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, analizando 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 conseqüências 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 contra-exemplos 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 deteção 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 evoluido 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 conseqüência 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 conseqüência, 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 capabilidade. 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 conseqüência direta o aumento da capabilidade 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.

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.

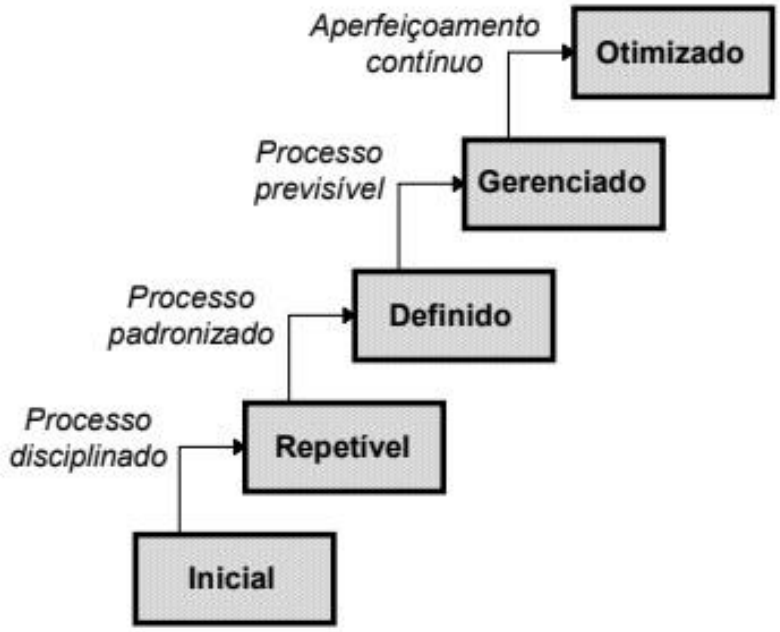

Figura 2.1 - Os níveis de maturidade de um processo de desenvolvimento de software.

A capabilidade 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 freqüentes, não apenas de um projeto a outro, mas também durante a realização de um mesmo projeto.

Neste nível, estimação 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 capabilidade de uma empresa situada neste nível pode ser caracterizada como disciplina, 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 capabilidade 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 capabilidade 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 capabilidade do processo de desenvolvimento de software. A figura 2.2 ilustra os componentes relacionados a este detalhamento.

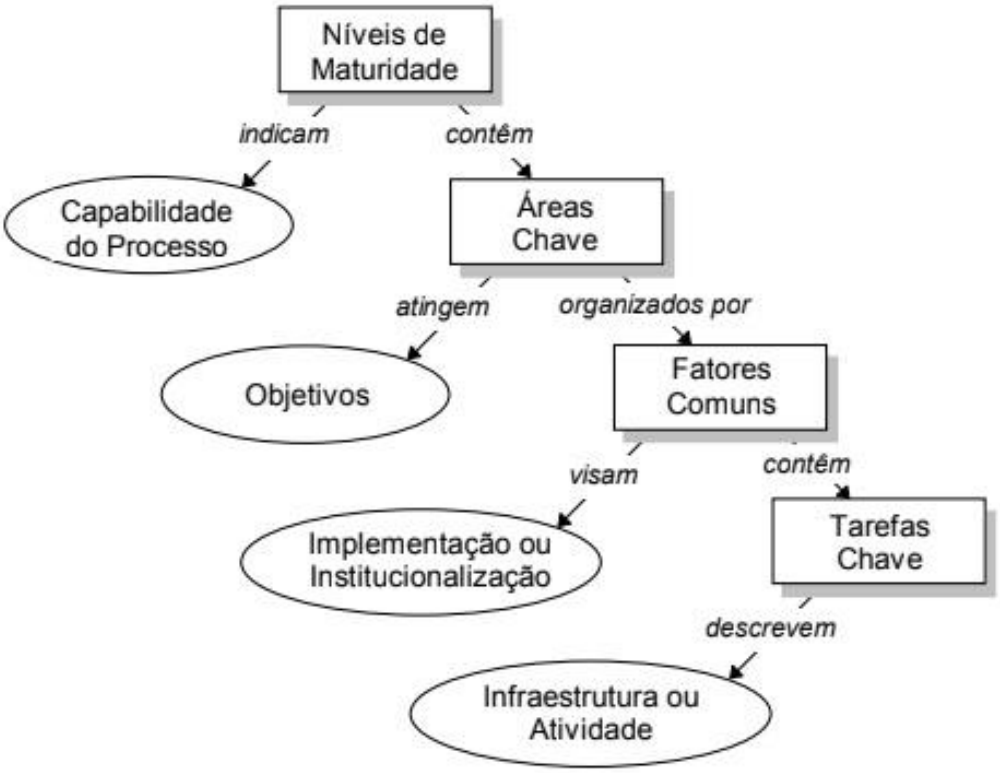

Figura 2.2 - Estrutura dos níveis CMM.

