Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

UMA INTRODUÇÃO ÀS MÉTRICAS DE SOFTWARE

DANIELA MARQUES
RENATA PONTIM DE MATTOS FORTES

Nº 92

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Ago./1999

Sumário

1. Introdução **5**

2. Total Quality Management (TQM) **5**

3. Qualidade de Software **5**

3.1 MODELO DE MCCALL 5
3.2 FURPS (MODELO DA HEWLETT-PACKARD) 5

4. Métricas de Software **5**

5. Modelos de Métricas **5**

5.1 MÉTRICAS PARA MODELO DE ANÁLISE 5
5.1.1 Métricas Baseadas em Função 5
5.1.2 Métricas para Especificação de Qualidade 5
5.2 MÉTRICAS PARA MODELO DE PROJETO 5
5.2.1 Métricas de Projeto de Alto Nível (caixa-preta) 5
5.2.2 Métricas de Projeto de Interface 5
5.3 MÉTRICAS PARA CODIFICAÇÃO 5
5.3.1 Contagem das Linhas de Código 5
5.3.2 Teoria de Halstead 5
5.4 MÉTRICAS PARA TESTE 5
5.4.1 Métricas Baseadas no Processo de Teste 5
5.4.2 Métricas Baseadas nas Características dos Testes 5
5.5 MÉTRICAS PARA MANUTENÇÃO 5
5.5.1 Padrão IEEE 982.1-1988 5

6. Conclusão **5**

Bibliografia **5**

Lista de Figuras

Figura 1. Elementos Chaves do TQM 5
Figura 2. Visão Profissional de Qualidade 5
Figura 3. Modelo de McCall 5
Figura 4. Métricas de Software 5
Figura 5. Ciclo de Vida Clássico 5
Figura 6. Métrica Morfológica 5
Figura 7. Diagrama de Transição. As ligações indicam a freqüência com que cada transição é feita [Sears, 93] 5
Figura 8. *Layout Satisfatório*, custo=3.92, *LA*=100. A linha contínua representa a seqüência mais freqüente de ações. A linha pontilhada a segunda seqüência mais freqüente de ações [Sears, 93] 5
Figura 9. *Layout Proposto, custo=4.23, LA*=93. A linha contínua representa a seqüência mais freqüente de ações. A linha pontilhada a segunda seqüência mais freqüente de ações [Sears, 93] 5

Lista de Quadros

Quadro 1. Fatores e Métricas de Qualidade [Pressman, 95] 5
Quadro 2. Questionário para PF 5
Quadro 3. Resumo das Métricas de Software X Fases do Ciclo de Vida 5

1. Introdução

Atualmente vivemos a era da qualidade tanto em serviços, como em produtos de modo geral. Há uma tendência no nosso mundo em se atingir a qualidade e aumentar a produtividade, onde este aumento na produtividade deve ser com qualidade.

A qualidade hoje é uma necessidade básica para a sobrevivência de qualquer produto ou serviço no mercado e não mais uma questão de luxo ou gosto. O software, como um produto, também participa dessa era. Ao adquirirmos um software exigimos que ele tenha qualidade, queremos algo confiável e que não tenha um valor exorbitante. A questão é: "*Como se obter qualidade nos produtos de software?*".

A Engenharia de Software, como disciplina dedicada a fornecer subsídios adequados ao processo de desenvolvimento de software com qualidade, utiliza as métricas para identificar melhor os atributos dos modelos que criamos e garantir qualidade nos produtos da engenharia ou nos sistemas que constrói.

As medidas de software, por sua vez, não são de conhecimento trivial (de fato, não se tratam de valores observáveis diretamente, e não indicam atributos de software capazes de distinguir um produto de outro sem qualquer subjetividade) como as medidas físicas (por exemplo, temperatura e velocidade). Apesar das medidas de software não serem absolutas, existem muitas pesquisas na área e em sua maioria, tenta-se transformar medidas indiretas em medidas diretas, ou seja, derivar a partir de medidas quantitativas, medidas qualitativas de modo a estimar o atributo especificado.

As métricas de software são temas para diversos debates, visto que alguns membros da comunidade da engenharia de software consideram as métricas imensuráveis devido ao fato delas não serem absolutas, e que deveríamos adiar as tentativas de medições até que elas fossem totalmente compreendidas [Pressman, 97].

Apesar de haver discordância, as métricas nos fornecem um caminho sistemático para garantir a qualidade baseada em um conjunto de regras claramente definida. Podemos, através das métricas, descobrir problemas que seriam catastróficos no futuro [Pressman, 97].

Este trabalho tem como objetivo apresentar de forma introdutória algumas das métricas de software encontradas na literatura e discuti-las no âmbito de suas contribuições para se obter indicativos de qualidade no produto de software. Antes das métricas serem discutidas, é dada uma breve visão do que é *"Total Quality Management"* (seção 2), "Qualidade de Software" (seção 3) e "Métricas de Software" (seção 4). Explicados esses conceitos básicos, são apresentadas algumas métricas interessantes, as que têm sido mais referenciadas na literatura revisada, e que têm algum atributo da qualidade de software como objetivo. A apresentação das métricas na seção 5 está organizada de forma a atender todos os passos do ciclo de vida clássico de desenvolvimento de software, ou seja, são apresentadas métricas para análise, projeto, codificação, testes e manutenção. Finalmente, na seção 6, são apresentadas as conclusões.

2. Total Quality Management (TQM)

O termo TQM *(Total Quality Management)*, em português Gerenciamento da Qualidade Total, é bastante difundido atualmente. Basicamente, trata-se de um método para criação de uma cultura na qual todos os membros da organização participam nos processos de melhoramento de produtos e de serviços. Os elementos chaves, segundo [Kan, 94] do TQM podem ser resumidos em:

- **Focos do Cliente**: o objetivo é concluir com êxito e obter a satisfação total do cliente.
- **Processos**: o objetivo é definir os processos necessários para atingir o processo de melhoramento contínuo.
- **Lado Humano da Qualidade**: o objetivo é criar uma cultura de qualidade em toda a organização, incluindo fatores humanos, sociais e psicológicos.
- **Medidas e Análises**: o objetivo é conduzir o melhoramento contínuo em todos os parâmetros da qualidade em um sistema orientado a metas.

A Figura 1 mostra esquematicamente a representação do TQM. Analisando a Figura 1, tona-se óbvio que a camada básica dos elementos chaves é a das métricas, medidas, modelos e análises, e que, portanto, desempenham o papel de elementos fundamentais para o melhoramento contínuo.

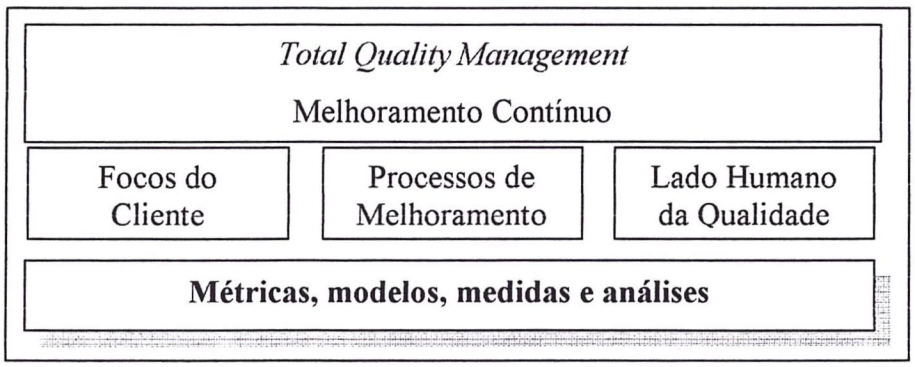

Figura 1. Elementos Chaves do TQM

É claro que para se ter uma ótima qualidade, todos os elementos do TQM devem ser satisfeitos. No caso deste relatório técnico, o enfoque está relacionado particularmente com métricas e medidas que visem a obtenção de qualidade de software. De fato, TQM é uma abordagem mais ampla, que tem sido adotada por organizações empresarias no âmbito de todos os setores que congrega. Ainda assim, foi aqui apresentada para enfatizar a importância das métricas e medidas como um suporte essencial na busca da "qualidade total".

3. Qualidade de Software

Era da Qualidade, esta é a era em que vivemos conforme histórico da Engenharia de Software. A Engenharia de Software segue a seguinte perspectiva [Pressman, 97]:

- Anos 60 – Era Funcional.
- Anos 70 – Era do Método.
- Anos 80 – Era do Custo.
- Anos 90 e depois – Era da Qualidade.

Atualmente, a qualidade não é mais uma questão de luxo, mas sim uma necessidade básica para a luta no mercado. Dentre as diversas visões de qualidade, pode-se definir a visão popular e a visão profissional do que seja qualidade.

