

## Capítulo 2

### Métricas de Software

---

*Este capítulo apresenta os principais conceitos e fundamentos sobre métricas e medições no contexto da Engenharia de Software, destacando as métricas de estimativa, grupo onde se insere a Análise de Pontos por Função.*

Os sistemas de computação estão cada vez mais presentes nas diversas áreas da vida humana (Oman, 1998). Isto acontece não apenas em relação às tarefas corriqueiras do cotidiano das pessoas, mas também na competitividade das organizações que se tornaram dependentes da qualidade do software desenvolvido ou adquirido para uso próprio.

Este acréscimo de importância do software provocou um incremento na quantidade de seus requisitos. Conseqüentemente, há a necessidade de que o gerente do projeto mantenha um controle preciso, previsível e repetível sobre o processo de desenvolvimento de software (Zuse, 1991). Isto pode ser melhor operacionalizado, quando a organização dispõe de um processo padrão para o desenvolvimento de seus produtos de software. Emam *et al.* (1998) definem o processo padrão como o processo básico que conduz o estabelecimento de um processo comum na organização. Para Maidantchik *et al.* (1999), um processo padrão define uma estrutura única a ser adotada pelas equipes envolvidas em um projeto de software, independentemente das características do produto a ser desenvolvido. No entanto, o processo padrão deverá ser adaptado a cada projeto, considerando-se fatores como o paradigma de desenvolvimento, o ciclo de vida adotado, as ferramentas e os métodos empregados, entre outros (Machado, 2000).

A utilização de um processo padrão na organização conduz a um controle gerencial mais acurado. Porém, como “não se pode controlar o que não se pode medir” (Demarco, 1989), as medições realizadas ao longo do ciclo de vida do software são a maneira mais eficiente de executar tal controle.

A seguir, são apresentadas as principais motivações de se aplicar processos de medição em engenharia de software.

## **2.1 Medições em Engenharia de Software**

A qualidade de produtos e processos de software é uma exigência cada vez mais presente. Segundo Strigel (2000), tal fato está provocando a eliminação de uma cultura tolerante a erros, não se admitindo mais que projetos de desenvolvimento de software estourem seus orçamentos ou cronogramas, pois isto coloca as organizações sob a ameaça de perder boas oportunidades de negócio ou, até mesmo, ir à falência.

A comunidade de engenharia de software está continuamente buscando novos métodos e novas ferramentas para desenvolver software de maior qualidade, em uma tentativa de solucionar estas questões. Porém, segundo (Fenton, 1997), muitos projetos ainda são deficientes porque:

- *não se estabelecem alvos mensuráveis, não havendo, portanto, garantias em relação à usabilidade, confiabilidade ou manutenibilidade do software;*
- *não há controle de custos;*
- *a qualidade do produto não foi quantificada nem prevista;*
- *novas tecnologias são adotadas sem estudo prévio a respeito de características como eficiência e eficácia.*

Os aspectos acima citados podem ser cobertos por um processo de medição de software. Porém, em muitas organizações, a medição ainda é vista como um ornamento ou é praticada com inconstância, inconsistência e, o que é pior, sem planejamento.

A engenharia de software vem absorvendo lições ensinadas e práticas consolidadas por outros ramos da engenharia, onde as medições constituem-se em procedimentos fundamentais no decorrer das atividades do processo.

De uma forma geral, a literatura ainda utiliza os termos métricas e medições indistintamente, para designar o conjunto de atividades realizadas que quantificam algum processo

ou produto do mundo real. Parte desta indisciplina deve-se ao fato de as palavras *measure* e *measurement* poderem ser usadas tanto como verbo ou como substantivo na língua inglesa.

No entanto, é importante que sejam observadas as definições destes termos, para que a comunicação entre os profissionais da área possa fluir sem ruídos. Lemes e Fernandes (1997) e Duarte (2000) estabelecem os conceitos:

- *Métrica*: propriedade utilizada como unidade de medição. Por exemplo, número de linhas de código.
- *Medida*: valor que é associado às métricas, correspondendo ao grau que uma característica é satisfeita. Consiste na aplicação das métricas para obtenção de dados quantitativos.

O IEEE (1993) define indicador como um dispositivo ou variável que pode ser atribuído a um determinado estado, baseado nos resultados ou na ocorrência de uma condição específica.

Portanto, a medição ou mensuração é o processo pelo qual números ou símbolos são designados para atributos de entidades do mundo real com o intuito de descrevê-las de acordo com regras bem definidas. A próxima seção aborda algumas formas de classificar as métricas de software.

## **2.2 Classificação das Métricas**

As métricas podem ser referentes a um ou mais atributos de um sistema, sendo importante agrupá-las ou categorizá-las de acordo com suas características mais comuns. No entanto, é possível encontrar diversos esquemas de classificação das métricas, cada um deles se revelando mais apropriado para um tipo específico de análise (complexidade de software, qualidade do produto etc.).

Em (Basili, 1980), há uma classificação específica para as métricas de complexidade de software. Estas são apontadas como estáticas, caso meçam a complexidade do código numa data específica, ou como históricas, quando avaliam a qualidade do código ao longo de um certo período de tempo.

A classificação de (Sheppard, 1988) separa as métricas de engenharia de software em métricas de código (linhas de código fonte, complexidade de McCabe etc), métricas de projeto (acoplamento do módulo etc) e métricas de especificação (pontos por função etc). Os dois esquemas apresentados concentram suas atenções para a complexidade do software. (Boehm *et al.*, 1976; Gilb, 1987) propõem métricas que avaliam a qualidade do produto.

Outra classificação largamente utilizada é a de (Fenton, 1991), resumida na Tabela 2.1.

