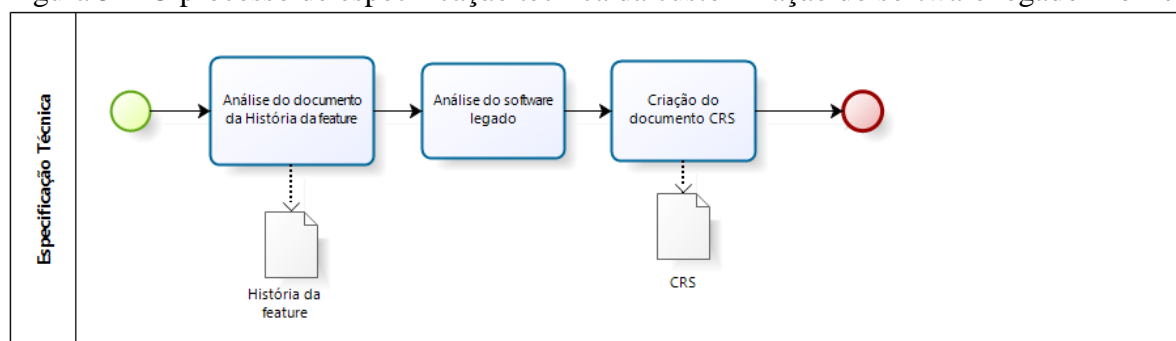
7.2.1.3 O processo de customização de software “*To Be*”

Conforme figura 50 do processo de customização do software legado, os subprocessos “especificação técnica das *features*”, “desenvolvimento”, “testes”, “validação do CRS com o teste” e “validação do CRS com o desenvolvimento” estão contornados por um quadro destacando que estes são os processos de desenvolvimento cruciais dentro do processo da customização do software legado.

No modelo foram adicionados os dois subprocessos: “Validação do CRS com o que foi desenvolvido”, esta atividade é feita em conjunto com o analista de requisitos, e a “Validação do CRS com os testes”. Este último subprocesso consiste nos testes de integração utilizando o *Behave*, que utiliza a metodologia do *BDD*, garantindo que as funcionalidades foram desenvolvidas e testadas de acordo com o CRS.

A figura abaixo ilustra o subprocesso de especificação técnica na forma “*To Be*”.

Figura 51- O processo de especificação técnica da customização do software legado “*To Be*”.

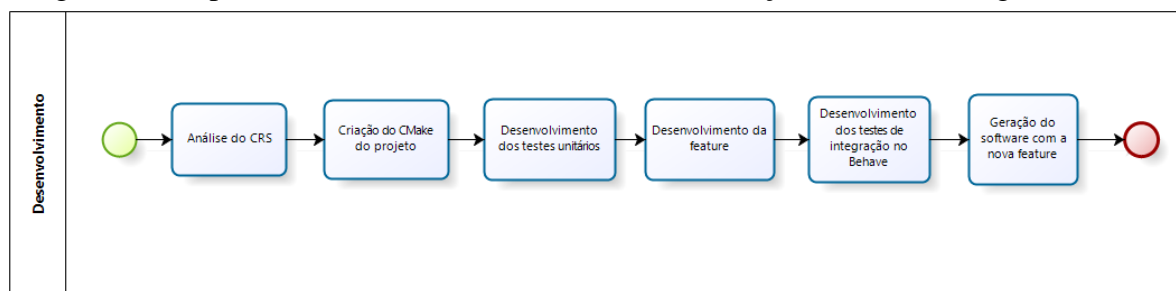


Fonte: Autor (2015)

Nesta etapa o analista de requisitos recebe a história da *feature* do coordenador de produto e a partir dela cria um documento chamado *CRS* – *Customization Requirements Specification*, conforme no apêndice B deste trabalho.

Depois da validação deste documento com o coordenador de produto, o desenvolvedor começa o desenvolvimento da *feature* no software legado, seguindo o CRS, conforme ilustrado na figura a seguir.

Figura 52 - O processo de desenvolvimento da customização do software legado "To Be".



Fonte: Autor (2015)

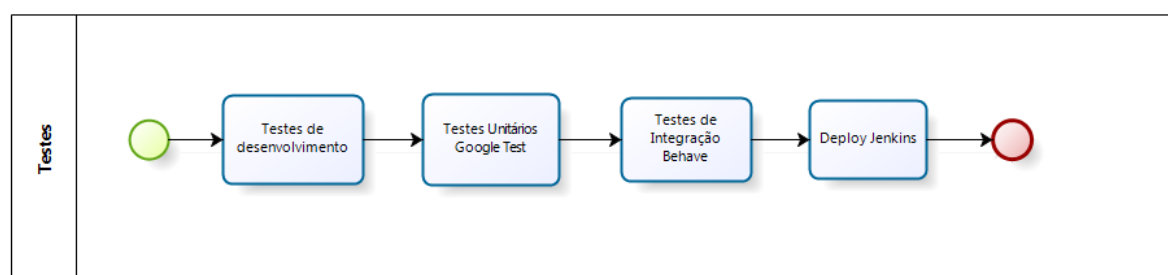
Em um primeiro momento nesta etapa o desenvolvedor faz a análise do *CRS* identificando as alterações que devem ser feitas no software legado, a partir disto começa a desenvolver o projeto no *CMake*. Caso o projeto seja um novo serviço, ou seja, um binário novo que será acoplado na solução, o *CMake* será responsável por criar os *makefiles*, as compilações dos serviços das bibliotecas e afins.

