CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana

Curso Superior de Bacharelado em Análise de Sistemas e Tecnologia da Informação

Qualidade de *Software*

Tiago Altomare

Americana, SP

2013

CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana

Curso Superior de Bacharelado em Análise de Sistemas e Tecnologia da Informação

Qualidade de *Software*

Tiago Altomare

Trabalho Monográfico desenvolvido em cumprimento à exigência curricular do Curso Superior em Análise de Sistemas e Tecnologia da Informação da Fatec-Americana, sob orientação do Prof. Mestre Alberto Martins Júnior.

Área: Engenharia de *Software*

Americana, SP

2013

BANCA EXAMINADORA

Prof. Mestre Alberto Martins Junior

(Orientador)

Prof. Dr.

Prof.

DEDICATÓRIA

Aos meus pais pela confiança e apoio depositados em mim durante toda minha graduação. A minha Avó Isabel que esta sempre olhando por mim. E a todos os demais familiares e amigos que contribuíram de alguma forma para minha formação.

AGRADECIMENTOS

Eu agradeço a Deus, pois toda perseverança que tive durante esses quatro logos anos veio dele, agradeço a cada pessoa que Ele pôs em meu caminho, cada dificuldade vencida, a cada momento de fraqueza superado.

Em especial agradeço aos meus familiares, pai mãe e irmão, sempre presentes nessa caminhada, as minhas queridas colegas de turma, Tamara, com quem compartilhei diversos momentos e sempre me ajudou a me manter firme, Josiane, sempre me levantando quando eu pensava em cair e Natália, por compartilhar algumas das mesmas dificuldades que eu tive, o que fez com que ela sempre me compreendesse.

Aos demais amigos da faculdade Marshall, Lucas, Amauri, Rodrigo entre muitos outros que ficarão na memória, pois fizeram parte dessa louca jornada.

Ao Professor Alberto Martins Júnior, que sempre me tratou com muito respeito, e marcou essa experiência tão importante na minha vida.

“Quando puder medir aquilo de que esta falando e expressar em números, você saberá algo a respeito.”

Lord Kelvin

RESUMO

A garantia da qualidade em tudo que se faz nos dias atuais é imprescindível para quem deseja alcançar o sucesso, mesmo que seja um individuo ou mesmo uma organização. Qualidade esta presente no ciclo de vida de um *software* assim como no ciclo de vida de qualquer produto, e deve ser medida de acordo com métricas bem definidas. Hoje existem centenas de métricas já desenvolvidas e muitos conjuntos dessas mesmas métricas são criados buscando melhores resultados, porém mesmo assim ainda existe um desafio na produção de *software* de qualidade, a otimização dos processos envolvidos no desenvolvimento do produto de *software*. Este estudo busca trazer em seu contexto, modelos e normas de melhoria de processos organizacionais, que visam garantir a excelência da qualidade do produto gerado.

Palavras-Chave: Qualidade, Métricas e Modelos.

ABSTRACT

These days in everything we do the assurance of quality is a must for those wanting to achieve success even if as a person or an organization. Quality is present in the cycle of life of a *software*, just as it is in any other product and it should be measured with well defined metrics. Today there are hundreds of metrics already developed, and many groups of these metrics are created in demand of better results, though there's still a challenge in the production of quality *software*. This study seeks to bring in its context, models and standards of improvement of organizational processes that aim to assure the excellence of the quality in the product generated.

Keywords: Quality, Metrics and Models.

LISTA DE FIGURAS

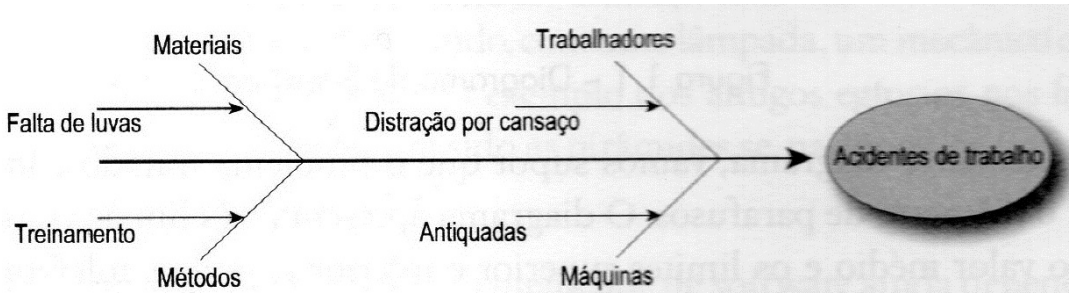
SUMÁRIO

1. INTRODUÇÃO

Os primeiros padrões de qualidade surgiram a mais de quatro mil anos no antigo Egito, com a medida padrão cúbito, que era baseada no tamanho do braço do Faraó reinante, e o controle era feito através de barras cortadas a cada lua cheia para assim manter a medida correta. No mundo moderno, com toda evolução da forma de se produzir e a mudança em seus processos se tornou inviável o controle individual das peças produzidas nas Fábricas, dessa forma em 1920 o controle de qualidade passou a ser estatístico, e temos em 1931, um dos primeiros trabalhos baseados no assunto com Walter Shewhart, *Economic Control of Quality of Manufactured Product*.

Já na década de 1940 surgiram alguns organismos para padronizar a qualidade, como por exemplo, ASQC (*American Society for Quality Control*) a ISO (*International Standardization Organization*) e a ABNT (Associação Brasileira de Normas Técnicas). Ainda na mesma década o Japão se destacou no assunto lançando o método Taguchi para projeto experimental e os diagramas de causa e efeito, conhecidos como diagrama espinha de peixe ilustrado na Figura 1.

Figura 1 - Diagrama Espinha de Peixe



Fonte: Qualidade de *Software* (2006)

Depois desse período vem-se aprimorando cada vez mais técnicas de qualidade com surgimento e manutenção de outras normatizações derivadas da ISO e também surgimento de normas próprias.

Seguindo a evolução das linhas de produção e dos processos industriais, outras áreas também tem se preocupado com a medição da

qualidade dos artefatos que produzem, buscando sempre entregar o melhor serviço ou produto a seus clientes.

No universo da informática fazer *software* de qualidade pode ser considerado um desafio antigo de muitas empresas enquanto que poder contar com um produto de qualidade é o desejo de muitos clientes, dessa forma hoje em dia a avaliação da qualidade é algo essencial no decorrer dos projetos todos os gestores de companhias e suas respectivas equipes, estão sempre visando melhorar a eficiência, diminuir a manutenção e o lançamento de versões de correção para seus produtos, o que é muito comum nos dias de hoje, para assim alcançar a excelência. A forma como essa qualidade é medida influência diretamente no resultado dos projetos, pois uma avaliação precária gera um produto de baixa qualidade e sem uma base confiável.

Esse projeto tem como objetivo após a conclusão auxiliar todos os profissionais da área que buscam encontrar novas métricas capaz de mostrar o quanto é eficiente o seu *software*, já que traz um novo conjunto de métricas previamente analisadas para tornar as avaliações de qualidade mais confiáveis e próximas da realidade.

1.1. Justificativa

O tema abordado neste trabalho é Qualidade de *Software*, uma área da Engenharia de *Software* que trata diretamente do aspecto final do *software*, ou seja, o resultado do termino do processo de desenvolvimento realizado pelas empresas.

A área de Qualidade de *Software* esta em evidência no cenário de desenvolvimento, devido a uma recente reformulação nas métricas e processos de desenvolvimento, conforme as normas ISO, seguidas em todo mundo para definir padrões de qualidade dentro de grandes companhias.

Segundo Koscianski & Soares (2006) as métricas utilizadas no processo de avaliação da Qualidade de *Software* são fator importante tanto antes da entrega final do produto como após o *software* pronto, pois dessa forma temos

uma coesão entre o que foi solicitado pelo cliente com um baixo índice de erros.

A evolução das tecnologias aumentou a exigência dos clientes, o que não permite que erros e deficiências nos *software*s sejam corrigidos após a entrega em futuras atualizações, isso também deve ser levado em consideração para avaliação das métricas de qualidade que serão utilizadas para definir se o produto entregue esta de acordo com o que foi solicitado e se atende os requisitos mínimos no que diz respeito as necessidades do solicitante.

1.2. Objetivos

O ambiente das empresas desenvolvedoras de *software* tem como uma de suas preocupações atuais a qualidade de seus produtos, a Engenharia de *Software* trata esse aspecto na Qualidade de *Software* e suas métricas, que definem o bom projeto como aquele com poucos erros e a máxima precisão com o que foi solicitado pelo cliente.