Tabela 2.1: Classificação de Fenton

Entidades	Atributos	
	Internos	Externos
<b>Produtos</b>		
Especificações	Tamanho, reuso, modularidade, redundância ...	Legibilidade, manutenibilidade ...
Projetos	Tamanho, reuso, acoplamento, modularidade ...	Qualidade, complexidade ...
Código	Funcionalidade, complexidade do algoritmo	Confiabilidade, usabilidade, reusabilidade ...
Dados de teste	Tamanho, cobertura	Qualidade, reusabilidade ...
...		
<b>Processos</b>		
Especificação	Tempo, esforço, número de mudanças nos requisitos ...	Qualidade, custo, estabilidade...
Projeto Detalhado	Tempo, esforço, número de falhas de especificação encontradas ...	Custo ...
Teste	Tempo, esforço, número de falhas no código encontradas ...	Custo, estabilidade
...		
<b>Recursos</b>		
Pessoas	Idade, custo ...	Produtividade, experiência ...
Equipes	Tamanho, nível de comunicação, estruturação ...	Produtividade, qualidade ...
Organizações	Tamanho, Certificação ISO, Nível CMM ...	Maturidade, liquidez
Software	Preço, tamanho	Usabilidade, confiabilidade ...
Hardware	Preço, velocidade, tamanho da memória	Confiabilidade
Escritórios	Tamanho, temperatura, luminosidade ...	Conforto, qualidade ...

Esta classificação pressupõe que qualquer métrica é uma tentativa de medir ou prever um atributo externo ou interno de algum produto, processo ou recurso. Este esquema é o mais geral de todos até então descritos, podendo ser ponto de partida para a criação de outros esquemas ou servir como guia para a seleção de métricas que juntas irão compor o conjunto de métricas adotadas por uma determinada organização.

A seguir estão apresentados alguns princípios que devem reger a construção do plano de medição, assim como características referentes às métricas.

### **2.3 Princípios de Medição e Atributos de Métrica Efetiva**

Antes de expor sobre o plano de medição propriamente dito, faz-se necessário descrever os princípios de medição, bem como listar os atributos de uma métrica efetiva. Segundo Roche (1994), um processo de medição é caracterizado por cinco atividades: *formulação, coleta, análise, interpretação e feedback*.

A formulação é a derivação de medidas e métricas de software apropriadas para a representação do software em avaliação. A coleta é o mecanismo usado para capturar os dados necessários, que alimentarão as métricas formuladas. A análise é o cálculo das métricas. A interpretação é a avaliação dos resultados e o *feedback* são as recomendações derivadas da interpretação.

Os princípios que devem estar associados à formulação são:

- *Os objetivos da medição devem estar determinados antes que a coleta se inicie;*
- *Cada métrica deve estar definida de uma maneira inequívoca; e*
- *As métricas devem ser derivadas com base em teoria válida para o domínio da aplicação.*

Para as atividades de coleta e análise são sugeridos os seguintes princípios:

- *Sempre que possível, a coleta e a análise dos dados devem ser automatizadas;*

- *Técnicas válidas de estatísticas devem ser aplicadas para estabelecer relações entre os atributos internos do produto e as características externas de qualidade; e*
- *Diretrizes e recomendações interpretativas devem ser estabelecidas para cada métrica.*

Para que as métricas sejam utilizadas com sucesso, recomenda-se que possuam os seguintes atributos (Ejiogu, 1991):

- *Simples para calcular seus valores;*
- *Intuitiva;*
- *Consistente e objetiva, produzindo resultados iguais a partir das mesmas informações;*
- *Consistente no uso de unidades e dimensões;*
- *Independente da linguagem de programação; e*
- *Útil no feedback de informações sobre a qualidade do produto final.*

A próxima seção apresenta uma visão geral do plano de medição.

## **2.4 Plano de Medição**

A medição de software é uma avaliação quantitativa de qualquer aspecto do processo de engenharia de software, produto ou contexto. A medição serve para entender ou ajudar no controle, previsão e melhoria do que produzir e como produzir.

Segundo Shari Pfleger em (Clements, 2000), a medição serve para desenvolver teorias, aplicá-las, avaliá-las e usá-las para melhorar produtos, processos e recursos. Semelhantemente, as medições de software ajudam no entendimento de como o software é criado e como este evolui, permitindo a seguinte evolução: entendimento, previsão, mudança e melhoria.

Desta forma, com o objetivo de entender e melhorar seus processos, uma organização deve selecionar as métricas que comporão seu plano de medição. O plano de medição é um documento que, além das métricas em si, deve conter (Fenton, 1998):

- *Os objetivos do projeto;*
- *O que de fato está sendo medido;*
- *Quais os momentos do processo de desenvolvimento onde deve haver medições; e*
- *Como identificar as ferramentas, técnicas e membros da equipe envolvidos na coleta e análise das métricas, indicando quem é o responsável por cada tarefa.*

Outro fator importante é a consciência de que as medições devem estar intimamente relacionadas com os objetivos do negócio. Logo, o plano de medição deve contemplar métricas, que forneçam informações aos diversos níveis da organização, conforme a *Tabela 2.2*.

Tabela 2.2: Retorno das medições para os níveis da organização (Layman, 1998)

Nível Organizacional	Informações
Empresa	Retorno do investimento, comparações, ...
Processo	Custo das atividades, <i>benchmarks</i> , ...
Projeto	Cronograma, custo dos recursos, crescimento, estabilidade, ...
Produto	Confiabilidade, manutenibilidade, ...

