

## Qualidade de Produto

Este capítulo apresenta os conceitos de *qualidade de produto*, iniciando com o modelo de qualidade da recém-aprovada Norma ISO/IEC 25010:2011 (Seção 11.1), que define os atributos de qualidade internos, externos e de uso de produtos de software. Em seguida, o capítulo apresenta um método para construir outros modelos de qualidade a partir de critérios definidos (Seção 11.2). Na sequência, apresenta informações sobre como instalar um *programa de melhoria de qualidade de produto* (Seção 11.3) e como fazer a *gestão da qualidade* (Seção 11.4). Finalmente, são apresentados os conceitos de *medição da qualidade* (Seção 11.5), *requisitos de qualidade* (Seção 11.6) e o método *GQM* (Seção 11.7), usado para avaliação da qualidade de produtos de software.

*Qualidade de software* é uma área dentro da Engenharia de Software que visa garantir bons produtos a partir de bons processos. Pode-se falar, então, de dois aspectos da qualidade: a *qualidade do produto* em si e a *qualidade do processo*. Embora não exista uma garantia de que um bom processo vá produzir um bom produto, em geral admite-se que a mesma equipe com um bom processo vá produzir produtos melhores do que se não tivesse processo algum.

Qualidade de software é um assunto amplo e de definição difusa. Existem várias dimensões de qualidade, e nem sempre é simples avaliar objetivamente cada uma delas. Pressman (2005) define dois tipos de qualidade para o produto de software:

- Qualidade de projeto*, que avalia quão bem o produto foi projetado.
- Qualidade de conformação*, que avalia quão bem o produto atende aos requisitos.

Em relação à qualidade, o SWEBOK<sup>1</sup> faz uma distinção entre técnicas estáticas e dinâmicas. As técnicas estáticas são apresentadas neste capítulo como “qualidade de software” e as técnicas dinâmicas são relacionadas ao teste do software (Capítulo 13).

### 11.1 Modelo de Qualidade SquaRE – ISO/IEC 25010:2011

A Norma NBR ISO/IEC 9126-1:2003<sup>2</sup> define *qualidade de software* como “a totalidade de características de um produto de software que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas”. A definição, propositalmente, não especifica os possuidores de tais necessidades, visto que se aplica a quaisquer atores envolvidos

<sup>1</sup>Disponível em: <[www.computer.org/portal/web/swebok](http://www.computer.org/portal/web/swebok)>. Acesso em: 21 jan. 2013.

<sup>2</sup>Disponível em: <[www.abntcatalogo.com.br/norma.aspx?ID=2815](http://www.abntcatalogo.com.br/norma.aspx?ID=2815)>. Acesso em: 21 jan. 2013.

**TABELA 11.1** Fases genéricas do ciclo de vida da Norma ISO/IEC 15288 e suas equivalentes SQuaRE

Fases da 15288	Agrupamento de fases de SQuaRE
Processo de definição de requisitos dos interessados	Requisitos de qualidade do produto de software
Processo de análise dos requisitos	Desenvolvimento do produto
Processo de <i>design</i> arquitetural	
Processo de implementação	
Processo de integração	
Processo de verificação	
Processo de transição	
Processo de validação	
Processo de operação	Uso do produto
Processo de manutenção	
Processo de aposentadoria	-

com a produção, encomenda, uso ou pessoas afetadas pelas consequências do software ou de seu processo de produção.

Essa norma, referência de atributos de qualidade por vários anos, foi substituída em julho de 2011 pela ISO/IEC 25010, parte da nova família de Normas ISO/IEC 25000: *Software Engineering: Software Product Quality Requirements and Evaluation (SQuaRE)*<sup>3</sup>.

Uma das motivações para a criação de uma nova norma está no fato de que as antigas aplicavam-se apenas ao processo de desenvolvimento e uso do produto de software, mas pouco tinham a dizer em relação à definição do produto.

Para a definição de uma segunda geração de normas de qualidade, foi então utilizado um modelo genérico de processo de desenvolvimento baseado na Norma ISO/IEC 15288 – *System Life Cycle Processes*<sup>4</sup> (Tabela 11.1).

O lado esquerdo da tabela apresenta as fases originais da Norma 15288, e o lado direito apresenta o agrupamento dessas fases para efeito da aplicação das normas de qualidade SQuaRE. Apenas o processo de aposentadoria de software ainda não é contemplado pelo novo conjunto de normas.

O modelo de qualidade SQuaRE avalia quatro tipos de indicadores de qualidade:

- Medidas de qualidade do processo*: avaliam a qualidade do processo usado para desenvolver os produtos e, consequentemente, a maturidade da empresa em termos de processos de engenharia de software. Mais detalhes sobre qualidade de processo podem ser encontrados no Capítulo 12.
- Medidas de qualidade internas*: avaliam aspectos internos da qualidade do software que normalmente só são percebidos pelos desenvolvedores (por exemplo, capacidade de manutenção).
- Medidas de qualidade externas*: avaliam aspectos externos da qualidade do software que podem ser avaliados pela equipe de desenvolvimento do ponto de vista do usuário (por exemplo, eficiência).
- Medidas de qualidade do software em uso*: avaliam aspectos da qualidade do software em seu ambiente final que só podem ser medidos pelos usuários finais (por exemplo, satisfação dos usuários).

Assim, as *qualidades internas* permitem que a equipe de desenvolvimento atinja seus objetivos de forma eficiente. As *qualidades externas* e de *uso* permitem que o usuário final do sistema atinja seus objetivos. As qualidades internas, portanto, nem sempre são importantes para o usuário final, pois ele não as percebe diretamente, mas pode ser afetado indiretamente por elas (no tempo de manutenção ou evolução do software, por exemplo).

Defende-se que a qualidade de processo influencia a qualidade interna, que influencia a qualidade externa, que, por sua vez, influencia a qualidade de uso do software (Figura 11.1).

<sup>3</sup>Disponível em: <[www.iso.org/iso/catalogue\\_detail.htm?csnumber=35683](http://www.iso.org/iso/catalogue_detail.htm?csnumber=35683)>. Acesso em: 21 jan. 2013.

<sup>4</sup>Disponível em: <[www.iso.org/iso/catalogue\\_detail?csnumber=43564](http://www.iso.org/iso/catalogue_detail?csnumber=43564)>. Acesso em: 21 jan. 2013.

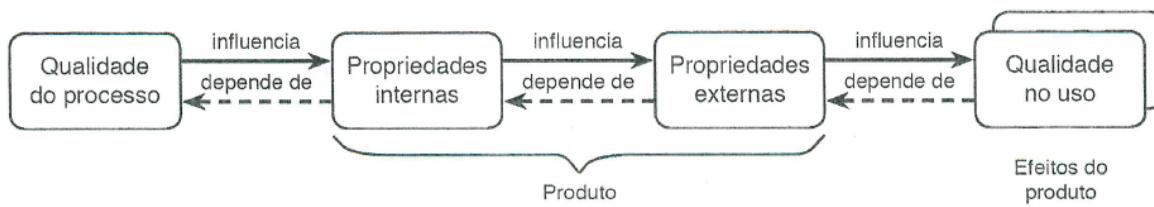


Figura 11.1 Abordagem conceitual para qualidade de acordo com a ISO/IEC 25010:2011.

Na figura, o bloco “Qualidade no uso” é mostrado como um bloco múltiplo, já que, em diferentes contextos de uso, o mesmo produto pode ter diferentes avaliações de sua qualidade no uso.

O modelo SQuaRE (Suryn & Abran, 2003)<sup>5</sup> é formado por um conjunto de normas dividido da seguinte forma:

- ISO/IEC 2500n – Divisão Gestão da Qualidade.
- ISO/IEC 2501n – Divisão Modelo de Qualidade.
- ISO/IEC 2502n – Divisão Medição da Qualidade.
- ISO/IEC 2503n – Divisão Requisitos de Qualidade.
- ISO/IEC 2504n – Divisão Avaliação da Qualidade.

A divisão de *gestão da qualidade* do modelo SQuaRE apresenta os modelos comuns, padrões básicos, termos e definições usados por toda a série de normas SQuaRE. Essa divisão inclui duas unidades:

- Guia do SQuaRE*, que apresenta a estrutura, terminologia, visão geral do documento, público-alvo, modelos de referência e partes associadas da série.
- Planejamento e Gerenciamento*, que apresenta os requisitos para planejar e gerenciar o processo de avaliação da qualidade de produtos de software.

O *modelo de qualidade* apresenta as características e subcaracterísticas de qualidade interna, externa e de uso. Os padrões na área de medidas de qualidade são derivados das Normas 9126 e 14598, cobrindo as definições matemáticas e o detalhamento da aplicação de medidas práticas de qualidade interna, externa e de uso. O documento inclui o seguinte:

- Modelo de referência e guia de medição*: apresenta uma introdução e explicação sobre a aplicação das medidas de qualidade.
- Medidas primitivas*: conjunto de medições básicas usadas para a definição das demais.
- Medidas internas*: conjunto de medidas quantitativas em termos de características e subcaracterísticas internas.
- Medidas externas*: conjunto de medidas quantitativas em termos de características e subcaracterísticas externas.
- Medidas de uso*: conjunto de medidas quantitativas em termos de características e subcaracterísticas de uso do software.

A divisão de *requisitos de qualidade* contém o padrão para suportar a especificação de requisitos de qualidade, tanto para a fase de elicitação dos requisitos de qualidade do software quanto como entrada para o processo de avaliação da qualidade do software.

A divisão de *avaliação da qualidade* provê as ferramentas para a avaliação da qualidade de um sistema de software tanto por desenvolvedores, compradores ou avaliadores independentes. Os seguintes documentos são disponibilizados:

- Guia e visão geral da avaliação da qualidade*: apresenta um *framework* para a avaliação da qualidade de um produto de software.
- Processo para desenvolvedores*: recomendações práticas para a avaliação da qualidade de um produto quando esta é feita paralelamente ao seu desenvolvimento.