Após a criação do *CMake*, o desenvolvedor começa a criação dos testes unitários aplicando a metodologia *TDD* e então conforme a metodologia vai realizando o desenvolvimento da *feature* em conformidade com os testes unitários.

Após o desenvolvimento da *feature* é feito o desenvolvimento dos testes de integração usando o *Behave* aplicando a metodologia do *BDD*. Estes testes são para testar toda a solução do software desde baixo nível, serviços, por exemplo, até a aplicação web.

Depois de concluído o desenvolvimento dos testes unitários, da *feature* e dos testes de integração e após gerado o software com a nova customização o desenvolvedor passa a realizar uma bateria de testes e fazer o deploy do software no *Jenkins*. Isso tudo para a geração dos relatórios dos testes unitários e de integração do *Behave* para poder validar posteriormente com o analista de requisitos os resultados, conforme ilustrado na figura a seguir.

Figura 53 - O processo de testes da customização do software legado "To Be".

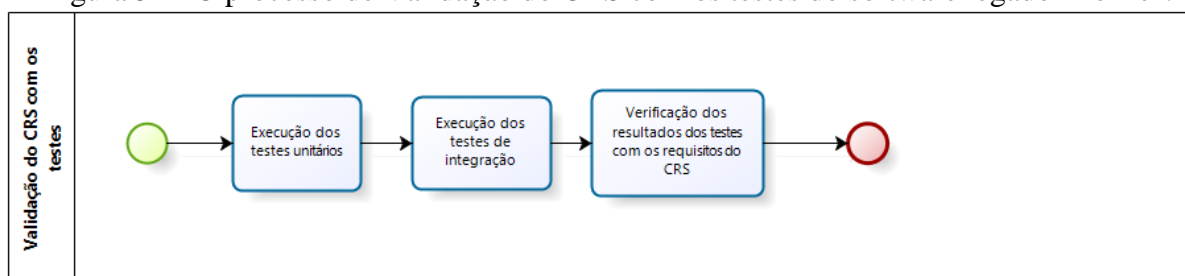


Fonte: Autor (2015)

Após a etapa de testes que servem para garantir que a customização feita no software legado está funcional e que as funcionalidades existentes não foram afetadas ocorre uma etapa de validação do CRS com os testes.

Nesta etapa ocorre a validação se os testes estão de acordo com os testes prescritos no CRS, a figura a seguir ilustra esta etapa.

Figura 54 - O processo de Validação do CRS com os testes do software legado "To Be".

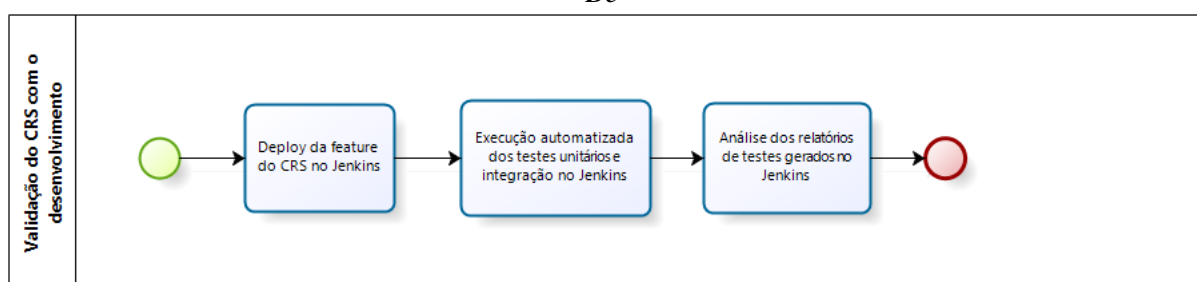


Fonte: Autor (2015)

A última etapa do processo de customização é a etapa em que é feita a validação se o item que foi desenvolvido está de acordo com o especificado no CRS, esta atividade ocorre em conjunto com o desenvolvedor e o analista de requisitos, através da verificação dos relatórios da funcionalidade desenvolvida e dos testes unitários e de integração que foram executados e gerados pelo Jenkins.

A figura abaixo ilustra a etapa de validação do CRS após a conclusão das fases de testes do software legado.

Figura 55 - O processo de validação do CRS com o desenvolvimento do software legado "To Be"



Fonte: Autor (2015)

7.2.1.4 Comparativo do processo “*To Be*” ao modelo MPS.BR nível G

A seguir será apresentado se o processo no modelo “*To Be*” atende os requisitos do nível G do modelo de qualidade MPS.BR, que é composto pelos processos de gerência de projetos e gerência de requisitos.

7.2.1.4.1 *Comparativo do processo “As Is” ao processo gerência de projetos*

Todos os resultados esperados para a gerência de projetos já eram atendidos pelo processo “*As Is*”. Dessa forma, o processo proposto também já contempla todos os requisitos de gerência de projetos.

7.2.1.4.2 *Comparativo do processo “To Be” ao processo gerência de requisitos*

O quadro a seguir ilustra se os resultados esperados da gerência de requisitos do nível G do modelo de qualidade MPS.BR são atendidos pelo novo processo de customização de software legado.