É recomendável que a organização adote alguma abordagem indicada pela literatura para compor seu plano de medição. Duas técnicas são sugeridas por Fenton (2000):

- *O trabalho de Grady e Caswell (Grady, 1987), que foi o primeiro relatório sobre um plano de medição adotado com sucesso e contém diretrizes que influenciaram muitos outros trabalhos; e*
- *O GQM, desenvolvido por Basili (1984), o qual se apresenta como esquema para garantir que as métricas sempre serão dirigidas pelos objetivos da organização.*

Para que um plano de medição possa ser implantado na organização satisfatoriamente, seguem algumas valiosas contribuições de Jones (1997), Layman (1998), Tavares (1999) e Solingen (2001):

- *Deve-se medir processos e não pessoas;*
- *Os objetivos a alcançar e os resultados esperados devem estar explícitos;*
- *As medições devem estar integradas aos processos de desenvolvimento de software ao longo do ciclo de vida;*
- *O apoio da alta administração é fundamental;*
- *Os profissionais envolvidos devem ser bem treinados no processo de medição;*
- *Todos os envolvidos devem estar cientes a respeito dos benefícios e das limitações do processo de medição;*
- *Deve-se trabalhar as questões culturais e mudanças de hábitos necessárias; e*
- *Deve-se medir apenas o que é importante, mantendo-se sempre que possível um plano simples.*

Os resultados das medições devem ser armazenados para futuras comparações, pois o próprio processo de medição está sob constante melhoria.

A medição é importante para que a empresa possa compreender melhor os relacionamentos entre seus processos, produtos e recursos. Assim sendo, seu corpo funcional estará apto para tomar decisões sobre critérios de qualidade e adoção de novas tecnologias.

O planejamento do desenvolvimento será fortalecido com cronogramas e custos mais precisos. Além disso, informações complementares podem identificar dificuldades encontradas e pontos do processo que necessitem de refinamentos. Todos estes benefícios contribuem para que a empresa alcance excelência no desenvolvimento de software, progredindo em relação aos níveis de maturidade determinados pelo CMM (CMM, 1993) ou atendendo aos requisitos estabelecidos pelas diretrizes da família ISO 9000.



No entanto, urge esclarecer que a medição não é a bala de prata (*silver bullet*), que irá eliminar todos os problemas concernentes ao processo de desenvolvimento de software. Um processo de medição, executado de acordo com um plano de medição claro e bem definido, está credenciando a empresa para a emergente industrialização de produtos de software.

A próxima seção aborda o paradigma GQM, o qual se destaca na literatura como o mais indicado para a construção de um plano de medição.

## 2.5 GQM

O GQM (*Goal/Question/Metrics*) foi oficializado em 1984 (Basili, 1984) como resultado de estudos das relações entre objetivos e métricas, realizados na NASA, pelos professores Victor Basili e David Weiss. Trata-se de um paradigma que permite o estudo do processo de desenvolvimento de software, considerando suas características de qualidade mais relevantes (Solingen, 1997a), de acordo com as necessidades (e cultura) particulares de cada organização. O processo de definição e interpretação do GQM pode ser visualizado abaixo.

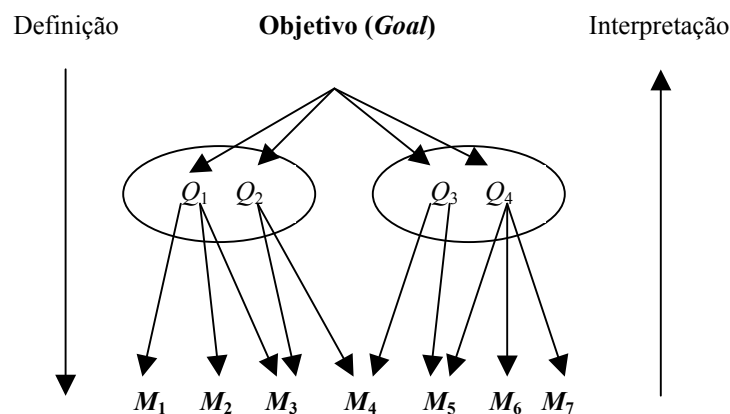


Figura 2.1: O Paradigma GQM

O GQM é uma abordagem que procura integrar objetivos com modelos de processos de software e de produtos de software, refletindo os interesses de qualidade particulares de cada organização. De acordo com a *Figura 2.1*, os objetivos são definidos, sendo posteriormente

refinados em questões que, por sua vez, são respondidas através de métricas. O GQM é dividido em duas partes:

- *O processo de definição top-down, onde são estabelecidos os objetivos, as questões e as métricas;* e
- *O processo de interpretação, bottom-up, durante o qual são analisados os dados coletados e são definidas as ações a serem tomadas.*

### 2.5.1 O Modelo de Processos do GQM

Analisando-se mais detalhadamente, o modelo de processos do GQM é composto por outras fases, além dos processos de definição e de interpretação, conforme a *Figura 2.2*. Segundo Solingen (1997c), o método GQM é constituído de quatro etapas:

- Planejamento;*
- Definição;*
- Coleta de dados;* e
- Interpretação.*

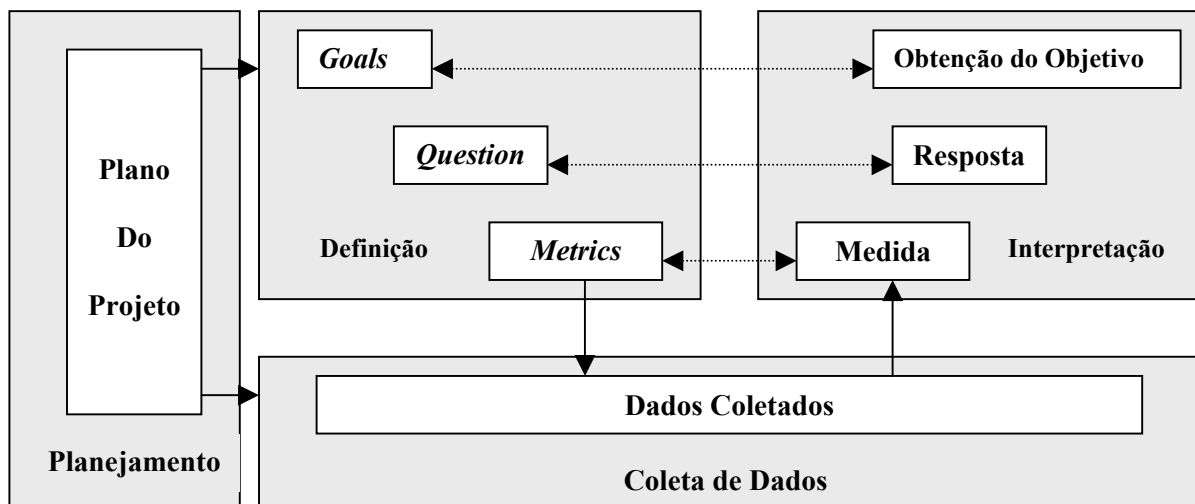


Figura 2.2: As quatro fases do GQM

A fase de planejamento é caracterizada por produzir um plano do projeto, no qual constam os motivos que justificam a escolha de um determinado sistema ou projeto, que foi selecionado para passar por um processo de medição. Aqui, todos os requisitos, que vão garantir o sucesso do GQM, devem ser preenchidos e planejados, como treinamento e envolvimento do nível gerencial. Além do projeto propriamente dito, a organização deve ser estudada, a fim de que os objetivos possam ser estabelecidos com mais segurança.

A partir do Plano do Projeto, resultado da primeira fase, os principais objetivos a serem melhorados são descritos. O processo é iniciado informalmente, sendo posteriormente refinado, indicando que objetivos serão definitivos. Esta fase de definição gera a árvore dos objetivos (*goals*), questões (*questions*) e métricas (*metrics*) da *Figura 2.1*, que está inserida no documento denominado Plano do GQM, onde também se encontra a descrição de cada elemento da árvore. Outro produto resultante da fase de definição é o Plano de Medição, onde se define o ambiente em que as métricas serão coletadas.

A fase de coleta de dados é marcada pela definição de formulários e de outros meios para captura dos dados, os quais devem ser armazenados em um formato que facilite a recuperação dos mesmos. Finalmente, inicia-se a interpretação desses dados. As métricas são usadas para responder as questões que, por sua vez, são utilizadas para atingir os objetivos. A interpretação é um processo progressivo de aprendizagem. Além disso, a interpretação é de fundamental importância para o sucesso do programa de medição, visto que dele saem as ações de melhoria tanto para o processo de desenvolvimento de software, como para o próprio programa de medição (Solingen, 1997a).

### **2.5.2 Definição dos Objetivos (*Goals*)**

Para definir corretamente os objetivos (*goals*) do GQM é necessário que antes sejam identificados os objetivos globais da organização e as áreas problemáticas ou que necessitem de melhorias. Vários métodos podem ser utilizados para estabelecer esses objetivos, sendo o *brainstorming* um dos mais adequados. Devem estar reunidos em uma sessão de *brainstorming* todos os funcionários que compõem o time do programa de medidas. Nesta etapa, devem ser

discutidos vários objetivos nos mais variados níveis da organização, percorrendo desde os objetivos estratégicos, até aqueles que dizem respeito a times específicos dentro da empresa (Solingen, 1995).

Algumas perguntas básicas podem ser feitas como: *Quais são os objetivos estratégicos de nossa empresa? Quais as forças externas que causam impacto em nossa empresa? Quais nossos maiores problemas?*

Ao final, os objetivos a serem medidos devem estar bem entendidos, sendo cada um deles apresentado de uma forma estruturada, seguindo o modelo (Basili, 1992):

<i>Analisar:</i>	[Var <sub>1</sub> ]
<i>Com o propósito de:</i>	[Var <sub>2</sub> ]
<i>Em respeito a:</i>	[Var <sub>3</sub> ]
<i>Sob o ponto de vista de:</i>	[Var <sub>4</sub> ]
<i>No seguinte contexto:</i>	[Var <sub>5</sub> ]

onde: Var<sub>1</sub> é uma fase ou todo o processo de desenvolvimento;

Var<sub>2</sub> é um verbo;

Var<sub>3</sub> é um objeto em estudo;

Var<sub>4</sub> é a comunidade de interesse; e

Var<sub>5</sub> é a empresa ou um setor dela.

Por exemplo, seja o *Objetivo 1*:

Analisar:	O processo de desenvolvimento de software
Com o propósito de:	Entender
Em respeito a:	Detecção de falhas
Sob o ponto de vista do:	Time de desenvolvimento de software
No seguinte contexto da:	Empresa XPTO

### 2.5.3 Definição das Questões

O passo seguinte é refinar os objetivos em questões (*questions*). Segundo Koomain (1996b), as questões devem ser definidas de modo a permitir uma interpretação de suas respostas em relação ao objetivo a que se referem.

Segundo Solingen (1995), as questões devem ser definidas em um nível de abstração mais baixo do que o objetivo correspondente, mas em um nível mais elevado dos que as métricas, que serão descritas na *Figura 2.3*. Finalmente, deve-se verificar se as questões resultarão em respostas das quais possam ser retiradas conclusões a respeito do objetivo. Um exemplo de questão para o *Objetivo 1*, supra citado é:

*Q<sub>1</sub>: Qual a distribuição das falhas detectadas nas diferentes fases de teste do processo de desenvolvimento?*