<sup>5</sup>Disponível em: <[prof.etsmtl.ca/wsurn/research/SQE-Publ/SQuaRE-second%20generation%20of%20standards%20for%20SW%20Quality%20%28IASTED03%29.pdf](http://profs.etsmtl.ca/wsurn/research/SQE-Publ/SQuaRE-second%20generation%20of%20standards%20for%20SW%20Quality%20%28IASTED03%29.pdf)>. Acesso em: 21 jan. 2013.

- c) *Processo para compradores*: recomendações práticas para a avaliação de produtos comprados em prateleira (COTS), feitos por encomenda, ou ainda para avaliação de modificações em sistemas existentes.
- d) *Processo para avaliadores*: recomendações práticas para a avaliação do software por terceiros, enfatizando a participação de vários agentes que precisam compreender e aceitar essa avaliação.
- e) *Documentação para o módulo de avaliação*: define a estrutura e o conteúdo da documentação usada no processo de avaliação.

O modelo de qualidade da ISO/IEC 25010 define um conjunto de oito características internas e externas de produto de software, subdivididas em subcaracterísticas e mais cinco características de software *em uso*, algumas das quais também são subdivididas em subcaracterísticas.

O modelo resultante é mostrado na Tabela 11.2. As características internas e externas do software são agrupadas nas chamadas *características do produto*, pois podem ser avaliadas no ambiente de desenvolvimento. Já as características do software *em uso* só podem ser avaliadas no contexto de uso do sistema. As características e subcaracterísticas da tabela são apresentadas em português e em inglês, pois a tradução pode nem sempre apresentar o espírito do termo na sua língua original.

O conjunto de características e subcaracterísticas dessa norma modificou-se bastante ao longo do tempo, enquanto ela era elaborada. Assim, é possível encontrar versões diferentes de características e subcaracterísticas na literatura, pois estas podem ter se baseado em versões intermediárias da norma. A lista apresentada aqui corresponde à versão definitiva, publicada em 2011<sup>6</sup>. A seguir, as características do modelo de qualidade são brevemente apresentadas.

### 11.1.1 ADEQUAÇÃO FUNCIONAL

A *adequação funcional* mede o grau em que o produto disponibiliza funções que satisfazem às necessidades estabelecidas e implicadas quando o produto é usado sob condições especificadas. Suas subcaracterísticas são:

- a) *Completude funcional*: o software efetivamente possibilita executar as funções que são apropriadas, ou seja, as entradas e saídas de dados necessárias para o usuário atingir seus objetivos são possíveis? Um software no qual faltam algumas funções necessárias não apresenta a qualidade de completude funcional.
- b) *Corretude funcional*: também denominada *acurácia*, essa subcaracterística avalia o quanto o software gera dados e consultas corretos e precisos de acordo com sua definição. Um software que apresenta dados incorretos ou com grau de imprecisão acima de um limite definido como tolerável não apresenta a qualidade de corretude funcional.
- c) *Funcionalidade apropriada*: essa subcaracterística indica em qual grau as funções do sistema facilitam a realização de tarefas e objetivos para os quais o sistema foi especificado.

### 11.1.2 CONFIABILIDADE

Um software *confiável* é aquele que, ao longo do tempo, se mantém com um comportamento consistente com o esperado. A confiabilidade tem relação com a minimização da quantidade de defeitos do software e com a forma como ele funciona perante situações anômalas. As subcaracterísticas da confiabilidade são:

- a) *Maturidade*: a maturidade é a medida da frequência com que um software apresenta defeitos. Um software mais maduro é aquele que apresenta menos defeitos ao longo de um período fixo de tempo. Espera-se que a maturidade de um sistema aumente com o tempo, mas processos de manutenção mal gerenciados, especialmente aqueles que deixam de realizar testes de regressão (Seção 13.2.6) ou refatoração, podem fazer que a maturidade de um sistema diminua com o passar do tempo, ou seja, em vez de reduzir a frequência dos erros, eles podem aumentar.
- b) *Disponibilidade*: essa subcaracterística avalia o quanto o software está operacional e disponível para uso quando necessário.
- c) *Tolerância a falhas*: a subcaracterística de tolerância a falhas tem relação com a forma como o software reage quando em situação anômala. Idealmente, requisitos de tolerância a falhas deveriam ser definidos durante o projeto do software. Um software que consegue continuar funcionando mesmo quando ocorrem

<sup>6</sup>Disponível em: <[www.meti.go.jp/policy/it\\_policy/softseibi/metrics/20110324product\\_metrics2010%28en%29.pdf](http://www.meti.go.jp/policy/it_policy/softseibi/metrics/20110324product_metrics2010%28en%29.pdf)>. Acesso em: 21 jan. 2013.

**TABELA 11.2** Modelo de qualidade da ISO 25010:2011

Tipo	Características	Subcaracterísticas	Termo em inglês
Características do produto	Adequação funcional ( <i>functional suitability</i> )	Completude funcional Corretude funcional (acurácia) Funcionalidade apropriada	<i>Functional completeness</i> <i>Functional correctness (accuracy)</i> <i>Functional appropriateness</i>
	Confiabilidade ( <i>reliability</i> )	Maturidade Disponibilidade Tolerância a falhas Recuperabilidade	<i>Maturity</i> <i>Availability</i> <i>Fault tolerance</i> <i>Recoverability</i>
	Usabilidade ( <i>usability</i> )	Apropriação reconhecível Inteligibilidade Operabilidade Proteção contra erro de usuário Estética de interface com usuário Acessibilidade	<i>Appropriateness recognisability</i> <i>Learnability</i> <i>Operability</i> <i>User error protection</i> <i>User interface aesthetics</i> <i>Accessibility</i>
	Eficiência de desempenho ( <i>performance efficiency</i> )	Comportamento em relação ao tempo Utilização de recursos Capacidade	<i>Time behavior</i> <i>Resource utilization</i> <i>Capacity</i>
	Segurança ( <i>security</i> )	Confidencialidade Integridade Não repúdio Rastreabilidade de uso Autenticidade	<i>Confidentiality</i> <i>Integrity</i> <i>Non-repudiation</i> <i>Accountability</i> <i>Authenticity</i>
	Compatibilidade ( <i>compatibility</i> )	Coexistência Interoperabilidade	<i>Co-existence</i> <i>Interoperability</i>
	Capacidade de manutenção ( <i>maintainability</i> )	Modularidade Reusabilidade Analysabilidade Modificabilidade Testabilidade	<i>Modularity</i> <i>Reusability</i> <i>Analyzability</i> <i>Modifiability</i> <i>Testability</i>
	Portabilidade ( <i>portability</i> )	Adaptabilidade Instalabilidade Substituibilidade	<i>Adaptability</i> <i>Instalability</i> <i>Replaceability</i>
Características do uso	Efetividade ( <i>effectiveness</i> )	Efetividade	<i>Effectiveness</i>
	Eficiência ( <i>efficiency</i> )	Eficiência	<i>Efficiency</i>
	Satisfação ( <i>satisfaction</i> )	Utilidade Prazer Conforto Confiança	<i>Usefulness</i> <i>Pleasure</i> <i>Comfort</i> <i>Trust</i>
	Uso sem riscos ( <i>freedom from risk</i> )	Mitigação de risco econômico Mitigação de risco a saúde e segurança Mitigação de risco ambiental	<i>Economic risk mitigation</i> <i>Health and safety risk mitigation</i> <i>Environmental risk mitigation</i>
	Cobertura de contexto ( <i>context coverage</i> )	Completude de contexto Flexibilidade	<i>Context completeness</i> <i>Flexibility</i>

fallas tem boa avaliação em relação a essa qualidade. Deve-se deixar claro, porém, que falhas e defeitos são coisas diferentes. A maturidade tem a ver com a minimização de *erros* que são oriundos de *defeitos* e, portanto, indesejáveis em qualquer sistema. Uma *falha*, porém, é uma situação que pode ocorrer mesmo que o software não apresente nenhum defeito. Por exemplo, uma linha de comunicação pode ser temporariamente interrompida, um dispositivo de armazenamento pode ser danificado, um processador pode estar danificado etc. Essas falhas são imprevisíveis e, na maioria das vezes, inevitáveis. A qualidade de tolerância a falhas, então, tem relação com a maneira como o software reage a essas situações externas indesejadas.

- d) *Recuperabilidade*: a recuperabilidade de um sistema está relacionada à sua capacidade de recuperar dados e colocar-se novamente em operação após uma situação de desastre. Idealmente, deveria haver requisitos de recuperabilidade definidos para a maioria dos projetos de software, pois situações negativas, como uma falha generalizada ou perda de dados, nem sempre são previstas em um projeto, a não ser que sejam explicitamente recomendadas. Em relação à tolerância a falhas, pode-se dizer que a recuperabilidade trata de situações mais críticas em que o problema ocorrido não é apenas temporário, como uma falha de comunicação, ele é mais definitivo, como a perda completa de um disco de dados.

#### 11.1.3 USABILIDADE

A *usabilidade* avalia o grau no qual o produto tem atributos que permitem que seja entendido, aprendido, usado e que seja atraente ao usuário, quando usado sob condições especificadas. Suas subcaracterísticas são:

- a) *Apropriação reconhecível*: mede o grau em que os usuários reconhecem que o produto é apropriado para suas necessidades.
- b) *Inteligibilidade*: tem relação com o grau de facilidade que um usuário tem para entender os conceitos-chave do software e, assim, tornar-se competente no seu uso.
- c) *Operabilidade*: avalia o grau no qual o produto é fácil de usar e controlar.
- d) *Proteção contra erro de usuário*: avalia o grau em que o produto foi projetado para evitar que o usuário cometa erros.
- e) *Estética de interface com usuário*: avalia o grau em que a interface com o usuário proporciona prazer e uma interação satisfatória.
- f) *Acessibilidade*: avalia o grau em que o produto foi projetado para atender a usuários com necessidades especiais.

