11 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, o capítulo 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), usando 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, usualmente 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 das dimensões. Pressman (2005) define dois tipos de qualidade para o produto de software:

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

Em relação à qualidade, o SWEBOK[138] 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[139] 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 ela se aplica a quaisquer atores envolvidos 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*)[140].

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.

[138] www.computer.org/portal/web/swebok
[139] www.abntcatalogo.com.br/norma.aspx?ID=2815
[140] Disponível para aquisição em: www.iso.org/iso/catalogue_detail.htm?csnumber=35683.

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 Cicle Processes*[141] (Tabela 11-1).

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 Processo de *Design* Arquitetural Processo de Implementação Processo de Integração Processo de Verificação Processo de Transição Processo de Validação	Desenvolvimento do Produto
Processo de Operação Processo de Manutenção	Uso do Produto
Processo de Aposentadoria	-

O lado esquerdo da figura apresenta as fases originais da norma 15288 e o lado direito o agrupamento destas 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:

a) *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.
b) *Medidas de qualidade internas*. Avaliam aspectos internos da qualidade do software que normalmente só são percebidos pelos desenvolvedores (por exemplo, facilidade de manutenção).
c) *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).
d) *Medidas de qualidade do software em uso*. Avaliam aspectos do qualidade do software em seu ambiente final que só podem ser medidos pelos usuários finais (por exemplo, satisfação dos usuários).

As *qualidades internas*, assim, 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, então, nem sempre são importantes para o usuário final, pois este não as percebe diretamente, mas pode ser afetado indiretamente por elas (no tempo de manutenção ou evolução do software, por exemplo).

[141] www.iso.org/iso/catalogue_detail?csnumber=43564

Defende-se que a qualidade de processo influencia a qualidade interna, que por sua vez influencia a qualidade externa, que por sua vez influencia a qualidade de uso do software (Figura 11-1).

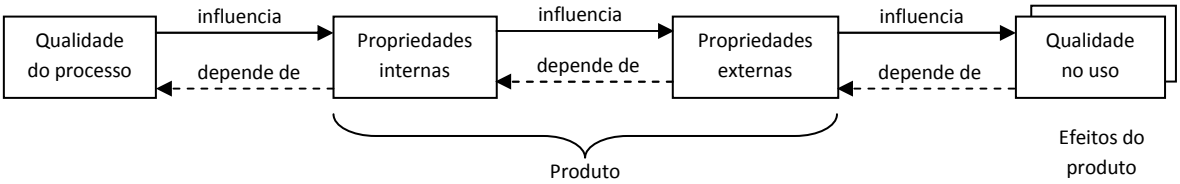

Figura 11-1: Abordagem conceitual para qualidade de acordo com a ISO/IEC 25010:2011.

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

O modelo SQuaRE (Suryn & Abran, 2003)[142] é formado por um conjunto de normas dividido da seguinte forma:

a) ISO/IEC 2500n – Divisão Gestão da Qualidade.
b) ISO/IEC 2501n – Divisão Modelo de Qualidade.
c) ISO/IEC 2502n – Divisão Medição da Qualidade.
d) ISO/IEC 2503n – Divisão Requisitos de Qualidade.
e) 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. Esta divisão inclui duas unidades:

a) *Guia do SQuaRE*, que apresenta a estrutura, terminologia, visão geral do documento, público alvo, modelos de referência e partes associadas da série.
b) *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:

a) *Modelo de referência e guia de medição*: que apresenta uma introdução e explicação sobre a aplicação das medidas de qualidade.
b) *Medidas Primitivas*: um conjunto de medições básicas usadas para a definição das demais.
c) *Medidas Internas*: O conjunto de medidas quantitativas em termos de características e subcaracterísticas internas.

[142] profs.etsmtl.ca/wsuryn/research/SQE-Publ/SQuaRE-second%20generation%20of%20standards%20for%20SW%20Quality%20%28IASTED03%29.pdf

d) *Medidas Externas*: O conjunto de medidas quantitativas em termos de características e subcaracterísticas externas.
e) *Medidas de Uso*: O 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 eliciação 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:

a) *Guia e visão geral da avaliação da qualidade*: que apresenta um *framework* para a avaliação da qualidade de um produto de software.
b) *Processo para desenvolvedores*: recomendações práticas para a avaliação da qualidade de um produto quando esta é feita paralelamente ao seu desenvolvimento.
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 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 agregadas 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 inglês, pois pode acontecer que a tradução nem sempre apresente o espírito do termo na sua língua original.