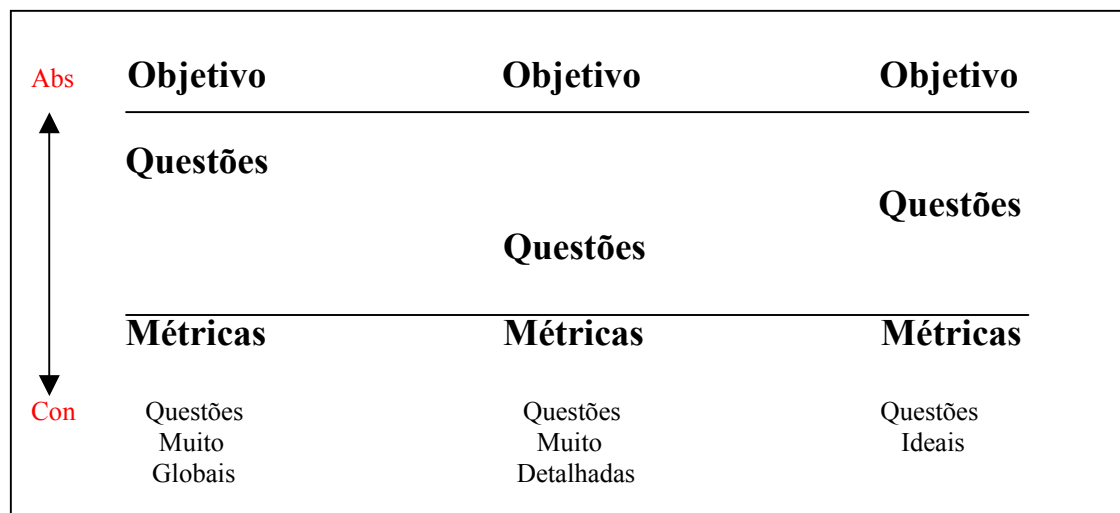


Figura 2.3: Definição das Questões

### 2.5.4 Definição das Métricas

Do mesmo modo que os objetivos são refinados em questões, as métricas (*metrics*) são os refinamentos das questões. Koomain (1996b) comenta que semelhantemente às questões, as métricas devem ser definidas de maneira que, no momento em que todos os dados tiverem sido coletados, haja informações suficientes para responder às questões. Além das métricas

propriamente ditas, os fatores que as influenciam também devem ser expostos na forma de métricas, pois os mesmos interferem indiretamente nas questões e nos objetivos.

Além das métricas, a equipe de desenvolvimento deve identificar as expectativas ou hipóteses em relação aos resultados das métricas. Tais suposições irão auxiliar o processo de interpretação dos dados coletados.

Seguem exemplos de métricas e hipótese para  $Q_1$ :

$M_1$ : *Quantidade de falhas detectadas na revisão do projeto, e*

$M_2$ : *Quantidade de falhas detectadas no teste de integração.*

$H_1$ : *O número de falhas detectadas em cada fase do processo de desenvolvimento é praticamente igual.*

O processo de refinamento dos objetivos, questões e métricas pode ser visualizado na *Figura 2.4*.

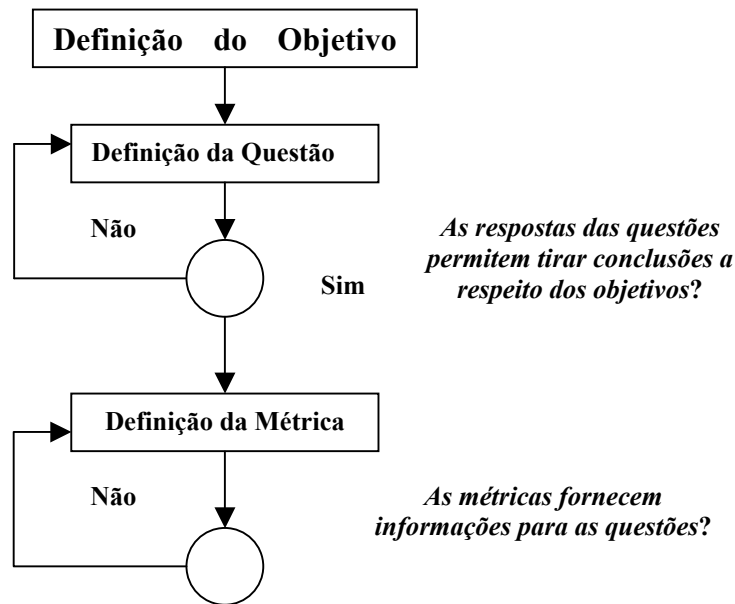


Figura 2.4: Fluxo do Processo de Definição do GQM

### 2.5.5 O Plano GQM e o Plano de Medição

De acordo com Koomain (1996b) e Solingen (1995), são elaborados dois documentos durante a aplicação do método GQM. O Plano GQM descreve todo o processo de refinamento

dos objetivos, questões e métricas, além de conter as hipóteses da equipe de desenvolvimento sobre os resultados das medidas. O Plano GQM é um importante meio de comunicação entre os membros da equipe de desenvolvimento, além de ser o ponto de partida para a definição dos meios de coleta de dados, assim como servirá de suporte para a interpretação dos mesmos.

O Plano de Medição descreve todos os procedimentos, manuais ou automatizados, de coleta de dados. Para cada métrica presente no Plano GQM, o Plano de Medição contém:

- *Uma definição formal da métrica;*
- *Possíveis valores da métrica;*
- *Ponto de associação (em qual momento a métrica é medida);*
- *O responsável pela medida; e*
- *O meio de coleta da métrica.*

Os procedimentos para a coleta dos dados podem ser manuais, através de formulários, ou automatizados, via *logs* de sistemas ou através de ferramentas tais como *Lotus Notes*, *Microsoft Project* e *Microsoft Excel*. A escolha da ferramenta deve seguir critérios de flexibilidade e variedade na forma de representação dos dados coletados, pois o processo de interpretação dos dados requer que as métricas sejam apresentadas nas mais variadas formas como, por exemplo, tabelas e gráficos.

#### **2.5.6 A interpretação dos Dados**

A partir dos dados coletados, executa-se o processo de interpretação dos dados, o qual analisa as medidas, visando responder as questões e alcançar os objetivos (Solingen, 1997a). A literatura indica que a interpretação dos dados coletados é um processo de aprendizagem, sendo de suma importância para o programa de medição.

Em Solingen (1997b), há uma indicação de que a interpretação dos dados implica conclusões e ações a serem tomadas. Os resultados desta fase devem ser divulgados em toda a

organização, para que todos possam tirar proveito dos ganhos obtidos. Acrescenta-se, ainda, que a interpretação dos dados não faz parte do método GQM.