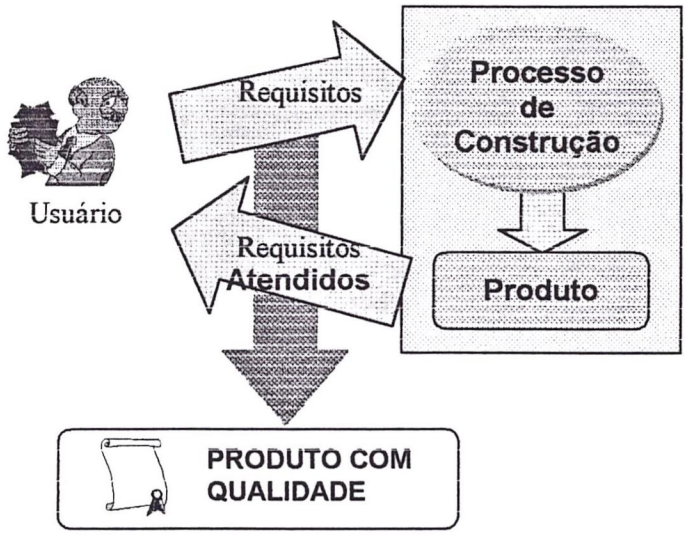

Figura 2. Visão Profissional de Qualidade

A visão popular considera que a qualidade pode ser sentida, discutida, julgada, mas não pode ser medida, controlada, nem gerenciada; qualidade é vista como luxo, classe e questão de gosto; produtos caros, sofisticados e mais complexos são considerados de maior qualidade que produtos similares mais simples. Em contrapartida, na visão profissional, qualidade está relacionada com os requisitos designados para o produto. A não conformidade aos requisitos é considerada defeito ou falta de qualidade (Figura 2). A qualidade, do ponto de vista profissional, pode e deve ser medida, definida, monitorada, gerenciada e melhorada.

A qualidade de software pode ser definida como um conjunto de atributos de software que devem ser satisfeitos de modo que o software atenda às necessidades dos usuários. A determinação dos atributos relevantes para cada software varia em função do domínio da aplicação, das tecnologias utilizadas, das características específicas do projeto e das necessidades do usuário e da organização. [Pressman, 97] define qualidade de software

como: *"conformidade aos requisitos de desempenho e de funcionalidade definidos que foram atendidos explicitamente, aos padrões de desenvolvimento explicitamente documentados e às características que são esperadas por todo software desenvolvido por profissionais"*.

A qualidade esperada por cada pessoa que utiliza o sistema depende da sua posição no processo. Os *usuários* avaliam o software sem conhecer seus aspectos internos, estão apenas interessados na facilidade de uso, no desempenho, na confiabilidade dos resultados e no preço. Os *desenvolvedores* avaliam os aspectos de conformidade em relação aos requisitos dos clientes e também aspectos internos do software (taxa de defeito, facilidade de manutenção, conformidade em relação aos requisitos funcionais de usuários). As *organizações* avaliam aspectos de desempenho dos desenvolvedores e também aspectos de custo e cronograma (cumprimento dos prazos, boa previsão de custo, boa produtividade).
Dessa forma, o primeiro passo para se ter um software de qualidade é determinar os seus requisitos, que sejam necessários ser satisfeitos, lembrando que dependendo do domínio, cada software terá suas próprias características de qualidade. Além disso, somente determinar as características necessárias não é suficiente para se ter um software de qualidade, além desse primeiro passo, deve-se ter um procedimento para acompanhar passo a passo a evolução da qualidade [Rocha, 98].

Nas subseções seguintes são discutidos três modelos de acompanhamento de qualidade de software, que têm sido mais amplamente discutidos na literatura. Tais modelos são denominados modelos de processo de desenvolvimento de software.

3.1 Modelo de McCall

O modelo de McCall é um dos mais citados na literatura, foi proposto em 1977, a fim de estabelecer o que realmente deve ser avaliado para se obter qualidade. [McCall et al., 77] propuseram um conjunto de fatores (Figura 3) que avalia o software sob três perspectivas do produto:

1. **Operação:** suas características operacionais, como por exemplo, se o software é aprendido facilmente, se é operado eficientemente e se os seus resultados são realmente aquilo que o usuário espera.
2. **Revisão:** facilidade de mudanças, relacionadas à correção de erros e à adaptação do sistema.
3. **Transição:** sua adaptabilidade a novos ambientes. Pode não ser importante para todas as aplicações, mas devido ao grande avanço tecnológico em hardware e de mudanças decorrentes deste avanço, esse é um fator que vem crescendo cada vez mais e que deve ser enfatizado.

A seguir, são descritos os fatores propostos por McCall, correspondentes às perspectivas:

1) Com relação à Operação do Produto:

- *Corretitude (- Ele faz aquilo que eu quero?):* garantia que o programa satisfaça suas especificações e cumpra os objetivos visados pelo usuário.
- *Confiabilidade (- Ele se comporta com precisão o tempo todo?):* habilidade do programa não ter falhas. Executar suas funções com precisão.

- *Eficiência (- Ele rodará em meu hardware tão bem quanto possível?):* refere-se à utilização de recursos. Quantos recursos de computação e de código são exigidos para que um programa execute sua função.
- *Integridade (- Ele é seguro?):* controle das pessoas não autorizadas ao software ou aos seus dados.
- *Usabilidade (- Ele foi projetado para o usuário?):* facilidade de utilização do software. Aprendizagem, operação, preparo da entrada e interpretação da saída de um programa.

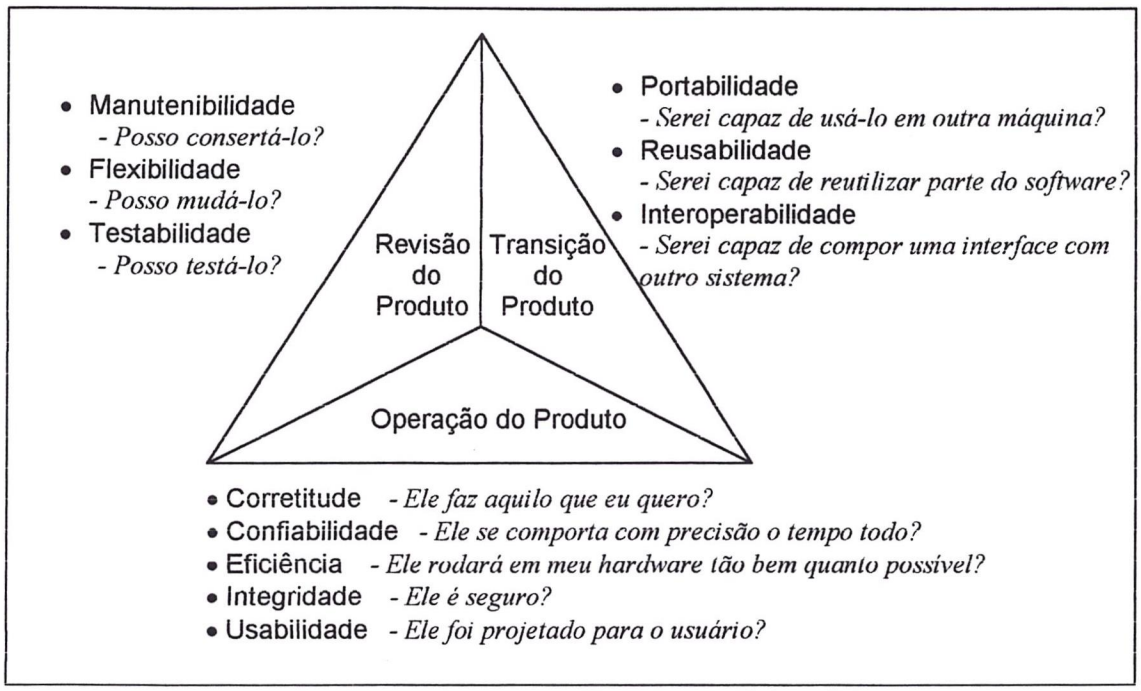

Figura 3. Modelo de McCall

2) Com relação à Revisão do Produto:

- *Manutenibilidade (- Posso consertá-lo?):* esforço exigido para localizar e reparar erros em um programa.
- *Flexibilidade (- Posso mudá-lo?):* facilidade de se realizar modificações necessárias no ambiente operacional.
- *Testabilidade (- Posso testá-lo?):* esforço necessário para testar um programa a fim de garantir que ele não contenha erros e execute a função pretendida.

3) Com relação à Transição do Produto:

- *Portabilidade (- Serei capaz de usá-lo em outra máquina?):* esforço necessário para transferir um programa de um ambiente (hardware e/ou software) para outro.
- *Reusabilidade (- Serei capaz de reutilizar parte do software?):* quanto um programa, ou partes dele, pode ser reutilizável em outra aplicação.
- *Interoperabilidade (- Serei capaz de compor uma interface com outro sistema?):* esforço necessário para se acoplar um programa a outro.

É difícil, e em certos casos impossível, desenvolver medidas diretas dos fatores de qualidade acima. Portanto, um conjunto de métricas é definido e usado para desenvolver expressões para cada um dos fatores, de acordo com a seguinte fórmula:

$$F_q = c_1 \times m_1 + c_2 \times m_2 + \ldots + c_n \times m_n$$

onde: F_q é um fator de qualidade de software,

c_n são coeficientes de regressão

m_n são métricas que afetam o fator de qualidade.