Quadro 24 - Atendimento do processo de customização de software legado "*To Be*" pelos resultados esperados da gerência de requisitos do nível G do MPS.BR

REQUISITO	RESULTADO ESPERADO	ATENDE	PARCIALMENTE	NÃO ATENDE
GRE 1	O entendimento dos requisitos é obtido junto aos fornecedores de requisitos;	Sim, é validado pelos autores envolvidos, desenvolvedor, analista de requisitos e coordenador de produto.		
GRE 2	Os requisitos são avaliados com base em critérios objetivos e um comprometimento da equipe técnica com estes requisitos é obtido;	Sim, o comprometimento é especificado em forma de um documento chamado <i>CRS</i> .		
GRE 3	A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida;	Sim, o documento <i>CRS</i> , pode ter várias versões e conforme o requisito muda pode ser rastreado.		
GRE 4	Revisões em planos e produtos de trabalho do projeto são realizadas visando identificar e corrigir inconsistências em relação aos requisitos;	Sim, o documento <i>CRS</i> , visa atender o que foi desenvolvido atende ao requisito esperado.		
GRE 5	Mudanças nos requisitos são gerenciadas ao longo do projeto.	Sim, o documento <i>CRS</i> é uma forma de documento a funcionalidade do sistema e pode sofrer alterações nas suas versões posteriormente.		

Fonte: Autor (2015)

7.2.2 Modelo atual do processo de manutenção do software legado, “*To Be*”

A seguir será exibido o processo na sua forma “*To Be*”.

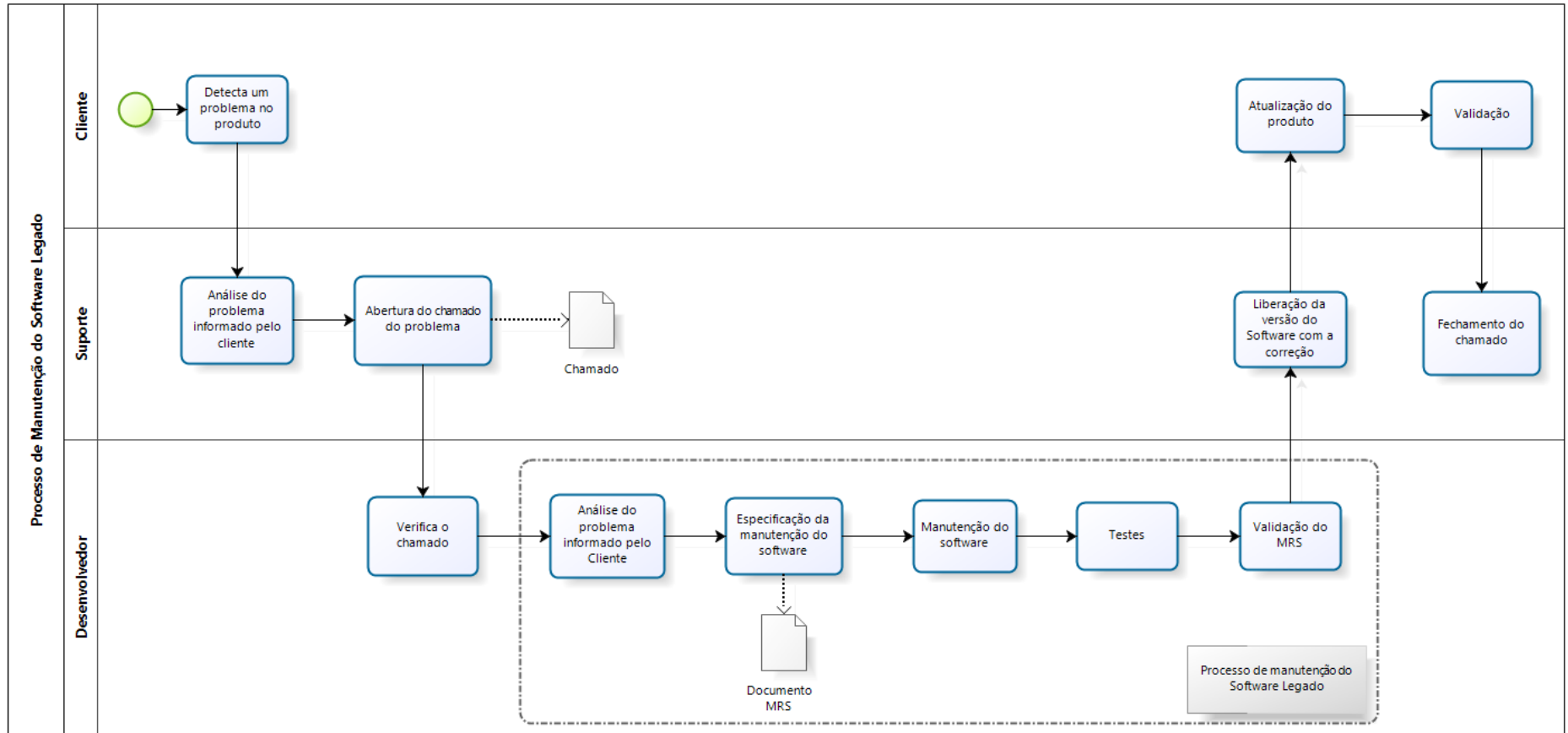
7.2.2.1 Atores do processo “*To Be*”

Os atores que farão parte deste novo processo são os mesmos existentes no modelo “*As Is*”, conforme a figura 37 e vista no capítulo anterior do estudo de caso da empresa.

