PARTE

três

GESTÃO DA QUALIDADE

Nesta parte do livro serão vistos os princípios, as técnicas e os conceitos aplicados ao gerenciamento e controle da qualidade de software. As seguintes questões são tratadas nos próximos capítulos:

- Quais são as características genéricas de um software de alta qualidade?
- Como revisar a qualidade e como são realizadas revisões eficazes?
- O que é garantia da qualidade de software?
- Quais são as estratégias aplicáveis aos testes de software?
- Quais métodos são usados para projetar casos de teste eficazes?
- Existem métodos práticos para garantir que um software está correto?
- Como gerenciar e controlar mudanças que sempre ocorrem quando um software é criado?
- Que medidas e métricas podem ser usadas para avaliar a qualidade dos modelos de requisitos e de projeto, código-fonte e casos de teste?

Assim que todas essas questões forem respondidas, garante-se uma melhor preparação para a produção de software de alta qualidade.

14 Conceitos de qualidade

Conceitos-Chave

O clamor por maior qualidade de software começou realmente quando o software passou a se tornar cada vez mais integrado em todas as atividades de nossas vidas. Na década de 1990, as principais empresas reconheciam que bilhões de dólares por ano estavam sendo desperdiçados em software que não apresentava as características e as funcionalidades prometidas. Pior ainda, tanto o governo como as empresas ficavam cada vez mais preocupados com o fato de que uma falha grave de software poderia inutilizar importantes infraestruturas, aumentando o custo em dezenas de bilhões. Na virada do século, a *CIO Magazine* [Lev01] anunciou em manchete: "Chega de desperdiçar US$ 78 bilhões por ano", lamentando o fato de que "as empresas americanas gastavam bilhões em softwares que não faziam o que supostamente deveriam fazer". A *InformationWeek* [Ric01] anunciou a mesma preocupação:

> Apesar das boas intenções, código mal feito continua a ser o "fantasma" do mercado de software, sendo responsável por até 45% do tempo de inatividade dos sistemas computacionais e custando às empresas americanas cerca de US$ 100 bilhões no último ano em termos de manutenção e redução da produtividade, afirma o Standish Group, uma empresa de pesquisa de mercado. Isso não inclui o custo da perda de clientes insatisfeitos. Pelo fato de as empresas de TI escreverem aplicações que dependem de pacotes de software de infraestrutura, código de má qualidade pode causar estragos em aplicações personalizadas bem como...
>
> Qual o prejuízo causado por software de má qualidade? As definições variam, mas especialistas dizem que bastam três ou quatro defeitos a cada 1.000 linhas de código para fazer com que um programa execute de forma inadequada. Acrescente a isso que a maioria dos programadores insere cerca de um erro a cada 10 linhas de código escrito, multiplicados por milhões de linhas de código em vários produtos comerciais. Assim, deduz-se que o custo dos fornecedores de software será de pelo menos a metade dos seus orçamentos para a realização dos testes e correção dos erros. Percebeu o tamanho do problema?

Panorama

O que é? A resposta não é tão simples quanto se imagina. Sabe-se o que é qualidade ao vê-la e, mesmo assim, pode ser algo difícil de definir. Porém, para software de computador, qualidade é algo que tem de ser definido e é isso o que é feito neste capítulo.

Quem realiza? Todo mundo: os engenheiros de software, gerentes, todos os interessados; todos os envolvidos na gestão de qualidade são responsáveis por ela.

Por que é importante? Pode-se fazer certo da primeira vez ou então fazer tudo de novo. Se uma equipe de software enfatizar a qualidade em todas as atividades de engenharia de software, ela reduzirá a quantidade de reformulações que terá de fazer. Isso resulta em custos menores e, mais importante ainda, menor tempo para a colocação do produto no mercado.

Quais são as etapas envolvidas? Para obter software de alta qualidade, devem ocorrer quatro atividades: processo e prática comprovados de engenharia de software, gerenciamento consistente de projetos, controle global de qualidade e a presença de uma infraestrutura para garantir a qualidade.

Qual o artefato? Software que atenda às necessidades do cliente, execute de forma precisa e confiável e gere valor para todos aqueles que o utilizam.

Como garantir que o trabalho foi realizado corretamente? Acompanhando a qualidade por meio da verificação dos resultados de todas as atividades de controle de qualidade, medindo a qualidade efetuando a verificação de erros antes da entrega e de defeitos que acabaram escapando e indo para a produção.

Em 2005, a *ComputerWorld* [Hil05] lamentou que "software de má qualidade está em praticamente todas as organizações que usam computadores, provocando horas de trabalho perdidas durante o tempo em que a máquina fica parada, dados perdidos ou corrompidos, oportunidades de vendas perdidas, custos de suporte e manutenção de TI elevados e baixa satisfação do cliente". Um ano mais tarde, a *InfoWorld* [Fos06] escreveu sobre "o estado de penúria da qualidade de software", relatando que o problema da qualidade não havia melhorado.

