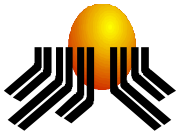

UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

AVALIAÇÃO DA APLICABILIDADE E EFICÁCIA DE UM MODELO DE MATURIDADE DE TESTE

Ariane Louise Corso

Orientador: Prof. Dr. Plínio R. S. Vilela

PIRACICABA, SP

2008

UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

AVALIAÇÃO DA APLICABILIDADE E EFICÁCIA DE UM MODELO DE MATURIDADE DE TESTE

Ariane Louise Corso

Orientador: Prof. Dr. Plínio R. S. Vilela

Dissertação apresentada ao Mestrado em Ciência da Computação, da Faculdade de Ciências Exatas e da Natureza, da Universidade Metodista de Piracicaba – UNIMEP, como requisito para obtenção do Título de Mestre em Ciência da Computação.

PIRACICABA, SP

2008

AVALIAÇÃO DA APLICABILIDADE E EFICÁCIA DE UM MODELO DE MATURIDADE DE TESTE

Autor: Ariane Louise Corso
Orientador: Prof. Dr. Plínio R. S. Vilela

Dissertação de Mestrado defendida e aprovada em 14 de Fevereiro de 2008, pela Banca Examinadora constituída dos Professores:

Prof. Dr. Plínio Roberto Souza Vilela - UNIMEP (Orientador)
Prof. Dr. Luiz Eduardo Galvão Martins – UNIMEP
Prof. Dr. Adalberto N. Crespo - CenPRA

DEDICATÓRIA

Ao

Meu marido Richard pela paciência e compreensão

Aos

Meus pais Carlos e Lourdes pelo apoio e dedicação

AGRADECIMENTOS

Ao professor Plinio Vilela a orientação, compreensão e Incentivo dispensado ao desenvolvimento deste trabalho.

Ao meu pai e professor Carlos Renato Corso pelas constantes revisões e leituras deste trabalho.

AVALIAÇÃO DA APLICABILIDADE E EFICÁCIA DE UM MODELO DE MATURIDADE DE TESTE

RESUMO

Estabelecer um processo de teste em uma organização é uma tarefa difícil, mas que ajuda a garantir a qualidade do software desenvolvido. Este trabalho busca pesquisar as vantagens e obstáculos da utilização de um Modelo de Maturidade de Teste em uma organização, além de avaliar sua eficácia através de medições realizadas durante o desenvolvimento do software e no período pós-implantação. O interesse está em conseguir abstrair as melhores práticas dessa experiência e garantir que alguns erros cometidos possam ser evitados futuramente.

PALAVRAS-CHAVE: Estudo empírico, TMM, Modelo de Maturidade de Teste, Engenharia de Software, Teste de Software.

APPLICABILITY AND EFFECTIVENESS ASSESSMENT OF A TEST MATURITY MODEL

ABSTRACT

Establishing a test process in a company is a difficult task. Nevertheless it helps assure software quality. This research work explores the advantages and drawbacks of using a test maturity model in a company and evaluates its effectiveness through measurements performed during the software development and after its deployment. We intend to select the best practices with our experience to avoid repeating the same mistakes in the future.

KEYWORDS: Empirical Studies, TMM, Test Maturity Model, Software Engineering, Software Testing.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integrated
SEI	Software Engineering Institute
TMM	Test Maturity Model
TPI	Test Process Improvement
LOC	Linhas de Código
KLOC	Milhares linhas de Código
FP's	Pontos-por-função

LISTA DE TABELAS

1. Introdução

Nos dias de hoje existe uma grande preocupação relacionada à qualidade de software. O processo de desenvolvimento tem se tornado alvo de muitas pesquisas que buscam a melhoria contínua em cada etapa da construção até a entrega do produto final. Isso ajuda a garantir que o software produzido seja entregue com a qualidade esperada, dentro de prazos e custos previstos.

A melhoria da qualidade do processo, por si só, não é suficiente para garantir a qualidade do produto final, mas um processo de má qualidade, certamente produzirá um software de má qualidade.

Segundo Crespo et al.(2004), na medida em que o emprego de sistemas de informação pela sociedade cresce ao ponto em que boa parte dos negócios depende cada vez mais de software e computadores, passa a ser de vital importância contar com software de qualidade, onde um dos fatores chave, é contar com sistemas que fornecem resultados corretos quando alimentados com dados válidos e que identificam corretamente dados de entrada inválidos. A área de teste de software é fundamental para avaliação da qualidade do software desenvolvido.

De acordo com Ntafos (1999), a confiabilidade do software é a probabilidade de um programa operar corretamente por um tempo específico em um ambiente específico. O custo total dos defeitos que permanecem no software liberado para produção e que, eventualmente se manifestam como falhas, tem sido estudado, por vários autores, a fim de se fazer uma estimativa para alocar melhor o esforço ligado aos testes e também reduzir o custo esperado das falhas.

Para melhorar o processo de desenvolvimento de software, vários modelos de maturidade foram idealizados e hoje servem como diretriz para muitas organizações. Entre eles, o CMMI (Capability Maturity Model Integrated – Modelo Integrado de Maturidade e Capacidade) é um modelo largamente usado como padrão para melhoria de processos de software em indústrias. O CMMI é modelo criado pela SEI (Software Engineering Institute) para ser um guia destinado a melhorar os processos organizacionais e a

habilidade desses em gerenciar o desenvolvimento, a aquisição e a manutenção de produtos e serviços (SOUZA, 2005). Embora os modelos de maturidade de software existentes tenham grande aceitação e aplicação pela indústria de software, eles não abordam adequadamente as questões relacionadas aos testes e nem possuem um processo de teste bem definido. Com isso, pesquisadores iniciaram um processo de desenvolvimento de modelos de maturidade de teste que completassem as deficiências na área de teste.

Assim como citado por Staab (2002), alguns dos modelos de maturidade de teste desenvolvidos são: *Software Testing Maturity Model* (SW-TMM ou TMM), *Test Process Improvement* (TPI), *Test Organization Maturity*, *Test Assessment Program*, que buscam deixar o processo de teste mais robusto para que os software se tornem mais confiáveis quando liberados para produção. Embora existam estes vários modelos de processo de teste disponíveis, os modelos de maturidade de teste que mais se adaptam às organizações, segundo Staab (2002), são o TMM e TPI. O modelo TPI foi desenvolvido baseado no conhecimento prático e em experiências do desenvolvimento do processo de teste (ANDERSIN, 2004). O modelo TMM busca endereçar as deficiências existentes tanto na adequação das questões que se relacionam ao processo de testes nas organizações quanto nas definições de um processo maduro de teste (BURNSTEIN; SUWANNASART;CARLSON, 1996). Ambos os modelos ajudam a definir passos para melhorias do processo de teste, de forma controlável e gradual.

Atualmente, há a necessidade das organizações em agilizar o processo de teste e garantir a qualidade do software entregue, aumentando a quantidade de defeitos encontrados durante o processo de desenvolvimento e procurando identificar primeiro aqueles que poderiam ter um custo mais elevado dentro do processo de desenvolvimento, ou impactar de forma mais significativa no negocio do cliente. Além disso, o interesse em reduzir o custo com manutenções corretivas contribui com a motivação para os investimentos em um processo de teste.

Projetos de software realizados sem a aplicação de modelos de processos de teste tendem a apresentar maior índice de falhas quando o

software já está em produção, podendo gerar altos custos para que os ajustes necessários sejam feitos.

Esta pesquisa tem como proposta avaliar se um projeto que segue um processo de teste avaliado em níveis mais altos do modelo TMM apresenta menos falhas em campo, encontrando-as previamente, se comparado a projetos de semelhante característica, cuja maturidade do processo de teste se encontra no nível mais baixo. A intenção deste trabalho é verificar empiricamente se a aplicação de um modelo de maturidade de teste é capaz de reduzir o custo com manutenções corretivas nas organizações. As pesquisas para melhoria dos processos de teste de software são instrumentos interessantes que podem alavancar mudanças significativas na busca pela excelência no desenvolvimento de software.

No próximo capítulo, a revisão da literatura apresenta o embasamento inicial sobre teste de software e alguns conceitos-chave para o desenvolvimento da idéia deste trabalho. São apresentados também, os modelos de maturidade de teste, e as contribuições de outros autores sobre a utilização de modelos como o TMM. O capítulo 3 descreve mais detalhes sobre o foco, hipóteses desta pesquisa e como ela foi desenvolvida. O capítulo 4 apresenta os resultados e discussões da pesquisa que são concluídos no capítulo 5.

2. Revisão da Literatura

As diretrizes de qualidade adotadas por organizações devem estar sempre alinhadas aos processos de testes. Embora a qualidade do software vá além dos testes realizados, ele é um processo importante que ajuda a garantir que o produto final atenderá aos requisitos existentes durante a sua concepção.

Entender o que é teste de software, como elaborá-lo, planejar sua execução, adotar modelos de maturidade de teste, não é uma tarefa trivial. Para que todo seu gerenciamento seja feito da melhor maneira possível, é necessário conhecer de forma mais detalhada esta disciplina na engenharia de software.

Os próximos pontos desta seção apresentam os conceitos e as terminologias utilizadas no trabalho, além de tendências na área sobre as questões apontadas acima, através da citação de algumas publicações realizadas por diversos autores.

2.1. Teste de Software

Teste de software é o processo de executar o software de uma maneira controlada com o objetivo de avaliar se ele se comporta conforme o especificado (CRESPO et al., 2004). O foco principal do teste é revelar a presença de defeitos no produto, para que estes possam ser corrigidos pela equipe de programadores, antes da entrega final.

Geralmente, os engenheiros de software distinguem ‘defeitos’ de ‘falhas’ e, quando se aborda o assunto ‘teste de software’, alguns conceitos devem estar claros para que não ocorra o seu uso incorreto. Neste trabalho usamos as seguintes definições:

- Defeito – Algo que está incorreto na programação ou software que pode ou não se manifestar em uma falha. O defeito decorre de um engano cometido por um desenvolvedor (analista, projetista, programador), como, por exemplo, um programa contendo um “<“ no lugar de um “ > “.
- Erro– É um estado inconsistente na execução do programa.
- Falha– É a violação do serviço que o software fornece, ou seja, o software não realiza as funções de acordo com o que o usuário espera. Exemplo: quando um programa fornece resultados errados aos usuários ou a outros sistemas, ou mesmo quando um programa aborta ou entra em laço infinito.

O teste do software é um dos sub-processos da engenharia de software que visa atingir um nível de qualidade de produto superior. De acordo com Pressman (2001), a atividade de teste é um elemento crítico da garantia da qualidade de software e representa a última revisão de especificação, projeto e codificação.

Há várias formas de fazer testes de software, mas testar efetivamente produtos completos é um processo de investigação, não é somente criar e seguir procedimentos. Apesar de a maioria dos processos de testes serem quase idênticos à revisão ou inspeção, a palavra testar está ligada a análise dinâmica do produto, verificação do comportamento do produto perante algumas situações.

Segundo Crespo et al. (2004), na fase de planejamento de teste, uma das etapas é a elaboração da estratégia de teste que contém a definição do nível de teste que será aplicado, da técnica de teste que será utilizada, do critério de teste a ser adotado e do tipo de teste que será aplicado ao software.

O nível de teste depende da fase de desenvolvimento do software em que o teste poderá ser aplicado, sendo composto por:

- Teste de unidade: a codificação dos módulos do sistema. Este teste busca identificar a presença de defeitos exercitando caminhos específicos dentro de um módulo ou componente do software. Um programa pode ser considerado um componente do software.

- Teste de Integração: integração dos módulos do sistema. Este nível de teste busca revelar a presença de defeitos na execução dos módulos que interagem entre si dentro de um sistema, ou mesmo nas interfaces com outros sistemas.
- Teste de Sistema: atendimento aos requisitos funcionais e não funcionais do sistema. Este teste busca identificar a presença de defeitos durante a execução das partes do sistema que trabalham juntas como descrito no projeto. Adicionalmente, deve-se referenciar a área de infra-estrutura para checar se a arquitetura utilizada está sendo usada corretamente.
- Teste de Aceitação: aceitação do sistema pelo usuário.
- Teste de Regressão: aplicado na fase de manutenção do sistema, tem o objetivo de revelar a presença de defeitos na execução das partes onde houve manutenções no sistema. Esta busca por defeitos nas partes alteradas do código ajuda a garantir que não surjam novos defeitos em componentes já testados e que o programa ainda satisfaz seus requisitos. Se, ao juntar o novo componente ou as suas alterações com os componentes restantes do sistema, surgirem novos defeitos em componentes inalterados, então se considera que o sistema regrediu.

As técnicas de teste podem ser divididas, basicamente, em técnicas denominadas "caixa-preta" e aquelas chamadas "caixa-branca". O teste "caixa-preta", também chamado *teste funcional*, considera a especificação do software, ou seja, não considera a estrutura interna ou a forma de implementação do sistema. Este é o único tipo de teste possível quando não se dispõe do código-fonte, por exemplo. O teste "caixa-branca" ou *teste estrutural*, por outro lado, deriva os requisitos de teste a partir do conhecimento da estrutura interna do programa, ou seja, sua implementação. Dessa forma, é necessário que tenhamos o fonte de todos os programas disponíveis, para podermos controlar o que foi e o que não foi testado (AGUIAR, 2004).

O critério de teste é direcionado pela técnica escolhida e serve para orientar o testador na geração dos casos de teste. Os elementos requeridos de um critério de teste são os elementos ou as características do software que deverão ser exercitados quando o software for testado. Os casos

de teste gerados devem exercitar os elementos ou as características do software definidos por aquele critério. Alguns dos critérios de um software que podem ser testados são: as linhas de comando; as funções implementadas; as variáveis definidas no software; os laços existentes no software; todos os ramos de uma decisão; e os requisitos do software (CRESPO et al., 2004).

Os tipos de teste referem-se às características do software que podem ser testadas, e compreende:

- Teste de Funcionalidade: testa a capacidade do software em fornecer funções que satisfazem as necessidades explícitas e implícitas para a finalidade a que se destina o produto.
- Teste de Interface: verifica se os dados das entradas (origem) e saídas (destino) que passaram por alguma interface do sistema, continuam com suas características iniciais mesmo após o envio da informação;
- Teste de Usabilidade: testa a facilidade de uso do software, a capacidade do software ser entendido, ser aprendido e ser atraente ao usuário quando usado sob as condições especificadas.
- Teste de Desempenho: é projetado para testar o desempenho do software durante a execução, no contexto de um sistema integrado.
- Teste de Carga (Stress): executa um sistema de um modo que demanda recursos em quantidade ou freqüência anormais dentro de um curto período de tempo. O objetivo é emular a carga gerada por um determinado número de usuários (centenas ou milhares) na aplicação a fim de analisar o seu comportamento e elaborar um plano de recomendações.
- Teste de Volume: O objetivo do teste de volume é encontrar limitações de software através do processamento de uma grande quantia de dados. Um teste de volume pode descobrir problemas que estão relacionados à eficiência do sistema, ex.: tamanhos incorretos de buffer, um consumo grande de espaço de memória, ou mostrar que uma mensagem de erro seria necessária avisando o usuário que o sistema não pode processar uma quantia grande de dados.

o Teste de Segurança: testes que verificam se os mecanismos de proteção incorporados ao sistema vão, de fato protegê-lo de invasão.

o Teste de Recuperação: é um teste de sistema que força o software a falhar de diversos modos e verifica se a recuperação é realizada.