7.2.2.2 O processo “*To be*”

A figura a seguir ilustra as atividades do processo de manutenção do software legado.

Figura 56 - O processo de manutenção de software legado "To Be".



Fonte: Autor (2015).

Conforme a figura anterior, ou o processo “To Be”, as atividades são as mesmas do processo “As Is”, porém com mudanças a partir do terceiro passo.

O primeiro passo do processo de manutenção do software legado começa quando a figura cliente entra em contato com o suporte da empresa informando a ocorrência do problema buscando uma solução para o problema.

O suporte, ao identificar a ocorrência de problema abre um chamado para o desenvolvedor no qual consta toda a descrição do problema encontrado pelo cliente.

O segundo passo é o desenvolvedor, ao receber a providência, analisar a solução e arquitetura do cliente e fazer uma identificação do problema ocorrido verificando a existência de *BUGs*.

O terceiro passo, caso tenha sido constatada a existência de *BUGs*, é o registro do mesmo e a especificação da manutenção do software. Nesta etapa é feita a identificação do *BUG* e sua correção, a definição dos testes unitários, de integração e da correção do *BUG* formalizando todas essas informações no documento MRS que é criado.

O quarto passo é a manutenção do software em si, nesta etapa é realizada a correção do problema encontrado.

O quinto passo consiste nos testes do que foi corrigido no software, utilizando testes unitários, de integração e testes específicos com o objetivo testar a correção do *BUG*.

O sexto e último passo é a validação da manutenção com o MRS, ou seja, os testes sendo executados no Jenkins analisando o relatório final dos testes, a fim de garantir que o que este escrito no MRS foi atendido e as funcionalidades existentes no software não foram quebradas.

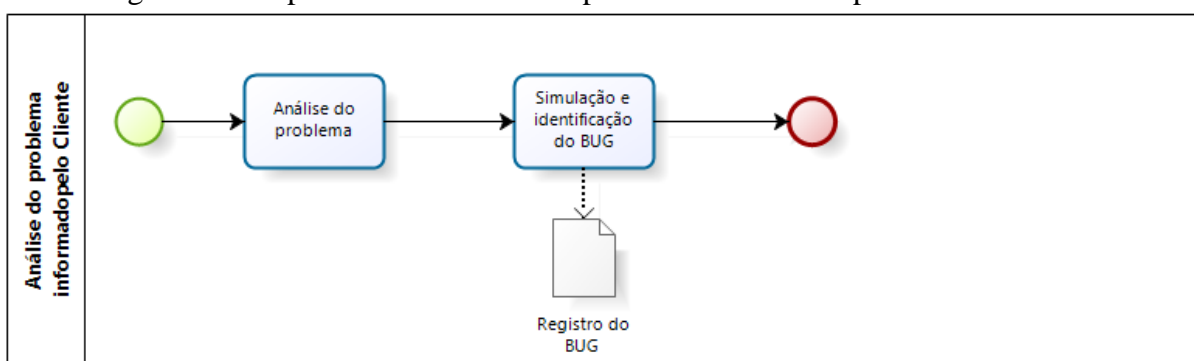
7.2.2.3 O processo de manutenção de software “To Be”

Conforme figura 57 do processo de manutenção do software legado, os subprocessos “Análise do problema informado pelo Cliente”, “Especificação da Manutenção do Software”, “Manutenção do Software”, “Testes” e “Validação do MRS” estão contornados por um quadro destacando que estes são os processos de desenvolvimento cruciais dentro do processo da manutenção do software legado.

No modelo foram adicionados os dois subprocessos: “Especificação da Manutenção do Software”, que é a especificação e a criação do documento *MRS* (*Maintenance Requirements Specification*) e “Validação do MRS”. Além disso, foram feitas diversas mudanças nos processos existentes a fim de melhorar o processo de manutenção do software legado.

A figura abaixo ilustra o subprocesso da “Análise do problema informado pelo Cliente” no seu modelo atual “*To Be*”.

Figura 57 – O processo de análise do problema informado pelo cliente “*To Be*”.

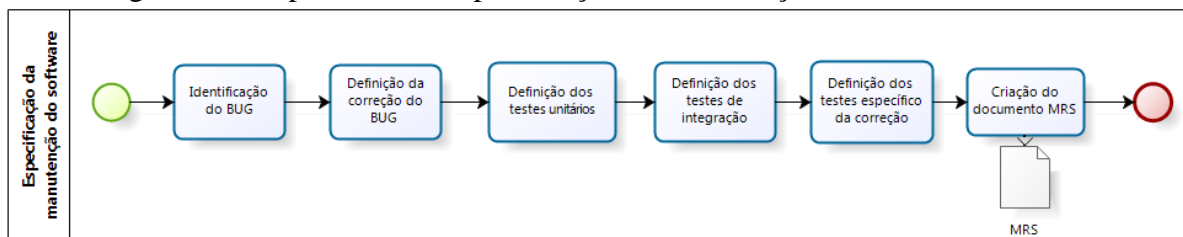


Fonte: Autor (2015).

