

5 GARANTIA E CONTROLE DE QUALIDADE

5.1 INTRODUÇÃO À QUALIDADE DE SOFTWARE

Origem: Wikipédia, a enciclopédia livre.

A **Qualidade de Software** é uma área de conhecimento da Engenharia de Software que objetiva garantir a qualidade do software através da definição e normatização de processos de desenvolvimento. Apesar dos modelos aplicados na garantia da qualidade de software atuarem principalmente no processo, o principal objetivo é garantir um produto final que satisfaça às expectativas do cliente, dentro daquilo que foi acordado inicialmente.

Segundo a norma ISO 9000 (versão 2000), a qualidade é o grau em que um conjunto de características inerentes a um produto, processo ou sistema cumpre os requisitos inicialmente estipulados para estes.

No desenvolvimento de software, a qualidade do produto está diretamente relacionada à qualidade do processo de desenvolvimento, desta forma, é comum que a busca por um software de maior qualidade passe necessariamente por uma melhoria no processo de desenvolvimento.

Rodney Brooks, diretor do Laboratório de Inteligência Artificial e Ciência da Computação do MIT, define qualidade como a conformidade aos requisitos. Essa definição exige determinar dois pontos: I) o que se entende por conformidade; e II) como são especificados - e por quem - os requisitos.

5.1.1 PRINCIPAIS TÓPICOS

Para um melhor entendimento e estudo, o SWEBOK divide a **Qualidade de Software** em três tópicos, cada tópico é subdividido em atividades, da seguinte forma:

- Fundamentos de Qualidade de Software
 - Cultura e Ética de Engenharia de Software
 - Valores e Custos de Qualidade
 - Modelos e Características de Qualidade
 - Melhoria da Qualidade
- Gerência do Processo de Qualidade de Software
 - Garantia de Qualidade de Software
 - Verificação e Validação
 - Revisões e Auditorias
- Considerações Práticas
 - Requisitos de Qualidade para Aplicações
 - Caracterização de Defeitos
 - Técnicas de Gerência de Qualidade de Software
 - Medidas de Qualidade de Software

Ainda segundo o SWEBOK, a **Qualidade de Software** é um tema tão importante que é encontrado, de forma ubíqua, em todas as outras áreas de conhecimento envolvidas em um projeto. Além disso, ele deixa claro que essa área, como nele definida, trata do aspectos *estáticos*, ou seja, daqueles que não exigem a execução do software para avaliá-lo, em contraposição à área de conhecimento Teste de software.

Porém, é normal que se encontrem autores e empresas que afirmam serem os testes de software uma etapa da **Qualidade de Software**.

5.1.2 REQUISITOS DE QUALIDADE

Requisitos de Qualidade é um tópico por si dentro do assunto Qualidade. Dentro da ótica desta última, espera-se que os requisitos sejam definidos de maneira a caracterizar completamente o produto a ser construído. Nesse aspecto - e em relação à definição de Brooks - é evidente que as zonas de sombra dentro de uma especificação abrem margem a todo tipo de problemas de avaliação de produtos.

Sommerville distingue requisitos funcionais e não funcionais. O modelo internacional mais recente Square, estabelecido pela norma ISO 25000, adota uma classificação um pouco diferente e utiliza uma descrição hierárquica. Dentro dessa descrição, "funcionalidade" é uma das seis divisões iniciais em que se classificam os requisitos de um produto de software.

Idealmente, a especificação de requisitos deve permitir que o processo de fabricação do software seja controlado. Isso significa que idealmente a qualidade de produtos intermediários deve poder ser mensurada e que os dados obtidos devem trazer informação que possa levar ao controle de desvios, localização de defeitos e outras ocorrências negativas.

Para permitir a avaliação de produtos finais e de produtos intermediários, a especificação de requisitos deve, idealmente, especificar qualitativa e quantitativamente os objetivos a serem atingidos. Esse objetivo é bastante desafiador. As experiências iniciais indicam que cada empresa deve construir sua base de dados própria, a partir da qual é possível refinar as especificações de novos produtos. Esse carácter localizado dos dados se explica pelo fato de que não há especificações universais de qualidade de software. Os requisitos variam caso a caso, não apenas em função dos clientes mas também dos critérios utilizados pela própria empresa fabricante.

5.1.3 O PROCESSO DE SOFTWARE

Nas últimas décadas foram propostas dezenas de metodologias e processos adaptados a diferentes cenários e produtos. Embora se possa justificar essa multiplicidade por outra lei de Brooks - a ausência de "balas de prata", é um fato que a situação se mostra confusa.

Há dezenas de trabalhos propostos para casos particulares. Exemplos das diversas iniciativas para tratar o assunto são metodologias como XP e Scrum; o modelo CMM, seguido de toda uma série de adaptações (como SW-CMM, people-CMM, etc.), mais tarde substituído pelo modelo CMMI; e dezenas de artigos e teses de mestrado e doutorado, abordando tópicos particulares em um ou mais de tais métodos, ou propondo ainda novas adaptações a casos particulares.

A situação deixa evidente que há um vácuo a ser preenchido - atacar a raiz do problema e identificar uma estrutura suficientemente geral, capaz de explicar o problema de qualidade e ser adaptada a todos os cenários diferentes. Se tal objetivo é possível resta a ser provado - assunto para novos artigos e teses.

5.1.4 GARANTIA DE QUALIDADE DE SOFTWARE

A Garantia da Qualidade de Software (GQS) é a área-chave de processo do CMM cujo objetivo é fornecer aos vários níveis de gerência a adequada visibilidade dos projetos, dos processos de desenvolvimento e dos produtos gerados. A GQS atua como "guardiã", fornecendo um retrato do uso do Processo e não é responsável por executar testes de software ou inspeção em artefatos.

Obtendo a visibilidade desejada, a gerência pode atuar de forma pontual no sentido de atingir os quatro grandes objetivos de um projeto de desenvolvimento de software, quais sejam, desenvolver software de alta qualidade, ter alta produtividade da equipe de desenvolvimento, cumprir o cronograma estabelecido junto ao cliente e não necessitar de recursos adicionais não previstos.

Para conseguir esses objetivos a área-chave de processo GQS estimula a atuação das equipes responsáveis pelo desenvolvimento de software em diversas frentes objetivando internalizar comportamentos e ações, podendo-se destacar:

- O planejamento do projeto e o acompanhamento de resultados;
- O uso dos métodos e ferramentas padronizadas na organização;

- A adoção de Revisões Técnicas Formais;
- O estabelecimento e a monitoração de estratégias de testes;
- A revisão dos artefatos produzidos pelo processo de desenvolvimento;
- A busca de conformidade com os padrões de desenvolvimento de software;
- A implantação de medições associadas a projeto, processo e produto;
- A utilização de mecanismos adequados de armazenamento e recuperação de dados relativos a projetos, processos e produtos; e
- A busca de uma melhoria contínua no processo de desenvolvimento de software.

Para facilitar o trabalho dos desenvolvedores e evitar geração de metodologias diversas, o Serpro desenvolveu o Processo Serpro de Desenvolvimento de Soluções (PSDS).

O PSDS foi construído por pessoas das unidades da empresa que procuraram aproveitar as melhores práticas existentes e consagradas.

O "CMM - Capability Maturity Model for Software /SEI" é uma estrutura-"framework", que descreve os principais elementos de um processo de desenvolvimento de software efetivo. O CMM descreve os estágios de maturidade através dos quais Organizações de software evoluem o seu ciclo de desenvolvimento de software através de sua avaliação contínua, identificação e ações corretivas dentro de uma estratégia de melhoria dos processos. Este caminho de melhoria é definido por cinco níveis de maturidade: inicial, repetitivo, definido, gerenciado e otimizado.

O Modelo CMM (CMM- Capability Maturity Model) fornece às organizações uma direção sobre como ganhar controle de seu processo de desenvolvimento de software e como evoluir para uma cultura de excelência na gestão de software. O objetivo principal nas transações destes níveis de maturidade é a realização de um processo controlado e mensurado como a fundação para melhoria contínua. Cada nível de maturidade possui um conjunto de práticas de software e gestão específicas, denominado áreas-chave do processo. Estas devem ser implantadas para a organização atingir o nível de maturidade em qualidade de software.

5.2 DEFINIÇÕES E CONCEITOS

Origem: Apostila sobre Engenharia de Software, site: Apostilando. PROF. VITÓRIO BRUNO MAZZOLA. Capítulo 2 – Qualidade de Software

1. INTRODUÇÃO

Como foi mencionado no capítulo anterior, o papel da Engenharia de Software é, principalmente, fornecer métodos e ferramentas para o desenvolvimento do software de qualidade e a baixo custo.

O fator qualidade é um dos aspectos importantes que deve ser levado em conta quando do desenvolvimento do software. Para isto, é necessário que se tenha uma definição precisa do que é um software de qualidade ou, pelo menos, quais são as propriedades que devem caracterizar um software desenvolvido segundo os princípios da Engenharia de Software.

Um outro aspecto importante é aquele relacionado à avaliação e ao aprimoramento do processo de desenvolvimento de software de uma organização. Nesta linha, foi desenvolvido pelo SEI (*Software Engineering Institute*) um modelo que permite definir parâmetros para a análise desta questão nas corporações, o modelo CMM (*Capability and Maturity Model*), cujas linhas gerais serão descritas na seção 5.

2. DEFINIÇÃO DE SOFTWARE DE QUALIDADE

Primeiramente, é importante discutir o conceito de software de qualidade. Segundo a Associação Francesa de Normalização, AFNOR, a qualidade é definida como "a capacidade de um produto ou serviço de satisfazer às necessidades dos seus usuários".

Esta definição, de certa forma, é coerente com as metas da Engenharia de Software, particularmente quando algumas definições são apresentadas. É o caso das definições de Verificação e Validação introduzidas por Boehm, que associa a estas definições as seguintes questões:

- **Verificação:** "Será que o produto foi construído corretamente?"
- **Validação:** "Será que este é o produto que o cliente solicitou?"

O problema que surge quando se reflete em termos de qualidade é a dificuldade em se quantificar este fator.

2.1. Fatores de Qualidade Externos e Internos

Algumas das propriedades que poderíamos apontar de imediato são a correção, a facilidade de uso, o desempenho, a legibilidade, etc... Na verdade, analisando estas propriedades, é possível organizá-las em dois grupos importantes de fatores, que vamos denominar fatores externos e internos.

Os fatores de qualidade **externos**, são aqueles que podem ser detectados principalmente pelo cliente ou eventuais usuários. A partir da observação destes fatores, o cliente pode concluir sobre a qualidade do software, do seu ponto de vista. Enquadram-se nesta classe fatores tais como: o desempenho, a facilidade de uso, a correção, a confiabilidade, a extensibilidade, etc...

Já os fatores de qualidade **internos** são aqueles que estão mais relacionados à visão de um programador, particularmente aquele que vai assumir as tarefas de manutenção do software. Nesta classe, encontram-se fatores como: modularidade, legibilidade, portabilidade, etc...

Não é difícil verificar que, normalmente, os fatores mais considerados quando do desenvolvimento do software são os externos. Isto porque, uma vez que o objetivo do desenvolvimento do software é satisfazer ao cliente, são estes fatores que vão assumir um papel importante na avaliação do produto (da parte do cliente, é claro!!!). No entanto, também não é difícil concluir que são os fatores internos que vão garantir o alcance dos fatores externos.

2.2. Fatores de Qualidade

2.2.1. Correção

É a capacidade dos produtos de software de realizarem suas tarefas de forma precisa, conforme definido nos requisitos e na especificação. É um fator de suma importância em qualquer categoria de software. Nenhum outro fator poderá compensar a ausência de correção. Não é interessante produzir um software extremamente desenvolvido do ponto de vista da interface homem-máquina, por exemplo, se as suas funções são executadas de forma incorreta. É preciso dizer, porém, que a correção é um fator mais facilmente afirmado do que alcançado. O alcance de um nível satisfatório de correção vai depender, principalmente, da formalização dos requisitos do software e do uso de métodos de desenvolvimento que explorem esta formalização.

2.2.2. Robustez

A robustez é a capacidade do sistema funcionar mesmo em condições anormais. É um fator diferente da correção. Um sistema pode ser correto sem ser robusto, ou seja, o seu funcionamento vai ocorrer somente em determinadas condições. O aspecto mais importante relacionado à robustez é a obtenção de um nível de funcionamento do sistema que suporte mesmo situações que não foram previstas na especificação dos requisitos. Na pior das hipóteses, é importante garantir que o software não vai provocar consequências catastróficas em situações anormais. Resultados esperados são que, em tais situações, o software apresente comportamentos tais como: a sinalização da situação anormal, a terminação "ordenada", a continuidade do funcionamento "correto" mesmo de maneira degradada, etc... Na literatura, estas características podem ser associadas também ao conceito de **confiabilidade**.

2.2.3. Extensibilidade

É a facilidade com a qual se pode introduzir modificações nos produtos de software. Todo software é considerado, em princípio, "flexível" e, portanto, passível de modificações. No entanto, este fator nem sempre é muito bem entendido, principalmente quando se trata de pequenos programas. Por outro lado, para softwares de grande porte, este fator atinge uma importância considerável, e pode ser atingido a partir de dois critérios importantes:

- a **simplicidade de projeto**, ou seja, quanto mais simples e clara a arquitetura do software, mais facilmente as modificações poderão ser realizadas;
- a **descentralização**, que implica na maior autonomia dos diferentes componentes de software, de modo que a modificação ou a retirada de um componente não implique numa reação em cadeia que altere todo o comportamento do sistema, podendo inclusive introduzir erros antes inexistentes.

2.2.4. Reusabilidade

É a capacidade dos produtos de software serem reutilizados, totalmente ou em parte, para novas aplicações. A necessidade vem da constatação de que muitos componentes de software obedecem a um padrão comum, o que permite então que estas similaridades possam ser exploradas para a obtenção de soluções para outras classes de problemas.

Este fator permite, principalmente, atingir uma grande economia e um nível de qualidade satisfatórios na produção de novos softwares, dado que menos programa precisa ser escrito, o que significa menos esforço e menor risco de ocorrência de erros. Isto significa de fato que a reusabilidade pode influir em outros fatores de qualidade importantes, tais como a correção e a robustez.

2.2.5. Compatibilidade

A compatibilidade corresponde à facilidade com a qual produtos de software podem ser combinados com outros. Este é um fator relativamente importante, dado que um produto de software é construído (e adquirido) para trabalhar convivendo com outros softwares. A impossibilidade de interação com outros produtos pode ser, sem dúvida, uma característica que resultará na não escolha do software ou no abandono de sua utilização. Os contraexemplos de compatibilidade, infelizmente, são maiores que os exemplos, mas podemos citar, nesta classe, principalmente, a definição de formatos de arquivos padronizados (ASCII, PostScript, ...), a padronização de estruturas de dados, a padronização de interfaces homem-máquina (sistema do Macintosh, ambiente Windows, ...), etc...

2.2.6. Eficiência

A eficiência está relacionada com a utilização racional dos recursos de hardware e de sistema operacional da plataforma onde o software será instalado. Recursos tais como memória, processador e co-processador, memória cache, recursos gráficos, bibliotecas (por exemplo, primitivas de sistema operacional) devem ser explorados de forma adequada em espaço e tempo.

2.2.7. Portabilidade

A portabilidade consiste na capacidade de um software em ser instalado para diversos ambientes de software e hardware. Esta nem sempre é uma característica facilmente atingida, devido principalmente às diversidades existentes nas diferentes plataformas em termos de processador, composição dos periféricos, sistema operacional, etc...

