Pontifícia Universidade Católica
do Rio de Janeiro

Ana Paula Lima Lucas

Gestão da manutenção de software: Um estudo de caso

Dissertação de Mestrado

Dissertação apresentada ao Programa de Pós-Graduação em Informática da PUC-Rio como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Arndt Von Staa

Rio de Janeiro

Setembro de 2016

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Ana Paula Lima Lucas

Gestão da manutenção de software: Um estudo de caso

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Departamento de Informática do Centro Técnico e Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Arndt Von Staa
Orientador
Departamento de Informática – PUC-Rio

Prof. Gustavo Robichez de Carvalho
Departamento de Informática – PUC-Rio

Prof. Julio Cesar Sampaio do Prado Leite
Departamento de Informática – PUC-Rio

Prof. Márcio da Silveira Carvalho
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 27 de Setembro de 2016

Ana Paula Lima Lucas

Graduou-se em Ciência da Computação pela UFF - Universidade Federal Fluminense em 2012. Atua no mercado de desenvolvimento de software como analista de sistemas e pesquisadora. Interessada em aprender sobre novas tecnologias e boas soluções para problemas reais.

Ficha Catalográfica

Lucas, Ana Paula Lima

Gestão da manutenção de software : um estudo de caso / Ana Paula Lima Lucas ; orientador: Arndt Von Staa. – 2016.
104 f. : il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2016.
Inclui bibliografia

1. Informática – Teses. 2. Manutenção de software. 3. Práticas do Modelo SMmm. 4. Processo definido. 5. TFS. I. Staa, Arndt von, 1942-. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Dedico este trabalho aos meus pais,
pelo apoio e confiança.

Agradecimentos

A Deus, em primeiro lugar, que sempre estive comigo, que sabe de meus medos, que fortalece meus sonhos, vigia meus passos e me coloca no colo nos momentos mais difíceis… Sempre presente. Que, com a minha história, eu possa melhorar tua obra.

Aos meus pais, pela educação, atenção e carinho de todas as horas.

Ao meu orientador Professor Arndt Von Staa pela parceria para a realização deste trabalho.

Aos meus colegas da PUC-Rio.

A todos os amigos e familiares que de uma forma ou de outra me estimularam ou me ajudaram.

Resumo

Lucas, Ana Paula Lima; Staa, Arndt Von. **Gestão da manutenção de software: Um estudo de caso.** Rio de Janeiro, 2016. 104p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A organização participante deste trabalho buscou implantar atividades relacionadas à manutenção de software em função de problemas relacionados à grande ocorrência de defeitos, constantes retrabalhos, entre outros. Para suprimir esses problemas, inicialmente, foi elaborado um estudo preliminar do sistema em questão, avaliando o estado atual da manutenção de software. Diante do diagnóstico realizado, ficou evidente a necessidade de mudanças na maneira como eram conduzidas as atividades de manutenção do sistema. Com isso, iniciou-se a busca por alternativas de melhorias com objetivo de reduzir a ocorrência de defeitos e também aumentar a manutenibilidade do sistema. Sabendo quais são os problemas e o que poderia ser feito para melhorar, propôs-se adotar algumas práticas do modelo de maturidade de manutenção de software - SMmm e integrar os conceitos destas práticas em um processo definido e adaptado às necessidades do sistema. Para apoiar essa implementação, a infraestrutura utilizada foi a plataforma Team Foundation Service – TFS que colaborou com a implementação das práticas selecionadas segundo as exigências do modelo SMmm, resultando em um processo definido apoiado pelo TFS que implementa parcialmente o modelo SMmm. Este trabalho apresenta um estudo de caso com o objetivo de avaliar os benefícios proporcionados pela utilização de algumas práticas do modelo SMmm. A avaliação realizada confronta os dados do estudo preliminar com dados coletados após adoção das práticas, os resultados analisados apontaram uma redução significativa da quantidade de defeitos.

Palavras-chave

Manutenção de Software; Práticas do Modelo SMmm; Processo Definido; TFS.

Abstract

Lucas, Ana Paula Lima; Staa, Arndt Von (Advisor). **Software maintenance management: A case study.** Rio de Janeiro, 2016. 104p. MSc. Dissertation - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The company participating in this work sought to implement activities related to software maintenance due to problems related to the great occurrence of defects, constant rework, among others. To suppress these problems, a preliminary study of the system in question has been elaborated, evaluating the current state of software maintenance. In view of the diagnosis, the necessity of changes became evident concerning the way that the system maintenance activities were conducted. With this, the search for alternatives of improvements began with the objective of reducing the occurrence of defects and also increase the maintainability of the system. Knowing the problems and what could be done to improve; it was proposed to accede some practices of the software maintenance maturity model - SMmm and to integrate the concepts of these practices into a defined process and adapted to the needs of the system. To support this implementation, the infrastructure used was the Team Foundation Service - TFS platform that collaborated with the implementation of the selected practices according to the requirements of the SMmm model, resulting in a defined process supported by the TFS that partially implements the SMmm model. This paper presents a case study with the objective of evaluating the benefits provided by the use of some practices of the SMmm model. The evaluation carried out compared the data from the preliminary study with data collected after adoption of the practices, the analyzed results pointed out a significant reduction in the number of issues.

Keywords

Software Maintenance Model; Practices SMmm; Defined Process; TFS.

Sumário

Lista de Figuras

Lista de Tabelas

LISTA DE ABREVIATURAS E SIGLAS

CMMI	*Capability Maturity Model Integration*
SMmm	*Software Maintenance Maturity Model*
MR-MPS	Modelo de Referência para Melhoria de Processo do Software
TFS	*Team Foundation Service*
ISO	Organização Internacional para Padronização
IEC	Comissão Eletrotécnica Internacional
SEI	*Software Engineering Institute*
XP	*eXtreme Programming*
CM3	Modelo de Maturidade de Manutenção Corretiva
GQM	*Goal Question Metric*
MTM	*Microsoft Test Manager*
PR	*Problem Report*
TFVC	*Team Foundation Version Control*
VSTF	*Visual Studio Team Foundation*
WIT	*Work Item Type*
PO	*Product owner*
PBI	*Product backlog Item*
IDE	Ambiente de desenvolvimento integrado

1 Introdução

1.1 Motivação

Quando um software é liberado em um ambiente, precisa se adaptar continuamente às novas necessidades e requisitos desse ambiente para não se tornar obsoleto. Qualquer organização que utilize software observa isso dando margem à lei da continua evolução formulada por [Lehman, 1980]. Ao assegurar elevada manutenibilidade, viabiliza-se que novas funções demandadas pelo ambiente em que ele está inserido sejam implementadas mais facilmente.

O desenvolvimento e a evolução de processos de software constituem linhas de pesquisa importantes para a manutenção de software, para apoiar a busca pelo aperfeiçoamento dos processos de software e pela melhoria da qualidade dos sistemas, a academia tem proposto e a indústria de software tem adotado diversas iniciativas de melhorias de processos de software, representadas principalmente por programas e modelos de maturidade. Segundo [Sommerville, 2011] o objetivo central dessas iniciativas consiste na evolução dos processos de software por meio da análise, da medição e do aperfeiçoamento das práticas de Engenharia de Software, visando também determinar a eficácia das práticas executadas e a maturidade de uma organização na execução de um conjunto de processos. Das principais iniciativas destacam-se o Capability Maturity Model Integration (CMMI) e Melhoria de Processo do Software Brasileiro (MR MPS), embora não sejam modelos destinados à manutenção de software, modelos de processos de software também apresentam contribuições para o aperfeiçoamento dos processos de manutenção.

Porém, [April, Abran e Dumke, 2004], [April et al., 2005] ressaltam que esses modelos não tratam adequadamente o conceito de maturidade para a manutenção de software, uma vez que focam somente o desenvolvimento do software. Durante a fase de manutenção, os processos passam a serem dirigidos pelas solicitações de manutenção, processos esses que estão responsáveis por definir o conjunto de atividades necessárias para atender a essas solicitações.

Diante das questões abordadas acima, há uma motivação do presente trabalho em adotar e avaliar os benefícios de boas práticas de manutenção e propor um

processo destinado à manutenção de software, elaborado a partir da adaptação e implementação de algumas práticas do modelo de maturidade da manutenção de software - SMmm.

1.2 Metodologia de Pesquisa

Quanto à natureza, este trabalho classifica-se como uma pesquisa aplicada. Com relação aos objetivos, pode ser caracterizado como uma pesquisa exploratória. A sua abordagem, é quantitativa, pois os resultados são medidos e avaliados utilizando métricas de software. Quanto ao procedimento, pode ser caracterizado como um estudo de caso.

1.3 O Problema

Considerando manutenção do software, os problemas que permeiam a realidade da organização em questão são os seguintes:

i. Quantidade considerável de defeitos [1] e de retrabalho;

ii. Inabilidade de mudar fácil e rapidamente o software, resultando em soluções incompatíveis;

iii. Ausência de um processo para as atividades de manutenção;

iv. Insatisfação do cliente quando solicitações de correção ou modificação resultam em soluções incompatíveis com consequências negativas;

v. Não existem métricas para que se possa avaliar a qualidade do software que sofreu alteração.

vi. Redução da qualidade global do software como resultado de mudanças que introduzem erros latentes no software mantido.

Resumindo, entre os fatores acima descritos, a questão que pretendo estudar é:

[1] Defeito é um fragmento de um artefato que, se utilizado, pode levar a um erro.

A implementação, usando Team Foundation Service – TFS, de práticas derivadas do SMmm contribui para redução de defeitos injetados pela manutenção?

1.4 Proposta

A solução proposta provém do estudo realizado no capítulo 3 desta dissertação. Inicialmente, para melhoria das atividades de manutenção de software, foi relevante compreender a situação do sistema, para isso, foi feito um estudo preliminar do status da manutenção no sistema em questão.

A proposta deste trabalho é definir, pôr em uso e avaliar algumas boas práticas de manutenção de software derivada do modelo SMmm, adaptado à realidade e necessidades do sistema e confrontar com o estado anterior a implementação das práticas. Os passos para desenvolvimento da solução são listados a seguir.

i. Realizar um diagnóstico inicial da situação do sistema piloto;
ii. Identificar possíveis pontos de melhoria;
iii. Formalizar o conjunto de práticas em um processo de manutenção;
iv. Implementar e acompanhar a execução das atividades de manutenção propostas;
v. Coletar dados e avaliar sua eficácia face ao diagnóstico inicial da situação do sistema piloto;
vi. Relatar os resultados obtidos.

1.5 Estrutura da Dissertação

Esta dissertação está organizada em seis principais capítulos. Apresentamos no capítulo 2 a fundamentação teórica para facilitar o entendimento dos principais conceitos utilizados neste trabalho. No capítulo 3 é apresentado o modelo de maturidade da manutenção de software - SMmm. O capítulo 4 apresenta o processo de manutenção definido. O capítulo 5 apresenta o estudo de caso desenvolvido e finalmente no capítulo 6 a conclusão.

2 Fundamentação Teórica – Contextualização

No presente capítulo são introduzidos conceitos gerais de engenharia de software, manutenção de software, melhoria no processo de software e padrões; fornecendo, assim, embasamento para um melhor entendimento deste trabalho.

2.1 Manutenção de Software

A manutenção de software, no qual se insere esta pesquisa de mestrado, é definida pelo Glossário de Termos de Engenharia de Software como "[...] o processo de modificação de um artefato após a entrega para corrigir falhas (um erro observado), melhorar o desempenho ou outros atributos, ou adaptá-lo a um ambiente modificado. [...]" [IEEE, 1993].

A ISO (traduzido do inglês, Organização Internacional para Padronização) e a IEC (traduzido do inglês, Comissão Eletrotécnica Internacional) ISO / IEC 14764 [ISO/IEC, 2006] define manutenção de software como "... a totalidade de atividades necessárias para fornecer suporte de baixo custo para um sistema de software. Atividades são realizadas durante a fase de pré-entrega, bem como a fase pós-entrega". As atividades pós-entrega incluem mudança de software, treinamento e operação de help desk, as atividades de pré-entrega incluem o planejamento para as operações pós-entrega.

Segundo [Sommerville, 2011], o divisor para saber onde termina o desenvolvimento e onde começa a manutenção está se tornando irrelevante, pois poucas aplicações de software são desenvolvidas a partir do zero. Assim é mais coerente considerar a engenharia de software como um processo evolutivo onde o software é continuamente alterado durante seu ciclo de vida para se adequar aos requisitos e necessidades dos usuários.

As mudanças no software podem ser feitas durante ou após o desenvolvimento, sempre para alcançar os requisitos dos usuários ou por questões tecnológicas. Entretanto, conceitualmente neste trabalho será considerado que desenvolvimento é o processo de criação do software desde concepção inicial até sua entrada em operação e a manutenção de software é o processo de mudança em um software que já está em uso [Sommerville, 2011].

A requisição de modificação classifica-se em correção ou melhoria, derivando dois tipos de manutenção para cada classificação [ISO/IEC, 2006], como ilustra a figura 1, esta dissertação adota a classificação dos tipos de manutenção sugerida pela ISO/IEC.

Figura 1: Classificações dos tipos de manutenção.

Fonte: [Pressman, 2011]

i. Manutenção Adaptativa: Visa manter um software utilizável em um ambiente alterado ou em alteração.

ii. Manutenção Corretiva: De acordo com a norma [ISO/IEC, 2006], a manutenção corretiva refere-se à modificação de um produto de software executada após a entrega a fim de corrigir os problemas detectados, consolidando o reparo do software para alinhá-lo aos requisitos.

iii. Manutenção Perfectiva: Envolve as manutenções responsáveis por inserir ou modificar os requisitos funcionais e não funcionais do software (pouco esforço) e melhorias.

iv. Manutenção Preventiva: Modifica o produto de software visando corrigir falhas latentes no mesmo, antes que se tornem falhas operacionais. Esse tipo de manutenção ocorre quando o software é modificado para melhorar características de confiabilidade ou manutenibilidade, reduzindo o esforço necessário para realização das manutenções futuras. Nessa categoria incluem-se reduzir a

complexidade do código ou eliminar a degradação inserida por outras atividades de manutenção.

Vale ressaltar que a manutenibilidade também afeta o desenvolvimento em virtude das mudanças (alteração feita durante o desenvolvimento em artefatos já aceitos), ou seja, solicitações de alteração de requisitos, de arquitetura e de projeto que ocorrem durante o desenvolvimento.

A norma [ISO/IEC, 2006] define manutenibilidade como a capacidade do produto de software ser modificado. As modificações podem incluir correções, melhorias ou adaptações do software para mudanças no ambiente, nos requisitos e especificações funcionais. De acordo com [Souza Lima, 2009], existem quatro dimensões que caracterizam a capacidade de manutenibilidade: capacidade de ser analisável, capacidade de ser modificada, estabilidade e testabilidade.

Estas dimensões precisam ser especificadas, revisadas e controladas durante a atividades de desenvolvimento de software, a fim de reduzir custos de manutenção. Segundo [Pressman, 2011], observa-se que a presença de processos sistemáticos e maduros, técnicas e ferramentas ajudam melhorar a manutenção de software.

Uma vez que o processo de manutenção é definido como uma parte fundamental do ciclo de vida do software e, a partir dele, é possível realizar correções, adaptações, aperfeiçoamentos e prevenções de erro em um software garantindo sua continuidade. Dessa forma, deve-se ter a constante preocupação com a manutenibilidade de software ao longo do seu desenvolvimento e manutenção de modo a garantir a qualidade do processo de manutenção.

Existem várias linguagens de programação no mercado, algumas delas têm mais destaque do que outras, dentre elas podemos citar as linguagens Java e C#. Estas possuem grande popularidade e são amplamente conhecidas, sendo que o Java é uma linguagem free, ou seja, livre para utilização, e a linguagem da Microsoft C# há certo custo financeiro ao utilizá-la. São linguagens muito semelhantes e poderosas que somam boa porcentagem do mercado de desenvolvimento de software, possuem qualidade que atendem as mais variadas complexidades que as aplicações exigem, mas se diferem em alguns aspectos.

[Bartelli, Fressatti, 2014], apresenta um quadro comparativo entre recursos das linguagens de programação Java e C# como ilustrado na figura 2.

A manutenibilidade tem muito mais a ver com a forma com que a aplicação foi construída, se usa boas práticas de arquitetura e design orientado a objetos, principalmente se tem testes automatizados, e se o acoplamento entre componentes está baixo.

As aplicações .Net podem se apoiar em um ecossistema de servidores/serviços como o IIS, Sharepoint, Windows Azure., tornando simples as tarefas como manutenção, escalabilidade, segurança, entre outras. A plataforma java também tem à sua disposição vários servidores/serviços como WAS, JBoss.

Recurso	Java	C#
Flexibilidade	Não permite o uso de Ponteiros.	Permite o uso de ponteiros, porém não é aconselhável por ser inseguro.
Portabilidade	Oferece Suporte através da JVM.	Oferece Suporte através da linguagem MISL. Interpretada por qualquer plataforma que rode o .NET framework.
IDE de desenvolvimento.	Java trabalha com algumas IDEs de Desenvolvimento como Netbeans, Eclipse.	Só é possivel trabalhar com a plataforma .NET Framework.
Custo.	Java é uma linguagem *Open Source*, livre para ser usada.	Há um custo financeiro para utilização.
Simplicidade	Java é uma linguagem de fácil aprendizado, aliado a produtividade da POO ajuda o desenvolvedor a conhecer a linguagem e usufruir de seus recursos.	Juntamente com o Visual Studio fica simples e fácil programar com C#, sem deixar de ser robusta e sofisticada.
Indexadores.	Não oferece suporte a indexadores.	Oferece esse recurso e ainda trata de sobrecargas.
Gerenciamento de Memoria	Faz a coleta de lixo, liberação de memória automática.	*Garbage Colletor* faz todo o gerenciamento da memoria.
Sobrecarga de Operadores.	Não há esse recurso.	Alguns operadores podem ser sobrecarregados, outros não. Ainda há o estouro aritmético para operadores que gerem resultados fora do intervalo de valores possíveis para o tipo numérico envolvido.

Figura 2: Comparativo de recursos Java x C#

Fonte: [Bartelli, Fressatti, 2014]

2.2 Padrões de Manutenção de Software

Um processo bem definido para manutenção de software pode ser observado, mensurado, e, assim, melhorado. Além disso, a adoção de processos permite a disseminação de práticas de trabalho eficazes mais rapidamente do que ganhando experiência pessoal.

O documento padrão de manutenção da ISO é chamado ISO/IEC 14764 (International Standard for Software Maintenance 14764, 2006) e o documento padrão de manutenção da IEEE (Institute of Electrical and Eletronics Engineers) é chamado IEEE-1219 (Standard for Software Maintenance, 1219-1998). Ambos os padrões descrevem processos de gestão e execução das atividades de manutenção.

Modelos que estabelecem o processo de manutenção de software apresentam diferentes focos, sendo que alguns se atentam mais a questões econômicas, outros ao produto e outros ao próprio processo. Todos possuem vantagens e desvantagens, não existindo um modelo único e aplicável às diversas situações. A melhor solução, muitas vezes, é utilizá-los em conjunto para obter o resultado que agregue mais benefícios para as organizações (YONGCHANG et al., 2011).

2.2.1 ISO/IEC 14764

A norma ISO/IEC 14764 [ISO/IEC, 2006] descreve um framework, cujo objetivo é apoiar o planejamento, o gerenciamento e a execução das atividades de manutenção, independentemente do tamanho e da complexidade do software, da criticidade das modificações e do domínio do negócio. As diretrizes apresentadas por essa norma englobam desde o planejamento da fase de manutenção e a definição da estratégia a ser adotada até a retirada do software do ambiente de produção ao final de sua vida útil.

O processo de manutenção proposto pela norma [ISO/IEC 14764] possui duas fases, a primeira é responsável pelas atividades executadas durante o desenvolvimento inicial e visam, por exemplo, auxiliar a transferência do conhecimento entre a equipe de desenvolvimento e a equipe de manutenção. Essas atividades também viabilizam o planejamento da execução do processo de

manutenção após a entrega do software, mitigando antecipadamente os riscos de insucesso. A segunda fase está relacionada ao período após o desenvolvimento inicial e almeja garantir a operacionalidade e a evolução do software para que esse acompanhe as necessidades dos usuários.

Para que uma requisição de mudança seja executada, é aconselhável que a implementação da manutenção seja guiada por um processo de manutenção. Com base nisso, na [ISO/IEC 14764] apresenta-se uma estrutura do processo de manutenção contemplando atividades e tarefas necessárias, como:

i. Implementação do Processo;
ii. Análise do Problema e da Modificação;
iii. Implementação da Modificação;
iv. Revisão/Aceitação da Manutenção;
v. Migração;
vi. Retirada.

A Figura 3 apresenta uma visão geral do Processo de Manutenção de software da norma ISO/IEC 14764 e como suas atividades interagem.