Nesta etapa o desenvolvedor recebe o problema do suporte e faz uma análise do problema que está ocorrendo no cliente, caso seja efetivamente um problema de software o desenvolvedor abre um *BUG* no processo de manutenção no software legado.

Depois da identificação do problema, o desenvolvedor começa a especificação da manutenção do software, realizando a definição da correção do problema, a definição dos testes unitários, de integração, e específico para a correção do problema. Depois de feita esta especificação cria o documento chamado *MRS*, contendo esta especificação. A figura a seguir ilustra este processo de “Especificação da manutenção do software”.

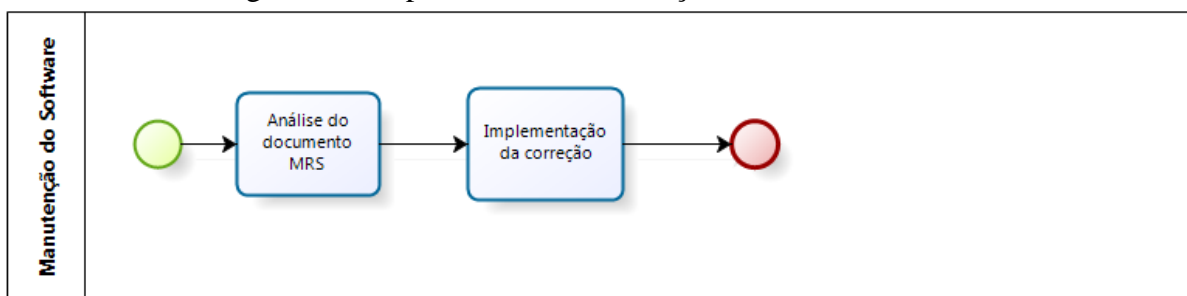
Figura 58 – O processo de especificação da manutenção do software “*To Be*”.



Fonte: Autor (2015).

Após a etapa de especificação do software, o desenvolvedor está apto a realizar a manutenção do software com as informações contidas no documento *MRS*, nesta etapa começa a implementação da correção no software legado, conforme figura a seguir.

Figura 59 – O processo de manutenção do software "*To Be*".

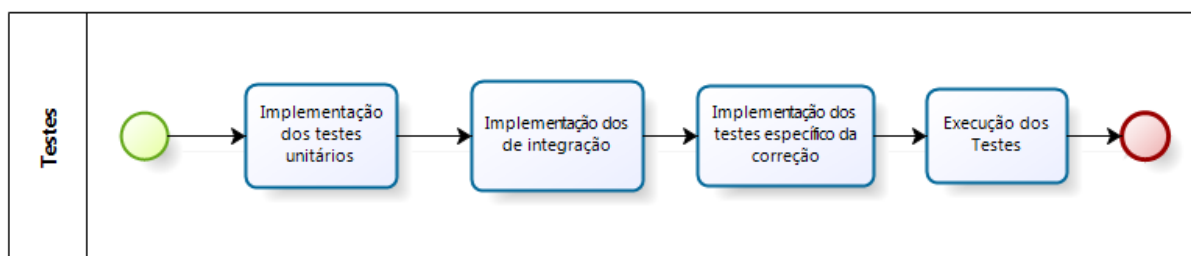


Fonte: Autor (2015).

Realizada a manutenção no software legado, começa a realização dos testes, neste processo é feita a implementação dos testes unitários e de integração caso aquela funcionalidade específica do software legado que está com problemas não tenha testes unitários ou de integração será implementado estes testes. Isto é muito comum em software legados que em geral são concluídos sem testes unitários e de integração.

Após esta etapa é feita a implementação de testes e cenários específicos para testar a correção que foi realizada na manutenção do software. Os testes unitários podem ser feitos seguindo a metodologia do TDD e os testes de integração utilizam a metodologia do BDD. A figura a seguir ilustra o processo de testes da manutenção do software.

Figura 60 – O processo de testes da manutenção do software "*To Be*".



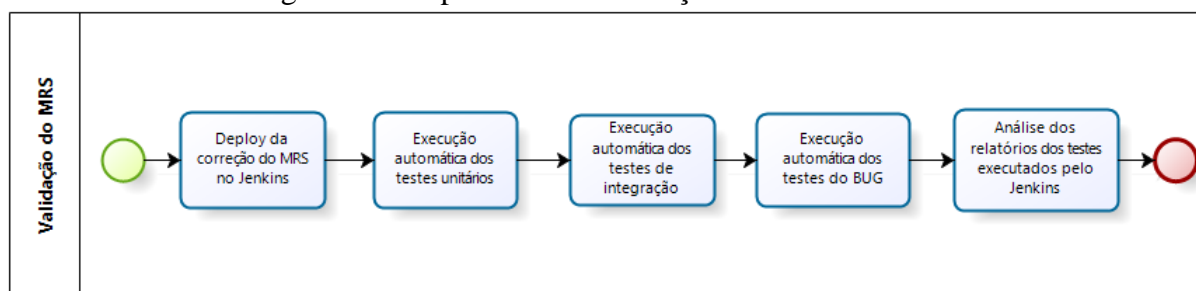
Fonte: Autor (2015).