A Figura 1, a seguir, demonstra a ligação existente entre os níveis de teste, as técnicas de teste, os tipos de teste e os critérios de teste que podem ser adotados ao se definir uma estratégia de teste.

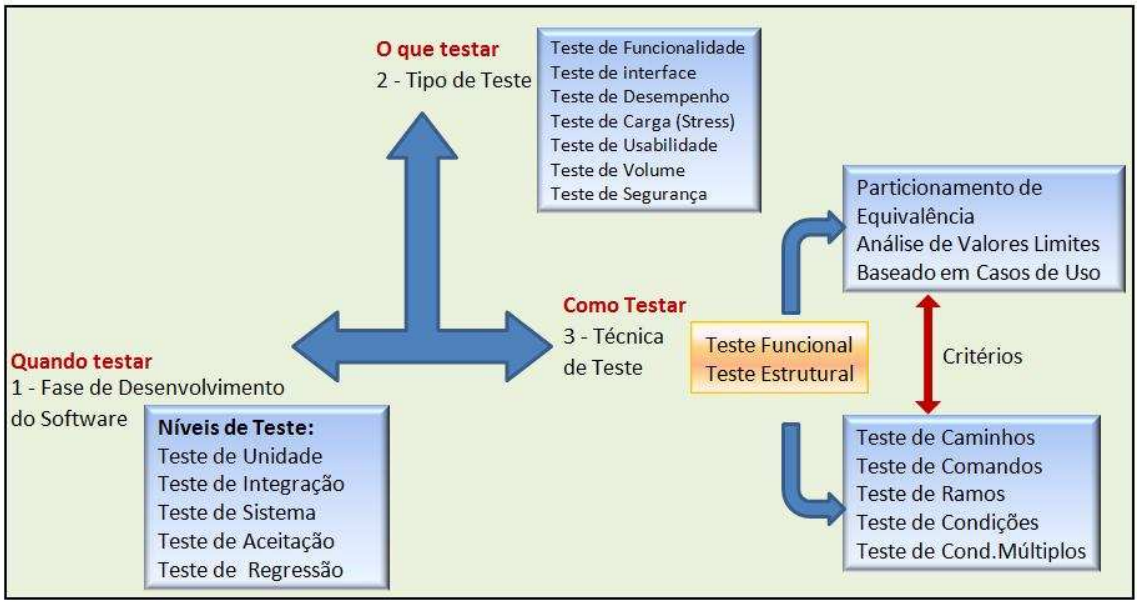

Figura 1 – Relação entre níveis, tipos e técnicas de teste (CRESPO et al.,2004)

2.1.1. Teste Funcional e o Conceito de Classes de Equivalência

O teste "caixa-preta" (funcional) parte de uma visão externa do sistema e refere-se aos testes realizados nas interfaces do software, onde demonstra-se que as funções do software são operacionais (PRESSMAN, 2001). Uma aproximação padrão para gerar teste funcional baseada na especificação é primeiro particionar o domínio de entrada de uma função a ser testada e então selecionar dados de teste para cada classe do particionamento. A intenção é que todos os elementos dentro de uma classe de equivalência sejam essencialmente os mesmos para os propósitos do teste (OSTRAND e BALCER, 1988). Uma vez que é inviável testar todos os casos

de teste existentes, as classes de equivalência dividem os casos de teste em classes, de modo que os casos dentro de cada classe sejam "equivalentes". Isso permite testar apenas um subconjunto deles, mantendo a representatividade. O problema passa a ser: como determinar as classes de equivalência?

O teste funcional avalia o comportamento do sistema. Dessa forma, as classes de equivalência devem ser construídas de modo que agrupem casos de teste para os quais o comportamento do sistema deva ser o mesmo. Na prática, precisaremos selecionar atributos das transações do sistema, que sejam determinantes para o comportamento esperado. Em um sistema bancário de contas-correntes, um atributo determinante para uma transação de depósito poderá ser *o tipo de depósito* (dinheiro, cheque da praça, cheque de outra praça, etc.). A combinação desse atributo com outros irá definir as classes de equivalência.

As classes de equivalência podem ser determinadas para as entradas das transações do sistema, caso em que são denominadas *classes de entrada*. As classes de entrada podem ser divididas em entradas válidas e entradas inválidas que são determinadas de acordo com as seguintes diretrizes:

1. Se as condições de entrada especificam um intervalo de valores, definir uma classe com entradas válidas e duas classes com entradas inválidas.
2. Se as condições de entrada especificam um valor específico (ou um determinado número de entradas), definir uma classe válida e duas inválidas.
3. Se uma condição de entradas especifica um conjunto de valores, duas situações são possíveis:
    a. Se cada elemento do conjunto é tratado de maneira diferente pelo software, definir uma classe válida para cada elemento, e uma classe inválida.
    b. Se cada elemento do conjunto é tratado da mesma maneira pelo software, definir uma classe válida e uma inválida.

4. Se uma condição de entrada especifica um valor lógico, definir uma classe válida e uma inválida.

A tabela 1, a seguir, ilustra exemplos de acordo com as diretrizes acima (mostradas em parênteses):

Tabela 1 – Condições de Entrada para Classes de Equivalência – Exemplo 1

Condições de entrada	**Classes Válidas**	**Classes Inválidas**
“O valor do desconto não pode ser inferior a R$ 1,00 e nem superior a R$ 50,00” (1)	C1: valor do desconto deve ser entre [1,50]	C2: valor do desconto < 1 C3: valor do desconto > 50
“O programa deve ler 3 valores...”(2)	C1: 3 valores são fornecidos	C2: fornecer menos de 3 valores C3: fornecer mais de 3 valores
“Para efeito de cálculo do salário um empregado pode ser horista, diarista ou mensalista” (3-a)	C1: empregado = horista C2: empregado = diarista C3: empregado = mensalista	C4: empregado não pertence a {horista, diarista, mensalista}
“Um identificador deve iniciar por uma letra”(3-b)	C1: 1º letra do identificador [A,Z]	C2: 1º letra do identificador não pertence a [A,Z]
“A senha pode ser ou não fornecida”(4)	C1: senha é fornecida	C2: senha não é fornecida

2.1.2. Definição dos Casos de Teste

Casos de teste são definidos com a finalidade de testar os elementos do software e são baseados nas condições de entrada e nas classes de equivalência. Os casos de teste podem ser elaborados para identificar defeitos nas estruturas internas do software, através de situações que exercitem adequadamente as estruturas utilizadas na codificação.

Para definir os casos de teste, podemos:

1. Escolher casos de teste que cubram cada uma das classes válidas pelo menos uma vez.
2. Escolher casos de teste que cubram cada uma das classes inválidas pelo menos uma vez, a cada teste.

A razão para que seja escolhida uma e somente uma classe inválida por vez é que o tratamento de uma entrada inválida pode mascarar o tratamento de outras entradas inválidas. Assim, por exemplo, se a especificação indica que o preço deve ser maior do que zero e a quantidade de itens não pode ser menor que 0 e nem superior a 999, a entrada: preço = 0 e quantidade = -1 pode fazer com que a parte do programa que trata valores inválidos de quantidade não seja exercitada, pois o programa pode terminar quando encontra um preço incorreto. Portanto, para o exemplo dado teríamos as seguintes classes de equivalências:

Tabela 2 – Condições de Entrada para Classes de Equivalência – Exemplo 2

Condições de entrada	**Classes válidas**	**Classes inválidas**
“O preço deve ser maior do que zero” (regra 1)	C1: Preço > 0	C2: Preço =< 0
“A quantidade não pode ser menor que zeros e nem maior que 999” (regra 2)	C1: Quantidade pertence a [0,999]	C2: Quantidade < 0 C3: Quantidade > 999

Os casos de teste seriam definidos da seguinte forma:

1. Preço > 0 e Quantidade entre [0,999] (regra 1)

2. Preço < = 0 e Quantidade entre [0,999] (regra 2)
3. Preço > 0 e Quantidade < 0 (regra 2)
4. Preço > 0 e Quantidade > 999 (regra 2)

É interessante montarmos casos de teste que exercitem os **valores limites** das classes, incluindo casos de teste que contenham valores imediatamente superior e inferior ao intervalo requerido, pois algumas falhas ocorrem justamente nas fronteiras ou limites entre as classes.

De posse dos casos de teste, esses devem ser combinados em *suites ou roteiros de teste*. Um roteiro de teste é constituído por um ou mais casos de teste, de modo a reproduzir um conjunto de transações (ou casos de uso) do mundo real. Um roteiro de teste poderia ser, por exemplo, a abertura de uma conta, seguida de um depósito e, finalmente, uma retirada. Para cada roteiro de teste definido, devem ser previamente determinadas as classes de resultados esperados. Em alguns casos será necessário calcular os valores exatos dos resultados. Em outros casos, bastará a classe de saída correspondente.

Com os roteiros de teste e os resultados esperados, passaremos à execução dos testes. Após a execução, os resultados obtidos deverão ser comparados com os esperados. As divergências serão as *ocorrências* a resolver. Uma ocorrência pode ser uma falha do sistema ou da especificação. Problemas na especificação devem ser remetidos aos usuários para correção e gerarão uma alteração no sistema. Falhas do sistema deverão dar ensejo à identificação dos componentes afetados, que por sua vez deverão ser objeto de correção. Uma vez resolvida a ocorrência, o teste que lhe deu origem deverá ser novamente executado. Adicionalmente, todos os testes deverão ser novamente executados em algum momento, antes da entrega do sistema. A re-execução dos testes anteriores é denominada *teste de regressão*. A intenção é que o sistema a ser implantado apresente um índice de falhas baixo de acordo com as expectativas.

2.2. Custo do Defeito de um Software

Uma importante questão que também tem sido abordada é a relação entre o custo de se encontrar uma falha em um software e o custo para se detectar o defeito causador daquela falha, antes que ele se propague. Além disso, é interessante observar se o benefício esperado de se encontrar o menor número de defeitos/falhas em campo é viável se comparado ao esforço da aplicação das atividades propostas por um modelo de maturidade de teste.

O custo de um defeito é o custo estimado para se corrigir um determinado problema encontrado no software, ou seja, qual o esforço que deverá ser despendido para que a correção aconteça. Este defeito, quando propagado causando falhas, pode gerar altos custos. Rothman (2002) cita que todos temos diferentes atitudes com relação à correção dos defeitos, especialmente em qual corrigir e quando fazê-lo. A escolha para fazer ou não a correção depende do tipo de produto que está sendo desenvolvido, do risco associado com a entrega do software com defeitos conhecidos ou não; o processo de desenvolvimento utilizado; e qual o custo de se corrigir o defeito, quando você realmente o faz. A junção destes fatores combinados com o grau de influência de cada um, no custo do defeito do software, pode ajudar a estimar, com maior precisão, o custo total do defeito.

Segundo Pressman (2001), de acordo com estudos feitos pela indústria, as atividades do projeto introduzem entre 50% e 65% de todos os defeitos durante a fase de desenvolvimento do processo de engenharia de software. A utilização de revisões técnicas formais é vista como um “filtro” para o processo de engenharia de software, sendo até 75% efetivas, conseguindo detectar os erros precocemente e reduzir o custo dos próximos passos nas fases de desenvolvimento e manutenção. Supondo que um erro descoberto durante a fase de projeto custe 1 unidade monetária para ser corrigido, o mesmo erro descoberto logo antes que as atividades de teste se iniciem, custará 6,5 unidades; durante os testes, 15 unidades; e após o lançamento, entre 60 e 100 unidades.

Não é simples entender o custo de se corrigir um defeito. O custo para se fazer a correção também é influenciado pela escolha do ciclo de

vida e do processo de desenvolvimento e auxilia na decisão do risco de se colocar algo em produção com defeito. Entretanto, muitos não sabem o custo para se corrigir um defeito dentro de sua organização.

Rothman (2002) elaborou uma estimativa para medir este custo. Vale lembrar que esta estimativa foi elaborada quando os esforços para correções estão voltados, totalmente, para a fase de teste de sistema, onde você tem 100% das pessoas dedicadas a encontrar e corrigir eventuais defeitos, contando o número de correções efetuadas. Nesta fase já se sabe quantas pessoas (desenvolvedores, testadores, e todos os envolvidos) trabalharam no projeto, e também a duração do teste de sistema. Isto permite que seja calculado o custo para corrigir um defeito durante esta fase do projeto. Segue a forma onde é possível determinar o custo médio para encontrar e corrigir um defeito:

$$CustoMédioparaCorrigirUmDefeito = \frac{(QtdPessoas * QtdDias)}{(QtdDefeitosCorrigidos)} * CustoPessoaDia$$

Note que o número de defeitos encontrados não é informação suficiente para estimar o custo, e sim o número de defeitos corrigidos. A fase de detecção de defeitos é somente um primeiro passo. Localizar o defeito, decidir como corrigi-lo, desenvolver testes (unitários, de sistema, etc) para as correções, e procurar outros defeitos que esta correção causou é o porque o valor da correção ser o que importa.

Os riscos associados à ocorrência de falhas em produção devem ser levados em consideração quando estamos avaliando quais as atividades de teste precisam ser realizadas para que o defeito seja encontrado precocemente. Quanto maior este risco, mais robusta deve ser a fase de teste e a metodologia utilizada.

2.3. Como Medir e Comparar Software

Pressman (2001) coloca que o software é medido por muitas razões e esta medição pode ser dividida em duas categorias: medidas diretas e medidas indiretas. As medidas diretas incluem o custo e o esforço aplicados e podem ser medidos através do número de linhas de código produzido (LOC, ou KLOC – mil linhas de código), velocidade de execução, tamanho de memória e defeito registrados ao longo de certo espaço de tempo. As medidas indiretas do produto incluem funcionalidade, qualidade, complexidade, eficiência, confiabilidade, manutenabilidade e muitas outras.

Existem várias técnicas para se estimar tempo e esforço para se construir um software. Ainda de acordo com Pressman, estas estimativas lançam as bases para outras atividades de planejamento de projetos e, uma vez que constitui um mapa com os caminhos para a bem-sucedida engenharia de software, é muito importante que todo projeto se inicie com ela. A realização de estimativas carrega riscos inerentes, e os fatores de risco podem ser representados na complexidade, no tamanho e na estrutura do projeto. Com estes dados estimados é possível identificar o quanto de esforço humano foi exigido, qual a duração cronológica do projeto e qual o seu custo.

Nas próximas seções são apresentadas algumas das métricas para se medir e comparar softwares como métricas orientadas ao tamanho, a função e a complexidade.

2.3.1. Métricas Orientadas ao Tamanho

Para se capturar a métrica de tamanho do software é necessário que o número de linhas de código do projeto seja conhecido. Juntamente a isso, é preciso saber o número de pessoas que trabalharam no desenvolvimento e o custo do projeto. Informações adicionais como o número de erros encontrados após a entrega ao cliente, dentro do primeiro ano de operação, também pode ser levantado para enriquecer as métricas. Deve-se notar que o esforço e custo representam todas as atividades de engenharia de software (análise, projeto, codificação e teste), não apenas codificação.

A partir dos dados brutos levantados pode ser montada uma tabela relacionando-os e gerar métricas de qualidade e de produtividade orientadas ao tamanho de cada projeto:

Produtividade = KLOC/pessoas-mês
Qualidade=defeitos/KLOC
Custo=$/LOC
Documentação=páginas de documentação/KLOC

Projeto	Esforço	$	KLOC	Págs.Docum.	Erros	Pessoas
Aaa-01	24	168	12.1	365	29	3
ccc-04	62	440	27.2	1224	86	5
Fff-03	43	314	20.2	1050	64	6
...	...	...	...	...	...	...