Tabela 11-2: Modelo de qualidade da ISO 25010:2011.

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

O conjunto de características e subcaracterísticas desta norma modificou-se bastante ao longo do tempo, enquanto ela era elaborada, de forma que 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[143]. 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 necessidades estabelecidas e implicadas quando o produto é usado sob condições especificadas. Suas subcaracterísticas são:

[143] www.meti.go.jp/policy/it_policy/softseibi/metrics/20110324product_metrics2010%28en%29.pdf

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 faltem 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*. Esta subcaracterística indica qual o grau em que 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 com que a maturidade de um sistema diminua com o passar do tempo, ou seja, ao invés 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 se tornar 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 falhas ocorrem tem boa avaliação em relação a esta 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. Estas 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 estas 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 a tolerância a falhas, pode-se dizer que a recuperabilidade trata de situações mais críticas, onde o problema ocorrido não é apenas temporário, como uma falha de comunicação, mas 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 em 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 possa cometer 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. Nestes casos, 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 então é aquele cujo tempo de processamento se aproxima deste 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 simultâneas.

11.1.5 Segurança

A característica de *segurança* avalia o grau em que as funções e 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*) que tem relação com a segurança dos dados e funções, com a qualidade de *uso seguro* (*safety*), que é uma qualidade do software em uso, relacionada com a 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 estejam acessíveis por quem tenha 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 posteriormente negados.
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 esta pessoa ou sistema.
e) *Autenticidade*. Avalia o grau em que a identidade de uma pessoa ou recurso seja 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 terá. No segundo caso, garante-se que quem não deve ter acesso, não 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 outros 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 Manutenibilidade

A *manutenibilidade* é uma característica interna do software que só é diretamente percebida 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 manutenibilidade então mede a facilidade de se realizar alterações no software para sua evolução, ou de detectar e corrigir erros.

A manutenibilidade 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 tenha impacto mínimo nas outras.
b) *Reusabilidade*. Avalia o grau em que partes do sistema podem potencialmente ser usadas para construir outros sistemas.
c) *Analisabilidade*. Um sistema é analisável quando permite encontrar defeitos (depurar) facilmente quando erros ou falhas ocorrem. Sistemas que quando falham travam o computador, por exemplo, podem ter um 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 degradando 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 se 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 o software 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 *transferabilidade* avalia o grau em que o software pode ser efetivamente e eficientemente transferido de um ambiente de hardware ou software para outro. Suas subcaracterísticas são:

a) *Adaptabilidade*. Avalia o quanto é fácil adaptar o software a outros ambientes sem a necessidade de aplicar ações ou meios além daqueles fornecidos com o próprio software.
b) *Instalabilidade*. Avalia a facilidade de se instalar o software.
c) *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 ambiente de desenvolvimento. As características são as seguintes:

a) *Efetividade*. É a capacidade que o produto de software tem para fazer com que o cliente atinja seus objetivos de negócio de forma correta e completa, no ambiente real de uso.
b) *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.
c) *Satisfação*. É a capacidade de o produto satisfazer aos usuários durante seu uso no ambiente final. Esta característica subdivide-se nas seguintes:
    a. *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.
    b. *Prazer.* Avalia o grau em que o usuário sente prazer em usar o sistema para satisfazer seus objetivos.
    c. *Conforto.* Avalia o conforto físico e mental do usuário ao usar o sistema.
    d. Confiança. Avalia o grau em que o usuário ou outros interessados confiam que o sistema faça o que é esperado dele.
d) *Uso sem riscos*. É a capacidade de o produto estar dentro de níveis aceitáveis de segurança relativamente a riscos envolvendo pessoas, negócios e meio ambiente. Essa característica subdivide-se em:
    a. *Mitigação de risco econômico*. Avalia o grau no qual o produto minimiza riscos financeiros potenciais, incluindo danos à propriedade e reputação de pessoas.
    b. *Mitigação de risco a saúde e segurança*. Avalia o grau no qual o produto minimiza riscos físicos às pessoas em seu contexto de uso.
    c. *Mitigação de risco ambiental*. Avalia o grau no qual o produto minimiza riscos ambientais ou à propriedade em seu contexto de uso.
e) *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 contextos além daquele. Essa característica subdivide-se em:
    a. *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 especificados de uso.
    b. *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 de estruturam e 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 este tipo de modelo é que não há um critério claro para decidir quais são as características de alto nível e quais as subcaracterísticas (Kitchenham & Lawrence, 1996). Assim, Dromey (1998)[144] desenvolveu um modelo que procura resolver esse e outros