Após a realização dos testes e a execução deles, começa o processo de validação do MRS. Nesta etapa é realizado o deploy da correção do software na ferramenta do Jenkins, que irá executar de forma automatizada os testes unitários e de integração utilizando as

tecnologias do *Google Test* e *Behave* afim de garantir que a correção foi realizada com sucesso e não teve efeitos colaterais em outras funcionalidades do software.

A figura abaixo ilustra este processo de validação do MRS no processo de manutenção do software legado.

Figura 61 – O processo de validação do MRS “*To Be*”.



Fonte: Autor (2015).

7.2.2.4 Comparativo do processo “*To Be*” ao modelo MPS.BR nível G

A seguir será apresentado se o processo no modelo “*To Be*” atende os requisitos do nível G do modelo de qualidade MPS.BR, que é composto pelos processos de gerência de projetos e gerência de requisitos.

7.2.2.4.1 Comparativo do processo “*As Is*” ao processo gerência de projetos

Todos os resultados esperados para a gerência de projetos já eram atendidos pelo processo “*As Is*”. Dessa forma, o processo proposto também já contempla todos os requisitos de gerência de projetos.

7.2.2.4.2 Comparativo do processo “To Be” ao processo gerência de requisitos

O quadro abaixo ilustra se os resultados esperados da gerência de requisitos do nível G do modelo de qualidade MPS.BR são atendidos pelo novo processo de manutenção do software legado.

Quadro 25 - Atendimento do processo de manutenção do software legado "To Be" pelos resultados esperados da gerência de requisitos do nível G do MPS.BR

REQUISITO	RESULTADO ESPERADO	ATENDE	PARCIALMENTE	NÃO ATENDE
GRE 1	O entendimento dos requisitos é obtido junto aos fornecedores de requisitos;	Sim, é validado pelos autores envolvidos, desenvolvedor, analista de requisitos e coordenador de produto.		
GRE 2	Os requisitos são avaliados com base em critérios objetivos e um comprometimento da equipe técnica com estes requisitos é obtido;	Sim, o comprometimento é especificado em forma de um documento chamado <i>MRS</i> .		
GRE 3	A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho é estabelecida e mantida;	Sim, o documento <i>MRS</i> , pode ter várias versões e conforme o requisito muda pode ser rastreado.		
GRE 4	Revisões em planos e produtos de trabalho do projeto são realizadas visando identificar e corrigir inconsistências em relação aos requisitos;	Sim, o documento <i>MRS</i> , visa atender o que foi desenvolvido atende ao requisito esperado.		
GRE 5	Mudanças nos requisitos são gerenciadas ao longo do projeto.	Sim, o documento <i>MRS</i> é uma forma de documento a funcionalidade do sistema e pode sofrer alterações nas suas versões posteriormente.		

Fonte: Autor (2015)

7.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foi apresentada uma proposta de melhoria nos processos da empresa estudo de caso. Essas melhorias são novos processos na customização e manutenção de software, no seu modelo *To Be*, utilizando metodologias, documentos e ferramentas propostas pelo autor.

8. CONCLUSÃO

Finalizando, este capítulo apresenta as conclusões deste trabalho, suas considerações gerais, contribuições e seus trabalhos futuros.

8.1 CONSIDERAÇÕES GERAIS

A proposta deste trabalho foi compreender na empresa *case* o fenômeno da manutenção e customização de software legados e as dificuldades para executar tarefas ligadas à manutenção de software e manter a qualidade do software legado. E partindo desta premissa propor uma mudança nos processos de manutenção e customização de software seguindo ferramentas, metodologias, procedimentos e de acordo com o modelo de qualidade MPS.BR nível G afim de facilitar os trabalhos relacionados a manutenção e aumentar a qualidade do software legado.

As atividades de manutenção e customização de software são as mais custosas no processo de evolução de software, portanto estas merecem atenção especial por parte das empresas quando o software legado ainda esta sendo comercializado e faz parte do faturamento da empresa.

Levando em consideração a importância em se manter este tipo de software, investir em um processo especializado para a manutenção e customização de software pode reduzir custos em retrabalhos de manutenção, mão de obra, erros de projeto e aumentar consideravelmente a qualidade das manutenções e customizações aumentando gradativamente a qualidade do software legado.

Atualmente é notável que a engenharia de software possua grande foco no processo de desenvolvimento de software, contendo vários modelos a serem seguidos além da existência de modelos de qualidade para serem aplicados no processo de desenvolvimento. Partindo de uma euforia inicial por desenvolvimento e formas de melhorar o desenvolvimento de software como novas ferramentas e metodologias e algumas vezes propostas inovadoras facilitando o desenvolvimento e até mesmo a própria evolução de software.

O fato é que, se percebe o surgimento de um volume grande de softwares em pleno funcionamento pelo mundo e que precisam continuar funcionando por tempo indeterminado, provendo a vitalidade deste legado que a evolução da computação produziu, porém a engenharia de software sofre de carência quando o assunto é a evolução do software, ou seja, a manutenção do software propriamente dita.

Verificando isto se percebe a carência da engenharia de software em fornecer metodologias e procedimentos, especializados na evolução do software, assim como da área acadêmica para tratar o assunto da manutenção de software.