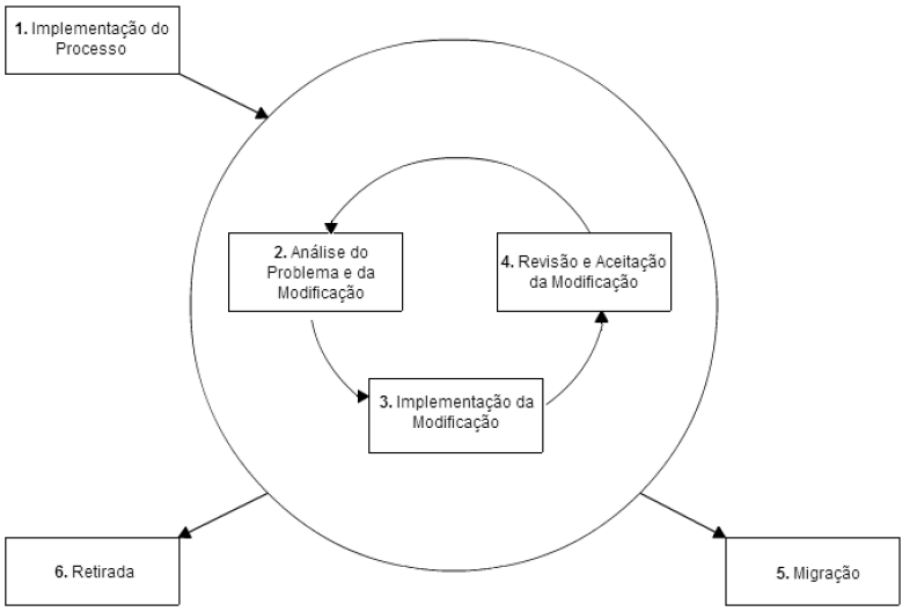

Figura 3: Processo de Manutenção de Software da ISO/IEC 14764.

Fonte: [ISO/IEC 14764, 2006]

Enquanto as atividades 1, 5 e 6 não costumam ser executadas mais de vez no processo de manutenção, as ações 2, 3 e 4 se comportam de forma cíclica e somente são finalizadas quando a modificação está em conformidade e é, de fato, aprovada.

O detalhamento do processo de manutenção é apresentado a seguir resumido em critérios de saída:

i. Implementação do Processo
- Os planos e procedimentos a serem executados durante a fase de manutenção são estabelecidos e aprovados. No caso deste trabalho, os planos e procedimentos para identificação e catalogação dos software legados devem ser propostos, executados e submetidos a aprovação;

ii. Análise dos Problemas e das Modificações
- As solicitações de manutenção são analisadas;
- As alternativas para a execução das modificações são propostas, documentadas, aprovadas e priorizadas.

iii. Implementação das Modificações
- As modificações aprovadas são desenvolvidas e testadas pelos mantenedores.

iv. Revisão/Aceitação da Manutenção
- As modificações realizadas são avaliadas para garantir a conformidade com o escopo aprovado e para determinar sua corretude.

v. Migração
- Os passos necessários para atender as solicitações de migração do software para uma nova plataforma de execução, sistema de banco de dados e demais necessidades vinculadas às solicitações adaptativas são definidos, documentados, aprovados e executados.

vi. Retirada

- A retirada segura do software do ambiente de produção ao final de sua vida útil é analisada e executada;
- A necessidade de aquisição ou de desenvolvimento de um novo sistema é avaliada.

De acordo com [APRIL; ABRAN, 2008] é importante ressaltar que organizações podem adaptar o modelo e suas atividades de acordo com o contexto em que estão inseridas. Diversas atividades de manutenção de software ainda não foram formalizadas e, consequentemente, não são tratadas em normas, mas são observadas diariamente na indústria.

2.2.2 IEEE/1219

A norma IEEE 1219 [IEEE 1219, 1998] descreve um processo iterativo para a gestão e execução de atividades de manutenção de software. A utilização desta norma não é restrita ao tamanho, complexidade, criticidade ou aplicação do produto de software.

Esta norma define os requisitos para o processo, controle e gestão do planejamento, execução e documentação das atividades de manutenção de software. Os critérios estabelecidos aplicam-se tanto no planejamento da manutenção de software, quanto no desenvolvimento, bem como o planejamento e execução das atividades de manutenção para produtos de software existentes. O ideal seria que o planejamento da manutenção iniciasse durante a fase de desenvolvimento de software.

A Figura 4 apresenta uma visão geral do Processo de Manutenção de software da norma IEEE 1219.

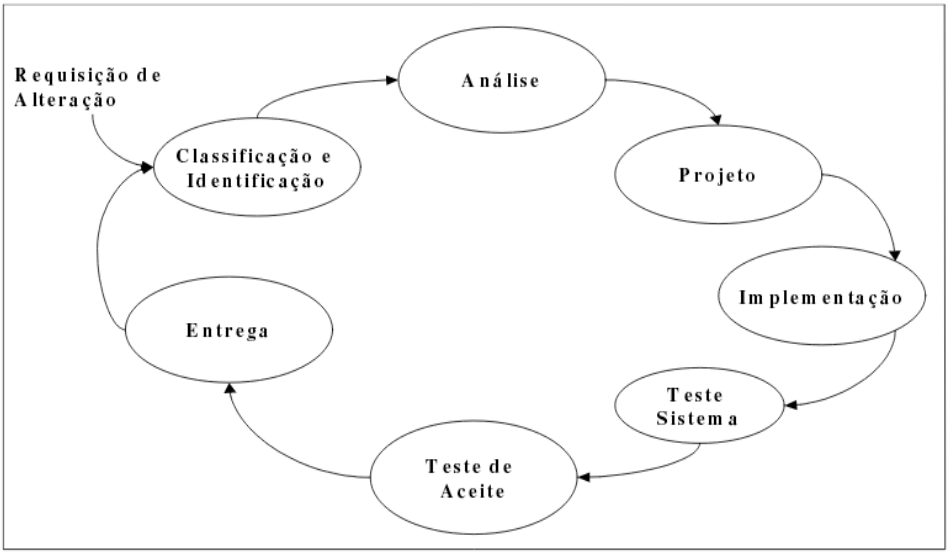

Figura 4: Processo de Manutenção de Software da IEEE 1219.

Fonte: [WEBOK (2004)]

As fases da norma IEEE 1219 são resumidas a seguir na tabela 1:

Fase	Objetivos
1. Identificação Problemas	Identificar, classificar e priorizar requisitos da modificação.
2. Análise	Determinar a possibilidade de realização, caminhos alternativos de solução e escopo da alteração.
3. Projeto	Desenvolver um projeto para as modificações aprovadas.
4. Implementação	Executar as mudanças requeridas no software.
5. Teste Sistema	Verificar se as modificações foram satisfeitas e nenhuma nova falha foi introduzida.
6. Teste Aceitação	Aplica teste de aceite nos sistemas de software modificado certificando-se da satisfação do cliente-usuário.
7. Entrega	Entrega as modificações nos sistemas de software e atualizações nas documentações.

Tabela 1: Fases do Processo de Manutenção da IEEE 1219

2.3 Melhorias de Processo de Software

Segundo [Pressman, 2011], processo é um conjunto de atividades, ações e tarefas realizadas na criação de algum artefato. No contexto da engenharia de software, um processo não é uma prescrição rígida de como desenvolver um software. Ao contrário, é uma abordagem adaptável que possibilita a equipe de software realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas.

2.3.1 Iniciativas de Melhorias de Processos de Software

Na visão de [Sommerville, 2011] as iniciativas de melhoria de processos representam o conjunto de esforços aplicados na evolução dos processos de software por meio da análise, da medição e do aperfeiçoamento das práticas de Engenharia de Software executadas ao longo do ciclo de vida de um sistema.

As metas comumente estabelecidas para a melhoria dos processos de software envolvem: a redução dos custos de desenvolvimento e de manutenção, o aperfeiçoamento da qualidade dos produtos entregues, o aumento da produtividade das equipes, a redução do índice de retrabalho e do "time to market" para o lançamento dos produtos e de suas evoluções e o aumento da satisfação dos usuários.

De acordo com [Pressman, 2011], os modelos de maturidade são classificados como abordagens de melhoria de processos de nível organizacional. Modelos de maturidade como o CMMI (Capability Maturity Model Integration) [SEI, 1995] e o MR MPS (Modelo de Referência para Melhoria de Processos do Software) [SOFTEX, 2009], estipulam meta-modelos compostos por práticas genéricas para desenvolver e avaliar a maturidade das organizações nos diversos processos executados ao longo do ciclo de vida de um software.

2.3.2 MR-MPS

De acordo com [SOFTEX, 2012a], o Modelo de Referência para Melhoria de Processo de Software (MR-MPS) contém os requisitos que as organizações devem atender para estar em conformidade com o Modelo MPS. A iniciativa foi responsável pelo desenvolvimento do Modelo de Referência para Melhoria do Processo de Software Brasileiro (MPS-SW), que levou em consideração normas e modelos internacionalmente reconhecidos, boas práticas da engenharia de software e as necessidades de negócio da indústria de software nacional. Com isso, foi possível estabelecer um caminho economicamente viável para que organizações, incluindo as pequenas e médias empresas, alcancem os benefícios da melhoria de processos e da utilização de boas práticas da engenharia de software em um intervalo de tempo razoável.

Segundo [SOFTEX, 2009], a divisão das metas e práticas em sete níveis de maturidade tem como objetivo facilitar a adoção do MR MPS pelas micros, pequenas e médias empresas e reduzir o tempo necessário para obtenção dos resultados das melhorias. Complementarmente, o modelo cobre o conteúdo de outros Modelos de Referência de Processo, como o CMMI com o qual é compatível.

2.3.3 CMMI (Capability Maturity Model Integration)

O CMMI para Desenvolvimento [CMMI-DEV, 2006] é um modelo de maturidade para melhoria de processos, destinado ao desenvolvimento de produtos e serviços, e composto pelas melhores práticas associadas a atividades de desenvolvimento e de manutenção que cobrem o ciclo de vida do produto desde a concepção até a entrega e manutenção.

De acordo com [CMMI-DEV, 2006] os elementos que compõe o modelo CMMI-DEV são agrupados em três categorias que refletem como eles devem ser interpretados:

Elementos Requeridos: Metas específicas e genéricas são elementos requeridos do modelo. Estes elementos devem ser estabelecidos pelos processos definidos, e implementados pela organização. São essenciais para avaliar a

satisfação de uma área de processo. O estabelecimento (ou satisfação) das metas é usado em avaliações como base para satisfação da área de processo e maturidade organizacional. Apenas a declaração das metas é um elemento requerido do modelo, o título e qualquer nota associada são considerados elementos informativos do modelo.

Elementos Esperados: Práticas específicas e as genéricas são elementos esperados no modelo. Estes elementos descrevem o que uma organização tipicamente irá implementar para satisfazer um componente requerido. Servem como guia para aqueles que implementam melhorias e ou executam avaliações. As práticas descritas, ou as alternativas aceitáveis para elas, espera-se que estejam presentes nos processos planejados e implementados da organização antes que as metas sejam consideradas satisfeitas. Apenas a declaração da prática é um elemento esperado do modelo, o título e qualquer nota associada são considerados elementos informativos do modelo.

Componentes Informativos: Sub-práticas, produtos de trabalho típicos, particularidades da disciplina, elaborações de prática s genéricas, títulos, notas e referências são elementos informativos do modelo. Estes elementos ajudam o usuário do modelo a entender as metas e práticas e como elas podem ser estabelecidas, fornecendo detalhes para ajudar a começar a pensar em como estabelecer as práticas e metas.

Os elementos não classificados do modelo constituem um grupo que fornece uma maneira de apresentar as práticas genéricas.

Quando se usa um modelo CMMI como guia, processos são planejados e implementados em conformidade com os elementos esperados e requeridos das áreas de processo. Conformidade com uma área de processo significa que nos processos planejados e implementados existe um processo associado (ou processos) que endereça as práticas específicas e genéricas da área de processo, ou alternativas, que geram um resultado de acordo com a meta associada com aquela prática específica ou genérica.

Ainda que o foco principal do modelo de maturidade [CMMI SEI, 1995] seja a fase de desenvolvimento, a manutenção também aproveita alguns benefícios obtidos pela adoção do modelo. Por exemplo, as orientações propostas podem

melhorar a manutenibilidade dos sistemas, garantir a evolução e minimizar os problemas gerenciais encontrados durante a manutenção. No entanto, conforme destacam [April, Abran e Dumke, 2004] e [April et al., 2005], existem dificuldades para aplicá-lo durante a fase de manutenção. Uma das causas ressaltada para essas dificuldades refere-se à ausência do tratamento do conceito de maturidade durante a manutenção, o tratamento superficial das práticas de melhoria de processos para a manutenção. Consequentemente, conforme destaca [Hall et al. 2001], as organizações não têm implementado efetivamente os modelos de maturidade capacitando-as a evoluir os processos de manutenção.

Para estabelecer mecanismos para desenvolver e avaliar o nível de maturidade das organizações em relação à manutenção e orientar a evolução dos processos dessa fase do ciclo de vida, alguns estudos têm proposto modelos de maturidade específicos para manutenção de software, conforme discutido a seguir.

2.3.4 Software Maintenance Maturity Model (SMmm)

Os autores Alain April e Alain Abran realizaram uma grande quantidade de pesquisas no desenvolvimento do modelo SMmm. Eles revisitaram trinta anos de publicações, dissecando modelos de maturidade e compilaram uma lista abrangente das melhores práticas que abrangem todos os aspectos da manutenção de software. As pesquisas conduzidas ao longo de alguns anos pelos autores citados, resultou na publicação do livro - *Software Maintenance Management: evaluation and continuous improvement*, o qual detalha a proposta do Modelo de maturidade único para a manutenção de software – SMmm [April, Abran 2008]. Esta referência bibliográfica é utilizada no próximo capítulo (capítulo 3) para fundamentação do SMmm.

Este modelo decorre da revisão das melhores práticas de normas e modelos de diversas áreas, entre as referências utilizadas para concepção do SMmm incluem-se as práticas sugeridas pela norma ISO / IEC 12207 [ISO/IEC, 2008] ISO/IEC 14764 [ISO/IEC, 2006], pelo padrão IEEE 1219 [IEEE / EIA 1219, 1998] e pelos modelos [CMMI-DEV, 2006], Control Objectives for Information and related Technology (COBIT) [ISACA, 2010] e Information Technology Infrastructure Library (ITIL) [OGC, 2011].

A proposta do SMmm pode ser vista como um complemento ao CMMI, com foco em manutenção de software, embora existam diferentes atividades, algumas são comuns a desenvolvedores e mantenedores, por exemplo:

i. Definição do processo
ii. Codificação
iii. Testes
iv. Gerenciamento de configurações
v. Garantia da Qualidade

O modelo SMmm foi concebido com base nas necessidades dos diversos envolvidos nas atividades de manutenção, abrangendo os usuários, as equipes de infraestrutura, os desenvolvedores, os gestores e as equipes de suporte. Ao todo são definidos três grupos de processos (processos primários, de apoio e organizacionais).

De acordo com [April e Abran, 2009] os objetivos dos processos propostos resumem-se na garantia da disponibilidade das aplicações, na reação rápida em caso de falhas, na garantia dos limites de atendimento impostos pelos contratos de nível de serviço, na manutenção da confiança dos usuários sobre a competência e disponibilidade da equipe, além da execução dos serviços requisitados dentro do orçamento destinado à manutenção. Para garantir esses objetivos a especificação do SMmm institui quatro Domínios de Processo, dezoito Áreas-Chave de Processo (KPAs), setenta e quatro Roteiros e quatrocentos e quarenta e três práticas recomendadas.

Segundo [April et al., 2005] , o modelo SMmm endereça as necessidades dos diversos envolvidos na execução de todos os tipos de solicitações de manutenção. O modelo de referência proposto contém práticas relevantes para garantir a satisfação dos envolvidos e o sucesso dos projetos de manutenção.

3 Software Maintenance Maturity Model (SMmm)

Este capítulo apresenta o Modelo de Maturidade de Manutenção de Software (SMmm) de forma estruturada, organizada por níveis, as boas práticas e suas referências. Segundo [April, Abran 2008], as organizações podem utilizar o modelo SMmm para suportar um programa de melhoria contínua para manutenção de software, o modelo auxilia na identificação dos pontos fortes e fracos nas atividades de manutenção e ajuda também a especificar quais questões abordar e como lidar com elas, proporcionando melhorias nos processos de manutenção de software.

É importante salientar que o escopo desta pesquisa de mestrado abordará algumas práticas do domínio de processo Engenharia de Evolução de Software, referenciando as seguintes Áreas–chave de processo: Evolução e correção de software e verificação e validação de software e suas respectivas práticas, como demonstra a Tabela 2.

Domínio do processo	Área–chave de processo	Roteiro
Engenharia de evolução de software	Evolução e correção de software	Projeto detalhado
		Construção (programação)
		Teste (unidade, integração, regressão)
		Documentação
	Verificação e validação de software	Revisões
		Teste de aceitação
		Transição para produção

Tabela 2: Domínio de Processo foco da dissertação

3.1 Fundamentos do Modelo SMmm

No SMmm, os processos de manutenção de software são agrupados em três classes, como ilustra a figura 5, a ideia principal é fornecer uma representação semelhante a utilizado pela norma ISO / IEC 12207 [ISO/IEC, 2008], mas com foco em atividades e processos de manutenção de software:

i. Processos Primários - processos operacionais de manutenção de software;

ii. Processos de Suporte - apoio aos processos primários;

iii. Processos Organizacionais.

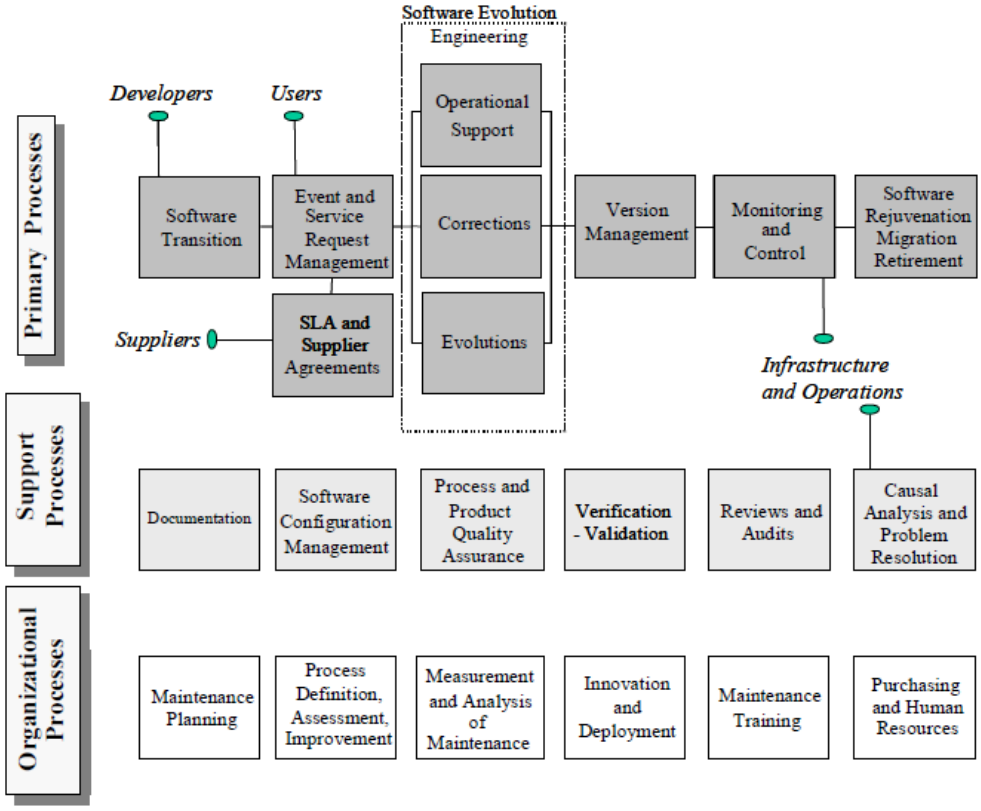

Figura 5: Classificação dos processos-chave de manutenção

Fonte: [April et al., 2005]

O modelo genérico de processo de manutenção de software, ilustrado na figura 5 ajuda a explicar e representar a vários processos-chave de manutenção de software.

3.1.1 Processos Operacionais de Manutenção de Software

De acordo com [April, Abran 2008], os processos operacionais, também chamados processos primários, que organizações de manutenção de software utilizam, iniciam com o processo de transição de Software. Este processo garante

que uma abordagem estruturada e coordenada seja utilizada para transferir o software para o mantenedor. Uma vez que o software passou para responsabilidade do mantenedor, o processo de gestão de eventos e solicitação de serviços administra todas as questões diárias, tais como: relatórios de problemas, solicitações de modificação e pedidos de suporte.

O primeiro passo neste processo é avaliar se um pedido deve ser endereçado, reencaminhado ou rejeitado, com base no acordo de nível de Serviço (SLA) e a natureza e tamanho do pedido [April, 2001]. As solicitações aceitas são documentadas, atribuída priorização e processadas em uma das categorias de serviços:

i. Processo de apoio operacional;
ii. Processo de correção de software;
iii. Processo de evolução do software.

[April, Abran, 2008], destaca que certos pedidos de serviços não levam a qualquer modificação do software, no SMmm, estes são chamados de atividades de apoio operacional, e consistem em:

i. Responder aos questionamentos;
ii. Prestação de suporte técnico, consultoria, monitoramento e gestão problema para os desenvolvedores e operações;
iii. Auxiliar os clientes a compreender melhor o software.

Os próximos processos primários referem-se ao processo de gestão de versionamento, que move itens para a produção e processo de monitoração e controle, o qual assegura que o ambiente operacional não foi degradado. Mantenedores devem monitorar o comportamento operacional do sistema e seus ambientes para encontrar possíveis sinais de degradação. Eles rapidamente vão alertar outros grupos de apoio (operadores, suporte técnico, programação, redes e suporte de desktop) quando algo não usual acontece e julgar se necessita ser investigado. O último processo primário, Rejuvenescimento, Migração e Reforma,

abordam atividades para melhorar a manutenção, as atividades de migração para mover um sistema para outro ambiente e as atividades de retirada quando um sistema é encerrado.

