UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Requisitos Mínimos de um Sistema da Qualidade para o Desenvolvimento de Software Embarcado em Ambientes de Pequenas Equipes

Antonio Valerio Netto
Paulo Aneas Lichti

Nº 165

RELATÓRIOS TÉCNICOS

São Carlos - SP

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

Requisitos Mínimos de um Sistema da Qualidade para o Desenvolvimento de Software Embarcado em Ambientes de Pequenas Equipes

Antonio Valerio Netto
Paulo Aneas Lichti

Nº 165

RELATÓRIOS TÉCNICOS

São Carlos – SP
Mai./2002

REQUISITOS MÍNIMOS DE UM SISTEMA DA QUALIDADE PARA O DESENVOLVIMENTO DE *SOFTWARE* EMBARCADO EM AMBIENTES DE PEQUENAS EQUIPES

Antonio Valerio Netto[1]

Paulo Aneas Lichti

Departamento de Ciências da Computação e Estatística
Instituto de Ciências Matemáticas e de Computação - ICMC
Universidade de São Paulo – USP – Campus São Carlos
Caixa Postal 668, CEP 13560-970, São Carlos, SP

1. INTRODUÇÃO

Atualmente há uma percepção difundida de que qualidade não é apenas uma questão pertinente ao produto, mas sim algo que se obtém em função das características do próprio processo do qual resulta o produto, desde as fases iniciais de pesquisa e desenvolvimento. Os clientes tornaram-se mais exigentes, não se importando apenas com o preço baixo e sim com um produto sem defeitos, confiável, que atenda às suas necessidades e a de seus clientes ou usuários. A crescente exigência por alta qualidade também ocorre com o *software*, já que há uma grande dependência da sociedade por produtos que sejam *software*, ou por produtos que contenham um *software*.

A qualidade no processo de *software* pode ser avaliada tanto pelas características do próprio produto quanto pelo processo através do qual o *software* é produzido. Uma maneira de se ter um produto de *software* com qualidade é ter um processo com qualidade [Lamprecht 1994]. Existem vários modelos para avaliação, gestão ou melhoria do processo de *software*[2]: o *Capability Maturity Model* - CMM (Curtis 1992); o *Software Process Improvement and Capability Determination* - SPICE (Lightoot 1995); *Quality Process Improvement/ Experience Factory* QPI/EF (McGarry 1994); *International Standards Organization* - ISO (ISO 9000-3 – Diretrizes para Implementação da ISO 9001 na indústria de *softwares,* 1994), além de requisitos normativos baseados em normas específicas, tais como as *Internationa Eletrotecnical Comission Standards* IEC 601-1-1-2 e 601-1-4, Síncrono-Estabilidade (modelo da Microsoft), etc. Constituindo-se assim em atividade cada vez mais complexa e exigente, torna-se necessária a adoção de práticas de engenharia que garantam o desempenho do processo e a conseqüente qualidade do produto obtido. Outro fator que contribui para aumentar a necessidade de que os processos de desenvolvimento de *software* tenham sua qualidade assegurada é a extrema velocidade e dinamismo dessa indústria, onde mudanças de tecnologia e mercado podem tornar um produto "obsoleto" ou "moderno", conforme o referencial adotado ou o padrão médio do mercado, inclusive no caso de produtos que possui o *software* "embarcado", foco deste trabalho.

[1] O autor agradece o apoio da FAPESP (Processos 00/04779-2).

[2] Processo de *Software*: conjunto de processos que vão da concepção do *software* ao pós–venda.

Portanto, a qualidade no processo de *software* é uma dimensão crítica para a competitividade dos produtos das empresas, e se bem entendida e aplicada, torna-se fonte de vantagem competitiva. Esse fator pode ser aproveitado por aqueles que compreenderem as crescentes demandas da indústria de equipamentos que incorporam cada vez mais funções controladas e monitoradas por *software*.

Atualmente, identificam-se claramente três requisitos básicos que devem existir nas organizações para que o processo de desenvolvimento de produtos torne-se crítico a ponto de justificar um sistema de gestão mais sofisticado: a) A exigência de um gerenciamento complexo e detalhado do projeto do produto; b) A possibilidade de existência de desempenhos diferentes dos processos em função das práticas adotadas; c) Competição gerando a necessidade de se lançar produtos onde o perfil do mercado, das características dos produtos e das tecnologias está em constante mudança. [Clark & Fujimoto 1991]. Estes requisitos são identificados nos equipamentos que incorporam *software*, como equipamentos com aplicação médica, aeroespacial, industrial, entre outros, onde o *software* possui uma dimensão crítica.

Assim, a combinação harmoniosa da criatividade e da organização para desenvolvimento de produtos *software* é o grande desafio dos centros privados ou públicos que se deparam com estas necessidades. A relativa igualdade de condições técnicas existentes hoje no Brasil comparada aos países lideres em *software*, combinada a uma recente massa crítica de profissionais de ciências exatas e a cada vez maior conscientização para a qualidade no país, expressa inclusive pela liderança brasileira na América Latina em certificações ISO 9000, proporcionam ao Brasil chances de se tornar um exportador de *software* e de produtos que agreguem o *software* como diferencial competitivo. Esta percepção pode-se observar pelo recente apoio da recém-criada Agência Brasileira de Exportações, que no início de suas atividades selecionou 55 setores para apoiar, sendo que o setor de *software* foi o primeiro a ser escolhido (Beck 1998).

Basicamente, cita-se comumente alguns fatores como cruciais para que o "item" *software* contribua para um diferencial positivo para o produto no qual está embarcado: a) Qualidade do *software* - que se deriva em grande parte da qualidade do processo do *software*; b) Realização e operacionalização das estratégias de marketing que os lançamentos e promoção dos produtos exigiriam e c) Integração plena do *software* com o produto hardware – permitindo a geração de novas versões de produtos, soluções completas e com maior valor agregado, mais seguras e eficazes.

Portanto, este presente trabalho tem o objetivo de apresentar os fatores críticos para a qualidade de *software* em um ambiente de desenvolvimento onde é um componente do produto e analisá-los sob a ótica dos sistemas da qualidade reconhecidos, propondo um conjunto de requisitos básicos a serem cumpridos no processo de *software* para empresas que os desenvolvem como componentes de produtos de alta tecnologia. O estudo de caso para o levantamento destes fatores foi realizado com base em situações reais de desenvolvimento: 2 equipamentos médicos e 2 equipamentos industriais.

2. Qualidade em *Software* - Uma visão geral

A qualidade do produto "*software*" diz respeito a certas características que contribuem para a sua utilidade (Pressman 1994). A qualidade focaliza três aspectos importantes do produto de *software*: sua adaptabilidade diante de novos ambientes, suas características operacionais e sua habilidade em suportar mudanças. A seguir apresenta-se uma descrição desses fatores da qualidade segundo MacCall (MacCall 1977), *International Software Organization* (ISO) (norma ISO9126 1994) e *Institution for Electrical Engeneers* (norma IEEE 1061 1992).

Tabela 1: Adaptabilidade a novos ambientes

MaCall	
Portabilidade	Esforço para transferir o programa de um hardware e/ou ambiente de *software* para outro.
Reusabilidade	A extensão que o programa, ou parte do programa, pode ser reutilizada em outras aplicações.
Interoperabilidade	Esforço necessário para unir um sistema a outro.
ISSO 9126	
Portabilidade	Habilidade do *software* de ser transferido de um ambiente para outro. O ambiente pode incluir ambiente organizacional de hardware ou de *software*.
Interoperabilidade	Capacidade de interagir com sistemas especificados.
IEEE 1061	
Portabilidade	Habilidade do *software* de ser transferido de um ambiente para outro.
Interoperabilidade	Grau em que o *software* pode ser facilmente conectado a outros sistemas e pode ser operado.

Tabela 2: Características operacionais do produto

MacCall	
Corretitude	A extensão do quanto o programa satisfaz suas especificações e o quanto cumprem a missão dos objetivos dos clientes.
Confiabilidade	A extensão do quanto o programa pode desempenhar as funções esperadas com uma certa precisão.
Eficiência	O total de recursos computacionais e código requerido pelo programa para desempenhar suas funções.
Integridade	A extensão do quanto o acesso ao *software* ou aos seus dados por pessoas não autorizadas pode ser controlado.
Usabilidade	O esforço requerido para aprender, operar, preparar dados de entrada e interpretar os dados de saída do programa.
ISO 9126	
Confiabilidade	Capacidade do *software* de manter o seu nível de desempenho sob determinadas condições, por um período de tempo determinado.
Eficiência	Relação entre o nível de desempenho do *software* e o total de recursos usados sob determinadas condições.
Usabilidade	Esforço necessário para o uso, e a taxação individual para este uso por um conjunto de usuários declarados ou envolvidos.