#### 11.1.4 EFICIÊNCIA DE DESEMPENHO

A *eficiência de desempenho* trata da otimização do uso de recursos de tempo e espaço. Espera-se que um sistema seja o mais eficiente possível de acordo com o tipo de problema que ele soluciona.

Existem problemas para os quais as soluções computacionais demandam quantidades de tempo e memória que as tornam intratáveis. Nesses problemas, a eficiência de tempo pode ser conseguida com sacrifício, por exemplo, da acurácia, como no caso de algoritmos que buscam soluções aproximadas para problemas intratáveis em tempo razoável.

A característica de eficiência de desempenho divide-se, então, em duas subcaracterísticas:

- a) *Comportamento em relação ao tempo*: essa qualidade mede o tempo que o software leva para processar suas funções. Existe para todos os problemas algorítmicos um limite mínimo de complexidade que pode ser demonstrado formalmente. Um sistema eficiente em termos de tempo, portanto, é aquele cujo tempo de processamento se aproxima desse mínimo.
- b) *Utilização de recursos*: normalmente associada a espaço de armazenamento ou memória, a eficiência de recursos também pode ser associada a outros recursos necessários, como, por exemplo, banda de transmissão de rede. Em algumas situações pode-se obter eficiência de tempo com sacrifício da eficiência de recursos e vice-versa.
- c) *Capacidade*: avalia o grau em que os limites máximos do produto atendem aos requisitos. A *capacidade* de um produto, portanto, é relativa aos seus requisitos. Por exemplo, pode-se avaliar que um produto capaz de processar 20 transações simultaneamente terá excelente capacidade se os requisitos exigirem 2 ou 3 transações simultâneas, mas o mesmo produto terá capacidade ruim caso os requisitos exijam 40 ou 50 transações



ELSEVIER

### 11.1.5 SEGURANÇA

A característica de *segurança* avalia o grau em que as funções e os dados são protegidos de acesso não autorizado e o grau em que são disponibilizados para acesso autorizado. Deve-se tomar cuidado para não confundir a qualidade de *segurança* (*security*) relacionada à segurança dos dados e funções com a qualidade de *uso seguro* (*safety*), que é uma qualidade do software em uso, relacionada à segurança das pessoas, instalações e meio ambiente.

As subcaracterísticas de segurança são:

- a) *Confidencialidade*: avalia o grau em que as informações e funções do sistema são acessíveis por quem tem a devida autorização para isso.
- b) *Integridade*: avalia o grau em que os dados e funções do sistema são protegidos contra acesso por pessoas ou sistemas não autorizados.
- c) *Não repúdio*: avalia o grau em que o sistema permite constatar que ações ou acessos foram efetivamente feitos, de forma que não possam ser negados posteriormente.
- d) *Rastreabilidade de uso*: avalia o grau em que as ações realizadas por uma pessoa ou sistema podem ser rastreadas de forma a comprovar que foram efetivamente realizadas por essa pessoa ou sistema.
- e) *Autenticidade*: avalia o grau em que a identidade de uma pessoa ou recurso é efetivamente aquela que se diz ser. Por exemplo, não é desejável que um usuário possa se passar por outro ou que um recurso (conjunto de dados) que se pensa ser uma coisa na verdade seja outra.

A diferença entre confidencialidade e integridade é sutil. No primeiro caso, garante-se que quem deve ter acesso o terá; no segundo, garante-se que quem não deve ter acesso não o terá.

### 11.1.6 COMPATIBILIDADE

A *compatibilidade* avalia o grau em que dois ou mais sistemas ou componentes podem trocar informação e/ou realizar suas funções requeridas enquanto compartilham o mesmo ambiente de hardware e software. Suas subcaracterísticas são:

- a) *Coexistência*: avalia o grau no qual o produto pode desempenhar as funções requeridas eficientemente enquanto compartilha ambiente e recursos comuns com outros produtos, sem impacto negativo nos demais produtos.
- b) *Interoperabilidade*: avalia o grau no qual o software é capaz de interagir com outros sistemas com os quais se espera que ele interaja. Um software incapaz de se comunicar adequadamente com outros sistemas-chave não apresenta a qualidade de interoperabilidade.

### 11.1.7 CAPACIDADE DE MANUTENÇÃO

A *capacidade de manutenção* é uma característica interna do software percebida diretamente apenas pelos desenvolvedores, embora os clientes possam ser afetados por ela na medida do tempo gasto pelos desenvolvedores para executar atividades de manutenção ou evolução do software. A manutibilidade, portanto, mede a facilidade de se realizarem alterações no software para sua evolução ou de detectar e corrigir erros.

A capacidade de manutenção se subdivide nas seguintes subcaracterísticas:

- a) *Modularidade*: avalia o grau em que o sistema é subdividido em partes lógicas coesas, de forma que mudanças em uma dessas partes tenham impacto mínimo nas outras.
- b) *Reusabilidade*: avalia o grau em que partes do sistema podem ser usadas para construir outros sistemas.
- c) *Analysabilidade*: um sistema é analisável quando permite encontrar defeitos (depurar) facilmente quando ocorrem erros ou falhas. Por exemplo, sistemas que, quando falham, travam o computador podem ter nível de analisabilidade baixo, porque é difícil encontrar um defeito em um sistema travado. Já sistemas que, ao falharem, apresentam mensagens relacionadas com exceções internas ocorridas são mais facilmente analisáveis.
- d) *Modificabilidade*: a modificabilidade tem relação com a facilidade que o sistema oferece para que erros sejam corrigidos quando detectados, sem que as modificações introduzam novos defeitos ou degradem sua organização interna. Boas práticas de programação, arquitetura bem definida, refatoração quando necessário, aplicação de padrões de projeto e padrões de programação e testes automatizados são exemplos de disciplinas que podem colaborar para que sistemas tenham melhor modificabilidade.
- e) *Testabilidade*: a testabilidade mede a facilidade de realizar testes de regressão (Seção 13.2.6). Essa qualidade não diz tanto respeito ao software em si quanto ao processo estabelecido para permitir que ele seja testado. Porém, algumas características internas do software, como a complexidade ciclomática ou a coesão modular, podem afetar significativamente a testabilidade.

### 11.1.8 PORTABILIDADE

A *portabilidade* avalia o grau em que o software pode ser efetiva e eficientemente transferido de um ambiente de hardware ou software para outro. Suas subcaracterísticas são:

- Adaptabilidade*: avalia o quanto é fácil adaptar o software a outros ambientes sem que seja necessário aplicar ações ou meios além daqueles fornecidos com o próprio software.
- Instabilidade*: avalia a facilidade de instalar o software.
- Substituibilidade*: avalia o grau em que o sistema pode substituir outro no mesmo ambiente e com os mesmos objetivos.

### 11.1.9 QUALIDADES DO SOFTWARE EM USO

As características de qualidade do software em uso são fatores externos que só podem ser plenamente avaliados quando o software está efetivamente em seu ambiente de uso final, ou seja, é muito difícil avaliá-las em um ambiente de desenvolvimento. As características são as seguintes:

- Efetividade*: é a capacidade que o produto de software tem para fazer que o cliente atinja seus objetivos de negócio de forma correta e completa, no ambiente real de uso.
- Eficiência*: avalia o retorno que o produto dá ao cliente, ou seja, a razão entre o que o cliente investiu e investe no sistema em relação ao que recebe em troca. Essa medida nem sempre é financeira.
- Satisfação*: é a capacidade do produto de satisfazer aos usuários durante seu uso no ambiente final. Essa característica subdivide-se nas seguintes subcaracterísticas:
  - *Utilidade*: avalia o grau no qual o usuário é satisfeito com a obtenção percebida de metas pragmáticas, incluindo os resultados e as consequências do uso do software.
  - *Prazer*: avalia o grau em que o usuário sente prazer em usar o sistema para satisfazer seus objetivos.
  - *Conforto*: avalia o conforto físico e mental do usuário ao usar o sistema.
  - *Confiança*: avalia o grau em que o usuário ou outros interessados confiam que o sistema faça o que é esperado dele.
- Uso sem riscos*: é a capacidade do produto de estar dentro de níveis aceitáveis de segurança relativa a riscos envolvendo pessoas, negócios e meio ambiente. Essa característica subdivide-se em:
  - *Mitigação de risco econômico*: avalia o grau no qual o produto minimiza riscos financeiros potenciais, incluindo danos à propriedade e à reputação de pessoas.
  - *Mitigação de risco à saúde e à segurança*: avalia o grau no qual o produto minimiza riscos físicos às pessoas em seu contexto de uso.
  - *Mitigação de risco ambiental*: avalia o grau no qual o produto minimiza riscos ambientais ou à propriedade em seu contexto de uso.
- Cobertura de contexto*: avalia o grau no qual o produto ou sistema pode ser usado com efetividade, eficiência, sem riscos e com satisfação tanto no contexto inicialmente especificado quanto em outros contextos. Essa característica subdivide-se em:
  - *Completude de contexto*: avalia o grau no qual o produto ou sistema pode ser usado com efetividade, eficiência, sem riscos e com satisfação em todos os contextos de uso especificados.
  - *Flexibilidade*: avalia o grau no qual o produto ou sistema pode ser usado com efetividade, eficiência, sem riscos e com satisfação em contextos diferentes daqueles inicialmente especificados.

## 11.2 Modelo de Qualidade de Dromey

Modelos de qualidade são estruturas conceituais que definem quais são as características da qualidade e como elas se estruturam e se relacionam entre si. A Norma 9126 e sua sucessora, a 25010, são exemplos importantes de modelos de qualidade nos quais as características são decompostas hierarquicamente.