[144] se.dongguk.ac.kr/metrics/SPQ-Theory.pdf

problemas apontados em relação aos modelos hierárquicos. Ele diz que é impossível construir características de qualidade de alto nível (como usabilidade) diretamente nos produtos. Ao invés 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 sistematicamente determinar 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 o que são 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 considere *comportamentos* como sendo gerais em relação ao sistema, ou seja, não são 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 *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 quando 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 as necessidades de todos os grupos de interesse associados ao software.
e) Cada *atributo de qualidade de software 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 meta modelo, já que modelos individuais podem ser instanciados a partir dele.

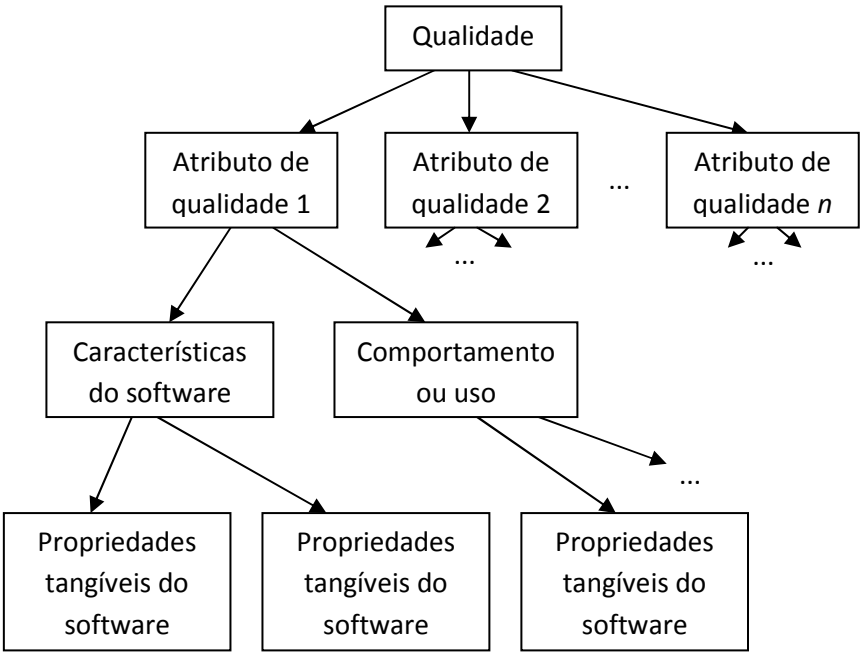

Figura 11-2: Modelo de qualidade de Dromey.

Dromey apresenta dois estudos de caso de aplicação deste modelo, um para a definição de *usabilidade* e outro para a definição de *confiabilidade*. No caso de usabilidade, a titulo 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 usar 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.

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

a) *Apreensibilidade*: uso.
b) *Transparência*: característica do software.

c) *Operabilidade*: uso.
d) *Responsividade*: comportamento (métrica).
e) *Customizabilidade*: uso.
f) *Disponibilização de outros idiomas*: característica do software.
g) *Comandos sensíveis ao contexto*: característica do software.
h) *Imediatismo operacional*: característica do software.
i) *Teclas de atalho*: característica do software.
j) *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 se perguntar para cada duas propriedades se uma é parte ou subtipo de outra. Desta forma, uma hierarquia como a seguinte poderia ser formada a partir das características mostradas no exemplo:

a) Usabilidade
    a. Apreensibilidade
        i. Consistência
        ii. Transparência
        iii. Comandos sensíveis ao contexto
    b. 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 as suas propriedades contextuais intrínsecas tangíveis e a forma como eles são compostos, vão determinar a qualidade do software. Ou seja, a qualidade do software começa na escolha dos nomes das variáveis, por exemplo, que são usadas para criar os métodos, que são 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.

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	Manutenibilidade
Atribuída	Correção	Funcionalidade e confiabilidade
Precisa	Correção	Funcionalidade e confiabilidade
Propósito único	Correção	Funcionalidade e confiabilidade
Encapsulada	Modularidade	Manutenibilidade e reusabilidade

Utilizada	Consistência	Manutenibilidade
Auto-descritiva	Analisabilidade	Manutenibilidade e reusabilidade
Comentada	Analisabilidade	Manutenibilidade e reusabilidade

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.