Observando esse aspecto da Engenharia de *Software* e a grande dificuldade das empresas em definir um alto padrão de qualidade, surgem muitas questões a serem levadas em consideração, como o processo de desenvolvimento, tipo de modelagem escolhida e por fim as métricas para avaliação da qualidade, que converge ao estudo que será realizado neste trabalho, abordando quais os caminhos devem ser seguidos para que a análise da qualidade do *software* seja feita de forma correta e precisa.

1.3. Objetivo geral

Essa pesquisa tem por objetivo ajudar na análise da Qualidade de *Software* de acordo com os projetos dos mesmos, realizando um apoio a escolha das métricas de qualidade tendo por base a complexidade dos projetos e as estratégias utilizadas no seu desenvolvimento, garantindo a entrega do melhor produto ao cliente.

1.4. Objetivos específicos

a) Realizar um levantamento bibliográfico sobre Qualidade de *Software*, suas métricas e processos que ajude a melhorar a análise final da qualidade dos produtos entregues aos clientes, diminuindo erros e defeitos.

b) Descrever as principais métricas e processos utilizadas visando esclarecer em quais projetos de *software* elas se encacham e em que momento elas devem ser definidas e utilizadas, identificando os mais eficientes para os diversos tipos de modelagem de sistemas aplicados pelas empresas de desenvolvimento.

2. METODOLOGIA

2.1. Modalidade de pesquisa

O projeto realizado busca minimizar a dificuldade das empresas em atingir uma qualidade aceitável em seus *software*s, já que os gastos com manutenção hoje muitas vezes supera o custo do projeto, ou se aproxima dele. Segundo Andrade (2009) devem ser aplicadas nas modalidades de pesquisa citadas abaixo.

De acordo com o ponto de vista da natureza do projeto a pesquisa será aplicada, a partir do conhecimento adquirido pretende-se alterar a forma de avaliação de qualidade de *software* e aplica-las em uma análise comparativa de dois processos.

Com relação à forma de abordagem do problema, será realizada uma pesquisa qualitativa já que busca avaliar a realidade, não baseando-se em dados estatísticos e sim no cotidiano do desenvolvimento de *software* nas empresas analisando os dados de forma indutiva.

Levando em conta os objetivos do projeto pretende-se realizar uma pesquisa exploratória, visando definir os melhores processos de desenvolvimento de *software*, usando mão de entrevistas com profissionais

envolvidos diretamente com o assunto, levantamento bibliográfico e uma aplicação dos resultados em uma comparação.

Os procedimentos técnicos terão base em uma pesquisa bibliográfica de materiais da área específica do projeto já publicados e analisados por outros autores, será feito uso de livros, artigos e outros projetos implementados com assunto similar.

2.2. Instrumentos de coletas de dados

Para coleta de dados será utilizado materiais da área especifica tais como Livros, artigos, revistas, palestras e seminários da área abordada no trabalho.

2.3. Critérios pra análise dos dados

Os dados serão analisados com base na aplicação de uma comparação que deve ser próxima da aplicação real modelos abordados.

2.4. Descrição das etapas de investigação

- Discussão do tema com orientador.

- Seleção de bibliografias pertinentes ao tema.

- Análise das leituras.

- Análise Comparativa entre dois modelos de Qualidade.

3. REVISÃO DE LITERATURA

3.1. QUALIDADE

O ser humano de forma implícita define padrões de qualidade para todas as coisas no meio em que está relacionado, sendo assim é natural que os usuários de sistemas também realizem tal análise, o que leva os desenvolvedores a ter uma preocupação com relação a qualidade daquilo que esta sendo desenvolvido, mas afinal o que é qualidade de *software*?

A qualidade de um projeto esta diretamente relacionada a sua complexidade ligada ao tamanho das especificações. Segundo Koscianski e Soares (2006), ao comparar a construção de um prédio de dez andares com a construção de uma residência, não pode-se levar em consideração apenas a diferenças na quantidade de tijolos. Em programas de computador também não deve-se ter essa ideia, pois por volta da década de 50, era comum o uso da lei de Grosch, onde o desempenho dos computadores seria proporcional ao quadrado do seu preço.

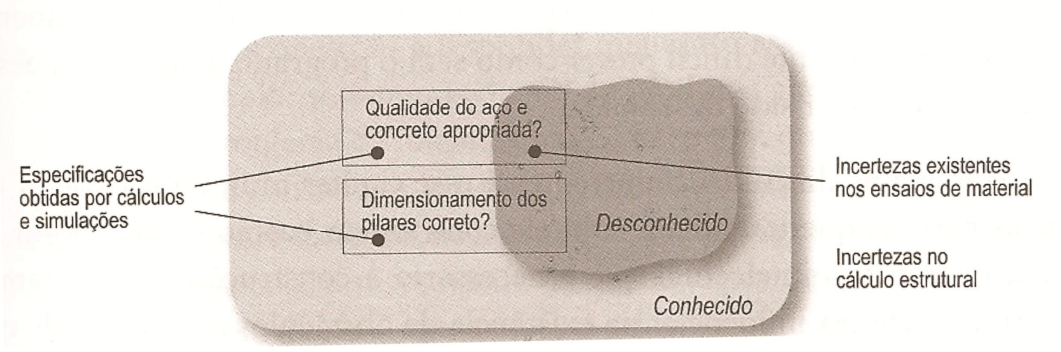
A evolução tecnológica subsequente a década de 1970, com a evolução das memórias e processadores, fez com que a produção de *software* se tornasse algo mais complexo assim como seus produtos.

Segundo Koscianski e Soares (2006), provavelmente o termo “Engenharia de *Software*” foi introduzido pela primeira vez em 1968 na Alemanha, em uma conferência do mesmo nome. Com a análise do relatório da conferência, encontram-se diversos problemas no desenvolvimento que persistem até hoje.

Um dos principais problemas de *software* conhecido foi bug do milênio, onde acreditava-se que aviões cairiam, saldos de contas bancárias desapareceriam e bombas seriam disparadas aleatoriamente. Mesmo com a ocorrência de problemas muito menores que os esperados, o bug do milênio leva a uma questão prática, a capacidade de produzir *software* de qualidade.

Ainda de acordo com os autores acima citados, o fato do desenvolvimento de *software* não ser uma atividade repetitiva, a torna imprevisível. Ao construir uma rodovia todo cálculo de estrutura, inclinação e terreno é realizado antes, prevendo as dificuldades, já com o *software* não temos cálculos precisos, como mostra a Figura 2:

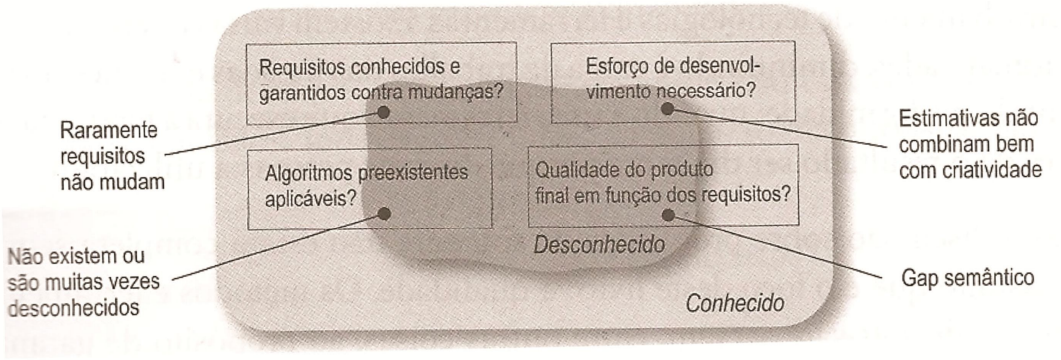
Figura 2 - Zonas de sombra de um projeto de Engenharia



Fonte: Qualidade de *Software* (2006).

Os fatores que influenciarão a rodovia depois de construída, pouco serão alterados. Já com o *software* os fatores que envolvem a área de sombra, ou seja, o desconhecido é muito mais abrangente, como mostra a Figura 3:

Figura 3 - Zonas de Sombra de um Projeto de *Software*



Fonte: Qualidade de *Software* (2006).