Alguns softwares, por outro lado, são construídos em diversas versões, cada uma delas orientada para execução num ambiente particular. Exemplos disto são alguns programas da Microsoft, como por exemplo o pacote Microsoft Office, que existe para plataformas Macintosh e microcomputadores compatíveis IBM.

2.2.8. Facilidade de uso

Este fator é certamente um dos mais fortemente detectados pelos usuários do software. Atualmente, com o grande desenvolvimento dos ambientes gráficos como Windows, X-Windows e o Sistema Macintosh, a obtenção de softwares de fácil utilização tornou-se mais freqüente. A adoção de procedimentos de auxílio em linha (help on line) é, sem dúvida, uma grande contribuição neste sentido. Dada a definição de software feita no início do curso, porém, não se pode deixar de registrar a importância de uma documentação adequada (manual de usuário) capaz de orientar o usuário em sua navegação pelo software considerado.

3. A METROLOGIA DA QUALIDADE DO SOFTWARE

Apresentados alguns fatores de qualidade de software, a dificuldade que se apresenta é como medir a qualidade do software. Ao contrário de outras disciplinas de engenharia, onde os produtos gerados

apresentam características físicas como o peso, a altura, tensão de entrada, tolerância a erros, etc..., a medida da qualidade do software é algo relativamente novo e alguns dos critérios utilizados são questionáveis.

A possibilidade de estabelecer uma medida da qualidade é um aspecto importante para a garantia de um produto de software com algumas das características definidas anteriormente.

O Metrologia do Software corresponde ao conjunto de teorias e práticas relacionadas com as medidas, a qual permite estimar o desempenho e o custo do software, a comparação de projetos e a fixação dos critérios de qualidade a atingir.

As medidas de um software podem ser as mais variadas, a escolha da medida devendo obedecer a determinados critérios: a pertinência da medida (interpretabilidade); o custo da medida (percentual reduzido do custo do software); utilidade (possibilidade de comparação de medidas).

As medidas de um software podem ser organizadas segundo duas grandes categorias:

3.1. Medidas dinâmicas

São as medidas obtidas mediante a execução do software, como, por exemplo, os testes, as medidas de desempenho em velocidade e ocupação de memória, os traços, etc...

Estas medidas podem assumir um interesse relativamente grande pois elas permitem quantificar fatores que podem implicar em retorno econômico ou que representem informações sobre a correção do programa; por outro lado, estas medidas só podem ser obtidas num momento relativamente tardio do ciclo de desenvolvimento de um software, ou seja, a partir da etapa de testes.

3.2. Medidas estáticas

São aquelas obtidas a partir do documento fonte do software, sem a necessidade de execução do software. Dentre as medidas estáticas, podemos destacar:

- As medidas de complexidade textual, a qual é baseada na computação do número de operadores e do número de operandos contidos no texto do software;
- A complexidade estrutural, a qual se baseia na análise dos grafos de controle associadas a cada componente do software;
- As medidas baseadas no texto, como o tamanho do programa, a taxa de linhas de comentários, etc...

4. QUESTÕES IMPORTANTES À QUALIDADE DE SOFTWARE

Uma vez que alguns parâmetros relacionados à qualidade foram detectados na seção 2.2, cabe aqui discutir alguns princípios que devem fazer parte das preocupações de uma equipe de desenvolvimento de software...

4.1. O princípio do uso: o software deverá ser utilizado por outros

Este princípio implica num cuidado em tornar o software acessível não apenas para o usuário imediato, mas também para futuros programadores que irão trabalhar no código fonte, seja para eliminação de erros, seja para introdução ou aprimoramento de funções.

Com base nestes princípios, o programador ou a equipe de programação deverá prover a necessária flexibilidade e robustez ao programa desde o início do desenvolvimento e não inserir estes aspectos à medida que os problemas vão surgindo.

Um outro ponto importante é que muitas vezes um programador julga que determinadas soluções encontradas a nível de um programa são tão específicas que mais ninguém, incluindo o próprio autor, irá utilizar tal solução novamente. As diversas experiências mostram que isto podem ser um grave erro de avaliação, pois conduz, na maioria dos casos, à geração de código incompreensível, representando um grande obstáculo à reutilização.

Algumas providências no desenvolvimento de um software suportam o respeito a este princípio:

- raciocinar, desde o início do desenvolvimento, em termos de um software público; isto vai proporcionar uma economia relativamente grande no que diz respeito ao tempo necessário para a depuração ou modificação de partes do código;

- documentar suficientemente o programa, o que vai permitir uma economia em horas de explicação a outras pessoas de como funciona o programa ou como ele deve ser utilizado;
- projetar o software para que ele seja utilizável por iniciantes;
- rotular todas as saídas, salvar ou documentar todas as entradas, o que pode facilitar o trabalho de interpretação de resultados obtidos durante a execução do software.

4.2. O princípio do abuso: o software será utilizado de maneira indevida

Este princípio diz respeito às manipulações incorretas que os usuários imprimem ao programa, seja por desconhecimento do modo correto de utilização, seja por simples curiosidade. Exemplos destas manipulações são a manipulação de arquivos de entrada de um programa nas mais diversas condições (arquivos vazios, arquivos binários, arquivos muito grandes, etc...) ou incorreções sintáticas. Muitas vezes, entradas sintaticamente corretas podem provocar erros de execução (valores fora de faixa, erros de overflow, divisão por zero, etc...).

Um outro problema é a tentativa, por parte de usuários, de utilização de um dado software para uma outra finalidade que não aquela para a qual o software foi desenvolvido.

Para levar em conta este princípio, os cuidados que o programador deve ter são os seguintes:

- garantir que o software ofereça alguma saída satisfatória para qualquer tipo de entrada;
- evitar que o programa termine de forma anormal;
- chamar a atenção para entradas alteradas ou saídas incorretas.

4.3. O princípio da evolução: o software será modificado

Este é outro aspecto de importância no desenvolvimento de software, o qual está fortemente ligado com o aspecto da facilidade de manutenção (ou extensibilidade). É fato reconhecido que os softwares deverão sofrer modificações, estas pelas razões mais diversas; seja devido à detecção de um erro residual do desenvolvimento; seja por novos requisitos surgidos após a instalação do software.

Isto torna evidente a necessidade de desenvolver o projeto e o código de modo a facilitar as modificações necessárias. A existência de um código bem estruturado e documentado é pelo menos 50% do trabalho resolvido.

Dentro deste princípio, os cuidados a serem tomados deverão ser:

- o desenvolvimento de programas com bom nível de legibilidade;
- a estruturação em componentes de software facilmente modificáveis;
- agrupar alterações visíveis ao usuário e eventuais correções em versões numeradas;
- manter a documentação atualizada.

4.4. O princípio da migração: o software será instalado em outras máquinas

Este último princípio leva em conta o fato de que a vida de muitos softwares deve ultrapassar a vida da própria máquina em que ele está executando. A velocidade com a qual as arquiteturas de computador têm evoluído nos últimos anos acentua a importância deste princípio.

Um problema relacionado a este princípio é que às vezes os programas são desenvolvidos explorando algumas particularidades das arquiteturas das máquinas para as quais eles estão sendo concebidos. Isto cria uma forte dependência do hardware, o que dificulta a futura migração para outras arquiteturas.

Isto significa, na verdade, que para um programa apresentar um nível satisfatório de portabilidade, o programador deve evitar amarrar-se a detalhes de hardware da máquina ao invés de "aproveitá-los".

Assim, podemos sintetizar as ações a serem preparadas com relação a este princípio:

- pensar os programas como meio para solucionar problemas e não para instruir determinado processador;
- utilizar as construções padronizadas das linguagens de programação ou sistemas operacionais;

- isolar e comentar todos os aspectos do programa que envolvam a dependência do hardware.

5. O MODELO CMM

A causa mais comum do insucesso dos projetos de desenvolvimento de software é a má utilização ou a completa indiferença aos métodos e ferramentas orientados à concepção. É possível que, em empresas de desenvolvimento de software onde o uso de metodologias consistentes não seja prática adotada, os projetos possam ter bons resultados. Mas, em geral, estes bons resultados são muito mais uma consequência de esforços individuais do que propriamente causadas pela existência de uma política e de uma infra-estrutura adequada à produção de software.

O modelo CMM — *Capability Maturity Model* — foi definido pelo SEI — *Software Engineering Institute* — com o objetivo de estabelecer conceitos relacionados aos níveis de maturidade das empresas de desenvolvimento de software, com respeito ao grau de evolução que estas se encontram nos seus processos de desenvolvimento.

O modelo estabelece também que providências as empresas podem tomar para aumentarem, gradualmente o seu grau de maturidade, melhorando, por consequência, sua produtividade e a qualidade do produto de software.

5.1. Conceitos básicos do CMM

Um Processo de Desenvolvimento de Software corresponde ao conjunto de atividades, métodos, práticas e transformações que uma equipe utiliza para desenvolver e manter software e seus produtos associados (planos de projeto, documentos de projeto, código, casos de teste e manuais de usuário). Uma empresa é considerada num maior grau de maturidade quanto mais evoluído for o seu processo de desenvolvimento de software.

A Capabilidade de um processo de software está relacionada aos resultados que podem ser obtidos pela sua utilização num ou em vários projetos. Esta definição permite estabelecer uma estimação de resultados em futuros projetos.

O Desempenho de um processo de software representa os resultados que são correntemente obtidos pela sua utilização. A diferença básica entre estes dois conceitos está no fato de que, enquanto o primeiro, está relacionado aos resultados “esperados”, o segundo, relaciona-se aos resultados que foram efetivamente obtidos.

A Maturidade de um processo de software estabelece os meios pelos quais ele é definido, gerenciado, medido, controlado e efetivo, implicando num potencial de evolução da capacidade. Numa empresa com alto grau de maturidade, o processo de desenvolvimento de software é bem entendido por todo o staff técnico, graças à existência de documentação e políticas de treinamento, e que este é continuamente monitorado e aperfeiçoado por seus usuários.

À medida que uma organização cresce em termos de maturidade, ela institucionaliza seu processo de desenvolvimento de software através de políticas, normas e estruturas organizacionais, as quais geram uma infra-estrutura e uma cultura de suporte aos métodos e procedimentos de desenvolvimento.

5.2. Níveis de maturidade no processo de desenvolvimento de software

O modelo CMM define cinco níveis de maturidade no que diz respeito ao processo de desenvolvimento de software adotado a nível das empresas, estabelecendo uma escala ordinal que conduz as empresas ao longo de seu aperfeiçoamento.

Um nível de maturidade é um patamar de evolução de um processo de desenvolvimento de software, correspondendo a um degrau na evolução contínua de cada organização. A cada nível corresponde um conjunto de objetivos que, uma vez atingidos, estabilizam um componente fundamental do processo de desenvolvimento de software, tendo como consequência direta o aumento da capacidade da empresa.

A figura 2.1 apresenta os cinco níveis de maturidade propostos no modelo CMM, na qual se pode observar também o estabelecimento de um conjunto de ações que permitirão a uma empresa subir de um degrau para o outro nesta escala.

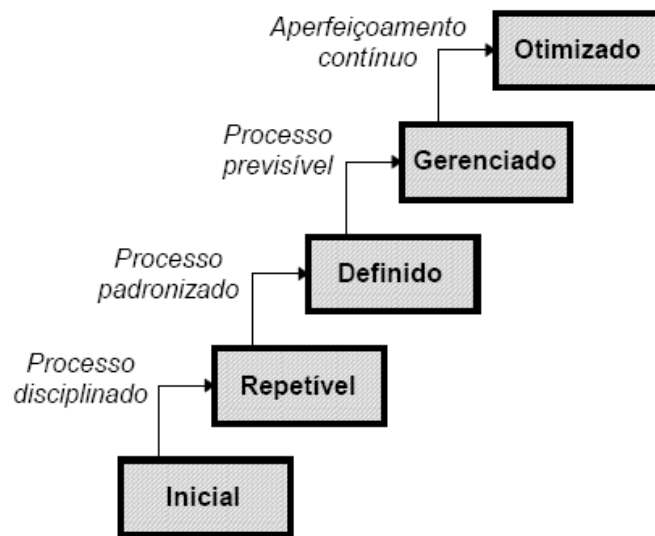


Figura 2.1 - Os níveis de maturidade de um processo de desenvolvimento de software.

5.2.1. Nível Inicial

No **nível inicial**, o desenvolvimento de software é realizado de forma totalmente “ad hoc”, sem uma definição de processos. No caso de problemas que venham a ocorrer durante a realização de um projeto, a organização tem uma tendência a abandonar totalmente os procedimentos planejados e passa a um processo de codificação e testes, onde o produto obtido pode apresentar um nível de qualidade suspeito.

A capacidade de uma empresa caracterizada como nível 1 é totalmente imprevisível, uma vez que o processo de desenvolvimento de software é instável, sujeito a mudanças radicais frequentes, não apenas de um projeto a outro, mas também durante a realização de um mesmo projeto.

Neste nível, estimativa de custos, prazos e qualidade do produto é algo totalmente fora do contexto e da política de desenvolvimento (que política?).

Embora não se possa “assegurar” o fracasso de um projeto desenvolvido por uma empresa situada neste nível, é possível dizer que o sucesso é, geralmente, resultado de esforços individuais, variando com as habilidades naturais, o conhecimento e as motivações dos profissionais envolvidos no projeto.

5.2.2. Nível Repetível

Neste nível, políticas de desenvolvimento de software e tarefas de suporte a estas políticas são estabelecidas, o planejamento de novos projetos sendo baseado na experiência obtida com projetos anteriores.

Para que uma empresa possa atingir este nível, é imprescindível institucionalizar o gerenciamento efetivo dos seus projetos de software, de modo que o sucesso de projetos anteriores possam ser repetidos nos projetos em curso.

Neste nível, os requisitos do software e o trabalho a ser feito para satisfazê-los são planejados e supervisionados ao longo da realização do projeto. São definidos padrões de projeto, e a instituição deve garantir a sua efetiva implementação.

A capacidade de uma empresa situada neste nível pode ser caracterizada como disciplinada, em razão dos esforços de gerenciamento e acompanhamento do projeto de software.

5.2.3. Nível Definido

No **nível definido**, o processo de desenvolvimento de software é consolidado tanto do ponto de vista do gerenciamento quanto das tarefas de engenharia a realizar; isto é feito através de documentação, padronização e integração no contexto da organização, que adota esta versão para produzir e manter o software.

Os processos definidos nas organizações situadas neste nível são utilizados como referência para os gerentes de projeto e os membros do staff técnico, sendo baseado em práticas propostas pela Engenharia de Software.

Programas de treinamento são promovidos ao nível da organização, como forma de difundir e padronizar as práticas adotadas no processo definido.

As características particulares de cada projeto podem influir no aprimoramento de um processo de desenvolvimento, sendo que para cada projeto em desenvolvimento, este pode ser instanciado.

Um processo de desenvolvimento bem definido deve conter padrões, procedimentos para o desenvolvimento das atividades envolvidas, mecanismos de validação e critérios de avaliação.

A capacidade de uma empresa no nível 3 é caracterizada pela padronização e consistência, uma vez que as políticas de gerenciamento e as práticas da Engenharia de Software são aplicadas de forma efetiva e repetida.

5.2.4. Nível Gerenciado

No **nível gerenciado**, é realizada a coleta de medidas do processo e do produto obtido, o que vai permitir um controle sobre a produtividade (do processo) e a qualidade (do produto).

É definida uma base de dados para coletar e analisar os dados disponíveis dos projetos de software. Medidas consistentes e bem definidas são, então, uma característica das organizações situadas neste nível, as quais estabelecem uma referência para a avaliação dos processos de desenvolvimento e dos produtos.