3.1.2 Processos de Apoio a Manutenção de Software

De acordo com a norma ISO / IEC 12207 [ISO/IEC, 2008], um processo que é utilizado, por um processo operacional é chamado de processo de apoio operacional. Em muitas das organizações, estes processos são compartilhados por desenvolvedores e mantenedores. Processos de apoio incluem tipicamente:

i. Processo de documentação;
ii. Processo gestão de configuração de software e ferramentas;
iii. Processo de garantia de qualidade e produto;
iv. Processo de verificação e validação;
v. Processos de revisão e auditoria;
vi. Processo de resolução de problemas.

Um processo de análise, que é utilizado para identificar problemas de longa data e suas causas, é por vezes, também encontrado e classificado como um processo de apoio. Estes são todos os processos de suporte de manutenção de Software.

3.1.3 Processos Organizacionais de manutenção de software

Processos organizacionais são normalmente oferecidos pela organização e por outros departamentos da organização, por exemplo, perspectivas de planejamento de manutenção, atividades relacionadas com o processo, medição, inovação, treinamento e recursos humanos. Embora estes processos sejam importantes para medir e avaliar, é mais importante para o mantenedor definir e aperfeiçoar os processos operacionais em primeiro lugar. Os processos de suporte operacional e os processos organizacionais seguem este, como observa [April, Abran, 2008].

3.2 Objetivos do Modelo SMmm

De acordo com [April, Hayes, Abran, Dumke, 2005], o SMmm foi concebido como um modelo de referência com foco no cliente, ou seja, um ponto de referência para:

i. Auditoria da capacidade de manutenção de software de um fornecedor de serviços de manutenção de software ou contratante;
ii. Melhorar internamente as organizações de manutenção de software.

O modelo foi desenvolvido a partir da perspectiva do cliente, através da experiência em ambiente comercial e competitivo. O objetivo final dos programas de melhoria iniciado após uma avaliação SMmm, foi aumentar a satisfação do cliente, ao invés da conformidade rígida com os padrões referenciados pelo modelo.

O nível mais elevado da maturidade no contexto SMmm significa, para os clientes:

i. Alcançar os objetivos em níveis de serviço, cumprindo as prioridades dos clientes;
ii. Implementar as melhores práticas disponíveis para os mantenedores de software;
iii. Obter transparência dos serviços de manutenção de software, submetendo-se a custos competitivos;
iv. Explorar prazos menores de entrega de serviço de manutenção de software.

Para uma organização de manutenção, alcançar um nível maior de maturidade, pode resultar em [April, Hayes, Abran, Dumke, 2005]:

i. Menores custos de manutenção e suporte;
ii. Menores ciclos e intervalos;
iii. Aumento da capacidade de alcançar os níveis de serviço;

iv. Acréscimo da capacidade de cumprir os objetivos de qualidade quantificáveis em todas as fases do processo de manutenção e serviço.

3.3 Escopo do Modelo SMmm

A partir de um modelo são criados processos definidos, esses são alinhados à organização em que são utilizados. Um processo que esteja em conformidade com um modelo assegura uma série de propriedades consideradas relevantes pelos formuladores do modelo. Para um melhor mapeamento da realidade do ambiente de manutenção, o SMmm sugere incluir muitas perspectivas essenciais do mantenedor de software e tanto quanto possível, do contexto de trabalho prático da manutenção.

Este modelo não se destina a descrever as técnicas específicas ou todas as tecnologias que devem ser utilizadas pelos mantenedores. As decisões relativas a seleção de técnicas ou tecnologias específicas são adaptados por cada organização. Para uma avaliação ou a concepção de um programa de melhoria, os usuários do modelo SMmm devem instanciar os modelos de referência (mostrado na figura 5), no contexto da sua organização. Para alcançar isso, o julgamento profissional se faz necessário, para avaliar a forma como uma organização se assemelha ao modelo de referência [April, Hayes, Abran, Dumke, 2005].

3.4 Domínios de Processo e Áreas-chave de Processo

Um domínio de processo é um conjunto de práticas relacionadas em uma área que, quando executada coletivamente, satisfaz um conjunto de metas consideradas importantes para fazer melhorias significativas na área.

As áreas-chaves de processo (KPAs) identificam um grupo de atividades relacionadas, que quando executadas em conjunto satisfazem um grupo de metas relevantes para a melhoria do processo. Cada uma das KPA é descrita em termos das práticas-chave que contribuirá à satisfação de seus objetivos. As práticas-chave descrevem as atividades que mais contribuirão à efetiva implementação dos KPAs.

A Figura 6 apresenta, do lado esquerdo, o quatro domínios do processo CMMi atuais e, à direita, a proposta de quatro processos domínios da manutenção de software. Esta alteração foi introduzida para refletir o gerenciamento de eventos/request de manutenção de software em contraste com o gerenciamento de projetos utilizados no desenvolvimento de software. O modelo de maturidade proposto concentra-se em atividades de manutenção que são baseados em pequenas solicitações de manutenção.

Ainda de acordo com [April, Abran, 2008] é especificado, na manutenção de software, que as atividades de engenharia e suporte deve ser centrada em torno da evolução do software e não na sua concepção inicial. Para refletir esta diferença fundamental, o domínio de engenharia CMMi foi renomeado "Engenharia de evolução." Engenharia de evolução está principalmente preocupada com a modificação, rejuvenescimento e migração software de produção. Entende-se como rejuvenescimento de software um conjunto de técnicas que facilitam a manutenibilidade e consequentemente a adaptação a novas tecnologias pela maioria dos softwares.

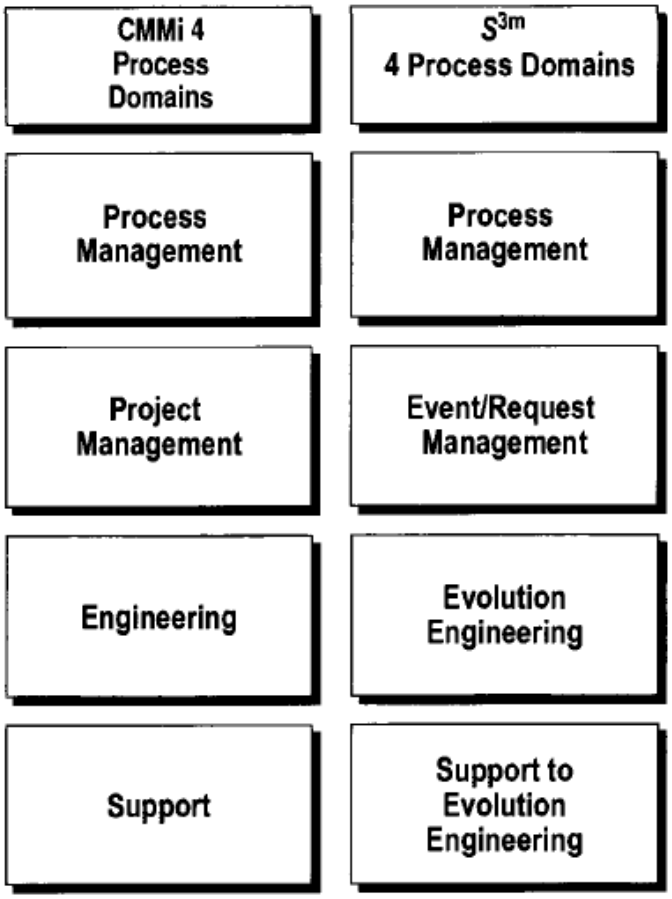

Figura 6: Domínio de Processo CMMI e SMmm.

Fonte: [April, Abran, 2008]

De acordo [April, Abran, 2008] as normas internacionais, como ISO 12207 e ISO 14764, foram aplicadas aos processos de manutenção de software e fazem parte dos conhecimentos fundamentais do domínio, estas normas propõem modelos de processos que são representações descritivas ou prescritivas de atividades que precisam ser seguidas ou adaptadas pelos mantenedores.

A figura 7 ilustra o que cada domínio de processo SMmm contém, com intuito de cobrir estas duas normas internacionais de software. Um modelo de manutenção que está em conformidade com essas duas normas cobre os quatro assuntos no seu processo de engenharia:

i. Aquisição
ii. Fornecimento
iii. Desenvolvimento

iv. Operações

Este mapeamento ajuda a identificar áreas-chave de processo (KPAs) do modelo.

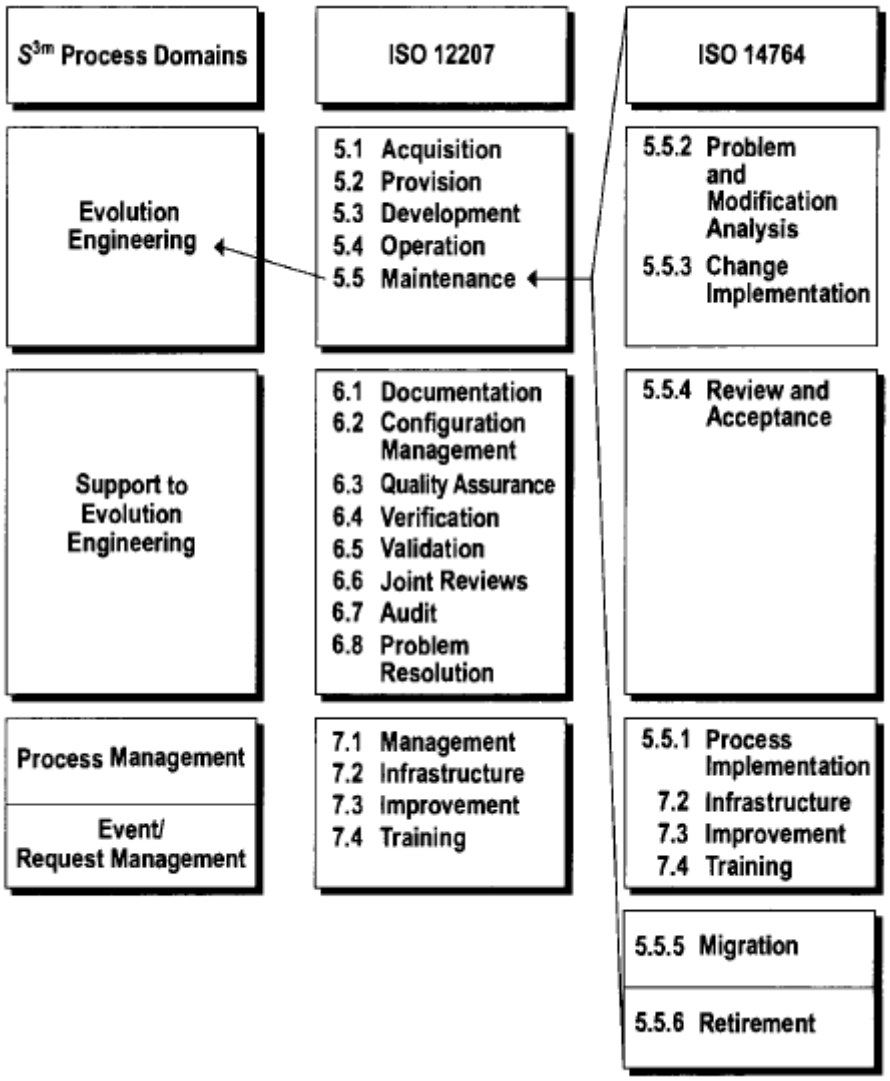

Figura 7: Padrões abordadas pelo modelo SMmm.

Fonte: [April, Abran, 2008]

O conteúdo proposto pelo SMmm é apresentado na figura 8, é composto por quatro domínios de processo, dezoito áreas-chave de processo (KPAs) cada uma contendo, por sua vez," Roteiros " para práticas de manutenção. Os roteiros são corpos de conhecimentos que concentram quatocentos e quarenta e três práticas recomendadas ligadas entre si. Enquanto alguns KPAs são específicos para

manutenção, outros foram derivados do CMMI e alguns modelos foram modificados para mapear mais de perto as características da manutenção diária.

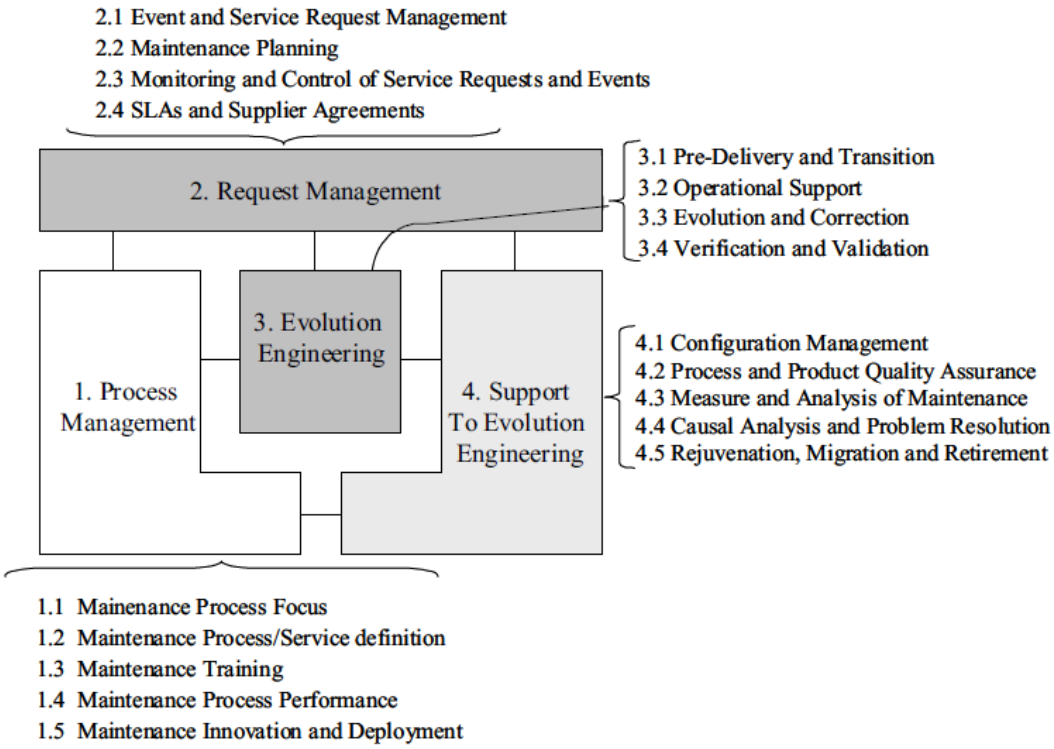

Figura 8: Domínios e Áreas-chave de processo doSMmm

Fonte: [April, Hayes, Abran e Dumke, 2005]

3.5 Domínio de Processo Engenharia de Evolução de Software

Este domínio de processo abrange atividades operacionais relativas à manutenção de software, contém práticas associadas a pré-entrega, transição, operação suporte, serviços de evolução e correção, verificação e validação, compreendendo os seguintes KPAs:

i. Pré-entrega e serviços de transição
ii. Serviços de Apoio Operacional
iii. Serviços evolução e correção de Software
iv. Verificação e Validação de software

A figura 9 mostra as interações entre os KPAs do Domínio de Processo Engenharia de Evolução de Software e outros domínios do processo. O KPA Pré-entrega e Serviços de Transição é o primeiro contato da equipe de manutenção com o software que eles terão de manter no futuro. O principal objetivo desta KPA é aumentar a compreensão do mantenedor e tentar influenciar a capacidade de manutenção de software durante seu desenvolvimento (se possível). O KPA Serviços de Suporte Operacional lida com pedidos de serviços que não são resolvidos pelo help desk. O serviços de suporte operacional não modificam o software, mas fornece um serviço de consultoria para ele.

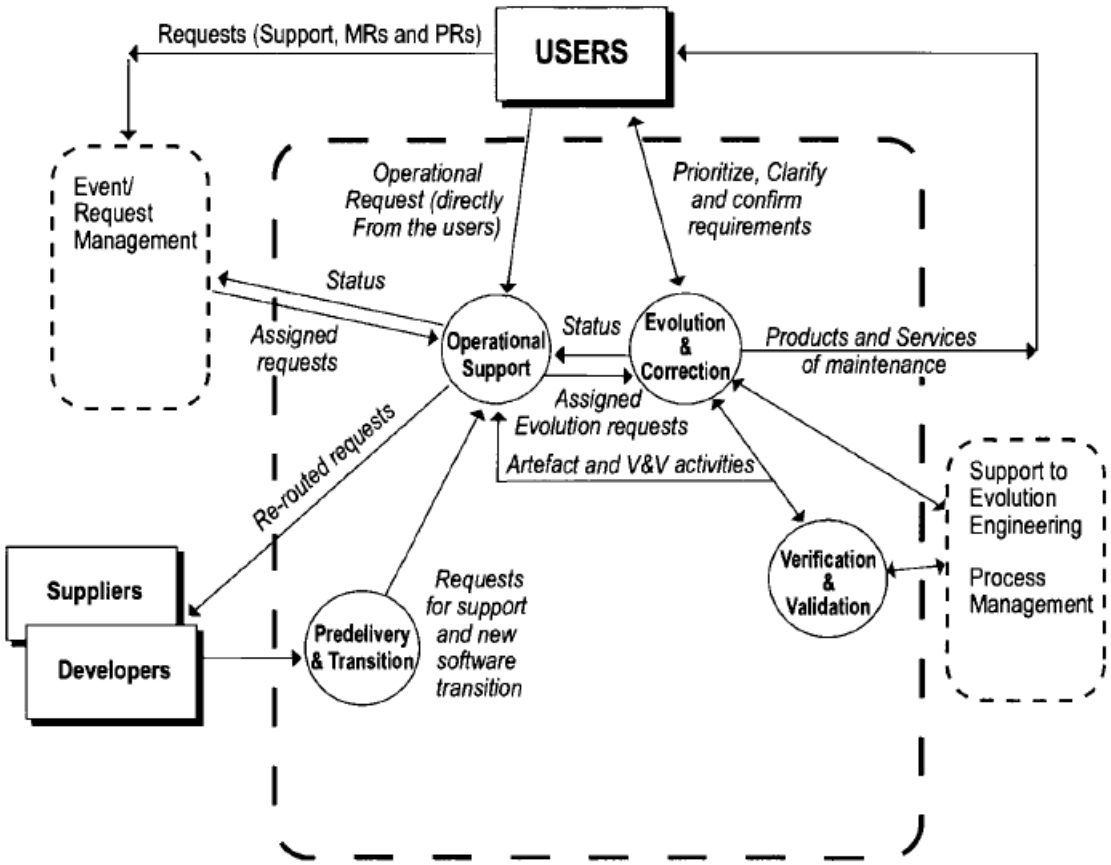

Figura 9: Interações dos KPAs - Dom. de Processo Eng. de Software

Fonte: [April, Abran e Dumke, 2004]

O KPA Serviços de Evolução e correção de software prossegue com a modificação do software e por fim o KPA Verificação e Validação de software

controla as solicitações de modificação, o qual foram atribuídos pelos recursos e estão atualmente em evolução.

3.5.1 KPA Evolução e Correção de Software

3.5.1.1 Roteiros do KPA Evolução e Correção de Software

De acordo com [April, Abran, 2008], o KPA Evolução e correção de software estabelece atividades / tarefas (roteiros) para modificação do software, que consiste em: projeto detalhado, evolução (adaptativa, perfectiva e preventiva), correções e atividades de teste. A equipe de manutenção, inicialmente, realiza análise de impacto da alteração e após prossegue com a descrição das especificações detalhadas da modificação necessária, ou seja, um projeto detalhado. Em seguida, o mantenedor efetua a alteração do software, assegurando a gestão das necessidades do cliente durante a modificação, realiza os testes e alterações pertinentes na documentação. Nesta KPA, as atividades de manutenção utilizam processos específicos de manutenção (e seu workflow), bem como métodos suportados, ferramentas e ambientes.

Processos de evolução de software são semelhantes aos processos de desenvolvimento, mas são adaptados para o uso do mantenedor [ISO/IEC, 2008]. Para serviços de correção, a equipe de manutenção pode optar por implementar uma solução temporária para restaurar o serviço e em seguida, avançar com a solicitação de manutenção para resolver o problema permanentemente [April, Abran, 2008].

3.5.1.2 Práticas do KPA Evolução e Correção de Software

De acordo com [April, Abran, 2008], as práticas obedecem ao seguinte esquema de nomenclatura: Cada prática inicia-se com um prefixo que corresponde à sua respectiva área de domínio (por exemplo, **Evo**), seguida de um número na forma x.y.z (por exemplo, **Evo 3.0.1**). O "x" corresponde ao número da área-chave de processo (KPA) com a qual a prática relaciona-se. O "y" corresponde ao nível de maturidade à qual está classificada a prática e o "z" corresponde à

numeração sequencial da prática. No exemplo citado acima o prefixo **Evo** corresponde ao domínio de processo Engenharia de Evolução de Software, a sequencia numérica **3.0.1** identifica que a prática está associada à terceira área-chave de processo – Evolução e correção de Software, no nível zero de maturidade e possui sequencial igual a um.

As práticas detalhadas deste KPA são apresentadas a seguir.

Evo 3.0.1 A organização de manutenção de software não fornece correção e serviços de evolução para software [SMmm].

Evo 3.1.1 A organização de manutenção de software fornece correção informal e serviços de evolução para software [SMmm].

Cada pedido de manutenção é atribuído ao mantenedor que segue suas próprias práticas locais para corrigir ou evoluir o software e abrir / fechar solicitações. Manutenção Corretivas e evolutivas são tipicamente realizadas de forma reativa. Pouca documentação formal e de controle (dificuldade em encontrar uma lista do da carga de trabalho de manutenção atual e alguns pedidos são perdidos) e não há uma medição clara de sucesso.