3.1.1. Gerenciamento de qualidade

Segundo Sommerville (2007), existe um aprimoramento significativo no que diz respeito a qualidade de *software* no últimos quinze anos, isso deve-se

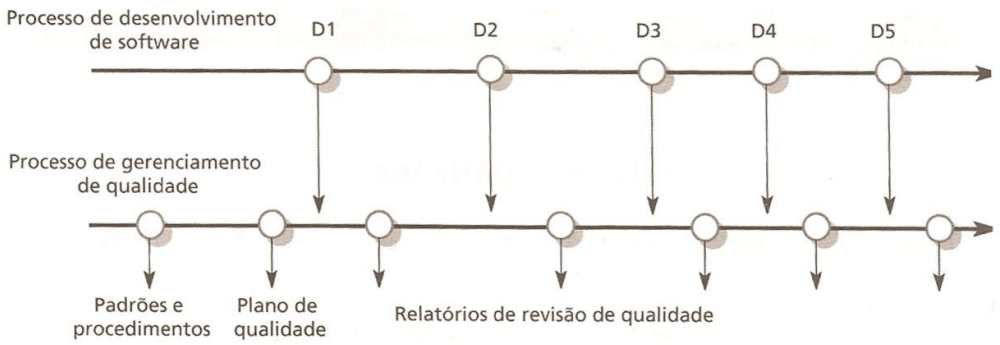
ao fato de as empresas estarem adotando novas técnicas e tecnologias no desenvolvimento de seus produtos. Para garantir a qualidade deve-se primeiro reconhecer os problemas com as especificações. Em grandes projetos o gerenciamento de qualidade é formalizado através de uma documentação que registra todas as informações do que é feito por todas as equipes. Já em projetos pequenos, pode-se utilizar uma comunicação informal, mas que todos os participantes do projeto tenham conhecimento e uma abordagem positiva sobre ela.

O gerenciamento de qualidade possui três atividades principais:

- Garantia de qualidade: estabelece um *framework* de todos os procedimentos e padrões que conduzem o *software*.
- Planejamento de qualidade: seleção dos padrões apropriados e adaptados do *framework*.
- Controle de qualidade: definição e aprovação de todos os processos que garante que a equipe seguiu os processos e padrões definidos.

Para que a qualidade seja garantida, o processo de gerenciamento deve ser independente do desenvolvimento e verificar todas as entregas realizadas. Isso fica exemplificado na Figura 4:

Figura 4 - Gerenciamento de Qualidade e Desenvolvimento de *Software*



Fonte: Engenharia de *Software* (2007)

3.1.1.1. Garantia da qualidade

De acordo Sommerville (2007), a garantia da qualidade é o processo que defini como a qualidade do *software* pode ser atingida e que qualidade foi atingida ao nível satisfatório. Para que a qualidade seja garantida devemos levar em consideração a ligação dos padrões dos processos e do produto. Os padrões relacionados ao produto são aplicados na saída do processo de *software*.

As importâncias dos padrões de *software* garantem que não exista repetição de erros, já que formam uma base de conhecimento que ajuda também a escolher as melhores práticas, que asseguram a utilização desses padrões e a continuidade, mesmo que ocorra troca na equipe.

Existem muitas instituições empenhadas em gerar padrões como a British Standards Institution Standard for IT Service Management, Motorola e Capability Mturity Model Integration, responsáveis pela criação de ISO's, Six Sigma e CMMI respectivamente. A equipe de garantia de qualidade deve basear-se especificamente em padrões como esses, para garantir o padrão dos processos e produtos da organização.

3.1.1.2. Planejamento de qualidade

O planejamento de qualidade segundo Sommerville (2007), é a definição de um plano de qualidade, que estabelece as qualidades desejadas e descreve como elas serão avaliadas. Esse planejamento evita a ambiguidade de informações e suposições conflitantes.

De acordo com Humphrey (1989) , uma estrutura geral para um plano de qualidade deve incluir:

- Apresentação do produto: descrição do produto, mercado a ser atingido e nível de qualidade.
- Planos de produtos: datas críticas, planos de serviço e responsabilidades.
- Descrição de processo: utilização de processos de desenvolvimento e de serviços.

- Metas de qualidade: identificação de atributos críticos de qualidade e metas e planos de qualidade.
- Riscos e gerenciamento de riscos: principais riscos que ameaçam a qualidade do produto e medidas para conter tais riscos.

Nenhum sistema pode ser otimizado para todo e qualquer atributo existente, pois há uma grande diversidade desses atributos, como facilidade de compreensão e complexidade, que são dois atributos de Qualidade de *Software*.

3.1.1.3. Controle de qualidade

Sommerville (2007), explica controle de qualidade como monitoramento do processo de desenvolvimento para garantir que os padrões e procedimentos estejam sendo seguidos. Existem duas abordagens diferentes que são usadas para verificar a qualidade dos produtos do projeto, revisão de qualidade e avaliação automatizada.

Revisão de qualidade é a realização de uma revisão por um grupo de pessoas, para verificar se os documentos e o *software* estão de acordo com o padrão definido. Os tipos mais comuns de revisão citados por Sommerville (2007) são Segurança, Proteção, Confiabilidade, Facilidade de recuperação e Robustez.

Avaliação automatizada é o processamento do *software* ou documentação sendo utilizado um programa para comparar e verificar se o padrão foi seguido de acordo com o definido, essa avaliação faz parte da revisão de qualidade.

3.2. IMPORTANCIA DA QUALIDADE DE *SOFTWARE*

A qualidade de *software* é algo muito importante, todo e qualquer erro gera prejuízos para quem o produz, pois ao desenvolver um *software*, é esperado que ele atenda os fins para os quais foi criado e que não exponha os dados que nele estão contidos. *Software* esta presente em uma lista infinda de

sistemas de diversas naturezas. Erros e mal funcionamento de *software*s já foram responsáveis por perdas de vidas humanas, prejuízos financeiros entre muitos outros problemas.

3.2.1. Ariane 501

O foguete Ariane 5 lançado em 1996 é um exemplo de falha que resultou em um prejuízo imenso, que na época foi avaliado em mais de 300 milhões de dólares segundo Koscianski e Soares (2006). O foguete saiu da rota, segundos após seu lançamento e explodiu em pleno voo há 3700 metros de altitude. O sistema responsável pelo controle do foguete possuía dois computadores de *backup* além do principal, para caso houvesse falhas no o *backup* era acionado, porem no calculo da rota houve uma conversão de dados 64 bits em 16 bits e o erro não foi capturado por nenhuma exceção do *software* responsável, e no *backup* também não, pois era idêntico ao principal. Diversas conversões dessa natureza foram realizadas e não foram tratadas, com isso o foguete saiu da rota e não cortou o ar, entrando em atrito e acabou explodiu.

De acordo com Koscianski e Soares (2006) a falha que ocasionou o acidente poderia ter sido solucionado com um comando if , e evidencia os principais problemas relacionados a qualidade de *software*:

- O caractere dominante dos requisitos sobre os resultados.
- Dificuldade de garantir que requisitos sejam consistentes em projetos.
- Testes não garantem ausência de erros.
- Dificuldade de verificar e validar programas.

3.2.2. Therac 25

Koscianski e Soares (2006) citam em seu livro outro caso envolvendo falha na qualidade de *software*, que acabou resultando em mortes. O Therac 25 foi uma maquina projetada para auxiliar no tratamento de pacientes que deviam se submeter a sessões de terapia radiológica, esse aparelho foi uma evolução de versões anteriores, com funcionalidades de controle mecânicas, para uma versão totalmente controlada via *software*, em um computador

chamado PDP-11. A nova versão não possuía travas mecânicas de segurança que visavam evitar erros de operação, essas travas ficaram por conta do PDP-11, que acabou falhando por erros de projeto e resultou na morte de pacientes.

Os autores relatam que durante a operação do equipamento ele apresentava mensagens de erro nada claras limitando-se por vezes a palavra malfunction seguida de um numero. As falhas apesar de comuns, não preocupava os operadores, já que haviam recebido a informação de que o equipamento possuía varias travas de segurança. Um incidente fatal com esse aparelho ocorreu em Ontário, Canadá, onde um operador recebeu uma mensagem de erro descrita como htilt error com indicações de no dose que informava que nenhuma radiação havia sido emitida, com isso o operador reiniciou o procedimento e por cinco vezes ocorreu o mesmo problema, até que o Therac-25 teve de ser reiniciado. Na autópsia da paciente que estava na maquina nesse incidente foi detectado que seu fêmur foi quase todo destruído pelo excesso de radiação.

Após o incidente o médico Frank Borger da Universidade de Chicago fez diversos testes no Therac -20 e constatou que dependendo dos parâmetros de edição que os operadores inseriam na maquina, alguns fusíveis paravam de funcionar, sendo assim avaliou os parâmetros no Therac-25 e detectou erros no projeto, tanto em código como mecânicas. Borger demonstrou em seus testes uma forma muito eficaz de isolar um defeito de *software*, algo que devia ter sido observado pelos responsáveis por projetos na AECL(Atomic Energy of Canada Limited) empresa canadense responsável pela produção do Therac-25.