Figura 2 – Métricas orientadas a tamanho (PRESSMAN, 2001)

Embora as métricas orientadas a tamanho sejam práticas para se medir um software, por se utilizar da contagem facilmente feita do número de linhas de código, existem controvérsias que giram em torno do uso das linhas de código como uma medida-chave. Alguns autores defendem que as medidas LOC são dependentes da linguagem de programação e que elas penalizam programas bem projetados, porém mais curtos (PRESSMAN, 2001).

2.3.2. Métricas Orientadas a Função

As métricas orientadas à função são consideradas métricas indiretas e são concentradas na “funcionalidade” ou utilidade do programa. O método sugerido por Pressman (2001), apud Albrecht (1979) chamado de *ponto-por-função* (FPs) é derivado usando uma relação empírica baseada em medidas de informações e complexidade do software, observando cinco características do domínio de informação:

1. Entrada Externa (EI – External Input): transações lógicas onde dados entram na aplicação e mantém dados internos.

2. Saída Externa (EO – External Output): transações lógicas onde dados saem da aplicação para fornecer informações para usuários da aplicação.
3. Consulta Externa (EQ – External Query): transações lógicas onde uma entrada solicita uma resposta da aplicação.
4. Arquivos lógicos internos (ILF – Internal Logical File): número de arquivos, grupo lógico de dados mantido pela aplicação.
5. Arquivos de Interface Externa (EIF – External Logical File) - número de interfaces externas, grupo lógico de dados referenciado pela aplicação, mas mantido por outra aplicação.

Segundo Tavares, Carvalho e Castro (2002), neste tipo de métrica são fornecidos tabelas e diretrizes para determinar a complexidade de cada elemento funcional. A complexidade dos ILF e EIF é baseada no número de registros lógicos e no número de itens relacionados e a complexidade das transações é baseada no número de Arquivos Referenciados e no número de itens de dados Referenciados.

Os pontos-por-função são computados completando-se a seguinte tabela (PRESSMAN, 2001):

Tabela 3 – Computando à métrica ponto-por-função

			Fator de Ponderação			
Parâmetro de medida	***Contagem***		***Simples***	***Médio***	***Complexo***	
Número de entradas do usuário		X	3	4	6	=
Número de saídas do usuário		X	4	5	7	=
Número de consultas do usuário		X	3	4	6	=
Número de arquivos		X	7	10	15	=
Número de interfaces externas		X	5	7	10	=
Contagem Total →						

Quando os dados acima forem reunidos, um valor de complexidade é associado a cada contagem. As organizações que usam métodos de pontos-por-função desenvolvem critérios para determinar se uma entrada particular é simples, média, ou complexa.

Para se computar os pontos-por-função, a relação a seguir é usada:

$$FP = contagem\ total\ x\ [0,65 + 0,01\ x\ SOMA(F_i)]$$

Onde a contagem total é a soma de todas as entradas de FP obtidas a partir da tabela acima. F_i $(i = 1\ a\ 14)$ são 'valores de ajuste da complexidade' baseados nas respostas às perguntas anotadas na tabela. Os valores constantes da equação acima e os fatores de ponderação que são aplicados às contagens do domínio de informações são determinados empiricamente.

2.3.3. Métricas Orientadas a Complexidade

Dependendo da complexidade do software e a quantidade de horas necessárias para desenvolver uma aplicação, ela pode custar tão pouco como algumas centenas de dólares até vários milhares de dólares. O valor total será calculado como sendo o número de horas multiplicado pelo valor da hora.

Considerando isto, podemos relacionar que quanto maior a complexidade, maior será o esforço e tempo para a correção de eventuais defeitos encontrados durante o desenvolvimento de um software.

A complexidade de um projeto pode ser avaliada e definida através de algumas teorias. Dentre eles, existe a teoria de Halstead e a teoria de McCabe.

2.3.3.1. Ciência de Software de Halstead

A teoria de Halstead leva em consideração para o cálculo da complexidade os seguintes elementos (VILELA et al., 2004 apud HALSTEAD, 1975).

n1 – o número de operadores distintos que aparecem num programa.

n2 – o número de operandos distintos que aparecem num programa.

N1 – o número total de ocorrências de operadores

N2 – o número total de ocorrências de operandos.

Onde operandos de um programa são as variáveis e constantes, enquanto os operadores (operadores aritméticos, operadores lógicos, palavras chave e delimitadores) afetam o valor ou ordem de um operando; ambos são facilmente tabulados por um compilador, por exemplo.

Segundo Pressman (2001), Halstead usa medidas primitivas para desenvolver expressões para o comprimento global do programa, o volume mínimo potencial para um algoritmo, o volume real (número de bits exigido para especificar um programa), o nível de programa (uma medida de complexidade de software), o nível de linguagem (uma constante para uma determinada linguagem) e outras características tais como esforço de desenvolvimento, o tempo de desenvolvimento e até mesmo o número projetado de falhas no software.

Na Tabela 4, são mostradas as cinco medidas derivadas destes números (VANDOREN SCIENCES; SPRINGS, 2007):

Tabela 4 – Medidas para calculo de complexidade de Halstead

Medida	**Símbolo**	**Fórmula**
Tamanho do Programa	N	N= N1 + N2
Vocabulário do Programa	n	n= n1 + n2
Volume	V	V= N * (LOG2 n)
Dificuldade	D	D= (n1/2) * (N2/n2)
Esforço	E	E=D * V

Segundo Vandoren, Sciences e Springs (2000) as vantagens e desvantagens da utilização das métricas de Halstead são colocadas como segue:

Vantagens da Utilização da métrica de Halstead:

- Não requer análise profunda da estrutura da programação.
- Prever taxa de erros.
- Prever esforço de manutenção.
- Útil nos relatórios e cronograma de projetos.
- Mede, sobretudo, a qualidade dos programas.
- Simples para calcular.
- Pode ser usado por qualquer linguagem de programação.

- Muitas indústrias estudam a utilização de Halstead na prevenção do esforço de programação e a média do número de ‘bugs’ no programa.

Desvantagens de Halstead:

- Depende do código completo.
- Tem pouco ou nenhum uso como previsor de um modelo de estimativa.

Mas o modelo de McCabe é mais apropriado para aplicações no nível de projeto.

2.3.3.2. O Número Ciclomático de McCabe

A métrica de complexidade de McCabe foi desenvolvida em 1976 e se baseia numa representação do fluxo de controle de um programa, na complexidade ciclomática de um gráfico de programa para um módulo (PRESSMAN, 2001). Ela mede o número de caminhos linearmente independentes num módulo de programa. Uma técnica para computar a métrica de complexidade ciclomática consiste em determinar o número de regiões num gráfico planar. De acordo com Pressman (2001), uma região pode ser informalmente descrita como uma área incluída no plano do gráfico. O número de regiões é computado contando-se todas as áreas delimitadas e a área não delimitada fora do gráfico. O gráfico da Figura 4 tem cinco regiões (anotadas como R1 a R5) e, dessa forma, tem uma métrica de complexidade ciclomática V(G) = 5.

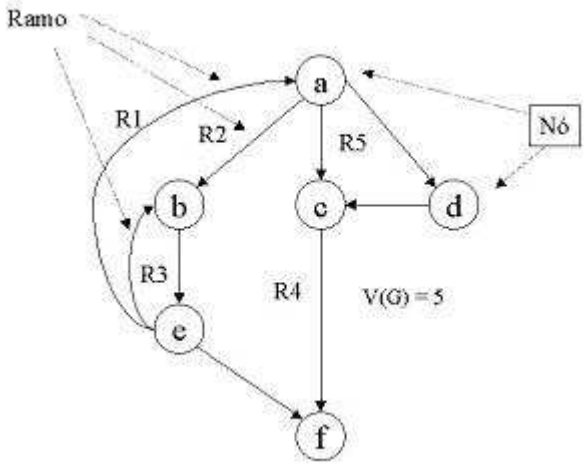

Figura 3 – Complexidade Ciclomática McCabe (PRESSMAN, 2001)

McCabe (1976) define o número da complexidade ciclomática V(G) de um gráfo G, com *n* vértices, *e* ramos, e *p* componentes conectados de acordo com a fórmula:

$$C(G) = e - n + p$$

Para o exemplo acima temos:

$$C(G) = 9 – 6 + 2 = 5$$

De acordo com a Tabela 5, quando maior o valor da complexidade ciclomática, maior será a avaliação do risco (VANDOREN; SCIENCES; SPRINGS, 2000).

Tabela 5 – Avaliação da Complexidade Ciclomática de McCabe

Complexidade Ciclomática	**Avaliação de Risco**
1 – 10	Um programa simples, sem muito risco
11 – 20	Mais complexo, risco moderado
21 – 50	Complexo, programa de alto risco
Maior que 50	Programa intestável (risco muito alto)

De acordo com KARAKAP e SULTANOOLU (1998) as vantagens e desvantagens da utilização das métricas de McCabe são colocadas como segue:

Vantagem da Complexidade Ciclomática de McCabe:

- Pode ser usada como uma métrica de fácil manutenção.
- Usada como uma métrica de qualidade, dando a complexidade relativa de vários projetos.
- Pode ser calculada previamente no ciclo de vida, mais que as métricas de Halstead.
- Mede o mínimo esforço e melhores áreas de concentração para testes.
- Guia o processo de teste limitando a lógica do programa durante o desenvolvimento.
- É fácil de ser aplicada.

Desvantagens da Complexidade Ciclomática de McCabe:

- A complexidade ciclomática é uma medida do controle de complexidade do programa e não a complexidade do dado.

- O mesmo peso é colocado nos loops aninhados ou não-aninhados. Entretanto, estruturas condicionais profundamente aninhadas são mais difíceis de entender do que estruturas não-aninhadas.
- Pode se tornar confuso com relação a muitas comparações simples e estruturas de decisão. Enquanto o método “fan-in fan-out” seria, provavelmente, mais aplicável já que ele pode rastrear o fluxo de dados.

As formas de medição de software apresentadas neste capítulo são apenas algumas das existentes na literatura. Para realização deste trabalho, foi utilizada a métrica de medição por tamanho, por ser de aplicação mais fácil e prática.

Os próximos tópicos deste capítulo apresentam uma visão geral dos dois modelos de maturidade de teste que mais se adaptam as organizações, segundo Staab (2002), o TPI (Test Process Improvement) e o TMM (Test Maturity Model).

2.4. TPI – Test Process Improvement

Neste modelo, Andersin (2004) descreve que para organizar os testes eficientemente, diferentes níveis de teste são usados, e cada nível aborda um certo grupo de requisitos ou especificações técnicas ou funcionais.

No modelo TPI é observado um processo de teste através dos diferentes pontos de vista, como: o uso de ferramentas de teste, técnicas de especificação de teste e relatórios. Estas são chamadas de áreas chaves no modelo TPI, onde cada área é classificada dentro de um nível de maturidade. Existem algumas dependências entre as áreas chave e os níveis. Por isso, uma matriz de maturidade de teste é usada. As classificações dos níveis são feitas através de pontos de checagem.

O modelo TPI contém quatro alicerces: o ciclo de vida (L) das atividades relacionadas ao ciclo de desenvolvimento, boa Organização (O), a correta infra-estrutura e ferramentas(I), e técnicas usáveis para realização das atividades (T).

Dentro destes alicerces um total de 20 áreas chaves podem ser reconhecidas para o modelo TPI. Estas áreas chaves cobrem o processo total de teste. Para permitir uma percepção no estado das áreas chaves, o modelo as fornece com níveis ascendentes, A a D. Em média existem de 3 a 4 níveis por área chave. Cada nível mais alto é melhor do que o anterior em termos de tempo, custo e qualidade.

Pontos de checagem são utilizados para determinar os requisitos dos certos níveis. Baseados nos pontos de checagem um processo de teste pode ser avaliado e para cada área chave o nível apropriado pode ser estabelecido. Os pontos de checagem são também cumulativos: para classificar para o nível B, o processo de teste precisa responder positivamente os pontos de checagem tanto do nível B como do nível A.

Devido ao grau de importância diferenciada entre cada área chave e nível associado, foi criada a Matriz de Maturidade de Teste que relaciona cada uma delas. A estrutura da Matriz de Maturidade de Teste é descrita na Tabela 6 a seguir.

Tabela 6 – Matriz de Maturidade de Teste para o TPI (ANDERSIN, 2004)

Escala / Area Chave	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Estratégia de Teste		A					B				C		D	
Modelo de Ciclo de vida		A			B									
Momento de envolvimento			A				B				C		D	
Estimativa e Planejamento				A							B			
Técnicas de Especificação de teste		A		B										
Técnicas de Teste estático					A		B							
Métricas						A			B			C		D
Automação de teste				A				B			C			
Ambiente de teste				A				B						C
Ambiente de escritório				A										
Comprometimento e Motivação		A				B						C		
Treinamento e funções do teste				A			B			C				
Escopo da metodologia					A						B			C
Comunicação			A		B							C		
Relatório		A			B		C					D		
Gerenciamento do Defeito		A				B		C						
Gerenciamento do Processo de teste		A		B								C		
Avaliação							A			B				
Teste de baixo nível					A		B		C					

As escalas da maturidade do teste pode ser divididas dentro de 3 categorias: Controlada (de 1 a 5), eficiente (de 6 a 10) e otimizada (de 11 a 13).

2.5. TMM - Modelo de Maturidade de Teste

Staab (2002) afirma que é possível melhorar o processo de teste com o uso do TMM, pois ele é um instrumento que pode ajudar a gerar alterações significativas no processo de teste de uma organização trazendo benefícios a ela. As seguintes características observadas por Staab (2002) para julgar a aceitação de um modelo de maturidade de teste foram as seguintes:

- Facilidade de entendimento e uso.
- Permite que as organizações pudessem realizar sua própria avaliação.
- Fornece uma linha base da função dos testes correntes e um guia para melhorias.
- Permite uso em telecomunicações, software baseados em WEB, e aplicações de teste de tecnologia da informação.
- Pode ser usado em conjunto com SW-CMM.

Após pesquisas realizadas, foi observado que o TMM era o modelo que mais se enquadra nos requisitos descritos acima.

O TMM foi projetado pela Dra.Ilene Burnstein do Instituto de Tecnologia de Illinois e seus associados, para ser um companheiro do SW-CMM, abordando importantes questões de gerentes de testes, especialistas de teste e certificação da qualidade de software. O TMM é um instrumento de melhoria do processo de teste e seu uso não somente ajuda a documentar o nível corrente, mas fornece um guia para fazer as melhorias necessárias para o processo.

O TMM contém 5 níveis de maturidade, com objetivos e sub-objetivos para cada nível. A descrição de cada nível, de acordo com Veenendaal (2003) é a seguinte:

Nível 1: Inicial - onde o processo de teste é caótico.

Nível 2: Definição de Fase - o teste é uma atividade planejada, separado da depuração e é definido como uma fase que segue a codificação.

Nível 3: Integração – o teste não é mais uma fase que segue a codificação, ele está integrado no ciclo de vida do software. Existe uma organização do teste e testar é reconhecido como uma atividade profissional.

Nível 4: Gerenciamento e Medições – teste é um processo medido e quantificado, revisões de todas as fases do processo de desenvolvimento são reconhecidas como teste e atividades de controle de qualidade.