Os processos de desenvolvimento exercem um alto controle sobre os produtos obtidos; as variações de desempenho do processo podem ser separadas das variações ocasionais (ruídos), principalmente no contexto de linhas de produção definidas. Os riscos relacionados ao aprendizado de novas tecnologias ou sobre um novo domínio de aplicação são conhecidos e gerenciados cuidadosamente.

A capacidade de uma organização situada este nível é caracterizada pela previsibilidade, uma vez que os processos são medidos e operam em limites conhecidos.

5.2.5. Nível Otimizado

No **nível otimizado**, a organização promove contínuos aperfeiçoamentos no processo de desenvolvimento, utilizando para isto uma realimentação quantitativa do processo e aplicando novas idéias e tecnologias. Os aperfeiçoamentos são definidos a partir da identificação dos pontos fracos e imperfeições do processo corrente e do estabelecimento das alterações necessárias para evitar a ocorrência de falhas. Análises de custo/benefício são efetuadas sobre o processo de desenvolvimento com base em dados extraídos de experiências passadas.

Quando os problemas relacionados à adoção de um dado processo de desenvolvimento não podem ser totalmente eliminados, os projetos são cuidadosamente acompanhados para evitar a ocorrência de problemas inerentes do processo.

5.3. Definição operacional do modelo CMM

O modelo CMM, além de definir os níveis de maturidade acima descritos, detalha, cada um deles, com respeito aos objetivos essenciais de cada um e das tarefas chave a serem implementadas para que estes objetivos sejam atingidos.

Para isto, foi realizado, à exceção do nível 1, um detalhamento nos diferentes níveis, estabelecendo uma estrutura que permitisse caracterizar a maturidade e a capacidade do processo de desenvolvimento de software. A figura 2.2 ilustra os componentes relacionados a este detalhamento.

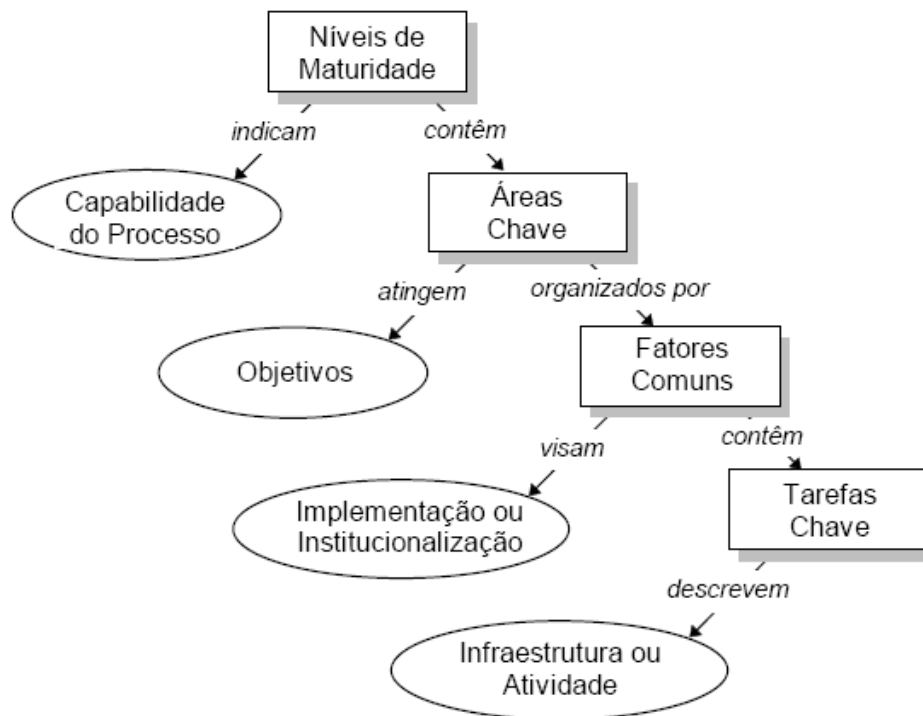


Figura 2.2 - Estrutura dos níveis CMM.

Como se pode notar na figura, cada nível de maturidade é composto por diversas **Áreas Chave**, as quais identificam um conjunto de atividades que, quando realizadas conjuntamente, permitem atingir os objetivos essenciais do nível considerado, aumentando a capacidade do processo de desenvolvimento de software.

A Tabela 2.1 apresenta as áreas chave associadas a cada um dos níveis do CMM (à exceção do nível 1, como já foi explicado).

Os **Fatores Comuns** indicam o grau de implementação ou institucionalização de uma dada Área Chave. No modelo, os Fatores Comuns foram definidos em número de cinco, como mostra a Tabela 2.2.

As **Tarefas Chave** correspondem às atividades que, uma vez realizadas de modo conjunto, contribuirão para o alcance dos objetivos da área chave. As tarefas chave descrevem a infra-estrutura e as atividades que deverão ser implementadas para a efetiva implementação ou institucionalização da área chave. As tarefas chave correspondem a uma sentença simples, seguida por uma descrição detalhada a qual pode incluir exemplos.

A figura 2.3 apresenta um exemplo de estrutura de uma tarefa chave para a Área Chave de Planejamento do Projeto de Software.

Nível	Áreas Chave
Repetível (2)	<ul style="list-style-type: none"> • Gerenciamento de requisitos • Planejamento do processo de desenvolvimento • Acompanhamento do projeto • Gerenciamento de subcontratação • Garantia de qualidade • Gerenciamento de configuração
Definido (3)	<ul style="list-style-type: none"> • Ênfase ao processo na organização • Definição do processo na organização • Programa de treinamento • Gerenciamento integrado • Engenharia de software • Coordenação intergrupo • Atividades de revisão
Gerenciado (4)	<ul style="list-style-type: none"> • Gerenciamento da qualidade de software • Gerenciamento quantitativo do processo
Otimizado (5)	<ul style="list-style-type: none"> • Prevenção de falhas • Gerenciamento de mudança de tecnologia • Gerenciamento de mudança do processo

Tabela 2.1 - Áreas Chave por nível de maturidade.

Fator Comum	Objetivos
Passível de realização	Descreve as ações a realizar para definir e estabilizar um processo
Capacidade de realização	Define pré-condições necessárias no projeto ou organização para implementar o processo de modo competente
Atividades realizadas	Descreve os procedimentos necessários para implementar uma área chave
Medidas e análises	Indica a necessidade de realização de atividades de medição e análise das medidas
Verificação da implementação	Corresponde aos passos necessários para assegurar que todas as tarefas foram realizadas adequadamente

Tabela 2.2 - Fatores comuns.

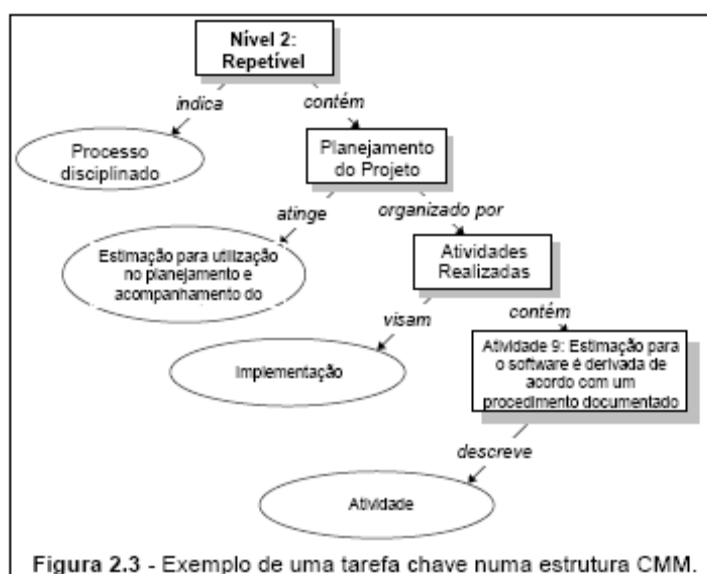


Figura 2.3 - Exemplo de uma tarefa chave numa estrutura CMM.

5.4. Utilização do modelo CMM

O objetivo fundamental do estabelecimento deste modelo é fornecer parâmetros para:

- de um lado, que as instituições que pretendam contratar fornecedores de software possam avaliar as capacidades destes fornecedores;
- por outro lado, para que as empresas de desenvolvimento de software possam identificar quais os procedimentos a serem implementados ou institucionalizados no âmbito da organização de modo a aumentar a capacidade do seu processo de software.

Embora os dois objetivos sejam diferentes, é definido no modelo um procedimento similar para a realização dos dois tipos de análise. As etapas deste procedimento, descritas a seguir, deverão ser realizadas num esquema seqüencial:

- seleção de uma equipe, cujos elementos serão treinados segundo os conceitos básicos do modelo CMM, sendo que estes já deverão ter conhecimentos de engenharia de software e gerenciamento de projetos;
- selecionar profissionais representantes da organização, para os quais será aplicado um questionário de maturidade;
- analisar as respostas obtidas do questionário de maturidade, procurando identificar as áreas chave;
- realizar uma visita (da equipe) à organização para a realização de entrevistas e revisões de documentação relacionadas ao processo de desenvolvimento; as entrevistas e revisões deverão ser conduzidas pelas áreas chave do CMM e pela análise realizada sobre o questionário de maturidade;
- ao fim da visita, a equipe elabora um relatório enumerando os pontos fortes e os pontos fracos da organização em termos de processo de desenvolvimento de software;
- finalmente, a equipe prepara um perfil de áreas chave, que indica as áreas onde a organização atingiu/não atingiu os objetivos de cada área.

5.3 NORMAS E MODELOS DE QUALIDADE DE PROCESSO DE SOFTWARE

Origem: BERTOLLO, Gleidson. Dissertação de Mestrado. Definição de processos em um ambiente de desenvolvimento de software. UFES. 2006.

2.2 Qualidade de Processos de Software

Organizações bem sucedidas melhoram continuamente seus processos. Com a definição de um processo padrão organizacional, a melhoria sistemática dos processos é mais efetiva e eficiente se guiada por normas e modelos de qualidade de processo. A finalidade da maioria desses padrões é ajudar as organizações de software a conseguir a excelência seguindo processos e atividades adotados por organizações com alto grau de maturidade. Contudo, não é fácil selecionar os padrões apropriados. Existem muitas opções com uma grande sobreposição entre elas. Muitas vezes, é vantagem para uma organização de software usar ou implementar mais de um padrão ao mesmo tempo. Nessa situação, é melhor implementá-los simultaneamente. Tal abordagem permite aos engenheiros de processo maximizar os pontos fortes de um padrão e diminuir suas fraquezas (MUTAFELIJA, STROMBERG, 2003). A seguir, é apresentado um breve resumo das principais normas e modelos de qualidade, a saber: ISO 9000, ISO/IEC 12207, ISO/IEC 15504, CMMI e MPS.BR. Além desses modelos e normas, o modelo do Processo Unificado da Rational (Rational Unified Process – RUP) é também descrito, dada a sua ampla disseminação e aceitação.

2.2.1 ISO 9000

Em 2000, uma revisão da família de normas ISO 9000 – Sistemas de gestão da qualidade – foi publicada, substituindo a versão anterior, cuja versão em português da ABNT datava de 1994.

Algumas normas da família ISO 9000 foram alteradas, outras canceladas e novas criadas, ficando a ISO 9000 resumida a quatro normas (ISO, 2000b):

- ISO 9000 – Fundamentos e Vocabulário (ISO, 2000b);
- ISO 9001 – Requisitos para sistemas de gestão da qualidade (ISO, 2000a);
- ISO 9004 – Diretrizes para Melhoria de Desempenho (ISO, 2000c).
- ISO 19011 – Diretrizes sobre auditoria de sistemas de gestão da qualidade e ambiental.

A principal alteração ocorreu na abordagem adotada: mudou-se de uma perspectiva prescritiva baseada em procedimentos, para práticas de gestão da qualidade baseadas em uma abordagem de engenharia de sistemas, orientada a processos, buscando a satisfação do cliente e a melhoria contínua (MUTAFELIJA, STROMBERG, 2003).

A norma ISO 9001:2000 é a única certificadora, ou seja, as organizações podem obter um certificado de sistema de gestão da qualidade se implementarem seus requisitos. Seu objetivo é estabelecer requisitos genéricos para o sistema de gestão da qualidade, sendo aplicáveis a todas organizações, sem considerar tipo, tamanho ou produto fornecido. O padrão é baseado em princípios de gestão da qualidade. Esses princípios não estão explicitamente na parte normativa da norma, mas são rastreáveis em seus principais requisitos. Os princípios são (ISO, 2000a): foco no cliente, liderança, envolvimento das pessoas, abordagem de processo, abordagem sistêmica para a gestão, melhoria contínua, abordagem factual para tomada de decisão e benefícios mútuos na relação com os fornecedores.

Organizações desenvolvem produtos e prestam serviços através de uma sistemática definida. A complexidade desses produtos e serviços pode requerer processos interdisciplinares que transformam requisitos do cliente, entradas e restrições em um produto, ou mais genericamente, em uma solução. Esses processos inter-relacionados formam um sistema, definido como uma combinação de elementos (humanos, software e hardware) e processos (facilidades, equipamentos, material e procedimentos) que são relacionados para satisfazer necessidades através de um ciclo de vida sustentável do produto (IEEE, 1998).

A mudança de foco na nova versão da norma teve o objetivo de encorajar a adoção da abordagem de processo para a gerência de uma organização. É natural que diferentes processos possam interagir e os mesmos devem ser gerenciados coletivamente para atingir os objetivos organizacionais. Para funcionarem de forma eficaz, as organizações têm que identificar e gerenciar processos inter-relacionados.

2.2.2 ISO/IEC 12207

A norma NBR ISO/IEC 12207 – Tecnologia da Informação – Processos de Ciclo de Vida de Software (ISO/IEC, 1995) estabelece uma estrutura comum para os processos de ciclo de vida de software, com terminologia bem definida, que pode ser referenciada pela indústria de software. A estrutura contém processos, atividades e tarefas que devem ser aplicados na aquisição, fornecimento, desenvolvimento, operação e manutenção de produtos de software. Esse conjunto de processos, atividades e tarefas foi projetado para ser adaptado, de acordo com as características específicas, a uma organização, projeto ou aplicação, o que pode envolver o detalhamento, a adição e a supressão de elementos de acordo com o objetivo de utilização.

Seu modelo de referência de processo fornece indicações de processos, descritos em termos de seus propósitos e resultados esperados, em conjunto com uma arquitetura que descreve as relações entre esses processos. Define, ainda, atividades e tarefas requeridas para implementar em alto nível tais processos, objetivando atingir a capacidade desejada para compradores, fornecedores, desenvolvedores, mantenedores e operadores de sistemas que contêm software.

Na ISO/IEC 12207 são consideradas três categorias de processos: Fundamentais, de Apoio e Organizacionais. Os processos fundamentais constituem um conjunto de cinco processos que atendem às partes fundamentais (pessoa ou organização) durante o ciclo de vida do software. Os processos fundamentais são: Processo de Aquisição, Processo de Fornecimento, Processo de Desenvolvimento, Processo de Operação e Processo de Manutenção. Os processos de apoio auxiliam um outro processo como parte integrante, com um propósito distinto, contribuindo para o

sucesso e qualidade do projeto de software. Um processo de apoio é empregado por outro processo, quando necessário. São eles: Processo de Documentação, Processo de Gerência de Configuração, Processo de Garantia da Qualidade, Processo de Verificação, Processo de Validação, Processo de Revisão Conjunta, Processo de Auditoria e Processo de Resolução de Problemas. Os processos organizacionais são empregados por uma organização para estabelecer e implementar uma estrutura subjacente de forma a melhorar continuamente os seus processos. São empregados tipicamente fora do domínio de projetos e contratos específicos; entretanto, ensinamentos desses projetos e contratos contribuem para a melhoria da organização. São eles: Processo de Gerência, Processo de Infra-estrutura, Processo de Melhoria e Processo de Treinamento .