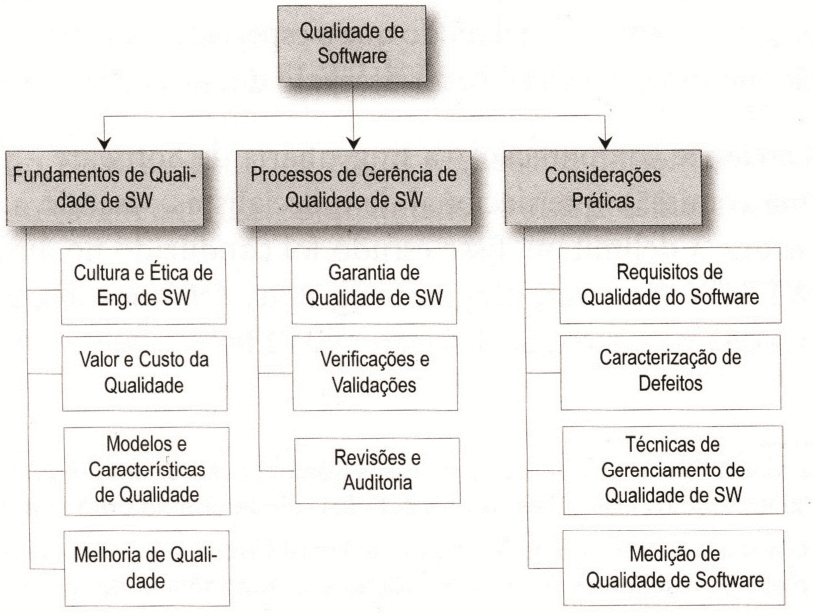
3.2.3. Swebok

O SWEBOK segundo Koscianski e Soares (2006), é o resultado de um estudo realizado por uma comissão internacional, para definir os limites da Engenharia de *Software* dentro das áreas existentes na computação, o credito desse estudo é dado a IEEE, e o *Software* Engineering Body of Knowledge (SWEBOK) data o ano de 2004.

O SWEBOK possui onze seções e uma dessas onze é exclusiva para tratar sobre o assunto Qualidade de *Software*, o que evidência mais ainda sua importância dentro da Engenharia de *Software*. Ele possui uma diferenciação entre as técnicas existentes no gerenciamento de qualidade, tanto em técnicas dinâmicas e estatísticas, ressaltando também a importância dos testes de *software* de acordo com Koscianski e Soares (2006).

A Figura 5 demonstra a organização em tópicos apresentada na seção de Qualidade de *Software* do SWEBOK.

Figura 5 - Tópicos da seção de Qualidade de *Software* do SWEBOK



Fonte: Qualidade de *Software* (2006)

3.3. METRICAS DE QUALIDADE DE *SOFTWARE*

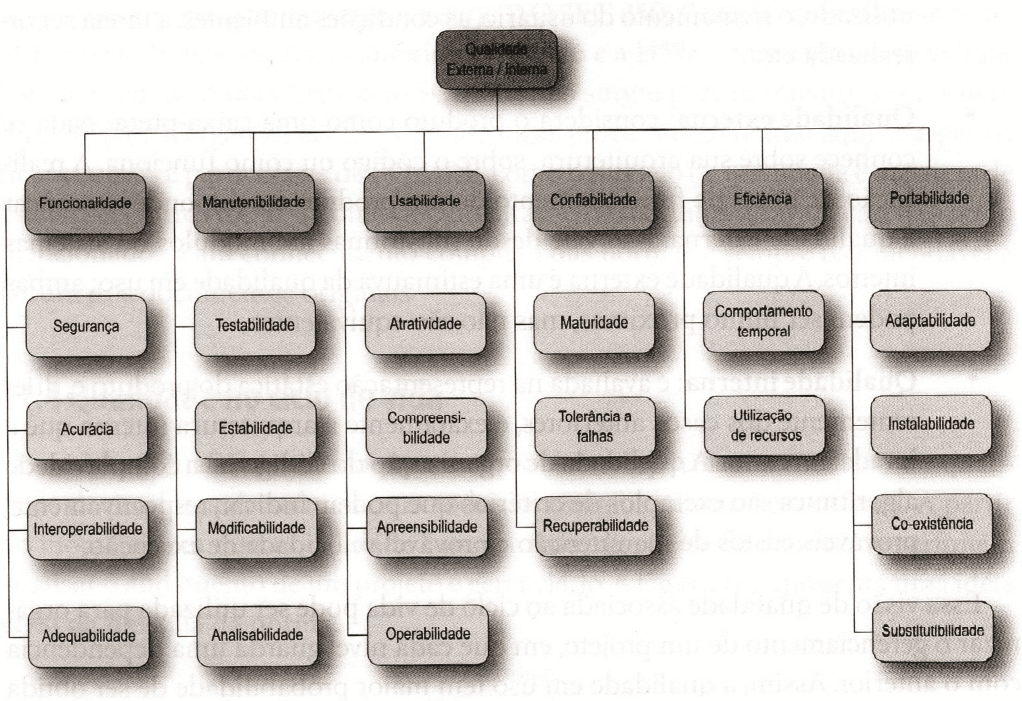
Assim como qualquer produto comercializado hoje em dia, *software* também tem um controle de qualidade pelo qual é submetido e pelo qual deve ser aprovado, esse controle é realizado através de métricas definidas na Engenharia de *Software*, que são tratadas nesse capitulo no que diz respeito a sua definição e aplicabilidade.

Segundo Koscianski e Soares (2006), previsões realizadas a partir de cálculos e simulações geram resultados em medições relevantes para determinar o resultado final de um produto.

> [...] Em *software*, as medições podem ser utilizadas de maneira semelhante. Antes mesmo de o produto existir , determina-se durante a análise de requisitos como deverá funcionar. Pode-se, por exemplo, estabelecer qual o tempo máximo que o programa poderá demorar para fornecer uma certa resposta.[...]

Ainda baseado em Koscianski e Soares (2006), levando em consideração essa ultima citação, as métricas ajudam na redução de influência de fatores subjetivos, no momento de definir a qualidade de um *software* especifico. Se os critérios forem baseados e determinados por uma equipe, eles auxiliam no que diz respeito à pressão exercida por prazos e entregas. As métricas tratadas nesse capitulo do estudo são baseadas no modelo SQuaRE, que por sua vez é baseado na evolução da ISO 9126 e também da ISO 14598. A Figura 6 mostra o esquema desse modelo e as principais métricas que serão abordadas.

Figura 6 - Esquema de métricas da ISO 9126



Fonte: Qualidade de *Software* (2006).

Seguindo os conceitos abordados por Koscianski e Soares (2006), a Figura 6 esta demonstrando a estrutura hierárquica do modelo de qualidade, indicando as subcaracterísticas de cada métrica principal. A elaboração das subcaracterísticas foi feita a fim de evitar a sobreposição entre as métricas, ou seja, de modo que não exista nenhuma interferência de confiabilidade ao avaliar, por exemplo, funcionalidade. Cada subcaracterística é definida por atributos que o *software* venha ou não a possuir, variando entre os produtos.

Através dessa hierarquia definida no modelo, podemos traçar um plano de requisitos de qualidade de acordo com o que o cliente deseja, levando em consideração os melhores aspectos de avaliação.

3.3.1. Funcionalidade

Funcionalidade e acordo com Koscianski e Soares (2006) consiste no que é realizado no momento em que o usuário requisita algo ao *software*, e se esta de acordo com o que foi solicitado. É plausível afirmar que requisitos

funcionais sejam atributos de funcionalidade, pois representam aquilo que se é esperado do *software*. O modelo no qual este capitulo esta baseado determina que se defina um escopo para estabelecimento de todas as funcionalidades do *software*, garantindo que seu requisitos estejam coerentes, como no caso de gravar dados em um banco, deve-se garantir que a conexão com o mesmo esteja estabelecida e que o disco esteja liberado para gravação.

Dentro das subcaracterísticas Koscianski e Soares (2006) definem adequabilidade como quanto o *software* esta adaptável a rotina do cliente, se caso ele precisar de um procedimento padrão como cadastro, porém de forma mais rápida, o *software* irá atende-lo ? a resposta de perguntas como essa definem a adequabilidade do produto.