Nível 5: Otimização e Prevenção de Falhas e Controle de Qualidade – devido a intra-estrutura fornecida pela realização dos níveis de maturidade dos 1 ao 4, o processo de teste pode ser definido, com custo gerenciado e a eficácia monitorada.

A Figura 4 apresenta os objetivos de maturidade de teste para todos os nível, exceto para o nível 1, o qual não possui um objetivo de maturidade.

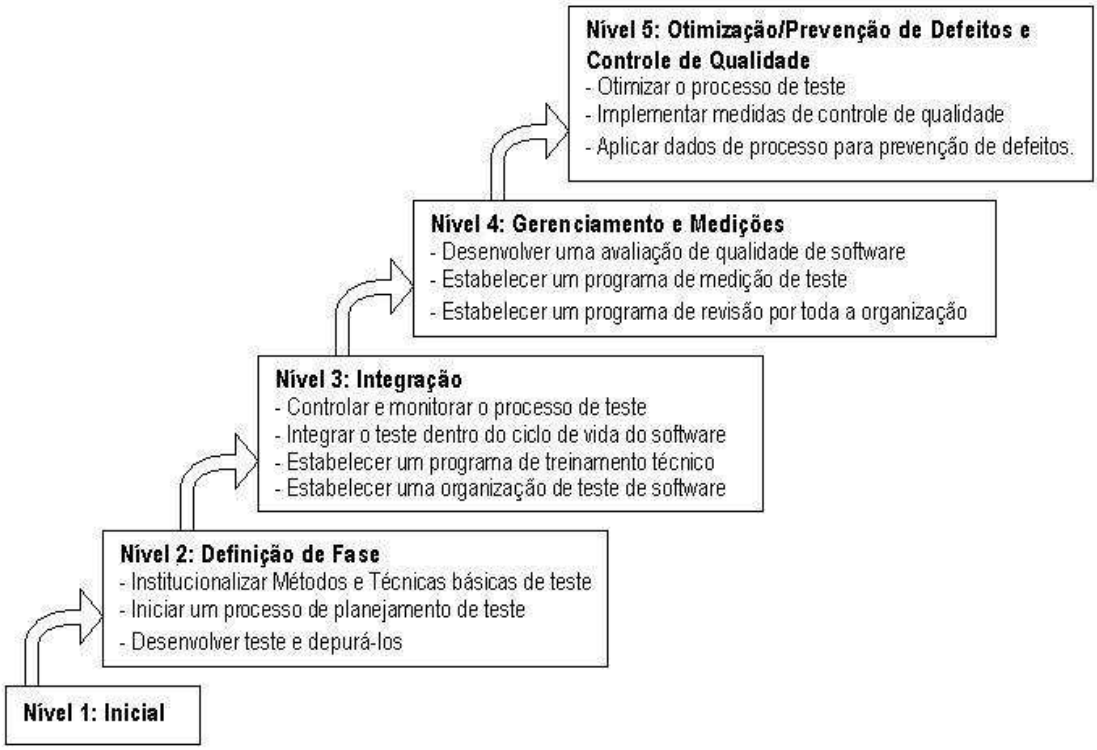

Figura 4 – Objetivos de Maturidade do TMM por Níveis (BURNSTEIN, SUWANNASART e CARLSON, Set.1996)

Cada um dos objetivos de maturidade descritos possui sub-objetivos que auxiliam a suportá-los. Ainda segundo Burnstein, Suwannasart e Carlson (1996), para o nível 2 – Definição de fase, os objetivos a serem alcançados são:

- Institucionalizar técnicas e métodos básicos de teste.
- Iniciar um planejamento do processo de teste.
- Desenvolver os objetivos dos testes e depuração.

Como esta pesquisa envolve a aplicação das diretrizes fornecidas por Veenendaal (2003) para alcançar o nível 2 do TMM, apenas este nível será descrito neste trabalho e o detalhamento de um processo de maturidade de teste nível 2 – Definição de fase – é apresentado no capítulo 3.

2.6 Trabalhos Relacionados

O TMM tem sido alvo de pesquisas na área de qualidade de software. Conforme descrito anteriormente, Staab (2002) descreve o TMM como uma das mais adequadas ferramentas para melhoria do processo de teste de software.

No experimento realizado por Olsen e Vinje (1998), foram selecionados 3 projetos para acompanhamento, onde buscou-se revelar as seguintes questões:

1. Seria possível prever o perfil total do processo de teste, em termos do grau de sucesso e satisfação expressados pelos envolvidos (stakeholders) em um projeto específico?
2. Seria possível prever os maiores problemas específicos no teste?
3. A área de teste poderia ser melhorada se o processo TMM fosse aplicado?

Projeto 1 - No primeiro projeto, um sistema de negócios de empreendimento integrado, a avaliação colocou a organização no nível 2. Previsões, usando TMM, deram ao projeto boas possibilidades no planejamento e organização do teste. Uma eficiência aceitável no teste, definido como a porcentagem de erros encontrados no teste, pôde ser esperado. Previsões dos tipos de problemas são baseadas em fraquezas já mencionadas. Estas fraquezas indicam ações

muito tardias nas deficiências do processo de teste. Os processos de teste tiveram as experiências de dois maiores problemas: 1) O produtor do software não fez o uso apropriado do material de alta qualidade de teste. 2) A organização usuária reagiu muito tarde e vagarosamente no teste, sendo tardia e inadequada. Um melhor monitoramento do processo de teste teria iluminado os problemas com o vendedor.
Neste caso o TMM não pôde prever os problemas com o vendedor, mas previu que a organização usuária reagiu de maneira inapropriada.

Projeto 2 - No segundo projeto, foi um sistema governamental de serviço civil, onde houve um novo versionamento, com novas funcionalidades e correções para o ano 2000 (Y2K). Um grande número de produtores de software foi envolvido e a organização estava no nível 1 de maturidade. O problema mais sério foi a falta de recursos. O esforço prometido não estava disponível. Então a questão é: estes problemas teriam surgido previamente se o foco da organização fosse baseado nas iniciativas do nível 2? A resposta é sim, haveria a insistência no planejamento dos testes com os recursos atuais e outras soluções seriam trazidas.

Projeto 3 - No terceiro projeto, um sistema de contas de telecomunicação, com reuso e adição kde novas funcionalidades, foi desenvolvido em três localizações geográficas com testadores localizados nas três áreas. A avaliação colocou a organização no nível 1, mas próximo ao nível 2. O TMM identificou fraquezas no planejamento de teste, técnicas e métodos que necessitaram atenção se o nível 2 é para ser atingido. O TMM também identificou fraquezas no monitoramento, gerenciamento e medição (níveis 3 e 4). Usando o TMM dá ao projeto mais possibilidades de organização e integração dos testes. Isto é importante para manter a equipe adequada ao processo de teste, quanto aos prazos finais e quando há a necessidade de se acompanhar de perto a programação.
Previsões dos tipos de problemas apontam o planejamento dos testes como área de preocupação, o que pode ser melhorado usando o TMM no planejamento do projeto.

3. TMM Nível 2 - Definição de Fase

De acordo com Veenendaal (2003), neste nível de maturidade, o processo, a estratégia e o plano de teste são estabelecidos. As áreas de processos abordadas para o nível 2 são:

- Políticas de Teste e Objetivos
- Planejamento de Teste
- Técnicas e Métodos de Teste
- Ambiente de teste

O detalhamento das atividades e objetivos que compõe as áreas de processo e a experiência em se cumprir cada um deles serão apresentados nas próximas seções, seguindo as diretrizes fornecidas por Veenendaal (2003).

3.1. Políticas de Teste e Objetivos

O que se espera desta área de processo é desenvolver e estabelecer uma política de teste e, sobretudo, a estratégia de teste contendo os objetivos, as responsabilidades e principais tarefas a serem realizadas em cada nível do teste. Neste ponto também são diferenciados o processo de teste e a depuração.

Os objetivos desta área de processo são:

- Definir e acordar uma política de teste alinhada com a política e qualidade do negócio.
- Definir e implantar uma estratégia de teste, identificando os níveis de teste, incluindo os objetivos, responsabilidades e principais tarefas de cada nível de teste.
- Definir e implantar um conjunto de indicadores de desempenho do processo de teste.

Deve existir uma política de negócio documentada e aprovada, especificando:

- Uma afirmação da missão.
- Usuário e necessidade de negócio ligado ao produto.
- Foco na qualidade do programa.

- Responsabilidades quanto à qualidade (melhorias).
- Pontos de partida quanto à aproximação da qualidade.

Um grupo com autoridade e conhecimento deve ser responsável por definir a política e estratégia de teste.

O grupo deve desenvolver, documentar, distribuir e suportar procedimentos, objetivos e políticas para os testes.

A seguir são apresentadas as atividades a serem realizadas para suportar o processo:

Atividade 1 – Definir e documentar a política de teste. Tal atividade inclui:

- A definição do teste.
- A definição da depuração (localização da falha e reparo), para permitir a distinção com o teste.
- Os objetivos do teste.
- Níveis de qualidade para serem alcançados.
- O nível de independência da organização do teste.
- Definição do processo de teste de alto nível.
- As responsabilidades chaves do teste.
- A aproximação organizacional para os objetivos de melhoria do processo de teste.
- Revisão da política pelos envolvidos.

Atividade 2 – Definir e documentar a estratégia de teste. Tal atividade inclui a definição de:

- Níveis e tipos de teste a serem executados.
- Objetivos, responsabilidade e tarefas de cada nível de teste.
- Técnicas de projetos de caso de teste a serem usadas.
- Critério de saída.
- Algum padrão que deva ser cumprido.
- Níveis de independência.
- O ambiente no qual os testes serão executados.
- Revisão da estratégia pelos envolvidos.

As atividades de teste unitário são separadas das atividades de depuração e a estratégia de teste é alinhada com a política de teste.

Atividade 3 – Definir e documentar os indicadores da execução do processo de teste:

- Os indicadores devem ser alinhados com a política e objetivos de teste para melhoria do processo de teste.

Atividade 4 – Desdobrar os indicadores da execução do processo de teste.

- Os indicadores de execução do teste devem ser registrados pelos projetos.
- Os indicadores de execução são armazenados numa base de dados.
- Os indicadores de execução são distribuídos para os envolvidos.

Para a verificação da implementação as documentações da política e estratégia de teste são gerenciadas e controladas utilizando práticas de gerenciamento de configuração.

Na aderência a implementação, os revisores ou auditores devem verificar se a política e estratégia de teste estão refletidas no plano de teste, se os profissionais de teste são familiares com a política e estratégia de teste e se os indicadores de execução de teste estão definidos e seus resultados distribuídos para os envolvidos.

3.2. Planejamento do Teste

O propósito do planejamento do teste é definir uma abordagem teste instanciada ao projeto, baseada na estratégia e estabelecer um plano bem fundado para realizar e gerenciar o teste. O planejamento deve ser constantemente definido e gerenciado.

De acordo com o risco de cada um dos requisitos será definido quais deverão ser testados e em qual nível. O objetivo é fornecer os melhores níveis possíveis de cobertura no momento e lugar certo.

O escopo do planejamento também abrange o esforço e o custo estimado para o teste. Esta área de processo cobre o objetivo do TMM de “Iniciar um Processo de Planejamento de Teste”.

Os objetivos desta área de processo são:

1. Definir e acordar uma aproximação do projeto de teste baseada nos requisitos.
2. Planejar e documentar as atividades de testes e compromissos.
3. Estabelecer estimativas bem fundamentadas para usar no planejamento e monitoramento do teste.
4. As políticas e estratégias de teste são refletidas nos planos de teste estabelecidos.

O projeto deve seguir uma política organizacional documentada para o plano de teste e especifica:

- Cada projeto define um plano de teste que inclui a aproximação de teste e o acompanhamento do esforço de teste e estimativa.
- A aproximação de teste do projeto é derivada da estratégia de teste.
- Um plano de teste deveria ser definido usando templates e procedimentos padrões.
- Os requisitos são usados como base para o planejamento das atividades de teste.
- O compromisso de testar é negociado com: gerência de recursos, de negócio e de projeto.
- O envolvimento de outros grupos afetados das atividades de teste é explicitamente combinado por estes grupos.
- Gerenciamento explícito revisa todo o compromisso de teste feito para grupos externos a organização.
- O plano de teste é gerenciado e controlado.

Uma tarefa documentada e aprovada deve existir para o projeto de teste, cobrindo o ‘sponsor’, a identificação do time de teste, os objetivos e metas, critério de saída, os itens e características a serem testados, o tipo de teste a ser feito, padrões impostos, custo e reserva de horários e de recursos.

As tarefas devem ser revisadas pela gerência de teste e dos outros grupos afetados.

O tempo adequado deve ser fornecido para a gerência de teste para realizar as atividades de planejamento de teste. Pessoas que são especialistas na aplicação objeto de teste e aquelas que têm experiência no processo de desenvolvimento devem estar disponíveis para suportar o desenvolvimento do plano de teste.

Ferramentas para suportar as atividades de planejamento de teste também devem estar disponíveis.

O gerente de teste deve ser responsável por negociar os compromissos e desenvolver o plano de teste, além de coordenar o processo de planejamento de teste do projeto.

Todos os envolvidos no planejamento dos testes devem ser treinados neste plano e acompanhar os procedimentos e técnicas, que consiste de:

- Teste Estruturado.
- Princípios de Planejamento.
- Avaliação de risco e aproximação de teste.
- Aspecto e padrões do plano de teste.
- Organização do teste.
- Esforço e custo estimado do teste.
- Cronograma do teste.
- Introdução em técnicas de projeto de teste.
- Ferramentas de suporte.

Um procedimento, que inclui um template para o planejamento do teste, deve ser especificado e documentado, e declarar que o planejamento está alinhado com a estratégia dos testes, além de endereçar todos os níveis de teste necessários.

Os procedimentos consistem de tarefas de teste de alto nível como:

- Revisão das tarefas dos testes.
- Revisão geral e estudos do sistema disponível e documentação do projeto.
- Determinar o teste base.

- Definir a abordagem do teste.
- Configurar a organização do teste.
- Especificar o que deverá ser entregue do teste.
- Especificar a infra-estrutura necessária.
- Organizar a gerência e controle.
- Analise de risco do processo de teste.
- Definir a estimativa e cronograma do teste.
- Desenhar o plano de teste.
- Aprovar e rever o plano de teste e seus compromissos.

E de tarefas de teste de baixo nível como:

- Especificar as técnicas e sua base lógica.
- Especificar o critério de saída.
- Especificar o grau de independência.
- Especificar a abordagem de integração (top-down, bottom-up).
- Especificar o ambiente de teste (hardware e software).
- Especificar o processo de teste.
- Especificar as atividades de teste.
- Especificar o cronograma de teste.

O template de teste deve cobrir:

- Uma visão total da identificação e introdução.
- Algo não compatível com a estratégia de teste.
- Itens do teste.
- Características a serem e a não serem testadas.
- Critérios de entrada e saída para os níveis de teste.
- Produtos dos testes.
- Necessidade do ambiente.
- Organização (apoio, treinamento e nível de independência).
- Tarefas e cronogramas.
- Riscos do processo de teste.
- Estimativas do esforço e custo do teste do projeto.

O procedimento para a análise de risco e determinação da aproximação do teste deve ser especificado e documentado. A avaliação dos riscos deve considerar o impacto dos defeitos, não detectados antes da

execução e durante a execução. Os envolvidos no processo devem contribuir para esta avaliação dos riscos.