A norma descreve a arquitetura dos processos de ciclo de vida de software, mas não representa uma abordagem de implementação particular, nem prescreve um modelo de ciclo de vida, metodologia ou técnica específica. O modelo de referência tem o objetivo de ser adaptado para uma organização, considerando suas necessidades de negócio e domínios de aplicação.

A ISO/IEC 12207 foi originalmente publicada em 1995 como uma norma internacional. Em 1998, foi publicada a sua versão brasileira que sustenta o mesmo nome que a internacional, acrescida das iniciais NBR. Em 2002 e 2004, foram feitas atualizações, chamadas de emendas 1 e 2 respectivamente, quando foram inseridas algumas melhorias. Essas melhorias criaram novos ou expandiram o escopo de alguns processos, inseriram para cada processo o seu propósito e resultados esperados e para os novos processos definiram suas atividades e tarefas. Essas modificações têm o objetivo de representar a evolução da área de processos de software, as necessidades vivenciadas pelos usuários da norma e sua harmonização com a série ISO/IEC 15504 - Avaliação de Processos de Software.

2.2.3 ISO/IEC 15504

A norma ISO/IEC 15504 foi publicada pela primeira vez em 1998 com a designação de Relatório Técnico (ISO/IEC TR 15504) (ISO/IEC, 1998). Essa designação é atribuída quando o assunto está ainda em desenvolvimento técnico ou quando existe a possibilidade de se alcançar um entendimento para uma Norma Internacional. Em 2003, a ISO/IEC 15504 (ISO/IEC, 2003) foi publicada como norma, sendo organizada em cinco partes sob o título genérico Tecnologias de Informação – Avaliação de Processos. As partes que a compõem são (ISO/IEC, 2003):

- Parte 1: Conceitos e Vocabulário (Informativa) – estabelece uma introdução geral sobre os conceitos de avaliação de processos e um glossário para os termos relacionados;
- Parte 2: Execução de uma avaliação (Normativo) – define os requisitos mínimos para execução de uma avaliação que garanta consistência e repetição dos processos;
- Parte 3: Guia para execução de uma avaliação (Informativo) – estabelece uma interpretação dos requisitos para execução de uma avaliação;
- Parte 4: Guia para utilização em processos de melhoria e na determinação da capacidade de processos (Informativo) – identifica a avaliação de processos como uma atividade a ser executada como parte de uma iniciativa de melhoria de processos ou como parte de uma abordagem de determinação de capacidade.
- Parte 5: Um exemplo de um modelo de avaliação de processos (Informativo) – contém um exemplo de um Modelo de Avaliação de Processo baseado no Modelo de Referência de Processo definido na ISO/IEC 12207, emendas 1 e 2.

A avaliação de processos é baseada num modelo bi-dimensional, que contém uma Dimensão de Processo e uma Dimensão de Capacidade. A Dimensão de Processo é constituída por processos de ciclo de vida definidos no Modelo de Referência de Processos, o qual define um conjunto de processos caracterizados por uma descrição dos objetivos e dos resultados. Um exemplo de modelo de referência é derivado diretamente da norma ISO/IEC 12207 emendas 1 e 2. Os processos encontram-se agrupados em nove categorias: Aquisição, Fornecimento, Engenharia, Operação, Apoio, Gerência, Melhoria de Processo, Recurso e Infra-estrutura, e Reuso.

A norma permite, ainda, que os modelos de referência de processos possam ser desenvolvidos fora do contexto da normalização ISO, desde que preencham um conjunto de requisitos de conformidade pré-estabelecidos. O patrocinador deve determinar qual modelo de referência de processo melhor

atende o requisito especificado ou objetivo de negócio definido, seguindo o guia ISO/IEC 15504-3 para seleção.

A Dimensão de Capacidade consiste numa plataforma de medição composta por seis níveis de capacidade e pelos atributos de processo associados a cada nível. O resultado da avaliação consiste na determinação do perfil de cada um dos processos através da classificação dos respectivos atributos. A avaliação pode incluir, ainda, o nível de capacidade atingido por cada processo (determinado a partir dos atributos). Os níveis de capacidade, ordenados de forma crescente de 0 (zero) a 5 (cinco), são: Incompleto, Executado, Gerenciado, Estabelecido, Previsível e Otimizado. Os atributos de processo são:

- Nível 1 – Execução de Processo;
- Nível 2 – Gerência da Execução e Gerência de Produtos de Trabalho;
- Nível 3 – Definição de Processo e Implementação de Processo;
- Nível 4 – Medição de Processo e Controle de Processo;
- Nível 5 – Inovação de Processo e Otimização de Processo.

2.2.4 CMMI

Em 1995, o Software Engineering Institute (SEI) criou um modelo de maturidade e capacidade para organizações de software, o SW-CMM. O modelo contém elementos essenciais para um processo efetivo em diversas disciplinas e descreve um caminho para as organizações evoluírem seus processos de um nível imaturo para um nível disciplinado. Assim como foi elaborado um modelo para software, diversas outras disciplinas criaram os seus modelos de maturidade cobrindo outras áreas de interesse, tais como o SA-CMM (Modelo de Maturidade e Capacidade para Aquisição de Software) e o SE-CMM (Modelo de Maturidade e Capacidade para Engenharia de Sistemas).

Embora esses modelos tenham sido bastante úteis para muitas organizações, o uso de múltiplos modelos tornou-se problemático. Por não serem desenvolvidos de forma integrada, diferenças na arquitetura, conteúdo e abordagem aumentaram os custos de treinamento, avaliações e atividades de melhoria.

Por estas razões, o SEI elaborou o CMMI (CMM Integration) (CHRISISS et al., 2003) com o objetivo de apoiar a melhoria de processos e produtos, diminuindo a redundância e eliminando a inconsistência que surge ao se utilizar diversos modelos independentes. Esse objetivo é atingido através da integração dos diversos CMMs numa estrutura única, todos com a mesma terminologia, processos de avaliação e estrutura. O projeto também se preocupou em tornar o CMMI compatível com a norma ISO/IEC 15504, de modo que avaliações em um modelo sejam reconhecidas como equivalentes aos do outro (CHRISISS et al., 2003).

O CMMI oferece duas abordagens diferentes para a melhoria de processos, sendo conhecidas por “modelo em estágios” e “modelo contínuo”. A representação por estágios prescreve a ordem para implementação de cada área de processo de acordo com níveis de maturidade, que definem um caminho de melhoria para uma organização do nível Inicial para o nível Otimizado. Atingindo um nível de maturidade, garante-se uma base adequada para buscar o próximo estágio.

A representação contínua oferece uma abordagem flexível para melhoria de processos. Uma organização pode escolher melhorar a qualidade de um processo específico ou trabalhar em diversas áreas de forma alinhada aos objetivos de negócio. Os níveis de capacidade são usados para medir uma área, indo de um processo não executado para um processo otimizado. Existem algumas limitações de escolha, devido à dependência entre as áreas de processos.

As duas representações contêm os mesmos componentes, usando os mesmos conceitos. A diferença entre as representações está na abordagem para melhoria de processos. Se uma organização planeja atingir uma maturidade organizacional, deve selecionar a representação por estágios. Se uma organização tem interesse na capacidade de um processo específico, deve selecionar a representação contínua.

O CMMI é estruturado em termos de áreas de processo (Process Area – PA), que consistem em práticas relacionadas que coletivamente satisfazem um conjunto de objetivos. Um objetivo genérico descreve a institucionalização requerida para atingir um nível de capacidade (representação contínua) ou de maturidade (representação por estágio). Cada objetivo genérico é associado a um

conjunto de práticas genéricas que descrevem atividades requeridas para a institucionalização de processos em uma determinada área de processo. Cada área de processo contém, ainda, objetivos específicos e práticas específicas, que descrevem atividades essenciais no atendimento aos objetivos específicos. Os componentes do modelo CMMI podem, ainda, ser agrupados em três categorias: requerido, esperado e informativo. A Figura 2.2 apresenta esses componentes.

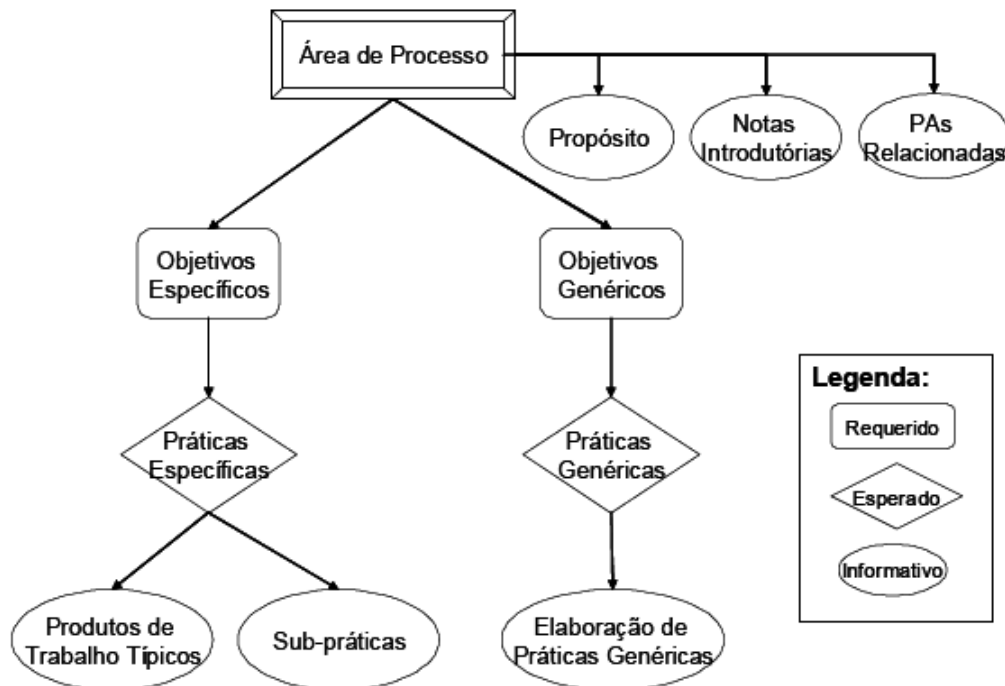


Figura 2.2 – Componentes do Modelo CMMI.

2.2.5 MPS.BR

O MPS.BR (Melhoria de Processo do Software Brasileiro) (SOFTEX, 2005) é um modelo para avaliação e melhoria do processo de software. A base técnica é composta pelas normas NBR ISO/IEC 12207 e suas emendas 1 e 2, e a ISO/IEC 15504, além de cobrir todo o conteúdo do modelo CMMI.

O MPS.BR baseia-se nos conceitos de maturidade e capacidade de processo para a avaliação e melhoria da qualidade e produtividade de produtos de software e serviços correlatos. Dentro desse contexto, o MPS.BR possui três componentes: Modelo de Referência (MR-MPS), que contém os requisitos a serem atendidos pelas organizações, bem como os níveis de maturidade e capacidade, e os processos; Modelo de Avaliação (MA-MPS), que contém o processo de avaliação, os requisitos para os avaliadores e os requisitos para averiguação da conformidade ao MR-MPS; Modelo de Negócio (MN-MPS), que contém as regras para implantação do MPS.BR pelas empresas de consultoria, de software e avaliação.

O MPS.BR é descrito através de documentos em formato de guias:

- Guia Geral: contém a descrição geral do MPS.BR e detalha o Modelo de Referência (MR-MPS), seus componentes e as definições comuns necessárias para seu entendimento e aplicação.
- Guia de Aquisição: contém recomendações para a condução de compras de software e serviços correlatos. Foi descrito como forma de apoiar as instituições que queiram adquirir produtos de software e serviços correlatos apoiando-se no MR-MPS.
- Guia de Avaliação: contém a descrição do processo de avaliação, os requisitos para o avaliador, os requisitos para a avaliação, o método e os formulários para apoiar a avaliação.

O Modelo de Referência MR-MPS define níveis de maturidade como uma combinação entre processos e capacidade de processos. Os níveis de maturidade estabelecem patamares de evolução de processo, caracterizando estágios de melhoria de implementação de processos na organização. O MR-MPS define sete níveis de maturidade: A (Em Otimização), B (Gerenciado Quantitativamente), C (Definido), D (Largamente Definido), E (Parcialmente Definido), F (Gerenciado) e G (Parcialmente Gerenciado). A escala de maturidade se inicia no nível G e progride até o nível A. Para cada um desses sete níveis de maturidade foi atribuído um perfil de processos e de capacidade de processos que indica onde a organização tem que colocar esforço para melhoria, de forma a atender os objetivos de negócio.

A capacidade do processo é um conjunto de atributos de processo descrito em termos de resultados. A capacidade estabelece o grau de refinamento e institucionalização com que o processo é executado na organização. À medida que evolui nos níveis, um maior ganho de capacidade para desempenhar o processo é atingido pela organização. A capacidade do processo possui cinco atributos de processo, que são detalhados, por sua vez, em termos dos resultados para alcance completo do atributo. Os atributos de processo do MPS.BR são:

- AP 1.1: O processo é executado;
- AP 2.1: O processo é gerenciado;
- AP 2.2: Os produtos de trabalho do processo são gerenciados;
- AP 3.1: O processo é definido e
- AP 3.2: O processo está implementado.

O progresso e a obtenção de um nível de maturidade ocorrem quando são atendidos todos os resultados e os propósitos dos processos e dos atributos de processo relacionados àquele nível e aos anteriores. A Tabela 2.1 apresenta os processos de cada nível de maturidade e a capacidade exigida.

Tabela 2.1 – Níveis de Maturidade.

Nível	Processo	Capacidade
A	Implantação de Inovações na Organização	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Análise de Causas e Resolução	
B	Desempenho do Processo Organizacional	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Gerência Quantitativa do Projeto	
C	Análise de Decisão e Resolução	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Gerência de Riscos	
D	Desenvolvimento de Requisitos	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Solução Técnica	
	Integração do Produto	
	Verificação	
E	Validação	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP 3.2
	Treinamento	
	Definição do Processo Organizacional	
	Avaliação e Melhoria do Processo Organizacional	
F	Adaptação do Processo para Gerência do Projeto	AP 1.1, AP 2.1 e AP 2.2
	Medição	
	Gerência de Configuração	
	Aquisição	
G	Garantia da Qualidade	AP 1.1 e AP 2.1
	Gerência de Requisitos	
	Gerência do Projeto	

5.4 REVISÕES

Origem: Engenharia de Software, Notas de Aula, prof. Ricardo Falbo, UFES, Departamento de Informática.

Para se controlar a qualidade em um projeto de software, uma abordagem bastante usada consiste em se realizar revisões. Nas revisões, processos, documentos e outros artefatos são revisados por um grupo de pessoas, com o objetivo de verificar se os mesmos estão em conformidade com os padrões organizacionais estabelecidos e se o propósito de cada um deles está sendo atingido, incluindo o atendimento a requisitos do cliente e dos usuários. Assim, o objetivo de uma revisão é detectar erros e inconsistências em artefatos e processos, sejam eles relacionados à forma, sejam em relação ao conteúdo, e apontá-los aos responsáveis pela sua elaboração.

O processo de revisão começa com o planejamento da revisão, quando uma equipe de revisão é formada, tendo à frente um líder. A equipe de revisão deve incluir os membros da equipe que possam efetivamente ser úteis para atingir o objetivo da revisão. Muitas vezes, a pessoa responsável pela elaboração do artefato a ser revisado integra a equipe de revisão.