Seguem as práticas relacionadas ao roteiro design:

Evo 3.2.1 O projeto detalhado da solicitação de manutenção elabora as necessidades do usuário, e é desenvolvida de acordo com um procedimento documentado [ISO 12207, 5.3.6; IEEE 1219, 4.3; SMmm].

Para atender a essa prática, uma série de atividades deve ser executada pelo mantenedor. Por exemplo, uma vez que o projeto detalhado da solicitação foi elaborado, identificam-se os componentes afetados do software, o mantenedor modifica a documentação para estes componentes, constrói casos de teste para verificação, valida o novo design (incluindo as questões sobre segurança e proteção), identifica e cria os casos de teste para a regressão e, por fim, atualiza a documentação com a lista de modificações.

Evo 3.2.2 As atividades de investigação e de design associado a uma falha (Documentada no *Problem Report* - PR) são realizadas de acordo com um procedimento documentado [SMmm].

PRs normalmente não são processados como solicitações de mudança. A maior diferença está no fato de que, numa situação de falha, não há nenhuma análise formal de impacto seguido de um pedido de autorização apresentado ao cliente. Na verdade, o cliente concorda que o mantenedor precisa trabalhar rapidamente para reparar uma falha, sem uma autorização. O engenheiro de manutenção procede diretamente com a solução da falha e efetua as mudanças na documentação depois.

Seguem as práticas relacionadas ao Roteiro Evolução/Correção:

Evo 3.2.3 As atividades de implementação (de programação) são executadas de acordo com um procedimento documentado [SMmm].

O mantenedor segue os padrões de programação e convenções de nomenclatura da unidade organizacional para a implementação de uma modificação, isto é, de codificação. Para isso, o mantenedor deve seguir o ciclo de vida de manutenção em etapas, tendo o cuidado de documentar seus resultados.

Evo 3.2.4 Os papéis e responsabilidades sobre as modificações de dados operacionais são formalmente definidas pelo engenheiro de manutenção [SMmm].

O engenheiro de manutenção terá que realizar modificações de dados (por exemplo, para modificar dados existentes ou destruir dados) no *software*. Estes dados operacionais são de propriedade dos usuários e clientes. Para atender a essa prática, é necessário:

i. Que o proprietário dos dados dê permissão antes dos dados operacionais serem modificados ou destruídos.
ii. Que a parte interessada seja informada dos eventos e alterações de dados.

iii. Um arquivo de registros de solicitações / comunicações / autorizações dos pedidos de manutenção (para auditorias internas, auditorias externas e de gestão de comentários).

Evo 3.2.5 A linguagem de programação original do software deve ser mantida para as correções e evolução do software [SMmm].

Para atender a essa prática, o software deve ser corrigido e evoluído na línguagem principal do software. A complexidade do software é reduzida quando o número de idiomas é reduzido ao mínimo. Se uma nova linguagem foi escolhida deve ser explicado, e as vantagens da utilização desta linguagem comprovada e a redução de custos e facilidade de manutenção confirmadas futuramente.

Evo 3.2.6 As inter-relações (rastreabilidade) entre os procedimentos administrativos, os módulos, os processos e os dados são atualizados quando o software é corrigido e/ou evoluído [SMmm].

Neste nível de maturidade, a documentação e todos os componentes envolvidos em um modificação deve ser identificado e atualizado. A rastreabilidade é mantida, esta prática é necessária para preservar um traço de todo o conjunto de mudanças para:

i. Processo de verificação e validação
ii. Manter uma medição e um histórico de recuperação de falhas
iii. Retrocesso em caso de problemas futuros

Armazenar todas as informações sobre uma solicitação através dos vários componentes de software é uma boa maneira de observar uma mudança, incorporando o ID único de um pedido em todos os lugares, onde é realizada a modificação, de modo a que o mantenedor será capaz, de encontrar os lugares exatos onde foram efetuadas as modificações.

Seguem as práticas relacionadas ao Roteiro de Teste (Unidade, Integração e Regressão):

Evo 3.2.7 Os testes de unidade devem ser executados para cada componente referente a modificação. [IEEE 1219,4.4.2.1].

Para atender a essa prática, a equipe de manutenção deve desenvolver:

i. Planos de teste de unidade
ii. Dados de teste, descrevendo todas as condições para ser testado.

O mantenedor deve certificar-se que a mudança (que possui impacto em vários componentes identificados na configuração) atende aos requisitos. O mantenedor deve também documentar os resultados dos testes de unidade com o objetivo de provar que o componente modificado não desestabilizou o software e que os componentes modificados ainda satisfazem as necessidades do cliente [ISO 12207, 5.5.3.2].

Evo 3.2.8 Os testes de integração e de regressão devem ser realizados para cada componente referente a modificação [SMmm].

Para atender a essa prática, a equipe de manutenção deve desenvolver:

i. Um plano para a integração e testes de regressão.
ii. Identificar os dados para testes de integração e regressão, descrevendo todas as condições para teste.

Deve certificar-se que os componentes que foram submetidos a uma mudança estão integradas no conjunto do software e não causam efeitos colaterais. Deve-se documentar os resultados dos testes[ISO 12207, 5.3.8].

Evo 3.2.9 A documentação do usuário deve ser submetida a teste pelo mantenedor durante os testes unitários e testes de integração [SMmm].

Qualquer documentação que será devolvida para o usuário está sujeita a testes pelo mantenedor para se certificar de que espelha bem as alterações

aplicadas ao software. Quando a documentação não descreve o que o software faz, a documentação falha deve ser assinalada e o mantenedor, posteriormente, deve modificá-la.

Seguem as práticas relacionadas ao Roteiro documentação:

Evo 3.2.10 A documentação interna do software é modificada, mantendo-se a data de acordo com um procedimento documentado [SMmm].

O mantenedor segue as recomendações do guia de correção e evolução sobre a necessidade de modificar a documentação interna do software e mantê-la atualizada. O mantenedor deve atualizar a documentação para que esta evolua com as correções e evoluções realizadas com o software.

Evo 3.2.11 A documentação externa do software é modificada, mantendo-se a data de acordo com um procedimento documentado [ISO 12207, 5.3.7.3].

A documentação externa é escrita para recursos individuias que não têm necessidade de referenciar o código-fonte, como analistas e designers. Esta documentação é mais geral e explica a o todo e suas inter-relações, ao invés de um componente específico. Como o software contém vários componentes, esta documentação muitas vezes usa diagramas e textos que explicam estas inter-relações. O mantenedor tem que atualizar a documentação existente, o mantenedor deve seguir as regras básicas com relação à [Pfleeger 2001, p. 321]:

i. Qual problema foi resolvido pela mudança e as opções consideradas
ii. Descrever o algoritmo ou a alteração feita na estrutura dos dados
iii. Documentar a modificação do dicionário de dados
iv. Atualizar a documentação para o usuário

3.5.1.3 Objetivos do KPA Evolução e Correção de Software

De acordo com [April, Abran, 2008], são listados abaixo os objetivos do KPA Evolução e Correção de Software:

i. As atividades de evolução e correção de Software não devem introduzir defeitos / falhas.
ii. O mantenedor deve utilizar a análise de impacto para acelerar as atividades de projeto detalhado.
iii. O mantenedor tem as ferramentas, o ambiente, e os mecanismos de apoio necessários.
iv. Os processos de modificação e correção de software devem melhorar a produtividade.
v. O software deve ser modificado em conformidade com o procedimento estabelecido, para certificar-se de não degradá-lo ao longo dos anos.
vi. Os testes devem ser realizados em conformidade com o procedimento estabelecido no pedido para certificar-se de não degradar o software ou de introduzir novos defeitos / falhas.
vii. Os resultados das especificações, de projeto detalhado, da execução de testes devem estar mutuamente coerentes e de acordo com as necessidades de cada solicitação.

3.5.1.4 Resultados esperados do KPA Evolução e Correção de Software

Ainda de acordo com [April, Abran, 2008], após o sucesso da implementação das práticas deste KPA, a seguir pode ser observado:

i. Os requisitos e quaisquer modificações negociados com o cliente são documentados.

ii. A modificação é documentada, há rastreabilidade entre as alterações e os componentes do projeto, bem como entre os elementos de codificação e os registros de teste.
iii. Os testes de unidade e de regressão são realizados, documentados, verificado e revisto.
iv. O mantenedor está apto a comunicar-se com o cliente para obter a aceitação da mudança e planejar o seu lançamento em produção.
v. O Software não sofre degradação, mas melhoras gradativas e contínuas.

3.5.2 KPA Verificação e Validação de Software

3.5.2.1 Roteiros do KPA Verificação e Validação de Software

Conforme [April, Abran, 2008], as atividades de verificação e validação da equipe de manutenção são semelhantes às dos desenvolvedores, porém endereçam problemas distintos. Validação demonstra que um serviço ou uma solicitação de manutenção, tal como previsto, irá funcionar como pretendido, enquanto que a verificação aborda, independente da solução, que o trabalho foi devidamente efetuado em relação aos requisitos da fase anterior, por exemplo, requisitos, projeto detalhado e programação. Ou seja, a validação assegura que "o trabalho correto foi feito", enquanto que a verificação garante que "o trabalho foi feito corretamente". Além disso, a equipe de manutenção deve especializar-se em testes de regressão.

3.5.2.2 Práticas do KPA Verificação e Validação de Software

De acordo com [April, Abran, 2008], segue o detalhamento das práticas do KPA Verificação e Validação de Software.

Evo 4.0.1 A organização de manutenção de software não realiza a verificação e validação das atividades no software mantido [SMmm].

A organização de manutenção não planeja atividades de verificação e validação em sua metodologia de manutenção. A gestão e gerentes de nível médio da manutenção do software não reconhecem que as atividades de verificação e validação são necessárias.

Evo 4.1.1 A organização de manutenção de software executa atividades para verificação e validação mas não de forma estruturada e coordenada [SMmm].

Cada engenheiro de manutenção trabalha de forma independente para verificar e validar pedidos de manutenção. O trabalho de verificação e validação é realizado de forma reativa, há pouca documentação sobre atividades ou acompanhamento (não há evidência que os serviços de manutenção estão sujeitos a qualquer verificação e validação) [Pressman, 2011].

Evo 4.2.1 As atividades de verificação e validação são documentados, conhecidas e planejadas. Os planos são documentados de acordo com um procedimento documentado [SMmm; ISO 90003, 5.7.2; 1 TIL 2007c, 4.5.5].

As atividades para teste são descritas e as responsabilidades divulgadas. A definição das ferramentas e ambientes também é notória. Um processo está configurado e documentado, as técnicas para testes e avaliações são examinados e documentados.

Um modelo para plano de teste é criado, as técnicas para testes e reviews são examinados e documentados, o planejamento de reviews e testes deve ser realizado em cada projeto para a transição e para solicitações de modificação. Assim, espera-se que as atividades específicas para verificação e validação sejam identificados nos planos formais para testar uma solicitação.

Espera-se também que a organização tenha conhecimento de aspectos importantes que influenciam os testes para garantir o seu sucesso. Estes planos são coordenadas com outros planos, se necessário. Estas metas simples são estabelecidas e a responsabilidade para cada tipo de teste é atribuído. A

complexidade dos planos mudam de acordo com o esforço associado a cada alteração.

Seguem as práticas relacionadas ao Roteiro Reviews:

Evo 4.2.2 Reviews e verificação/validação são realizadas, de forma seletiva, após a produção de alguns produtos chave intermediários [SMmm; ISO 90003, 7.3.1 b].

A organização de manutenção executa reviews seletivos em alguns produtos julgados mais críticos. A review não é necessariamente realizada de forma independente, mas é realizada formalmente e registros de solicitações são acessíveis. Reviews formais e projetos detalhados, análise de impacto, neste caso, resultará em melhor qualidade.

Evo 4.2.3 Atividades de fornecimento de manutenção são seletivamente revisados em determinados passos [SEI 2002, PA166.IG102.SP102.SubP103.N101].

Reviews podem ser formais ou simples e incluem os seguintes passos:

i. Prepara-se a review;
ii. Certifica-se que as partes interessadas participem da review;
iii. Gerenciamento da review;
iv. Identificar, documentar e acompanhar todos os itens a serem corrigidos após a review;
v. Preparar e distribuir a ata da review aos interessados;

O SEI identifica a revisão técnica, que objetiva verificar se o produto intermédio atende ou não aos padrões e requisitos. [SEI 2002 PA166.IG102.SP102.SubP104]. Esta avaliação técnica inclui as seguintes atividades [SEI 2002 A166.IG102.SP102.SubP104.N101]:

i. Assegurar que a equipe tenha entendido bem as necessidades e exigências do cliente;
ii. Revisão das atividades técnicas, verificação se interpreção e implementação do projeto detalhado foi efetuada corretamente e determinação se os produtos estão em conformidade com as normas;
iii. Assegurar que as responsabilidades técnicas sejam atendidas e que os questionamentos são comunicados e resolvidos de uma forma oportuna;
iv. Fornecimento de informações adequadas e apoio técnico para o fornecedor.

Seguem as práticas relacionadas ao Roteiro testes de aceitação:

Evo 4.2.4 O teste de aceitação é realizada de acordo com um procedimento documentado para mostrar que o produto modificado está em conformidade e atende as necessidades do cliente [ISO 90003, 5.7.4 e 5.7.5].

O teste de aceitação (incluindo os testes funcionais e testes de desempenho) para solicitações regulares é realizada e documentada para mostrar que a modificação do software atende aos requisitos do cliente.

Evo 4.2.5 Testes de regressão são sistematicamente e formalmente realizados em todas as partes do software afetadas por modificações, incluindo correções [ISO 14764, 83.2.2; SMmm].

A ISO 14764 identifica uma necessidade específica durante a manutenção, que é para assegurar que os componentes existentes do software que não foram alterados são verificados para garantir que não foram afetados indiretamente por estas alterações [ISO 14764, 8.3.2.2 a e b]. [ISO 14764, 8.3.2.2 a e b]. O mantenedor deve utilizar técnicas de teste regressão para verificar todo o sistema.

Seguem as práticas relacionadas ao Roteiro mover para produção uma mudança ou versão:

Evo 4.2.6 A coordenação da instalação do teste no local e da manutenção de software com o utilizador são realizadas de acordo com um procedimento documentado [90003 ISO, 5.9.3; SMmm].

A implementação final é realizada de acordo com cliente, e segue um procedimento documentado.

Evo 4.2.7 As Práticas relacionadas com a instalação de modificações de software (incluíndo a documentação de instalação) são planejados para reduzir interrupções do funcionamento diário do Sistema [SMmm].

Evo4.2.8 O engenheiro de manutenção participa de testes de aceitação durante o transição final do software do subcontratado e fornecedor, de acordo com um procedimento documentado [ISO 90003, 6.7.3; SMmm].

3.5.2.3 Objetivos do KPA Verificação e Validação de Software

Conforme [April, Abran, 2008], os objetivos do KPA Verificação e Validação de Software são:

i. Certificar-se de que as atividades de verificação e validação são planejadas, a fim de ter a certeza de que serão realizadas;
ii. Garantir que os procedimentos de execução estabelecidos são seguidos;
iii. Certificar-se de que o produto de trabalho de manutenção atende aos requisitos especificados;
iv. Mostrar que os produtos resultantes dos serviços de manutenção foram executados conforme projetado;
v. Prever que ambos, produtos e manutenção estão sujeitos a revisão antes da sua entrega aos clientes;
vi. Descobrir defeitos antes da entrega aos clientes.

3.5.2.4 Resultados esperados do KPA Verificação e Validação de Software

De acordo com [April, Abran, 2008], os objetivos do KPA Verificação e Validação de Software são:

i. Técnicas de verificação e validação são adaptados para manutenção;

ii. A existencia de procedimentos que descrevem como realizar verificação e validação no contexto de manutenção e por tipo de serviço;

iii. Testes de regressão adequada é realizado para cada solicitação de manutenção.

3.6 Trabalhos relacionados

Ao pesquisar sobre trabalhos relacionados foram identificados artigos que abordam o modelo de maturidade de manutenção de software, dentre eles, há um artigo que introduz o modelo de maturidade proposto para a manutenção de software. O artigo [April et al., 2005] apresenta o modelo de maturidade de manutenção de software (SMmm) proposto para avaliar a maturidade da função de manutenção de software e ajudar na identificação de atividades de melhoria. Neste trabalho são descritos as principais características, fundamentos, arquitetura do modelo. Para ilustrar resumidamente o conteúdo detalhado do SMmm, o artigo apresenta os objetivos de um KPA, juntamente com as práticas detalhadas dos níveis 1-3 e por fim apresenta uma visão geral do processo de validação inicial do modelo. Além disso, faz uma análise do estado da prática de manutenção de software e identifica, baseado na experiência do setor, nos padrões internacionais e na literatura sobre manutenção de software os principais processos e especialmente o que torna a manutenção única e diferente do desenvolvimento de software. Também apresenta um inventário de modelos de maturidade de engenharia de software, identifica aqueles que incluem tópicos de manutenção de software e discute as limitações do modelo CMMI com relação a

atividades exclusivas de manutenção de software. O artigo sugere que estudos empíricos sobre o uso da SMmm como ferramenta para melhorias contínuas na gestão da manutenção podem contribuir para o desenvolvimento de uma melhor compreensão dos problemas da função de manutenção de software.

O artigo [Paquette 2006] apresenta um estudo de caso, em uma organização, que ilustra a avaliação da maturidade de manutenção utilizando o modelo SMmm, foi desenvolvido um método SMmm de mini-avaliação. Este método produziu uma classificação da maturidade confiável, sem o investimento de recursos indisponíveis e também permitiu a seleção de componentes de avaliação individuais para concentrar a investigação em problemas específicos e para adaptar o âmbito de aplicação do esforço de avaliação e classificação de um nível adequado para a organização de manutenção de software.

O procedimento de avaliação desenvolvido neste estudo de caso foi um questionário, contendo as práticas do modelo SMmm para os níveis de maturidade 0, 1 e 2, originado das perguntas / declarações abordadas durante as reuniões envolvendo recursos de manutenção de software e da administração. O artigo ressalta que as reuniões também ajudaram na troca de informações e na sensibilização das práticas do modelo SMmm em cada nível.

Os resultados da avaliação do modelo SMmm permitiu compreender melhor o estado de trabalho atual da manutenção do software avaliado, apresentando os pontos fortes e fracos da manutenção e o seu nível de maturidade correspondente, bem como as questões de manutenção decorrentes de algumas falhas do processo de desenvolvimento. A avaliação da manutenção do software através do modelo SMmm permitiu a esta organização identificar as fraquezas do processo e apresentar um roteiro para a mudança para sua equipe de gerenciamento.

Um estudo conduzido por [April, Abran, 2009], apresenta a visão geral de algumas práticas de medição utilizadas para os níveis 3 e 4 do modelo de maturidade de manutenção de software SMmm. O artigo salienta a importância da compreenção do escopo das atividades de manutenção e o contexto em que os mantenedores de software trabalham diariamente. Além disso, apresenta os tópicos de medição de manutenção de software fornecendo uma visão geral das fontes de informação usadas pelo SMmm para medir processos, produtos e serviços de manutenção de software. Este trabalho destaca a necessidade de

estudos empíricos para verificar a eficiência do modelo como uma ferramenta de melhoria contínua na gestão da manutenção.

As experiências da implementação de práticas de Programação extrema (do inglês eXtreme Programming) - XP em projetos de manutenção de software são apresentadas em [Poole, Huisman; 2001] e [Svensson, Host; 2005]. A adoção das práticas de XP para a realização de atividades de manutenção é discutida em [Poole, Huisman; 2001]. Os resultados da introdução de XP em um ambiente de manutenção e evolução de software são apresentados em [Svensson, Host; 2005]. Este estudo conclui que muitas das práticas de XP têm de ser modificadas se quiserem trabalhar em ambientes de manutenção. Esses dois últimos estudos focam na adaptação das práticas e / ou princípios de XP para o propósito de realização de práticas de manutenção de software. No entanto, estas propostas não descrevem explicitamente como gerir e liderar as atividades e práticas envolvidas na manutenção do software. Também não consideram outras atividades e práticas que são exclusivas e fundamentais para a manutenção de software.

Uma proposta que trata da manutenção de software em grande detalhe é o modelo de maturidade de manutenção corretiva (CM^3), o qual foi apresentado e discutido em [Kajko-Mattsson, Forssander, Olsson, 2001]. CM^3 é um modelo de processo para manipular explicitamente a categoria de manutenção relacionada à manutenção corretiva.

Este modelo consiste em vários processos (que desempenham uma tarefa claramente definida) colaborando uns com os outros. Cada processo tem uma estrutura bem definida, fornecendo orientação detalhada para as organizações ao construir ou melhorar seus processos de manutenção [Kajko-Mattsson, 2002]. Este modelo foca apenas um tipo específico de manutenção; As organizações precisam de uma proposta que lhes permita abordar adequadamente todos os tipos de manutenção. Nesse sentido, é importante ressaltar que, no mundo real, as solicitações de manutenção dos clientes não são filtradas previamente pelo tipo de manutenção antes que elas cheguem até os mantenedores.

4 Processo de manutenção derivado do modelo SMmm

Este capítulo apresenta a formalização de um conjunto de atividades de manutenção derivadas das práticas do modelo SMmm [April, Abran 2008], sintetizada em um processo definido adaptado ao modo de trabalho da organização utilizada neste trabalho.

4.1 Escopo do processo definido

Uma área de processo chave possui um conjunto de atividades relacionadas que, quando realizadas adequadamente, atendem um conjunto de objetivos considerados importantes para aumentar a capacidade do processo.