Funcionalidade	Existência de um conjunto de funções e suas propriedades especificadas.
IEEE 1061	
Corretitude	O grau em que todas as funções do *software* são especificadas.
Confiabilidade	Capacidade do *software* de manter o seu nível de desempenho sob condições estabelecidas e por intervalo de tempo estabelecido.
Eficiência	Um atributo que confirma a relação entre o nível de desempenho e a quantidade de recursos utilizados sob condições estabelecidas.
Usabilidade	Esforço necessário par ao uso (incluindo preparação para uso e resultados de avaliação) e na avaliação individual de tal uso.
Funcionalidade	Existência de certas funções e suas propriedades que satisfaçam as necessidades implícitas e explícitas dos usuários.

Tabela 3: Habilidade de suportar mudanças

McCall	
Manutenibilidade	Esforço requerido para localizar e corrigir um erro no programa.
Flexibilidade	Esforço necessário para modificar um programa operacional.
Testabilidade	Esforço requerido para testar um programa para garantir que desempenhe a função pretendida.
ISO 9126	
Manutenibilidade	Esforço necessário para fazer modificações especificadas no *software*. As modificações podem incluir correções, melhorias ou adaptações do *software* devido a mudanças no ambiente, ou nos requisitos.
Testabilidade	Esforço necessário para validar o *software* modificado.
IEEE 1061	
Manutenibilidade	Um atributo que confirma a necessidade de mudanças específicas.
Testabilidade	Esforço requerido para testar o *software* modificado.

Porém, trabalhar a qualidade do produto "*software*" tornou-se insuficiente. As falhas nos processos de gestão e manutenção do desenvolvimento de *software* foram identificadas como inibidores no crescimento da qualidade e produtividade (Lamprecht 1994). Assim, um processo de *software* definido e documentado, utilizado para integrar profissionais, tarefas, ferramentas e métodos pode prover uma base para assegurar a qualidade do produto final, sendo que o conhecimento do tipo de processo existente e as características dos produtos a serem produzidos durante o desenvolvimento são informações importantes para aperfeiçoar ou quantificar a qualidade do produto. Os princípios básicos dessa quantificação são os mesmos do controle estatístico de processo, muito utilizados em outras áreas. Um processo é estável, ou sob controle

estatístico, se é possível prever seu desempenho dentro dos limites estatísticos de controle (Humphrey 1989).

2.1 Resumo dos Modelos Analisados

Cinco modelos foram considerados na análise. Os modelos CMM, SPICE, ISO 9001, PSP e Síncrono-Establidade são resumidos a seguir, com base nos trabalhos de ROSA (1997), CUSSUMANO&SELBI (1995) e BARRETO (1998).

2.1.1 Síncrono-Establidade (SE)

Pode-se sintetizar as características do SE como segue:

a) Diretriz 1: ***Priorizar a criatividade por meio da evolução de novas características e da fixação de recursos***

Esta diretriz trata da definição de novos produtos e de processos de desenvolvimento que facilitem o direcionamento dos mercados de interesse da Microsoft, que são os mercados de massa. Cussumano e Selby (1995) reuniram cinco princípios que embasam esta diretriz 1:

1. Divisão de grandes projetos em múltiplos ciclos de desenvolvimento com *buffers* (tempos tampão utilizado para reavaliações ou contingências imprevistas);
2. Declaração de visão e esboço da especificação das características para orientação dos projetos;
3. Seleção das características do produto e suas priorizações com base nas atividades e dados de usuários;
4. Desenvolvimento de arquitetura de projeto modular em que a estrutura do produto espelhe-se na do projeto de desenvolvimento;
5. Controle dos comprometimentos individuais com pequenas tarefas e recursos fixos de projetos;

A abordagem geral da Microsoft para o processo de desenvolvimento compõe-se de três fases: planejamento, desenvolvimento e estabilização, como pode ser visto na figura 1 (Síncrono – Estabilidade). Esta abordagem para o processo de *software* é o resultado de uma longa reorganização que se iniciou em meado da década de 1980, quando a Microsoft começou a ter sérios problemas com qualidade, incluindo prazos de entrega de produtos, grupos cada vez maiores e produtos cada vez mais complexos. Este estilo contrasta com o de muitas empresas de *software*, que utilizam uma abordagem mais estrutural e seqüencial no processo de desenvolvimento: as empresas desenvolvem os componentes, "congelam uma especificação" e integram e testam o produto em uma única fase. Este modelo tem sido cada vez menos utilizado, e um estilo interativo, incremental e simultâneo tem sido cada vez mais aceito, e esta é a principal característica do processo de "Síncrono – Estabilidade".

Tabela 4: Visão Geral do Processo de Síncrono - Estabilidade

Fase de Planejamento: define visão do produto, a especificação e o cronograma.
• Declaração de Visão: os gerentes de produto e programa empregam exaustivamente a contribuição do cliente para identificar e priorizar as características dos produtos.

• Documentação de Especificação: baseada na declaração de visão, no gerenciamento do programa e no desenvolvimento do grupo, a documentação define a funcionalidade das características, as questões arquitetônicas do *software* e as interdependências dos componentes do *software.* • Cronograma e Formação da equipe de características: baseado na documentação de especificações, o gerente de programa coordena o cronograma e organiza as equipes de características que contém, cada uma, aproximadamente um gerente de projeto, três a oito desenvolvedores e testadores, que trabalham em paralelo e em número de 1:1 com os desenvolvedores.
Fase de Desenvolvimento: desenvolvem-se características em três ou quatro subprojetos seqüenciais que resultam, cada um, numa versão associada a um evento.
Os gerentes de programa coordenam a evolução das especificações. Os desenvolvedores projetam, codificam e corrigem erros. Os testadores formam pares com os desenvolvedores para realização contínua de testes. • Subprojeto I: 1/3 inicial das características – a maior parte de características críticas e componentes compartilhados. • Subprojeto II: mais 1/3 das características. • Subprojeto III: 1/3 final das características menos críticas. • Otimização, testes e correções de erros.
Fase de Estabilização: São feitos testes exaustivos, internos, externos e estabilização do produto final.
Os gerentes de programa coordenam as versões de teste e monitoram o retorno dos clientes. Os desenvolvedores fazem as correções finais dos erros e a estabilização do código. Os testadores recriam e isolam os erros. • Teste interno: testes completos de produto dentro da empresa. • Teste externo: testes completos do produto fora da empresa. • Preparação de versão: preparação da versão final da mídia "golden master" e documentação para a produção em série.

O ciclo de vida para as fases apresentadas na tabela 4 varia de 12 a 24 meses, do evento zero (início do planejamento) até a fabricação. O planejamento possui um tempo de 3 a 12 meses, e termina com a aprovação do projeto. O desenvolvimento possui tempo de 6 a 12 meses. Os finais dos subprojetos I, II e III marcam as versões, até a otimização e obtenção completa da codificação. No final de cada subprojeto a os tempos de *buffer* (1/3 aproximadamente de cada subprojeto) para contingências não previstas. Por fim, a fase de estabilização, com tempo de 3 a 8 meses, contendo a versão "zero-defeito" e a versão de estabilização que marca o fim do ciclo de desenvolvimento e o anúncio do lançamento do produto.

b) Diretriz 2: ***Fazer tudo paralelamente e em sincronia constante***

Esta diretriz confere, segundo Cussumano & Selby (1995), um pouco de estrutura à cultura pouco rígida da Microsoft. Há liberdade significativa para pequenos grupos que trabalham paralelamente, sem desconectar sua subordinação ao grupo maior. Isso permite desenvolver produtos em grande escala com baixo custo e muito rápido. Durante o andamento da produção dos códigos, os desenvolvedores podem projetar detalhes novos. No entanto, o processo de

desenvolvimento estimula-os a sincronizarem seus projetos com os de outros desenvolvedores na busca de soluções conjuntas. Para garantir certa ordem, algumas regras são observadas: uma delas prescreve que o horário de sincronização deve ser estabelecido de maneira prévia. Isso permite que os grupos compatibilizem os componentes, criando novos modelos (novas combinações de componentes). Isso também permite verificar quais funções estão operantes e quais são problemáticas. Outro procedimento fundamental refere-se à regra que prevê que se o responsável pelo desenvolvimento do produto constatar a existência de um código que prejudique o sistema, ele deverá repará-lo prontamente. Isto é similar ao sistema de produção da Toyota, em que a fábrica interrompe as linhas de produção sempre que se percebe um defeito em um automóvel em fase de montagem (Cussumano, 1985).