8.2 CONTRIBUIÇÕES

A realização deste trabalho possibilitou conhecer um pouco da história da empresa Dígitro Tecnologia, seus produtos e serviços, estrutura organizacional e os processos de customização e manutenção de software do setor de desenvolvimento de software da empresa.

Durante o desenvolvimento do estudo de caso feito pelo autor, percebeu-se a preocupação da empresa em investir em gerência de projetos, requisitos e na abordagem dos seus processos a fim de aumentar a qualidade nos seus softwares e principalmente processos de desenvolvimento de software.

As contribuições deste trabalho incluem a apresentação e estudo dos fatos relacionados à manutenção de software, principalmente seus problemas e as dificuldades das organizações em manter os softwares legados e a realização de uma proposta de melhoria nos processos de manutenção e customização dos softwares legados na empresa.

Esta proposta apesar de não ter sido aplicada na empresa, foi utilizada pelo autor em alguns projetos de trabalho, assim sendo percebeu-se algumas facilidades na sua utilização. Essas vantagens na sua aplicação são descritas a seguir.

- Os procedimentos fornecem uma documentação viva do projeto e a integração da equipe de desenvolvimento com o entendimento das funcionalidades,
- As metodologias do TDD e BDD fornecem facilidade na realização das customizações e manutenção dos softwares.
- Essas metodologias também provêm os benefícios das ferramentas para realização de testes utilitários e de integração, reduzindo custos com as atividades de manutenção.

- A adoção do Jenkins fornece sempre uma versão do software testado, funcionando e com qualidade para ser utilizado.

Outra grande contribuição é o fato de qualificar as atividades de manutenção em conformidade com o modelo de qualidade MPS.BR nível G, atendendo especificamente a gerência de requisitos nas atividades de customização e manutenção, podendo ser um grande diferencial para a empresa *case*.

8.3 TRABALHOS FUTUROS

Como trabalho futuro poderia se pensar na aplicação destas recomendações na empresa *case* e verificação do atendimento do nível G do MPS.BR pela organização.

Outra proposta de continuidade para este trabalho também é a análise, verificação e geração de recomendações para se atingir algum dos outros níveis superiores do MPS.BR.

O trabalho futuro mais evidente é a criação de um modelo de processo para as atividades de manutenção e customização de software que use como base as indicações observadas na proposta de melhoria dos processos de customização e manutenção de software apresentadas neste trabalho.

Uma análise mais detalhada destes novos processos, definidos pelo autor, pode apresentar como resultados finais um modelo genérico para as atividades de manutenção de software, seguindo metodologias, procedimentos e ferramentas, ou seja, um modelo tal como os modelos de desenvolvimento de software cascata ou espiral, entre outros.

Este modelo para as atividades relacionadas à evolução do software iria servir como base para empresas que ainda dispõem de comercialização de software legados, podendo ainda ser aplicado um modelo de qualidade neste modelo, tais como MPS.BR, CMMI e etc.

REFERÊNCIAS

ANICHE, Maurício, Mestre em Ciência da Computação, 2014, Disponível em <<http://tdd.caelum.com.br/>> Acesso em 23/09/2015.

AUDY Jorge Luis Nicolas. **Desenvolvimento Distribuído de Software**, Rio de Janeiro, Elsevier 2008.

BRAUDE, Eric **Projeto de Software: Da programação à arquitetura: Uma abordagem baseada em Java**, 1ª, ed , Artmed ed, 2004.

BEHAVIOR, 2015, Disponível em < <http://pythonhosted.org/behave/>> Acesso em 23/09/2015.

CMAKE, 2015, Disponível em < <https://cmake.org/>> Acesso em 28/09/2015.

DÍGITRO TECNOLOGIA, 2015a. Disponível em <<http://www.digitro.com.br/pt/index.php/a-digitro>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015b. Disponível em <<http://www.digitro.com.br/pt/index.php/a-digitro/historia>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015c. Documentos Internos.

DÍGITRO TECNOLOGIA, 2015d. Disponível em <<http://www.digitro.com.br/pt/index.php/solucoes-e-produtos>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015e. Disponível em <<http://www.digitro.com.br/pt/index.php/solucoes-e-produtos/todas-solucoes/pabx>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015f. Disponível em <<http://www.digitro.com.br/pt/index.php/solucoes-e-produtos/todas-solucoes/call-center>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015g. Disponível em <<http://www.digitro.com.br/pt/index.php/solucoes-e-produtos/todas-solucoes/ura>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015h. Disponível em <<http://www.digitro.com.br/pt/index.php/solucoes-e-produtos/todas-solucoes/gravacao>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015i. Disponível em <<http://www.digitro.com.br/pt/index.php/solucoes-e-produtos/todas-solucoes/contact-center>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015j. Disponível em
<<http://www.digitro.com.br/pt/index.php/solucoes-e-produtos/mercados/plataformas>> Acesso em 28/08/2015.

DÍGITRO TECNOLOGIA, 2015k. Disponível em
<<http://www.digitro.com.br/pt/index.php/a-digitro/qualidade-digitro>> Acesso em 28/08/2015.

ENGHOLM, Júnior Hélio. **Engenharia de Software na prática**, São Paulo: Novatec Editora, 2010.

FALBO, Ricardo A. **Integração de Conhecimento em um Ambiente de Desenvolvimento de Software**. Tese de Doutorado, Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

FONSECA, J. J. S. **Metodologia da pesquisa científica**. Fortaleza; UEC, 2002. Apostila.