Já a acurácia esta presente principalmente em *software*s que geram resultados através de cálculos, e é determinada por exemplo no numero de casas decimais que terá, que impacta diretamente na precisão.

Interoperabilidade é a característica definida pela capacidade de operar com outros componentes presentes em sistemas complexos, esses componentes podem ser hardwares ou outros *software*s.

A subcaracterística de segurança, visa definir quão seguro é o *software* contra acessos não permitidos, e se os dados estão realmente seguros enquanto são operados.

3.3.2. Manutenibilidade

Segundo Koscianski e Soares (2006) a possibilidade de alteração e adaptação do *software* depois de pronto, é a representação da característica de manutenibilidade. Ela é fundamental para programadores, já que não diz respeito à parametrização e sim a modificações no código para possíveis mudanças na regra de negócio devido a legislação por exemplo. Essa capacidade na alteração do código implica na subcaracterística de modificabilidade que tem como fatores importantes a documentação e estrutura interna, tipo de arquitetura e código fonte claro.

A analisabilidade mostra a facilidade ou falta dela, no momento de encontrar falhas ou defeitos que geram imperfeições no funcionamento. A estabilidade é a garantia de que não haverão comportamentos errôneos e inesperados após uma alteração e essa característica é completada através da testabilidade, que é a possibilidade de validação das alterações, uma vez realizadas no produto.

3.3.3. Usabilidade

Koscianski e Soares (2006) definem usabilidade como uma das mais complexas características a ser definida, pois possui um alto grau de abstração no momento de estabelecer requisitos, e a interface com usuário é indispensável. Essa característica possui uma influência de fatores externos muito diferente das outras, nesse caso até mesmo a capacidade motora e conhecimento de informática por parte de quem testa pode mudar o resultado, gerando disparidade na avaliação final do produto.

Todas as subcaracterísticas de usabilidade dependem de habilidades especificas de quem as testa, a operabilidade que define a possibilidade de controle das operações por parte do usuário é uma delas. A operabilidade deve medir se as operações existentes podem ser acessadas e se elas permitem retornar e acessar outras, um *software* não pode ter “ruas sem saída”, porém existe por vezes deficiências na compreensão do usuário aos símbolos presentes na aplicação que significam atalhos para que esse retorno seja possível.

A compreensibilidade é medida através das entradas passadas pelo usuário e pelos dados retornados por ele. A entrada de dados deve ser feita através da interface e os resultados serão mostrados também por essa mesma interface, a forma como isso acontece também é um aspecto importante para essa subcaracterística. Os textos apresentados, sequenciamento de tarefas interdependentes são outros fatores de importantes.

Koscianski e Soares (2006) relatam que apreensibilidade mensura se o *software* ajuda o usuário a aprender a opera-lo, se existem ícones intuitivos, por exemplo, isso facilita o treinamento.

A atratividade gera uma certa discórdia por parte dos comitês de engenharia de *software* segundo Koscianski e Soares (2006), porém não deve ser confundida como medida de beleza do *software*, na verdade diz respeito a capacidade de prender a atenção do usuário de fazer com que ele se sinta bem operando o sistema, para isso são utilizadas técnicas psicológicas cognitivas e estatísticas.

3.3.4. Confiabilidade

Confiabilidade é algo muito desejado por quem compra um *software*, porém como é possível evitar falhas na execução de um programa? A característica de confiabilidade possui duas subcaracterísticas que respondem a pergunta seguindo os estudos de Koscianski e Soares (2006). A tolerância deve avaliar a capacidade do *software* continuar funcionando mesmo que algo inesperado ocorra, por exemplo falhas de execução por falta de espaço em disco.

Outra subcaracterística que deve ajudar no aumento da confiabilidade é a recuperabilidade, que representa a capacidade do *software* retomar sua execução de onde parou, como faz o editor de texto da Microsoft, o Word guarda versões do arquivo que esta sendo editado e se é interrompido abruptamente, ao ser reiniciado da opção de retomar a versão que havia salvo.

A maturidade é a característica que mostra a robustez do *software*, a capacidade de trabalhar com tratativa de falhas, e controle de exceções, incluídas através de resultados obtidos com testes de erros improváveis. Essa subcaracterística esta presente em *software*s maduros e utilizados por uma grande variedade de usuários.

3.3.5. Eficiência

Koscianski e Soares (2006) relatam a importância de executar as medições sobre essa característica em um ambiente controlado, pois os fatores desse ambiente influenciam diretamente no resultado, mais que nas outras características. É um desafio controlar completamente o ambiente de execução do *software*, porem se faz necessário já que é afetada por fatores como

quantidade de memória e velocidade da CPU, que são altamente voláteis. Suas subcaracterísticas se confundem no ambiente de medição já que comportamento temporal avalia a eficiência durante determinado tempo e a utilização de recursos mede exatamente essa utilização durante a medição, se existe, por exemplo, uma grande variação de uso de memória, e em que momento foi mais exigido da CPU. Faz parte do desafio da medição dessa característica a definição de valores máximos e mínimos para os recursos disponibilizados, já que em um ambiente mesmo que controlado uma medida como “memória cheia” pode vir a depender do trafego de informações, e uma variação desse parâmetro leva a disparidade nos resultados.

3.3.6. Portabilidade

Para o modelo estudado, a portabilidade segue uma ideia diferente do habitual, que é a capacidade do código fonte operar em diferentes plataformas, esse conceito foi adaptado para ambientes organizacionais, e essa característica deve medir se o *software* se adapta ou não a esses ambientes.

A subcaracterística de instalabilidade diz respeito a como o *software* reage no processo de instalação e configuração inicial, onde são definidos diversos parâmetros importantes para o uso da aplicação.

A adaptabilidade verifica a capacidade de operar mudanças em parâmetros embutidos do *software* no momento em que ele é transferido de ambiente, o produto deve mostrar-se fácil de configurar sem a necessidade de um técnico intervir deixando isso a cargo do usuário final.

Coexistência pode ser definida como a capacidade de existir com outras aplicações, sem que haja troca de dados entre elas, segundo Koscianski e Soares (2006) a única coisa que se compartilha na coexistência são recursos computacionais no caso de *software*s.

A subcaracterística de substitutibilidade indica a capacidade do *software* ser utilizado no lugar de um outro similar, em um mesmo ambiente cumprindo função idêntica ou semelhante.

3.4. PROCESSOS DE QUALIDADE DE *SOFTWARE*

Existem diversas regulamentações e procedimentos para garantir a melhor aproveitamento dos processos das empresas, otimizar a produção, elevar o lucro, e deixar o cliente satisfeito com um produto ou serviço de qualidade e para *software* o crescimento é significativo nos últimos anos de acordo com Fernandes e Abreu (2012, p. 313)

> [...] Provavelmente os processos de *software* sejam aqueles para os quais mais modelos de melhores práticas foram desenvolvidos ao longo dos anos. Entre esses modelos figuram técnicas de engenharia de *software*, metodologias e padrões para as várias etapas do desenvolvimento, adaptações de métodos de gerenciamento de projetos etc.[...]

Temos muitos *Framework*s, Normas e Modelos que determinam como as empresas devem proceder para alcançar o melhor resultado de seu ramo de negócio. Alguns exemplos são o Modelo CMMI, o *Framework* Solutions da Microsoft e as Normas ISO/IEC 12207 e 9126 que serão tratados a seguir.

3.4.1. Modelo CMMI

Em 1991 foi criado o SW-CMM (Capability Maturity Model for *Software*) pelo *Software* Engineering Institute (SEI) para ser usado modelo de qualidade no processo de *software*. Muitas empresas começara a utilizar o modelo e aplicar variações do mesmo para atender suas necessidades de acordo com aquisição de *software*, engenharia de sistemas e desenvolvimento integrado de produtos e processos. Com as variações os modelos acabaram que gerando uma implementação e arquitetura próprias, dificultando a integração com os processos já existentes nas organizações.

Em 2002 para solucionar esse problema o SEI, criou o CMMI (Capability Maturity Model Integration), de acordo com Fernandes e Abreu (2012) o objetivo do modelo é “combinar suas varias disciplinas em uma estrutura única, flexível e componentizada que pudesse ser utilizada de forma integrada por organizações que demandavam processos de melhoria em âmbito corporativo”

fazendo dessa forma alterações em conceitos até então confundidos, como empresa e organização, e também uma real valorização nos processos de verificação e validação aplicados pelo modelo.