Infelizmente, muitas das métricas sugeridas por McCall são subjetivas e estão na forma de um *checklist* (lista de conferência) usado para graduar atributos específicos do software e o esquema de graduação é uma escala de 0 (baixo) a 10 (alto).
As métricas que afetam as características citadas acima e que são usadas no esquema de graduação são descritas a seguir:

1. *Auditabilidade:* facilidade com que se pode obter conformidade com padrões.
2. *Precisão:* precisão dos cálculos e de controle.
3. *Comunicação Genérica (Communication Commonality):* quanto que interfaces padrões, protocolos e largura de bandas são utilizados.
4. *Completitude:* quanto da implementação total da função requerida foi conseguida.
5. *Concisão:* compactação do programa em termos de linhas de código.
6. *Consistência:* uso de técnicas de projeto e documentação uniformes ao longo do projeto de desenvolvimento de software.
7. *Dados Genéricos (Data Commonality):* uso de estruturas e tipos de dados padrões ao longo do programa.
8. *Tolerância a Erros:* dano que ocorre quando um programa encontra um erro.
9. *Eficiência de Execução:* desempenho em tempo de execução de um programa.
10. *Expansibilidade:* quanto os projetos de arquitetura, procedimentais e de dados podem ser ampliados.
11. *Generalidade:* quão genérica é a aplicação dos componentes do programa.
12. *Independência de Hardware:* quanto o software é desvinculado do hardware em que opera.
13. *Instrumentação:* quanto o programa monitora sua própria operação e identifica erros que venham a ocorrer.
14. *Modularidade:* independência funcional dos componentes do programa.
15. *Operabilidade:* facilidade de operação de um programa.
16. *Segurança:* disponibilidade de mecanismos que controlem ou protejam programas e dados.
17. *Autodocumentação:* quanto o código-fonte apresenta documentação significativa.
18. *Simplicidade:* quanto um programa pode ser entendido sem dificuldade.
19. *Independência do Software Básico:* quanto um programa é independente de particularidades não-padronizadas de linguagens de programação *nonstandard* e das características de sistemas operacionais.
20. *Rastreabilidade:* capacidade de rastrear uma representação de projeto ou componente de programa até os requisitos.

21. Treinamento: quanto o software auxilia no sentido de ajudar novos usuários a utilizarem o programa.

A Quadro 1 mostra a relação entre os fatores de qualidade e as métricas listadas acima. Vale salientar que o peso dado a cada métrica depende dos produtos e das preocupações locais.

Quadro 1. Fatores e Métricas de Qualidade [Pressman, 95]

Fator de Qualidade / Métrica de Qualidade	Corretitude	Confiabilidade	Eficiência	Integridade	Manutenibilidade	Flexibilidade	Testabilidade	Portabilidade	Reusabilidade	Interoperabilidade	Usabilidade
1 Auditabilidade				x			x				
2 Precisão	x										
3 Comunicação Genérica										x	
4 Completitude		x				x	x				
5 Concisão			x		x	x					
6 Consistência	x	x			x	x					
7 Dados Genéricos										x	
8 Tolerância a Erros		x									
9 Eficiência de execução			x								
10 Expansibilidade						x					
11 Generalidade						x		x	x	x	
12 Independência de Hardware								x	x		
13 Instrumentação				x	x		x				
14 Modularidade		x			x	x	x	x	x	x	
15 Operabilidade			x								x
16 Segurança				x							
17 Autodocumentação					x	x	x	x	x		
18 Simplicidade		x			x	x	x				
19 Independência do Software Básico								x	x		
20 Rastreabilidade	x										
21 Treinamento											x

3.2 FURPS (Modelo da Hewlett-Packard)

A Hewlett-Packard, baseando-se no modelo de McCall, desenvolveu mais cinco fatores de qualidade e os considerou como um modelo de qualidade que foi denominado pela sigla FURPS. Esta sigla foi derivada das iniciais das palavras inglesas para funcionalidade *(Functionality)*, usabilidade *(Usability)*, confiabilidade *(Reliability)*, desempenho *(Performance)* e suportabilidade *(Supportability)* [Grady & Caswell, 87].
Os seguintes atributos são definidos para cada um dos cinco principais fatores:

- *Funcionalidade:* conjunto de características e capacidades do programa, generalidades das funções, segurança do sistema global.
- *Usabilidade:* fatores humanos, estética global, consistência e documentação.
- *Confiabilidade:* freqüência e gravidade das falhas, precisão dos resultados de saída, tempo médio entre falhas, previsibilidade do programa.
- *Desempenho:* velocidade de processamento, tempo de resposta, consumo de recursos e eficiência.
- *Suportabilidade:* capacidade de ampliar o programa, adaptabilidade, capacidade de serviço, capacidade de teste, capacidade de organizar e controlar elementos da configuração de software, facilidade com que um sistema pode ser instalado, facilidade com que problemas podem ser detectados.

Os fatores e atributos do modelo de qualidade FURPS descritos acima podem ser usados para estabelecer métricas de qualidade para cada passo do processo de engenharia de software. A matriz sugerida para orientar a escolha de medições pode ser encontrada em [Pressman, 95].

4. Métricas de Software

As métricas de software são importantes para três atividades básicas segundo [Fenton & Pfleeger, 97]:

- Entender o desenvolvimento e o processo de manutenção.
- Controlar os projetos de software.
- Melhorar processos e produtos.

Com essas três atividades em mente, as métricas podem então ser usadas também para: indicar a qualidade do produto e do processo e, melhorá-los; auxiliar a avaliação da produtividade dos que desenvolvem o produto; mostrar benefícios derivados de novos métodos e ferramentas de engenharia de software; formar uma base para as estimativas; ajudar na justificativa de aquisição de novas ferramentas ou de treinamentos adicionais; auxiliar na avaliação do progresso do projeto durante o desenvolvimento.

As medidas de software podem ser classificadas em: medidas diretas e medidas indiretas (Figura 4). As medidas diretas são aquelas obtidas diretamente de quantidades observadas, por exemplo: custo, esforço, linhas de código, velocidade de execução, memória, número

de erros. As medidas indiretas são aquelas que não podem ser obtidas diretamente dos itens relacionados ao software. Elas dizem respeito, por exemplo, a funcionalidade, qualidade, complexidade, eficiência, confiabilidade e manutenibilidade.

As métricas de software não são tão simples de serem obtidas porque existem diversos pontos de vista sobre qualidade de um mesmo produto. Suponhamos a compra de um carro, verificamos o seu "atrativo". Cada pessoa que verificar o carro pode determinar "atrativo" como sendo: o *design* do carro, outras podem considerar "atrativo" as características mecânicas, outras se o carro terá boa aceitação de venda quanto estiver velho, outras se ele é econômico, e assim sendo, é difícil determinar uma dessas características como singular com outras, portanto é difícil derivar um simples valor para "atrativo". Este é o problema encontrado nas métricas. [Fenton, 94] cita que: *"O perigo de se tentar encontrar medidas que caracterizam muitos atributos diferentes é que, inevitavelmente, a medida tem que satisfazer conflitos de objetivos. Isto é contrária a teoria representacional de medições".*

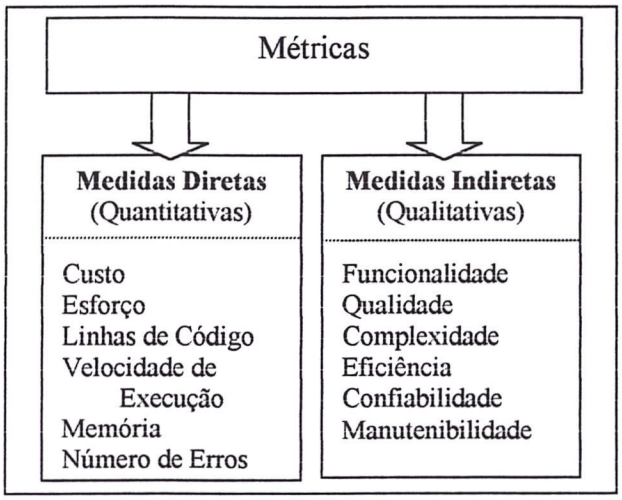

Figura 4. Métricas de Software

Apesar do problema relacionado ao caráter de não ser conseguir isolar completamente os atributos por meio das medições existentes, é necessário medir e controlar. Às vezes as métricas que foram geradas podem não ser úteis ao engenheiro de software de imediato, e as métricas relacionadas à qualidade muitas vezes não medem a qualidade em si, mas sim indicar alguma manifestação de qualidade, e qualquer indicativo de qualidade é melhor do que nada.