O capítulo 3 apresentou o modelo SMmm, detalhando o domínio de processo Engenharia de evolução de Software, as duas áreas- chave de processo foco deste trabalho são: Evolução e correção de software e Verificação / validação de software. Este processo definido, implementa onze práticas destes dois KPAs, como resume a tabela 3.

Visando identificar quais práticas seriam aplicáveis ao contexto do sistema em questão, o critério de seleção utilizado levou em consideração as necessidades de melhoria do sistema piloto. As práticas foram analisadas conforme suas características: práticas relacionadas planejamento da modificação; relacionadas com a fase de design, construção, testes e, por fim, relacionadas à revisão do artefato. As práticas adotadas foram:

- Roteiro Design Detalhado

i. Evo 3.2.1 – Projeto detalhado da solicitação de manutenção,
ii. Evo 3.2.2 – Atividades de investigação e design associados a falha ambas

- Roteiro Evolução/Correção

i. Evo 3.2.3 – Atividades de implementação (programação)
ii. Evo 3.2.6 – As inter-relações (rastreabilidade) são atualizados quando o software é corrigido e/ou evoluído.

- Roteiro de Teste (Unidade, Integração e Regressão)

i. Evo 3.2.7 – Os testes de unidade e regressão devem ser realizados para cada componente referente a modificação.

ii. Evo 3.2.8 - Os testes de integração e regressão devem ser realizados para cada componente referente a modificação.

- Roteiro de revisão

i. Evo 4.2.1 – Atividades de verificação e validação são conhecidas e planejadas

ii. Evo 4.2.2 – Revisão e verificação/validação são realizadas de forma seletiva

iii. Evo 4.2.3 – Atividades de manutenção são revisadas em determinados passos

- Roteiro de Testes de Aceitação

i. Evo 4.2.4 – O teste de aceitação é realizado de acordo com procedimentopara mostrar que o artefato modificado está em conformidade e atende as necessidades do cliente

ii. Evo 4.2.5 – Teste de regressão são sistematicamente realizados em todas as partes do software afetadas por modificações incluindo correções

Domínio de processo: Engenharia de evolução de software	
KPA: Evolução e correção de software	**KPA:** Verificação / validação de software
Evo 3.2.1	**Evo 4.2.1**
Evo 3.2.2	**Evo 4.2.2**
Evo 3.2.3	**Evo 4.2.3**
Evo 3.2.6	**Evo 4.2.4**
Evo 3.2.7	**Evo 4.2.5**
Evo 3.2.8	

Tabela 3: Práticas do SMmm selecionadas para o estudo de caso.

4.2 Plataforma Team Foundation Service – TFS

A plataforma adotada foi o Team Foundation Service – TFS, uma solução amplamente utilizada no mercado, baseada em um conjunto de ferramentas que funcionam integradas para proporcionar a gestão do ciclo de vida de uma aplicação de software, esta solução oferece: colaboração, controle de qualidade, integração contínua.

4.3 Mapeamento das práticas do modelo SMmm e TFS

Com a intenção de verificar se o TFS dá suporte a implementação das práticas mencionadas na sessão 4.1, será apresentado nesta sessão um mapeamento das funcionalidades do TFS relacionado as práticas selecionadas. Neste mapeamento considera-se que o TFS "Implementa", "Implementa parcialmente" ou "Não implementa" uma prática do modelo SMmm, como sumariza a tabela 3.

Pontuação	Significado
Implementa	TFS implementa completamente a prática do modelo SMmm
Implementa parcialmente	TFS implementa parcialmente a prática do modelo SMmm
Não implementa	TFS não implementa adequadamente a prática do modelo SMmm

Tabela 4: Pontuações do mapeamento e seus significados

Evo 3.2.1 O projeto detalhado da solicitação de manutenção elabora as necessidades do usuário, e é desenvolvida de acordo com um procedimento documentado [ISO 12207, 5.3.6; IEEE 1219, 4.3; SMmm].

Análise: O Visual Studio Team System possui uma funcionalidade através da qual podemos definir a criação de um documento de visão totalmente integrado ao TFS. Os Tipos de item de trabalho (Work Item Types - WITs) são uma espécie de

formulários para entrada de dados, que possibilita capturar informações necessárias para rastrear e gerenciar o fluxo de trabalho, por exemplo:

i. Definir prioridade
ii. Especificar estimativas
iii. Alterar status
iv. Inserir as informações
v. Vincular a outros objetos ou itens de trabalho

Esta funcionalidade pode receber os seguintes dados de entrada:

i. Descrição/detalhamento dos itens de escopo;
ii. Descrição do requisito;
iii. Atividades que serão implementadas;.
iv. Documento de regras de negócio;
v. Documento de análise de impacto;

Sob a forma de textos redigidos nos campos do formulário ou anexos de documentos como arquivos .doc ou planilhas no formato Excel e produz como resultado um documento de visão. O TFS possui total controle sobre os Work Item Types.

Diagnóstico: Implementa.

Evo 3.2.2 As atividades de investigação e de design associado a uma falha são realizadas de acordo com um procedimento documentado [SMmm].

Análise: O Visual Studio Team define um WIT de bug (problema) para correção de defeitos. Pode receber como dados de entrada:

i. Descrição do comportamento ocorrido;
ii. Descrição do comportamento esperado
iii. Descrição dos passos necessários para reprodução do defeito.
iv. Descrição dos critérios de aceite
v. Análise de impacto

vi. Plano de testes

Produz como resultado um documento de visão de Bug.

Diagnóstico: Implementa

Evo 3.2.3 As atividades de implementação (de programação) são executadas de acordo com um procedimento documentado [SMmm].

Análise: O TFS oferece suporte necessário as atividades de implementação provendo a gestão do código fonte. O Visual Studio oferece uma IDE para codificação e, além disso, provê alguns recursos como code review, cobertura de código e uma estrutura para testes de unidade integrada.

O gerenciamento de código fonte é feito por um componente do TFS chamado de TFVC (Team Foundation Version Control), que oferece diversos outros recursos que facilitam o controle do código fonte.

O Team Foundation Build (servidor de build do TFS) permite automação na criação de build viabilizando a integração continua, garantindo a realização de diversas ações durante a geração da versão do software, como validação dos testes unitários integrados e políticas do projeto.

Diagnóstico: Implementa.

Evo 3.2.6 As inter-relações (rastreabilidade) entre os procedimentos administrativos, os módulos, os processos e os dados são atualizados quando o software é corrigido e/ou evoluído [SMmm].

Análise: O TFS promove um ambiente integrado, com toda a comunicação baseada em item de trabalho (Work item), quando uma build é gerada, o MSBuild automaticamente identifica e associa as changesets, work items e outros artefatos com a build. Com isso, você consegue ter uma rastreabilidade entre requisitos, código fonte, build e resultados de testes, proporcionando rastreabilidade entre estes, atendendo a um resultado esperado pelo modelo SMmm.

Diagnótico: Implementa.

Evo 3.2.7 Os testes de unidade devem ser executados para cada componente referente a modificação. [IEEE 1219,4.4.2.1].
Evo 3.2.8 Os testes de integração e de regressão devem ser realizados para cada componente referente a modificação [SMmm].

Análise: Com a solução do Visual Studio é possível elaborar teste de unidade e teste de componentes.

O MSBuild possibilita vincular os testes de unidade e de componentes com o Build, para execução automática destes logo após a compilação da Solution.

O TFS funciona como um repositório, permitindo que o Visual Studio faça conexão para consumir os testes armazenados.

Diagnóstico: Implementa.

Evo 4.2.5 Testes de regressão são sistematicamente e formalmente realizados em todos as partes do software afetadas por modificações, incluindo correções [ISO 14764, 83.2.2; SMmm].

Análise: O Microsoft Test Manager (MTM), é uma ferramenta de gerenciamento de testes, baseada na elaboração de plano de testes e criação e execução de casos de testes. O MTM também possibilita executar testes automatizados que foram gerados através do Visual Studio.

O TFS permite conexão com MTM para consumo dos testes armazenados, proporcionando a integração entre os casos de teste e o item de trabalho e consequentemente ao código fonte correspondente.

Com auxílio do MSBuild, a cada check-in é possível executar testes de automatizados garantindo que as funcionalidades existentes não foram quebradas (teste de regressão);

Diagnóstico: Implementa.

Evo 4.2.4 O teste de aceitação é realizada de acordo com um procedimento documentado para mostrar que o produto modificado está em conformidade e atende as necessidades do cliente [ISO 90003, 5.7.4 e 5.7.5].

Evo 4.2.1 As atividades de verificação e validação são documentados, conhecidas e planejadas. Os planos são documentados de acordo com um procedimento documentado [SMmm; ISO 90003, 5.7.2; 1 TIL 2007c, 4.5.5].

Análise: O recuso Work item possui um campo específico para descrição de critérios de aceite do item de trabalho, os dados informados neste campo servem como diretrizes para elaboração de um plano de teste de aceitação. Com a possibilidade de vinculação do Work item com caso de teste do MTM proporcionado pelo TFS, é possível executar os testes e gravar a execução dos casos de teste.

Diagnóstico: Implementa.

Evo 4.2.2 Reviews e verificação/validação são realizadas, de forma seletiva, após a produção de alguns produtos chave intermediários [SMmm; ISO 90003, 7.3.1 b].

Evo 4.2.3 Atividades de fornecimento de manutenção são seletivamente revisados em determinados passos [SEI 2002, PA166.IG102.SP102.SubP103.N101].

Análise: As atividades de verificação e validação podem ser planejadas e executadas com auxílio do conjunto de ferramentas relacionadas com a garantia da qualidade, o Visual Studio Testing Tools na qual toda comunicação entre as funcionalidades deste conjunto de ferramentas é orquestrado pelo TFS.

Diagnóstico: Implementa

Assim como sumariza a tabela 5, o TFS corresponde suficientemente bem à implementação das práticas selecionadas, oferecendo o suporte necessário para que as atividades possam ser desenvolvidas segundo as exigências do modelo SMmm.

Área-chave de processo	Práticas	Pontuação
Evolução e correção de software	**Evo 3.2.1**	Implementa
	Evo 3.2.2	Implementa
	Evo 3.2.3	Implementa
	Evo 3.2.6	Implementa
	Evo 3.2.7	Implementa
	Evo3.2.8	Implementa
Verificação e validação de software	**Evo 4.2.1**	Implementa
	Evo 4.2.2	Implementa
	Evo 4.2.3	Implementa
	Evo 4.2.4	Implementa
	Evo 4.2.5	Implementa

Tabela 5: Mapeamento entre TFS e práticas do modelo SMmm

4.4 Descrição do Processo definido

4.4.1 Introdução

Segundo [Pfleeger, 2004], um processo de software pode ser visto como um conjunto de atividades, métodos, práticas e transformações utilizadas para desenvolver e manter um sistema de software e seus produtos relacionados (plano de projeto, documentos de design, código, casos de teste, manuais de usuário), a figura 10 ilustra esta definição.

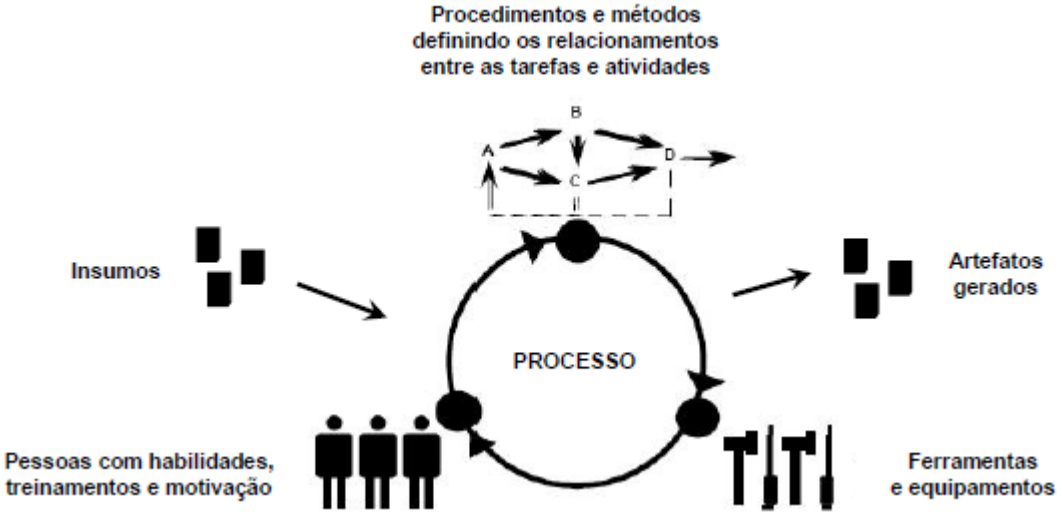

Figura 10: Visão integrada de um processo

Fonte: [Pessoa, 2003]

- Insumo

É elemento necessário para a realização de uma tarefa ou atividade, também pode ser um elemento de saída de outras atividades ou tarefas.

- Artefato gerado

É um resultado de uma atividade, por exemplo:

i. Documento revisto e aceito;
ii. Módulo implementado, testado e aceito;
iii. Construto integrado, testado.

- Atividade

É o conjunto de tarefas que levam a um ou mais artefatos de qualidade controlada.

- Ferramentas e equipamentos

Auxiliam a execução das atividades e tarefas dos processos, podem automatizar partes da execução das atividades.

4.4.2 Fases do Processo

O processo definido é composto por três fases: Planejamento da modificação, Implementação da modificação, Revisão/Validação da manutenção como mostra a figura 11. O objetivo de cada fase é descrito a seguir:

Planejamento da modificação

O objetivo desta fase é efetuar planejamento e analise do item de trabalho, as tarefas desta fase são importantes por contribuírem para o entendimento do escopo dos itens de trabalho, para a mitigação de riscos e para a garantia da qualidade geral, servem como guia para a implementação da alteração.

Implementação da modificação

Nesta fase ocorre a implementação (codificação) da modificação de acordo com os aspectos identificados na fase de análise, após modificação dos módulos e estruturas ocorre a validação dos mesmos. O objetivo é modificar o produto de software existente, preservando a sua integridade.

Revisão/validação da manutenção

Nesta fase são efetuadas revisões para verificar a integridade do sistema e se o resultado obtido foi satisfatório de acordo com o que foi especificado. A validação refere-se a um conjunto de tarefas que asseguram que a modificação foi efetuada de maneira correta.

Figura 11: Fases do Processo de Manutenção

4.4.3 Atividades do Processo

Nessa seção, são descritas as atividades para as três fases do processo de manutenção proposto, cada subseção está relacionada a uma fase do processo.

4.4.3.1 Planejamento da modificação

Durante essa fase deve-se entender a demanda e definir as ações que serão realizadas durante a manutenção. Esta fase é composta das seguintes atividades:

Atividade: Identificação
Artefatos de Entrada: Item de trabalho cadastrado no TFS
Artefatos de Saída: Descrição detalhada do item de trabalho (documento de visão)
Tarefas:

i. Compreensão dos requisitos do item de trabalho.
ii. Classificação quanto ao tipo de tipo de manutenção (corretiva, adaptativa, perfectiva ou preventiva).

iii. Identificação, de acordo com a necessidade, de quais unidades de software, versões e documentação relacionada precisam ser modificadas.
iv. Identificação do alcance da alteração (tamanho da modificação).
v. Descrição dos critérios de aceite do item de trabalho.
vi. Priorização

Atividade: Análise
Artefatos de Entrada: Item de trabalho cadastrado no TFS
Artefatos de Saída: Resultado da análise, plano de implementação, plano de testes.
Tarefas:

i. Analise de impacto da alteração.
ii. Identificação de possíveis ameaças (riscos) para o sistema, conflitos potenciais e respectivas alternativas para sana-los.
iii. Discussão de alternativas para a implementação da modificação.
iv. Elaboração do plano de teste para as alterações (requisitos de teste, casos de teste e especificações de teste).
v. Identificação de testes de regressão aplicáveis.

4.4.3.2 Implementação da modificação

Esta fase é composta das seguintes atividades:

Atividade: Codificação
Descrição: Implementação da modificação de acordo com os aspectos identificados na atividade de análise.
Artefatos de Entrada: Insumos gerados na fase de planejamento/análise.
Artefatos de Saída: Código fonte dos itens alterados.
Tarefas:

i. Alteração do código fonte, de acordo com o padrão de codificação acordado.
ii. Refatoração do código fonte objetivando melhoria na engenharia do sistema.

iii. Solicitar revisão de código (code review).

Atividade: Testes (unidade, interação e regressão).
Artefatos de Entrada: Código fonte dos itens alterados, plano de testes.
Artefatos de Saída: Resultado dos testes executados.
Tarefas:

i. Testes de unidade e módulo das alterações efetuadas são realizados para verificar a presença de erros e comportamento adequado em nível das funções e módulos básicos do sistema.
ii. Testes de integração das alterações efetuadas são realizados para averiguar a reunião dos diferentes módulos e a interação entre estes quando operando em conjunto.
iii. Elaboração de casos de teste para as alterações. Um caso de teste define um cenário, um conjunto de ações e dados que visam testar um determinado aspecto do artefato sob teste – AST.
iv. Testes de regressão são realizados no sistema para assegurar-se que as mudanças feitas no código não afetaram nenhuma funcionalidade já existente.

4.4.3.3 Revisão/Validação da manutenção

Nesta fase é conduzida reunião de revisão com a parte interessada para assegurar a integridade do sistema modificado e assim obter a concordância da mesma, de que a tarefa foi realizada satisfatoriamente, de acordo com o especificado. Esta fase é composta das seguintes atividades:

Atividade: Revisão do item de trabalho
Descrição: Avaliações técnicas de artefatos específicos, para verificar se estão conformes com padrões e especificações e se, eventuais modificações nos artefatos foram efetuadas de maneira correta.
Artefatos de Entrada: Código fonte dos itens alterados.
Artefatos de Saída: Resultado da revisão.
Tarefas:

i. Efetuar revisão do código fonte.
ii. Checar os itens comentados e validá-los.

Atividade: Teste de aceitação

Artefatos de Entrada: Regras do critério de aceite elaborado na fase de planejamento.

Artefatos de Saída: Resultado dos testes.

Tarefas:

i. Teste de aceitação das alterações efetuadas.
ii. Revisão e atualização da documentação do sistema são feitas, se necessário, para manter a adequação com a documentação original.

5 Estudo de caso

5.1 Introdução

Este capítulo apresenta o estudo de caso, realizado em ambiente real, com o intuito de avaliar os benefícios proporcionados pela utilização de algumas práticas de manutenção proposto pelo modelo SMmm [April, Abran, 2008] e assim direcionar possíveis ações de melhoria para o sistema.

Segundo [Mafra e Travassos, 2006], estudo de caso é uma estratégia de pesquisa com o propósito de se investigar e entender uma entidade ou fenômeno dentro de um determinado espaço ou intervalo de tempo específico. Na Engenharia de Software os estudos de caso podem ser utilizados para monitorar e avaliar atributos presentes em projetos, atividades ou atribuições através de análises quantitativas e/ou qualitativas.

5.2 Planejamento do Estudo de Caso

5.2.1 Estudo preliminar

Um estudo preliminar foi realizado na organização a fim de obter um diagnóstico da situação atual no que se refere à manutenção de software, visando delinear estratégias para um processo de manutenção de software. Esse diagnóstico foi executado através da extração de informações do repositório de alterações do código fonte.

5.2.1.1 Cenário inicial

O estudo de caso foi conduzido em uma organização privada do setor de comércio eletrônico, fundada em maio de 2000, com capital 100% nacional e está localizada na cidade do Rio de Janeiro, com cerca de 80 colaboradores. A equipe que mantém o sistema onde o estudo foi realizado é composta por oito integrantes: seis desenvolvedores, um líder técnico e um coordenador. O sistema estudado utiliza a linguagem C# (C Sharp) com .NET Framework, foi concebido em 2005 e possui mais de 150 mil linhas de código, atendendo, no momento da pesquisa, a

uma base de 30 clientes. O software estudado corresponde a um sistema voltado à conciliação de vendas a crédito, ou seja, ao procedimento de conferir e confrontar o extrato fornecido pela operadora de cartão de crédito com o extrato de controle interno do lojista.

Devido às demandas apresentadas pelos clientes, que diariamente solicitam melhorias em relatórios, ajustes em módulos em produção, novas funcionalidades e correção de bugs, o sistema sofreu, ao longo do tempo, muitas modificações. A equipe não planejava nem possuía processo para administrar as atividades de manutenção do código. Este fator contribuía para um índice considerável de introdução de defeitos, retrabalho e desperdício.

5.2.1.2 Análise e Interpretação dos dados preliminares

Para [Yin, 1984], a análise dos dados consiste na tabulação, exame ou recombinação das evidências coletadas, buscando compreender, esclarecer, validar ou refutar os objetivos iniciais do estudo.

A metodologia de análise buscou estudar o repositório de dados na qual constava os registros das manutenções. Inicialmente na figura 12, é apresentado o quantitativo de demandas registradas mensalmente no período entre janeiro e outubro de 2015.