Dando início ao processo de revisão propriamente dito, normalmente, o autor do artefato apresenta o mesmo e descreve a perspectiva utilizada para a sua construção. Além disso, o propósito da revisão deve ser previamente informado e o material a ser revisado deve ser entregue com antecedência para que cada membro da equipe de revisão possa avaliá-lo.

Uma vez que todos estejam preparados, uma reunião é convocada pelo líder. Essa reunião deverá ser relativamente breve (duas horas, no máximo), uma vez que todos já estão preparados para a mesma. Durante a reunião, o líder orientará o processo de revisão, passando por todos os aspectos relevantes a serem revistos. Todas as considerações dos demais membros da equipe de revisão devem ser discutidas e as decisões registradas, dando origem a uma ata de reunião de revisão, contendo uma lista de defeitos encontrados.

Revisões de Qualidade

Origem: Engenharia de Software, Ian Sommerville, 2007

As revisões são métodos de validação de qualidade de um processo ou produto amplamente usados. Envolvem um grupo de pessoas que examina parte ou o todo de um processo de software, sistema ou sua documentação associada para descobrir problemas potenciais. As conclusões da revisão são formalmente registradas e passadas para o autor ou quem mais for o responsável por corrigir os problemas detectados.

Os softwares ou documentos podem ser 'assinados' em uma revisão, o que significa que o progresso para o próximo estágio de desenvolvimento foi aprovado pela gerência.

Existem diferentes tipos de revisão, com objetivos diferentes:

- Inspeções para remoção de defeitos (produto);
- Revisões para avaliação de progresso (produto e processo);
- Revisões de qualidade (produto e padrões).

Funções de revisão:

- Função de qualidade – é parte do processo geral de gerenciamento de qualidade.
- Função de gerenciamento de projeto – fornecem informações para os gerentes de projeto.
- Função de treinamento e comunicação – o conhecimento do produto é passado entre os membros da equipe de desenvolvimento.

O principal objetivo é a descoberta de defeitos e inconsistências de sistema, e apontá-los para o projetista ou o autor do documento. Quaisquer documentos produzidos no processo podem ser revisados, não se restringindo apenas ao código-fonte.

As equipes de revisão devem ser relativamente pequenas, apesar de os membros de tal equipe poderem convidar outros membros a integrar temporariamente a equipe durante a realização de uma revisão específica.

As revisões devem ser bem curtas, no máximo duas horas. A reunião de revisão engloba basicamente percorrer o documento, havendo um moderador e um membro que realiza o registro da revisão. O moderador será responsável por garantir que as mudanças necessárias indicadas durante a reunião sejam realizadas. Dependendo da quantidade de alteração proposta uma revisão de acompanhamento pode ser organizada.

Sobre a revisão realizada:

- Os registros devem ser sempre mantidos para revisões de qualidade, de maneira formal;
- Os comentários feitos durante a revisão devem ser classificados:
 - Nenhuma ação. Nenhuma mudança no software ou na documentação é necessária;
 - Se referir ao reparo. O projetista ou o programador deve corrigir um defeito identificado;

- Reconsiderar o projeto global. O problema identificado na revisão tem impacto sobre outras partes do projeto. Algum julgamento geral deve ser feito sobre a maneira mais adequada em termos de custo para resolver o problema;
- Os erros de requisitos e de especificação devem ser comunicados ao cliente.

5.5 DOCUMENTAÇÃO

Origem: Engenharia de Software, Notas de Aula, prof. Ricardo Falbo, UFES, Departamento de Informática.

A documentação produzida em um projeto de software é de suma importância para se gerenciar a qualidade, tanto do produto sendo produzido, quanto do processo usado para seu desenvolvimento.

No desenvolvimento de software, são produzidos diversos documentos, dentre eles, documentos descrevendo processos (plano de projeto, plano de qualidade etc.), registrando requisitos e modelos do sistema (documentos de especificação de requisitos, análise e projeto) e apoiando o uso do sistema gerado (manual do usuário, ajuda *on-line*, tutoriais etc).

Uma documentação de qualidade propicia uma maior organização durante o desenvolvimento de um sistema, facilitando modificações e futuras manutenções no mesmo. Além disso, reduz o impacto da perda de membros da equipe, reduz o tempo de desenvolvimento de fases posteriores, reduz o tempo de manutenção e contribui para redução de erros, aumentando assim, a qualidade do processo e do produto gerado. Dessa forma, a criação da documentação é tão importante quanto a criação do software em si [4].

Há, portanto, a necessidade de se definir um processo para controlar a documentação de uma organização, dito processo de documentação, incluindo atividades de planejamento, análise, aprovação ou reprovação, identificação de alterações, situação da revisão atual, disponibilidade das versões pertinentes de documentos aplicáveis, dentre outras. Algumas dessas atividades estão relacionadas com o controle e a garantia da qualidade de software, outras com a gerência da configuração do software, conforme discutido a seguir.

É importante notar que o planejamento da documentação tem uma estreita relação com o processo de software definido para o projeto. Ou seja, os documentos a serem gerenciados são aqueles previstos como saídas das atividades do processo. Assim, tendo sido definido o processo do projeto, o planejamento da sua documentação consiste apenas em selecionar quais artefatos, dentre os muitos produzidos ao longo do processo, serão efetivamente submetidos à gerência de configuração de software e ao controle e garantia da qualidade.

Padrões de Documentação

Origem: Engenharia de Software, Ian Sommerville, 2007

Os padrões de documentação em um projeto de software são particularmente importantes, pois os documentos são a manifestação tangível do software.

Tipos de padrões de documentação:

- Padrões de processo de documentação - Relacionado ao modo como os documentos devem ser desenvolvidos, validados e mantidos;
- Padrões de documentos - Relacionado ao conteúdo, à estrutura e à aparência de documentos;
- Padrões de intercâmbio de documentos - Relacionado à compatibilidade de documentos eletrônicos.

Os padrões de processo de documentação definem o processo para produzir documentos. O que envolve estabelecer: procedimentos para o desenvolvimento de documentos, ferramentas a serem

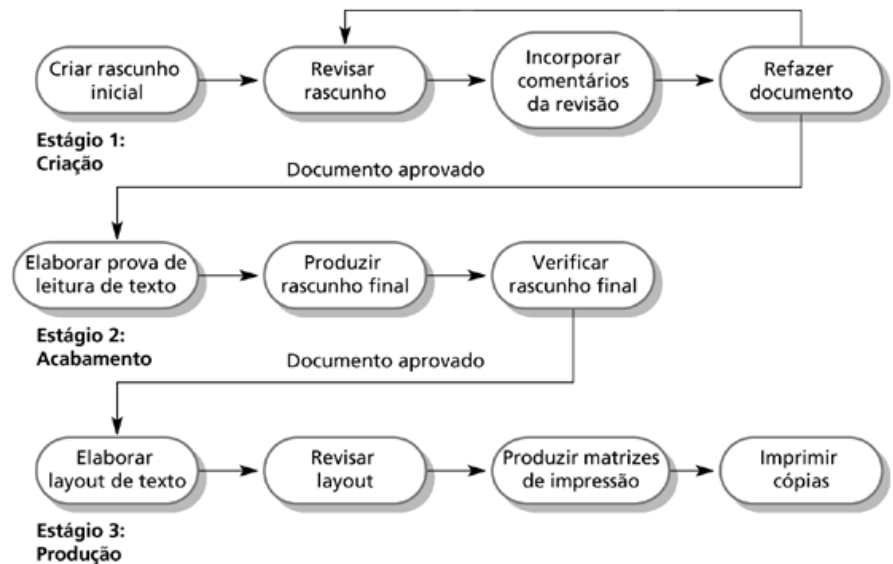
utilizadas, procedimentos de verificação e refinamento para garantia de qualidade. Tais processos devem ser suficientemente flexíveis para lidar com todos os tipos de documentos.

É uma boa prática utilizar o mesmo “estilo” para todos os documentos gerados durante o processo de desenvolvimento de software.

A figura a seguir é um modelo de processo de documentação.

Figura 27.4

Processo de produção de documentos incluindo verificação de qualidade.



Alguns exemplos de padrões de documentos que podem ser desenvolvidos:

- Padrões de identificação de documentos - Como os documentos são unicamente identificados.
- Padrões de estrutura de documentos - Estrutura padrão para documentos de projeto.
- Padrões de apresentação de documentos - Definem fontes e estilos, uso de logotipos, etc.
- Padrões de atualização de documentos - Definem como as mudanças nas versões anteriores são refletidas em um documento.

Padrões de Intercâmbio de documentos

- Padrões para intercâmbio permitem que documentos eletrônicos sejam trocados, enviados pelo correio, etc.
- Documentos são produzidos usando sistemas e computadores diferentes. Mesmo quando ferramentas padronizadas são usadas, os padrões são necessários para definir convenções para o seu uso, por exemplo, o uso de folhas de estilo e macros.
- Necessidade de arquivamento. O tempo de vida dos sistemas de processamento de texto pode ser muito menor que o tempo de vida do software que está sendo documentado. Um padrão de arquivamento pode ser definido para assegurar que o documento possa ser acessado no futuro.

5.6 GERÊNCIA DE CONFIGURAÇÃO

Origem: Engenharia de Software, Notas de Aula, prof. Ricardo Falbo, UFES, Departamento de Informática.

Durante o processo de desenvolvimento de software, vários artefatos são produzidos e alterados constantemente, evoluindo até que seus propósitos fundamentais sejam atendidos. Ferramentas de software, tais como compiladores e editores de texto, e processos também podem ser substituídos

por versões mais recentes ou mesmo por outras, no caso de ferramentas. Porém, caso essas mudanças não sejam devidamente documentadas e comunicadas, poderão acarretar diversos problemas, tais como: dois ou mais desenvolvedores podem estar alterando um mesmo artefato ao mesmo tempo; não se saber qual a versão mais atual de um artefato; não se refletir alterações nos artefatos impactados por um artefato em alteração. Esses problemas podem gerar vários transtornos como incompatibilidade entre os grupos de desenvolvimento, inconsistências, retrabalho, atraso na entrega e insatisfação do cliente.

Assim, para que esses transtornos sejam evitados, é de suma importância o acompanhamento e o controle de artefatos, processos e ferramentas, através de um processo de gerência de configuração de software, durante todo o ciclo de vida do software [5].

A Gerência de Configuração de Software (GCS) visa estabelecer e manter a integridade dos itens de software ao longo de todo o ciclo de vida do software, garantindo a completeza, a consistência e a correção de tais itens, e controlando o armazenamento, a manipulação e a distribuição dos mesmos. Para tal, tem de identificar e documentar os produtos de trabalho que podem ser modificados, estabelecer as relações entre eles e os mecanismos para administrar suas diferentes versões, controlar modificações e permitir auditoria e a elaboração de relatórios sobre o estado de configuração.

Pelos objetivos da GCS, pode-se notar que ela está diretamente relacionada com as atividades de garantia da qualidade de software.

As atividades da GCS podem ser resumidas em:

- Planejamento da GCS: Um plano deve ser elaborado descrevendo as atividades da gerência de configuração, procedimentos e responsáveis pela execução dessas atividades.
- Identificação da Configuração: refere-se à identificação dos itens de software e suas versões a serem controladas, estabelecendo linhas básicas.
- Controle de Versão: combina procedimentos e ferramentas para administrar diferentes versões dos itens de configuração criados durante o processo de software.
- Controle de Modificação: combina procedimentos humanos e ferramentas automatizadas para controlar as alterações feitas em itens de software. Para tal, o seguinte processo é normalmente realizado: solicitação de mudança, aprovação ou rejeição da solicitação, registro de retirada para alteração (*check-out*), análise, avaliação e realização das alterações, revisão e registro da realização das alterações (*check-in*).
- Auditoria de Configuração: visa avaliar um item de configuração quanto a características não consideradas nas revisões, tal como se os itens relacionados aos solicitados foram devidamente atualizados.
- Relato da situação da configuração: refere-se à preparação de relatórios que mostrem a situação e o histórico dos itens de software controlados. Tais relatórios podem incluir, dentre outros, o número de alterações nos itens, as últimas versões dos mesmos e identificadores de liberação.

Gerenciamento de Configuração

Origem: Engenharia de Software, Ian Sommerville, 2007

Novas versões de sistemas de software são criadas quando eles:

- Mudam para máquinas/OS diferentes;
- Oferecem funcionalidade diferente;
- São configurados para requisitos de usuários particulares.

O gerenciamento de configuração está relacionado ao desenvolvimento e o uso de padrões e procedimento para o gerenciamento de sistemas de software em desenvolvimento. O CM pode ser visto como parte de um processo mais geral de gerenciamento de qualidade.

Considerando que requisitos de sistemas sempre mudam e as mudanças devem compor uma nova versão do sistema, é preciso ter uma rastreabilidade de qual versão contém quais mudanças, a fim de evitar problemas.

É considerado que:

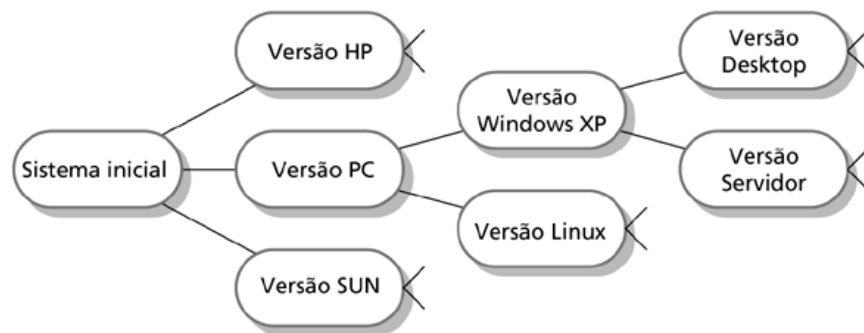
- Mudança de sistema é uma atividade de equipe;
- O CM (*Configuration Management*) tem por objetivo controlar os custos e o esforço envolvidos na realização das mudanças em um sistema.

Quando liberados para o CM, os sistemas de software são chamados, algumas vezes, de *baselines*, visto que são um ponto de partida para desenvolvimento posterior.

Há muitas razões por que os sistemas existem em diferentes configurações. Configurações podem ser produzidas para diferentes computadores, para diferentes sistemas operacionais, incorporando funções especificadas de clientes, etc. Os gerentes de configuração são responsáveis por manter a rastreabilidade das diferenças entre versões de software, para assegurar que as novas versões sejam derivadas de maneira controlada e liberar novas versões para clientes certos no momento certo.

Figura 29.1

Famílias de sistemas.



Padrões de CM

O CM deve ser sempre baseado em um conjunto de padrões que são aplicados dentro de um organização. Tais padrões devem definir como os itens são identificados, como as mudanças são controladas e como as novas versões são gerenciadas. Eles podem ser baseados em padrões de CM externos (por exemplo, o padrão IEEE para CM).

Alguns padrões existentes são baseados em um modelo de processo cascata – novos padrões de CM são necessários para desenvolvimento evolucionário.

Para prover um desenvolvimento incremental, algumas organizações desenvolveram uma abordagem modificada para o gerenciamento de configuração que suporta o desenvolvimento atual e o teste de sistema. Ela é baseada nos seguintes componentes:

- Um horário (digamos 14:00 horas) para entrega de componentes do sistema é acordado;
- Uma nova versão de um sistema é construído a partir de componentes pela compilação e ligação desses componentes;
- Essa nova versão é entregue para teste usando testes predefinidos;
- Os defeitos que são descobertos durante o teste são documentados e enviados para os desenvolvedores do sistema. E serão corrigidos para a versão seguinte.

Construção freqüente do Sistema

É mais fácil encontrar problemas que surgem das interações de componentes no início do processo. Isso encoraja o teste unitário – os desenvolvedores estão sob pressão para não ‘quebrar a construção’.