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 capabilidade 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.

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 capabilidade do seu processo de software.

Nível	Áreas Chave
Repetível (2)	• *Gerenciamento de requisitos* • *Planejamento do processo de desenvolvimento* • *Acompanhamento do projeto* • *Gerenciamento de subcontratação* • *Garantia de qualidade* • *Gerenciamento de configuração*
Definido (3)	• *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)	• *Gerenciamento da qualidade de software* • *Gerenciamento quantitativo do processo*
Otimizado (5)	• *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.

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;

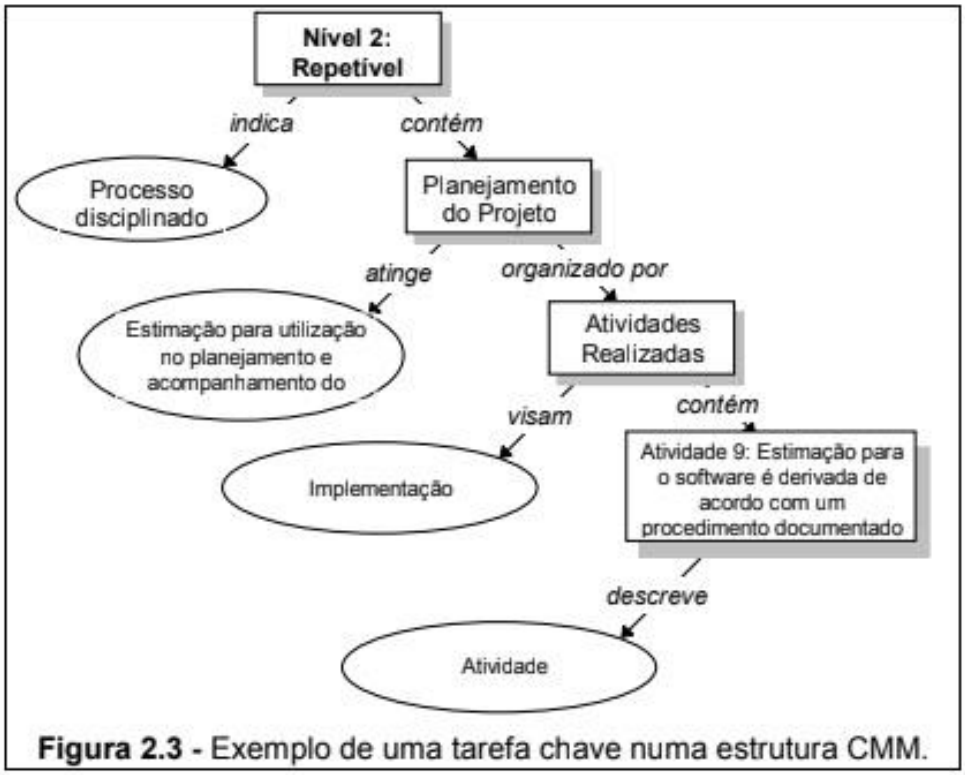

Figura 2.3 - Exemplo de uma tarefa chave numa estrutura CMM.

- 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.

Acerca de qualidade de software, julgue o seguinte item.

1 - Revisões e inspeções são atividades de controle de qualidade que verificam a qualidade dos entregáveis de projeto. Isso envolve examinar o software, sua documentação e os registros do processo para descobrir erros e omissões e verificar se os padrões de qualidade foram seguidos. Certo (x) Errado ( )

2 - Uma parte importante da garantia de qualidade é a definição ou a seleção de padrões que devem ser aplicados no processo de desenvolvimento de software. Como parte desse processo de QA, devem ser escolhidos ferramentas e métodos para suportar o uso desses padrões. Certo (x) Errado ( )

3 - A garantia da qualidade estabelece a infraestrutura que suporta métodos sólidos de engenharia de software, gerenciamento racional de projeto e ações de controle de qualidade. Certo (x) Errado ( )

4 - O controle de qualidade engloba um conjunto de ações de engenharia de software que ajuda a garantir que cada produto resultante atinja suas metas de qualidade. Os modelos são revistos de modo a garantir que sejam completos e consistentes. Certo (x) Errado ( )