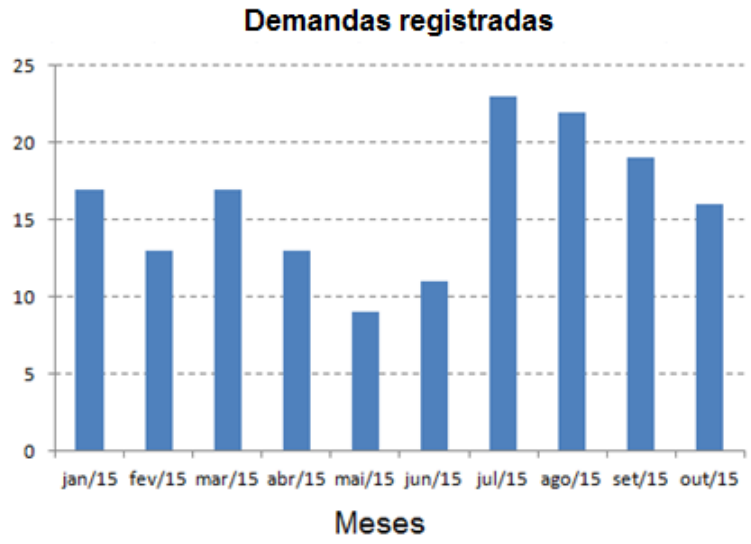

Figura 12: Quantitativo de demandas de manutenção

Não houve uma regularidade na quantidade de demandas mensais. Em julho foi registrado o maior número de demandas, uma razão plausível para o aumento das demandas pode ser o fato das entregas no mês ter acarretado falhas decorrentes de defeitos provocados em outros módulos do sistema.

Do ponto de vista da classificação dos tipos de manutenção de software (adaptativa, perfectiva, corretiva e preventiva) definidas no capítulo 2, realizou-se a quantificação das manutenções efetuadas relativa a esta classificação, o resultado obtido é apresentado na figura 13.

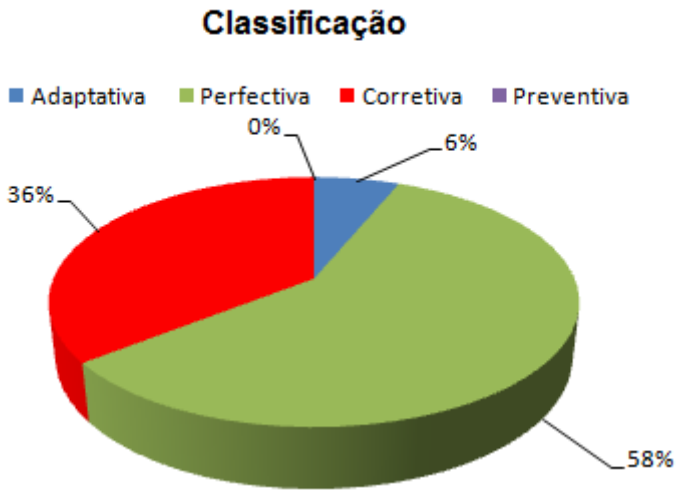

Figura 13: Classificação dos tipos de manutenções – estudo preliminar

Ao observar o gráfico apresentado na figura 14, é possível identificar que o maior esforço da equipe está em atender solicitações de evolução do sistema. A manutenção preventiva não possui a atenção necessária. Um dos grandes problemas da manutenção é ela tender a desestruturar o sistema, tornando manutenções futuras mais complicadas e tendentes a injeção de defeitos, a preocupação em melhorar características de confiabilidade ou manutenibilidade do código poderia evitar futuros problemas.

As atividades de manutenção eram efetuadas sem planejamento ou método, em razão da necessidade de atender ao volume de demanda dos clientes, geralmente ligada a prazos muito curtos, o que muitas vezes, resultou em soluções incompatíveis com consequências negativas.

Dentre as demandas de manutenção, a figura 14 demonstra que 43% das demandas levaram de 6 a 10 dias, a partir do momento em que foi iniciado até sua conclusão e 30 % tiveram sua conclusão no período de 1 a 10 dias.

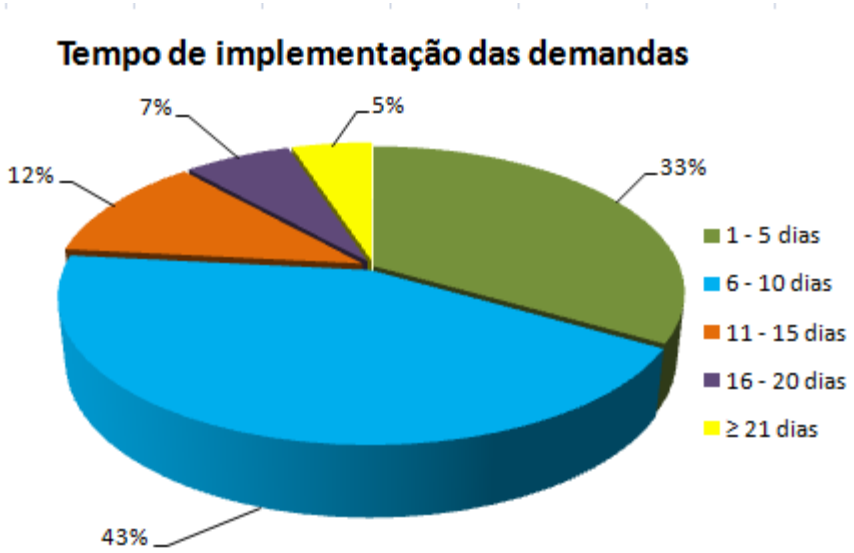

Figura 14: Tempo de implementação das demandas de manutenção

Na análise seguinte, verificaram-se os cinco módulos representativos do software que mais foram alterados no período entre janeiro e outubro de 2015 (por questões de confidencialidade foram referenciados por letras), traçando um paralelo destes com os tipos de alterações, temos como resultado a ilustração da figura 15.

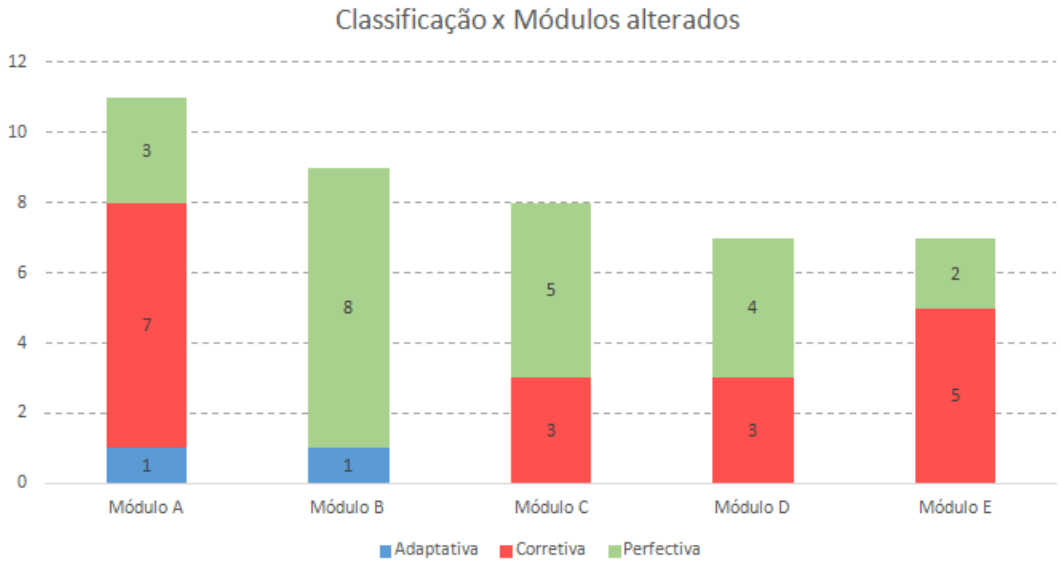

Figura 15: Distribuição entre os cinco módulos mais alterados

Como podemos verificar na figura 15, o módulo A, obteve o maior número de alterações, das quais a maior parte foi decorrente de correções de defeitos, ao investigar os defeitos do módulos A identificamos quatro ocorrências de defeitos reincidentes.

Erros são inevitáveis em qualquer sistema. A grande questão é minimizar a sua ocorrência e impedir que os mesmos se repitam. Quando um defeito é reincidente, significa que não houve o tratamento adequado do problema, do contrário, não teria ocorrido mais de uma vez.

Como ponto de melhoria para este tipo de problema, uma estratégia, poderia ser concentrar esforços nos motivos que culminaram no defeito. Quando a causa é localizada, a mesma deve ser combatida e eliminada completa e corretamente para que não haja reincidência.

A figura 16 apresenta um gráfico com a distribuição da ocorrência de defeitos reportados durantes o período entre janeiro 2015 a outubro 2015.

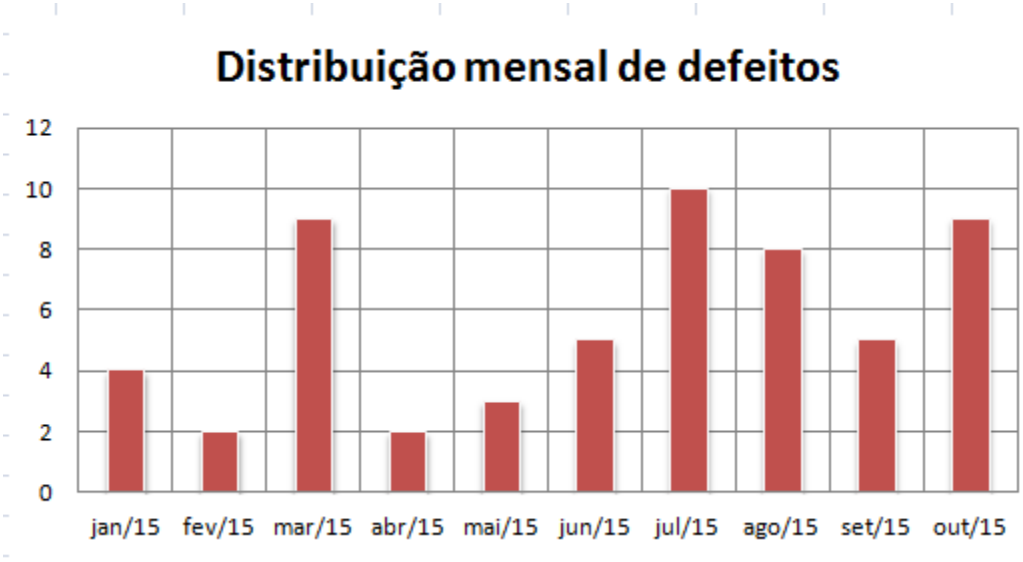

Figura 16: Distribuição dos defeitos

O crescimento da quantidade de casos de teste existente ao longo do período entre janeiro e outubro de 2015 foi baixo, conforme podemos observar na figura 17. Entre os meses de abril e maio foram adicionados poucos casos de teste, isto pode ter acarretado o aumento do número de defeitos no mês de julho, de acordo com a figura 16.

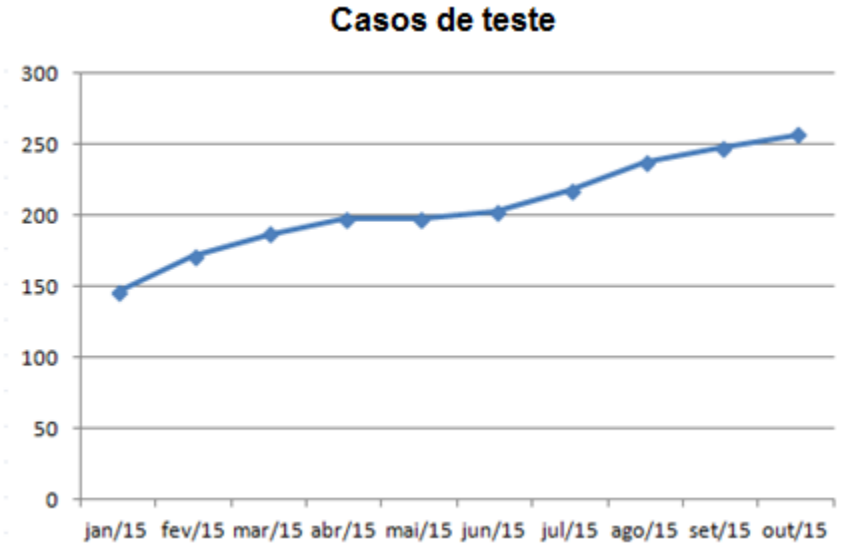

Figura 17: Distribuição da quantidade de casos de teste

A realização de muitas demandas evolutivas sem o cuidado e prevenção da injeção de defeitos, a existência de poucos casos de testes para verificar falhas, ocasionou elevação da quantidade de defeitos. A partir desse diagnóstico, o próximo passo será identificar e planejar a implantação de melhorias relacionadas a atividades de manutenção.

5.2.2 Planejamento da implantação das melhorias

Diante da análise efetuada com base no estudo preliminar, ficou clara para os membros da equipe a necessidade de mudanças na maneira como eram conduzidas as atividades de manutenção do sistema, com isso, deu-se início a um busca por alternativas de melhorias com objetivo de reduzir a ocorrência de defeitos e também aumentar a manutenibilidade do sistema.

Após a realização de estudos e pesquisas por soluções para as questões abordadas no estudo preliminar foram encontrados na literatura, artigos científicos abordando roteiros para atividades direcionadas a manutenção de software. Ao efetuar pesquisas com maior profundidade sobre o assunto identificamos que as atividades propostas pelo SMmm poderiam ser utilizadas para projetar melhorias no processo de manutenção de software.

O próximo passo foi estudar o conjunto de práticas sugeridas pelo modelo SMmm e avaliar quais eram elegíveis para serem implementadas de acordo com necessidade do sistema. Os estudos realizados nesta etapa foram determinantes para fundamentar a escolha. Foi elaborado uma proposta de solução abordando a implementação de algumas práticas do modelo de maturidade para atividades de manutenção de software SMmm e apresentada a coordenação, que em comum acordo, com a equipe de desenvolvimento decidiram adotar a solução apresentada.

O planejamento para implantação das práticas se baseava na criação de um plano de ação, denominado Plano de Implantação que identifica as atividades a serem executadas e os recursos necessários para a implantação das práticas. A elaboração de um plano de implantação das práticas foi determinante para que todas as etapas fossem realizadas com os cuidados necessários e para que o desenvolvimento de novas atividades de manutenção que agregassem valor para a

organização tivesse continuidade. O plano de implantação das práticas foi dividido em três etapas, como ilustra a figura 18.

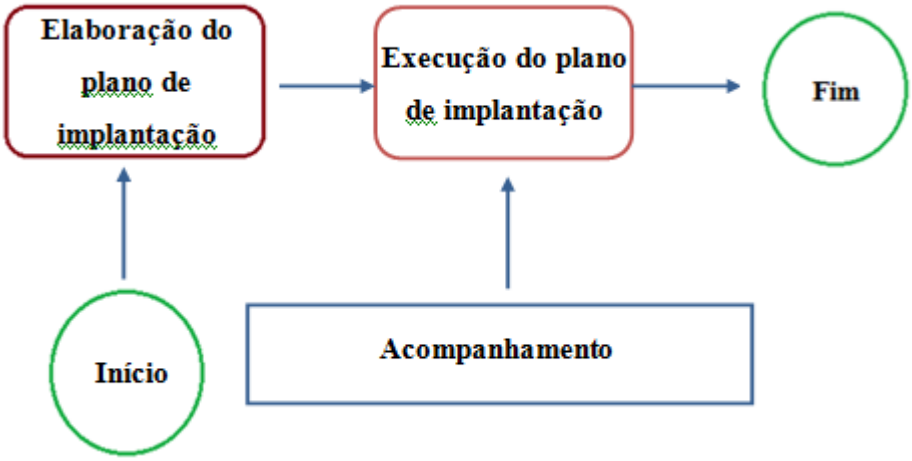

Figura 18: Etapas da Implantação das Práticas SMmm.

O objetivo da elaboração do plano de implantação das práticas é validar os conceitos apresentados, efetuar eventuais adaptações na prática original, realizar as modificações estruturais no ambiente de desenvolvimento, definir formalmente os recursos de hardware e software a serem utilizados pelo sistema e configurar/adaptar as ferramentas facilitadoras do TFS para implementação das práticas do modelo SMmm.

A organização possui a infraestrutura do Team Foundation Server (TFS), que fornece suporte no controle de versão, controle de mudança, integração contínua, testes, além de rastreamento de defeitos e lançamento de tarefas utilizando a estrutura de itens de trabalho. O mapeamento das funcionalidades adequadas do TFS proporcionou a sustentação necessária para produtividade da equipe envolvida e viabilizou a introdução das práticas em um cenário real.

O planejamento das atividades foi baseado no resultado de reuniões e discussões sobre os conceitos que fundamentam as práticas, que também foram fundamentais para compreensão da equipe, de forma a traduzi-las corretamente na realização das atividades. Assim, nessa etapa da implantação, foi gerada a documentação de ações que deveriam ser executadas dentro do propósito de implantar as práticas sugeridas pelo SMmm no sistema em questão.

A execução do plano de implantação das práticas confirmou a possibilidade de utilização das práticas, com suas devidas adaptações, ao contexto do sistema em questão e, a partir de então, foram realizadas reuniões semanais para dar continuidade à evolução das práticas. A fase de acompanhamento da execução do plano resumiu-se em acompanhar a etapa de execução do plano de implantação e verificar se as atividades propostas estavam sendo executadas corretamente. Caso isso não estivesse acontecendo, analisavam-se as causas, os impactos e propunham-se soluções alternativas.

Uma das dificuldades encontradas durante a fase de implantação das práticas do modelo SMmm, diz respeito à dificuldade de controlar as atividades. Para contornar essa situação, decidimos adotar a prática de visualização das atividades proposta pelo Kanban, que nos auxiliou na visualização e controle das atividades, permitindo a identificação de possíveis gargalos.

Outra dificuldade encontrada na fase de implantação das práticas foi referente à indisciplina nas atividades por parte alguns integrantes da equipe. Por exemplo, percebeu-se que mudanças na forma de trabalho, principalmente nas atividades tradicionalmente executadas de forma "ad hoc" foram difíceis de serem aceitas por alguns membros da equipe. A estratégia selecionada para minimizar a resistência da equipe às mudanças na rotina de trabalho foi incentivar a equipe a participar ativamente das propostas de melhoria;

Durante a fase de implantação das práticas, que durou cerca de um mês, vivenciamos também problemas de comunicação entre os membros da equipe.

Durante a fase de implantação de melhorias, o ambiente do sistema piloto sofreu mudanças que favoreceram a iniciativa de melhoria do processo de manutenção de software. Como parte das ações de melhoria do sistema, após a realização de algumas reuniões com membros da equipe do sistema e a coordenação, foi decidida a adoção da Programação extrema (do inglês *eXtreme Programming*), ou simplesmente XP como metodologia de desenvolvimento. Segundo [BECK, ANDRES, 2004], o XP é um método leve, que procura fazer o necessário para trazer valor para o cliente por meio do software funcionando.

O motivo da escolha do XP baseou-se em alguns fatores, por exemplo, por este compor alguns valores e práticas de programação, que a equipe julgou ser

fundamental para o início da iniciativa de melhoria. O conhecimento da equipe sobre XP não era grande, então houve a necessidade de realização da capacitação da equipe. O membro da equipe com mais experiência recebeu treinamento intensivo e após propagou o conhecimento adquirido para o restante da equipe. Durante o período de transição não houve rotação de membros da equipe, isto foi um fator favorável para o período de transição do sistema.

Decidimos adotar prática de revisão de código, ou *code review*, para realizar inspeções sistemáticas no código produzido, esta atividade era facilitada pelo TFS. São normalmente associadas à integração de novo código ao sistema de controle de versão, com objetivo de encontrar defeitos e identificar eventuais melhorias de implementação. A prática estabelece dois papéis: o desenvolvedor e o revisor. A ideia é que o código produzido pelo desenvolvedor seja aprovado pelo revisor após eventuais modificações sugeridas, adotamos também a padronização de código fonte.

Os desenvolvedores da equipe ao implementarem as modificações no código, sempre que necessário, também realizavam refatorações. A refatoração não altera as responsabilidades do código, elas somente organizam a sua estrutura interna, tornando-a manutenível.

Segundo [Swebok, 2004], teste de software consiste na verificação dinâmica do comportamento de um programa, através de um conjunto finito de casos de teste, adequadamente selecionado a partir de um conjunto infinito de possibilidades, contra um comportamento esperado especificado.

Um defeito é um fragmento de artefato de código que, se utilizado de certa maneira, provoca um erro. Um erro é um desvio do que seria esperado para o comportamento desejado. Uma falha é a observação de um erro. A partir de uma falha identificada ao testar, é necessário diagnosticar a sua causa e consequentemente determinar o defeito causador.

Da mesma forma que surgiu a necessidade de automatizar os testes, ficou evidente ao longo do processo de implantação das práticas sugeridas pelo modelo SMmm que seria necessário à absorção da cultura de testes, por parte da equipe de desenvolvedores, para auxiliar a inserção dessa cultura no cotidiano da equipe foram realizadas reuniões e ministrados treinamentos.

A estratégia inicial utilizada para automatização foi estabelecer como um critério de seleção os módulos mais críticos ou que apresentavam defeitos com certa frequência, refatorá-los e introduzir testes automatizados nestes. Para automatizar os testes foi utilizada a ferramenta de gerenciamento e acompanhamento dos testes e da qualidade do software , o *Microsoft Test Manager* (MTM). Com o MTM foi possível o gerenciamento de casos de testes, gerenciamento da qualidade do build, gerenciamento de ambientes de testes (automatizados e manuais) e elaboração de Scripts para automatização de testes.

5.3 Implementação das práticas do modelo SMmm

Esta seção descreve a implementação das práticas derivadas do modelo SMmm, com apoio do TFS.

Para o sistema em questão foi configurado junto ao TFS o template[2] de processo ágil e a ferramenta *Visual Studio Team Foundation* (VSTF) que disponibiliza a funcionalidade *Work Item Types* - WITs (documento de visão) é um formulário de entrada que corresponde ao item de trabalho é possível criar WITs através do portal web do TFS. As demandas de manutenção são cadastradas utilizando dois tipos se WITs como mostra a figura 19, WIT de *Product backlog Item* (PBI) e WIT de Bug para correção de defeitos. A lista de WITs cadastrados constitui o backlog.