A avaliação de risco é baseada em dois elementos: na *probabilidade* de um defeito ocorrer em um item do teste e no *impacto* se um defeito ocorrer em um certo item de teste.

As estimativas de teste são baseadas em três elementos: no *tamanho* e *complexidade* do produto a ser testado, na *aproximação do teste* e na *produtividade* do time de teste. Elas podem ser feitas através do top-down, usando, por exemplo, pontos de função, pontos de teste, dados de desenvolvimento de software.

A seguir são apresentadas as atividades a serem realizadas para suportar o processo:

Atividade 1 – Basear e manter o plano de teste conforme as tarefas de testes aprovadas, os requisitos, a estratégia e política de teste.

- A documentação deve estar baseada em um plano de teste padrão, além de ser revisada pela gerência e pelos outros envolvidos.

Atividade 2 – Elaborar a análise de risco utilizando o ‘input’ de vários envolvidos, sendo revisadas por eles.

- A aproximação do teste é documentada no plano de teste, incluindo sua lógica.
- Os riscos associados com o projeto de teste são identificados, avaliados e documentados.
- Identificação dos riscos do processo de teste em uma reunião com os envolvidos. Os riscos são priorizados baseados na probabilidade e impacto potencial para o processo de teste.
- As contingências para os riscos do processo de teste também devem ser identificadas.
-

Atividade 3 – Alinhar o ciclo de vida do teste com a aproximação do teste.

- Os principais estágios são: o planejamento, a preparação e a execução dos testes.
- Os marcos são identificados para cada etapa dos testes.

Atividade 4 – Todos os compromissos do projeto de testes, feitos para grupos externos a organização, devem ser explicitamente revisados pela gerência.

Atividade 5 – Fazer as estimativas de acordo com o procedimento documentado. Tal atividade deve incluir:

- Custo e esforço.
- Número de engenheiros de testes necessários.
- Outros recursos como ambiente de teste e ferramentas.

Atividade 6 – Gravar os dados da estimativa de teste e incluir as informações associadas necessárias para sua reconstrução.

- As estimativas devem ser revisadas e agregadas.

O Plano de Teste documentado deve ser gerenciado e controlado usando diretrizes do gerenciamento de configuração, como controle de versão, controle e histórico de alterações, identificação de status e uso de ferramentas para armazenamento dos documentos.

As medições orientadas as metas a serem atingidas são feitas e usadas para determinar o status das atividades do planejamento dos testes.
Exemplos de medições:

- Porcentagem de planos de teste estabelecidos de acordo com o procedimento e template.
- Porcentagem de planos de teste que tem uma analise de risco completa e aproximação de teste.
- Porcentagem de planos de teste formalmente revisados e aprovados pelo gerente.
- Esforço do planejamento do teste.
- Acurácia da estimativa de teste.

O grupo que assegura a qualidade revê e/ou audita o planejamento das atividades de teste e o produto de trabalho.

Os resultados devem ser reportados para:

- Gerente de recurso.
- Gerente de Teste.

No mínimo, os revisores e auditores verificam:

- A adequação com a estratégia de teste.
- A adequação com os padrões.
- A aproximação de testes e analise de risco.
- A estimativa de teste.

Assuntos tipicamente endereçados para revisões:

- Aplicação da aproximação dos testes.
- O desempenho técnico, de custo, e cronograma.
- Conflitos e assuntos não resolvidos no nível baixo.
- Identificação do risco do processo de teste.

Um relatório sumarizado da revisão é preparado e distribuído para os grupos afetados.

3.3. Técnicas e Métodos de Teste

O propósito das técnicas e métodos de teste é melhorar a capacidade do processo de teste durante a fase de projeto e execução dos testes através da aplicação de métodos e técnicas básicas de testes e gerenciamento de incidentes.

Esta área de processo aborda a aplicação das técnicas no projeto de teste para derivar os casos de teste e subseqüentemente executar os testes baseados na estrutura dos procedimentos de teste. Também endereça o uso de ferramentas e como dito antes, o gerenciamento de incidentes. A meta de maturidade coberta por esta área de processo é a “Institucionalizar Técnicas e Métodos básicos de teste”.

Os objetivos a serem cumpridos nesta etapa são:

1. Técnicas do projeto de teste são avaliadas, recomendadas e constantemente aplicadas através das fases de projeto de teste.
2. A execução do teste é realizada usando documentos do procedimento do teste.
3. Incidentes encontrados durante os testes são reportados usando um esquema de classificação de incidentes e gerenciados usando um procedimento documentado.

Os compromissos para realização são apresentados a seguir:

1. Um conjunto conveniente de técnicas de projeto de teste devem ser identificadas, definidas e constantemente aplicadas através da organização.
2. Técnicas de projeto de teste devem ser suportados por templates apropriados.
3. A execução dos testes é feita usando o procedimento de teste documentado.
4. Técnicas e métodos de teste são aplicados nos níveis de unidade, integração, sistema e aceitação.
5. Incidentes são documentados e reportados de acordo com um procedimento de teste.
6. Os status dos incidentes são gerenciados e rastreados.
7. Os incidentes reportados são avaliados, reportados e processados de acordo com o procedimento documentado. Um esquema de classificação básica de incidente é estabelecido.
8. Um repositório de incidente básico é gerado.

Os recursos que devem estar disponíveis para suportar esta área são:

1. O projeto de teste e as atividades de execução devem estar no cronograma.
2. Ferramentas de suporte ao projeto de teste e atividades da execução devem estar disponíveis, como: ferramenta de projeto de teste base, análise de cobertura, planilhas de gerenciamento de incidentes.

Todos os envolvidos nos testes devem ser treinados nas técnicas e métodos e nos procedimentos de acompanhamento.

As técnicas e métodos são consistidos pelos seguintes elementos:

- Atividades do projeto de teste.
- Aplicação das técnicas do projeto de teste.
- Relatórios e procedimentos dos incidentes dos testes.
- Ferramentas de suporte.

Os processos que auxiliam nesta etapa são:

Processo 1

Documentos de técnicas de projeto de teste (incluindo os templates) devem estar disponíveis nos diferentes níveis de teste.

1. Técnicas apropriadas de projeto de teste são avaliadas, selecionadas e documentadas para os níveis de teste identificados.

   As técnicas são divididas em caixa-branca, baseada na análise da estrutura do código do componente, e caixa-preta, baseada na analise do componente sem referencia para a estrutura interna.

   A técnica de projeto de teste de caixa-branca inclui: Cobertura de código, cobertura de condição e cobertura de decisões.

   A técnica de projeto de teste de caixa-preta inclui: Particionamento de equivalência, análise de valor limite, gráfico de causa-efeito, teste de transição de estado.
2. Mais de uma técnica de projeto de teste deve estar disponível para cada nível de teste a fim de efetuar a aproximação do teste e diferenciar na cobertura do código.
3. Um template de projeto de teste cobre, tipicamente (VEENENDAAL, apud IEEE 829,1994):
   - Identificador de especificação de projeto de teste
   - Itens e características para serem testadas
   - Refinamentos da aproximação
   - Especificação dos casos de teste
     i. Especificação de Input
     ii. Especificação de Output
     iii. Necessidades ambientais

- Critérios sobre o que passou ou falhou no teste

4. Um template de procedimento do teste, tipicamente cobre (VEENENDAAL, apud IEEE 829,1994):
    - Identificador de especificação de procedimento de teste
    - Propósito
    - Requisitos especiais (pré-condições de execução)
    - Passos do procedimento (ações de teste e checagens).

Processo 2

Um procedimento para execução é especificado e documentado.

1. O procedimento consiste das seguintes tarefas (VEENENDAAL, apud Pol et al, 2002):
    - Objetos de teste de entrada
    - Configuração inicial da base de dados de teste
    - Execução de (re) testes
    - Checagem e avaliação dos resultados dos testes
    - Relatório dos testes

Processo 3

Um procedimento para o relatório e gerenciamento dos incidentes de teste é especificado e documentado.

1. O relatório dos incidentes dos testes, normalmente, contém os seguintes elementos:
    - Identificador reportado do incidente do teste
    - Resumo
    - Descrição do incidente (itens incluídos resultados esperados, anomalias, data e hora, passos do procedimento, ambiente, testadores, observadores)
    - Fase do projeto
    - Repetitividade
    - Prioridade
    - Severidade
2. Para os itens selecionados do relatório de incidentes, um esquema de classificação deve estar disponível.
3. O procedimento consiste, tipicamente, dos seguintes passos:

- Reconhecimento
- Investigação
- Ação
- Disposição

As atividades a seguir suportam esta área de processo:

Atividade 1 – Especificar os casos de teste de acordo com as técnicas do projeto de teste documentadas.

- Técnicas de projeto de teste apropriadas são selecionadas de acordo com a estratégia e aproximação dos testes.
- Os procedimentos e projetos de teste são documentados e revisados.
- Os procedimentos e projetos de teste são alterados sempre que existe a alteração dos requisitos, projeto ou código sob os testes.

Atividade 2 – Realizar a execução dos testes usando os procedimentos de teste e o procedimento documentado.

Atividade 3 – Reportar e gerenciar os incidentes dos testes de acordo com o procedimento documentado.

- Os incidentes identificados durantes os testes são documentados e armazenados para serem rastreados até sua resolução.
- Os incidentes dos testes são classificados usando um esquema de classificação básica de incidentes.
- Os incidentes dos testes são documentados e usados como base para determinar se o sistema satisfaz seus requisitos.
- Os incidentes dos testes são gerenciados e controlados.

Os direcionamentos da implementação devem ser compostos por:

Gerenciamento de Configuração – Os projetos de teste documentados e os procedimentos de teste são gerenciados e controlados usando diretrizes do gerenciamento de configuração. Como, por exemplo: controle de versão, controle e histórico de alterações, identificação de status e uso de ferramentas de gerenciamento de configuração para armazenamento.

Medições – medições orientadas a metas são feitas e usadas para determinar o status das técnicas de teste e atividades dos métodos.

1. Medições são baseadas sob as metas das técnicas de teste e política de métodos.
2. Medições focam o nível de desdobramento, e a eficácia e eficiência do projeto de teste e execução de atividades.

Exemplos de medições incluem:

- Número de projetos de teste estabelecidos usando a técnica de projeto de teste (caixa preta / caixa branca).
- Tempo gasto por projeto de teste.
- Incidentes reportados pela prioridade e severidade.
- Eficácia das técnicas de projetos de teste.
- Eficiência das técnicas de projeto de teste.

Aderência – O grupo de certificação da qualidade revisa e/ou audita as técnicas de teste e os métodos das atividades.

1. Os resultados devem ser reportados ao gerente de recurso e ao gerente de testes
2. No mínimo, os revisores e/ou auditores verificam:
    - Aplicação das técnicas de projeto de testes selecionadas.
    - Se o procedimento para execução dos testes foi aplicado.
    - Se os templates foram aplicados.
    - Se o procedimento para reportar e gerenciar os incidentes de teste foi aplicado.
    - O treinamento recebido pelos engenheiros de teste.

Revisão – As atividades para as técnicas, métodos e execução dos testes são revisadas com gerenciamento em uma base periódica e em uma base de evento dirigido.

1. Questões tipicamente endereçadas:
    - A eficácia e eficiência das técnicas do projeto de teste.
    - A qualidade dos casos de teste derivados.
    - Conflitos e questões não endereçáveis em níveis mais baixos.
    - Gerenciamento de incidentes.

2. Revisar os itens de ação relacionados é associar, revisar e rastrea-los até o fechamento.
3. Um relatório sumarizado é preparado e distribuído para os grupos afetados.

3.4. Ambiente de Teste

O propósito do ambiente de teste é estabelecer e manter um ambiente no qual é possível executar e especificar testes de uma forma gerenciável e repetitiva, refletindo sempre uma situação real. Isto é especialmente verdade para o nível de teste mais alto.

A especificação do ambiente de teste é feita previamente nos projetos e deve ser revisada para certificar se está correta, viável e se representa o verdadeiro ambiente real de produção.

A disponibilidade do ambiente de teste cerca o número de questões necessárias que devem ser detalhadas como: é necessário para o teste ter um ambiente por nível de teste? Um ambiente separado pode ser muito caro. Deve ser, portanto, decidido como usar o ambiente de forma tão eficiente quanto possível.

Toda parte do projeto de ambiente de teste está sujeito à mudanças devido a alterações de hardware, desenvolvimento de ambiente de teste incremental e alterações no objeto de teste. Um gerenciamento de configuração completo do ambiente de teste é necessário para lidar com estas mudanças.

Esta área de processo endereça todas as atividades para especificar, gerenciar e certificar a disponibilidade de um ambiente de teste para ambos (baixo e alto) níveis de teste.

Os objetivos a serem atingidos nesta área de processo são:

1. O ambiente de teste deve ser especificado previamente e sua disponibilidade é certificada no tempo dos projetos.
2. Para os níveis de teste mais altos, o ambiente de teste é tão idêntico ao mundo real.

3. Ambientes de teste são gerenciados e controlados usando procedimentos documentados.

Os projetos seguem uma política organizacional documentada para especificação, gerenciamento e controle do ambiente de teste.Esta política especifica tipicamente:

1. Um relatório é preparado e distribuído para os grupos afetados.
2. Especificação, gerenciamento e controle do ambiente de testes são realizados de acordo com os procedimentos documentados.
3. Responsabilidades com respeito ao ambiente de testes são associadas.
4. A especificação do ambiente de testes deve ser feita previamente no ciclo de vida e é parte do processo de planejamento do teste.
5. Altos níveis de teste são carregados no ambiente que deve refletir a situação real tanto quanto possível.
6. Níveis de teste mais baixos, como unidade e teste de integração, devem aplicar stubs e drivers para o teste.

Recursos adequados e consolidados são fornecidos para especificar, gerenciar e controlar o ambiente de teste.

1. Experiências individuais, quem tem especialização e conhecimento técnico são disponíveis para suportar a especificação e implementação do ambiente de teste.
2. Ferramenta de gerenciamento de configuração deve estar disponível para gerenciar o ambiente de teste.
3. Tempo adequado e recursos são fornecidos para o projeto para implementar e adequar o ambiente de teste.
4. Tempo adequado e recursos são fornecidos para engenheiros para desenvolver stubs e drivers necessários para o nível de tese mais baixo.

Um grupo ou pessoa é designado para ser responsável pelo gerenciamento e controle do ambiente de teste.

1. As responsabilidades, geralmente cobrem:

- Suporte com respeito a especificação do ambiente de teste.
- Implementação do ambiente de teste.
- Gerenciamento de configuração do ambiente de teste.
- Resolução de problemas técnicos relacionados ao ambiente de teste.

- Certificar que os testes são reproduzidos com respeito ao ambiente de teste.
- Suporte e consultoria nos procedimentos do ambiente de teste relacionado e questões técnicas.
- Certificação da disponibilidade do ambiente de teste.

Os processos para suportar esta etapa são:

Processo 1 – O procedimento do planejamento de teste endereça a especificação para o ambiente de teste.

1. Este procedimento do planejamento de teste consiste das seguintes tarefas:

- Definição dos requisitos do ambiente de teste.
- Definição de uma especificação do ambiente de teste.
- Identificação de riscos e não conformidades para os requisitos do ambiente de teste.
- Definição de tarefas e responsabilidades.
- Estimar o custo, esforço e cronograma relacionado ao ambiente de teste.

Processo 2 – Os procedimentos para gerenciar e controlar o ambiente de teste é especificado e documentado.