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

No nível seguinte estão os comandos como atribuições e estruturas de controle, as quais usam expressões. As qualidades são identificadas na Tabela 11-5.

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

As comunicações com o hardware também são analisadas e algumas propriedades indicadas na Tabela 11-6.

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 *start-up*, reinício ou recuperação de erro	Sobrevivência	Confiabilidade
Checagem periódica de *status* operacional	Sobrevivência e tolerância a falhas	Confiabilidade
Checagem periódica de *status* operacional parametrizada	Flexibilidade	Manutenibilidade
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	Modularidade e flexibilidade	Manutenibilidade, reuso e portabilidade

parametrizadas Detalhes arquiteturais de dispositivos estão encapsulados e parametrizados	Modularidade e flexibilidade	Manutenibilidade, reuso e portabilidade

A análise das propriedades que transferem qualidade ligadas à conexão entre o software e a base de dados são apresentadas na Tabela 11-7.

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	Manutenibilidade, portabilidade e reusabilidade
Chamadas à base de dados devem ser processadas fora da unidade que faz a chamada	Independência de aplicação	Manutenibilidade, portabilidade e reusabilidade
A aplicação deve ser independente de detalhes da estrutura da base de dados	Independência de representação e independência de sistema	Manutenibilidade, portabilidade e reusabilidade
O *design* de gerenciamento da base de dados deve fornecer uma forma comum e controlada para adicionar, recuperar e alterar dados	Consistência, interoperabilidade e segurança	Manutenibilidade, reusabilidade e funcionalidade
Itens da base de dados não devem ser referenciados por mais do que um nome	Consistência e eficiência de armazenamento	Manutenibilidade, eficiência e confiabilidade
Modificações na base de dados devem ser feitas por transações que são atômicas, consistentes, íntegras e duráveis	Tolerância a falhas e consistência	Confiabilidade e manutenibilidade
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 design do sistema o protege contra acesso não autorizado	Acessibilidade e segurança	Funcionalidade

As qualidade referentes à comunicação de dados são apresentadas na Tabela 11-8.

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 contém rótulos indicando o tipo de dados	Interoperabilidade	Funcionalidade
Intervalos válidos para todos os dados em nas mensagens são indicados	Tolerância a falhas e analisabilidade	Confiabilidade e manutenibilidade
Propósito e formato de cada item de dados nas mensagens são definidos	Analisabilidade	Manutenibilidade

Glossário técnico para rótulos de mensagens é fornecido	Analisabilidade	Manutenibilidade
Formato comum especificado é usado para posicionamento, empacotamento de dados e transmissão de blocos	Interoperabilidade e analisabilidade	Manutenibilidade e funcionalidade
Representações de dados em mensagens atendem padrões especificados em contrato	Interoperabilidade e analisabilidade	Manutenibilidade e funcionalidade

Mais detalhes podem ser encontrados no trabalho de Dromey (1998)[145].

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 em estabelecer uma anistia geral na empresa (Belchior, 1997), ou seja, não se busca 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, primeiro, acordado e conhecido por todos os envolvidos e, segundo, que se torne parte da cultura da empresa. Planos apenas colocados no papel que são abandonados frente à primeira dificuldade, logo são esquecidos.

Assim, os planos devem ser consistentes, factíveis, gradualmente implementados e, principalmente, todos devem levar os planos a sério. Note-se que planos inconsistentes, impossíveis de executar e que caem do céu prontos e acabados dificilmente serão levados a sério.

Deming (Vasconcelos, Rouiller, Machado, & Medeiros, 2006)[146] estabelece 14 princípios fundamentais para o sucesso de um programa de melhoria de qualidade, dentre os quais destacam-se:

a) Melhorar constantemente o sistema de produção e serviços de forma maximizar o binômio qualidade/produtividade.
b) Institucionalizar os novos métodos de treinamento no trabalhos.
c) Institucionalizar e fortalecer os papeis de liderança.
d) Eliminar os medos.
e) Quebrar as barreiras entre departamentos.
f) Eliminar slogans, exortações e metas de produtividade, pois isto pode levar a 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 em maior detalhe um modelo de implantação e melhoria contínua de processos visando à qualidade.