JENKINS, 2015, Disponível em <<https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins>> Acesso em 29/09/2015.

KOSCIANSKI, André; SOARES Michel dos Santos. **Qualidade de software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. 2ª Ed. São Paulo : Novatec, 2007.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Metodologia do trabalho científico: procedimentos básicos, pesquisa bibliográfica, projeto e relatório, publicações e trabalho científico**. 6ª. ed. São Paulo: Atlas, 2001.

MINAYO, M. C. S. **O desafio do conhecimento. Pesquisa qualitativa em saúde**. São Paulo: HUCITEC, 2007.

PETERS, James F. **Engenharia de Software Teoria e Prática**, 1ª ed, Rio de Janeiro: Campus, 2001.

PADUELLI, M. M. **Manutenção de Software: problemas típicos e diretrizes para uma disciplina específica**, 2007, 144p Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2007.

PRESSMAN, Roger S. **Engenharia de Software Uma Abordagem Profissional**, 7ª Ed, AMGH 2011.

RAUPP, F. M.; BEUREN, I. M. **Metodologia da Pesquisa aplicável às ciências sociais**. In:BEUREN, I. M. (Org.). **Como elaborar trabalhos monográficos em contabilidade: teoria e prática**. 2ª. ed. São Paulo: Atlas, 2004.

REZENDE Denis Alcides, **Engenharia de Software e Sistemas de Informação**, 3ª, ed Rio de Janeiro, Brasport 2005.

SCHWABER Ken; SUTHERLAND Jeff. **Guia do Scrum: Um guia definitivo para o Scrum: As regras do jogo**, 2013. Disponível em
<<http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>>

SEM Arpan, **A quick introduction to the Google C++ Testing Framework: Learn about key features for ease of use and production-level deployment**, 2011.

< <http://www.ibm.com/developerworks/aix/library/au-googletestingframework-pdf.pdf>>

Acesso em 15/10/2015.

SOARES Michel dos Santos, **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**, INFOCOMP, 2004.

< <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>> Acesso em 10 de novembro de 2014.

SOFTEX, **Guias do MPS-BR, 2015**. Disponível em

< <http://www.softex.br/mpsbr/guias/>> Acesso em 01 de junho de 2015.

SOFTEX, **MPS.BR-Guia Geral MPS de Gestão de Pessoas**. 2014, Disponível em

< http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_RH_2014_Versao_Beta-1.pdf> Acesso em 01 de junho de 2015.

SOFTEX, **MPS.BR – Guia Geral MPS de Serviços**, 2012a. Disponível em

< http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Servicos_2012.pdf> Acesso em 01 de junho de 2015.

SOFTEX, **MS-BR – Guia Geral de Software**, 2012b. Disponível em

< http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2012-c-ISBN-1.pdf> Acesso em 01 de junho de 2015.

SOFTEX, **MPS-BR – Guia de Implementação – Parte 1: Fundamentação para Implementação do Nível G do MR-MPS-SW**, 2013. Disponível em

< http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de_Implementacao_Parte_1_2013.1.pdf> Acesso em 02 de junho de 2015.

SOMMERVILLE, IAN. **Engenharia de Software**. 8ª, ed, Pearson Education, 2011.

SCHACH Stephen R. **Engenharia de Software: Os Paradigmas Clássico e Orientação a Objetos**, 7ª ed, 2010.

APÊNDICE A

CRONOGRAMA TCC 2015/2

ATIVIDADE	JUNHO	JULHO	AGOSTO	SETEMBRO	OUTUBRO	NOVEMBRO
Correção cap. 1 e 2						
Estudo de caso						
Proposta						
Correções cap 4 e 5.						
Resumo e Abstract						
Conclusão						
Revisão geral.						
Defesa						

APÊNDICE B

CRS – Customization Requirements Software

- **NOME DA CUSTOMIZAÇÃO**

INTRODUÇÃO DA CUSTOMIZAÇÃO.

- **Motivação**

Motivação da customização.

- **Descrição**

Descritivo da nova funcionalidade

- **Requisitos**

Requisitos da nova funcionalidade

- **Funcionais**

Requisitos funcionais.

- **Não Funcionais**

Requisitos não funcionais

- **Regras de negócio**

Regras de negócio.

- **Testes**

Descritivos dos testes e sua abrangência.

- **Testes unitários**

Testes unitários das classes novas.

- **Testes de funcionalidade**

Testes de funcionalidade da nova customização.

- **Testes de integração**

Testes de integração com os demais módulos do software.

- **Diagramas**

Diagramas, classe, casos de uso, sequencia, etc.

- **Correlatos**

Impacto em outras funcionalidades.

APÊNDICE C

MRS – Maintenance Requirements Software

•NOME DA MANUTENÇÃO OU IDENTIFICAÇÃO DO BUG

INTRODUÇÃO DO BUG.

•Descrição do problema

Motivação da manutenção.

•Descrição da solução

Descritivo da correção.

•Testes

Descritivos dos testes e sua abrangência.

•Testes unitários

Testes unitários das classes.

•Testes de integração

Testes de integração com os demais módulos do software.

•Testes específicos da correção

Testes específicos para validação da correção.

•Correlatos

Impacto em outras funcionalidades.