Os grupos de desenvolvimento testam também as características introduzidas no produto à medida que o fabricam sob múltiplas perspectivas, trazendo inclusive clientes da rua para testarem os protótipos. Praticamente todos os grupos da Microsoft trabalham em um único local, com ferramentas comuns, visando solucionar problemas com presença física. A Microsoft também se utiliza de um pequeno conjunto de medidas quantitativas ou métricas que servem de orientação para se saber quando deve-se prosseguir com um projeto ou quando deve-se lançar o produto no mercado. Esse processo, claramente semi-estruturado é muito eficaz para o grupo de produtos aplicativo[3]: o resultados tem sido um fluxo contínuo de novos produtos, com características incorporadas incrementalmente, bem como características novas para produtos antigos.

Os cinco princípios, portanto, que embasam a segunda diretriz são: (Cussumano&Selby, 1995):

1. Trabalho em grupos paralelos e em sincronia, depuração e registros dos códigos que obtiverem sucesso diariamente – não há projetos com especificação completa e definitiva antes do desenvolvimento;
2. Ter produtos que possam ser, teoricamente, lançados em versões para todas as plataformas e mercados importantes;
3. Utilização de uma linguagem de programação comum em um único local de desenvolvimento;
4. Teste e correção contínuos do produto, à medida que é desenvolvido;
5. Utilização de dados de mensuração para determinação da conclusão de eventos e o lançamento do produto.

Especificamente para o princípio 4 - *teste e correção contínuos do produto, à medida que é desenvolvido* – identificou-se os seguintes métodos para controle da qualidade:

- **Teste rápido**: ferramenta chave automatizada que permite ao desenvolvedor conhecer se as mudanças propostas não interferem com outras características do produto e indiretamente provoque falha. Se o teste obtiver sucesso, o desenvolvedor registra as alterações do código para a aquela versão particular;
- **Testes de versões particulares**: os testadores, que se reportam ao gerente de testes, e não ao gerente do desenvolvimento, porém trabalhando em conjunto com os desenvolvedores, testam novos códigos e revisam especificações – atividade obrigatória antes de se registrar a versão diariamente. O objetivo básico desta atividade é a detecção de defeitos antes de eles penetrarem no código-fonte principal e antes de o mesmo ser compartilhado com outros desenvolvedores da equipe;

[3] Aplicativo: tipo de *software* que serve a uma aplicação específica. Exemplo: um processador de textos.

- **Código de eliminação de erros**: códigos depuradores de erros inseridos no código fonte do produto. Os desenvolvedores redigem seus próprios códigos fonte como parte de seu trabalho regular. Esses códigos podem assumir vários papéis como o de verificação da memória e da estrutura de dados, checagem de informações que devem ser sempre verdadeiras e sondagem de características mais acionadas pelos usuários;
- **Revisões de código**: antes de implementar as características, os grupos da Microsoft executam um volume limitado de revisões de projeto formais e exaustivas. O objetivo é detectar códigos ausentes ou problemas de consistência antes da execução das atividades de testes, que exigem que o produto seja "rodado";
- **Teste de versões e documentação da versão para testes**: há a produção de versões de teste para os grupos de testadores. Os erros são registrados. Essas versões apresentam cada característica no primeiro momento de sua conclusão, e este documento é enviado para os testadores. O documento de especificação do produto também é utilizado pelos testadores como fonte de informações para testes;
- **Teste de uso**: é um método de controle da qualidade usado pela Microsoft, durante o desenvolvimento, num ambiente formal de laboratório. Neste tipo são testadas principalmente as interfaces para usuários e características de aplicativos, sempre enfatizando a facilidade de uso do produto como característica da qualidade. Este método prevê a participação de usuários utilizando as funções do produto, com gravação em vídeo e observação externa. As taxas de sucesso obtidas por usuários sem consultar o manual são registradas. A modelagem do usuário é então obtida e respostas sobre o produto são obtidas instantaneamente;
- ***Benchmarking* de desempenho e testes "gorila":** Os produtos, em especial os sistemas operacionais4, devem ter alto desempenho e altos níveis de confiabilidade. Os testes utilizados de *benchmark* são padrão na indústria de computadores (para sistemas operacionais), como os testes de interface de programação de aplicativo (IPA) de baixo nível, que são enviados aos produtores independentes de *software*, pois assim os mesmos podem iniciar o desenvolvimento de novos aplicativos antes do lançamento do sistema operacional. Paulatinamente os testes passam para níveis mais altos, pois as IPAs já não detectam muitos erros;
- **Uso interno e *Self-Hosting***: consiste em usar o produto novo ou em desenvolvimento internamente. Quando esses testes são utilizados no projeto para desenvolver novas características, os mesmos são chamados de *self-hosting*. São muito utilizados na Microsoft principalmente para produtos de sistemas operacionais (Windows 95, por exemplo), devido ao baixo custo. A limitação do teste *self-hosting* é que os funcionários da empresa nem sempre representam o perfil dos usuários dos produtos;
- **Testes "betas"**: devido à complexidade dos produtos de *software*, seus cenários de uso, quantidade de entrada de dados, configurações e comandos diversos, há uma grande quantidade de combinações possíveis em sua operação. Um exemplo: um sistema com 100 características: cada uma pode gerar uma conclusão de execução normal e pode gerar duas mensagens de erro. O produto deve rodar em 20 *drivers* de discos de diferentes fornecedores e suportar 100 dos mais populares aplicativos, além de 50 dos comandos utilizados com maior freqüência para cada um desses aplicativos. Para simplesmente iniciar os testes da qualidade do produto, os testadores terão de configurar um laboratório com capacidade de suportar mais de 9 bilhões de cenários de uso (100X3X20X4X15X5X100X50). Mesmo que fosse possível montar este laboratório de testes, ele seria anti-econômico – sem levar em

[4] Sistema Operacional: *software* que é a plataforma sobre a qual os outros *softwares* são executados.

consideração que esta lista está longe de ser completa. O teste beta então é aplicado, que é o teste em campo, mais amplo, e que leva um tempo razoável. Os testes "betas" da Microsoft utilizam-se dos próprios usuários de PC, e o correio eletrônico e a Internet tem sido uma ferramenta amplamente utilizada para relatório de falhas apontadas pelos testadores betas. Historicamente, os testes beta têm sido responsáveis pela detecção de parte significativa dos erros;

- **Comparação de eficácia de técnicas de testes da qualidade**: todos os testes descritos são comparados na forma de percentual de erros detectados.

O quinto princípio também é significativo para a qualidade do processo de desenvolvimento de *software*, – *Utilização de dados de mensuração para determinação da conclusão de eventos e o lançamento do produto* - pois define quando um processo deve parar, concluindo o evento de se obter um *software* apto para ser lançado no mercado com um nível aceitável de qualidade. Mensurações empíricas e dados estatísticos são utilizados como forma de compreender, rastrear e visualizar o progresso do processo. São dados fundamentais para subsidiar as decisões dos gerentes de programa e desenvolvedores: passam-se para o próximo evento, se atrasam ou lançam o produto na data prevista. Os métodos empregados no sistema são:

- **Utilização dos dados de erros (*bugs*) para avaliação da conclusão do evento**: os gerentes da Microsoft usam erros como base para a decisão de definir quando um *software* atinge a qualidade necessária para o cumprimento do evento principal. O erro é definido como a ocorrência de não cumprimento de uma função que sua especificação previa. Há uma correlação positiva entre qualidade total do *software* e quantidade de erros, bem com entre este e taxa de melhoria na qualidade durante o processo de desenvolvimento. As correções são diariamente monitoradas e incorporadas à versão mestra do código fonte pelo desenvolvedores com dados fornecidos pelos testadores. Os principais indicadores utilizados são: ***bugs abertos*** – que é o total de erros encontrados; ***bugs fixos*** – total de *bugs* corrigidos; ***bugs resolvidos*** – total de *bugs* adiados, duplicados, etc. e ***bugs ativos*** – aqueles localizados, porém ainda não corrigidos ou resolvidos. A regra geral é que a conclusão real do evento se dê quando o número total de *bugs* ativos que possam causar o desligamento do sistema ou destruição de dados esteja próximo de zero enquanto os testes ainda prosseguem. Ainda há também a classificação e mapeamento de erros em níveis de gravidade;
- **Detecção e correção de *bugs* em tempo integral**: após a conclusão do evento de desenvolvimento do código completo, os desenvolvedores dedicam-se em grande parte do tempo à detecção de erros, chegando às vezes à dedicação integral. Isto é feito para aproveitar a experiência dos mesmos e porque possuem grande conhecimento do código fonte, o que facilita o trabalho de detecção;
- **Listas de verificação baseadas em mensurações para características e conclusão do produto**: Um desenvolvedor, testador ou gerente de programa utiliza-se de uma lista de verificação para cada característica, e verifica se a mesma está completa. Estas listas podem conter informações como: verificar se todos os testes estão 100% concluídos ou verificar se os erros de alta gravidade foram testados pelo menos duas vezes;
- **Indicadores "Quando Lançar"**: o processo de módulos diários garante um produto em constante aperfeiçoamento. Todos testam e buscam erros, sejam desenvolvedores, testadores ou gerentes. Porém somente os desenvolvedores os corrigem e modificam o código fonte. Esses indicadores são: taxa de detecção de novos erros – que deve ser próxima de zero à

medida que se aproxima da data-alvo de lançamento e quantidade de erros corrigidos igual ao total de erros detectados ao longo do processo.

c) Diretriz 3: ***Construção de uma organização disposta a aprender: melhore por meio da autocrítica contínua, do feedback e do compartilhamento***

Esta diretriz é baseada nos seguintes princípios:

1. Aprendizado sistemático com projetos e produtos do passado e do presente;
2. Incentivo ao *feedback* e ao aperfeiçoamento por meio de mensurações e de *benchmarks*;
3. Encarar o suporte ao cliente como uma característica da qualidade do produto e como informação básica para a sua melhoria;
4. Promoção de conexões e compartilhamentos entre os grupos de produtos.

A organização que aprende (*Learning Organization*) é um tema de estudo bastante freqüente. E os princípios que existem na Microsoft também são encontrados em outras empresas. Não são muitas organizações de *software* que conduzem, rotineiramente, relatórios *post mortem* de projetos ou realizam auditorias. Os grupos da Microsoft o fazem, com a preocupação básica de não se tornar excessivamente burocrática. Isso leva a empresa a não realizar tantas mensurações quanto o fazem a IBM, Hewlett – Packard ou Fujitsu (Cussumano,M & Selby, R. – 1995). Porém, a Microsoft possui mais clientes que fazem perguntas via telefônica e correio eletrônico do que qualquer empresa de *software*. Constuindo em grande vantagem comparativa.

O primeiro princípio – *aprendizado sistemático com projetos e produtos do passado* – é implementado a partir de relatórios para garantir um aprendizado constante e o compartilhamento no processo de desenvolvimento. Esses relatórios são análises críticas que resultam em relatórios de 10 a mais de 100 páginas. Eles englobam detalhes e análises sobre a forma e a evolução dos processos de desenvolvimento, relatórios detalhados de erros, descrição das atividades de teste, avaliação de cronograma, etc. As outras ferramentas para operacionalização desse princípio são:

- **Auditorias de processo**: as auditorias verificam constantemente dados de projeto e da qualidade, e a visão é a de buscar constantemente formas de melhorar os procedimentos, com base inclusive nos resultados dos outros grupos. A auditoria é enfatizada como sendo uma forma de "troca técnica";
- **Retiros**: é um padrão de análise crítica "macro" dentro da Microsoft. Desde o início dos anos 80, os executivos da empresa retiram-se pelo menos uma vez por ano para reflexões sobre temas estratégicos como tendências na área de multimídia, comércio on-line, questões de como estruturar a organização para um novo cenário e qualidade. Leituras sobre qualidade (livros de Crosby) e sobre cronograma de projetos (livros de Tom DeMarco) foram amplamente utilizadas nesses retiros, e melhorias que poderiam ser obtidas pela Microsoft a partir dessas discussões foram efetivamente testadas.
- **Compartilhamento entre grupos**: pela dificuldade de troca de informações pelo porte atual de empresa, é amplamente estimulada a realização de almoços e reuniões informais regulares entre gerentes, havendo até a apresentação de palestras visando o compartilhamento de informações a respeitos de projetos específicos. Algumas são gravadas em vídeo e repassadas pela empresa. Esses encontros são chamados de "bandeja – azul" para gerentes, "pasta – marrom" para desenvolvedores.

Instituto de Ciências Matemáticas e de Computação
Biblioteca

O segundo princípio - *Incentivo ao feedback e ao aperfeiçoamento por meio de mensurações e de benchmarks* – é uma resposta, ao menos parcial, aos relatórios *post mortem* da Microsoft. Mensurações e dados possuem um lugar de destaque na Microsoft. Essas mensurações que embasam o *feedback*, tornando-o não emocional ou subjetivo são:

- **Qualidade no Processo**: como já visto, incluem taxas de detecção de erros diárias ou semanais, níveis de gravidade de erros (gravidade 1 – falha geral; 2 – características inoperantes sem solução possível, 3 – uma característica inoperante e 4 – erro aparente ou insignificante), com solução possível, resultados de laboratórios de uso, índices de satisfação do cliente, taxa de solução de erros, freqüência de erros em clientes, grau de reutilização de códigos, listas de verificação para conclusão de eventos, datas de conclusão estimadas versus reais, etc;
- **Qualidade do Produto**: número de características, tamanho do produto em número de linhas de código; *bytes* de arquivos executáveis, bytes de memória, etc.

Todas essas informações são utilizadas para mensurações sobre a qualidade ao longo do desenvolvimento do produto. E antes do início do projeto, os dados de mensuração são aplicados para prever-se os requisitos de cronograma e recursos de projeto. E um *feedback* constante é obtido ao longo do processo, e os mesmos são explorados por gerentes, testadores e desenvolvedores para destacar tendências e oportunidades de melhoria. Porém, segundo apontou o estudo, a Microsoft ainda possui muitas regras empíricas e as mensurações são limitadas em comparação com outras empresas como Motorola, HP, NEC e Hitachi.

Os *benchmark* dos procedimentos mais eficazes têm sido usados como incentivo também ao compartilhamento de padrões, termos e mensurações comuns entre grupos.

O terceiro princípio - *Encarar o suporte ao cliente como uma característica da qualidade do produto e como informação básica para a melhoria* – visa sustentar o aprendizado da Microsoft não apenas com base em fontes internas. Os clientes são fontes inestimáveis de informações, e para isso utilizam-se de *inputs* dos clientes durante o processo de desenvolvimento.

O último princípio - *Promoção de conexões e compartilhamentos entre os grupos de produtos* – refere-se à ênfase da Microsoft com a integração de seus produtos – como, por exemplo, em Suítes de *Software* como o *Office*, que integram vários produtos em um único pacote, não exatamente é uma característica de seu sistema da qualidade, mas de sua estratégia de mercado. Portanto, não faz parte do escopo deste trabalho.

2.1.2 Personal Software Process - PSP

O SEI, após perceber a necessidade de definir um modelo mais simples que o CMM para pequenas empresas ou até mesmo para empresas individuais, desenvolveu o PSP, que possuem quatro níveis de acordo com a tabela 5 [Barreto, 1998].

Tabela 5: Níveis do PSP

Nível	Nome	Atividades
PSP 0 e PSP 0.1	Medição pessoal	• Registro de tempo • Registro de defeitos • Padrão de tipos de defeitos • Padrão de codificação • Medida de tamanho

		• Proposta de melhoramento de processo
PSP 1e PSP 1.1	Planejamento pessoal	• Estimativa de tamanho • Relatório de testes • Planejamento de tarefas • Cronogramas
PSP 2 e PSP 2.2	Qualidade pessoal	• Revisões de código • Revisões de projeto • Padrões de projeto
PSP 3	Processo cíclico pessoal	• Desenvolvimento cíclico

Fonte: Barreto, 1998.

No nível de medição pessoal registra-se o tempo gasto em cada etapa do ciclo de desenvolvimento, além de mapear os defeitos encontrados. Isto é conseguido através do uso de formulários adequados. O nível PSP 0.1 inclui o uso de um padrão de codificação, de medidas padronizadas e do formulário de proposta de melhoria do processo.