[145] se.dongguk.ac.kr/metrics/SPQ-Theory.pdf
[146] www.cin.ufpe.br/~if720/*downloads*/Mod.01.MPS_Engenharia%26QualidadeSoftware_V.28.09.06.pdf

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. Ela consiste no planejamento e execução das ações necessárias para que o produto satisfaça os requisitos de qualidade estabelecidos (Seção 11.6).

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

a) *Desconhecimento*. Quando a empresa não sabe sequer que tem problemas com qualidade. Não há compreensão de que qualidade seja um objetivo ou processo de negócio, não são usadas ou conhecidas ferramentas, 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 para a empresa.
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 mudança visando a melhoria da qualidade.
e) *Certeza*. A gestão da qualidade é uma constante e uma parte essencial do funcionamento da empresa. Quase todos os problemas são prevenidos e eliminados antes de surgirem.

Segundo Crosby, o custo relativo da qualidade vai diminuindo gradativamente a medida que se sobe nos níveis de maturidade.

Duas técnicas conhecidas para controle de qualidade (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, o qual é 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*, que é um ciclo de vida a parte, que visa, a partir de especificações formais, produzir produtos de software que sejam isentos de defeitos desde sua origem, não necessitando assim de testes de unidade e integração. As subseções seguintes vão apresentar alguns detalhes a mais sobre as técnicas *walkthrough* e inspeção, bem como sobre o modelo *cleanroom*.

11.4.1 *Walkthrough*

O *walkthrough* (Yourdon, 1985) é uma forma de avaliação do produto que utiliza uma equipe de especialistas, onde cada um faz uma análise prévia do produto, e depois reúnem-se (3 a 5 pessoas) por um período de cerca de duas horas, para trocar suas 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 dos analistas, como por exemplo, "este requisito devia ter sido implementado desta forma mesmo?".

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

Os papeis, em uma reunião *walkthrough* são os seguintes (entre outros)[147]:

a) *Apresentador*. Geralmente é o autor do artefato que o descreve, bem como as razões para ele ser desta forma. Antes da reunião, ele entrega as especificações do artefato ao coordenador, que as distribui à equipe com antecedência.
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. Possivelmente suas notas deverão ser, ao final, aprovadas pelos participantes.
d) *Oráculo de manutenção*. É o inspetor de garantia de qualidade, cujo trabalho é certificar-se que o código produzido seja compreensível e manutenível, de acordo com os padrões da empresa.
e) *Guardião dos padrões*. Seu trabalho é certificar-se 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 a discussão de irrelevâncias.
f) *Representante do usuário*. Ele pode estar presente em algumas reuniões, especialmente aquelas que discutem requisitos, para garantir que o cliente realmente receba o produto que ele espera.
g) Outros desenvolvedores poderão participar também para dar sua visão e contribuição à discussão sob outros pontos de vista.

É importante mencionar que a reunião deve seguir estritamente o planejamento inicial, mantendo-se a discussão produtiva e objetiva, e que o objetivo principal é avaliar os defeitos, e não os desenvolvedores. Além disso, o objetivo da reunião não consiste em 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 for aquele tipo de gênio arrogante, impaciente e de mente estreita, ele pode causar problemas. Devem ser valorizados, pois são capazes de detectar defeitos com facilidade (alimentam seu ego com isso). O

[147] www.hep.wisc.edu/~jnb/structured_*walkthrough*s.html

coordenador da sessão deve ter humildade e controle para não iniciar discussões com eles, nem deixar que outros o façam, pois tornará o trabalho improdutivo.

b) *Pessoas defensivas e inseguras*. Deve-se ter cuidado com esses, pois frequentemente se sentirão atingidos pessoalmente pelas críticas feitas ao seu código. É preciso tomar muito cuidado para que o trabalho seja mantido na discussão do produto, e não dos 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 fieis 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 discussões improdutivas sejam iniciadas por eles.

d) *Alienados.* Estes não estão interessados no mundo real. Eles primam mais pelo processo do que pelo produto, e podem ser uma incomodação séria. O coordenador deve ter em mente sempre que o processo só é útil quando ajuda a produzir o produto da melhor forma possível. O processo não é uma religião que se deva seguir cegamente. As regras existem porque tem objetivos a alcançar, e não porque foram ditadas por alguma divindade da computação. Mas os alienados muitas vezes não percebem isso.

Enfim, muitos problemas interpessoais poderão surgir nas primeiras reuniões. 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 nos objetivos, que são a detecção de defeitos nos programas.