Houve em 2006 o lançamento da versão 1.2 do CMMI, relatada por Fernandes e Abreu (2012) como um processo de melhora e simplificação, no que diz respeito a unificação dodas disciplinas de engenharia de sistemas e *software* e a integração no desenvolvimento de produto e processo, o modelo teve ainda uma alteração em sua arquitetura, para possibilitar a expansão para novos focos, como por exemplo o de serviços. Em 2010 foi lançada a versão 1.3, onde ocorreu um refinamento de processos, pra que haja uma reflexão sobre outros modelos e melhores práticas de mercado, alguns modelos refletidos foram os ágeis e o Six Sigma, A principal mudança porem foi a eliminação de práticas genéricas.

Como havia citado, os autores definem o modelo como uma proposta de integrar as diversas variações que ocorreram do CMM, sendo assim Fernandes e Abreu (2012) reforçam essa ideia.

> [...] O principal propósito do CMMI é fornecer diretrizes baseadas em melhores práticas para melhoria dos processos e habilidades organizacionais, cobrindo o ciclo de vida de produtos e serviços completos, nas fases de concepção, desenvolvimento, aquisição, entrega, e manutenção. [...]

3.4.1.1. Visão geral do modelo

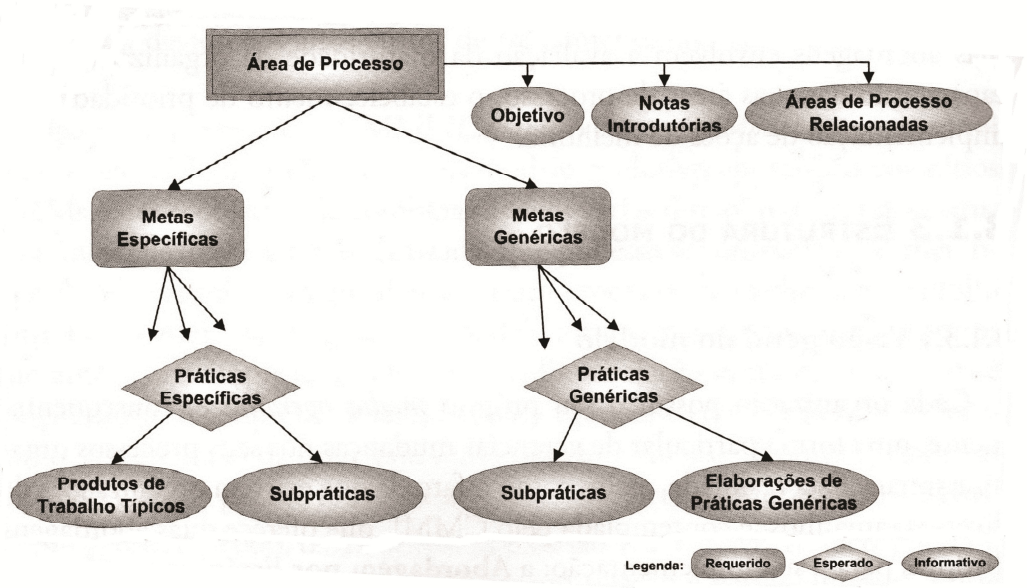
Existem organizações com diferentes formas de operar seus processos e com tamanho de operação completamente distintos, de acordo com seus clientes e ramo particular de desenvolvimento, pra essa realidade atual e tendo como base Fernandes e Abreu (2012), temos diferentes formas de implementação do CMMI, tratadas como abordagens, a de Estágios e a Continua, na versão 1.3 o modelo traz as duas implementações em um mesmo documento dividido de acordo com escopo de cada constelação. Cada constelação consiste em um conjunto de componentes que provem do *framework* CMMI, cada uma delas possui um modelo fundamental, materiais de

treinamento e a documentação de avaliação, essas constelações podem sofrer adições que consistem na expansão do modelo e a seguir estão listadas as três constelações que fazem parte da versão 1.3.

- CMMI-DEV: para desenvolvimento com diretrizes para mensurar e gerenciar processos.
- CMMI-SVC: para gerenciamento de entrega de serviços.
- CMMI-ACQ: para gerenciamento de aquisições de produtos e serviços.

A Figura 7 mostra os componentes principais da estrutura do CMMI-DEV 1.3

Figura 7 - Componentes do escopo do CMMI-DEV



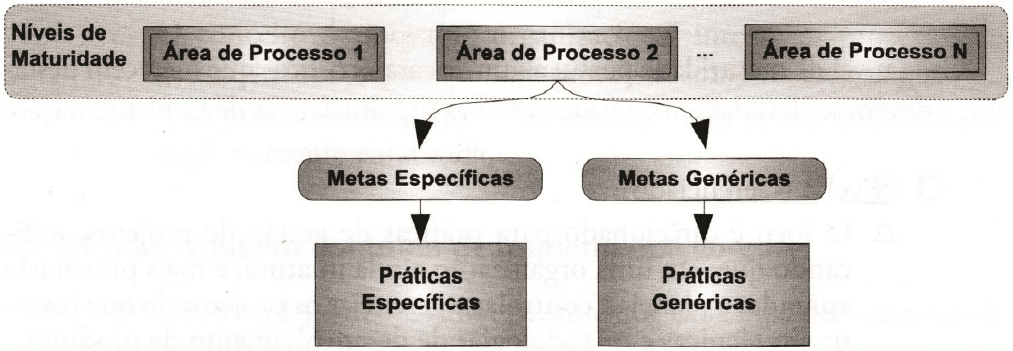
Fonte: Implantando a Governança de TI (2012).

O CMMI-DEV que é voltado diretamente para o desenvolvimento de *software*, possui alguns componentes importantes como, as Áreas de Processos, que formam um conjunto inter-relacionado de práticas, que visam melhorar uma área especifica onde são aplicadas, áreas essas que possuem Metas Especificas que também são componentes importantes do CMMI-DEV.

3.4.1.2. Implementação por estágios

Segundo Fernandes e Abreu (2012), a abordagem por estágios, nasceu de uma evolução direta do CMM, já que conta com cinco níveis de maturidade, que podem ser usados como degraus diretos para alcançar melhorias em áreas distintas no processo de desenvolvimento de *software*, levando em consideração suas metas especificas e genéricas. Essa abordagem inter-relaciona os componentes do modelo como mostra a Figura 8.

Figura 8 - CMMI abordagem por estágios



Fonte: Implantando a Governança de TI (2012).

Os níveis de maturidade são definidos a seguir, de acordo com Fernandes e Abreu (2012).

- Nível Inicial: Nível que determina as necessidades e situação atual da organização.
- Nível Gerenciado: Esse nível determina o foco para o aprendizado do planejamento e não para desenvolvimento de produtos, já que a organização ainda se mostra imatura. O gerenciamento durante o andamento dos projetos garante a qualidade dos produtos gerados, e aderência aos processos formalizados com outras organizações, e existe nesse nível a visível preocupação no nascimento de uma infraestrutura para análise de processos e resultados.
- Nível Definido: O foco é transferido para uma engenharia de produtos, que considera o ciclo de vida básico, Concepção,

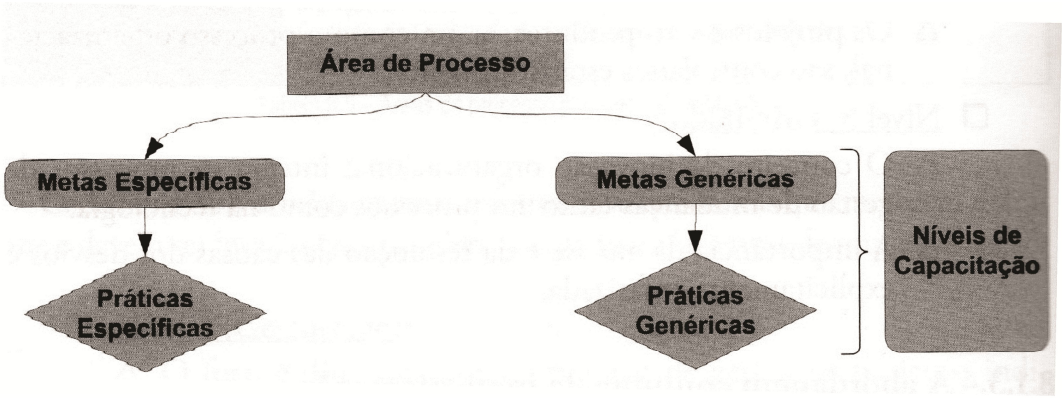
Análise e Desenho, Testes e Implantação. Esse nível trabalha com a integração de equipes no ambiente organizacional e estimula práticas que proporcionam uma preocupação com riscos e tomada de decisão, para bem gerencia-los.