Uma das críticas a esse tipo de modelo é que não há um critério claro para decidir quais são as características de alto nível e quais são as subcaracterísticas (Kitchenham & Lawrence, 1996). Assim, Dromey (1998)<sup>7</sup>

<sup>7</sup>Disponível em: <[se.dongguk.ac.kr/metrics/SPQ-Theory.pdf](http://se.dongguk.ac.kr/metrics/SPQ-Theory.pdf)>. Acesso em: 21 jan. 2013.

desenvolveu um modelo que procura resolver esse e outros problemas apontados em relação aos modelos hierárquicos. Ele diz que é impossível construir características de qualidade de alto nível (como usabilidade) diretamente nos produtos. Em vez disso, é necessário construir componentes de software que exibam um conjunto harmonioso, completo e consistente de propriedades que resultem na manifestação dessas características de alto nível.

Assim, antes de criar uma hierarquia de características de qualidade, Dromey estabelece um método para determinar sistematicamente essas características, a partir do qual é possível avaliar se a decomposição é consistente e completa. O método se baseia nos seguintes princípios:

- a) Um *comportamento* pode ser decomposto, e assim definido, em termos de propriedades subordinadas, as quais podem ser tanto comportamentos quanto características do software.
- b) Um *uso* pode ser decomposto, e assim definido, em termos de propriedades subordinadas, que podem ser tanto usos quanto características do software.

O desafio, então, passa a ser diferenciar os comportamentos e usos subordinados hierarquicamente (ou seja, que são subtipos do comportamento ou usos de mais alto nível) das características de software, que não são subtipos, mas elementos agregados aos comportamentos ou usos.

Algumas propriedades, como modularidade e acurácia, são claramente características do software, enquanto outras, como tolerância a falhas, não são tão facilmente distinguíveis de comportamentos.

A sugestão de Dromey é que se considerem os *comportamentos* como gerais em relação ao sistema, ou seja, não sendo aplicáveis apenas a algumas de suas partes. Enquanto isso, as *características* de software podem ser aplicadas a componentes específicos.

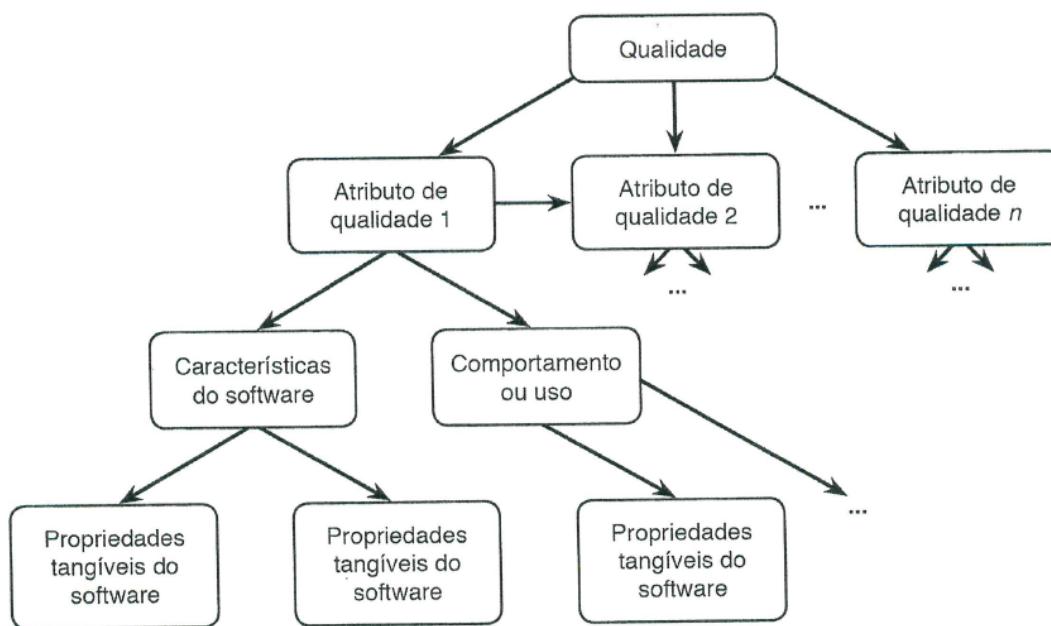
A construção de um modelo de qualidade deve ainda considerar os seguintes princípios:

- a) Propriedades abstratas, chamadas de *atributos de qualidade*, podem ser associadas ao software.
- b) A *qualidade do software* pode ser caracterizada por um conjunto de atributos de qualidade de alto nível.
- c) Os *atributos de qualidade do software* correspondem tanto a um conjunto de comportamentos de software independentes de domínio quanto a um conjunto de usos de software independentes de domínio.
- d) Os *atributos de qualidade de um modelo de qualidade* devem ser suficientes para satisfazer às necessidades de todos os grupos de interesse associados ao software.
- e) Cada *atributo de qualidade de alto nível* é caracterizado por um conjunto de propriedades subordinadas, que podem ser tanto comportamentos quanto usos ou características do software.
- f) Cada *característica* do software é determinada ou composta por um conjunto de propriedades tangíveis, que serão chamadas de *propriedades que transferem qualidade (quality carrying)*.
- g) *Propriedades que transferem qualidade* podem incorporar tanto propriedades funcionais quanto não funcionais.

A Figura 11.2 apresenta, de forma esquemática, a composição do modelo de qualidade de Dromey, que, na verdade, pode ser considerado um metamodelo, já que modelos individuais podem ser instanciados a partir dele.

Dromey apresenta dois estudos de caso de aplicação desse modelo, um para a definição de *usabilidade* e outro para a definição de *confiabilidade*. No caso de usabilidade, a título de ilustração, ele começa definindo um conjunto de propriedades (não necessariamente completo) que podem caracterizar a usabilidade:

- a) *Apreensibilidade*: facilidade de aprender como usar.
- b) *Transparência*: facilidade de entender ou lembrar como se usa uma funcionalidade.
- c) *Operabilidade*: facilidade de aplicar uma funcionalidade eficientemente.
- d) *Responsividade*: capacidade de executar todas as funções em tempo razoável.
- e) *Customizabilidade*: capacidade de personalizar interfaces para as necessidades do usuário.
- f) *Disponibilização de outros idiomas*: capacidade de mudar a língua da interface.
- g) *Comandos sensíveis ao contexto*: mostrar comandos usados no contexto.
- h) *Imediatismo operacional*: permitir a execução de comandos de forma direta.
- i) *Teclas de atalho*: permite uso de teclas de atalho para as opções mais frequentes.
- j) *Consistência*: comandos consistentes com o ambiente.



**Figura 11.2** Modelo de qualidade de Dromey.

Inicialmente, essas características são classificadas em usos, comportamentos ou características do software, resultando no seguinte:

- Apreensibilidade: uso.
- Transparência: característica do software.
- Operabilidade: uso.
- Responsividade: comportamento (métrica).
- Customizabilidade: uso.
- Disponibilização de outros idiomas: característica do software.
- Comandos sensíveis ao contexto: característica do software.
- Imediatismo operacional: característica do software.
- Teclas de atalho: característica do software.
- Consistência: característica do software.

Comportamentos e usos são candidatos a serem determinantes de usabilidade de nível mais alto, embora alguns comportamentos ou usos possam ser subordinados a outros. O processo de definição da hierarquia consiste em perguntar, para cada duas propriedades, se uma é parte ou subtipo de outra. Dessa forma, uma hierarquia como a seguinte poderia ser formada a partir das características mostradas no exemplo:

- Usabilidade
  - Apreensibilidade
    - i. Consistência
    - ii. Transparência
    - iii. Comandos sensíveis ao contexto
  - Operabilidade
    - i. Customizabilidade
      - 1. Disponibilização de outros idiomas
    - ii. Responsividade
    - iii. Imediatismo operacional
    - iv. Teclas de atalho

O modelo é também fundamentado no axioma de que, se o software é composto por componentes, então suas propriedades contextuais intrínsecas tangíveis e a forma como eles são compostos vão determinar a qualidade do

**TABELA 11.3** Propriedades de variáveis que impactam na qualidade de software

Propriedade que transfere qualidade	Característica do software	Impacto na qualidade
Convenção de nomes	Analisabilidade	Capacidade de manutenção
Atribuída	Correção	Funcionalidade e confiabilidade
Precisa	Correção	Funcionalidade e confiabilidade
Propósito único	Correção	Funcionalidade e confiabilidade
Encapsulada	Modularidade	Capacidade de manutenção e reusabilidade
Utilizada	Consistência	Capacidade de manutenção
Autodescritiva	Analisabilidade	Capacidade de manutenção e reusabilidade
Comentada	Analisabilidade	Capacidade de manutenção e reusabilidade

**TABELA 11.4** Propriedades de expressões que impactam na qualidade de software

Propriedade que transfere qualidade	Característica do software	Impacto na qualidade
Computável	Correção	Funcionalidade e confiabilidade
Livre de efeitos colaterais	Correção	Funcionalidade e confiabilidade
Efetiva	Não redundância	Eficiência
Ajustável	Parametrização	Manutenibilidade e reusabilidade

**TABELA 11.5** Propriedades de comandos que impactam na qualidade de software

Propriedade que transfere qualidade	Característica do software	Impacto na qualidade
Completo	Corretude	Funcionalidade
Efetivo	Não redundância	Eficiência

software. Ou seja, a qualidade do software começa na escolha dos nomes das variáveis, por exemplo, que são usadas para criar os métodos usados para criar as classes etc.

A Tabela 11.3 mostra algumas propriedades de variáveis que, por transferirem qualidade para certas características do software, têm impacto na qualidade. Essa tabela pode ser utilizada como um *checklist* para uma eventual inspeção de qualidade no código-fonte. Por exemplo, uma variável definida mas nunca atribuída é um potencial defeito no software que precisa ser corrigido.