Um processo de gerenciamento rigoroso de mudanças é necessário para manter a rastreabilidade dos problemas que foram descobertos e reparados.

Planejamento de gerenciamento de configurações

Todos os produtos do processo de software podem ser gerenciados: Especificações, Projetos, Programas, Dados de teste, Manuais de usuário. Milhares de documentos separados podem ser gerados para um sistema grande e complexo de software. Tornando necessário o estabelecimento de um plano de trabalho.

O PLANO DE CM deve:

- Definir os tipos de documentos a serem gerenciados e um esquema de identificação de documentos;
- Definir quem tem a responsabilidade pelos procedimentos e pela criação de *baselines* de CM;
- Definir políticas para controle de mudanças e gerenciamento de versões;
- Definir os registros de CM que devem ser mantidos;
- Descrever as ferramentas que devem ser usadas para o processo de CM e quaisquer limitações de seu uso;
- Definir o processo de uso da ferramentas;
- Definir a base de dados de CM usada para registrar informações de configuração;
- Pode incluir informações, tais como o CM de software externo, a auditoria de processos, etc.

Identificação de Item de Configuração

Projetos grandes produzem tipicamente milhares de documentos que devem ser unicamente identificados. Alguns desses documentos devem ser mantidos pelo tempo de vida do software. E assim é preciso manter a rastreabilidade das informações necessárias, ou seja, é preciso ter um esquema de identificação consistente.

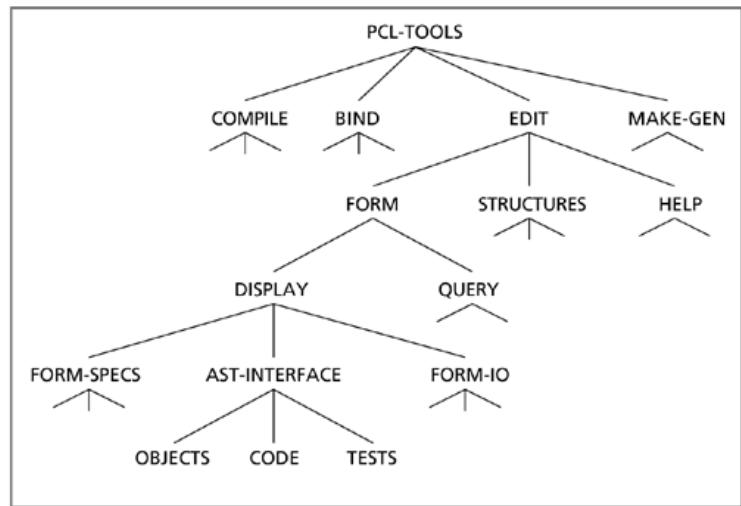
Nem todos os arquivos produzidos devem ser colocados sobre o controle de configuração. Se forem documentos temporários podem ser tratados de forma mais simplificada.

O esquema de identificação de documentos deve ser definido de tal modo que documentos relacionados tenham nomes relacionados.

Um esquema de hierarquia com múltiplos nomes é provavelmente a abordagem mais flexível. Exemplo: PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE/CODE

Figura 29.2

Hierarquia da configuração usada para designar identificadores de itens.



Banco de Dados de Configuração

Um banco de dados de configuração não inclui apenas informações sobre itens de configuração. Pode também registrar informações sobre usuários de componentes, clientes de sistemas, plataformas de execução, mudanças propostas, etc.

Todas as informações de CM devem ser mantidas em uma base de dados de configuração. Isso deve permitir consultas sobre configurações a serem respondidas:

- Quem tem uma versão particular de sistema?
- Qual plataforma é necessária para uma versão particular?
- Quais versões são afetadas pela mudança do componente X?
- Quantos defeitos foram reportados na versão T?

A base de dados de CM deve ser preferivelmente ligada ao software que está sendo usado para gerenciamento. Tal base pode ser parte de um ambiente integrado para apoiar o desenvolvimento de software (A base de dados de CM e os documentos gerenciados são todos mantidos no mesmo sistema). Ferramentas CASE podem ser integradas de tal modo que há um relacionamento próximo entre ferramentas CASE e ferramentas de CM.

Mais comumente, no entanto, a base de dados de CM é mantida separadamente, visto que isso é mais barato e mais flexível.

Gerenciamento de Mudanças

Sistemas de software estão sujeitos às solicitações contínuas de mudanças: De usuários; De desenvolvedores; De forças de mercado.

O gerenciamento de mudanças está relacionado à manutenção da rastreabilidade dessas mudanças e à garantia de que elas sejam implementadas da maneira mais adequada em termos de custo.

Procedimentos de gerenciamento de mudança dizem respeito à análise de custo e benefício das mudanças propostas, à aprovação das mudanças viáveis e à rastreabilidade de quais componentes do sistema foram alterados. O processo de gerenciamento de mudanças (como proposto na figura abaixo) deve surtir efeito quando o software / documentação são colocados em baseline pela equipe de gerenciamento de configurações.

Figura 29.3

Processo da gestão de alteração.

```

Solicita a alteração mediante o preenchimento do formulário de solicitação de alteração
Analisa a solicitação de alteração
If alteração é válida then
  Acessa como a alteração poderá ser implementada
  Acessa a alteração de custo
  Registra a solicitação no banco de dados
  Submete a solicitação ao controle de alteração
  If a alteração é aceita then
    repeat
      realize a alteração no software
      registre as alterações e link com a solicitação de alteração associada
      submete a alteração do software para a aprovação da qualidade
    até que a qualidade do software esteja adequada
    crie uma nova versão do sistema
  else
    rejeite a solicitação de alteração
  else
    rejeite a solicitação da alteração
  
```

O primeiro estágio no processo de gerenciamento de configurações é completar um formulário de solicitação de mudança (como proposto na figura abaixo). A definição de um formulário de solicitação de mudança é parte do processo de planejamento de CM.

Esse formulário registra a mudança proposta, o requisitante da mudança, a razão pela qual a mudança foi sugerida e a urgência da mudança (a partir do requisitante da mudança). Também registra a avaliação da mudança, a análise do impacto, o custo da mudança e as recomendações (pessoal de manutenção de sistema).

Figura 29.4

Formulário de solicitação de mudança parcialmente preenchido.

Formulário de Solicitação de Mudança	
Projeto: Proteus/Ferramenta PCL	Número: 23/02
Solicitante da mudança: I. Sommerville	Data: 1/12/02
Mudança solicitada: Quando um componente é selecionado da estrutura, apresentar o nome do arquivo onde ele está armazenado.	
Analista da mudança: G. Dean	Data da análise: 10/12/02
Componentes afetados: Display-Icon.Select, Display-Icon.Display	
Componentes associados: FileTable	
Avaliação da mudança: Relativamente simples de implementar se uma tabela de nome de arquivo estiver disponível. Requer o projeto e a implementação de um campo na tela. Não é requerida nenhuma mudança dos componentes associados.	
Prioridade da mudança: Baixa	
Implementação da mudança:	
Esforço estimado: 0,5 dia	
Data para o CCB: 15/12/02	Data de decisão do CCB: 1/2/03
Decisão do CCB: Aceita a mudança. Mudança a ser implementada no Release 2.1	
Implementador da mudança:	Data da mudança:
Data de submissão ao GQ:	Decisão do GQ:
Data da submissão ao CM:	
Comentários	

O maior problema no gerenciamento de mudanças é o acompanhamento do status da mudança. No entanto, ferramentas de acompanhamento de mudanças acompanham o status de cada solicitação de mudança e asseguram, automaticamente, que as solicitações de mudança sejam enviadas para as pessoas certas no tempo certo. São integrados a sistemas de e-mail, permitindo a distribuição eletrônica da solicitação de mudança.

As mudanças devem ser revisadas por um grupo externo, que decide se elas são ou não adequadas em termos de custo, a partir de um ponto de vista estratégico ou organizacional ao invés de um ponto de vista técnico. O grupo deve ser independente do responsável de projeto pelo sistema. Esse grupo é, algumas vezes, chamado de Comitê de Controle de mudanças (CCB) e pode conter representantes do cliente e do pessoal fornecedor.

Procedência histórica

É um registro das mudanças realizadas em um documento ou um componente de código. Deve registrar, em linhas gerais, a mudança feita, a lógica da mudança, quem fez a mudança e quando foi implementada.

Pode ser incluída como um comentário no código. Se um estilo de cabeçalho padrão é usado para a procedência histórica, as ferramentas podem processar isso automaticamente.

Figura 29.5

Informação de cabeçalho de componente.

```
// Projeto BANKSEC (IST 6087)
//
// BANKSEC-TOOLS/AUTH/RBAC/USER_ROLE
//
// Objeto: currentRole
// Autor: N. Perwaiz
// Data de criação: 10 de novembro de 2002
//
// (c) Lancaster University 2002
//
// Histórico de modificação
// Versão  Implementador  Data      Mudança      Razão
//1.0     J. Jones      1/12/2002  Adicionar cabeçalho  Submetido ao CM
//1.1     N. Perwaiz    9/4/2003   Novo campo          Solicit. de mud. R07/02
```

Gerenciamento de versões e release

É preciso:

- Definir um esquema para identificação e manutenção da rastreabilidade das versões de sistema.
- Planejar quando uma nova versão de sistema será produzida.
- Assegurar que procedimentos e ferramentas de gerenciamento das versões sejam adequadamente aplicados.
- Planejar e distribuir releases do novo sistema.

Versões / Variantes / Releases

- Versão - É uma instância de um sistema que é funcionalmente distinta, de alguma maneira, de outras instâncias de um sistema. A alteração pode envolver funções, desempenhos aprimorados, erros corrigidos, mudanças de configuração de hardware/software;
- Variante - É uma instância de um sistema que é funcionalmente idêntica, mas não funcionalmente distinta de outras instâncias de um sistema. São versões com variações pequenas;
- Release - É uma versão de um sistema distribuída para os usuários fora da equipe de desenvolvimento.

Identificação de Versões

Os procedimentos para identificação de versões devem definir uma maneira não ambígua de identificação de versões de componentes.

Existem três técnicas básicas para identificação de componentes:

- Numeração de versões.

É um esquema simples de numeração que usa uma derivação linear. Exemplo: V1, V1.1, V1.2, V2.1, V2.2 etc.

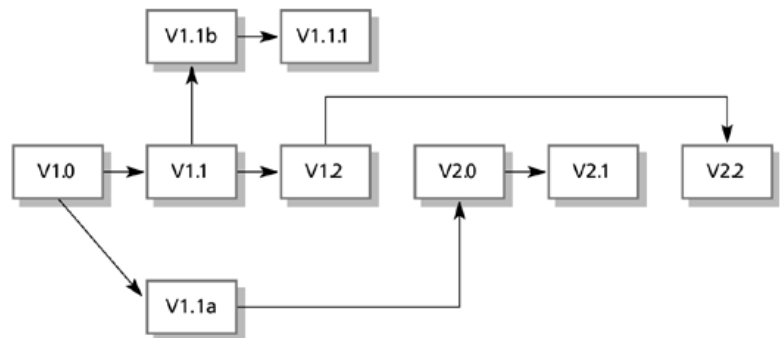
A estrutura de derivação real é uma árvore ou uma rede, e não uma seqüência.

Os nomes não são significativos.

Um esquema de hierarquia de atribuição de nomes conduz a poucos erros na identificação de versões.

Figura 29.6

Estrutura de derivação de versões.



- Identificação baseada em atributos.

Os atributos podem ser associados a uma versão com a combinação de atributos que a identificam. Exemplos de atributos: Data, Criador, Linguagem de Programação, Cliente, Status, etc.

É mais flexível do que um esquema explícito de atribuição de nomes para recuperação de versões. Contudo, pode causar problemas com a unicidade – o conjunto de atributos deve ser escolhido de tal modo que todas as versões possam ser unicamente identificadas.

Na prática, uma versão também necessita de um nome associado à facilidade de referência.

Uma importante vantagem da identificação baseada em atributos, é que ela pode apoiar consultas tais que você pode encontrar 'a mais recente versão em Java', etc. A consulta seleciona uma versão dependendo dos valores de atributos:

AC3D (linguagem =Java, plataforma = XP, data = Jan 2003).

- Identificação orientada a mudanças.

Integra as versões e as mudanças feitas para criar essas versões.

É usada para sistemas, não para componentes.

Cada mudança proposta tem um conjunto que descreve as mudanças feitas para implementar essa mudança.

Conjuntos de mudanças são aplicados em seqüência de tal modo que, em princípio, uma versão do sistema que incorpora um conjunto arbitrário de mudanças pode ser criado.

Gerenciamento de Releases

Os releases devem incorporar mudanças forçadas sobre o sistema por meio dos erros descobertos pelos usuários e pelas mudanças de hardware/software. Eles devem também incorporar a funcionalidade do novo sistema.

O planejamento de releases está relacionado a quando produzir uma versão de sistema como um release.

Um release de um sistema não é somente um conjunto de programas executáveis. Pode incluir também:

- Arquivos de configuração definindo como o release é configurado para uma instalação particular;
- Arquivos de dados necessários para a operação do sistema;
- Um programa de instalação para instalar o sistema no hardware alvo;
- Documentação eletrônica e em papel;
- Empacotamento e publicidade associada.

Os sistemas são, hoje em dia, normalmente liberados em discos ópticos (CD ou DVD) ou arquivos de instalação baixados via Web.

O cliente pode não querer um novo *release* do sistema. Eles podem estar satisfeitos com seu sistema atual, visto que a nova versão pode fornecer funcionalidade indesejada. O gerenciamento de releases não deve assumir que todas as releases anteriores foram aceitas. Todos os arquivos necessários para um release devem ser recriados quando um novo release for instalado.

A preparação e distribuição de um release de sistema é um processo dispendioso. Fatores, tais como a qualidade técnica do sistema, competição, requisitos de marketing e solicitações de mudança do cliente devem influenciar na decisão de quando produzir um novo release de sistema.

Alguns fatores que influenciam a estratégia de liberação de release estão abaixo identificados:

Tabela 29.1 Fatores que influenciam a estratégia de liberação de sistema

Fator	Descrição
Qualidade técnica do sistema	Se defeitos sérios de sistema, que afetam a forma como muitos clientes o usam, são comunicados, pode ser necessário criar um release de reparo de defeito. Entretanto, defeitos pequenos de sistema podem ser reparados pela criação de patches (frequentemente distribuídos pela Internet) que podem ser aplicados no release atual do sistema.
Mudanças de plataforma	Você pode criar um novo release de uma aplicação de software quando uma nova versão da plataforma do sistema operacional é liberada.
A quinta lei de Lehman (veja o Capítulo 21)	Ela sugere que o incremento da funcionalidade que é incluída em cada release é aproximadamente constante. Portanto, um release de sistema com funcionalidades novas significantes poderia ser seguido por um release de reparo.
Competição	Um release novo de sistema pode ser necessário porque uma competição de produtos está em curso.
Requisitos de marketing	O departamento de marketing de uma organização pode ter se comprometido a disponibilizar os releases numa determinada data.
Propostas de mudanças do cliente	Para sistemas sob encomenda, os clientes podem ter feito e pago por um conjunto específico de propostas de mudanças de sistema, e eles esperam um release de sistema tão logo quanto elas forem implementadas.

A criação de um release:

Envolve a coleta de todos os arquivos e a documentação necessária para criar um release de sistema.

Descrições de configuração têm de ser escritas para hardware diferente e scripts de instalação têm de ser escritos.

Documentação de um release: O release específico deve ser documentado para registrar exatamente quais arquivos foram usados para criá-lo. Isso permite que ele seja recriado, se necessário.