A execução da interpretação dos dados deve ser feita através de sessões de *feedback* com a equipe de desenvolvimento, a qual é a responsável pelas principais conclusões, visto que é a detentora de conhecimentos detalhados dos processos ou produtos sob avaliação. Nas sessões de *feedback*, os resultados são apresentados via gráficos e tabelas, para que a interpretação seja facilitada. Em Koomain (1996a) e Koomain (1996b), podem ser encontradas algumas indicações para o processo de interpretação:

- *Usar planilhas com um nível de abstração que permita uma visão global das métricas coletadas;*
- *Verificar e manter dados históricos;*
- *Estimular o envolvimento da equipe de desenvolvimento, fazendo com que esta seja a principal responsável pela análise e interpretação do material em estudo;*
- *Nas sessões de feedback, o material apresentado deve estar simples e compacto;*
- *Os pontos de ação devem ser reavaliados nas sessões de feedback seguintes;*
- *As métricas mais essenciais devem estar detalhadas;*
- *Analisar as métricas que foram obtidas indiretamente antes de elaborar conclusões;*
- *Buscar as causas das tendências identificadas;*
- *Focar em processos, não em indivíduos;*
- *Apresentar gráficos e tabelas em conjunto;*
- *Usar consistentemente diferentes tipos de gráficos com suas respectivas escalas;*
- *Destacar os valores mais importantes nos gráficos e tabelas;*
- *Usar valores percentuais para enfatizar tendências;*
- *Considerar as hipóteses colhidas na fase de definição das métricas; e*
- *Agrupar métricas semelhantes em classes.*



A seguir estão apresentadas as principais métricas de estimativa, grupo que engloba as métricas que determinam valores referentes ao tamanho do software, foco deste trabalho, bem como o custo e esforço para desenvolvê-lo.

## 2.6 Métricas de Estimativa

Estimar é uma das primeiras atividades realizadas no planejamento do projeto e embora ainda seja mais arte do que ciência (Pressman, 2000), as técnicas de estimativa devem ser usadas enquanto não houver ciência mais exata (Demarco, 1989). A seguir, serão discutidas algumas das métricas de estimativa mais tradicionais como: *Linhas de código*, *Sistema métrico de Halstead*, *COCOMO*, *FPA*, e *Estimativas para novas tecnologias*.

### 2.6.1 Linhas de Código (LOC)

A técnica de linhas de código (*Line of code* – LOC), é uma das técnicas mais antigas para estimar tamanho de software (Arifoglu, 1993; Kemerer, 1992). Como o próprio nome sugere, essa métrica propõe que o tamanho de um sistema deve ser medido pela contagem das linhas de seu código fonte. Como a quantidade exata de linhas de código só é conhecida após a conclusão do sistema, a estimativa pode ser feita de duas formas (Simon, 2000):

- *Através de variáveis de estimativa usadas para cada elemento do sistema; ou*
- *Com o uso de dados de projetos anteriores, juntamente com variáveis de estimativa, projetando-se o custo e esforço de novos projetos.*

Percebe-se que LOC é uma métrica frágil e imprecisa, com várias desvantagens:

- *A definição de linha de código é obscura, sem padrões, não estando claro se deve incluir ou não declarações de dados, JCL, macro-instruções etc. (Simon, 2000);*
- *É uma medida técnica, sem significado para o usuário (Dumke, 1999);*
- *Não são consistentes, ou seja, algumas linhas são mais trabalhosas que outras (Demarco, 1989);*
- *Apresenta problemas de definição para linguagens não procedurais (Pressman, 2000);*

- *LOC na linguagem Assembly, por exemplo, não é comparável com LOC em uma outra linguagem de alto nível* (Fenton, 1999);
- *Não há dados para novas linguagens* (Bernstein, 2001);
- *As facilidades de herança e reutilização de código mascaram a contagem de linhas* (Tavares, 1999); e
- *Não contempla as primeiras fases do ciclo de desenvolvimento de sistemas* (Tavares, 1999).

Apesar de suas desvantagens, LOC não deve ser completamente descartada. As organizações podem usá-las para medir o que de fato foi produzido. Além disso, possuem alguns pontos relevantes:

- *Simplicidade e farta literatura disponível* (Pressman, 2000);
- *Servem como medida de normalização para qualidade de software, como, por exemplo, quantidade de defeitos por linhas de código* (Fenton, 1999);
- *Avaliar a produtividade* (Fenton, 1999); e
- *A contabilização das linhas código pode ser feita automaticamente.*

### **2.6.2 Sistema Métrico de Halstead**

Em 1972, Halstead iniciou estudos sobre algoritmos, tentando testar empiricamente a hipótese de que os operadores (comandos e palavras reservadas) e os operandos (itens de dados) contidos num programa deviam relacionar-se com a quantidade de erros no algoritmo. Foi definido um conjunto de primitivas, que podem ser obtidas a partir do código de um programa (Halstead, 1977):

- $n_1$ : *número de operadores distintos que aparecem no programa;*
- $n_2$ : *número de operandos distintos que aparecem no programa;*
- $N_1$ : *número de todas as instâncias de operadores;* e
- $N_2$ : *número de todas as instâncias de operandos.*

A partir destas primitivas, foram criadas expressões para tamanho total de um programa, volume potencial mínimo para um algoritmo, volume real de um módulo ou programa, nível do programa, nível da linguagem e outras derivações como esforço de desenvolvimento, tempo de desenvolvimento, e o número estimado de defeitos num programa.

O tamanho  $N$  de um programa pode ser estimado através da fórmula:

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

De forma semelhante, o volume de um programa pode assim ser definido:

$$V = N \log_2(n_1 + n_2)$$

A partir de então, o sistema métrico de Halstead foi objeto de várias pesquisas, apresentando-se como um bom indicador de esforço (Demarco, 1989; Pressman, 2000).