1. Os procedimentos endereçam as seguintes questões:

- Back-up e Restore.
- Gerenciamento de mudanças.
- Gerenciamento de configuração.

Processo 3 – Um procedimento para certificar e disponibilizar o ambiente de teste é especificado e documentado.

1. O procedimento consiste das seguintes partes:

- Fazer reservas para o ambiente de teste.
- Alinhar tempo entre o desenvolvimento e teste.
- Desligar o ambiente corretamente após o uso (por exemplo: informações de como atualizar as alterações de log’s ambiente, como trazê-lo de volta em um status conhecido e remover os arquivos de teste).

As seguintes atividades devem ser realizadas para suportar a área de processo:

Atividade 1 – Especificar o ambiente de teste previamente no projeto de acordo com o procedimento documentado.

A especificação do ambiente de teste inclui:

- Componentes de Rede.
- Componentes de Software.
- Simuladores, stubs e drivers.
- Identificação de documentação de suporte como guias de usuários, guias técnicos, e manual de instalação.
- Ferramentas de suporte ao desenvolvimento de stubs e drivers.

A especificação do ambiente de teste é revisada pelos:

- Gerente de projeto.
- Especialista em teste.
- Outros grupos afetados.

A especificação é tipicamente revisada em:

- Precisão técnica.
- Ambiente para os propósitos do teste apropriados.
- Representatividade do ambiente na situação real.
- Possibilidades técnicas e econômicas.
- Possibilidade de entregas em dia.

Atividade 2 – Realizar altos níveis de teste em um ambiente de teste que é o mais possível da vida real.

- O ambiente de teste operacional está de acordo com os requisitos especificados.
- Riscos de não conformidade para os requisitos do ambientes de teste são discutidos com gerenciamento.

Atividade 3 – Gerenciar e controlar o ambiente de teste de acordo com o procedimento documentado.

- Mudanças propostas para o ambiente de teste são revisadas pelo gerente de teste.

Atividade 4 – Coordenar a disponibilidade e uso do ambiente de teste de acordo com o procedimento documentado.

Atividade 5 – Reportar os incidentes no ambiente de teste de acordo com o procedimento documentado.

- Incidentes identificados do ambiente de teste são documentados e rastreados até que sejam resolvidos.
- Incidentes do ambiente de teste são gerenciados e controlados.

Medições orientadas a metas são feitas e usadas para determinar o status das atividades do ambiente de teste.

1. Medições são baseadas sobre a área de processo do ambiente de teste e sobre as políticas da organização.
2. Medições focam o nível de implantação e a eficácia e eficiência das atividades do ambiente de teste.

Exemplos de medições incluem:

- Número de conflitos na reserva de ambiente.
- Esforço necessário para manter, reparar e atualizar.
- Número de casos de teste que falharam devido ao ambiente de teste.
- Média de tempo inoperante do ambiente de teste.
- Número de incidentes do ambiente de teste reportados.
- Porcentagem do ambiente de teste disponível em tempo e de acordo com a especificação.

O grupo de certificação de qualidade revisa e audita as atividades e produto de trabalho para o ambiente de teste.

1. Resultados reportados para:

- Gerenciamento de recurso.
- Gerenciamento de teste.
- Gerenciamento de projeto.

2. No mínimo, os revisores e/ou auditores verificam que:

- A especificação do ambiente de teste é escrita previamente no projeto de acordo com os procedimentos documentados.

- O ambiente de teste é o mais parecido possível com a situação real de produção, especialmente para os testes de alto nível.
- A disponibilidade do ambiente de teste está no nível adequado e de acordo com o procedimento documentado.
- O gerenciamento e controle do ambiente de teste devem estar de acordo com o procedimento documentado.

As atividades do ambiente de testes são revisadas com o gerenciamento em bases periódicas e bases de evento dirigido.

2. As questões endereçam tipicamente:

- Adequação do ambiente de teste.
- O desempenho técnico, de custos e da equipe.
- Conflitos e questões não resolvidas nos níveis mais baixos.

3. As revisões de itens de ação relacionadas são associadas, revisadas e rastreados até que sejam resolvidos.
4. Um relatório resumo é preparado e distribuído para os grupos afetados.

3.5. O Modelo de Avaliação TMM

Burnstein et al. (1999), descreve que o modelo de avaliação TMM é composto por três componentes: o procedimento de avaliação, o instrumento de avaliação (um questionário), e um treinamento do time e critérios de seleção.

1. O Procedimento de avaliação.

As principais metas do procedimento de avaliação são: 1) Suportar o desenvolvimento de um perfil de processo de teste e a determinação de um nível TMM; 2) Guiar a organização no desenvolvimento de planos de ação para melhoria do processo de teste; 3) Certificar que a avaliação é executada com o uso eficiente dos recursos da organização; 4) Guiar o time de avaliação na coleta, organização, e análise dos dados da avaliação.

Os passos para o procedimento da avaliação são:

- Preparação: este passo inclui a seleção e treinamento do time de avaliação, escolha de um líder, desenvolvimento de um plano de avaliação, seleção dos projetos, e preparo das unidades organizacionais participantes na avaliação.
- Condução da avaliação: após a coleta de informações necessárias, uma matriz de rastreabilidade TMM, poderá ser usada para checagem dos dados, consistência e objetividade. O nível de TMM da organização é determinado pela análise dos dados coletados usando um algoritmo de classificação, que requer primeiramente uma consideração das submetas de maturidade, então as metas de maturidade e, finalmente o nível de maturidade.
- Relatório de resultados da avaliação: as saídas incluem o perfil do processo, o nível de TMM, e o registro de avaliação. O perfil também inclui um resumo das fraquezas e limites dos processos de teste, assim como as recomendações por melhoria. O nível do TMM é um valor de 1 a 5, descrito nas seções anteriores, que indicará o nível de maturidade do processo de teste da organização. O último passo é uma descrição escrita da avaliação atual que inclui: nomes dos membros do time avaliador, as entradas e saídas da avaliação, custos e cronogramas atuais, tarefas realizadas, duração das tarefas, dados coletados, e problemas que ocorrerão.
- Análise da saída da avaliação: as saídas são usadas para identificar e priorizar as melhorias. Os alvos das melhorias do processo de teste quantitativo precisam ser estabelecidos nesta fase e estes alvos suportarão os planos de ação desenvolvidos no próximo passo.
- Plano de ação: são baseados nas melhorias de alta prioridade identificadas no passo anterior. O plano de ação descreve atividades específicas, recursos e cronogramas necessários para melhorar as práticas existentes e adicionar que faltam na organização para elevar seu nível do TMM.
- Implementando melhorias: Após os planos de ação terem sido desenvolvidos e aprovados, eles são aplicados em projetos pilotos

selecionados, os quais devem ser monitorados e rastreados para garantir o progresso das tarefas e o atendimento do objetivo.

2. O questionário de avaliação TMM.

É importante notar que o questionário TMM não é a fonte de entrada para determinar a classificação e gerar os resultados de avaliação dos testes. Os dados do questionário devem ser argumentados e confirmados, usando informações coletadas de entrevistas e apresentações, além das inspeções de documentos relevantes.

O questionário TMM é consistido de oito partes: 1) Instruções de uso; 2) origens de contexto; 3) origem organizacional; 4) meta de maturidade e questões de sub meta; 5) questões do uso de ferramenta de teste; 6) questões de tendências de teste; 7) recomendações de melhoria do questionário; 8) glossário de termos de teste.

As partes 2 e 3 do questionário são usadas para reunir informações sobre o respondente, a organização, e as unidades que serão envolvidas na avaliação TMM. As questões das metas e submetas de maturidade, na parte 3, são organizadas por níveis do TMM e inclui um gerente de teste/desenvolvimento, e uma visão de cliente/usuário. As questões são desenvolvidas para determinar se a organização tem mecanismos para alcançar os níveis de maturidade e resolver as questões de cada nível do TMM. Os componentes da ferramenta de teste registram o tipo e freqüência do uso de ferramentas de teste. A seção de tendências de teste fornece uma perspectiva de como o processo de teste da organização tem sido envolvido. Esta informação é útil para preparar o perfil e registro da avaliação.

3. Treinamento de Avaliação e Critérios de Seleção do time.

Uma vez que a avaliação do TMM pode ser feita dentro da organização, o gerente deve suportar a própria avaliação e esforços de melhoria, certificar que os recursos apropriados estarão disponíveis para conduzir a avaliação, e certificar que recomendações para melhoria serão implementadas. O treinamento do time de avaliação é necessário e este time deve entender as metas da avaliação, ter conhecimento, experiência e

habilidade próprios, além de forte habilidade de comunicação e estar confiante na melhoria do processo de teste. O tamanho do time de avaliação deveria ser apropriado ao propósito e escopo da avaliação.

As atividades do treinamento incluem exercícios de construção em times, um processo de avaliação ‘walk-throught’, preenchimento de um questionário de amostra e outros formulários de avaliação relacionados, e aprendizado no preparo de relatórios finais.

Para suportar o time de avaliação e certificar que as avaliações estão consistentes foram desenvolvidos formulários, além de uma ferramenta que implementa uma versão baseada em WEB do questionário do TMM. Os formulários e ferramentas incluem o perfil do processo e formulário de registro de avaliação, contendo:

- Template dos registros dos dados do treinamento do time.
- Matriz de rastreabilidade: é completada enquanto os dados são coletados, permite identificar as fontes de dados, checa as consistências e corretividade dos dados e resolve qualquer questão relacionada aos dados.
- Questionário baseado em WEB: projetado para auxiliar na coleta dos dados de avaliação vindos de lugares distribuídos e organizados e armazenados em um repositório central para posterior análise.

Estes componentes de avaliação do TMM contribuem para que as organizações possam direcionar suas atividades a fim de avaliar e melhorar seu processo de teste.

4. Aplicação do Modelo de Maturidade de Teste

Os benefícios gerais ao se utilizar um modelo de maturidade de teste no desenvolvimento de software são conhecidos e descritos por muitos autores, porém a avaliação quantificada destes benefícios ainda é incerta. O foco desta pesquisa é a avaliação da aplicabilidade e da eficácia do modelo de maturidade de teste TMM. O interesse está em diagnosticar se o custo/benefício da aplicação do nível 2 de maturidade de teste é suficientemente válido para reduzir o número de correções de um sistema após sua implantação, mesmo com um aumento do esforço nos testes. A intenção é comparar um Projeto de desenvolvimento de software avaliado no nível 2 do TMM com sistemas desenvolvidos sem maturidade no processo de teste.

4.1. Perfil da organização – Contexto Experimental

Este trabalho foi realizado dentro de uma empresa multinacional, cujo negócio principal não é desenvolver software. A organização não possui maturidade no seu processo de desenvolvimento de software, quando realizado pela própria empresa. Levando em consideração o modelo CMM de maturidade de processo, pode-se dizer que esta organização se enquadra no nível 1 de maturidade, apresentando um estado caótico e heróico no desenvolvimento de seus sistemas. Portanto, tanto para o processo de desenvolvimento, quanto para o processo de teste de software não existe nenhum modelo de maturidade que os suportem.

A empresa em questão conta com cerca de 70 pessoas trabalhando diretamente com a área de desenvolvimento de software, sendo 20 na área de infra-estrutura e 50 na área de desenvolvimento de sistemas, divididas entre analistas e programadores. As plataformas de desenvolvimento existentes são bastante variadas, possuindo, em sua maioria, sistemas desenvolvidos em linguagens de programação Cobol MVS (mainframe), Cobol Unix, Visual Basic e Notes, utilizando bancos de dados IMS, Oracle e DB2. O número aproximado de sistemas presentes na organização é de 230, sendo a maioria desenvolvido dentro da organização. Existem também pacotes comprados prontos no mercado, com algumas parametrizações e

customizações para se adequar às necessidades da empresa. A diversidade de arquiteturas dos softwares chega a 45 variações, tornando o ambiente de desenvolvimento bastante complexo.

4.2. Hipóteses da Pesquisa

A primeira hipótese derivou das afirmações feitas por Veenendaal (2003) e Burnstein, Suwannasart e Carlson (1996) que dizem que devido ao fato do planejamento do teste ocorrer no final do ciclo de vida do software, muitos problemas de qualidade são encontrados nesta fase. Além disso, defeitos se propagam no código, vindos da fase de requisitos e design. É importante lembrar que apenas a fase de teste do referido projeto foi suportada por um modelo de maturidade. A segunda hipótese levantada surgiu de estudos preliminares que indicam que o TMM fornece melhorias da qualidade do software através da ampliação da cobertura de teste (STAAB,2002).

Com isso, temos:

Hipótese 1. Existe uma correlação entre o aumento da qualidade da fase de teste, através da utilização do modelo TMM e um subseqüente aumento do esforço gasto no retrabalho para as correções dos defeitos encontrados nesta fase sendo capaz de aumentar o período de tempo estimado inicialmente para a atividade de teste.

Hipótese 2. Existe uma correlação entre o aumento da capacidade/maturidade da empresa, em termos do modelo TMM, e um subseqüente aumento da qualidade sendo capaz de reduzir o esforço destinado a correções no período pós-implantação e de manutenção do projeto piloto, se comparado a projetos onde não houve a utilização deste modelo para suportar o desenvolvimento.

4.3. Projeto Experimental e Metodologia utilizada

Conforme citado anteriormente, o modelo de maturidade de teste usado como referência neste trabalho foi o TMM. As características observadas por Staab (2002) a respeito do julgamento de aceitação de um modelo de maturidade de teste, citadas na seção 2.5 deste trabalho, se enquadraram nas necessidades da organização auxiliando nesta escolha.

Como o foco desta pesquisa está relacionado ao processo de teste da organização, durante seis meses (de Novembro de 2006 à Abril de 2007) foram capturadas informações sobre seu cenário atual, capazes de identificar o quanto de esforço era gasto para corrigir falhas encontradas em campo, originadas por insuficiência na atividade de teste no software. Tais informações são, posteriormente, comparadas com os dados resultantes do projeto piloto desta pesquisa.

As diretrizes fornecidas por Veenendal (2003) para se alcançar o nível 2 de maturidade de teste foram seguidas e para cada área de processo presente no modelo TMM (Políticas de Teste e Objetivos, Planejamento de Teste, Métodos e Técnicas de Teste, e Ambiente de teste), houve a necessidade de se rever os artefatos apresentados em fases anteriores ao teste no projeto. Esta revisão melhorou consideravelmente a qualidade da documentação entregue, uma vez que contribuiu para que as definições, tanto dos próprios requisitos quanto dos casos de teste, se tornassem mais minuciosas.

4.4. Condução do Experimento e Coleta de Informações

Conforme citado na seção anterior, a primeira informação necessária para analisar o funcionamento do cenário atual da área de desenvolvimento de sistemas era saber a quantidade aproximada de programas colocados em produção em um determinado período de tempo, o motivo pelo qual esta efetivação foi necessária e o tempo gasto nas possíveis correções de sistemas. Para que a captura destas informações ocorresse com

a maior acurácia possível, a atividade de coleta foi realizada da seguinte maneira:

1. Primeiramente, foi preparado um filtro para detectar todos os códigos de programas que eram colocados em produção. Para que este filtro ocorresse, antes de efetivar um programa, os analistas e programadores deveriam apresentar as documentações e evidências de teste necessárias, solicitadas pela organização, para um grupo de verificadores (formado por 3 pessoas), os quais além de aprovar as efetivações, acumularam a tarefa de contabilizar manualmente as características de cada programa que iria para produção.
2. Para tais códigos, houve uma separação inicial por nome do sistema e tipo de manutenção: melhoria, preventiva ou corretiva.
3. Devido à quantidade de correções apresentadas, a partir destas manutenções, foram selecionadas somente as alterações de caráter corretivo e realizado um novo filtro, identificando o motivo das correções e tempo gasto em cada uma delas. Com isso, houve a subdivisão das correções em: funcionalidades não testadas por completo, requisitos não avaliados nos casos de teste, alterações/inclusões de requisitos não comunicadas à área de sistemas e requisitos omissos.
4. Os dados foram coletados no período de 06 meses, de Novembro de 2006 à Abril de 2007, e armazenados em planilhas de dados para posterior comparação e análise.