11.4.2 Inspeções Fagan

As *inspeções Fagan* (Fagan, 1976)[148] 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 ela 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.

Então 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, por exemplo, defeitos graves e triviais. Um *defeito grave* é caracterizado por um não funcionamento do produto, como por exemplo, uma função faltando. Um *defeito trivial* é uma característica errada que não afeta a capacidade de funcionamento do software, como, por exemplo, um erro ortográfico em uma janela de sistema.

Os papeis na equipe de inspeção são normalmente os seguintes:

a) *Autor*. O programador, *designer* ou analista, ou seja, a pessoa que produziu o artefato.

[148] www.mfagan.com/pdfs/ibmfagan.pdf

b) *Narrador*. Ele analisa, interpreta sumariza o artefato e seus critérios de aceitação.
c) *Revisores*. Eles 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, segue as seguintes atividades (Fagan, 1986)[149]:

a) *Planejamento*: que inclui a preparação dos materiais (artefatos), convite aos participantes e alocação do espaço de trabalho.
b) *Visão geral*: que inclui a instrução prévia (apresentação) aos participantes sobre os materiais a serem inspecionados e a atribuição de papeis aos participantes.
c) *Preparação*: onde os participantes analisam os artefatos sob inspeção e material de suporte de forma a anotarem possíveis defeitos e questões para a reunião de inspeção.
d) *Reunião de inspeção*: é quando efetivamente se discutem e decidem quais os defeitos encontrados.
e) *Retrabalho*: é a atividade sob responsabilidade do autor do artefato na qual ele corrige os defeitos apontados na reunião de inspeção.
f) *Prosseguimento* (*follow-up*): a atividade de prosseguimento considera que todos os defeitos foram corrigidos e o produto está aprovado para prosseguir para a fase seguinte ou entrega.

É responsabilidade do moderador da inspeção decidir se todos os defeitos foram corrigidos e se o produto pode ir para *follow-up*. Caso o moderador considere que os defeitos não foram adequadamente corrigidos ou que novos defeitos foram introduzidos pelo processo de correção, ele vai determinar que o produto retorne ao processo de inspeção.

Defeitos triviais, possivelmente podem simplesmente ir para retrabalho sem necessidade de novas inspeções. Mas 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 afinamento 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 as reuniões possam acontecer de forma virtual, bem como a disponibilização eletrônica do material, de forma a minimizar também o uso de papel (Genuchten, Cornelissen, & Dijk, 1997).

11.4.3 Método *Cleanroom*

O método *cleanroom* (Mills, Dyer, & Linger, 1987)[150] é uma maneira bastante formal de desenvolver software com foco na qualidade, a qual procura evitar que defeitos sejam introduzidos durante o desenvolvimento. O nome foi inspirado nas salas de fábricas de circuitos integrados (as salas limpas), onde se estabelece que um circuito não precisa ser limpo

[149] www.mfagan.com/pdfs/aisi1986.pdf
[150] ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1695817

depois de fabricado porque ele já é produzido em um ambiente sem sujeira. Da mesma forma, espera-se que software produzido em um ambiente *limpo* possa estar isento de defeitos.

O principal objetivo do método é produzir software que apresente taxa zero de defeitos durante seu uso (Linger & Trammell, 1996)[151].

Os princípios básicos do método *cleanroom* são:

a) *Desenvolvimento baseado em métodos formais*. Usa-se estruturas de *caixas* (ver abaixo) para especificar software. Uma equipe de revisão verifica se o produto satisfaz a 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 feito por uma equipe independente verifica que a qualidade permaneça em nível aceitável.
c) *Programação estruturada*. Apenas um conjunto limitado de estruturas de abstração são permitidas. O processo de desenvolvimento de programas é um processo de refinamento passo a passo que usa 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. São realizados diretamente os testes de sistema.

O método *cleanroom* é definido em termos de 14 processos e 20 artefatos (produtos do trabalho) os quais são 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. Programas podem, então, ser projetados como decomposições de suas especificações funcionais e, assim, podem ser 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*. Ela define o comportamento esperado de um uma função do sistema em todos os seus contextos de uso: *estímulo corrente + histórico de estímulos -> resposta*.
b) *State box*. Ela pode ser derivada de uma *black box* e define o histórico de estímulos como um estado: *estímulo corrente + estado atual -> resposta + novo estado*.
c) *Clear box*. Ela é idêntica a uma *state box*, mas sua especificação é feita por procedimentos em linguagem de programação, ou seja, ela é a realização de uma *state box*.