Na sequência, as variáveis são compostas para criar expressões no software. Algumas propriedades de expressões podem definir qualidade e ter impacto. A Tabela 11.4 apresenta algumas qualidades que podem ser atribuídas a expressões (linhas de código) que afetam a qualidade do software.

No nível seguinte estão os comandos como atribuições e estruturas de controle, as quais usam expressões. As qualidades são identificadas na Tabela 11.5.

As comunicações com o hardware também são analisadas e algumas propriedades, indicadas na Tabela 11.6.

A análise das propriedades que transferem qualidade ligadas à conexão entre o software e a base de dados é apresentada na Tabela 11.7.

As propriedades referentes à comunicação de dados são apresentadas na Tabela 11.8.

Mais detalhes podem ser encontrados no trabalho de Dromey (1998)<sup>8</sup>.

<sup>8</sup>Disponível em: <se.dongguk.ac.kr/metrics/SPQ-Theory.pdf>. Acesso em: 21 jan. 2013.

**TABELA 11.6** Propriedades de interfaces com hardware que impactam na qualidade de software

Propriedade que transfere qualidade	Característica do software	Impacto na qualidade
Inicialização no <i>startup</i> , reinício ou recuperação de erro	Sobrevivência	Confiabilidade
Checagem periódica de <i>status</i> operacional	Sobrevivência e tolerância a falhas	Confiabilidade
Checagem periódica de <i>status</i> operacional parametrizada	Flexibilidade	Capacidade de manutenção
Relatório e resolução quando o dispositivo não está operacional	Visibilidade e sobrevivência	Funcionalidade e confiabilidade
Intervalo de valores permitidos do dispositivo é documentado	Tolerância a falhas	Confiabilidade
Todos os estados operacionais do dispositivo são resolvidos e completos	Corretude e tolerância a falhas	Funcionalidade e confiabilidade
Todas as interações para cada dispositivo de hardware estão encapsuladas e parametrizadas	Modularidade e flexibilidade	Capacidade de manutenção, reuso e portabilidade
Detalhes arquiteturais de dispositivos estão encapsulados e parametrizados	Modularidade e flexibilidade	Capacidade de manutenção, reuso e portabilidade

**TABELA 11.7** Propriedades de interfaces com a base de dados que impactam na qualidade de software

Propriedade que transfere qualidade	Característica do software	Impacto na qualidade
Chamadas à base de dados não devem depender de conhecimento sobre seu esquema de gerenciamento	Independência de aplicação	Capacidade de manutenção, portabilidade e reusabilidade
Chamadas à base de dados devem ser processadas fora da unidade que faz a chamada	Independência de aplicação	Capacidade de manutenção, portabilidade e reusabilidade
A aplicação deve ser independente de detalhes da estrutura da base de dados	Independência de representação e independência de sistema	Capacidade de manutenção, portabilidade e reusabilidade
O <i>design</i> de gerenciamento da base de dados deve fornecer uma forma comum e controlada para adicionar, recuperar e alterar dados	Consistência, interoperabilidade e segurança	Capacidade de manutenção, reusabilidade e funcionalidade
Itens da base de dados não devem ser referenciados por mais de um nome	Consistência e eficiência de armazenamento	Capacidade de manutenção, eficiência e confiabilidade
Modificações na base de dados devem ser feitas por transações atômicas, consistentes, íntegras e duráveis	Tolerância a falhas e consistência	Confiabilidade e capacidade de manutenção
A integridade da base de dados deve ser recuperada a partir de condições de erro	Tolerância a falhas	Confiabilidade
O acesso à base de dados é controlado	Acessibilidade e segurança	Funcionalidade
O <i>design</i> do sistema o protege contra acesso não autorizado	Acessibilidade e segurança	Funcionalidade

**TABELA 11.8** Propriedades de comunicação de dados que impactam na qualidade de software

Propriedade que transfere qualidade	Característica do software	Impacto na qualidade
Mensagens contêm rótulos indicando o tipo de dados	Interoperabilidade	Funcionalidade
Intervalos válidos para todos os dados nas mensagens são indicados	Tolerância a falhas e analisabilidade	Confiabilidade e capacidade de manutenção
Propósito e formato de cada item de dados nas mensagens são definidos	Analisabilidade	Capacidade de manutenção
Glossário técnico para rótulos de mensagens é fornecido	Analisabilidade	Capacidade de manutenção
Formato comum especificado é usado para posicionamento, empacotamento de dados e transmissão de blocos	Interoperabilidade e analisabilidade	Capacidade de manutenção e funcionalidade
Representações de dados em mensagens atendem a padrões especificados em contrato	Interoperabilidade e analisabilidade	Capacidade de manutenção e funcionalidade



### 11.3 Instalação de um Programa de Melhoria de Qualidade

Para que a gestão da qualidade seja eficaz, deve existir um *programa de melhoria de qualidade* definido na empresa. A instalação inicial do programa implica estabelecer uma anistia geral na empresa (Belchior, 1997), ou seja, não se buscam culpados para o que aconteceu até o momento. Busca-se promover uma melhoria geral conjunta.

Para que o programa de melhoria de qualidade funcione, é necessário que ele seja acordado e conhecido por todos os envolvidos e, também, que se torne parte da cultura da empresa. Planos apenas colocados no papel que são abandonados na primeira dificuldade logo são esquecidos.

Assim, os planos devem ser consistentes, factíveis, gradualmente implementados e, principalmente, levados a sério por todos. Planos inconsistentes, impossíveis de executar e que caem do céu prontos e acabados dificilmente são levados a sério.

Deming (*apud* Vasconcelos, Rouiller, Machado & Medeiros, 2006)<sup>9</sup> estabelece 14 princípios fundamentais para o sucesso de um programa de melhoria de qualidade, entre os quais destacam-se os seguintes:

- a) Melhorar constantemente o sistema de produção e os serviços de forma a maximizar o binômio qualidade/produtividade.
- b) Institucionalizar os novos métodos de treinamento no trabalho.
- c) Institucionalizar e fortalecer os papéis de liderança.
- d) Eliminar os medos.
- e) Quebrar as barreiras entre departamentos.
- f) Eliminar *slogans*, exortações e metas de produtividade, pois isso pode levar à queda na qualidade.
- g) Eliminar cotas-padrão arbitrárias e gerenciamento por objetivos.
- h) Institucionalizar um vigoroso programa de educação e automelhoria.
- i) Colocar todos para trabalhar pela modificação em prol da qualidade.

A Seção 12.5 apresenta, com detalhes, um modelo de implantação e melhoria contínua de processos visando à qualidade.

### 11.4 Gestão da Qualidade

A gestão da qualidade do produto de software pode ser considerada uma atividade de gerenciamento que pode ser efetuada pelo gerente de projeto, mas preferencialmente deveria ser realizada por um gerente ou equipe especializados. Consiste no planejamento e na execução das ações necessárias para que o produto satisfaça aos requisitos de qualidade estabelecidos (Seção 11.6).

Crosby (1979) define um modelo de maturidade organizacional em relação à qualidade baseado em cinco estágios:

- a) *Desconhecimento*: quando a empresa não sabe sequer que tem problemas com qualidade. Não há compreensão de que a qualidade seja um objetivo ou processo de negócio, ferramentas não são usadas ou conhecidas, e inspeções de qualidade não são realizadas.
- b) *Despertar*: a empresa reconhece que tem problemas com a qualidade e que precisa começar a lidar com eles, mas ainda vê isso como um mal necessário, não como fonte de lucro.
- c) *Alinhamento*: o gerenciamento da qualidade se torna uma ferramenta institucional e os problemas vão sendo priorizados e resolvidos à medida que surgem.
- d) *Sabedoria*: a prevenção de problemas, e não apenas sua correção, torna-se rotina na empresa. Problemas são identificados antes que surjam, e todos os processos e rotinas estão abertos a mudanças visando à melhoria da qualidade.
- e) *Certeza*: a gestão da qualidade é uma constante e parte essencial do funcionamento da empresa. Quase todos os problemas são prevenidos e eliminados antes de surgirem.

<sup>9</sup>Disponível em: <[www.cin.ufpe.br/~if720/downloads/Mod.01.MPS\\_Engenharia%26QualidadeSoftware\\_V.28.09.06.pdf](http://www.cin.ufpe.br/~if720/downloads/Mod.01.MPS_Engenharia%26QualidadeSoftware_V.28.09.06.pdf)>. Acesso em: 21 jan. 2013.

Segundo Crosby, o custo relativo da qualidade vai diminuindo gradativamente à medida que sobem os níveis de maturidade.

Duas técnicas de controle de qualidade conhecidas (Belchior, 1997) são o *walkthrough* e as *inspeções*. Ambas baseiam-se em um processo de verificação sistemática e cuidadosa dos produtos do trabalho por terceiros para detectar defeitos. Outra técnica de garantia de qualidade é o teste sistemático de software, abordado em detalhes no Capítulo 13.

Além das técnicas de inspeção e teste, que podem ser aplicadas com qualquer ciclo de vida, pode-se mencionar também o modelo *cleanroom*. Trata-se de um ciclo de vida à parte que visa, a partir de especificações formais, produzir produtos de software isentos de defeitos desde sua origem, não necessitando de testes de unidade e integração. As subseções seguintes vão apresentar mais alguns detalhes sobre as técnicas *walkthrough* e de *inspeção*, bem como uma breve introdução ao modelo *cleanroom*.

#### 11.4.1 WALKTHROUGH

O *walkthrough* (Yourdon, 1985) é uma forma de avaliação do produto que utiliza uma equipe de especialistas na qual cada um faz uma análise prévia do produto. Depois, três a cinco pessoas reúnem-se por cerca de duas horas para trocar impressões sobre o produto e sugerir melhorias.

Além dos analistas, a reunião de *walkthrough* deve contar preferencialmente com desenvolvedores e usuários, que poderão apresentar rapidamente respostas a eventuais dúvidas, como “este requisito devia ter sido implementado dessa forma mesmo?”.