Construção de sistemas

A construção de um sistema é o processo de compilação e ligação de componentes de software em um sistema executável. Esse processo é, atualmente, sempre apoiado por ferramentas automatizadas que são dirigidas por 'scripts de construção'.

Sistemas diferentes são construídos a partir de combinações diferentes de componentes.

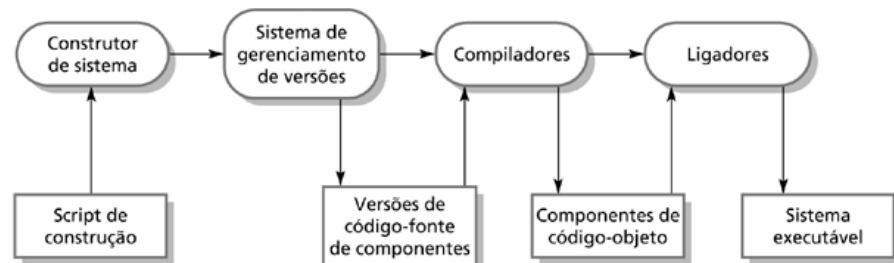
Algumas questões que devem ser respondidas:

- As instruções de construção incluem todos os componentes necessários? Quando existem muitas centenas de componentes constituindo um sistema, é fácil perder um deles. Isto deve ser normalmente detectado pelo linker.
- A versão apropriada de componente é especificada? É um problema mais grave, pois uma construção de sistema com a versão errada pode funcionar inicialmente, mas falhar após a entrega.
- Todos os arquivos de dados estão disponíveis? A construção não deve basear-se em arquivos de dados 'padrões', pois os padrões variam de lugar para lugar.
- As referências aos arquivos de dados dentro de componentes estão corretos? Embutir nomes absolutos em código quase sempre causa problemas, visto que as convenções de nomes diferem de lugar para lugar.
- O sistema que está sendo construído é para a plataforma certa? Algumas vezes, você deve construir para uma versão específica de OS ou configuração específica de hardware.
- A versão certa do compilador e outras ferramentas de software estão especificadas? Versões diferentes de compilador podem gerar código diferente, e o componente compilado exibirá comportamento diferente.

O processo normalmente seguido está identificado na figura abaixo:

Figura 29.7

Construção de sistemas.



Ferramentas CASE para gerenciamento de configurações

Os processos de CM são padronizados e envolvem a aplicação de procedimentos predefinidos. Eles lidam com grandes quantidades de dados e exigem atenção a detalhes. O apoio de uma ferramenta CASE para realização desses processos torna-se fundamental.

Ferramentas CASE maduras para apoiar o gerenciamento de configuração variam desde ferramentas *stand-alone* até *workbenches* integrados de CM. Sobre tais ferramentas pode-se comentar:

- Workbenches abertos - Ferramentas para cada estágio no processo de CM são integrados por meio de procedimentos organizacionais e scripts. Dá flexibilidade à seleção de ferramentas.
- Workbenches integrados - Fornece apoio integrado ao processo todo para gerenciamento de configuração. São ferramentas mais fortemente integradas e mais fáceis de usar. Contudo, são complexas e dispendiosas, e muitas organizações referem usar ferramentas de apoio individuais, mais baratas e simples.

O gerenciamento de mudanças é um processo procedural e, assim, pode ser modelado e integrado a um sistema de gerenciamento de versões.

Alguns recursos disponíveis em ferramentas de gerenciamento de mudanças podem ser os seguintes:

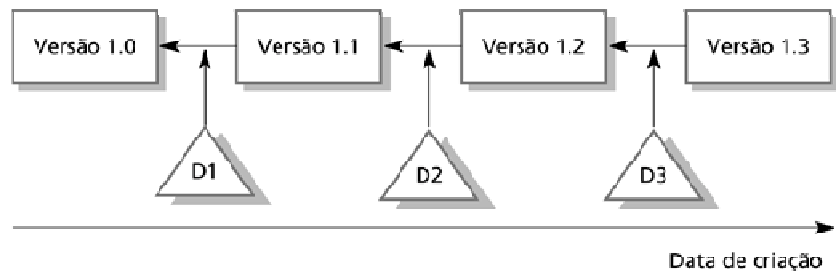
- Editor de formulários para apoiar o processamento de formulários de solicitação de mudança;
- Sistema de workflow para definir quem faz o quê e para automatizar a transferência de informações;
- Base de dados de mudanças que gerencia propostas de mudança e é ligado a um sistema de gerenciamento de versões (VM);
- Sistema de relato de mudanças que gera relatórios de gerenciamento sobre o status das solicitações de mudança.

O gerenciamento de versões envolve o gerenciamento de grande quantidade de informação e assegura que as mudanças de sistema sejam registradas e controladas. Alguns recursos habitualmente disponíveis para ferramentas de gerenciamento de versão são:

- Identificação de versões e releases - Os sistemas atribuem identificadores automaticamente quando uma versão é submetida a ele.
- Gerenciamento de armazenamento - O sistema armazena as diferenças entre as versões ao invés de todo o código da versão. Na figura abaixo essas diferenças entre versões é denominada de delta.

Figura 29.8

Versões baseadas em delta.



- Registro do histórico de mudanças - Registra as razões para criação de versão.
- Desenvolvimento independente - Somente uma versão por vez pode ser verificada para mudança. Trabalho paralelo sobre versões diferentes.
- Apoio a projetos - Pode gerenciar grupos de arquivos associados a um projeto ao invés de somente arquivos únicos.

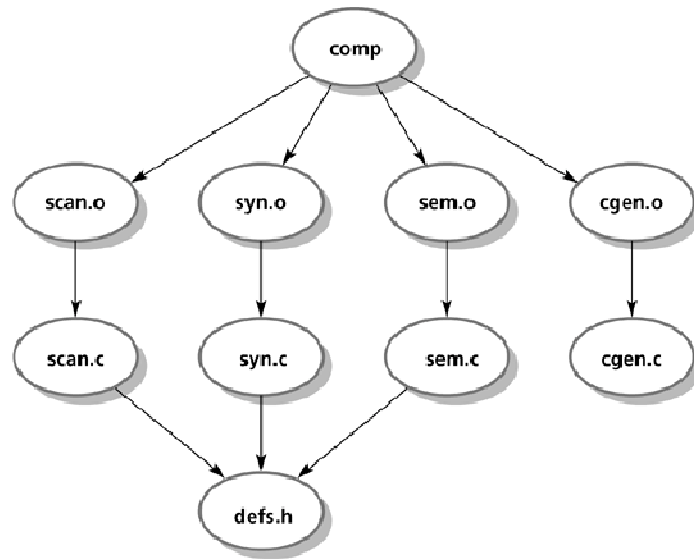
A construção de um sistema grande é computacionalmente dispendioso e pode levar várias horas de compilação e ligação de todos os componentes. Centenas de arquivos podem estar envolvidos.

As ferramentas de construção de sistemas visam auxiliar tal processo e podem fornecer:

- Uma linguagem de especificação de dependência e um interpretador associado;
- Seleção de ferramentas e apoio à instanciação;
- Compilação distribuída;
- Gerenciamento de objetos derivados. Um exemplo está demonstrado na figura abaixo:

Figura 29.9

Dependências entre componentes.



5.7 MÉTRICAS E MEDIÇÃO

Origem: Engenharia de Software, Roger Pressman, 2006.

Engenharia de Software, Ian Sommerville, 2007.

Internet.

Um elemento-chave de qualquer processo de engenharia é a medição. Usamos medidas para entender melhor os atributos dos modelos que criamos e para avaliar a qualidade dos produtos ou sistemas submetidos à engenharia que construímos.

A medição de software se dedica a derivar um valor numérico para algum atributo de um produto ou de processo de software. Assim, possibilita-se comparações objetivas entre técnicas e processos.

Embora algumas empresas tenham introduzido programas de medição, a maioria das organizações ainda não fazem uso sistemático de medição de software.

Também há de se destacar o fato de que existem poucos padrões estabelecidos nessa área.

Qualidade de Software

É a satisfação de requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo o software desenvolvido profissionalmente.

A definição serve para enfatizar três pontos:

- Requisitos de software são a fundação a partir da qual a qualidade é medida. Falta de conformidade com os requisitos é falta de qualidade.
- Normas especificadas definem um conjunto de critérios de desenvolvimento que guiam o modo pelo qual o software é construído. Se os critérios não são seguidos, quase sempre vai resultar em falta de qualidade.
- Há um conjunto de requisitos implícitos que frequentemente não são mencionados (por exemplo, o desejo de facilidade de uso). Se o software satisfaz seus requisitos explícitos, mas deixa de satisfazer os requisitos implícitos, a qualidade do software é suspeita.

A norma identifica seis atributos-chave de qualidade:

- Funcionalidade. Grau em que o software satisfaz as necessidades declaradas. Subatributos: adequabilidade, precisão, interoperabilidade, atendabilidade e segurança.
- Confiabilidade. Período de tempo em que o software está disponível para uso. Subatributos: maturidade, tolerância a falha, recuperabilidade.
- Usabilidade. Grau em que o software é fácil de usar. Subatributos: inteligibilidade, facilidade de aprendizado, operabilidade.
- Eficiência. Grau em que o software faz uso otimizado dos recursos do sistema. Subatributos: comportamento em relação ao tempo, comportamento em relação aos recursos.
- Manutenibilidade. Facilidade com a qual podem ser feitos reparados no software. Subatributos: analisibilidade, mutabilidade, estabilidade, testabilidade.
- Portabilidade. Facilidade com a qual o software pode ser transposto de um ambiente para outro. Subatributos: adaptabilidade, instalabilidade, conformidade, permutabilidade.

Um arcabouço para métricas de produto

Alguns conceitos necessários:

- Medida fornece uma indicação quantitativa da extensão, quantidade, dimensão, capacidade ou tamanho de algum atributo de um produto ou processo.
- Medição é o ato de determinar uma medida.
- Métrica é uma medida quantitativa do grau em que um sistema, componente ou processo possui um determinado atributo.
- Um engenheiro de software coleta medidas e desenvolve métricas de modo que indicadores sejam obtidos.
- Um indicador é uma métrica ou combinação de métricas que fornece profundidade na visão do processo de software, projeto de software ou produto em si.

Alguns exemplos de métricas:

- Tamanho do produto de software (ex: Número de Linhas de código);
- Número de pessoas necessárias para implementar um caso de uso;
- Número de defeitos encontrados por fase de desenvolvimento;
- Esforço para a realização de uma tarefa;
- Tempo para a realização de uma tarefa;
- Custo para a realização de uma tarefa;
- Grau de satisfação do cliente (ex: adequação do produto ao propósito, conformidade do produto com a especificação).

O Processo de medição pode ser caracterizado por cinco atividades:

- Formulação. A derivação de medidas e métricas de software adequadas para a representação do software que está sendo considerado.
- Coleta. Mecanismo usado para acumular os dados necessários para derivar as métricas formuladas.
- Análise. Cálculo de métricas e aplicação das ferramentas matemáticas.
- Interpretação. Avaliação das métricas em um esforço para ganhar profundidade na visão da qualidade de representação.

- Realimentação. Recomendações derivadas da interpretação das métricas de produto transmitidas à equipe de software.

Métricas de software serão úteis apenas se forem caracterizadas efetivamente e validadas de modo que seu valor seja aprovado. Os seguintes princípios são representativos de muitos que podem ser propostos para caracterização e validação de métricas:

- Uma métrica deve ter propriedades matemáticas desejáveis. Isto é, o valor da métrica deve estar em um intervalo significativo (por exemplo, zero a um, em que zero na realidade significa ausência, um indica o valor máximo, e 0,5 representa o “ponto médio”). Também, uma métrica que se propõe a estar em uma escala racional não deve ser composta por componentes que são apenas medidos em uma escala ordinal.
- Quando uma métrica representa uma característica de software que aumenta quando acontecimentos positivos ocorrem ou diminui quando acontecimentos indesejáveis são encontrados, o valor da métrica deve aumentar ou diminuir do mesmo modo.
- Cada métrica deve ser validada empiricamente em uma ampla variedade de contextos antes de ser publicada ou usada para tomar decisões. Uma métrica deve medir o fator de interesse, independentemente de outros fatores. Ela deve se “adequar” a sistemas grandes e funcionar em uma variedade de linguagens de programação e domínios de sistemas.

O que pode ser medido

Métricas podem ser aplicadas aos mais variados itens. No que se refere a software normalmente as medidas são relacionadas a:

- Recursos – Como estão sendo utilizados os recursos disponíveis;
- Produtos – Quanto mede e como está a qualidade do produto de software;
- Clientes – Como está sendo recebido o trabalho e os produtos do trabalho;
- Processo – Como está sendo realizado o trabalho de desenvolvimento de software;
- Gestão – Como está sendo feita a gestão de Ti.

Tipos de Métricas

Há atualmente uma série de classificações que podem ser utilizadas quando se discute métricas. Uma bastante coerente está abaixo exposta:

- Métricas Primárias (1ª. Ordem)
 - Também denominadas Medidas Diretas;
 - Apontamentos dos fatos (reais) – Medidas
 - Informações objetivas da realidade. Exemplos: defeitos, horas trabalhadas, custo, reclamações, ...
 - Focam na Produtividade;
 - Tendência à expressão numérica.
- Métricas Secundárias (2ª. Ordem)
 - Também denominadas Medidas Indiretas;
 - São indicadores, expressam um comportamento além dos números;
 - Focam na Qualidade;
 - Resultado de uma relação de: Métrica / Fator;
 - Exemplo: densidade de defeitos, defeitos por fase do projeto, ...

Métricas de Software

É o que permite que o software e o processo de software sejam quantificados.

Uma métrica de software é qualquer tipo de medição relacionada a um sistema de software, processo ou documentação associada. Por exemplo: Linhas de código em um programa, o número de defeitos relatados em um produto de software entregue, o número de pessoas-dia necessárias para desenvolver um componente.

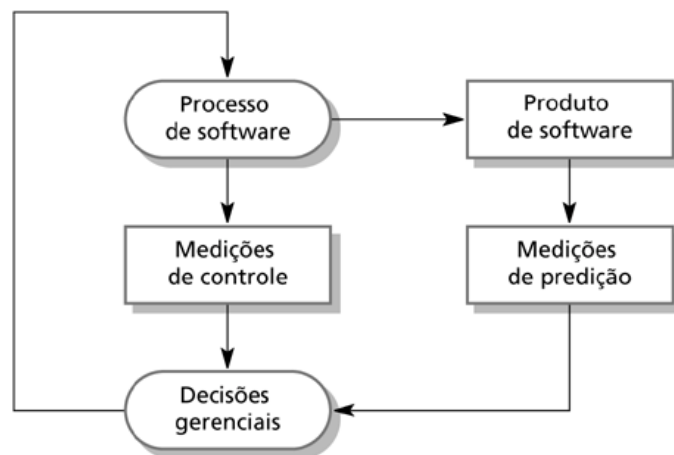
Pode ser usado tanto para prever atributos de produto, como para controlar o processo de software.

As métricas de produto podem se usadas para previsões gerais ou para identificar componentes anômalos.

As métricas de software podem ser tanto métricas de controle como de predição. Ambas podem influenciar na tomada de decisão de gerenciamento, conforme mostrado na figura abaixo. As métricas de controle são usualmente associadas ao processo de software, enquanto as de predição ao produto de software.

Figura 27.5

Medições de predição e de controle.



Geralmente, é impossível medir os atributos de qualidade de software diretamente. Os atributos de qualidade, como facilidade de manutenção, facilidade de compreensão e facilidade de uso são atributos externos que se relacionam a como desenvolvedores e usuários vêem o software. Eles são influenciados por muitos fatores e não existe uma maneira simples para medi-los. Em vez disso, mede-se algum atributo interno do software (como o tamanho) e presume-se que há um relacionamento entre o que você pode medir e o que realmente quer conhecer.