### 2.6.3 COCOMO

Em 1981, Boehm (Boehm, 1981) apresentou o COCOMO (*Constructive Cost Model*), que é uma hierarquia de modelos de estimativas de software. Essas estimativas compreendem esforço, prazo e tamanho da equipe para o desenvolvimento de produtos de software. O método pode ser aplicado nas várias fases do ciclo de desenvolvimento, fornecendo resultados mais satisfatórios em estágios mais avançados do projeto (Masse, 1997). Adicionalmente, o modelo de ciclo de vida adotado deve ser o cascata ou o de refinamento sucessivo (Babst, 1994).

A hierarquia dos modelos está dividida nas seguintes categorias:

- *Modelo 1 – COCOMO Básico*: calcula o esforço e o custo de desenvolvimento de software em função do tamanho do programa estimado em linhas de código.
- *Modelo 2 – COCOMO Intermediário*: calcula o esforço de desenvolvimento de software em função do tamanho do programa e de um conjunto de “direcionadores de custos”, que contêm avaliações subjetivas do produto, do hardware, da equipe e dos atributos do projeto.

- *Modelo 3 – COCOMO Avançado*: reúne todas as características da versão intermediária com uma avaliação do impacto dos direcionadores de custo em cada fase (análise, projeto, etc.) do processo de engenharia de software.

O modelo COCOMO está definido para três classes de projetos:

- *Modo orgânico*: projetos de software relativamente pequenos e simples, nos quais pequenas equipes com boa experiência em aplicações trabalham em um conjunto de requisitos não muito rígidos.
- *Modo semi-destacado*: projetos de software intermediários em tamanho e em complexidade, nos quais equipes com níveis heterogêneos de experiência devem alcançar uma combinação de requisitos variando de rígidos a não muito rígidos.
- *Modo embutido*: projetos que devem ser desenvolvidos dentro de um conjunto rígido de restrições operacionais de hardware e software.

As equações do COCOMO Básico são apresentadas a seguir:

$$E = a_b (\text{KLOC}) \exp(b_b)$$

$$D = c_b (E) \exp(d_b)$$

onde,  $E$  é o esforço medido em pessoas-mês,  $D$  é o tempo de desenvolvimento em meses, e KLOC é o número estimado de linhas de código do projeto (em milhares). Os valores dos coeficientes  $a_b$  e  $c_b$  e dos expoentes  $b_b$  e  $d_b$  estão na *Tabela 2.3*.

Tabela 2.3: COCOMO Básico

Projeto de software	$a_b$	$b_b$	$c_b$	$d_b$
Orgânico	2,4	1,05	2,5	0,38
Semidestacado	3,0	1,12	2,5	0,35
Embutido	3,6	1,2	2,5	0,32

O modelo básico foi estendido com o objetivo de considerar um conjunto de atributos direcionadores de custo, que podem ser agrupados em quatro categorias:

*1) Atributos do produto*

- Confiabilidade exigida do software;
- Tamanho do banco de dados da aplicação;
- Complexidade do produto.

*2) Atributos do hardware*

- a) Restrições de desempenho na execução;
- b) Restrições de memória;
- c) Volatilidade do ambiente de máquina virtual;
- d) Tempo de *turn around* (tempo para completar o ciclo) exigido.

*3) Atributos de pessoal*

- a) Capacidade de análise;
- b) Capacidade de engenharia de software;
- c) Experiência em aplicações;
- d) Experiência em máquina virtual;
- e) Experiência em linguagens de programação.

*4) Atributos de projeto*

- a) Uso de ferramentas de software;
- b) Aplicação de métodos de engenharia de software;
- c) Cronograma de atividades de desenvolvimento exigido.

Cada um dos atributos destas categorias é classificado de acordo com uma escala de seis pontos, que varia de “muito baixo” a “extremamente elevado”, em importância ou valor, como pode ser visto na *Tabela 2.4*.

O produto de todos os multiplicadores origina um Fator de Ajuste de Esforço (EAF). Os valores do EAF variam de 0,9 a 1,4. Se os valores para todas as categorias forem “Normal”, o EAF será 1,00 e os resultados serão idênticos ao COCOMO básico para um projeto semi-destacado.

Tabela 2.4: Classificação dos atributos direcionadores de custo

Direcionador de custo	Muito baixo	Baixo	Normal	Alto	Muito alto	Extremamente alto
Confiabilidade exigida do software	0,75	0,88	1,00	1,15	1,40	-
Tamanho do banco de dados	-	0,94	1,00	1,08	1,16	-
Complexidade do produto	0,70	0,85	1,00	1,15	1,30	1,65
Restrições de desempenho	-	-	1,00	1,11	1,30	1,66
Restrições de memória	-	-	1,00	1,06	1,21	1,56
Volatilidade da máquina virtual	-	0,87	1,00	1,15	1,30	-
Tempo de <i>turnaround</i>	-	0,87	1,00	1,07	1,15	-
Capacidade de análise	1,46	1,19	1,00	0,86	0,71	-
Capacidade e engenharia de software	1,42	1,17	1,00	0,86	0,70	-
Experiência em aplicações	1,29	1,13	1,00	0,91	0,82	-
Experiência em máquina virtual	1,21	1,10	1,00	0,90	-	-
Experiência em linguagens de programação	1,14	1,07	1,00	0,95	-	-
Uso de ferramentas	1,24	1,10	1,00	0,91	0,83	-
Aplicação de métodos de engenharia de software	1,24	1,10	1,00	0,91	0,82	-
Cronograma de desenvolvimento	1,23	1,08	1,00	1,04	1,10	-

A equação do COCOMO Intermediário é apresentada a seguir:

$$E = a_i (\text{LOC}) \exp(b_i) \times \text{EAF}$$

onde,  $E$  é o esforço em pessoas-mês e LOC, o número estimado de linhas de código para o projeto. Os valores dos coeficientes  $a_i$  e  $b_i$  estão na Tabela 2.5.