Ao término da reunião, os participantes votam pela *aceitação do produto*, pela *aceitação com modificações parciais* ou pela *rejeição*. Sempre que houver modificações recomendadas, o produto deverá passar por um novo *walkthrough*.

Os papéis em uma reunião *walkthrough* são os seguintes (entre outros)<sup>10</sup>:

- a) *Apresentador*: geralmente é o autor do artefato, que o descreve, bem como as razões para ele ser dessa forma. Antes da reunião, ele entrega as especificações do artefato ao coordenador, que as distribui à equipe.
- b) *Coordenador*: é o moderador da reunião. Seu trabalho é manter todos focados nas tarefas e não se envolver em discussões. O ideal é que esse papel seja executado por alguém de fora da equipe.
- c) *Secretário*: é o responsável por tomar nota das discussões e decisões. Suas notas deverão ser aprovadas pelos participantes ao final da reunião.
- d) *Oráculo de manutenção*: é o inspetor de garantia de qualidade cujo trabalho é certificar-se de que o código produzido seja compreensível e manutenível, de acordo com os padrões da empresa.
- e) *Guardião dos padrões*: seu trabalho é certificar-se de que o código produzido esteja de acordo com os padrões de programação estabelecidos previamente pela equipe. Se não houver padrões estabelecidos, possivelmente muito tempo da reunião será perdido com discussões irrelevantes.
- f) *Representante do usuário*: pode estar presente em algumas reuniões, especialmente naquelas que discutem requisitos, para garantir que o cliente realmente receba o produto que espera.
- g) Outros desenvolvedores poderão participar e contribuir com a discussão apresentando outros pontos de vista.

É importante mencionar que a reunião deve seguir estritamente o planejamento inicial, mantendo a discussão produtiva e objetiva. O objetivo principal é avaliar os defeitos, e não os desenvolvedores. Além disso, a reunião não será usada para corrigir defeitos, apenas encontrá-los. O processo de reparação vai ocorrer depois.

Erros triviais, como erros ortográficos em janelas, não necessitam de discussão. Apenas os erros mais graves. Em relação à psicologia das reuniões, os seguintes perfis devem ser observados:

- a) *Programadores gênios*: especialmente se forem aqueles gênios arrogantes, impacientes e de mente estreita, podem causar problemas. Devem ser valorizados, pois são capazes de detectar defeitos com facilidade (e alimentam seu ego com isso). O coordenador da sessão deve ter humildade e controle para não iniciar discussões com eles, nem deixar que outros o façam, pois isso tornará o trabalho improdutivo.

- b) *Pessoas defensivas e inseguras*: deve-se ter cuidado com essas pessoas, pois frequentemente se sentirão atingidas pessoalmente pelas críticas feitas ao seu código. Também é preciso tomar muito cuidado para que seja mantida a discussão sobre o produto, e não sobre os programadores. A reunião de *walkthrough* não é o momento para tentar resolver a vida deles.
- c) *Conservadores*: também poderão causar problemas algumas vezes, pois buscam se manter fiéis às tradições estabelecidas. Deve-se dar atenção às suas opiniões, porque a área de programação é muito sujeita a modismos, mas deve-se também procurar evitar que iniciem discussões improdutivas.
- d) *Alienados*: não estão interessados no mundo real e primam mais pelo processo do que pelo produto, podendo tornar-se um incômodo sério. O coordenador sempre deve ter em mente que o processo só é útil quando ajuda a produzir da melhor forma possível. O processo não é uma religião que se deva seguir cegamente. As regras existem porque há objetivos a alcançar, e não porque foram ditadas por alguma divindade da computação, mas, muitas vezes, os alienados não percebem isso.

Enfim, muitos problemas interpessoais poderão surgir nas primeiras reuniões. Por isso, é necessário que o coordenador seja experimentado e competente na condução das reuniões para que, com o tempo, a equipe aprenda a se manter estritamente focada em seu objetivo, que é a *detecção de defeitos nos programas*.

#### 11.4.2 INSPEÇÕES FAGAN

As *inspeções Fagan* (Fagan, 1976)<sup>11</sup> consistem em um processo estruturado para tentar encontrar defeitos no código, diagramas ou especificações. Uma inspeção Fagan parte do princípio de que toda atividade que tenha critérios de entrada e saída bem definidos pode ser avaliada de forma a verificar se efetivamente produz a saída especificada.

Como as atividades de processos de software (Seção 2.3) devem ser sempre definidas em termos de artefatos de entrada e saída, elas se prestam bem a serem avaliadas por inspeções Fagan.

Assim, os artefatos de entrada e saída equivalem aos critérios de entrada e saída para as inspeções, e qualquer desvio encontrado nos artefatos de saída é considerado um *defeito*. Defeitos podem ser classificados em diferentes tipos, como *graves* e *triviais*. Um *defeito grave* é caracterizado pelo não funcionamento do produto, como uma função faltando, por exemplo. Um *defeito trivial* é uma característica errada que não afeta a capacidade de funcionamento do software, como um erro ortográfico em uma janela de sistema, por exemplo.

Normalmente, os papéis na equipe de inspeção são os seguintes:

- a) *Autor*: o programador, *designer* ou analista, ou seja, a pessoa que produziu o artefato.
- b) *Narrador*: analisa, interpreta, sumariza o artefato e seus critérios de aceitação.
- c) *Revisores*: revisam o artefato com o objetivo de detectar eventuais defeitos.
- d) *Moderador*: é o responsável pela sessão de inspeção e pelo andamento do processo.

O processo de inspeção, tipicamente, abrange as seguintes atividades (Fagan, 1986)<sup>12</sup>:

- a) *Planejamento*: inclui preparação dos materiais (artefatos), convite aos participantes e alocação do espaço de trabalho.
- b) *Visão geral*: inclui instrução prévia (apresentação) aos participantes sobre os materiais a serem inspecionados e a atribuição de papéis a esses participantes.
- c) *Preparação*: nessa atividade, os participantes analisam os artefatos sob inspeção e o material de suporte, anotando possíveis defeitos e questões para a reunião de inspeção.
- d) *Reunião de inspeção*: é o momento em que efetivamente se discutem os defeitos encontrados e se decide o que fazer com eles.
- e) *Retrabalho*: é a atividade em que o autor do artefato vai corrigir os defeitos apontados na reunião de inspeção.
- f) *Prosseguimento (follow-up)*: essa atividade considera que todos os defeitos foram corrigidos e o produto está aprovado para prosseguir para a fase seguinte, ou entrega.

<sup>11</sup>Disponível em: <[www.mfagan.com/pdfs/ibmfagan.pdf](http://www.mfagan.com/pdfs/ibmfagan.pdf)>. Acesso em: 21 jan. 2013.

<sup>12</sup>Disponível em: <[www.mfagan.com/pdfs/aisi1986.pdf](http://www.mfagan.com/pdfs/aisi1986.pdf)>. Acesso em: 21 jan. 2013.

É responsabilidade do moderador da inspeção verificar se todos os defeitos foram corrigidos e se o produto pode ir para *follow-up*. Caso ele considere que os defeitos não foram adequadamente corrigidos ou que novos defeitos foram introduzidos no processo de correção, deverá determinar que o produto retorne ao processo de inspeção.

Defeitos triviais podem simplesmente ir para retrabalho sem que sejam necessárias novas inspeções. Contudo, os defeitos graves devem ser novamente analisados pelo processo de inspeção, que deve ser reiniciado na sua primeira fase (planejamento).

As reuniões são importantes, pois produzem sinergia no grupo, incluindo a necessária troca de experiências e a afinação de discurso, o que funciona também como atividade de formação continuada de inspetores. Porém, em função dos custos com deslocamento de pessoas para reuniões desse tipo, cada vez mais têm sido utilizados meios eletrônicos para que elas possam acontecer de forma virtual. O mesmo ocorre com o material, disponibilizado eletronicamente para minimizar também o uso de papel (Genuchten, Cornelissen & Dijk, 1997).

#### 11.4.3 MÉTODO CLEANROOM

O método *cleanroom* (Mills, Dyer & Linger, 1987)<sup>13</sup> é uma maneira bastante formal de desenvolver software com foco na qualidade, que procura evitar que defeitos sejam introduzidos durante o desenvolvimento. O nome foi inspirado nas salas de fábricas de circuitos integrados (“salas limpas”). O método estabelece que um circuito não precisa ser limpo depois de fabricado, porque já é produzido em um ambiente sem sujeira. Da mesma forma, espera-se que um software produzido em um ambiente *limpo* possa ser isento de defeitos.

O principal objetivo do método é produzir software que apresente taxa zero de defeitos durante seu uso (Linger & Trammell, 1996)<sup>14</sup>.

Os princípios básicos do método *cleanroom* são os seguintes:

- a) *Desenvolvimento baseado em métodos formais*: usam-se estruturas de *caixas* (ver a seguir) para especificar o software. Uma equipe de revisão verifica se o produto satisfaz à especificação.
- b) *Desenvolvimento incremental sob controle estatístico de qualidade*: utiliza-se desenvolvimento baseado em ciclos iterativos, nos quais funcionalidades vão sendo agregadas gradualmente, e o controle estatístico é feito por uma equipe independente, cuja função é garantir que a qualidade permaneça em nível aceitável.
- c) *Programação estruturada*: apenas um conjunto limitado de estruturas de abstração é permitido. O desenvolvimento de programas é um processo de refinamento passo a passo que utiliza essas estruturas e transformações que garantem a preservação da corretude em relação à especificação, até chegar ao código.
- d) *Verificação estática*: o software desenvolvido é verificado estaticamente, usando inspeções de código rigorosas. Não há teste de unidade nem de integração.
- e) *Testes baseados em medição estatística*: um conjunto de testes baseados em especificações formais e estatisticamente representativo é selecionado e aplicado. Os testes de sistema são realizados diretamente.