[Roche, 94] sugere um processo de medição que pode ser caracterizado por cinco atividades:

1. *Formulação*: formular medidas de software e métricas que sejam apropriadas para a representação do software existente.
2. *Coleta*: mecanismo usado para acumular dados requeridos das métricas formuladas.
3. *Análise*: computação das métricas e a aplicação de ferramentas matemáticas.
4. *Interpretação*: avaliação dos resultados das métricas num esforço de ganhar qualidade.

5. *Feedback*: recomendações derivadas das interpretações de métricas técnicas transmitidas pela equipe de software.

Apesar do processo de medição ser proposto, muitas métricas não são úteis de imediato. Mas como saber se as métricas são efetivas? [Ejiogu, 91] define um conjunto de atributos que podem ser observados por métricas de software efetivas. As métricas e as medidas que sejam efetivas devem ser:

- *Simples e computável*: deve ser relativamente fácil para aprender como derivar a métrica, e sua computação não devem demandar grandes esforços e tempo.
- *Empírica e de persuasão intuitiva*: a métrica deve satisfazer a noção intuitiva do engenheiro sobre o atributo do produto analisado.
- *Consistente e objetiva*: a métrica deve sempre produzir resultados não ambíguos, ou seja, utilizando as mesmas informações em partes independentes deve-se chegar ao mesmo resultado.
- *Consistente em seu uso em unidades e dimensões*: a matemática computacional da métrica deve usar medidas que não induzam a combinações sem sentido de unidades. Como exemplo: várias pessoas em um mesmo projeto dão nomes diferentes a variáveis do programa que resultam em uma mistura de dúvidas de unidades que não é intuitiva e perceptiva.
- *Independente de linguagem de programação*: as métricas devem ser baseadas no modelo de análise, projeto, ou, na própria estrutura do programa.
- *Mecanismo efetivo para feedback de qualidade*: as métricas devem fornecer informações para que possam levar à qualidade final do produto.

As métricas, para serem efetivas, devem obedecer aos critérios acima, mas isso não significa que as que não satisfizerem um ou mais critérios devam ser descartadas. A métrica é um elemento básico para se alcançar/medir a qualidade. Na próxima seção são descritas algumas métricas, segundo adequação a aplicabilidade das mesmas de acordo com as diferentes etapas do processo genérico de software.

5. Modelos de Métricas

Após terem sido introduzidos os principais conceitos relacionados à Qualidade de Software e Métricas de Software, foram observados que critérios diferentes devem ser considerados para que a utilização de métricas de software possam efetivamente considerar a qualidade do produto. Uma das considerações importantes se refere ao processo de desenvolvimento do produto, que traz consigo a elaboração de produtos intermediários que podem então ser mensurados. Sendo assim, nesta seção são apresentadas algumas métricas interessantes, as que têm sido mais referenciadas na literatura revisada, e que têm algum atributo da qualidade de software como objetivo. As próximas subseções apresentam as métricas referentes a cada passo do ciclo de vida clássico de desenvolvimento de software (Figura 5), ou seja, são apresentadas métricas para análise, projeto, codificação, testes e manutenção.

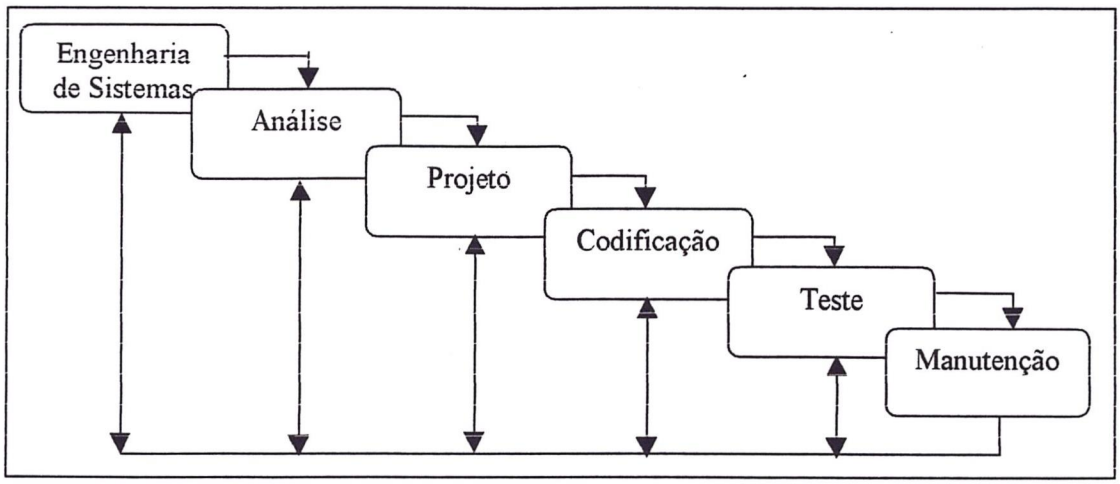

Figura 5. Ciclo de Vida Clássico

5.1 Métricas para Modelo de Análise

Nesta fase do ciclo de vida faz-se a coleta dos requisitos e se tem a base para o projeto, compreendendo o domínio da informação, a função, desempenho e interfaces exigidos. Segundo [Pressman, 97], não existem muitas métricas na literatura para esta fase, mas pode-se adaptar métricas do projeto de aplicação para este contexto. As métricas nesta fase são definidas com o intuito de predizer o tamanho do sistema resultante ou assegurar qualidade ao modelo de análise.

Nas próximas seções são descritas as Métricas Baseadas em Função (que fornecem o tamanho do sistema com base no modelo de análise) e as Métricas para Especificação de Qualidade (asseguram qualidade no modelo de análise).

5.1.1 Métricas Baseadas em Função

A métrica pontos por função (PF) pode ser usada para predizer o tamanho do sistema que será originado do modelo de análise. Esta métrica foi proposta pela primeira vez por [Albrecht, 79].

Os PFs são derivados usando uma relação empírica baseada em medidas do domínio de informação e da complexidade do software. Esta métrica pode ser aplicada através de três passos:

1) Computar a métrica PF;
2) Responder a um questionário, considerando uma escala de 0 a 5;
3) Ajustar os PF de acordo com a complexidade do sistema.

Para computar a métrica PF deve-se completar a seguinte tabela:

Parâmetro	Contagem		Fator de Ponderação Simples	Médio	Complexo	
Nro de Entradas do Usuário		x	3	4	6	
Nro de Saídas do Usuário		x	4	5	7	
Nro de Consultas do Usuário		x	3	4	6	
Nro de Arquivos		x	7	10	15	
Nro de Interfaces Externas		x	5	7	10	
Contagem Total	→					

Onde os parâmetros significam:

- *Número de Entradas do Usuário*: entradas de usuário que forneçam diferentes dados orientados a aplicação.
- *Número de Saídas do Usuário*: saídas de usuário que forneçam informações orientadas a aplicação (relatórios, telas, mensagens de erro).
- *Número de Consultas de Usuário*: entradas *on-line* que resultam em saídas *on-line.*
- *Número de Arquivos*: cada arquivo lógico.
- *Número de Interfaces Externas*: todas as interfaces legíveis por máquina, usadas para transmitir informação para outro sistema.

Depois de preenchida a tabela acima, deve-se responder um questionário (14 questões) numa escala de influência de 0 a 5. Onde 0 - significa nenhuma influência, 1 - pouca influência, 2 - influência moderada, 3 – influência média, 4 – influência significativa e 5 – influência essencial. O questionário é apresentado no Quadro 2.

Quadro 2. Questionário para PF

1. O sistema exige backup e recuperação confiáveis?
2. É requerida comunicação de dados?
3. Existem funções de processamento distribuído?
4. O desempenho é crítico?
5. O sistema funcionará num ambiente operacional existente, intensivamente utilizado?
6. O sistema requer entrada de dados *on-line*?
7. A entrada de dados *on-line* exige que a transição de entrada seja elaborada em múltiplas telas ou operações?
8. Os arquivos mestres são atualizados *on-line*?
9. A entrada, saída, arquivos ou consultas são complexas?
10. O processamento interno é complexo?
11. O código foi projetado de forma a ser reusável?
12. A conversão e a instalação estão incluídas no projeto?
13. O sistema é projetado para múltiplas instalações em diferentes organizações?
14. A aplicação é projetada de forma a facilitar mudanças e o uso pelo usuário?

Depois de respondido o questionário, basta ajustar os PF de acordo com a complexidade do sistema, através da seguinte fórmula:

$$PF = Contagem\ Total * (0{,}65 + 0{,}01 * \sum_{i=1}^{14}(Fi))$$

F_i = valores de ajuste da complexidade das perguntas 1-14

Este valor de PF corresponde ao tamanho total do sistema (todas as unidades) originado pelo modelo de análise.

5.1.2 Métricas para Especificação de Qualidade

Conforme já mencionado, esta métrica assegura a qualidade no modelo de análise. Além de garantir qualidade às especificações dos requisitos correspondentes através de uma lista de características. Esta métrica foi proposta por [Davis et al., 93] e a lista de características do software que são medidas são: especificação (falta de ambigüidade), completitude, corretitude, entendimento, rastreabilidade, modificabilidade, precisão e reusabilidade. Para completar, [Davis et al., 93] sugere que para se ter uma especificação de alta qualidade é melhor que a mesma seja armazenada e executada eletronicamente, caso contrário, que sejam bem documentadas as relações importantes, em nível correto de detalhes.