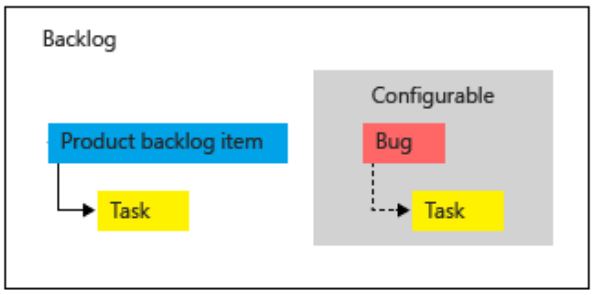

Figura 19: Work items do Backlog do TFS

[2] Um *template* é um modelo a ser seguido, com uma estrutura predefinida que facilita o desenvolvimento e criação do conteúdo a partir de algo construído a priori.

No início de cada iteração, que possui duração de duas semanas, ocorre a reunião de planejamento da iteração. Nesta reunião, com a participação dos membros da equipe e o *Product owner* (PO), são planejadas as demandas que farão parte da iteração.

Nesta reunião são executadas as atividades Identificação e Análise, e suas respectivas tarefas, da fase chamada Planejamento da modificação do processo definido. As atividades desta fase auxiliaram no desenvolvimento da modificação, visto que houve uma discussão prévia, proporcionando um direcionamento para o desenvolvedor ao implementar o item de trabalho.

Os insumos resultantes destas atividades são registrados no item de trabalho. Os de acordo com a análise do PBI ou Bug, são criados WITs de *Task* para implementação dos mesmos.

A seguir será listada as práticas no padrão de nomenclatura de acordo como descrito na referência [April, Abran, 2008] e suas respectivas descrições da implementação.

Evo 3.2.1 O projeto detalhado da solicitação de manutenção elabora as necessidades do usuário, e é desenvolvido de acordo com um procedimento documentado [ISO 12207, 5.3.6; IEEE 1219, 4.3; SMmm].

As atividades identificação e análise da fase de planejamento do processo definido utilizam a funcionalidade *Work Item Types* – WITs (item de trabalho), provendo informações detalhadas para o item de trabalho, todo controle dos WITs é feito pelo TFS.

As atividades identificação e análise de um item de trabalho são efetuadas durante as reuniões de planejamento da iteração e os insumos gerados destas atividades, são registradas no WIT de forma a ter registrado e unificado todas as informações pertinentes ao item de trabalho.

2) **Evo 3.2.2** As atividades de investigação e de design associadas a uma falha são realizadas de acordo com um procedimento documentado [SMmm].

Para o caso de falha é criado um WIT de Bug com o reporte da falha para que seja conduzida correção. São executadas as atividades das três fases do processo definido:

i. Planejamento da modificação

ii. Implementação da modificação
iii. Revisão e validação da manutenção

3)**Evo 3.2.3** As atividades de implementação (de programação) são executadas de acordo com um procedimento documentado [SMmm].

O desenvolvedor utiliza o ambiente de desenvolvimento integrado (IDE) do Visual Studio para implementar um item de trabalho (codificar), utilizando as práticas e princípios do XP. Um membro da equipe ao efetuar *check-in* do código, utilizando o *Team Foundation Version Control* – TFVC, controle de versão nativo do TFS, na qual foram configuradas algumas políticas de realização de *check-in* para reforçar as práticas de desenvolvimento, tais como:

i. Comentários do conjunto de alterações: Exige que os usuários forneçam um comentário que não seja vazio quando eles se fazem o *check-in* de alterações pendentes.
ii. Itens de Trabalho: Requer que um ou mais itens de trabalho estejam associados com o *check-in*.

Com o Team Foundation Build estamos utilizando a configuração da integração contínua para compilar a cada *check-in*, tornando possíveis os seguintes recursos:

i. Execução de testes de unidade associados
ii. Análise estática de código permitindo apontar rapidamente problemas de padrões de nomenclatura de classes, parâmetro entre outros.
iii. Cobertura de código para identificar verificar o percentual de código que estão cobertos pelos testes unitários.

4) **Evo 3.2.6** As inter-relações (rastreabilidade) entre os procedimentos administrativos, os módulos, os processos e os dados são atualizados quando o software é corrigido e/ou evoluído [SMmm].

A partir da configuração feita no *Team Foundation Build*, ao fazer um *check-in*, Team Foundation Build identifica automaticamente e associam as changesets, *work items* e outros artefatos com a *build*. Consequentemente é estabelecida uma rastreabilidade entre *work items*, código fonte, *build* e os resultados de testes. O TFS oferece toda infraestrutura para prover a rastreabilidade.

5) Este item agrupa as seguintes práticas:

Evo 3.2.7 Os testes de unidade devem ser executados para cada componente referente à modificação. [IEEE 1219,4.4.2.1].

Evo3.2.8 Os testes de integração e de regressão devem ser realizados para cada componente referente à modificação [SMmm].

Evo 4.2.4 O teste de aceitação é realizada de acordo com um procedimento documentado para mostrar que o produto modificado está em conformidade e atende as necessidades do cliente [ISO 90003, 5.7.4 e 5.7.5].

Evo 4.2.5 Testes de regressão são sistematicamente e formalmente realizados em todos as partes do software afetadas por modificações, incluindo correções [ISO 14764, 83.2.2; SMmm].

Para suprir as necessidades de melhoria do qualidade do software, utilizamos uma ferramenta da família Visual Studio voltadas para testes, o Microsoft Test Manager (MTM), possibilitando a gestão dos testes de software. Quando é criado um caso de teste, na verdade, está sendo criando um *Work Item* (item de trabalho) no TFS do tipo *Test Case* e quando este é executado, as informações relacionadas a cada execução do teste são salvas na base do TFS. Com isso, é possível compartilhar com todo o time as informações geradas a partir da execução de um caso de teste, e no caso do teste gerar uma falha, é possível gerar um Bug utilizando as informações que tinham sido armazenadas anteriormente no test run. O MTM também executa testes automatizados que foram gerados através do Visual Studio.

Com a configuração do Team Build, o build do sistema é realizado de forma automática, ou seja, a cada build realizado é executada uma bateria de testes associados, sejam eles unitários, funcionais e de regressão.

A organização na qual este estudo de caso foi desenvolvido não possui uma equipe ou setor especializado em testes, a elaboração e execução de testes é responsabilidade do desenvolvedor como parte de sua implementação.

Durante a reunião de planejamento são especificados e cadastrados no formulário do TFS os critérios de aceite, que são utilizados como diretriz para os teste de aceitação.

4) Este item agrupa as seguintes práticas:

Evo 4.2.1 As atividades de verificação e validação são documentadas, conhecidas e planejadas. Os planos são documentados de acordo com um procedimento documentado [SMmm; ISO 90003, 5.7.2; 1 TIL 2007c, 4.5.5].

Evo 4.2.2 Reviews e verificação/validação são realizadas, de forma seletiva, após a produção de alguns produtos chave intermediários [SMmm; ISO 90003, 7.3.1 b].

Algumas iniciativas no aspecto de revisão, verificação e validação foram adotadas objetivando principalmente a qualidade, tais como: programação em par, revisão de código e reunião de revisão. O Visual Studio em conjunto com o TFS possibilitou a atividade de revisão de código, permitindo que o desenvolvedor solicite uma revisão das alterações que ele realizou, buscando um parecer sobre o código, podendo evitar que sejam propagados erros mais básicos. A revisão de código proporcionou maior aprendizado entre os desenvolvedores da equipe, pois houve muita troca de experiências nos questionamentos e sugestões ocasionados por esta atividade.

A função de validação também consistia em acompanhar as etapas e verificar se as atividades propostas estavam sendo executadas corretamente. Caso isso não estivesse acontecendo, analisavam-se as causas, os impactos e propunham-se soluções de contorno.

Após a fase de revisão/validação, são produzidos dois documentos. Um é o documento de manutenção evolutiva descrevendo as novas funcionalidades que foram liberadas em ambiente produtivo.O outro é um documento de publicação, listando os artefatos liberados em de produção (*release note*), estes documentos

são publicados no Yammer (ferramenta corporativa da Microsoft) que auxilia no compartilhamento das informações corporativas).

5.4 Considerações

A adoção de algumas cerimônias de métodos ágeis, tais como, reunião de planejamento, reunião de review e retrospectiva muito contribuiram para auxiliar a implementação das atividades de manutenção.

i. Reunião de planejamento da iteração

   A reunião de planejamento auxiliou na implantação das práticas, pois viabilizava analisar a demanda de manutenção mais detalhadamente, expondo fatores importantes para implementação da modificação.

ii. Reunião de revisão

   Ao fim de cada iteração foram realizadas reuniões de revisão, com objetivo de verificar a conformidade das demandas de manutenção.

iii. Retrospectiva

   Esta cerimônia foi utilizada para identificar o que pode ser melhorado de uma iteração para a outra, além de possibilitar que sejam identificados problemas em geral durante a iteração. Caso houvesse algum problema, durante a iteração, as opiniões a respeito das causas do problema seriam expostas, gerando uma discussão para que nas próximas iterações fosse possível mitigar ou até eliminar suas causas.

iv. Programação em pares

   Esta prática ajudou no compartilhamento de conhecimentos entre os membros da equipe estimulando discussões sobre os conceitos adotados e melhor forma de aplicação e também colaborou com a

revisão contínua do artefato de software na qual o parceiro exerce também a função de revisor do código, em tempo de implementação.

5.5 Elaboração do plano de medição

Para obter dados que possam mensurar características do software, a engenharia de software costuma utilizar métricas, tanto na fase de desenvolvimento quanto na fase de manutenção.

Tanto na fase de desenvolvimento quanto na fase de manutenção, a engenharia de software analisa características do software para avaliar sua qualidade. Para se mensurar quantitativamente os atributos de qualidade de um software são utilizadas métricas de software. Uma métrica é um método para determinar se um sistema, componente ou processo possui certo atributo [ISO/IEC, 2006]. Uma vez que o software é um produto intangível, as métricas de software proporcionam uma forma de mensurar algumas características de qualidade do software. Os resultados podem ser, por exemplo, usados para melhorias na qualidade do código [PRESSMAN, 2011].

Com base na informação de contexto do estudo prévio, foi utilizada a abordagem Goal Question Metric (GQM) [BASILI et al., 1996] para definição das métricas. A abordagem GQM é um mecanismo usado para definir e avaliar um conjunto de objetivos operacionais usando métricas. Essa abordagem representa uma sistemática para ajuste e integração de objetivos com modelos de processos, produtos e perspectivas de qualidade de software, baseadas em necessidades específicas do projeto e da organização.

Esta é uma abordagem orientada a metas de mensuração, ou seja, um mecanismo para definição e interpretação de métricas de software. Inicialmente foi definida uma meta (GOAL), especificando o propósito da medição, objeto a ser medido, foco e o ponto de vista ao qual a medição está sendo obtida, como demonstra a tabela 6.

Dimensão	***Definição***
Objeto de estudo	***O que será analisado***
Processo de manutenção de software	A aplicação boas de práticas de manutenção de software.
Objetivo	***Porque o objeto será analisado***
Redução do número de defeitos nos artefatos de software entregues oriundos manutenção	Pela necessidade de se entender e implementar melhorias no processo de manutenção de software.
Enfoque de qualidade	***Qual atributo do objeto será analisado***
Propriedade do objeto que será analisado	Correções, modificações, quantidade de caso de testes, número de erros recorrentes, tempo e manutenibilidade.
Ponto de vista	***Quem deverá usar os dados coletados***
Quem vai utilizar os dados coletados	A coordenação e equipe integrante do sistema
No contexto do	***Em qual ambiente está localizado***
Sistema utilizado no estudo de caso	Organização utilizada no estudo de caso

Tabela 6: Planejamento GQM

O objeto de estudo foi escolhido pela necessidade de reduzir a ocorrências de defeitos no sistema, planejar, adaptar e aperfeiçoar as atividades de manutenção. A avaliação visa comparar e aferir os ganhos após a implementação das praticas baseadas no modelo SMmm e o enfoque de qualidade é consistente com a prioridade de metas de melhoramento organizacional.

Ainda de acordo [BASILI et al., 1996] para dividir o objetivo em seus principais componentes, divide-se a meta em várias perguntas. Cada pergunta é derivada em métricas objetivas. Uma mesma métrica pode ser utilizada para responder perguntas diferentes de uma mesma meta. Metas diferentes podem possuir perguntas e métricas em comum.

Para delinear o planejamento da abordagem GQM, foram realizadas reuniões de equipe e o resultado das discussões das reuniões foi usado como uma

base para a derivação das perguntas, modelos e medidas relevantes à meta, como mostra a figura 20.

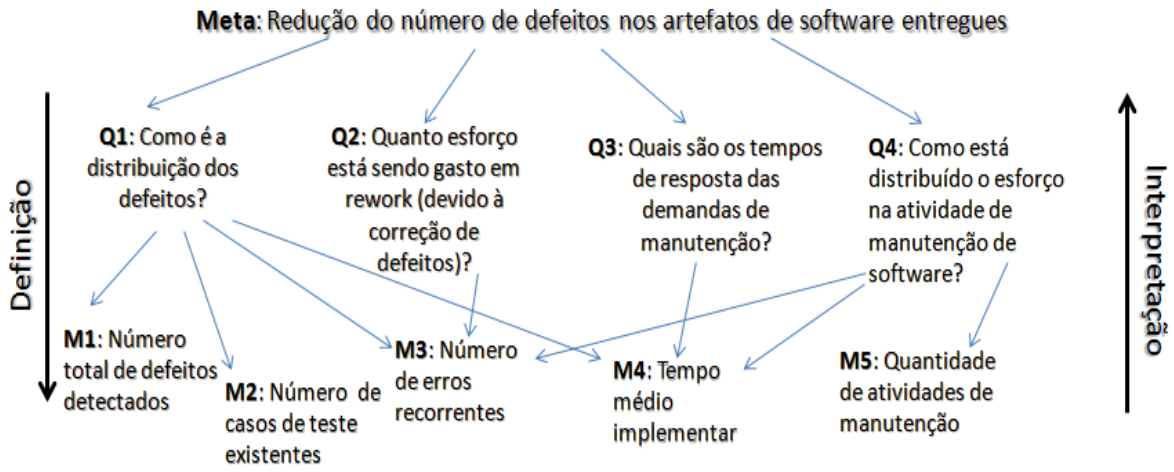

Figura 20: Abordagem Goal Question Metrics (GQM)

O processo de coleta realizado durante o acompanhamento do estudo de caso foi direcionada pelas atividades de coletar e registrar as métricas, em forma de notas, ao fim de cada iteração e consolidar estes dados na forma de um conjunto de informações gerenciáveis (planilha eletrônica) e a partir daí era determinado quais os indícios válidos encontrados, garantindo a cobertura de todo escopo determinado pelo plano de medição.

[BECK, ANDRES, 2004] descreve o papel do tracker em uma equipe de XP como alguém responsável por coletar frequentemente métricas a partir de dados obtidos. Por meio da colaboração do tracker da equipe, as métricas planejadas foram coletadas com suporte oferecido pela infraestrutura do TFS. O tracker atualizava as informações a respeito das métricas produzindo gráficos e tabelas ilustrando a evolução dos dados durante as iterações. É importante ressaltar que ao final de cada iteração (período de duas semanas) foram discutidos os pontos que necessitavam de melhorias e o processo era adaptado progressivamente.

Para que as métricas fossem coletadas de forma eficaz, foi necessária a colaboração de todos os membros da equipe, fornecendo as informações pertinentes. As informações coletadas ofereceram fundamentos estatísticos para

realização de uma avaliação objetiva sobre o impacto da adoção de práticas de manutenção sugeridas pelo modelo SMmm.

5.6 Análise dos resultados

Uma das últimas etapas da pesquisa em estudo de caso é a avaliação, que segundo [BORGES, HOPPEN e LUCE, 2009] consiste em "examinar, categorizar, tabular e recombinar os elementos de prova, mantendo o modelo conceitual e as proposições iniciais do estudo como referências".

Dessa forma, a estratégia de avaliação utilizada neste estudo de caso foi comparar as métricas colhidas em períodos antes e após a implantação do conjunto de práticas.

Os seguintes atributos foram analisados:

i. Quantidade de defeitos
ii. Quantidade erros recorrentes
iii. Quantidade de testes automatizados
iv. Tempo de implementação das demandas de manutenção
v. Classificação dos tipos de manutenção

No processo de avaliação os dados analisados no estudo preliminar, descritos na seção 5.2.1.2, foram confrontados com dados colhidos ao longo de doze iterações (cinco meses) período de fevereiro a julho de 2016 após implantação das práticas do modelo SMmm.

Os dados coletados foram tratados estatisticamente, consolidados em gráficos de frequências, percentuais, gráficos de setores e de colunas. A análise estatística foi apoiada pela disponibilização destes gráficos, que permitiu uma melhor visualização de pontos relevantes para pesquisa. A seguir será apresentado o resultado da avaliação deste estudo de caso.

Ao sintetizar os relatórios gerados com informações das iterações apresentamos, na tabela 7, os dados referentes às quantidades: de defeitos, testes automatizados, tipos de manutenção.

				Classificação quanto ao tipo de manutenção			
Iterações	Qtde defeitos	Qtde de casos de teste criados	Qtde de casos de testes existentes	Corretiva	Perfectiva	Adaptativa	Preventiva
Iteração 1	2	20	397	2	4	0	2
Iteração 2	1	30	417	1	4	1	1
Iteração 3	2	51	447	2	8	0	0
Iteração 4	0	42	498	0	7	0	2
Iteração 5	1	10	540	1	8	2	3
Iteração 6	1	27	550	1	5	0	1
Iteração 7	0	84	577	0	8	1	4
Iteração 8	0	10	661	0	12	0	0
Iteração 9	1	25	671	1	6	2	2
Iteração 10	0	48	696	0	9	0	0
Iteração 11	0	15	744	0	7	1	2
Iteração 12	0	22	759	0	6	1	3

Tabela 7: Síntese de coleta de dados

Os dados apresentados acima serviram como base para criar as representações gráficas a seguir. Uma das métricas mais importantes diz respeito ao número de defeitos reportados, pois estes fornecem informações sobre a qualidade das entregas.

A figura 21 apresenta o gráfico de acompanhamento da quantidade de defeitos injetados pelas modificações realizadas, registrados durante as iterações.

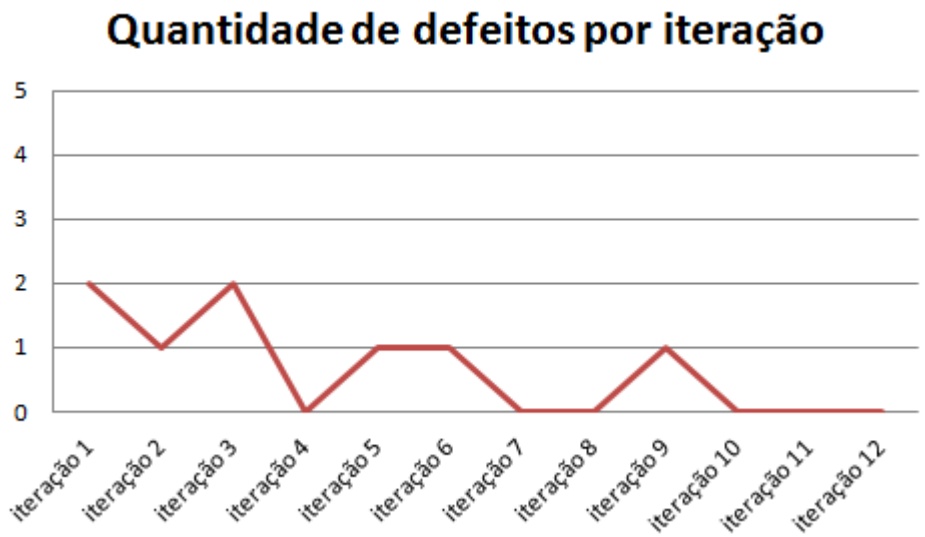

Figura 21: Distribuição dos defeitos por iteração

Como podemos observar na figura 21, as iterações 1 e 3 concentraram o maior número de defeitos, dois por cada iteração, correspondendo respectivamente a 25% do total de demandas na iteração 1 e 20% na iteração 2.

Na figura 22 é ilustrada a visão da distribuição dos defeitos injetados pelas modificações realizadas por mês, o mês fevereiro que correspondeu as duas primeiras iterações e concentrou a maior quantidade de defeitos, equivalendo a 20% do total de demandas do mês.