[151] www.sei.cmu.edu/reports/96tr022.pdf

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 deste método é a dificuldade em se obter desenvolvedores suficientemente preparados em lógica e matemática para atuarem com especificação formal de sistemas[152].

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 um pouco mais 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:

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

Quando usuários pensam em qualidade de software, eles usualmente lembram-se da característica de confiabilidade (Seção 11.1.2), isto é, o 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 pré-determinado 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 encontre 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 passar a ser identificada mais cedo, é porque a mudança de processo foi salutar em relação a essa métrica.

[152] www.cs.st-andrews.ac.uk/~ifs/Books/SE9/Web/Cleanroom/experience.htm

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[153]). 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 usado refazendo o que já havia sido aprovado.

Poderá ainda 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 defeitos ou erro nos requisitos será efetivamente contabilizado como uma atividade não produtiva, pois a melhoria ou aprimoramento de outras qualidades do software será um investimento.

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 possivelmente podem ser obtidas a partir de um bom processo de desenvolvimento, como por exemplo, o software ser livre de defeitos. Mas outras qualidades (como portabilidade, por exemplo) poderão ser eletivas e o custo de sua inclusão no software poderá não ser justificável. Belchior (1997)[154] 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, são definidos para o software como um todo e não para uma função individual. Mas também podem ser requisitos não funcionais, quando se aplicam a apenas uma ou 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 um 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).

[153] en.wikipedia.org/wiki/Year_2000_problem
[154] www.boente.eti.br/fuzzy/tese-fuzzy-belchior.pdf

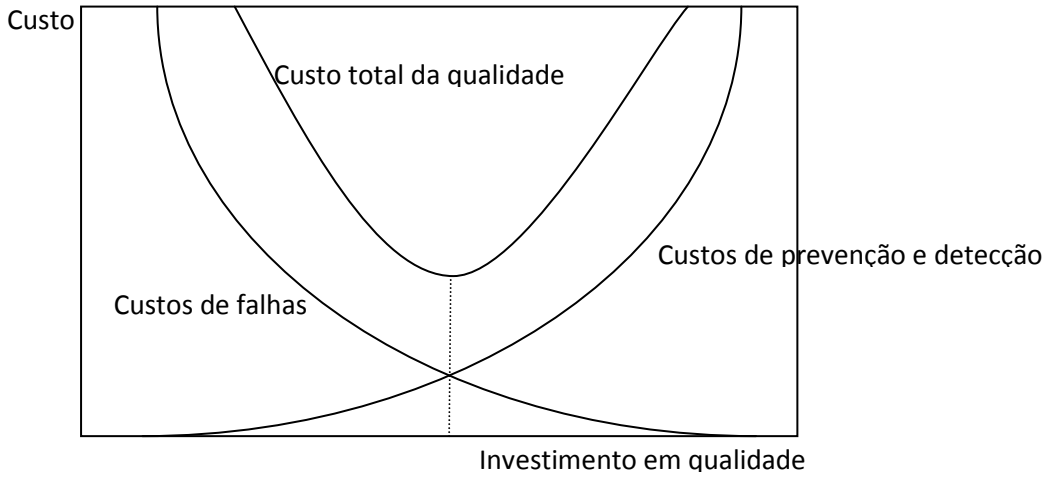

Figura 11-3: Relação entre investimento em qualidade e economia relacionada a falhas[155].

Se por um lado as medidas de qualidade mais fundamentais (como software livre de defeitos) podem até 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? Este requisito está colocado de maneira subjetiva, ou seja, duas pessoas poderão ter uma opinião diferente sobre se um determinado produto de software é ou não fácil de usar. Desta forma não há como avaliar se o requisito foi atendido. Mas o problema aqui 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". Desta forma, o produto final pode ser inspecionado e o requisito de qualidade, conforme especificado, pode ser avaliado como satisfeito ou não.

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

O método GQM (Basili, Caldiera, & Rombach, 1994) [156] , um acrônimo para "*Goal*/*Question*/*Metric*", é uma abordagem para avaliação de qualidade de software. O GQM define um modelo de mensuração em três níveis:

a) *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 relativamente a um ambiente em particular (os *objetos* a serem medidos podem ser: produtos, processos ou recursos).