Esta métrica, apesar de sugerir muitas características qualitativas, pode ser interpretada utilizando uma ou mais métricas. Exemplificando, se queremos saber o número de requisitos numa especificação - n_r, somamos os números de requisitos funcionais - n_f e não funcionais (por exemplo, desempenho) - n_{nf}, ou seja, $n_r = n_f + n_{nf}$. Para determinar a especificação dos requisitos (falta de ambigüidade), é proposta uma métrica com base na consistência de interpretações revisadas de cada requisito, $Q_1 = n_{ui} / n_r$, onde dividimos o número de requisitos que todos os revisores tem interpretação idêntica (n_{ui}) pelo número de requisitos de uma especificação. Quando o valor de Q_1 é igual a 1, significa que existe a menor ambigüidade de especificação. Assim sendo, é possível através de uma ou mais métricas interpretar todas as características citadas acima.

5.2 Métricas para Modelo de Projeto

Na fase do projeto, é proposto que sejam medidos os quatro atributos do programa em desenvolvimento: estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização da interface. É inaceitável que um projeto seja conduzido sem que sejam definidos quais suas características que serão medidas, quais as métricas que serão utilizadas para que o projeto possa evoluir. Temos que nos preocupar com isso nesta fase, mesmo que as métricas existentes não sejam perfeitas, pois a utilização delas é melhor do que deixar o projeto sem controle algum.

Nas próximas seções, uma métrica que se enquadra na arquitetura de software e outra que se encaixa na caracterização da interface são descritas.

5.2.1 Métricas de Projeto de Alto Nível (caixa-preta)

Esta métrica focaliza um dos atributos no qual a fase de projeto se concentra, particularmente relacionada à fase de arquitetura do software. O foco desta métrica é sobre a estrutura arquitetural e de chamada dos módulos.

A métrica definida por [Henry & Kafura, 81] foi utilizada para avaliação e implementação do núcleo do sistema operacional UNIX, que exibe uma interconectividade de módulos típica de sistemas de grande porte. A métrica baseia-se no número de caminhos possíveis de fluxo de informação através de um módulo, ou seja, *fan-in* e *fan-out*. A complexidade definida por [Henry & Kafura, 81] é:

$$complexidade = tamanho(i) * (f_{in}(i) * f_{out}(i))^2$$

onde:

tamanho(i) é o número de LOC estimado no módulo i,

$f_{in}(i)$ é o número de fluxos locais (são criados por passagem de parâmetros) terminando no módulo i

$f_{out}(i)$ é o número de fluxos locais originando-se no módulo i.

O termo *(fan-in * fan-out)* representa o número possível de todas as combinações de módulos de um sistema. Quando grandes quantidades de módulos são envolvidas num sistema, a dificuldade em entendê-lo, testá-lo e modificá-lo é maior, e por esse motivo o termo *(fan-in * fan-out)* é elevado ao quadrado.

[Fenton, 91] sugere várias métricas simples relacionadas à forma de projeto, que possibilitam arquiteturas de programas diferentes serem comparadas usando um conjunto de dimensões diretas. Conforme Figura 6, [Fenton, 91] define a métrica:

$$size = n + a \quad (= 17+18 = 35)$$

onde: n é o número de nós e a é o número de arcos

depth (profundidade) = o caminho mais longo da raiz até o nó folha (= 4)

width (largura) = número máximo de nós de qualquer um dos níveis da arquitetura (= 6)

$$r = a/n \ (= 18/17 = 1{,}06)$$

onde r fornece uma simples indicação da ligação da arquitetura.

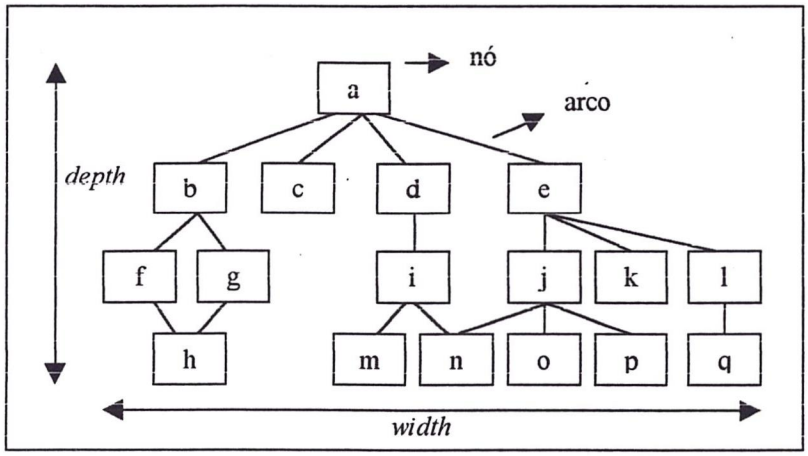

Figura 6. Métrica Morfológica

A Força Aérea dos EUA desenvolveu diversos indicadores de qualidade de software baseados nos conceitos propostos pela IEEE Std. 982.1-1988 e utiliza informações obtidas dos dados e projeto arquitetural para derivar um Índice de Qualidade de Estrutura do Projeto (*Design Structure Quality Index* - DSQI), que varia de 0 a 1. Para se computar o DSQI é necessária a obtenção dos seguintes valores [Charette, 89]:

- S_1: número total de módulos na arquitetura de programa.
- S_2: número de módulos cuja função dependa da fonte de entrada de dados ou que produza dados a serem usados em outro lugar.
- S_3: o número de módulos cuja função correta dependa de processamento anterior.
- S_4: o número de itens de banco de dados.
- S_5: o número total de bancos de dados únicos.
- S_6: o número de segmentos do banco de dados (registros diferentes ou objetos individuais).
- S_7: o número de módulos com uma única entrada e uma única saída.

Após determinar os valores de S_1 a S_7, deve-se computar os valores intermediários, que são:

- *Estrutura de Programa*: D_1, onde D_1 é definido da seguinte maneira: se o projeto arquitetural foi desenvolvido usando-se um método específico (por exemplo, projeto orientado a fluxo de dados ou projeto orientados a objeto), então $D_1=1$; caso contrário, $D_1=0$.
- *Interdependência modular:* $D_2 = 1 - (S_2 / S_1)$
- *Módulos não dependentes de processamento anterior:* $D_3 = 1 - (S_3 / S_1)$
- *Tamanho do BD*: $D_4 = 1 - (S_5 / S_4)$
- *Compartimentalização do BD*: $D_5 = 1 - (S_6 / S_4)$
- *Característica de Entrada/Saída modular*: $D_6 = 1 - (S_7 / S_1)$

Com todos os valores determinados, o DSQI é computado da seguinte maneira:

$$DSQI = \sum w_i D_i$$

onde i = 1 a 6,

w_i = peso relativo da importância de cada um dos valores intermediários e

$\sum w_i = 1$ (se todos os D_i tiverem um peso igual, então $w_i = 0{,}167$)

O valor do DSQI para os últimos projetos pode ser determinado e comparado com um valor de projeto que esteja permanentemente em desenvolvimento. Se o DSQI for insignificantemente inferior a média dos últimos projetos, são indicados uma revisão e um trabalho de projetos adicionais. Similarmente, se grandes mudanças tiverem de ser feitas em um projeto existente, o efeito dessas mudanças sobre o DSQI pode ser calculado.

5.2.2 Métricas de Projeto de Interface

Existe uma vasta literatura sobre a Interface Usuário-Computador (*Human-Computer Interaction* - HCI), mas poucas são as publicações sobre métricas que avaliem sua qualidade ou usabilidade.

[Sears, 93] sugere um *Layout Apropriado(LA)* como uma métrica. A idéia do *Layout Apropriado* é que para todo *layout* pode ser atribuído um custo que corresponde a sua medida de tempo ou preferência do usuário. Para computar o *LA*, o projetista deve providenciar um conjunto de objetos (ícones gráficos, textos, menus) usados na interface, a seqüência de ações desenvolvidas pelos usuários, e com que freqüência cada ação é usada. O custo pode ser determinado conforme cada seqüência de ação, ou seja:

$$custo = \sum [freqüência\ de\ transição(k) * custo\ de\ transição(k)]$$

onde k é a transição específica de um objeto para o próximo como uma tarefa específica.

O *LA* é definido como:

$$LA = 100 * [(custo\ de\ LA - layout\ satisfatório) / (custo\ do\ layout\ proposto)]$$

LA = 100 para um *layout* satisfatório.

Para se determinar o *layout satisfatório* tem-se a seguinte fórmula:

$$número\ de\ layout\ possíveis = [N! / (K! * (N-K)!] * K!$$

onde para cada layout existe N possíveis posições e K diferentes objetos por posição.