O método *cleanroom* é definido em termos de 14 processos e 20 artefatos (produtos do trabalho), definidos por Linger e Trammell (1996).

O princípio básico do método é que programas podem ser vistos como regras para funções ou relações matemáticas. Um programa faz transformações em uma entrada, produzindo uma saída, o que pode ser especificado por uma função de mapeamento. Assim, programas podem ser projetados como decomposições de suas especificações funcionais e verificados formalmente.

As estruturas de caixas são três estruturas matemáticas usadas pelo método, as quais mapeiam *estímulos* (entradas) e *históricos* de estímulos (entradas prévias) em *respostas* (saídas):

- a) *Black box*: define o comportamento esperado de uma função do sistema em todos os seus contextos de uso: *estímulo corrente + histórico de estímulos → resposta*.
- b) *State box*: pode ser derivada de uma *black box* e define o histórico de estímulos como um estado: *estímulo corrente + estado atual → resposta + novo estado*.

<sup>13</sup>Disponível em: <[ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=1695817](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1695817)>. Acesso em: 21 jan. 2013.

- c) *Clear box*: é idêntica a uma *state box*, mas sua especificação é feita por procedimentos em linguagem de programação, ou seja, é a realização de uma *state box*.

Em orientação a objetos, a *black box* pode ser entendida como o comportamento especificado para um objeto, a *state box* como o encapsulamento de dados do objeto, e a *clear box* como seus métodos.

Um dos maiores problemas para a adoção desse método é a dificuldade de se obterem desenvolvedores suficientemente preparados em lógica e matemática para atuar com especificação formal de sistemas<sup>15</sup>.

## 11.5 Medição da Qualidade

A Seção 9.5 já apresentou algumas questões referentes a métricas e medições do ponto de vista do processo de gerência de projetos de software. Aqui será aprofundado o aspecto de medição da qualidade do produto de software.

Kitshenham e Lawrence (1996) indicam que a qualidade de um produto, do ponto de vista da satisfação do usuário, será resultado de três fatores:

- Funcionalidade*, cuja medida será estar *presente* ou *ausente*.
- Comportamento*, isto é, as qualidades não funcionais, que normalmente são mensuráveis em um dado *intervalo*.
- Restrições*, que determinam como o usuário pode usar o produto.

Quando os usuários pensam em qualidade de software, em geral lembram-se da característica de confiabilidade (Seção 11.1.2), isto é, do tempo em que o produto funciona sem apresentar defeitos. Porém, caso o software já seja relativamente confiável, outras qualidades entrarão com mais ênfase nas expectativas do usuário, como usabilidade e eficiência.

Gilb (1987) sugere que essas características podem ser medidas de forma objetiva. Por exemplo, o tempo de aprendizagem de um sistema pode ser medido como o tempo médio que os usuários levam para aprender a executar um conjunto predeterminado de tarefas.

A técnica de Gilb pode ser generalizada para outras características. A ideia é quebrar a característica de qualidade em outras menores até que se encontrem aquelas que possuam um *procedimento operacional objetivo* para serem avaliadas.

Do ponto de vista do desenvolvedor, a qualidade pode ser medida a partir de duas variáveis principais: a quantidade de defeitos e os custos com retrabalho ao longo do desenvolvimento.

A contagem de defeitos deve ser sempre relacionada com o momento em que os defeitos são introduzidos e, principalmente, encontrados. Por exemplo, encontrar um defeito durante os testes de unidade ou integração não é tão sério quanto encontrar um defeito em um produto já instalado no cliente.

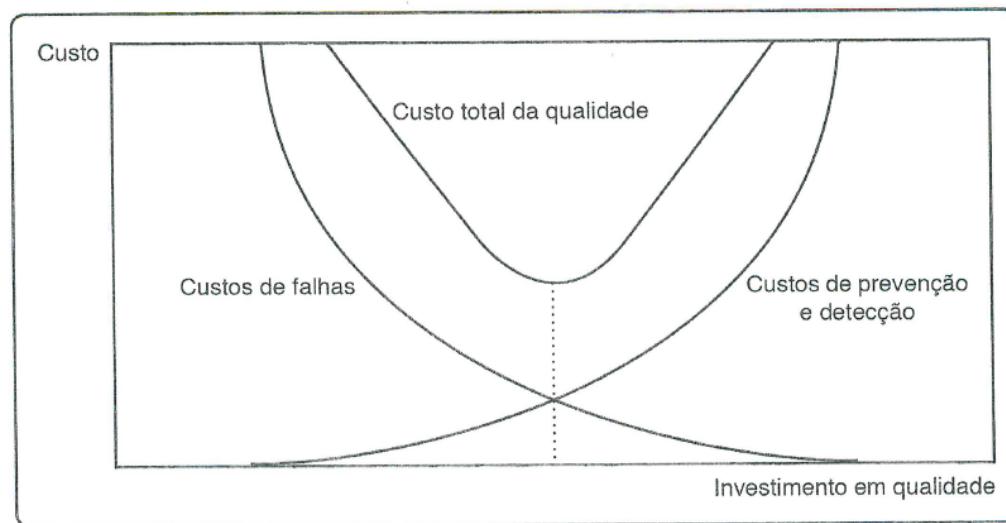
A contagem de defeitos nas diferentes fases poderá dar uma boa medida da eficácia dos processos da empresa, bem como permitir a avaliação de mudanças nesses mesmos processos ou nas ferramentas de desenvolvimento. Se a quantidade de defeitos diminuir ou se os defeitos começarem a ser identificados mais cedo, isso ocorrerá porque a mudança de processo foi salutar em relação a essa métrica.

O número de defeitos em um sistema não tem uma relação necessariamente linear com os custos de retrabalho. Por vezes, defeitos são muito simples de depurar e corrigir. Outras vezes, um único defeito pode ter um impacto catastrófico no projeto, exigindo grandes mudanças estruturais e consumo de tempo e recursos para sua correção (como o caso do *bug* do ano 2000<sup>16</sup>). A contagem de tempo com retrabalho é de difícil implementação, mas uma opção é definir “retrabalho” como um dos tipos de ação nas folhas de tempo (Seção 9.4.1). Assim, pode-se verificar quanto esforço efetivamente foi empregado refazendo o que já havia sido aprovado.

Poderá ser útil distinguir o retrabalho causado por defeitos do software, o causado por erros nos requisitos ou, ainda, o causado pela necessidade de aprimorar aspectos não funcionais do software, como sua eficiência. Normalmente, apenas o retrabalho causado por defeito ou erro nos requisitos será efetivamente contabilizado como uma atividade não produtiva, pois a melhoria ou o aprimoramento de outras qualidades do software será um investimento.

<sup>15</sup>Disponível em: <[www.cs.st-andrews.ac.uk/~ifs/Books/SE9/Web/Cleanroom/experience.htm](http://www.cs.st-andrews.ac.uk/~ifs/Books/SE9/Web/Cleanroom/experience.htm)>. Acesso em: 21 jan. 2013.

<sup>16</sup>Disponível em: <[en.wikipedia.org/wiki/Year\\_2000\\_problem](http://en.wikipedia.org/wiki/Year_2000_problem)>. Acesso em: 21 jan. 2013.



**Figura 11.3** Relação entre investimento em qualidade e economia relacionada a falhas<sup>18</sup>.

## 11.6 Requisitos de Qualidade

Os requisitos de qualidade de software podem ser catalogados, mas cada produto terá um conjunto de requisitos diferente, pois qualidade também tem custo. Algumas subcaracterísticas de qualidade são sempre desejáveis e podem ser obtidas a partir de um bom processo de desenvolvimento, como um software livre de defeitos, por exemplo. Contudo, outras qualidades (como portabilidade, por exemplo) poderão ser eletivas e o custo de sua inclusão no software poderá não ser justificável. Belchior (1997)<sup>17</sup> observa que qualidade não é sinônimo de perfeição, mas algo factível, relativo, dinâmico e evolutivo que se amolda aos objetivos a serem atingidos.

Os requisitos de qualidade devem fazer parte da própria especificação do produto. Normalmente são requisitos suplementares, ou seja, definidos para o software como um todo, e não para uma função individual. Mas também podem ser requisitos não funcionais quando se aplicam a uma ou a poucas funções.

Como os requisitos de qualidade são suplementares ou não funcionais, é de se esperar que possam ser classificados em diferentes graus de obrigatoriedade. Pode-se usar aqui o padrão MOSCOW (*Must*, *Should*, *Could* e *Would*) para determinar o grau de necessidade que determinado requisito de qualidade seja cumprido. Kerzner (1998) indica que existe um ponto ótimo para o investimento em qualidade que baixa os custos com falhas o suficiente para compensar o investimento (Figura 11.3).

Se, de um lado, as medidas de qualidade mais fundamentais (como software livre de defeitos) podem ficar subentendidas, as medidas de qualidade eletivas (como portabilidade) somente serão incorporadas ao produto se forem explicitamente solicitadas nos requisitos.

O ideal é que cada requisito de qualidade seja definido por uma especificação objetiva ou, melhor ainda, uma métrica que possa ser usada para medir o produto final e confirmar se atende ou não ao requisito.

Por exemplo, se o requisito de qualidade for “O software deve ser fácil de usar”, como avaliar se o produto final atende a essa especificação? Esse requisito está colocado de maneira subjetiva, ou seja, duas pessoas poderão ter opiniões diferentes sobre o fato de determinado produto de software ser ou não fácil de usar. Dessa forma, não há como avaliar se o requisito foi atendido, mas o problema, nesse caso, é que o requisito em si não é objetivo.

Melhor seria estabelecer um requisito como “Todas as janelas de sistema devem ter acesso a uma tela de ajuda acessível por F1”. Dessa forma, o produto final pode ser inspecionado e o requisito de qualidade, conforme especificado, pode ser avaliado como satisfeito ou não.