No nível de planejamento pessoal, aprende-se a planejar. Deve-se obter a capacidade de estimar o tempo a ser gasto em determinada tarefa com base em medições feitas em tarefas semelhantes anteriormente. Aprende-se neste nível a assumir compromissos que efetivamente possam ser cumpridos. O nível PSP 1.1 inclui o planejamento de tarefas e a elaboração de cronogramas.

No nível de qualidade pessoal aprende-se a lidar com os erros. Deve-se ter uma idéia precisa de quantos erros são em média cometidos para cada fase do desenvolvimento. O modelo PSP propõe que a melhor forma de tratar os erros é a prevenção. Uma lista de verificação deve ser criada para ser utilizada nas revisões de projeto e de código. O nível PSP 2.1 inclui a criação de padrões de projetos e métodos de análise e prevenção de defeitos

O nível de processo cíclico pessoal é a última etapa do PSP. Neste nível, o modelo trata de produtos mais complexos, porém ainda dentro de uma pequena estrutura. O conceito básico é o da divisão dos grandes projetos em pequenos projetos tratáveis no PSP 2. O desenvolvimento acontece neste caso em passos incrementais.

O treinamento no PSP é realizado através de 10 exercícios de desenvolvimento de programas. Além de servirem como exemplos de desenvolvimento, os exercícios são utilitários que ajudam a guiar a implementação do PSP, possibilitando o cálculo de desvios – padrão, intervalos e medir número de linhas e objetos nos programas.

2.1.3 CMM - Capability Maturity Model

O *Capability Maturity Model* (CMM) foi desenvolvido pelo *Software Engineering Institute* (SEI) (Curtis 1992, Lamprecht 1994). Este modelo propõe diretrizes de como aumentar o controle sobre seu processo de desenvolvimento em engenharia de *software* (Humphrey 1989, Tsukumo 1996). No modelo CMM, os processos de desenvolvimento de *software* são classificados em cinco níveis de maturidade que representam a situação em que se encontra a organização em termos de processo (Lamprecht 1994). Cada nível é composto por um conjunto de metas que, quando satisfeitas, são um marco na capacitação do processo de *software* (Humphrey 1989).

Nível 1: Inicial

O nível inicial de maturidade é caracterizado pela descrição de uma organização "imatura", onde os procedimentos e controles de desenvolvimento de *software* são frágeis(Lamprecht 1994). Uma das características das organizações "nível 1" é que compromissos irreais são assumidos pelos gerentes, sem que haja uma consulta aos responsáveis pelo desenvolvimento do *software*, sobre a possibilidade desses compromissos serem atingidos (Tsukumo 1996). Quando metas irreais são impostas, há normalmente um comprometimento da funcionalidade do produto e uma eliminação ou diminuição de etapas de desenvolvimento, como testes e revisões, para que o cronograma seja obedecido. Nestas organizações, a documentação é incompleta e quase sempre não está atualizada. Não há uma história de como o produto foi desenvolvido e porquê as decisões de projeto e implementação foram tomadas de determinada maneira. Isto gera dificuldades na manutenção do *software* em organizações que se encontram no "nível 1" (Curtis 1992).

Nível 2: Repetitivo

A principal diferença desse nível com relação ao anterior é o processo ter como fundamento um controle sobre os procedimentos, sobre os compromissos e sobre outras atividades, através de fundamentos gerenciais de processos de desenvolvimento de *software* (Curtis 1992). Com o planejamento e a garantia dos procedimentos, os projetos podem obter constantes sucessos e a metodologia de desenvolvimento pode ser assimilada e aplicada a projetos similares. Outra situação que envolve mudanças no processo é o desenvolvimento de novos tipos de produtos, o que pode causar um desequilíbrio no projeto, nas estimativas de custos e nos cronogramas, uma vez que um fator relevante para a organização neste nível é a dependência das experiências anteriores. Nesta situação, problemas em atingir os compromissos são analisados no momento em que os mesmos aparecem (Lamprecht 1994).

Nível 3: Definido

Neste nível é definido e documentado um processo de *software* padrão para toda a empresa, incluindo o processo de desenvolvimento e de gerenciamento do projeto de *software* (Lamprecht 1994). Cada projeto de *software* utiliza o processo padrão da organização como base para implementar seu próprio processo. A existência de um processo definido permite que as atividades de engenharia e gerenciamento sejam integradas a cada projeto (Curtis 1992). Um grupo para processos de engenharia de *software* (SEPG) é estabelecido para facilitar atividades de definição e melhoria de processos. Cada projeto possui revisões cuidadosas para aumentar a qualidade do produto. Um programa de treinamento para toda organização é implementado para assegurar que todos os engenheiros e gerentes tenham o conhecimento e a capacidade requerida para desenvolver suas tarefas, utilizando as ferramentas e os métodos que selecionaram (Lamprecht 1994). Este é, provavelmente, o primeiro nível em que as ferramentas e métodos de engenharia de *software* podem ser aplicados de maneira significativamente útil (Tsukumo 1996).

Nível 4: Gerenciamento

Neste nível, a organização estabelece metas quantitativas de qualidade para os produtos de *software*. Além disso, a produtividade e a qualidade são medidas em todas as etapas do processo de *software* e para todos os projetos da organização. Estas medidas têm como requisito fundamental o estabelecimento do processo de *software* efetuado no nível 3 (Curtis 1992). Os

projetos adquirem controle sobre seus produtos e processos através da diminuição da variação do seu desempenho para dentro de limites quantitativos aceitáveis. É neste nível que a organização começa a aplicar métricas de controle de qualidade para aumentar a qualidade e a produtividade do *software* entregue aos clientes.

À medida que a organização adquire mais conhecimento sobre o produto, tem a oportunidade de remover várias causas que comprometem a qualidade final (Humphrey 1989). Isto proporciona a oportunidade de colocar o produto sob um controle estatístico de qualidade.

Nível 5: Otimizado

Neste nível, a organização focaliza suas atenções numa contínua melhoria do processo. Baseada nas capacidades quantitativas estabelecidas no nível 4, ela pode identificar pontos fracos do processo e fortalecê-los com o objetivo de aumentar a qualidade (Lamprecht 1994). Tecnologias que proporcionem mais retorno, para processos específicos utilizados pela organização, são selecionadas para serem introduzidas, de maneira gerenciável na organização. Apesar de o processo ser "bom", ele é alvo de contínuas melhorias (Tsukumo1996). Os grupos de projetistas analisam o rendimento do projeto para determinar as causas dos defeitos. Os processos de *software* são avaliados para prevenir tipos de defeitos conhecidos devido à recorrência, e as lições aprendidas são disseminadas para outros projetos. Neste nível foi atingido um ambiente de excelência em engenharia de *software*. Por possuírem dados históricos de processos e de produtos, necessários para se fazer os aperfeiçoamentos, as operações de manutenção são as primeiras a atingir o nível 5 (Curtis 1992).

2.1.4 Software Process Improvement and Capability dEtermination (SPICE)

O SPICE (*Software Process Improvement and Capability Determination*) tem o objetivo de gerar normas para avaliação de processos de *software*, visando a melhoria contínua do processo e a determinação da sua capacidade (Lighfoot 1995 A). A estrutura SPICE fornece meios para avaliar os processos de *software* usados pela organização. Os resultados da avaliação de processos de *software* podem ser usados tanto para a determinação da capacidade dos processos, quanto podem ser usados nas atividades de melhoria do processo Uma visão geral dos relacionamentos entre avaliação de processo, melhoria de processo e determinação da capacidade do processo é mostrada a seguir:

Parte 1 - Conceitos e Guia Introdutório

Descreve como as outras partes do modelo se integram e fornece diretrizes para sua seleção e uso. Explica os requisitos do modelo e sua aplicabilidade para conduzir uma avaliação, para construção e seleção de ferramentas de suporte e para construção de um processo estendido (Lighfoot 1995A). Processos estendidos ou são processos inteiramente novos, ou são processos que incluem práticas que excedem as práticas básicas, procurando adequar o processo às características específicas das organizações.

Parte 2 - Descrição de um Modelo Idealizado de Processo

Um modelo de processo define as atividades fundamentais que são essenciais para que o mesmo seja considerado completo e efetivo. Um modelo descreve quais atividades são exigidas e não como devem ser implementadas. Um modelo de processo contendo as atividades fundamentais para uma boa engenharia de *software* deve ser definido. Este modelo deve ser

expresso em termos de características comuns e de práticas genéricas ao processo como um todo. Estas práticas representam as atividades necessárias para gerenciar um processo e melhorar sua capacidade para atingir as saídas desejadas (Lighfoot 1995B).