Tabela 2.5: COCOMO Intermediário

Projeto de software	$a_i$	$b_i$
Orgânico	3,2	1,05
Semidestacado	3,0	1,12
Embutido	2,8	1,20

O COCOMO avançado utiliza diferentes multiplicadores de esforço para cada fase do projeto. O método define seis fases no ciclo de vida: requisitos, projeto do produto, projeto detalhado, codificação (e teste de unidade), integração (e teste) e manutenção. Isto evidencia a importância de se ter vários níveis de previsibilidade em cada fase do ciclo de desenvolvimento.

Em 1995, surgiu a primeira publicação do COCOMO versão 2.0 (Boehm *et al.*, 1995), superando algumas limitações do COCOMO original, aliando-se aos benefícios da análise de pontos por função. O COCOMO 2.0 possui três modelos de custo:

- *Modelo de Composição da Aplicação*: busca resolver a questão da imprecisão quando se utilizam pacotes COTS (*Commercial off-the-shelf*). O modelo contabiliza as saídas, baseando-se na métrica *object point*, a qual é obtida a partir do número de telas geradas, relatórios produzidos ou módulos integrados.
- *Modelo Inicial do Projeto*: usa pontos por função não ajustados, ou LOC, como entrada. Foi especialmente construído para projetar custo e duração de um projeto nas fases iniciais do ciclo de vida, sendo melhor aplicado quando há somente informações parciais sobre o produto. Os direcionadores de custo foram reduzidos a sete.
- *Modelo Pós-Arquitetura*: um novo conjunto de equações, regras de contagem e direcionadores de custo foi desenvolvido, para medir com maior precisão o custo e a duração de um projeto, quando este apresentar sua arquitetura delineada. Esse modelo é melhor aplicado quando os riscos do projeto já tiverem sido solucionados. Possui dezessete direcionadores de custo.

Segundo Masse (1997), o COCOMO 2.0 acrescenta novas características ao COCOMO original, como:

- *Estimativas de projeto quando o modelo de ciclo de vida adotado for o espiral ou a prototipação*;
- *Endereçamento de questões relacionadas ao desenvolvimento baseado em componentes*; e
- *Adaptação às técnicas de orientação a objetos*.

#### **2.6.4 FPA (Análise de Pontos por Função)**

No ano de 1979, a FPA foi publicada como resultado das pesquisas de Allan Albecht (Albrecht, 1979). Baseando-se principalmente no projeto lógico, a métrica utiliza uma semântica própria para representar a funcionalidade, que deverá ser entregue ao usuário final. Essa métrica e suas principais extensões serão apresentadas no próximo capítulo.

#### **2.6.5 Estimativas para Novas Tecnologias**

Normalmente, a implantação de um site na Internet está intimamente relacionada com as operações estratégicas da organização. As consequências financeiras de um projeto sem sucesso podem por em risco a sustentação dessa organização. Ademais, o comércio eletrônico acirrou ainda mais a competitividade entre as empresas, fazendo com que o prazo para a finalização de um projeto tenha-se tornado fundamental para o sucesso do empreendimento.

O desenvolvimento de sistemas para a plataforma web assemelha-se em muitos aspectos aos sistemas cliente-servidor como, por exemplo, a elaboração de regras de negócio e a necessidade de interface com outros sistemas. No entanto, além das diferenças tecnológicas, a Internet provocou uma maior integração entre consultores de marketing, administradores e desenvolvedores de software. Esta diferença é de fundamental importância, pois com ela vieram novos ciclos de desenvolvimento de software, surgindo a necessidade de ajustar os modelos paramétricos para o cálculo de volume ou tamanho de software (LOC, FPA, ...). Esses modelos podem ser utilizados para a plataforma web, desde que se padronize o tratamento dos elementos próprios desta tecnologia, como páginas, objetos XML, etc.

Opcionalmente, pode-se fazer uso de modelos mais recentes, os quais mapeiam os objetos da plataforma web de uma forma mais direta. São eles (Roetzheim, 2000):

- *Internet Points*: projetados para estimar projetos, que serão executados na internet ou intranet. Podem ser considerados uma extensão da FPA, acrescentando, por exemplo, um melhor reconhecimento de tipos de páginas web.



- *Domino Points*: metodologia foi criada pelo *Cost Xpert Group*, sendo especializada para projetos a serem desenvolvidos com o *Lotus Domino*. A funcionalidade de um sistema é descrita em termos de *Forms*, *Navigators*, *Pages* e *Views*.
- *Use Case Points*: podem ser utilizados em projetos, que serão desenvolvidos usando o RUP (*Rational Unified Process*). Neste caso, os cenários dos casos de uso devem estar de acordo com a metodologia da *Rational*.
- *Class-Method Points*: trata-se de uma alternativa para sistemas desenvolvidos com as abordagens de projeto e programação orientados a objeto. No RUP, pode ser usado nas fases de elaboração e construção.

## 2.7 Conclusão

Este capítulo expôs uma visão geral sobre o uso de métricas no desenvolvimento de software, abordando-se entre outras questões, o plano de medição, o paradigma GQM, e métricas de estimativa.

As métricas de software desempenham um importante papel na engenharia de software. As medições avaliam situações, fornecem informações sobre a qualidade do processo e, no contexto desta dissertação, permitem que os gerentes estimem o tamanho e o custo de seus projetos de desenvolvimento de software.

No entanto, o processo de medição não pode ser aplicado de forma acidental. É necessário haver um plano de medição onde haja definições de aplicabilidade, métodos de coleta e cálculo, critérios de interpretação, dentre outros elementos. Deve-se, ainda, discutir as expectativas sobre a precisão das medições e definir formas para tratar grandes desvios. Diante destas considerações, medir sem planejar pode ser pior do que não medir.

No capítulo seguinte, será tratada especificamente a métrica de análise de pontos por função (FPA), objeto central deste trabalho.