Conforme o número de posições de *layout* aumenta, o número de possíveis *layouts* cresce mais ainda. Portanto, pode-se concluir que o número de *layouts* pode ser muito grande para que possa ser analisado um a um. [Sears, 93] não está preocupado com esse fato, mas sim em determinar entre o *layout apropriado* e o *proposto* qual é o melhor. O *LA* serve como uma diretriz na escolha do melhor *layout* para uma aplicação particular. Mas a

decisão final neste tipo de métrica deve partir de um *feedback* do usuário baseado em um protótipo do *layout*, pois apesar de se ter um valor menor o usuário pode desejar um *layout* da sua maneira e o valor do *LA* ser superior ao sugerido.

Como exemplo desta métrica, podemos considerar a abertura de um arquivo em uma janela *Windows*. O primeiro passo é saber as transições feitas pelo usuário e os seus custos. Na Figura 7, é mostrado o diagrama de transição indicando qual a freqüência de cada transição feita [Sears, 93].

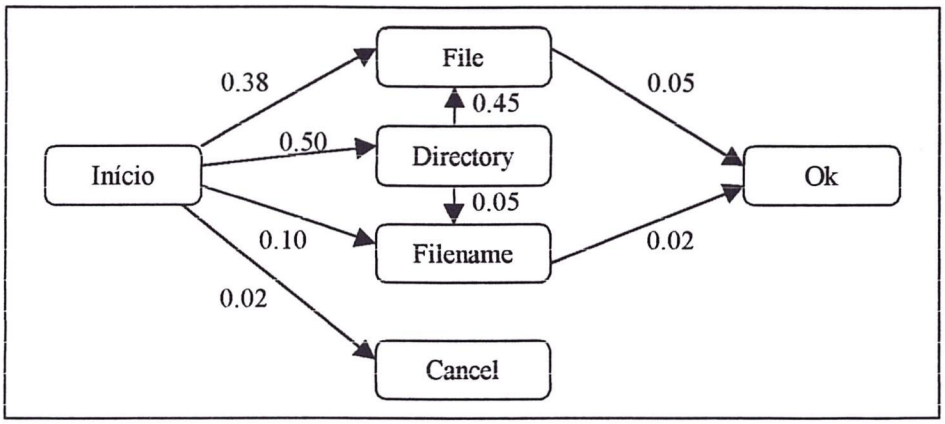

Figura 7. Diagrama de Transição. As ligações indicam a freqüência com que cada transição é feita [Sears, 93]

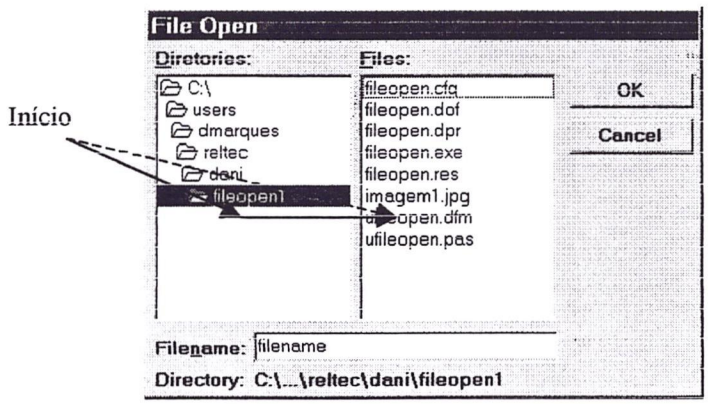

Figura 8. *Layout Satisfatório*, custo=3.92, *LA*=100. A linha contínua representa a seqüência mais freqüente de ações. A linha pontilhada a segunda seqüência mais freqüente de ações [Sears, 93]

O *Layout Satisfatório* é dado pela Figura 8 e o *Layout Proposto* é dado pela Figura 9 [Sears, 93]. Calculando o custo e o *LA* de cada *layout* conclui-se que o primeiro *layout* proposto é o melhor, seu custo é de 3.92 e o *LA* é de 100, seu *layout* segue a maior seqüência de ação, o segundo *layout* tem o custo de 4.23 e o *LA* é de 93, a desvantagem desse segundo *layout* sobre o primeiro, é que neste o primeiro objeto a ser apresentado é o *Filename*, que não é a seqüência mais utilizada.

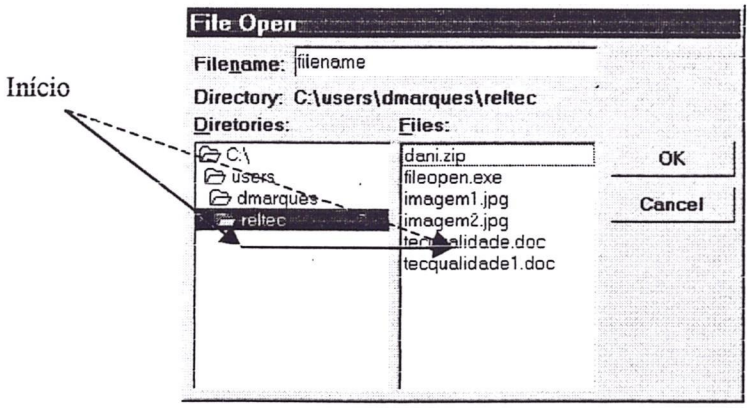

Figura 9. *Layout Proposto, custo=4.23, LA*=93. A linha contínua representa a seqüência mais freqüente de ações. A linha pontilhada a segunda seqüência mais freqüente de ações [Sears, 93]

5.3 Métricas para Codificação

Nesta fase do ciclo de vida é feita a tradução das representações do projeto para uma linguagem "artificial" resultando em instruções executáveis pelo computador. As métricas nesta fase relacionam-se diretamente à complexidade medida das linhas de código. Algumas métricas foram propostas considerando principalmente fatores como LOC, outras além deste fator, consideram o número de operadores e operandos utilizados no programa.

5.3.1 Contagem das Linhas de Código

Esta métrica de complexidade é considerada uma das mais conhecidas e fáceis de se utilizar. A complexidade medida refere-se às linhas de código (LOC) de um programa, pois geralmente a cada linha de código é associado um comando. Em caso de programas muito grandes é apropriado se medir em milhares de linhas de código (KLOC).

Esta métrica geralmente se aplica a linguagens procedimentais e não se tem um número ideal de linhas. Alguns autores dizem que o número ideal de linhas em um módulo é 50, outros já discordam dizendo que é 30 [Herbert & Price, 95]. Esta métrica geralmente penaliza projetos muitos bem estruturados que tem um pequeno número de linhas.

5.3.2 Teoria de Halstead

[Halstead, 77] estuda software como ciência. A proposta de sua teoria se baseia no fato de que a complexidade não pode estar somente relacionada com as linhas de código, mas também à contagem dos operandos e operadores.

A consideração de linhas de código, operandos e operadores [Halstead, 77] fazem com que os programas que utilizam muitas fórmulas, com grandes manipulações de dados, possuam um grande valor de complexidade associado ao fluxo de dados. O mesmo ocorrendo com programas que utilizam muitas comparações envolvendo variáveis e operadores.

As medidas primitivas da ciência de software de Halstead que podem ser derivadas depois que o código é gerado, ou estimado assim que o projeto está completo são:

- n_1: número de operadores distintos que aparecem num programa.
- n_2: número de operandos distintos que aparecem num programa.
- N_1: número total de ocorrências de operadores.
- N_2: número total de ocorrências de operandos.

Baseado nas medidas primitivas, Halstead desenvolveu expressões para e extensão global do programa, o volume mínimo potencial para um algoritmo, o volume real (número de bits exigido para especificar um programa), o nível do programa (uma medida de complexidade de software), o nível de linguagem (uma constante para uma determinada linguagem) e outras características tais como esforço de desenvolvimento, o tempo de desenvolvimento e até mesmo o número projetado de falhas no software. As expressões propostas são:

- *Vocabulário:* $n = n_1 + n_2$
- *Comprimento:* $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$
- *Volume:* $V = N \log_2(n_1 + n_2)$
- *Nível Volumétrico:* $L = (2 / n_1) \times (n_2 / N_2)$
- *Esforço:* $E = V / L$
- *Falhas:* $B = V / S^*$, onde S^* é um número significativo de decisões entre erros (S^* é 3000 de acordo com Halstead)

O trabalho de Halstead teve um grande impacto nas medidas de software. Apesar de haver críticas ao trabalho, o trabalho é receptivo à verificação experimental e foi feita uma grande pesquisa para investigá-lo [Herbert & Price, 95] [Kan, 97] [Pressman, 95] [Pressman, 97].

5.4 Métricas para Teste

As métricas para testes podem ser classificadas de duas formas. A primeira refere-se às características internas do software, garantindo que todas as instruções tenham sido testadas; a segunda refere-se ao processo de software, ou seja, são métricas que estimam os esforços dos testes e são computadas nas fases de análise, projeto e codificação. Neste trabalho, o objetivo é explorar essas duas formas.