A partir destes dados coletados, foi verificado que a população estudada deveria conter as mesmas características que o sistema piloto deste trabalho. Com isso, primeiramente buscou-se os sistemas que estão ativos e os que foram desenvolvidos internamente na organização. Deste subconjunto de sistemas, foram selecionados três que possuíam, aproximadamente, o mesmo número de usuários, o mesmo número de linhas de código (métrica utilizada devido à facilidade de aplicação), e ao menos parcialmente, a mesma plataforma de desenvolvimento do sistema piloto.

O projeto piloto usado neste trabalho, como referência de avaliação TMM 2, foi desenvolvido internamente na organização e nas linguagens: Lotus Notes (WEB) e Cobol MVS (Mainframe). Toda a parte de interação com o usuário foi codificada em Lotus Notes, e a parte de carga da base de dados, em Cobol.

Os dados em comum entre o projeto piloto e os demais sistemas a serem comparados foram caracterizados da seguinte forma: os sistemas apresentam aproximadamente 760 mil linhas de código (KLOC), 15 usuários e serem desenvolvidos, ao menos, parcialmente, na plataforma Mainframe (Cobol).

Cada área de processo que compõe a estrutura do nível 2 do modelo TMM foi abordada a fim de garantir que suas atividades fossem realizadas da melhor maneira possível dentro dos recursos disponíveis para este projeto na organização, como ferramentas, infra-estrutura e pessoas. Foram analisadas as vantagens e dificuldades com relação a aplicabilidade das práticas do modelo.

Para este projeto, inicialmente foi estimado um período total de 180 horas para a fase de testes, incluindo os testes de aceitação do software, contando com um engenheiro de teste, um programador e cinco representantes da área usuária.

Nas atividades relativas à área de processo *Planejamento do Teste*, o plano de teste definido para o projeto foi composto por uma série de indicativos para se capturar as características do teste a ser realizado no módulo em questão. Os níveis, tipos, casos de teste e valores dos casos de teste foram indicados em lugares específicos da planilha elaborada, além de haver a indicação do roteiro para se chegar ao ponto específico do teste e qual o componente que está sendo testado com os dados do caso de teste.

Para a área de processo *Métodos e Técnicas de Teste*, foi utilizada técnica de teste funcional, como a definição de classes de equivalência. Os seguintes tipos de teste foram realizados no projeto: funcionalidade, interface, usabilidade, desempenho, carga, volume e segurança. E os níveis de teste aplicados foram: testes unitários, de integração, de sistema, de aceitação e de regressão. A maneira de armazenar

os casos de teste foi elaborada de forma simples e eficaz para suportar todos os níveis e tipos de teste definidos.

Também foi criada uma planilha de dados para registrar todos os incidentes gerados a partir dos testes. Durante a execução da fase de teste as falhas encontradas foram registradas, indicando: o local da falha, qual o roteiro para que esta falha fosse re-testada, qual a severidade da ocorrência e a prioridade em sua resolução.

Com relação à área de processo relacionada ao *Ambiente de Teste*, de acordo com a especificação o projeto deveria conter os aplicativos de planilhas de dados e de conexão com DB2 instalados, além de ser capaz de realizar o processamento dos agentes do Notes adequadamente com alto volume de informações.

Durante os primeiros testes com os agentes do Notes rodando automaticamente, sendo agendados de tempos em tempos, notou-se que o servidor não conseguia disparar tais processos. Para resolução desta questão, houve a alocação e reconfiguração de um novo servidor que atendia estas necessidades. Embora houvesse um esforço adicional para reproduzir a situação do ambiente de produção, não foi possível nivelar todas as configurações do ambiente de teste e produção, principalmente com relação ao Notes, o que acarretou em alguns problemas durante a implantação do projeto.

A princípio, este software foi liberado para um grupo de seis (6) usuários representando o perfil operacional de todas as áreas envolvidas na utilização do software para verificação da aceitação do produto. Cinco (5) deles não conheciam o software até meados da sua liberação e apenas um (1) havia participado ativamente no desenvolvimento e testes do produto.

4.4.1. Indícios da Veracidade das hipóteses

Os indícios de que a primeira hipótese seja verdadeira só seriam possíveis de analisar após o final da fase de testes do projeto, quando se teria o tempo total destinado a ela. Para poder identificar o encerramento da fase de testes, dois indicadores foram apontados: 1) ou o final da fase de testes se daria quando todos seus casos de teste fossem testados, havendo

uma cobertura das principais características funcionais do sistema; 2) ou quando o tempo total gasto na fase de testes demorasse o dobro do tempo previsto e atingisse uma cobertura de, no mínimo, 80% dos casos de testes executados e validados.

Para analisar a segunda hipótese deste estudo de caso, a mesma coleta de dados realizada para os três sistemas citados anteriormente foi feita para o projeto piloto deste trabalho, no período de seis meses posterior a data de sua implantação. Este projeto piloto foi disponibilizado para produção em Maio de 2007 e, de Maio à Outubro de 2007 o esforço gasto em correções originadas por problemas nos testes foi medido para ser comparado com os sistemas similares onde não houve um processo maduro de teste sendo aplicado. O resultado obtido desta comparação seria capaz de avaliar a hipótese referente a redução do esforço gasto nas correções de sistemas que utilizam o TMM no seu processo de teste.

No próximo capítulo serão apresentadas as avaliações da aplicabilidade, aderência e eficácia do modelo de maturidade de teste TMM pela organização.

5. Resultados e Discussões

5.1. Resultados da coleta inicial

Os primeiros dados da pesquisa vieram da coleta de informações sobre o cenário atual da organização. Os resultados observados através das efetivações ocorridas no período mostraram que a alteração da maioria dos códigos colocados em produção do total de sistemas da organização surgiu de manutenções corretivas, conforme mostra a figura a seguir:

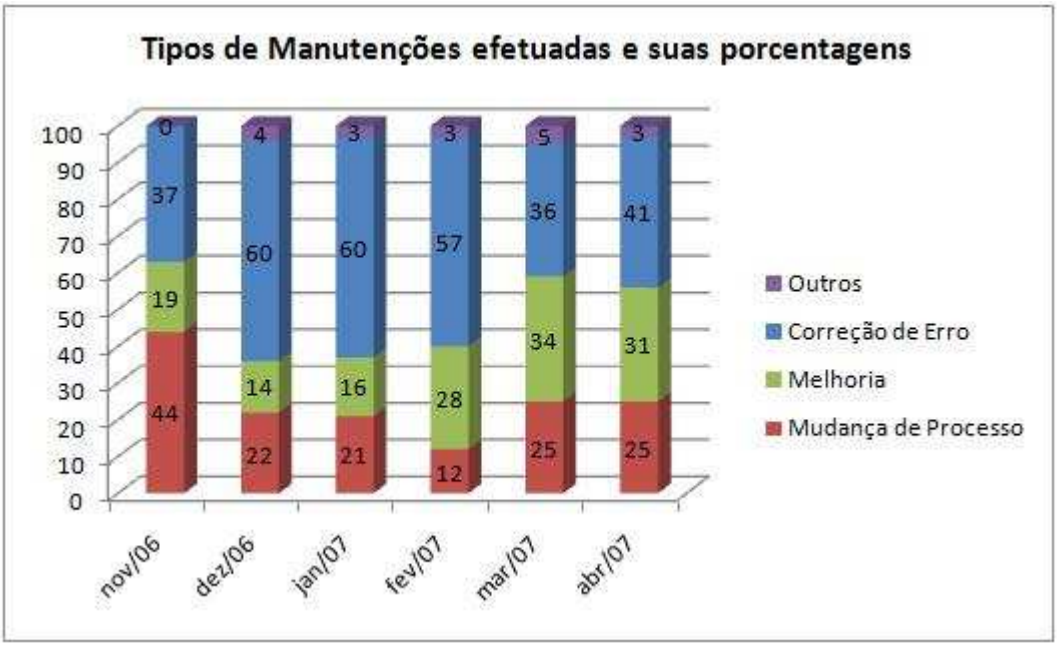

Figura 5 – Porcentagens (%) de Manutenções efetuadas em códigos de programas de todos os sistemas da empresa no período de Novembro à Abril de 2007.

5.2. Aplicabilidade e aderência ao modelo

Para que o processo de teste do projeto piloto conseguisse atingir nível 2 de maturidade no TMM, foram necessárias mudanças consideráveis no processo de teste, agregando melhorias na maneira de documentação e organização de todo o planejamento e execução dos testes. Para cada área de processo descrita anteriormente no modelo TMM foi

analisada e avaliada a aplicabilidade de cada atividade e a aderência da organização perante as práticas envolvidas.

Na área de processo *Políticas de Teste e Objetivos*, a definição do nível de qualidade a ser atingido foi alvo de questionamentos, pois na medida em que os testes eram realizados, novos requisitos eram impostos e o nível esperado de qualidade tinha que ser revisto. As negociações com a área usuária chegou ao ponto de haver a necessidade de limitar tais solicitações para evitar renegociações de prazos. Com isso, tornou-se difícil quantificar o nível de qualidade esperada do software.

Outro aspecto observado no decorrer destas atividades foi com relação à dificuldade da área usuária em assumir as responsabilidades perante os testes e em cumprir os prazos estipulados para realização dos testes de aceitação. Mesmo ciente de seus compromissos com o projeto, houve momentos de desvio de atenção dos usuários envolvidos, seja pela ocorrência de outras atividades prioritárias naquela área ou mesmo pela constante mudança de usuário-chave do projeto piloto.

Como na maioria dos projetos de desenvolvimento de software existe uma grande instabilidade com relação aos requisitos levantados, é de extrema importância que o seu gerenciamento e definição de escopo seja feito e refletido no planejamento dos testes. Isso influencia diretamente na qualidade esperada do software. A definição do nível de qualidade só é possível quando as definições estão adequadamente documentadas e revisadas pelos envolvidos. Além disso, a proximidade, aceitação e responsabilidade dos envolvidos nas atividades que compõe o processo têm um grande peso com relação ao sucesso do projeto.

Para a área de processo *Métodos e Técnicas de Teste*, surpreendentemente, o teste de regressão revelou ser um grande aliado na descoberta de defeitos e atentou para o fato do esforço gasto em re-trabalho na codificação. O prazo inicial de testes planejado em 180 horas foi praticamente dobrado devido a quantidade de defeitos encontrada e tempo de re-trabalho para efetuar as correções necessárias.

Com relação ao preparo do ambiente de teste para refletir a situação real de produção, percebeu-se que, infelizmente, a falta de

investimento nesta área de processo se torna um problema visível quando existe a ocorrência de incidentes no período pós-implantação relacionados a configuração de ambiente diferenciada. Para os softwares desenvolvidos internamente na organização, acredita-se que a maior dificuldade esteja em conseguir um ambiente adequado para os testes, principalmente devido ao custo e esforço para refletir a situação de produção.

Embora a forma de documentação deste projeto fosse completamente manual (em editor de texto e planilhas de dados), ela auxiliou tanto na medição dos defeitos encontrados, através da utilização de um relatório de ocorrências, quanto em saber o momento em que um determinado requisito deveria ser testado e qual o resultado esperado de cada nível de teste, pois o plano de teste usado foi preparado de acordo com as especificações e apresentava todas as informações necessárias para verificação de tais requisitos. O maior problema em se estruturar manualmente as documentações está na dificuldade em manter as alterações atualizadas e conseguir rastrear os requisitos e casos de teste, o que pode comprometer a qualidade do produto final. A automatização das documentações, através de ferramentas adequadas, será futuramente avaliada na organização.

5.3. Avaliação da Eficácia do modelo

Os dados coletados do cenário atual da organização mostram que os sistemas a serem comparados com o projeto fruto desta pesquisa, apresentaram, durante seis meses, um total de 331 horas de correções, conforme descreve a Figura 6.

Figura 6 – Horas e porcentagens gastas em correções durante seis meses, nos sistemas a serem comparados com o estudo de caso deste trabalho.

Para o projeto piloto foram projetados 411 casos de teste diferentes para serem executados. Foi executado um total de 359 casos de teste, realizando uma cobertura de cerca de 87% do total de casos de teste projetados. A Figura 7, abaixo, mostra estes dados.

Figura 7 – Número de casos de teste projetados e executados no Projeto Piloto.

Os dados extraídos do Relatório de Incidentes, documento preenchido durante a execução dos casos de testes, mostram que foram

encontradas 68 falhas neste período. Deste total, com a utilização das práticas do TMM, a equipe de teste detectou 65 falhas, e 3 falhas foram encontradas pela área usuária. O esforço gasto para correção destas falhas resultou no aumento do prazo inicialmente estimado para a fase de teste. A eficiência da equipe de teste é mostrada na Figura 8.

Figura 8 – Relação entre as falhas encontradas pela equipe de teste e pela área usuária no Projeto Piloto.

Após a implantação do projeto, os dados relativos a qualquer necessidade de alteração do código no ambiente de produção foram capturados. Estas alterações foram divididas em 3 categorias principais: correções, melhorias e outros. A coleta dos dados feita nos seis meses após a implantação do projeto com processo de teste no nível 2 do TMM está representada na Figura 9.

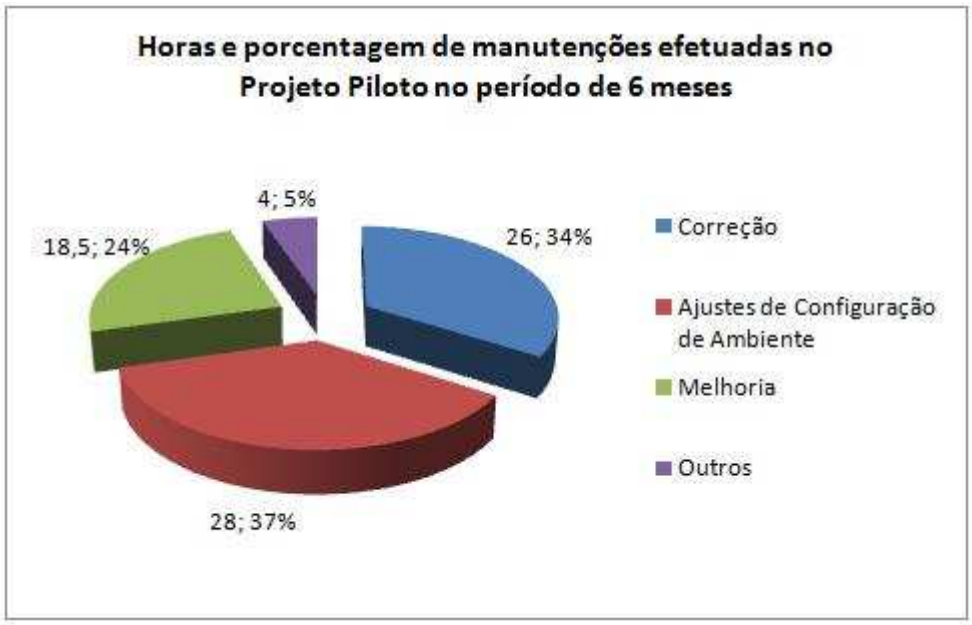

Figura 9 –Horas e porcentagens gastas nas alterações realizadas no Projeto avaliado no nível 2 do TMM no período de Maio à Outubro de 2007.