- Nível Gerenciado Quantitativamente: Auxilia o processo de desempenho e qualidade com medições e indicadores dos projetos e produtos gerados através desses projetos, tendo um controle estatístico para ambos.
- Nível Otimizado: Trabalha com o foco nos processos e tecnologia para prover inovação e integração dos processos de gestão de mudanças.

3.4.1.3. Implementação continua

De acordo com Fernandes e Abreu (2012), essa abordagem trabalha com níveis de capacitação e tem como característica a evolução de suas áreas de forma independente. Nessa abordagem as práticas genéricas relacionadas possuem uma meta genérica para descrever o grau da organização, e também inter-relaciona os componentes do CMMI-DEV como mostra Figura 9.

Figura 9 - CMMI abordagem contínua



Fonte: Implantando a Governança de TI (2012).

Dentro do programa de melhorias do CMMI, a organização pode através de perfis e níveis de capacitação acompanhar sua evolução, já que cada área de processo esta em um nível. Existe a comparação dos perfis gerados ao

decorrer do tempo no programa de melhorias com um perfil base, que deve ser atualizado com o passar do tempo, para garantir a evolução continua do processo em questão.

3.4.1.4. Aplicabilidade do CMMI

O modelo CMMI pode ser aplicado em qualquer organização que tenha como foco de desenvolvimento produtos, tanto de hardware, *software* e sistemas em geral como publicado por Fernandes e Abreu (2012).

Suas duas abordagens são para diferentes tipos de organização, a abordagem por estágios é recomendada para empresas que já possuem algum nível de maturidade de CMM ou CMMI, e que já estão de certa forma habituadas a melhorias continuas em processos que tem relação direta com qualidade de seus produtos. Já a abordagem continua é recomendada para as organizações que pretendem integrar graus de capacitação com processo a processo existente, o que possibilita diluir o valor do investimento ao longo do processo de evolução. Mas a abordagem continua exige mais esforços por parte da organização que a abordagem por estágios, o que pode dificultar sua implementação em organizações pequenas.

3.4.1.5. Benefícios do modelo CMMI

O CMMI, aplicado por diversas organizações mostrou uma lista de benefícios atingidos em diferentes categorias como, custo, prazo, produtividade, qualidade, satisfação dos clientes e retorno sobre investimento. De acordo com dados da SEI 2005, vinte e cinco grandes empresas relataram melhoria de no mínimo 50% na qualidade de seus produtos, com metas atingidas, como por exemplo, no quesito de defeitos foram encontrados apenas mais ou menos cinco defeitos a cada mil linhas de código e desses defeitos apenas 2%, encontrados no *software* em produção e uma redução de 7% no pedido de mudanças na versão também em produção.

3.4.2. Modelo MR-MPS

Fernandes e Abreu (2012) abordam a questão do MR-MPS como uma evolução na padronização da qualidade de *software* brasileiro, no século XXI

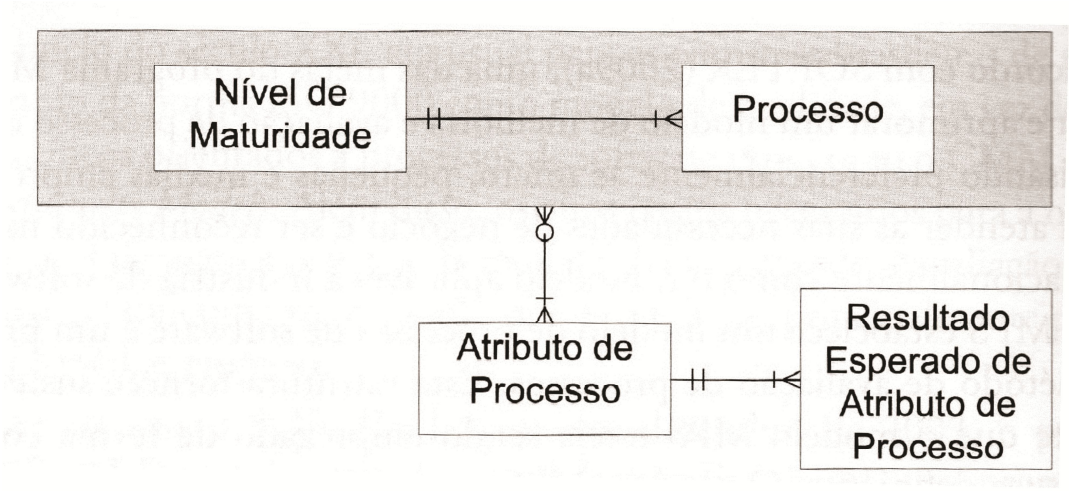
era muito usado para essa padronização a ISO 9000, que foge do foco de desenvolvimento de *software*, e possui um alto custo ao se buscar um nível de maturidade elevado. Pensando nesse cenário a SOFTEX, uma organização de interesse publico que busca tornar a indústria de *software* brasileira mis competitiva, desenvolveu o MPS.BR, um programa que busca melhorar o processo de *software* brasileiro, esse programa, tem uma proposta chamada MR-MPS, que se baseia em outros modelos como CMMI-DEV e ISO 15504, e busca melhorar as práticas de engenharia de *software*. Esse modelo tem prioridade de chegar a todas as organizações do país envolvidas em seu contexto, com prazo e custo razoável, atingindo desde grandes até pequenas empresas. No período de 2004 a 2007 foram lançadas três versões do modelo MR-MPS, a 1.0, 1.1 e 1.2 que entrou em processo de consolidação de 2008 a 2011.

Com relação o objetivo do modelo, ele tem em sua proposta principal, aprimorar modelos e avaliação de processos de *software*, atingindo principalmente micro, pequenas e medias empresas, e através dos modelos e processos atender a necessidade de negócio das organizações e garantindo que o modelo seja empregado de forma correta com o que diz respeito a sua filosofia.

3.4.2.1. Estrutura do MR-MPS

O modelo esta dividido em três componentes principais, Modelo de Referencia, que possui guias específicos para conhecimento e preparação das organizações que desejam implementar o modelo, Modelo de Avaliação, contendo os requisitos dos avaliadores, e Modelo de Negócio, que contem a regra de negócio geral do modelo no que diz respeito a implementação, avaliação e treinamento. Os componentes do modelo são ilustrados na Figura 10.

Figura 10 - Componentes da estrutura do modelo MR-MPS



Fonte: Implantando a Governança de TI (2012).

Assim como o CMMI, o modelo MR-MPS possui níveis de maturidade, que são relacionados a um conjunto de processos e cada processo desses possui um conjunto de atributos também relacionados. O MR-MPS é dividido em sete níveis definidos por letras de A à G, onde G é o mais básico e A o mais elaborado, de acordo com Fernandes e Abreu (2012).

- Nível G: Este nível, chamado de Parcialmente Gerenciado representa o foco no gerenciamento e projetos e mantém um plano de atividades e utilização de recursos, bem como gerência de requisitos ao adquirir novos trabalhos.
- Nível F: Este nível considerado Gerenciado trabalha com a aquisição de produtos e serviços, preocupa-se em manter íntegros todos os produtos de trabalho, inicia e planeja projetos que sejam necessários, sustentáveis e suficientes, além de garantir que a qualidade de todos os processos que envolvam os produtos de trabalho seja alta.
- Nível E: Denominado Parcialmente Definido, esse nível foca nos processos organizacionais e na contribuição dos mesmos para busca dos objetivos de negócio, auxilia na manutenção de processos organizacionais definidos e garante recursos humanos aos projetos e a organização.

- Nível D: Este nível é definido como Largamente Definido dentro do modelo MR-MPS, existe um foco no desenvolvimento de requisitos de cliente e produto, procura transformar os componentes de produto em algo mais consistente, constrói valida e verifica o produto para que haja uma coerência com relação aos requisitos.
- Nível C: O nível C é o Definido, que tem uma gerência voltada para decisões criticas baseadas em um processo formal, trabalha na reutilização de ativos da organização, buscando implantar um programa que garanta tal reutilização e através da gerência de riscos, proporciona uma identificação, tratativa e redução, de riscos.
- Nível B: Este processo está identificado como Gerenciado Quantitativamente, e a alta maturidade da organização reflete diretamente no gerenciamento de projetos de forma quantitativa, que mostra que todos os processos estão correspondendo com a medição e controle.
- Nível A: Quando atinge nível A, a organização chega ao nível otimizado, e todo e qualquer processo dentro da organização deve ser coerente com os processos de medição e controle.