5.4.1 Métricas Baseadas no Processo de Teste

As métricas para processo de teste podem ser divididas em três tipos básicos [Herbert & Price, 95]:

- técnicas baseadas na análise estática não auxiliada por computador;
- técnicas baseadas na especificação do programa (teste caixa-preta ou teste funcional);
- técnicas baseadas na estrutura do programa.

5.4.1.1 Técnicas Baseadas na Análise Estática não Auxiliada por Computador

O princípio desta técnica é que os programas devem ser lidos por pessoas, sendo este um processo revelador de erros e que possibilita o entendimento pleno do programa. Esta técnica tem uma certa incredibilidade por ser informal, mas experiências realizadas verificaram que cerca de 30% a 70% do total de erros do programa foram encontrados utilizando-a [Herbert & Price, 95]. Os tipos mais comuns desta técnica são:

- Inspeções

  As inspeções devem ser realizadas por um grupo de aproximadamente quatro pessoas, das quais uma deve ser o moderador (programador experiente), a outra o programador do programa em questão, o outro um projetista e o outro um especialista em testes. Estas pessoas receberão a listagem do programa alguns dias antes da inspeção. As reuniões que devem durar até 2 horas e devem ser realizadas em ambiente calmo, sem interrupções.
  Neste caso o programa será lido comando a comando pelo programador com o objetivo de entender o programa e encontrar supostos erros. Os erros anotados serão entregues ao programador, mas sem solução. A solução será proposta em outro processo. Caso existam muitos erros, é recomendável analisar o prazo de entrega e independente de erros ou não, que os resultados sejam divulgados somente às pessoas envolvidas no programa.

- *Walkthroughs*

  O *walkthrough* segue o mesmo estilo das inspeções. Deve existir de três a cinco pessoas envolvidas na reunião, a reunião deve ser feita em lugar calmo e sem interrupções, o programa é entregue às pessoas envolvidas alguns dias antes da reunião. Dentre as pessoas envolvidas deve existir um moderador, outro que registrará as informações do processo (tais como erros encontrados e informações adicionais), e o programador que irá manter o programa, as demais pessoas podem ser programadores ou especialistas em linguagem de programação, e os programadores podem ser do mesmo projeto ou não.
  Nesta técnica os participantes simulam a tarefa a ser desenvolvida pelo computador. Existe uma pessoa responsável pelos testes que deverão ser feitos, com suas entradas e saídas esperadas, isso já deve ser feito previamente antes da reunião. Durante a simulação, os participantes irão checar os erros encontrados, os erros novamente não serão corrigidos, mas sim listados e entregues ao final da reunião. Com a lista de erros na mão, o programador responsável corrigirá os erros o quanto antes possível.

5.4.1.2 Teste Funcional (caixa-preta)

O teste funcional envolve a submissão de dados de entrada ao programa a ser testado, e a comparação dos dados de saída do programa com os dados esperados (que devem ser previamente calculados). Por se tratar de uma caixa-preta, a estrutura interna do programa não é considerada, ou seja, não há interesse em se saber como o programa gerou suas saídas, só há interesse em se saber se as saídas estão corretas.

5.4.1.3 Teste Estrutural

O teste estrutural considera a estrutura interna do programa. O programa é tratado como uma caixa-branca, onde os dados de saída e os caminhos executados para chegar a esses dados são de igual importância. O teste estrutural é dependente da implementação e por esse motivo existe uma grande quantidade de informação utilizada por eles, o que torna essencial o uso de ferramentas automatizadas para execução desta tarefa.

A seguir são apresentadas algumas técnicas que se baseiam no teste estrutural:

- Teste de Módulo

  Este teste é realizado seguindo alguns critérios de seleção de caminhos para a unidade [Rapps & Weyuker, 85]. Estes critérios para serem satisfeitos, constam de uma lista de subcaminhos no grafo de fluxo de controle que devem ser executados. Em casos de testes, os subcaminhos são selecionados e através de várias execuções do módulo, avalia-se quais deles foram percorridos.

- Teste de Sistema

  Este tipo de teste enfatiza a análise do comportamento da estrutura hierárquica de chamadas de módulos, verificando se os parâmetros estão seguindo um fluxo normal e se seus valores coincidem com os valores esperados [Harrison & Magel, 81].

5.4.2 Métricas Baseadas nas Características dos Testes

A seguir citamos algumas métricas que podem estimar os esforços dos testes:

- Métricas baseadas em função: podem ser usadas como uma previsão para o esforço total dos testes (ao nível de projeto temos como exemplo: esforços de testes e tempo, erros encontrados, número de casos de testes produzidos);
- Métricas de projeto de Alto Nível: fornece informações de fácil ou difícil associação com testes de integração e a necessidade de testes de software especializados;
- Teoria de Halstead: estima esforços de teste. Usando a definição de volume *(V)*, e nível de programa *(L)*, o esforço do conhecimento de software *(E)* pode ser definido como: $E = V / L$.

5.5 Métricas para Manutenção

Quando um produto de software é completamente desenvolvido e existe a necessidade de alterações, entra-se na fase de manutenção do ciclo de vida. A fase de manutenção pode ser de quatro categorias: *corretiva*, quando são realizadas mudanças para correção de erros no software; *preventiva*, quando são realizadas mudanças para melhorar o software existente com foco na manutenibilidade e confiabilidade; *perfectiva*, quando são feitas alterações para melhorar o software em qualidade ou na sua documentação e *adaptativa*, ocorre para acomodar mudanças ocorridas no ambiente externo.

As métricas propostas até esta seção servem tanto para a criação de um novo software, como para sua manutenção, mas existem métricas explicitamente propostas para esta fase

do ciclo de vida. Descreveremos a seguir o Padrão IEEE 982.1-1988 [IEEE, 94] que se enquadra como um modelo de métrica para a fase de manutenção.

5.5.1 Padrão IEEE 982.1-1988

Este padrão sugere um índice de maturidade de software SMI *(Software Maturity Index)* que fornece uma indicação da estabilidade do produto de software (baseado na alteração que ocorre para cada versão do produto). Para se determinar o SMI do produto é necessário antes determinar as seguintes informações:

- M_T: número de módulos na versão corrente.
- F_c: número de módulos na versão corrente que foram alterados.
- F_a: número de módulos na versão corrente que foram adicionados.
- F_d: número de módulos na versão precedente que foram removidos na versão corrente.

Coletados esses dados, o SMI é dado pela seguinte fórmula:

$$SMI = [M_T - (F_a + F_c + F_d)] / M_T$$

Observa-se que esse índice SMI considera principalmente a unidade modular de composição do software, uma vez que todas as informações coletadas dizem respeito à quantidade de módulos envolvidos tanto no processo de manutenção, quanto os que não sofreram manutenção. Outras métricas podem ser encontradas, cuja unidade pode variar conforme a característica arquitetural do software (por exemplo, para software construídos sob a abordagem Orientada a Objetos, classes, métodos seriam mais convenientes).

6. Conclusão

Este trabalho apresenta diversas métricas existentes na literatura que visam garantir a qualidade do produto de software através de um caminho sistemático baseado em um conjunto de regras claramente definidas. Inegavelmente, a qualidade de software é uma característica fundamental nos produtos de software.

Inicialmente, foi introduzido o tema *"Total Quality Management"* e quais são seus elementos chaves, mostrando o relevante papel das métricas para obtenção de qualidade. A seguir, "Qualidade de Software" e suas diversas concepções foram discutidas, bem como os modelos de qualidade como de McCall e FURPS, com o intuito de deixar claro como as métricas podem ser derivadas em outros atributos de forma que se tornem mensuráveis.

O estudo de "Métricas de Software" foi realizado com o objetivo de esclarecer sua utilidade, as divisões que elas podem ter (medidas diretas ou indiretas). Além disso, pode-se observar que, para que uma métrica seja útil no mundo real é necessário que a mesma seja simples e computável, de persuasão intuitiva, consistente e objetiva, que possa ser implementada em linguagem independente e que forneça um *feedback* efetivo ao engenheiro de software.

Foram então apresentadas as métricas de software, tendo como foco a qualidade em cada fase do ciclo de vida. Na fase de análise, a ênfase é dada com o intuito de predizer o tamanho do sistema resultante ou assegurar qualidade no modelo de análise. Na fase de projeto a preocupação é com os quatro atributos do programa: estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização da interface. As métricas relatadas nesta fase focalizam a arquitetura de software e a caracterização da interface. Na fase de codificação o enfoque é dado diretamente à complexidade do código, utilizando fatores como número de código, número de operadores e operandos utilizados no programa. Apesar de não existirem muitas métricas para uso direto em teste e manutenção de software, foram apresentadas métricas técnicas para conduzir o processo de teste e assegurar a manutenibilidade de um programa de computador.

O Quadro 3 resume as métricas estudadas neste trabalho, relacionando-as às fases do ciclo de vida para as quais são pertinente e os dados que devem ser coletados para que cada métrica seja computada.