A Figura 10 apresenta a comparação entre as horas gastas nas correções, em um período de seis meses, do Sistema Piloto desta pesquisa e os Sistemas equivalentes a ele, em termos de LOC, número de usuários, e plataforma de desenvolvimento.

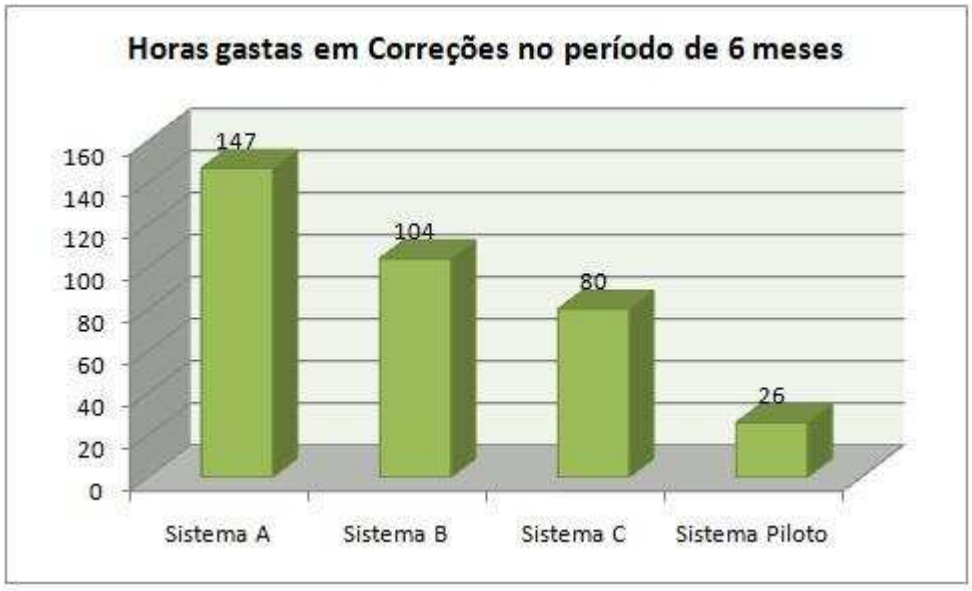

Figura 10 – Horas gastas nas correções dos Sistemas envolvidos na pesquisa durante um período de 6 meses.

Na Figura 10, pode-se observar que a quantidade de horas gastas em correções devido a problemas relacionados a teste nos sistemas que não possuíam maturidade de teste variou entre 80 e 147 horas, enquanto a quantidade de horas gastas no sistema piloto foi muito inferior, não ultrapassando 26 horas. Com este resultado, consegue-se perceber a eficácia de se utilizar o modelo TMM para dar o embasamento na fase de teste de um sistema.

Considerando que a comparação foi feita entre sistemas que já estavam em uma fase de manutenção e um sistema novo, onde o índice de incidentes ainda é alto, foi realizada uma nova medição para avaliar o sistema piloto em fase de manutenção. Para isso, foram selecionados os dados de correções descartando os dois primeiros meses após implantação do sistema piloto e isolando os 4 meses onde este sistema já estava em fase de manutenção.

A Figura 11 mostra o número médio de horas gastas por mês em correções tanto para o Sistema Piloto quanto para os demais Sistemas na fase de manutenção. Pode-se observar que os resultados desta última medição são ainda melhores.

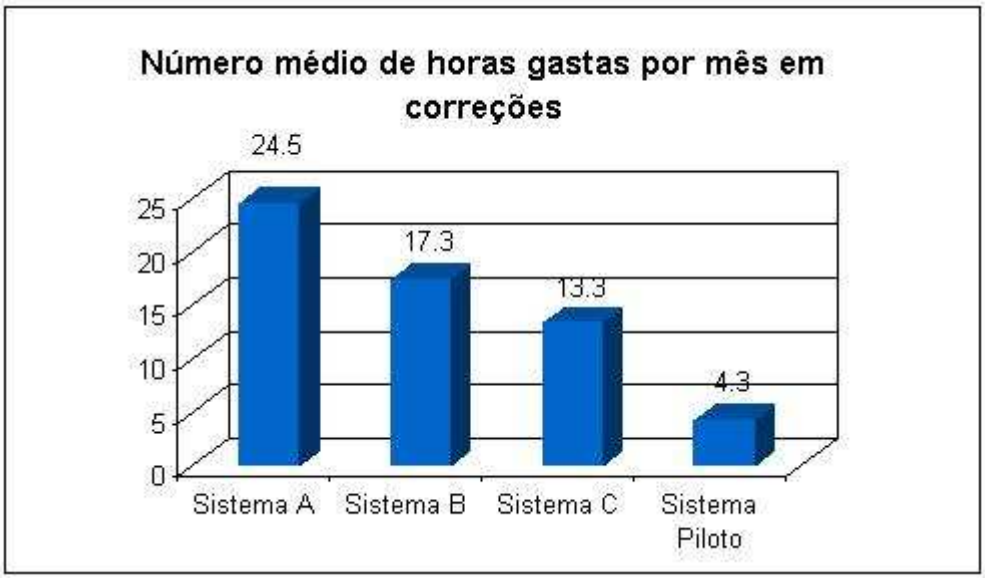

Figura 11 – Média de horas gastas por mês nas correções dos Sistemas envolvidos na pesquisa em fase de manutenção.

Mesmo estando em fase de manutenção, onde é possível existir a introdução de novos defeitos, a permanente utilização do modelo manteve reduzido o esforço gasto em correções.

Como dito anteriormente, a área específica para questões relacionadas ao desenvolvimento e suporte a sistemas na organização não possui nenhuma maturidade quanto ao processo de desenvolvimento de software, e a expectativa era de que, com a utilização de métodos e técnicas de teste, o número de defeitos encontrados nessa fase pudesse prorrogar o prazo de entrega do software. Isto foi confirmado com resultado apresentado através dos relatórios de incidentes reportados durante o projeto.

Com relação à primeira hipótese, verificou-se que, com os defeitos encontrados e a necessidade de re-trabalho, o prazo estimado para os testes seria insuficiente para a execução de todas as etapas necessárias para finalização desta fase. Assim, o período de testes foi prorrogado chegando a, aproximadamente, o dobro do tempo inicialmente estimado. Contando que foram previstas 180 horas de teste, identificou-se que foram gastas 340 horas para esta fase. Com isso, praticamente a totalidade dos casos de testes foi executada, deixando de executar somente aqueles que não foram julgados relevantes ou que já haviam sido cobertos por outros casos de teste.

Já com relação à segunda hipótese onde se colocou a provável existência de uma correlação entre o aumento da capacidade/maturidade da empresa e um subseqüente aumento da qualidade, reduzindo o esforço destinado à correções no período pós-implantação e de manutenção do projeto piloto, viu-se que este esforço, se comparado a projetos desenvolvidos sem a utilização do modelo TMM, foi reduzido em mais de 50%.

A partir dos dados apresentados nesta seção pode-se dizer que existem indícios de que as duas hipóteses elaboradas neste estudo de caso sejam verdadeiras.

6. Conclusão e Trabalhos Futuros

Este estudo buscou intensificar a preocupação com relação à melhoria da qualidade do software desenvolvido, através das atividades relacionadas ao processo de teste de software. Foram aplicadas as práticas do modelo de maturidade de teste TMM para alcançar o nível 2 de maturidade e, com isso, obtidas informações interessantes dentro de uma organização capaz de auxiliar futuros trabalhos de implantação deste modelo.

A aplicação de um modelo de maturidade de teste em uma organização pode ser uma prática possível e eficiente na descoberta de defeitos precocemente no software. Para isso, é imprescindível o comprometimento dos envolvidos e gerentes de projeto em garantir que as atividades essenciais das áreas de processo existentes no modelo sejam executadas. Além disso, o apoio e investimento da alta administração perante as atividades dos testes são importantes no suporte necessário para alcançar o sucesso do processo.

Os resultados apresentados para suportar as hipóteses levantadas mostraram que com a maior maturidade obtida no processo de teste, mesmo que aplicada tardiamente no projeto, é possível entregar um software mais confiável para o cliente. Neste caso, o que sempre deverá ser levado em consideração é que se não houver um processo maduro que suporte as fases de desenvolvimento do software, é provável que haja um aumento do prazo estimado para a fase de teste.

Através dos dados conseguidos neste trabalho, pretende-se realizar novas experiências ligadas a aplicação do processo de maturidade de teste para que melhorias contínuas das práticas relacionadas ao modelo possam ser adquiridas.

7. Referências Bibliográficas

AGUIAR, Mauricio. Ti Métricas para medir melhor seu processo de Software – Teste de Software e Melhoria de Processos. **A PSM Transmition Organization**, 2004.

ANDERSIN, Jari. TPI – A model for Test Process Improvement. In: Seminar on Quality Models for Software Engineering Department of Computer Science, University of Helsinki, Oct. 2004.

ANGELO, Fernanda. Qualidade de software vai além de teste. **Jornal COMPUTERWORLD**, Edição 461. Outubro 2006. Disponível em: <http://computerworld.uol.com.br/gestao/2006/10/04/idgnoticia.2006-10-04.5523272156/IDGNoticia_view >. Acesso em 23 Mar. 2007.

BURNSTEIN, Ilene, SUWANNASART, Taratip, CARLSON, Robert. Developing a Testing Maturity Model: Part I. **Crosstalk, Software Technology Support Center**, Aug. 1996.

BURNSTEIN, Ilene; SUWANNASART, Taratip; CARLSON, C.Robert. Developing a Testing Maturity Model: Part II. **Crosstalk, Software Technology Support Center**, Sep. 1996.

BURNSTEIN, Ilene; HOMYEN, Ariya; SUWANASSART, Taratip; SAXENA, G., GROM, R., A Testing Maturity Model for Software Test Process Assessment and Improvement. Illinois Institute of Technology. **SQP 99**, Nº 4, 1999.

CRESPO, Adalberto; SILVA, Odair; BORGES Carlos; SALVIANO Clênio; JINO Mario. **Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo**. CenPRA - Centro de Pesquisa RenatoArcher. In: Simpósio Brasileiro de Qualidade de Software, 2004. Disponível em <www.sbc.org.br/bibliotecadigital/download.php?paper=254>. Acesso em 18 Nov.2006.

HUBER, Jon. Efficiency and Effectiveness Measures to Help Guide the Business of Software Testing, **Applications of Software Measurement**- Hewlett Packard Company, 1999.

Institute of Electrical and Eletronic Engineers. IEEE 829 Standard for Software Test Documentation. IEEE Standards Boards, New York: IEEE Computer Society, Sep. 1998.

KARAKAP, Umit; SULTANOOLU, Senser. **Complexity Metrics and Models**, Out. 1998. Disponível em <http://yunus.hacettepe.edu.tr/~sencer/complexity.html>. Acesso em 15 Fev. 2007.

KITCHENHAM, Barbara, PFLEEGER, Shali, JONES, Peter, HOAGLIN, David, EMAM, Khaled, ROSENBERG, Jarret, Preliminary Guidelines for Empirical Research in Software Engineering, **IEEE Transactions on Software Engineering**, Vol.28, NO.8, August 2002.

McCABE, Thomas, A Complexity Measure. **IEEE Transactions on Software Engineering**, Vol.SE-2, No.4, December 1976.

NTAFOS, Simeon. Testing and the Cost of Field Failures. In: International Symposium on Software Reliability Engineering - ISSRE, 1999.

NTAFOS, Simeon. The Cost of Software Failures, In: Proc. Of International Association of Science and Technology for Development - IASTED International Conference On Software Engineering, pp.53-57, Nov.1997.

OLSEN, Klaus; VINJE, Poul. **Using the Testing Maturity Model in practical test-planning and post-evaluation**. Disponível em <http://www.csan.iit.edu/~tmm>, Acesso em 16 Out. 2006.

OSTRAND, Thomas; BALCER, Marc. The category-partition method to specifying and generating functional tests. **Communication of the ACM**. Vol.32, Nro 6, June 1988.

PRESSMAN, Roger. **Software Engineering.** 5º Edição, Ed.McGraw-Hill,2001.

ROTHMAN, Johanna**. What Does It Cost to Fix a Defect?**, Feb.2002. Disponível em <http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=3223>, Acesso em 18 Mai.2006.

SOUZA, Adilson. **Implementando o CMMI (Capability Maturity Mode Integration) como ferramenta para gerenciamento de projetos de Software**, 01 Dec, 2005. Disponível em <http://kplus.cosmo.com.br/materia.asp?co=30&rv=Vivencia>, Acesso em 17 Set. 2006.

STAAB, Thomas. **Using SW-TMM to Improve the Testing Processes**. Nov.2002. Disponível em <http://www.stsc.hill.af.mil> .Acesso em 14 Out.2006.

TAVARES, Helena; CARVALHO, Ana Elizabete; CASTRO, Jaelson. **Medição de Pontos por Função a Partir da Especificação de Requisitos**. Serpro – Empresa do Ministério da Fazenda, Universidade Federal de Pernambuco, 2002.

Tmap home pages, Tmap – Sogeti Nederland B.V., 2004, Disponivel em < http://www.tmap.net >. Acesso em 16 Out. 2006.

VEENENDAAL, Erik. **Guidelines for Testing Maturity – "The Test Maturity Model",** 2003.

VEENENDAAL, Erik; SWINKELS R. Guidelines for Testing Maturity – Part 1: The TMM Model. **Professional Tester**, Vol 3, Nro 1, Mar. 2002.

VEENENDAAL Erik. Guidelines for Testing Maturity – Part 2: Test Maturity Model level 2. **Professional Tester**, Vol 3, Nro 2, Jun. 2002.

VILELA, Plinio; LUCCA, Waldo; CORSO, Ariane; JINO, Mario; Comparison of size and Complexity Metrics as Predictores of the Number of Software Faults. In: JIISIC, 2004.

VANDOREN, Edmond; SCIENCES, Kaman; SPRINGS, Colorado. **Halstead Complexity Measures**, Jan. 2007. Disponível em <http://www.sei.cmu.edu/str/descriptions/halstead_body.html>. Acesso em 10 Fev. 2007.

VANDOREN, Edmond; SCIENCES, Kaman; SPRINGS, Colorado. **Cyclomatic Complexity.** Jun. 2000. Disponível em <http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html> Acesso em 12 Fev. 2007.

Wikipedia – The Free Encyclopedia. **Teste de Regressão**. Disponível em <http://pt.wikipedia.org/wiki/Teste_de_Regress%C3%A3o>, Acesso em 23 Mar. 2007.

Wikipedia –The Free Encyclopedia. **Software testing.** Disponível em <http://en.wikipedia.org/wiki/Software_testing>. Acesso em 10 Fev. 2007.

WOHLIN, Claes; HÖST, Martin; HENNINGSSON, Kennet; **Empirical Research Methods in Software Engineenring**. In Empirical Methods and Studies in Software Engineering: Experiences from ESERNET, pp.7-23, editors Reidar Conradi and Alf Inge wang, Lecture Notes in Computer Science, spinger-Verlag, germany, LNCS 2965.