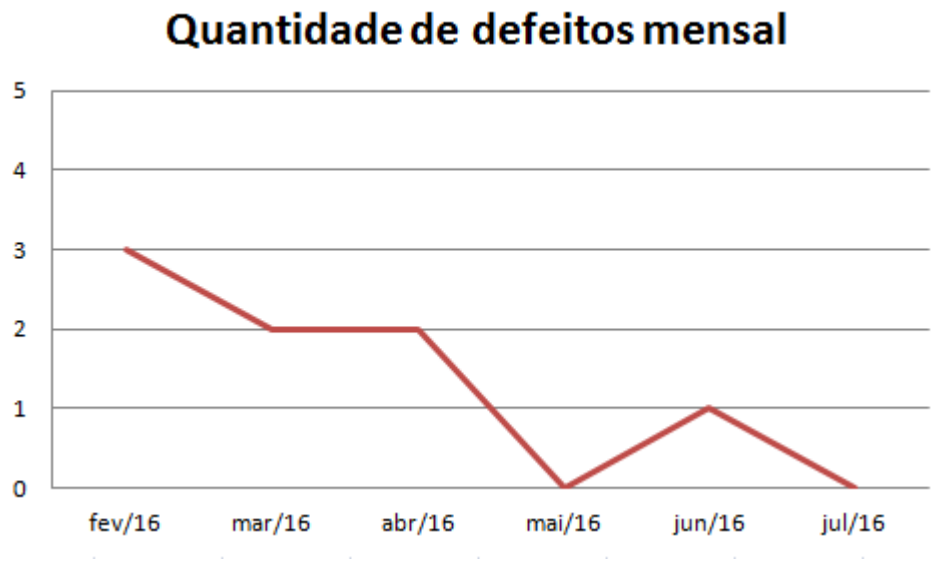

Figura 22: Distribuição mensal dos defeitos

Comparando a quantidade de defeitos reportados no período entre fevereiro e julho de 2015 com o mesmo período em 2016, podemos observar como mostra a figura 23 que houve redução na quantidade de defeitos. Isto ratifica a análise entre os dados apresentados nas figuras 16 e 23, comparado o quantitativo de defeitos em 2015 com o de 2016, revelando uma redução significativa no número de defeitos cadastrados.

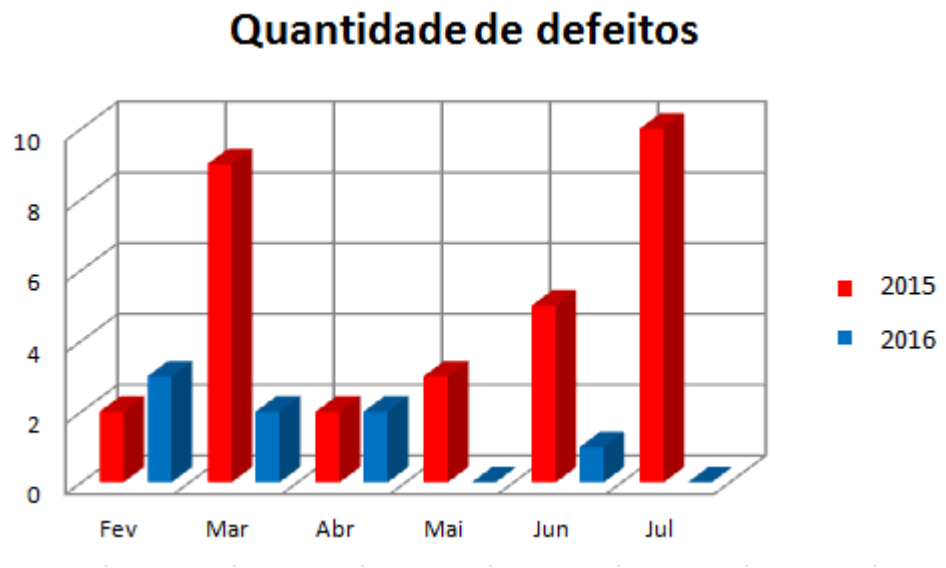

Figura 23: Comparativa quantidade de defeitos

Dentre os defeitos analisados no mês de fevereiro, identificamos a ocorrência de um caso de recorrência de defeito como demonstrado na tabela 8.

Defeito recorrente	**Estudo preliminar**	**Meta** (mensal)	**Meses 2016**					
			Fev	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**
Quantidade	4	0	1	-	-	-	-	-

Tabela 8: Defeito recorrente

Outro fator importante são os testes. Estes são um item decisivo para garantir a qualidade de um sistema. Apresentamos na figura 24 a representação gráfica da quantidade de testes automatizados criados por iteração. Podemos observar que a evolução ao longo das iterações teve crescimento acentuado na oitava iteração.

Figura 24: Evolução dos testes automatizados por iteração

Na figura 25, apresentamos um comparativo entre a evolução dos testes entre o período que antecedeu a adoção das práticas (2015) e o período pós (2016). Isto confirma uma maior aderência dos membros da equipe na adoção da cultura de testes.

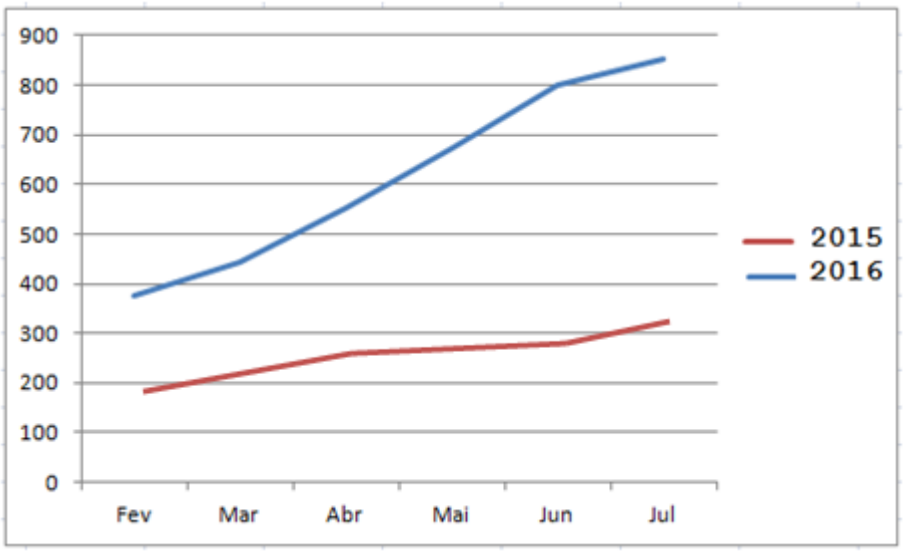

Figura 25: Evolução da quantidade testes automatizados

Nas Figuras 26 e 27 são apresentados respectivamente os gráficos da distribuição das atividades de manutenção e a representação em termos de percentual.

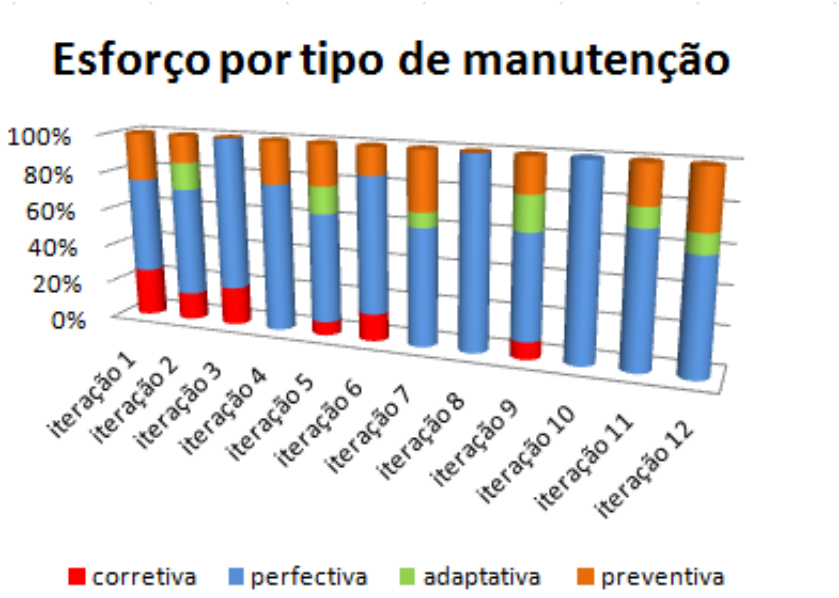

Figura 26: Distribuição do esforço por tipo de manutenção

Ao analisar a classificação das atividades por tipo de manutenção entre o período que antecedeu a adoção das práticas (figura 13) e o período depois da implementação das práticas, demonstrado na figura 26 percebe-se que as demandas do tipo perfectivas continuam expressivas. É interessante destacar a preocupação proativa referente a ações preventivas em torno de 17% do total de itens de trabalho no período analisado como mostra a figura 27. Acreditamos que há um maior comprometimento, visando aumentar a manutenibilidade e/ou confiabilidade do sistema.

Figura 27: Percentual dos tipos de manutenção

Na figura 28, podemos observar que em relação ao tempo utilizado para implementar as demandas, a partir do momento em que foi iniciado até o fim, a maior frequência está concentrada entre a faixa de 6 a 10 dias e a faixa de 11 a 15 dias. Se comparado ao período anterior à adoção das práticas demonstrado na figura 14, na qual a maior incidência estava na faixa de 6 a 10 dias e a segunda maior na faixa de 1 a 5 dias, consideramos uma elevação no tempo de implementação.

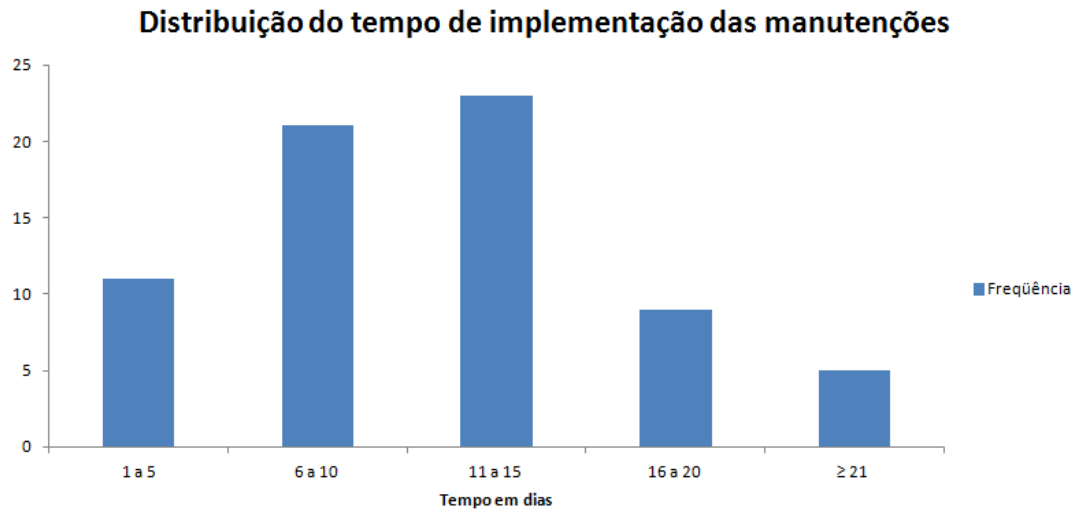

Figura 28: Tempo de implementação das demandas de manutenção

Cabe mencionar como limitações dessa avaliação:

- A avaliação foi feita apenas pelo pesquisador podendo gerar erros de interpretação;
- O período avaliado pós-implantação das práticas foi de 06 meses.
- O estudo avaliado utilizou apenas um sistema piloto na organização.

5.7 Conclusões sobre o resultado

A execução do plano de melhoria descrito na (Seção 5.2.2) deste capítulo, mais especificamente a adoção das práticas sugeridas pelo modelo SMmm apoiadas pelo TFS, promoveu um conjunto de benefícios. Em relação aos principais problemas inicialmente identificados (Seção 1.3) do capítulo 1, as seguintes melhorias puderam ser observadas:

i. Redução da quantidade de defeitos;
ii. Planejamento e organização para atividades de manutenção;
iii. Aumento das atividades que garante qualidade ao software;
iv. Maior visibilidade e controle das atividades de manutenção;
v. Maior manutenibilidade do sistema;
vi. Maior confiabilidade nos prazos estabelecidos.

Podemos considerar que, em meio a algumas limitações, a solução implementada neste estudo aponta uma redução significativa na quantidade de defeitos.

5.8 Lições aprendidas

A realização do estudo apresentado neste capítulo proporcionou algumas lições, descritas a seguir.

i. O processo de adoção das práticas, independente dos resultados observados, ocasionou um grande aprendizado para os envolvidos.

ii. O compartilhamento das lições aprendidas faz parte do plano de melhoria contínua do processo definido, evitando que erros comuns se repitam, um exercício de inspeção e adaptação.

iii. A conscientização para adoção de novas rotinas de trabalho é diretamente proporcional ao grau de participação dos membros da equipe no planejamento e elaboração dos novos processos de trabalho.

iv. A implantação das práticas mencionadas causou uma mudança considerável para equipe envolvida na forma de trabalhar, a introdução das mesmas tem que ser planejada e realizada de forma que todos os envolvidos se sentissem confortáveis com a mudança.

5.9 Dificuldades encontradas

- Inicialmente, houve certa indisciplina no desenvolvimento de algumas atividades por parte de dois membros da equipe. Esta dificuldade foi tratada e solucionada pelo coordenador da equipe.
- Período de adaptação da equipe aos novos conceitos e atividades apresentados. Para reduzir este período, foram realizadas palestras para compartilhar o conhecimento das práticas utilizadas.

6 Conclusão

Este trabalho propôs a adoção de algumas práticas do modelo SMmm como uma alternativa para resolver alguns problemas que permeavam a realidade da organização em questão e objetivou avaliar os benefícios proporcionados. A plataforma TFS mostrou uma sinergia com as práticas adotadas oferecendo suporte na execução das atividades.

A escolha das práticas considerou as necessidades do ambiente ao qual foi aplicado. Sendo necessária realização de uma análise prévia do ambiente para identificar os pontos de carência, onde as práticas do SMmm agregavam maior valor e as ferramentas que viabilizavam a implementação segundo as exigências do modelo.

A adesão e o comprometimento da equipe foi um fator importante para correta execução das atividades, a adaptação a nova forma de trabalho proposto nesta pesquisa, foi um grande desafio, exigiu muita disciplina, mas os objetivos principais foram alcançados, sendo perceptível a equipe.

Os resultados obtidos corroboram que a implementação, usando TFS, de práticas derivadas do SMmm contribui para uma melhora perceptível da redução de defeitos injetados pela manutenção.

Destacamos como restrições deste trabalho:

i. O estudo foi efetuado em apenas um sistema da organização, sem análise de validade externa e nem generalização;
ii. A avaliação foi feita apenas pelo pesquisador podendo gerar erros de interpretação;
iii. A avaliação pós-implantação das práticas levou em consideração a coleta de métricas de fevereiro até julho de 2016.

6.1 Contribuições

Este trabalho pode ser utilizado como ponto de partida para o estabelecimento de novos direcionamentos, processos e práticas de manutenção de software no contexto de melhoria de processo de software.

O modelo SMmm não fornece informações de como implementar, mas orienta qual o propósito das práticas e quais os resultados esperados. Com isto, o diferencial desta proposta de solução é a adaptação do modelo às necessidades e ambiente do sistema utilizado neste estudo.

6.2 Trabalhos Futuros

Devido à limitação do escopo do estudo de caso, este trabalho foi focado em algumas práticas sugeridas pelo modelo SMmm. Um caminho natural para continuidade dos trabalhos seria o aproveitamento de outras práticas a situações reais, com estudos e análises direcionadas.

Alguns trabalhos futuros podem ser encaminhados:

- Evoluir o plano de métricas para abranger outros fatores e assim permitir avaliações do impacto da adoção das práticas sobre estes fatores.
- Refinar o processo após avaliações e comparações realizadas durante as medições de sua aplicação.
- Realizar estudos com outros sistemas e em outras organizações para propor melhorias em um contexto mais amplo.
- Avaliar a possibilidade de adoção das práticas propostas com outros ferramentais de apoio.

7 Referências bibliográficas

ALAIN A., ALAIN A. **Software Maintenance Management: Evaluation and Continuous Improvement**, Wiley-IEEE Computer Society Press, 2008.

APRIL A., HAYES J., ABRAN A. AND DUMKE R. - **Software Maintenance Maturity Model (SMmm): the software maintenance process model**. Journal of Software Maintenance and Evolution: Research and Practice, 2005; 17:197–223

APRIL, A., ABRAN, A., DUMKE, R.: **Software Maintenance Productivity Measurement:. How to Assess the Readiness of Your Organisation.** International Conference on Software Measurement (lWSM/MetriKon), Germany, 2004c.

APRIL, A. et al. **Software maintenance maturity model (SMMM): the software maintenance process model**. Journal of Software Maintenance and Evolution: Research and Practice, New York, v. 17, n. 3, p. 197–223, 2005.

APRIL, A; ABRAN, A. **A software maintenance maturity model (SMmm): Measurement practices at maturity levels 3 and 4**. Electronic Notes Theoretical Computer Science, Holanda, v. 233, p. 73–87, 2009.

APRIL, A., J. BOUMAN, A. ABRAN, D. AL-SHUROUGI. **Software Maintenance in a Service-Level Agreement: Controlling Customer Expectations**, FESMA, Heidleberg, Germany, 2001.

BARTELLI R., FRESSATTI, W. **Uma abordagem comparativa entre as linguagens de programação Java e C#.** Universidade Paranaense - Paranavaí – PR, 2014.

BASILI, V.; CALDEIRA, G.; ROMBACH, D. **The goal question metric approach.** Encyclopedia of Software Engineering, pages 528–532, 1996.

BORGES, M. HOPPEN, N.; LUCE, F. B. **Information technology impact on market orientation in e-business**. Journal of Business Research, v. 62, p. 883-890, 2009.

CMMI para Desenvolvimento – Versão 1.2 **Melhoria de processos visando melhores Produtos**, Pittsburgh – 2006.

FILHO, P.; PÁDUA, W. Engenharia de software: fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: LTC, 2003.

HALL, T. et al. **An empirical study of maintenance issues within process improvement programmes in the software industry**. In: 17th IEEE International Conference on Software Maintenance (ICSM). Florença, Itália: IEEE Computer Society, 2001. p.422–428.

IEEE Standards Collection: **Software Engineering**, IEEE Standard 610.12-1990, IEEE, 1993.

International Standard ISO/IEC/IEEE 14764 Software **Engineering - Software Life Cycle Processes - Maintenance**. Piscataway, EUA, 2006.

ISACA. **COBIT Framework for IT Governance and Control**. 2010. Disponível em: http://www.isaca.org/Knowledge-Center/COBIT/Pages/Overview.aspx. Acesso em: 07 ago. 2016.

ISO/IEC. International Standard ISO/IEC 12207 **Software Life Cycle Processes**. Genebra - Suíca, 2008.

IEEE standard for software maintenance, IEEE std 1219-1998. In: **IEEE Standards Software Engineering**, Volume Two: Process Standards. New York: IEEE Press, 1999. v. 2.

KAJKO-MATTSSON, M. **Problem management maturity within corrective maintenance.** Journal of Software Maintenance and Evolution: Research and Practice 2002; 14(3):197–227.

KAJKO-MATTSSON, M., FORSSANDER, S., OLSSON, U. **Corrective maintenance maturity model (CM3): Maintainer's education and training.** The 23rd International Conference on Software Engineering (ICSE 2001). IEEE Computer Society: Toronto, Canada, 2001; 610–619.

BECK, K., ANDRES, C. **Extreme Programming Explained : Embrace Change**. Addison-Wesley Professional, 2nd edition, 2004.

LEHMAN, M. **"On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle"**. Journal of Systems and Software. 1: 213–221. (1980)

MAFRA, S., TRAVASSOS, G. **Estudos primários e secundários apoiando a busca por evidência em engenharia de software.** In: Relatório Técnico RT – ES 687/06. Rio de Janeiro: Programa de Engenharia de Sistemas e Computação, 2006.

OGC. **Official ITIL Website**. 2011. Disponível em: http://www.itil-officialsite.

com/home/home.asp. Acesso em: 07 Ago. 2016.

PAQUETTE, D., A. APRIL, AND A. ABRAN, **Assessment Results using the Software Maintenance Maturity Model (SMmm).** 16th International Workshop on Software Measurement - (IWSM-Metrikom 2006), 2006.

PESSOA, M. **Introdução ao CMM – Modelo de Maturidade da Capacidade de Processo de Software** – Lavras: UFLA/FAEPE, 2003.

PFLEEGER, L. Engenharia de software: teoria e prática. [Traduzido do original: Software engineering - theory an practice]. Tradução de Dino Franklin. 2. ed. São Paulo: Prentice Hall, 2004.

POOLE, C., HUISMAN, J. Using extreme programming in a maintenance environment. IEEE Software 2001; 18(6):42–50.

PRESSMAN, R. **Engenharia de Software - Uma Abordagem Profissional** - 7ª Ed. - Amgh Editora, 2011.

SEI. **The Capability Maturity Model: Guidelines for Improving the Software Process**. Software Engineering Institute, Carnegie Mellon University, EUA: Addison-Wesley Publishing Company, 1995. 441 p. (SEI Series in Software Engineering).

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro**, Guia Geral. 2009. Disponível em: http://www.softex.br/mpsbr. Acesso em: 05 ago. 2016.

SOFTEX. MPS.BR – **Guia Geral MPS de Software**:2012. Disponível em: http://www.softex.br. Acesso em: 06 ago. 2016.

SOMMERVILLE, I. **Engenharia de Software.** 9ª. Edição. São Paulo: Pearson, 2011.

SOUZA LIMA, C. **Proposta de gestão de manutenção de software utilizando a abordagem de serviços do modelo ITIL.** São Paulo, 2009. Dissertação de Mestrado - IPT - Instituto de Pesquisas do Estado de São Paulo.

SVENSSON, H., HOST, M. **Introducing an agile process in a software maintenance and evolution organization.** The Ninth European Conference on Software Maintenance and Reengineering (CSMR'05). IEEE Computer Society: Silver Spring MD, 2005; 256–264.

SWEBOK - **Guide to the Software Engineering Body of Knowledge IEEE** – 2004 Version - Versão no site http://computer.org

YIN, R. K. **Case study research: design and methods**. London: Sage, 1984.

YONGCHANG, R., TAO, X., ZHONGJING, L. XIAOJI, C. **Software Maintenance Process Model and Contrastive Analysis**, 2011.