Assim, uma propriedade de software pode ser medida. Mas há um caminho a seguir.

Existe o relacionamento entre o que podemos medir e o que queremos conhecer. Podemos somente medir atributos internos, mas estamos, muitas vezes, mais interessados em atributos externos de software.

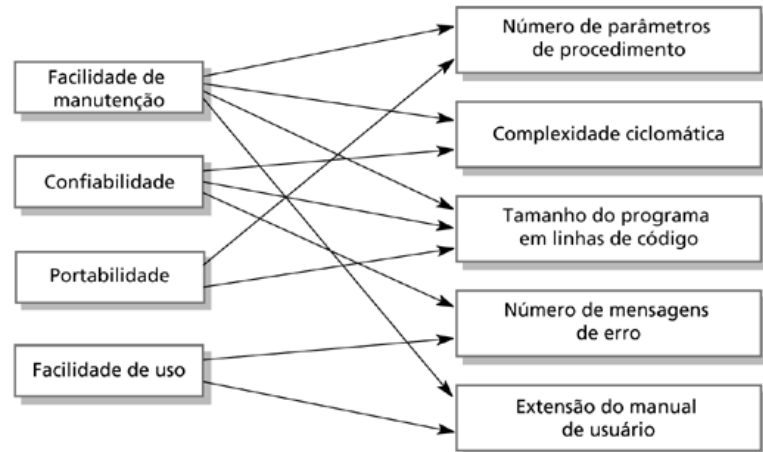
Pode ser difícil relacionar o que pode ser medido em atributos externos de qualidade desejáveis. Alguns exemplos podem ser visto na figura a seguir.

Se a medida do atributo interno deve ser uma previsão útil de uma característica externa do software, então três condições devem ser mantidas:

- O atributo interno deve ser medido com precisão;
- Deve existir um relacionamento entre o que podemos medir e o atributo externo de comportamento no qual estamos interessados;
- Esse relacionamento é compreendido, foi validado e pode ser expresso em termos de uma fórmula ou modelo.

Figura 27.6

Relacionamentos entre atributos de software internos e externos.



O Processo de Medição

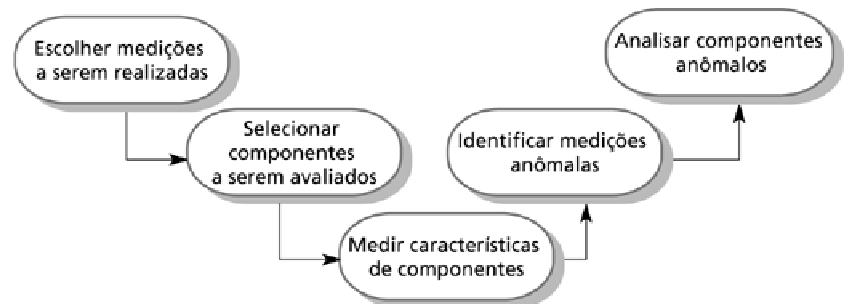
Um processo de medição de software pode ser parte de um processo de controle de qualidade. Os dados coletados durante este processo devem ser mantidos como um recurso da organização.

Uma vez que um banco de dados de medições foi estabelecido, comparações entre projetos tornam-se possíveis.

Um exemplo de processo está destacado na figura a seguir.

Figura 27.7

Processo de medição de produto.



Sobre a coleta de dados:

Um programa de métricas deve ser baseado em um conjunto de dados de produto e de processo.

Os dados devem ser coletados imediatamente (não em retrospecto) e, se possível, automaticamente.

Há três tipos de coleta automática de dados:

- Análise estática de produto;
- Análise dinâmica de produto;
- Comparação de dados de processo.

Sobre precisão de dados:

- Não colete dados desnecessários - As questões a serem respondidas devem ser decididas previamente e os dados necessários devem ser identificados.
- Conte às pessoas por que os dados estão sendo coletados - Não deve ser parte da avaliação de pessoal.
- Não dependa da memória - Colete dados quando são gerados, e não depois que um projeto foi finalizado.

Classes de métrica de produto

- Métricas dinâmicas que são coletadas por meio das medições realizadas em um programa em execução;
- Métricas estáticas que são coletadas pelas medições realizadas em representações do sistema.

As métricas dinâmicas ajudam a avaliar a eficiência e a confiabilidade; métricas estáticas ajudar a avaliar a confiabilidade, a facilidade de compreensão e a facilidade de manutenção.

Métricas dinâmicas são bem relacionadas aos atributos de qualidade de software. É relativamente fácil medir o tempo de resposta de um sistema (atributo de desempenho) ou o número de falhas (atributo de confiabilidade).

Métricas estáticas têm um relacionamento indireto com atributos de qualidade. Você necessita tentar e derivar um relacionamento entre essas métricas e as propriedades, tais como complexidade, facilidade de compreensão e de manutenção.

A tabela abaixo apresenta algumas métricas genéricas.

Tabela 27.4 Métricas estáticas do produto de software

Métrica de software	Descrição
Fan-in/Fan-out	Fan-in é uma medida do número de funções ou métodos que chamam alguma outra função ou método (digamos X). Fan-out é o número de funções chamadas pela função X. Um valor alto para fan-in significa que X está firmemente acoplado com o resto do projeto, e mudanças em X terão grande impacto. Um alto valor para fan-out sugere que a complexidade geral de X pode ser alta devido à complexidade da lógica de controle necessária para coordenar os componentes chamados.
Extensão de código	É uma medida do tamanho de um programa. Geralmente, quanto maior for o tamanho do código de um componente, mais complexo e propenso a erros esse componente será. A extensão de código foi mostrada como uma das métricas mais confiáveis de previsão de propensão a erros em componentes.
Complexidade ciclomática	É uma medida da complexidade de controle de um programa. Essa complexidade de controle pode estar relacionada à facilidade de compreensão do programa. Explico como calcular a complexidade ciclomática no Capítulo 22.
Extensão de identificadores	É uma medida da extensão média de identificadores distintos em um programa. Quanto maiores forem os identificadores, mais eles serão significativos e, por isso, mais compreensível será o programa.
Profundidade de aninhamento de declarações condicionais	É uma medida da profundidade de aninhamento de declarações IF em um programa. As declarações IF profundamente aninhadas são difíceis de compreender e são potencialmente propensas a erros.
Índice de Fog	É uma medida da extensão média das palavras e das sentenças em documentos. Quanto maior for o valor para o índice de Fog, mais difícil será a compreensão de documentos.

Já a próxima tabela apresenta algumas métricas orientadas a objetos.

Tabela 27.5 Métricas orientadas a objetos

Métrica orientada a objetos	Descrição
Profundidade de árvore de herança	Representa o número de níveis discretos na árvore de herança da qual as subclasses herdam atributos e operações (métodos) das superclasses. Quanto maior for a profundidade da árvore de herança, mais complexo será o projeto. Muitas classes de objeto podem ser compreendidas ao se compreender as classes de objeto das folhas da árvore.
<i>Fan-in/fan-out</i> de método	Está diretamente relacionado com fan-in e fan-out, conforme descrito na Tabela 27.4, e significa essencialmente a mesma coisa. Contudo, pode ser apropriado fazer uma distinção entre chamadas de outros métodos dentro do objeto e chamadas com base em métodos externos.
Métodos ponderados por classe	Este é o número de métodos incluídos em uma classe, ponderados pela complexidade de cada método. Portanto, um método simples pode ter uma complexidade de 1 e um método grande e complexo pode ter um valor muito mais alto. Quanto maior for o valor dessa métrica, mais complexa será a classe de objeto. Os objetos complexos são, provavelmente, mais difíceis de compreender. Eles podem não ser logicamente coesos, de modo que não podem ser reusados eficazmente como superclasses em uma árvore de herança.
Número de operações sobrescritas	Este é o número de operações, em uma superclasse, sobrescritas em uma subclasse. Um valor alto dessa métrica indica que a superclasse usada pode não ser uma classe-mãe apropriada para a subclasse.

Análise de medições

Nem sempre é óbvio o que os dados significam. A análise de dados coletados é muito difícil. Estatísticos profissionais devem ser consultados, se estiverem disponíveis.

A análise de dados deve levar em conta as circunstâncias locais.

As medições podem, às vezes, revelar algumas surpresas. Por exemplo:

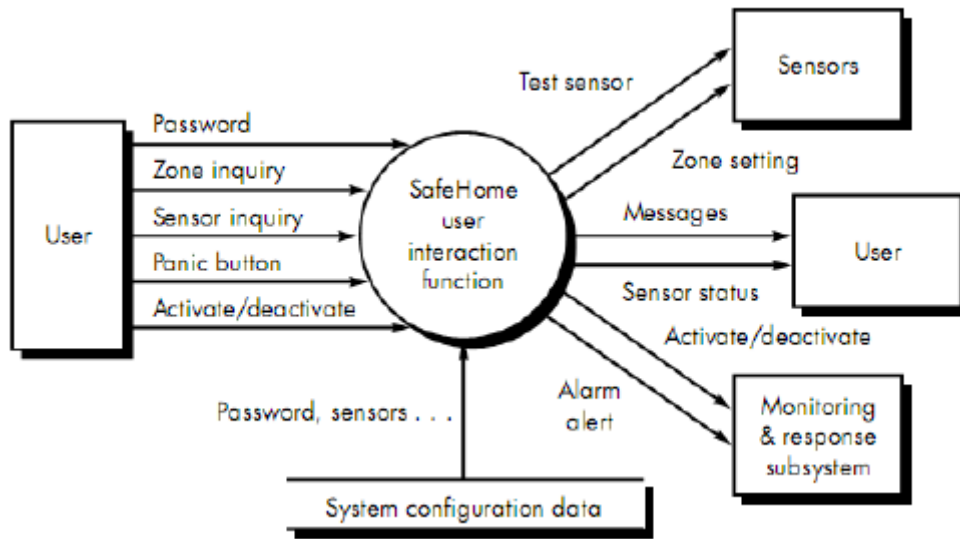
Uma redução do número de defeitos em um programa conduz a um número maior de chamadas de help desk - O programa é, agora, considerado mais confiável e, assim, tem um mercado diverso mais amplo. O percentual de usuários que chamam o help-desk pode ter diminuído, mas o total pode aumentar; Um sistema mais confiável é usado de uma maneira diferente de um sistema onde os usuários trabalham em torno dos defeitos. Isso conduz a mais chamadas de help desk.

Isso mostra que a interpretação de dados quantitativos sobre um produto ou um processo é subjetiva. Os processos e os produtos em medição não são isolados de seu ambiente e as mudanças para esse ambiente podem tornar inválidas as comparações de dados. Os dados quantitativos sobre atividades humanas não podem sempre ser considerados pelo valor em si. As razões implícitas que poderiam ser consideradas para os valores medidos devem ser investigadas.

Métricas baseadas em função

A métrica ponto por função (FP) pode ser usada efetivamente como um meio para medir a funcionalidade entregue por um sistema. Usando dados históricos, o FP pode então ser usado para

- Estimar o custo ou esforço necessário para projetar, codificar e testar o software;
- Prever o número de erros que vão ser encontrados durante o teste; e
- Prever o número de componentes e/ou o número de linhas de código projetadas no sistema implementado.



Três entradas externas – senha, botão de pânico e ativar/desativar – são mostradas na figura junto com duas consultas externas – consulta de zona e consulta de sensores. Um arquivo (arquivo de configuração do sistema) é mostrado. Duas saídas externas (mensagens e estado do sensor) e quatro arquivos de interface externa (teste de sensor, estabelecimento de zona, ativar/desativar e alerta do alarme) também estão presentes.

Measurement parameter	Count	Weighting Factor			=	Result
		Simple	Average	Complex		
Number of user inputs	3	3	4	6	9	
Number of user outputs	2	4	5	7	8	
Number of user inquiries	2	3	4	6	6	
Number of files	1	7	10	15	7	
Number of external interfaces	4	5	7	10	20	
Count total	→					50

$$FP = 50 \times [0,65 + (0,01 \times 46)] = 56$$

Considere que dados anteriores indicam que um FP traduz-se em 60 linhas de código (uma linguagem orientada a objetos deve ser usada) e que 12 FPs são produzidos para cada pessoa-mês de esforço. Considere também que projetos anteriores encontraram uma média de três erros por ponto por função durante as revisões de análise e projeto e quatro erros por ponto por função durante o teste de unidade e integração.

Esses dados fornecem informações históricas importantes, que podem ser utilizadas durante o planejamento do projeto.

Medições de Processo

Medições de processo são dados quantitativos sobre o processo de software, sendo essencial para o aprimoramento de um processo.

Sempre que possível, dados quantitativos de processo devem ser coletados. Contudo, onde as organizações não têm padrões de processo claramente definidos, isso é muito difícil, já que você não sabe o que medir. Um processo deve ser definido antes que qualquer medição seja possível.

As medições de processos devem ser usadas para avaliar os aprimoramentos de processos. Mas isso não significa que as medições devem ser dirigidas aos aprimoramentos. O direcionador de aprimoramento deve ser o objetivo da organização.

Três classes de métricas de processo podem ser coletadas:

- Tempo para que as atividades de processo sejam concluídas. Por exemplo, o prazo ou o esforço para concluir uma atividade ou um processo.
- Recursos necessários para processos ou atividades. Por exemplo, esforço total em pessoas-dia.
- Número de ocorrências de um evento específico. Por exemplo, número de defeitos descobertos.

A dificuldade fundamental da medição de processo é justamente saber o que mudar. Para auxiliar a decidir quais medições devem ser realizadas e como elas devem ser usadas foi proposto, por Basili e Rombach, o paradigma Objetivo-Questão-Métrica (GQM – *Goal/Question/Metric*).

- Objetivos: O que a organização está tentando obter? O objetivo do aprimoramento de processo é satisfazer esses objetivos.
- Questões: Questões sobre áreas de incerteza relacionados aos objetivos. Você necessita de conhecimento do processo para deduzir essas questões.
- Métricas: Medições a serem coletadas para responder às questões.

5.8 DICAS DE LEITURA

- PRESSMAN, Roger. **Engenharia de software**. 2006. McGraw-Hill. Capítulo 15 – Métricas de Produto para software; Capítulos 22 – Métricas de Processo e Projeto; Capítulo 26 – Gestão de Qualidade; Capítulo 27 – Gestão de Modificações;
- SOMMERVILLE, Ian. **Engenharia de software**. 2007. Pearson Education. Capítulo 27 – Gerenciamento de Qualidade; Capítulo 28 – Aprimoramento de Processo; Capítulo 29 – Gerenciamento de Configurações;
- KOSCIANSKI, André. SANTOS SOARES, Michel dos. **Qualidade de software**. 2006.
Link para o livro, incluindo o capítulo 1 disponibilizado:
<http://www.novateceditora.com.br/livros/qualidadesoftware/>
- BERTOLLO, Gleidson. Dissertação de Mestrado. **Definição de processos em um ambiente de desenvolvimento de software**. UFES. 2006.
- Portal da Qualidade de Software: <http://qualidadesoftware.org.br/?id=1062§ion=noticias>
- Artigo - Uma ontologia de Qualidade de Software, autores: Kátia Cristina Duarte e Ricardo de Almeida Falbo. Link: <http://www.inf.ufes.br/~falbo/download/pub/Wqs2000.pdf>
- Qualidade de Software – Visões de Produto e Processo de Software. Link: <http://www.prodepa.psi.br/sqp/pdf/Qualidade%20de%20Software.pdf>