Diferentes modelos idealizados de processo de *software* podem ser desenvolvidos de acordo com o tipo de organização, com base nas normas ou modelos gerais existentes.

Parte 3 - Classificação dos Processos

A parte 3 do SPICE define os critérios para pontuar a capacidade de processos em relação ao processo idealizado. Além disso, nesta parte deve ser definido um formato de registro e saída das avaliações. O SPICE não impõe qualquer formato específico para isto. Esta parte é importante para garantir que os resultados das avaliações sejam confiáveis e reproduzíveis (Olavarrieta 1996).

Parte 4 - Guia de Orientação para Condução de Avaliação

A parte 4 fornece um guia de orientação para conduzir a equipe de avaliação. As diretrizes deste guia devem ser suficientemente genéricas para serem aplicáveis aos vários tipos de organização. Este guia fornece base para avaliar e interpretar os critérios definidos na parte 3.

Parte 5 - Diretrizes para elaboração de um instrumento de Avaliação

Um instrumento de avaliação é uma ferramenta, ou um conjunto de ferramentas, usados durante a realização de uma avaliação. A parte 5 do modelo SPICE fornece diretrizes para construção de um instrumento de avaliação. A estrutura fornecida prescreve um conjunto de elementos comuns que devem ser incorporados em qualquer tipo de instrumento de avaliação:

- Indicadores de avaliação que ajudam a analisar e fazer avaliações consistentes sobre as práticas implementadas.
- Um mecanismo para armazenar anotações e resultados que podem fornecer informações para uso na melhoria do processo ou determinação da capacidade do processo.
- Assistência ao avaliador para analisar pontuações e obter perfis dos processos.

Nesta parte não é fornecido um formato padrão para um instrumento de avaliação. O instrumento pode ser implementado como um questionário, um sistema especialista ou uma ferramenta manual ou on-line (Lighfoot 1995B).

Parte 6 - Qualificação e Treinamento de Avaliadores

Esta parte fornece orientação para a preparação e qualificação dos avaliadores. Está relacionado com treinamento, experiência, competência e educação do avaliador. Esta parte inclui mecanismos que podem ser usados para demonstrar competência e validar a educação, treinamento e experiência (Olavarrieta 1996).

Parte 7 - Guia de Orientação para Melhoria de Processo

Esta parte descreve como definir as entradas para a avaliação do processo e como usar os resultados da avaliação para o propósito de melhoria de processo. O guia inclui exemplos de aplicação de melhoria de processo numa variedade de situações. Os conceitos e princípios são

apropriados para diferentes necessidades de negócios, domínios de aplicação e tamanho de organização, e eles podem ser usados por qualquer tipo de organização de *software* para orientar suas atividades de melhoria.

Parte 8 - Guia de Orientação para Determinação da Capacidade do Processo

Esta parte descreve como definir entradas para a avaliação do processo e como usar os resultados da avaliação para o propósito de determinar a capacidade do processo. O guia especificamente direciona a determinação da capacidade do processo tanto para uso numa organização que deseja determinar os riscos associados pelo comprometimento com novo projeto, quanto pode ser usado por um comprador para avaliar fornecedores externos.

Parte 9 - Vocabulário

Esta parte contém um dicionário envolvendo todos os termos definidos especificamente para o propósito do modelo SPICE.

2.1.5 ISO 9000-3

A ISO 9000-3 (ISO 9000-3) é um guia para a aplicação da ISO 9001 (ISO 9001) no desenvolvimento, fornecido e manutenção de *software*. A Norma 9001 faz parte da série de normas ISO 9000, voltadas para a gestão e garantia da qualidade. Estas normas especificam os requisitos mínimos para que as empresas possam assegurar a qualidade de seus produtos e serviços, não definindo modelos ou impondo sistemas de qualidade a serem implementados nas organizações. As empresas definem seus próprios modelos de gestão de qualidade, dependendo do seu tipo de negócio e suas características.

A ISO 9001 é aplicável a empresas em geral que atuam em projeto, desenvolvimento, produção, instalação e assistência técnica. Como esses documentos da Série ISO 9000 são genéricos, foi necessária a elaboração de um documento complementar onde fossem abordados alguns aspectos específicos do *software*.

Assim, em 1993 foi criada a Norma ISO 9000-3 com diretrizes para aplicação da ISO 9001 ao desenvolvimento, fornecimento e manutenção de *software*. Para cada item da ISO 9001 existe um correspondente na ISO 9000-3 que o detalha e o adapta ao *software* (Lamprecht 1994). As diretrizes propostas na ISO 9000-3 destinam-se a descrever os controles e métodos para a produção de *software* que atendam aos requisitos do comprador, evitando-se não conformidades em todos os estágios, desde o desenvolvimento até a manutenção. A ISO 9000-3 é dividida em três partes: estrutura, atividades do ciclo de vida e atividades de suporte:

Estrutura

Descreve aspectos organizacionais, relacionados ao sistema de qualidade. São detalhadas as responsabilidades e ações relacionadas à qualidade que devem ser tomadas tanto pelo fornecedor quanto pelo comprador (ISO 9000-3). Os pontos são:

- Responsabilidade da Administração: o fornecedor: definir e documentar sua política e objetivos para a qualidade, e assumir o compromisso com estes. O comprador: cooperar com o fornecedor, a fim de prover todas as informações necessárias em tempo hábil e resolver itens pendentes.

- Sistema da Qualidade do fornecedor: estabelecer e manter documentado um sistema de qualidade. Este sistema deve ser um processo integrado por todo ciclo de vida. A prevenção de problemas deve ser enfatizada mais do que a correção.
- Auditorias Internas do Sistema da Qualidade (pelo fornecedor): implantar um sistema abrangente de auditorias internas de qualidade planejadas e documentadas, para verificar se as atividades de qualidade estão em conformidade com a forma planejada e para determinar a eficácia do sistema de qualidade.
- Ação Corretiva: o fornecedor deve estabelecer, documentar e manter procedimentos para investigar as causas do produto não conforme, detectar e eliminar caudas potenciais, iniciar ações preventivas para tratar dos problemas, aplicar controles para assegurar a realização das ações corretivas e implementar e registrar nos procedimentos resultantes de ações corretivas.

Atividades do Ciclo de Vida

Descreve as atividades de desenvolvimento de *software*. A Norma define que o desenvolvimento de *software* deve ser feito segundo um determinado modelo de ciclo de vida, e as atividades relacionadas à qualidade devem ser planejadas e implementadas de acordo com a natureza deste modelo utilizado (ISO 9000-3). Independentemente do modelo de ciclo de vida estabelecido pela organização, a Norma define que as atividades do ciclo de vida devem ser agrupadas em nove categorias:

- Análise Crítica do contrato: analisar cada contrato para assegurar que os objetivos, os requisitos e os riscos sejam definidos e documentados; que as informações reservadas sejam protegidas; e que o comprador tenha capacitação para atender os requisitos contratuais.
- Especificação dos Requisitos do Comprador: os requisitos devem incluir todos os aspectos necessários à satisfação das necessidades do comprador, como desempenho, segurança contra riscos, privacidade e confiabilidade. Tais requisitos devem ser estabelecidos de maneira precisa, o suficiente para permitir a validação durante a aceitação do produto.
- Planejamento de Desenvolvimento: definir um processo ou metodologia disciplinada, para transformar a especificação dos requisitos do comprador num produto de "*software*". Isto envolve a identificação de:

1. Fases de desenvolvimento a serem realizadas;
2. Entradas requeridas para cada fase;
3. Saídas requeridas para cada fase;
4. Procedimentos de verificação a serem executados em cada fase;
5. Análise dos problemas potenciais associados às fases de desenvolvimento e ao atendimento dos requisitos especificados.

- Planejamento da Qualidade: o fornecedor deve preparar um plano de qualidade como parte do planejamento de desenvolvimento. Este plano deve ser atualizado, ao longo da execução do desenvolvimento, deve conter análises críticas e ensaios, gestão de configuração e controle de alteração, e também controle de defeitos e ação corretiva.