<sup>17</sup>Disponível em: <[www.boente.eti.br/fuzzy/tese-fuzzy-belchior.pdf](http://www.boente.eti.br/fuzzy/tese-fuzzy-belchior.pdf)>. Acesso em: 21 jan. 2013.

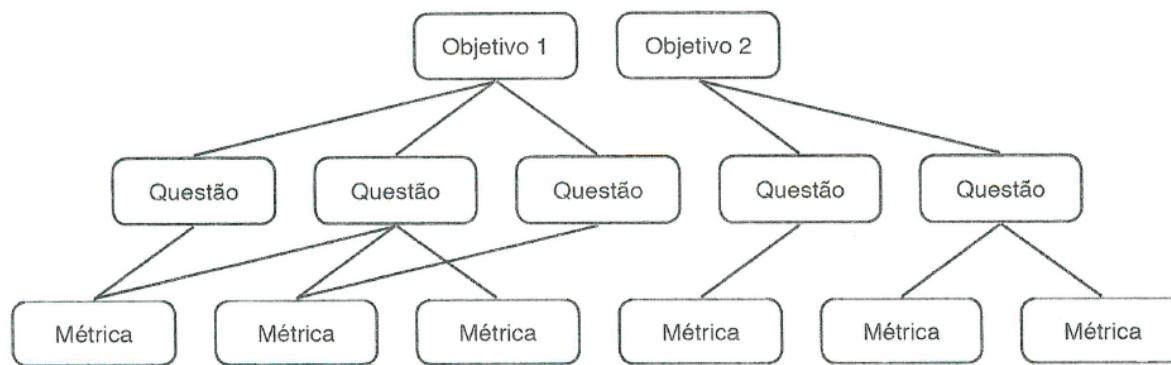


Figura 11.4 Estrutura hierárquica do modelo GQM.

### 11.7 GQM (Goal/Question/Metric) e Avaliação da Qualidade

O método GQM (Basili, Caldiera & Rombach, 1994)<sup>19</sup>, um acrônimo para *Goal/Question/Metric*, é uma abordagem para a avaliação de qualidade de software. O GQM define um modelo de mensuração em três níveis:

- Nível conceitual (Goal/objetivo)*: um objetivo é definido para um objeto por uma variedade de razões, com respeito a vários modelos de qualidade, a partir de vários pontos de vista e relativo a um ambiente em particular (os *objetos* a serem medidos podem ser produtos, processos ou recursos).
- Nível operacional (Question/questão)*: um conjunto de questões é usado para definir modelos de objetos de estudo e, então, focar no objeto que caracteriza a avaliação ou a obtenção de um objetivo específico.
- Nível quantitativo (Metric/métrica)*: um conjunto de dados baseados nos modelos é associado a cada questão para respondê-la de forma quantitativa (os dados podem ser *objetivos*, se dependerem apenas do objeto avaliado, ou *subjetivos*, se dependerem de uma interpretação do avaliador).

Um guia completo de aplicação de GQM também pode ser consultado na internet (Solingen & Berghout, 1999)<sup>20</sup>. Outra publicação com exemplos práticos de aplicação de GQM é o trabalho de Wangenheim (2000).

Em GQM, a definição do processo de avaliação é feita de forma *top-down*, ou seja, dos objetivos para as métricas, enquanto a interpretação dos resultados é feita de forma *bottom-up*, ou seja, das métricas para os objetivos. A Figura 11.4 apresenta o modelo GQM esquematicamente. Observe que as métricas não são exclusivas para as questões.

Santos e Pretz (2009)<sup>21</sup> apresentam um estudo de caso em que GQM é usado para avaliar um projeto de desenvolvimento de software. Cada risco importante do sistema é analisado como um objetivo, para o qual são definidas questões e métricas. Inicialmente, os autores associam os riscos identificados às características e subcaracterísticas de qualidade da Norma 9126 (Tabela 11.9), na época ainda em vigência.

Assim, para cada risco identificado e possivelmente para cada subcaracterística de qualidade associada ao risco, um objetivo é estabelecido. Para cada objetivo, uma ou mais questões são colocadas, e, para cada questão, uma ou mais métricas são definidas (Tabela 11.10).

Como se pode ver na tabela, o *objetivo* é especificado de acordo com um padrão estabelecido pelo próprio GQM, que sugere que objetivos sejam estabelecidos a partir de diferentes dimensões:

- Propósito*: verbo que representa o objetivo, como “avaliar”.
- Questão*: adjetivo referente ao objeto, como “a maturidade de”.
- Objeto*: objeto em avaliação, como “o software”.
- Ponto de vista*: para quem a avaliação é feita, como “do ponto de vista do cliente”.

<sup>19</sup>Disponível em: <<ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf>>. Acesso em: 21 jan. 2013.

<sup>20</sup>Disponível em: <[www.iteva.rug.nl/gqm/GQM%20Guide%20non%20printable.pdf](http://www.iteva.rug.nl/gqm/GQM%20Guide%20non%20printable.pdf)>. Acesso em: 21 jan. 2013.

<sup>21</sup>Disponível em: <[tconline.feevale.br/tc/files/6163\\_38.pdf](http://tconline.feevale.br/tc/files/6163_38.pdf)>. Acesso em: 21 jan. 2013.

**TABELA 11.9** Associação de riscos às características e subcaracterísticas de qualidade<sup>22</sup>

Sistema Exemplo		NBR ISO/IEC 9126	
Requisito	Risco	Característica	Subcaracterística
Envio e recepção de nova versão à base centralizada	R001 – Indisponibilidade do sistema para o usuário.	Confiabilidade	Tolerância a falhas
	R002 – Insuficiência dos recursos envolvidos com a produção do sistema, causando indisponibilidade.		Maturidade
	R003 – Interceptação de informações sigilosas no tráfego de rede utilizado pelo sistema.		Recuperabilidade
Envio e recepção de informações dos sistemas relacionados	R004 – Acesso liberado dos usuários da aplicação às informações inseridas no banco local, instalado na estação de trabalho do usuário.	Eficiência	Utilização de recursos
			Segurança de acesso
Funcionalidade		Funcionalidade	

**TABELA 11.10** Exemplo de aplicação do modelo GQM2<sup>23</sup>

Risco R001 – Indisponibilidade do sistema para o usuário					
Característica	Subcaracterística	Objetivo	Questão	Métrica	
Confiabilidade	Maturidade	Avaliar a capacidade de prevenção de falhas do sistema do ponto de vista do usuário.	Quantas falhas foram detectadas durante um período definido de experimentação?	Número de falhas detectadas/número de casos de testes	
		Avaliar a disponibilidade do sistema do ponto de vista do usuário.	Quantos padrões de defeitos são mantidos sob controle para evitar falhas críticas e sérias?	Número de ocorrências de falhas sérias e críticas evitadas conforme os casos de testes de indução de falhas/número de casos de testes de indução de falhas executados	
			Quão disponível é o sistema para uso durante um período de tempo específico?	Tempo de operação/(tempo de operação + tempo de reparo) Total de casos em que o sistema estava disponível e foi utilizado com sucesso pelo usuário / número total de casos em que o usuário tentou usar o software durante um período de tempo	
	Tolerância a falhas e recuperabilidade		Qual é o tempo médio em que o sistema fica indisponível quando uma falha ocorre, antes da inicialização?	Tempo ocioso total (indisponível)/número de quedas do sistema	
			Qual o tempo médio que o sistema leva para completar a recuperação desde o início?	Soma de todos os tempos de recuperação do sistema inativo em cada oportunidade/número total de casos em que o sistema entrou em recuperação	

**TABELA 11.10** Exemplo de aplicação do modelo GQM22 (*Continuação*)

**Risco R002 – Insuficiência dos recursos envolvidos com a produção do sistema, causando indisponibilidade**

Característica	Subcaracterística	Objetivo	Questão	Métrica
Eficiência	Utilização de recursos	Avaliar a eficiência na utilização de recursos de produção do ponto de vista do usuário.	Qual é o limite absoluto de transmissões necessárias para cumprir uma função?	Número máximo de mensagens de erro e falhas relacionadas à transmissão do primeiro ao último item avaliado/máximo requerido de mensagens de erro e falhas relacionadas à transmissão
			O sistema é capaz de desempenhar tarefas dentro da capacidade de transmissão esperada?	Capacidade de transmissão / capacidade de transmissão específica projetada para ser usada pelo software durante sua execução

**Risco R003 – Interceptação de informações sigilosas no tráfego de rede utilizado pelo sistema**

Característica	Subcaracterística	Objetivo	Questão	Métrica
Funcionalidade	Segurança de acesso	Avaliar a integridade dos dados do sistema do ponto de vista do usuário.	Qual é a frequência de eventos de corrupção de dados?	Número de vezes que o maior evento de corrupção de dados ocorreu/número de casos de testes executados que causaram eventos de corrupção de dados (número de vezes que o menor evento de corrupção de dados ocorreu/número de casos de testes executados que causaram eventos de corrupção de dados)

**Risco R004 – Acesso liberado dos usuários da aplicação às informações inseridas no banco local, instalado na estação de trabalho do usuário**

Característica	Subcaracterística	Objetivo	Questão	Métrica
Funcionalidade	Segurança de acesso	Avaliar o controle de acesso ao sistema do ponto de vista do usuário.	Quão completa é a trilha de auditoria sobre o acesso do usuário ao sistema e dados?	Número de acessos do usuário ao sistema e dados gravados no log de acesso/número de acessos do usuário ao sistema e dados realizados durante a avaliação
			Quão controlável é o acesso ao sistema?	Número (tipos diferentes) de operações ilegais detectadas/ número (tipos diferentes) de operações ilegais especificadas

Os autores acrescentam, ainda, as técnicas de teste, que permitirão avaliar a questão de acordo com a métrica definida. Essa informação foi omitida na tabela, mas pode ser consultada em Santos e Pretz (2009).