[155] Fonte: Kerzner (1998).
[156] ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf

b) *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.
c) *Nível quantitativo* (*Metric*/métrica): um conjunto de dados, baseados nos modelos, é associado com cada questão para respondê-la de forma quantitativa (os dados podem ser *objetivos*, se dependem apenas do objeto avaliado, ou *subjetivos*, se dependem de uma interpretação do avaliador).

Um guia completo de aplicação de GQM também pode ser consultado na internet (Solingen & Berghout, 1999)[157]. 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 até as métricas, enquanto que a interpretação dos resultados é feita de forma *bottom-up*, ou seja, das métricas até os objetivos. A Figura 11-4 apresenta esquematicamente o modelo GQM. Observa-se que as métricas não são exclusivas para as questões.

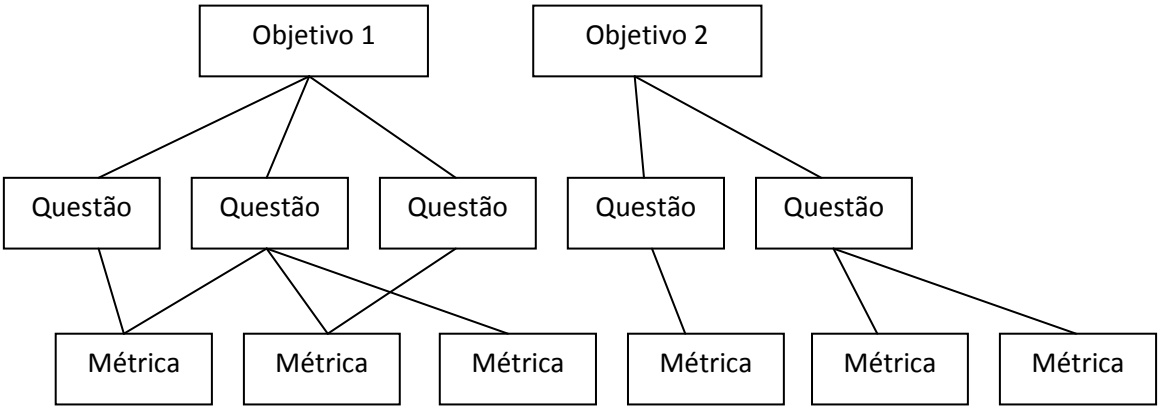

Figura 11-4: Estrutura hierárquica do modelo GQM.

Santos e Pretz (2009)[158] apresentam um estudo de caso onde 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 (Figura 11-5), na época ainda em vigência.

[157] www.iteva.rug.nl/gqm/GQM%20Guide%20non%20printable.pdf
[158] tc*online*.feevale.br/tc/files/6163_38.pdf

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
			Maturidade
			Recuperabilidade
	R002-Insuficiência dos recursos envolvidos com a produção do sistema, causando indisponibilidade.	Eficiência	Utilização de recursos
Envio e recepção de informações dos sistemas relacionados	R003 – Interceptação de informações sigilosas no tráfego de rede utilizado pelo sistema.	Funcionalidade	Segurança de acesso
	R004 – Acesso liberado, aos usuários da aplicação, às informações que ficam inseridas no banco local instalado na estação de trabalho do usuário.		

Figura 11-5: Associação de riscos às características e subcaracterísticas de qualidade[159].

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 (Figura 11-6).

[159] Fonte: Santos e Pretz (2009).

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
	Tolerância a falhas e Recuperabilidade	Avaliar a disponibilidade do sistema do ponto de vista do usuário	Quantos padrões de defeitos são mantidos sob controle para evitar falhas criticas 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
			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
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 a transmissão do primeiro ao último item avaliado / máximo requerido de mensagens de erro e falhas relacionadas a 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, aos usuários da aplicação, às informações que ficam 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

Figura 11-6: Exemplo de aplicação do modelo GQM[160].

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

a) *Propósito*. Um verbo que representa o objetivo, como, por exemplo, “avaliar”.
b) *Questão*. Um adjetivo referente ao objeto, como, por exemplo, “a maturidade de”.
c) *Objeto*. O objeto em avaliação, como, por exemplo, “o software”.
d) *Ponto de vista*. Para quem a avaliação é feita, como, por exemplo, “do ponto de vista do cliente”.

Adicionalmente 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).

[160] Fonte: Santos e Pretz (2009).