- Projeto e Implementação: transformam as especificações dos requisitos do comprador num produto de "*software*". Estas atividades devem ser realizadas de maneira disciplinada, a fim de produzir um produto de acordo com a especificação, e não depender das atividades de ensaio e de validação para assegurar a qualidade.

- Ensaios e validação (pelo fornecedor): os resultados dos ensaios devem ser registrados, sendo que as áreas atingidas por quaisquer alterações devem ser identificadas e ensaiadas novamente. A adequação e pertinência dos ensaios devem ser avaliadas. Antes de libertar o produto, o fornecedor deve validar sua operação como um produto completo, quando possível, sob condições semelhantes às do ambiente de aplicação.

- Aceitação (pelo comprador): julgar se o produto é aceitável ou não, conforme os critérios previamente acordados. O método para tratar dos problemas detectados deve ser documentado.

- Cópia, Entrega e Instalação: a cópia é uma etapa que deve ser conduzida antes da entrega. A cópia deve conter versão, manuais e guias do usuário, definição e acordo quanto aos direitos autorais e licenças. Prescrições devem ser feitas para verificar a correção e completeza das cópias dos produtos de "*software*" entregues. Na instalação as funções, responsabilidades e obrigações do fornecedor e do comprador devem ser claramente estabelecidas.

- Manutenção: é estipulada no contrato, após a entrega e instalação inicial do produto de "*software*". É classificada em: resolução de problemas, modificação de interface e expansão funcional ou aprimoramento de desempenho.

Atividades de Suporte

Descreve as atividades que apóiam as atividades do ciclo de vida de desenvolvimento. Estão organizadas em nove itens: gestão de configuração; controle de documentos; registros da qualidade; medição; regras, práticas e convenções; ferramentas e técnicas; aquisição; produto de *software* incluído e treinamento.

- Gestão de configuração: fornecer um mecanismo para identificação, controle e acompanhamento das versões de cada item de "*software*".

- Controle de Documentos (pelo fornecedor): estabelecer e manter procedimentos para controlar todos os documentos relacionados ao conteúdo da Norma ISO 9001-3. Os documentos abrangem: aprovação e emissão de procedimentos; procedimentos de alteração; entradas e saídas da fase de desenvolvimento; planos e resultados de verificação e validação; documentação para o comprador e o usuário e documentação

- Registro da qualidade (pelo fornecedor): estabelecer e manter procedimentos para identificar, coletar, indexar, arquivar, armazenar, manter e dispor registros da qualidade.

- Medição: as medidas devem ser relatadas e utilizadas para gerenciar o processo de desenvolvimento e entrega, e devem ser pertinentes ao produto de "*software*" específico.

- Regras, Práticas e Convenções: o fornecedor deve suprir regras, práticas e convenções de modo a tomar efetivo o sistema da qualidade especificado na Norma ISO 9003. O fornecedor deve analisar criticamente estas regras, práticas e convenções e revisá-las, quando requerido.

- Ferramentas e Técnicas (pelo fornecedor): utilizar ferramentas, recursos e técnicas para tomar efetivas as diretrizes do sistema da qualidade. O fornecedor deve aprimorar as ferramentas e técnicas, quando requerido.

- Aquisição (pelo fornecedor): assegurar que um produto ou serviço adquirido está de acordo com os requisitos especificados.

- Produto de "*software*" incluído: a inclusão ou uso de produtos de "*software*" fornecidos pelo comprador, ou por terceira parte, pode ser requerido ao fornecedor. O fornecedor deve estabelecer e manter procedimentos para avaliação, armazenamento, proteção e manutenção de tais produtos.

- Treinamento (pelo fornecedor): estabelecer e manter procedimentos para identificar as necessidades de treinamento, e providenciá-lo para todo o pessoal que executa atividades que influenciam a qualidade.

3. NECESSIDADES VERIFICADAS NO ESTUDO DE CASO

Num ambiente de equipes de pequeno porte baseadas no desenvolvimento de *software* embarcado para produtos opto-eletrônicos (onde o *software*, apesar de sua importância, adquire a característica de ser um componentes do produto), o estudo de caso apontou vários aspectos que se diferenciam com relação a um ambiente onde o projeto de *software* resulte em um produto final.

É importante salientar que neste tipo de desenvolvimento, o *software* é tratado como um item importante e relevante no conjunto do projeto como um todo, porém, ele é apenas mais um item. Quase sempre, os projetos que relacionam este tipo de *software*, predomina-se a multidisciplinalidade, isto é, agregam várias áreas de conhecimento como: eletrônica, mecânica, óptica, etc.

Dessa forma, a equipe que irá desenvolver o projeto deve normalmente possuir um profissional de cada área. Por isto, a equipe de desenvolvimento de *software* é bastante reduzida, normalmente 2 ou 3 profissionais, porém não é difícil a existência de casos onde apenas 1 profissional é responsável por todo o planejamento e desenvolvimento do *software* agregado.

Este número reduzido de profissionais no desenvolvimento do *software* embarcado acarreta diferenças tanto no perfil do profissional para a realização do mesmo, como na metodologia ou seqüência de eventos que é utilizado para este desenvolvimento.

Com relação ao conhecimento do profissional, se faz necessário além do básico para o desenvolvimento de *software* (linguagens computacionais, sistemas operacionais, etc.), um conhecimento amplo nas áreas de eletrônica, mecânica e afim, pois em muitos projetos o hardware do equipamento tanto em nível de processamento, *design* e funcionalidade é desenvolvido paralelamente ao *software*. É de vital importância que os profissionais envolvidos no projeto consigam "conversar" num nível de linguagem que possa ser entendido por todos. Muitas vezes características do sistema computacional adotado pode afetar o projeto de hardware do equipamento.

Os *softwares* embarcados, por estarem vinculados a um hardware específico, não sofrem da necessidade de serem intercambiáveis (mudança de plataforma computacional) ou mesmo suscetíveis a versões muito amplas e inovadoras, como é o caso de um *software* de planilha eletrônica ou de visualização gráfica.

O equipamento desenvolvido tem uma classe de cliente específico, e dessa forma a partir do momento que este equipamento tem a aprovação de grupo de clientes, o *software* não mais sofrerá alterações. Pode em alguns casos ser adaptado para um dos clientes (modificação do idioma, acréscimo de alguma saída digital ou analógica, etc).

Neste ambiente de desenvolvimento citado acima, pode se denotar alguns pontos positivos, mas também pontos negativos de se trabalhar na criação de um *software* embarcado.

Os pontos positivos identificados são:

- Facilidade e rapidez no momento da alteração de rotinas ou do programa principal ainda no momento da criação do protótipo devido a uma imposição do hardware ou melhoria no desempenho do *software*. Com uma equipe pequena onde o profissional participou do planejamento do hardware, e muitas vezes foi o único a desenvolver o *software*, este conhecimento de que necessita para as modificações estão bastante "vivas" para ele. Este ponto foi considerado positivo, porém embutindo riscos de perda de informações.

- Muitas das rotinas desenvolvidas (módulos) para um equipamento da empresa, pode ser aproveitada no *software* embarcado de outro produto. O nível de aproveitamento destes módulos é bastante grande, já que este profissional ao participar do projeto de hardware de um outro produto, pode contribuir para o projeto de um hardware que possa sem sair das novas características do produto em si, fazer uso das rotinas que ele anteriormente projetou.

- Baixo custo e prazos relativamente curtos de desenvolvimento, com a utilização de módulos intercambiáveis.

Pontos negativos identificados:

- Dificuldades para migração para novas plataformas de hardware e de *software*. Os projetistas preferem sempre utilizar, ou melhor, reutilizar partes do sistema já desenvolvidos no passado. Dificultando as inovações, muitas vezes o projeto não é otimizado devido a posição dos projetistas de quererem sempre utilizar o que já está funcionando. Seja por comodidade ou mesmo por não quererem abrir mão de sistemas já validados. O controle de projeto, com a geração de capacidade de transferir requisitos de entrada e saída de etapas do projeto como definido na ISO 9001 torna-se aqui uma implementação necessária.

- Dificuldade de manter atualizado o controle de documentação para posterior manutenção. Devido a ser um grupo pequeno, onde muitas vezes a manutenção ou mesmo as melhorias são feitas pelo mesmo profissional, estes não se sente motivado para manter a documentação atualizada, ficando dessa forma muitos detalhes apenas no conhecimento deste profissional. Na ausência do mesmo, o tempo gasto para se fazer uma modificação ou mesmo a qualidade para uma melhor otimização desta modificação ficam muito a desejar. Verifica-se que, portanto, o controle de documentos e dados é um item importante para a fixação dos recursos, características, do conhecimento gerado e do controle do desenvolvimento.