Apesar das métricas de software não serem medidas absolutas, elas devem ser usadas para melhorar a qualidade do software, pois sem elas os softwares não teriam o nível de qualidade que têm hoje. A cada dia, novas métricas são propostas com o objetivo de se obter qualidade, visando atender diferentes perspectivas. Um exemplo de perspectiva é a utilização de métricas definidas para diferentes domínios.

Quadro 3. Resumo das Métricas de Software X Fases do Ciclo de Vida

Fase do Ciclo de Vida	Métrica	Dados a serem coletados
Análise	Pontos por Função	Número de entradas do usuário, número de saídas do usuário; número de consultas do usuário; número de arquivos; número de interfaces externas e respostas graduadas das questões: Sistema exige backup e recuperação confiáveis? É requerida comunicação de dados? Existem funções de processamento distribuído? O desempenho é crítico? O sistema funcionará num ambiente operacional existente, intensivamente utilizado? O sistema requer entrada de dados *on-line*? A entrada de dados *on-line* exige que a transição de entrada seja elaborada em múltiplas telas ou operações? Os arquivos mestres são atualizados *on-line*? A entrada, saída, arquivos ou consultas são complexas? O processamento interno é complexo? O código foi projetado de forma a ser reusável? A conversão e a instalação estão incluídas no projeto? O sistema é projetado para múltiplas instalações em diferentes organizações? A aplicação é projetada de forma a facilitar mudanças e o uso pelo usuário?
	Especificação	Número de requisitos funcionais; número de requisitos não funcionais e número de requisitos que todos os revisores tem interpretação idêntica.

Projeto	Complexidade	Número de LOC estimado no módulo; número de fluxos locais (são criados por passagem de parâmetros) terminando no módulo e número de fluxos locais originando-se no módulo.
	Size, Depth e Width	Número de nós e número de arcos.
	DSQI	Número total de módulos na arquitetura de programa; número de módulos cuja função dependa da fonte de entrada de dados ou que produza dados a serem usados em outro lugar; número de módulos cuja função correta dependa de processamento anterior; número de itens de banco de dados; número total de bancos de dados únicos; número de segmentos do banco de dados (registros diferentes ou objetos individuais) e número de módulos com uma única entrada e uma única saída.
	Layout Apropriado	Conjunto de objetos (ícones gráficos, textos, menus) usados na interface; a seqüência de ações desenvolvidas pelos usuários e com que freqüência cada ação é usada.
Codificação	Contagem LOC	Linhas de código.
	Teoria de Halstead	Número de operadores distintos que aparecem num programa; número de operandos distintos que aparecem num programa; número total de ocorrências de operadores e número total de ocorrências de operandos.
Testes	Teste de Módulo	Lista de subcaminhos no grafo de fluxo de controle que devem ser executados.
	Teste de Sistema	Estrutura hierárquica de chamadas de módulos.
	Baseada nas Características dos Testes	Número de entradas do usuário, número de saídas do usuário; número de consultas do usuário; número de arquivos; número de interfaces externas; número de LOC estimado no módulo; número de fluxos locais (são criados por passagem de parâmetros) terminando no módulo e número de fluxos locais originando-se no módulo; número de nós e número de arcos; número de operadores distintos que aparecem num programa; número de operandos distintos que aparecem num programa; número total de ocorrências de operadores e número total de ocorrências de operandos.e respostas graduadas das questões: O sistema exige backup e recuperação confiáveis? É requerida comunicação de dados? Existem funções de processamento distribuído? O desempenho é crítico? O sistema funcionará num ambiente operacional existente, intensivamente utilizado? O sistema requer entrada de dados *on-line*? A entrada de dados *on-line* exige que a transição de entrada seja elaborada em múltiplas telas ou operações? Os arquivos mestres são atualizados *on-line*? A entrada, saída, arquivos ou consultas são complexas? O processamento interno é complexo? O código foi projetado de forma a ser reusável? A conversão e a instalação estão incluídas no projeto? O sistema é projetado para múltiplas instalações em diferentes organizações? A aplicação é projetada de forma a facilitar mudanças e o uso pelo usuário?
Manutenção	SMI	Número de módulos na versão corrente; número de módulos na versão corrente que foram alterados; número de módulos na versão corrente que foram adicionados; número de módulos na versão precedente que foram removidos na versão corrente.

Bibliografia

[Albrecht, 79] ALBRECHT, A. J. Measuring Application Development Productivity. In: IBM Applic. Dev. Symposium, *Proceedings.* Monterey, CA, outubro de 1979. p.83-92.

[Card & Glass, 90] CARD, D. N.; GLASS, R. L. *Measuring Software Design Quality.* Prentice-Hall, 1990.

[Charette, 89] CHARETTE, R. N. *Software Engineering Risck Analysis and Management.* McGraw-Hill/Intertext, 1989.

[Davis et al., 93] DAVIS, A.; OVERMYER, S.; JORDAN, K.; CARUSO, J.; DANDASHI, F.; DINH, A.; KINCAID, G.; LEDEBOER, G.; REYNOLDS, P.; SRIMANI, P.; TA, A.; THEOFANOS, M. Identifying and Measuring Quality in a Software Requirements Specification. In: *Proc. 1st Intl. Software Metric Symposium*, IEEE, Baltimore, MD, maio de 1993. p.141-152.

[DeMarco, 82] DEMARCO, T. *Controlling Software Projects.* Yourdon Press, 1982.

[Ejiogu, 91] EJIOGU, L. *Software Engineering with Formal Metrics.* QED Publishing, 1991.

[Fenton, 91] FENTON, N. *Software Metrics.* New York, Chapman & Hall, 1991.

[Fenton, 94] FENTON, N. Software Measurement: A Necessary Scientific Basis. *IEEE Trans. Software Engineering*, vol. 20, no. 3, p.199-206, Março de 1994.

[Fenton & Pfleeger, 97] FENTON, N.; PFLEEGER, S.L. *Software Metrics – A rigorous & Practical Approach.* 2nd Edition, Boston, Thompson Computer Press, 1997.

[Grady & Caswell, 87] GRADY, R. B.; CASWELL, D. L. *Software Metrics: Establishing a Company-Wide Program.* Prentice-Hall, 1987.

[Halstead, 77] HALSTEAD, M. H. *Elements of Software Science.* Elsevier Computer Science Library, New York, 1977.

[Harrison & Magel, 81] HARRISON, W.; MAGEL, K. A Complexity Measure based on Nesting Level. *ACM SIGPLAN Notices*, vol. 16, no. 3, p.63-74, Março de 1981.

[Henry & Kafura, 81] HENRY, S.; KAFURA, D. Software Structure Metrics Based on Information Flow. *IEEE Trans. Software Engineering*, vol. SE-7, no.5, p.510-518, Setembro de 1981.

[Herbert & Price, 95] HERBERT, J. S.; PRICE, A. M. A. Métodos para a Avaliação da Qualidade de Software. In: Congresso da Sociedade Brasileira de Computação,

15. – JAI'95 –Jornada de Atualização em Informática, 14. Canela – RS, julho de 1995.

[IEEE, 94] *IEEE Software Engineering Standards*, 1994 edition, IEEE 1994.

[Kan, 94] KAN, S. H. *Metrics and Models in Software Quality Engineering.* Addison Wesley, 1997.

[McCall et al., 77] MCCALL, J.; RICHARDS, P.; WALTERS, G. *Factors in Software Quality.* três volumes, NTIS AD-A049-014, 015, 055, novembro de 1977.

[Mendes et al., 98] MENDES, E.; HARRISON, R.; HALL, W. Evaluation of Reuse and Maintenance in Hypermedia Applications for Education: Validation of Metrics. In: Congresso Ibero-americano de Informática na Educação (RIBIE'98), 4., *Anais.* Brasília – DF, outubro de 1998.

[Petzold & Yao, 97] PETZOLD, C.; YAO, P. *Programando para Windows 95.* Makron Books, São Paulo, 1997.

[Pressman, 95] PRESSMAN, R. S. *Engenharia de Software*. Makron Books, 3ª. Edição, São Paulo – SP, 1995.

[Pressman, 97] PRESSMAN, R. S. *Software Engineering - A Practitioner's Approach.* 4th Edition. McGraw-Hill, USA, 1997.

[Rapps & Weyuker, 85] RAPPS, S.; WEYIKER, E. "Selecting Software Test Data using Data Flow Information". *IEEE Trans. On Software Engineering*, vol. 11, no. 4, Abril de 1985.

[Roche, 94] ROCHE, J. M. "Software Metrics and Measurement Principles". *Software Engineering Notes (ACM)*, vol. 19, no 1, p.76-85, Janeiro de 1994.

[Rocha, 98] ROCHA, A. R. C. Planejamento e Avaliação da Qualidade de Software. In: Conferencia Internacional de Tecnologia de Software (CITS), 9., *Anais.* Curitiba – PR, junho de 1998, .

[Sears, 93] SEARS, A. Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout. *IEEE Trans. Software Engineering*, vol. 19, n. 7, p.707-719, Julho de 1993.