3.4.2.2. Aplicabilidade do MR-MPS

De acordo com Fernandes e Abreu (2012), o MR-MPS pode ser implementado em qualquer organização, independente de tamanho, mesmo que trabalhem com operação contínua, como Fábrica de *Software* ou Teste. A formatação do modelo, definida através de seus sete níveis de maturidade, permite que ele seja implantado com menor prazo e gradualmente, trazendo uma vantagem para pequenas e medias empresas que é um diferencial na hora de fechar contratos e prestar serviços de desenvolvimento. Dentro do “universo” do programa MPS.BR, existem empresas que auxiliam na implementação do modelo através de consultoria, chamadas Instituições Implementadoras, e também Instituições Avaliadoras, que fazem um trabalho

de avaliar as organizações que possuem o modelo implementado, sob método credenciado MA-MPS.

3.4.2.3. Benefícios do MR-MPS

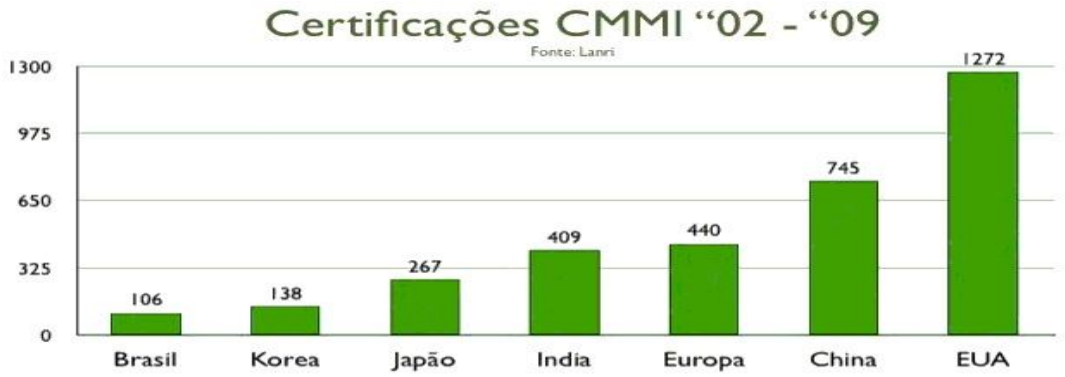
Pesquisa realizada pela SOFTEX aponta uma satisfação de mais de 92% por parte das empresas que adotaram o modelo, que se mostrou útil na gerencia de projetos maiores, satisfação dos clientes com o que é entregue e prazo estimado mais coerentemente. Ouve ainda índices de maior aplicabilidade de conceitos de Engenharia de *Software*, se levado em conta pontos como prazo, custo e qualidade.

3.4.3. CMMI X MR-MPS

Ambos os modelos são baseados nas normas ISO/IEC 12207 e ISO/IEC 15504 e são compatíveis em alguns de seus níveis de maturidade, de acordo com Fernandes e Abreu (2012), o que facilita a evolução de um modelo para outro. Hoje o que pode diferir bastante um do outro é o fato de que o CMMI tem um custo de implementação maior e sua evolução não é tão horizontal como MR-MPS, já que possui níveis a menos de maturidade e demanda mais tempo para aplica-los.

Por possuir mais níveis de maturidade, o MR-MPS permite que as empresas avancem de forma vertical, ou seja, rápida em suas mudanças procedurais com menor esforço e custo, isso beneficia as empresas de menor porte como as pequenas e médias, dessa forma popularizou-se o modelo como voltado para empresas desse porte Mas na verdade a sua proposta é para todo tipo de organização assim como CMMI, que nesse cenário acabou assumindo o papel de modelo das grandes organizações e hoje é largamente utilizado no Brasil, em empresas como Itaú, HP, Stefanini entre outras. A Figura 11 mostra o crescimento gradual das avaliações do CMMI no período de 2002 à 2009.

Figura 11 - Avanço das avaliações do CMMI no decorrer dos anos



Fonte: http://www.blogcmmi.com.br

Em pesquisas disponíveis no site da SOFTEX, responsável pelo modelo MR-MPS, existem diversas listas de medias e pequenas empresas que obtiveram o selo MR-MPS, e estão aguardando atingir um grau de maturidade mais elevado para migrar seus processos para o modelo CMMI. Essa tendência comprova a proximidade e eficácia dos modelos, quando utilizados de forma evolutiva.

Apesar das diferenças os modelos são complementares quando tratamos da realidade das empresas de *software* em âmbito nacional, e o fato de terem de certa forma a mesma origem faz com que empresas que optem por um ou outro tenham um resultado final semelhante, apenas com custo e tempo diferente, sendo assim a escolha do melhor modelo se torna uma estratégia financeira com retorno garantido a longo prazo.

4. CONCLUSÃO

Foi dito em todo esse estudo que *software* esta se tornando um produto muito comercializado nos últimos anos, e como todo produto deve ter sua qualidade avaliada. Essa preocupação com a qualidade é algo antigo e remonta civilizações de outras épocas, como os antigos egípcios citados anteriormente, e pode-se afirmar que no decorrer dos séculos ficou embutido em nós de forma implícita um julgamento que nos leva a sempre medir a qualidade daquilo que adquirimos.

O mundo organizacional não se difere do mundo onde o cidadão comum esta inserido no quesito julgar a qualidade daquilo que se procura obter, é obvio que ao adquirir uma maquina espera-se que ela dure por muito tempo sem precisar de reparos e que faça aquilo para o que é adquirida, se é uma maquina de pintar portas automotivas, deve pintar portas automotivas e faze-lo corretamente. De alguns anos pra cá se ouve muito sobre *software* de gerenciamento de diferentes áreas, financeira, operacional, vendas, contábil entre muitas outras. O crescimento das áreas geridas por *software* criou um ambiente favorável para empresas de desenvolvimento que cresceram na mesma proporção e hoje depositam seus produtos aos montes no mercado. Isso deixa de certa forma as organizações interessadas nesses *software*s um tanto perdidas com relação a qual escolher.

Ao tratar as métricas de qualidade, se evidenciou para pessoas e organizações algumas das características que um bom *software* deve possuir, e como o modelo SQuaRE deixa de forma clara alguns métodos que devem ser utilizados pelas organizações na hora da avaliação e comparação entre dois *software*s.

Nesse estudo ao mostrar como funcionam os processos de melhoria da qualidade dos projetos de *software*, abordando os modelos CMMI e MR-MPS buscou-se esclarecer para os produtores de *software* a importância de criar esses processos e segui-los mostrando que o resultado final é realmente satisfatório quando se obtém determinado grau de maturidade, com o comparativo realizada foi evidenciado que o modelo MR-MPS é uma boa saída

para empresas que buscam realizar a melhoria nos processos de forma mais gradual e com custo reduzido, mesmo que seja voltado para qualquer tipo de organização, o modelo se mostra ideal para pequenas e medias empresas, já o CMMI é um modelo que pode ser implementado após atingir-se um nível de maturidade maior no MR-MPS ajudando a garantir o sucesso e retorno na mudança de modelo, e como ambos são baseados nas normas ISO/IEC 12207 e ISO/IEC 15504, a compatibilidade na mudança é um facilitador. Ainda é possível afirmar que foi criado um norte para empresas que pretendem adquirir *software*s realmente de qualidade, já que ficou claro que empresas que utilizam e mantem esses processos melhoram a qualidade do produto final e os selos e certificações dos dois modelos é uma garantia.

5. BIBLIOGRAFIA

ANDRADE, Maria Margarida. **Introdução à Metodologia do Trabalho Científico**. 9°ed. São Paulo: Atlas Editora, 2009.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS: **Citação:** NBR-10520/ago-2002. Rio de Janeiro: ABNT, 2002.

_____**. Referências:** NBR-6023/ago. 2002. Rio de Janeiro: ABNT, 2002.

FERNANDES, Aguinaldo Aragon; ABREU, Vladimir Ferraz. **Implantando a Governança de TI**. 3°ed. Rio de Janeiro: Brasport Editora, 2012.

KOSCIANSKI, André; SOARES, Michel Dos Santos. **Qualidade de *Software***. 2°ed. São Paulo: Novatec Editora, 2006.

PRESSMAN, Roger S. **Engenharia de *Software***. 6ªed. São Paulo: Pearson Editora, 2006.

SOMMERVILLE, Ian. **Engenharia de *Software*.** 8ªed. São Paulo: Pearson Editora, 2007.

< http://www.blogcmmi.com.br/avaliacao/lista-de-empresas-cmmi-no-brasil >. Acesso em: 24 abr. 2013

< http://www.softex.br >. Acesso em: 12 abr. 2013.