- Por ser uma equipe pequena, e o profissional que desenvolverá o *software* o fará em geral sozinho, o projeto poderá refletir a experiência e os critérios adotado pelo mesmo. Corre-se o risco, muitas vezes pela falta de experiência do profissional, ou o pouco conhecimento das várias formas de se proceder ao desenvolvimento, de não ser a melhor opção aquela adotada

por este profissional e dessa forma esta opção prejudicar o hardware do equipamento, contaminando-o com o projeto falho do *software* embarcado. Neste caso o processo corre o risco de não possuir a repetibilidade suficiente para garantir que os resultados sejam minimamente previsíveis.

4. CONCLUSÕES

Com base nos aspectos positivos e negativos apontados, conclui-se que a adoção do PSP na íntegra não atende às necessidades de uma organização com as características apontadas e aos requisitos exigidos em produtos do tipo estudado. A implementação na íntegra de um modelo baseado na ISO, SPICE ou CMM seria uma situação ideal em casos de necessidade de "certificação" do sistema para possibilitar a comercialização do produto ou para a implementação de uma estrutura da qualidade que incorpore todo o processo de desenvolvimento e não apenas o *software*. Assim, a adoção de requisitos mistos derivados dos modelos é recomendável para gerar as condições mínimas para uma organização eficaz no desenvolvimento de *software* embarcado, entre as quais pode-se recomendar:

- A implementação de requisitos de Controle de Projeto (ISO): garantir a troca e a transferência dos módulos e requisitos do *software* entre projetos diferentes ou no mesmo projeto;
- Controle de Documentos e Dados (ISO): garantir o registro, a identificação e o controle de documentos, permitindo a rastreabilidade e prospecção de informações geradas, evitando-se repetição de erros e o aproveitamento do conhecimento adquirido;
- Análise Crítica de Contrato (ISO): garantir a plena incorporação dos requisitos dos clientes ou dos usuários ao *software* como componente da boa funcionalidade e eficiência do equipamento;
- Atingir como meta e implementar "no mínimo" os requisitos Nível 2 do CMM - registrando as informações e garantindo certa repetibilidade no processo de desenvolvimento;
- Implementar a auditoria interna (ISO) como ferramenta de melhoria de processo e
- Aplicar os conceitos de indicadores do Síncrono-Estabilidade para identificar gargalos e monitorar a evolução do produto e a melhoria de processo. Pode-se recomendar as seguintes métricas iniciais: medição de tempo de desenvolvimento, quantidade de linhas de código e indicadores de evolução de quantidade de falhas. É recomendável que haja validação do *software* em versões beta através de grupos pequenos de potenciais usuários antes de lançar o produto no mercado;
- É recomendável a prática de designar um "testador" para cada módulo independente do profissional ou equipe desenvolvedora. Os testes devem ser registrados.

REFERÊNCIAS BIBLIOGRÁFICAS

BARRETO, J. Qualidade de *Software*. **http://www.barreto.com.br/qualidade**. Acesso: 15/03/1998.

BASILI, V. R.; ROMBACH, H. D.: **Tailoring the Process to project Goals and Environments**. IEEE Computer Society, March, pg. 345-357, 1987.

BASILI, V. R.; ROMBACH, H. D.: **Tailoring the Process to project Goals and Environments**. IEEE Computer Society, March, pg. 345-357, 1987.

CLARK, K. B.; FUJIMOTO, T. **Product Development Performance: strategy, organization and management in the world auto industry**. Boston School Press, 1991

CURTIS, B.: **Manteining the *Software* Process**. IEEE Conference on *Software* Maintenance. Orlando, pg. 2-8, 1992

CUSUMANO, M. A: The **Japanese Automobile Industry: technology and management at Nissan and Toyota**. Harvard University Press, 1985

CUSUMANO, M. A; SELBY, R. W.:.**Microsoft Secrets**. Simon & Shuster, New York, capítulos 3, 4, 5 e 6, 1995.

GARVIN, D. A.: **Managing Quality**. Free Press, New York, 1988.

HAYES, R. H.; WHEELWRIGHT, S. C.; CLARK K. B.: **Dynamic Manufacturing**. Free Press, New York, 1988.

HUMPHREY, W. S.: **Characterizing the *Software* Process: A Maturity Framework**. IEEE *Software*, March, 1988.

HUMPHREY, W. S.: **Managing the *Software* Process.** Primeira Edição, Massachussets. Addison-Wesley, 1989.

ISO 9001: **Quality Systems: Model for Quality Assurance in *Design*, Development, Production, Installation and Servicing**, 1994.

ISO 9000-3: **Quality Management and Quality Assurance Standards. Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of *Software***, 1991.

ISO/IEC 9126: **Information Technology. *Software* Product Evaluation. Quality Characteristics and Guidelines for their use**, 1991

ISO/IEC 12119 : **Information Technology *Software* Packages – Quality Requirements and Testing**, 1994.

ISO/IEC 12119: **Information Technology *Software* Life-Cicle Process**, 1995.

ISO/IEC 14598: ***Software* Product Evaluation** (Draft), 1995.

JONES,C.: **Critical Problems in *Software* Measurement,** versão 1.0, Burlington, Massachussets. *Software* Product Research, Agosto, 1992

LAMPRECHT, J. L.: **ISO 9000 e o Setor de Serviços: uma interpretação crítica da revisão 1994**, Rio de Janeiro, Qualitymark, pg. 237-253, 1994.

LIGHTFOOT, T. A Framework of the SPICE, 1995 **http://www.sqi.citgu.edu.au/spice/general/spiceovr.html**, acesso: 02/02/97.

LIGHTFOOT, T. An Introduction to SPICE Documents, 1995, **http://www.sqi.citgu.edu.au/spice/general/spice.html**, acesso: 02/02/97.

LUCENA, C. J. P.: **Tecnologias de *Software*: tendências e interesses para o Brasil**. In: Seminário Avançado de *Software*. CITS, Curitiba, Outubro, 1991.

McCALL, J. P. R.: **Factors in *Software* Quality**, pg. 014, 015, 055, Boston Press, Boston, 1977.

MAcFARLANE, M.: **Eating the Elephant One Byte at a Time: effective Implementation of ISO 9001/Tick.** -IT.ISO/IEC 12119 1994: **Information Technology *Software* Life-Cicle Processes**, 1995.

McGARRY, F. et al: ***Software* Improvement in the NASA *Software* Engeneering Laboratory.** Technical Report CMU/SEI-94-TR-22, 1994

OLAVARRIETA, A. D. A brief description of the project SPICE, 1996, **http://www.serinf.com/ article.html**, acesso: 05/02/97.

PAULK, M. C. et al: **Key Practices of the Capability Maturity Model, Version 1.1.** Technical Report, CMU/SEI-93-TR-025, Fevereiro, 1993.

PAULK, M. C.: **Comparision of ISO 9001 and CMM.** IEEE *Software*, pg. 74-82, Janeiro, 1995

PRESSMAN, R. S.: ***Software* Engeneering - A Practitioner's Approach.** McGrall-Hill Book Co., 1994

ROSENBLOOM, R. S.; CUSUMANO, M. A.: **Technological pioneering and competitive advantage: the birth of VCR.** California Management Review. V.29, n.9, pg. 51-76, 1987

ROSA, P. G.: **Avaliação de Processo de *Software* para a Pequena Empresa.** Dissertação de Mestrado. ICMSC, USP, São Carlos, 1997.

SOLINGEN, R.: **Goal Oriented *software* measurement in practice – introducing *software* measurement in Schlumberger Retail Petroleum Systems.** Master Thesis, Schlumberger, 1995.

Software Business: **O mercado cresce.** pg. 2. Softex- Programa Nacional de *Software* para Exportação, São Paulo, 1996.

ISO/IEC SPICE's Project: Part 1 to 9 – Concepts and introdutory guide, 1997, **http://www-sqi.cit.gu.edu.au/spice**, acesso: 15/12/97.

TSUKUMO, A. N. et al.: **ISO/IEC 9126: An experiment of application on brazilian *software* products.** In: 2nd International *Software* Engeneering Standards Symposium (ISESS'95). Montreal, Agosto, 1995.

WHEELWRIGHT, S. G.; CLARK, K. B.: **Revolutionizing product development.** Free Press, New York, 1992.