Hoje em dia, a qualidade de software continua a ser um problema, mas a quem culpar? Os clientes culpam os desenvolvedores, argumentando que práticas descuidadas levam a um software de baixa qualidade. Os desenvolvedores de software culpam os clientes (e outros interessados), argumentando que datas de entrega absurdas e um fluxo contínuo de mudanças os forçam a entregar software antes de eles estarem completamente validados. Quem está com a razão? *Ambos* — e esse é o problema. Neste capítulo, considera-se a qualidade de software um conceito e examina-se por que vale a pena considerá-la seriamente toda vez que as práticas de engenharia de software forem aplicadas.

14.1 O QUE É QUALIDADE?

Em seu livro místico, *Zen and the Art of Motorcycle Maintenance,* Robert Persig [Per74] comentou sobre aquilo que denominamos *qualidade*:

> Qualidade... Sabemos o que ela é, embora não saibamos o que ela é. Mas essa afirmação é contraditória. Mas algumas coisas são melhores que outras; ou seja, elas têm mais qualidade. Mas quando tentamos dizer o que é qualidade, excetuando as coisas que a têm, tudo desaparece como num passe de mágica! Não há nada para dizer a respeito. Mas se não conseguimos dizer o que é Qualidade, como saber o que ela é ou como saber até se ela existe mesmo? Se ninguém sabe o que ela é, então para fins práticos ela não existiria. Porém, para fins práticos, ela realmente existe. Em que mais se baseia a qualidade? Por que outro motivo as pessoas pagariam fortunas por algumas coisas e jogariam outras na lata de lixo? Obviamente, certas coisas são melhores que outras... Mas o que é o melhor?... E por aí vai (andando em círculos), girando rodas mentais e em nenhum lugar encontrando um ponto de tração. Mas o que é mesmo Qualidade? O que é isso?

De fato — o que é isso?

Quais são as diferentes maneiras em que a qualidade pode ser visualizada?

Em um nível mais pragmático, David Garvin [Gar84], da Harvard Business School, sugere que "qualidade é um conceito complexo e multifacetado" que pode ser descrito segundo cinco pontos de vista diferentes. A *visão transcendental* sustenta (assim como Persig) que qualidade é algo que se reconhece imediatamente, mas não se consegue definir explicitamente. A *visão do usuário* vê a qualidade em termos das metas específicas de um usuário final. Se um produto atende a essas metas, ele apresenta qualidade. A *visão do fabricante* define qualidade em termos da especificação original do produto. Se o produto atende às especificações, ele apresenta qualidade. A *visão do produto* sugere que a qualidade pode ser ligada a características inerentes (por exemplo, funções e recursos) de um produto. Finalmente, a *visão baseada em valor* mede a qualidade tomando como base o quanto um cliente estaria disposto a pagar por um produto. Na realidade, qualidade engloba todas essas visões e outras mais.

Qualidade de projeto refere-se às características que os projetistas especificam para um produto. A qualidade dos materiais, as tolerâncias e as especificações de desempenho, todos são fatores que contribuem para a qualidade de um projeto. Quanto mais materiais de alta qualidade forem usados, tolerâncias mais rígidas e níveis de desempenho maiores forem especificados, a qualidade de projeto de um produto aumentará se o produto for fabricado de acordo com essas especificações.

No desenvolvimento de software, a qualidade de um projeto engloba o grau de atendimento às funções e características especificadas no modelo de requisitos. A *qualidade de conformidade* focaliza o grau em que a implementação segue o projeto e o sistema resultante atende suas necessidades e as metas de desempenho.

"As pessoas esquecem quão rápido um trabalho foi realizado — mas elas sempre lembram quão bem ele foi realizado."

Howard Newton

Mas a qualidade do projeto e a qualidade de conformidade são as únicas questões que os engenheiros de software devem considerar? Robert Glass [Gla98] sustenta que uma relação mais "intuitiva" é o indicado:

satisfação do usuário = produto compatível + boa qualidade + entrega dentro do orçamento e do prazo previsto

Ou seja, Glass argumenta que a qualidade é importante, mas se o usuário não estiver satisfeito, nada mais importa. DeMarco [DeM98] reforça esse ponto de vista ao afirmar: "A qualidade de um produto é função do quanto ele transforma o mundo para melhor". Essa visão de qualidade sustenta que se um produto de software fornece benefício substancial a seus usuários finais, é possível que eles estejam dispostos a tolerar problemas ocasionais de confiabilidade ou desempenho.

14.2 Qualidade de Software

Até mesmo os desenvolvedores de software mais experientes concordarão que software de alta qualidade é um objetivo importante. Mas como definir *a qualidade de software*? No sentido mais geral, a qualidade de software pode ser definida[1] como: *uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam.*

Não há dúvida nenhuma de que essa definição pode ser modificada ou estendida e debatida interminavelmente. Para os propósitos deste livro, a definição serve para enfatizar três pontos importantes:

1. Uma *gestão de qualidade efetiva* estabelece a infraestrutura que dá suporte a qualquer tentativa de construir um produto de software de alta qualidade. Os aspectos administrativos do processo criam mecanismos de controle e equilíbrio de poderes que ajudam a evitar o caos no projeto — um fator-chave para uma qualidade inadequada. As práticas de engenharia de software permitem ao desenvolvedor analisar o problema e elaborar uma solução consistente —, aspectos críticos na construção de software de alta qualidade. Finalmente, as atividades de apoio como o gerenciamento de mudanças e as revisões técnicas têm muito a ver com a qualidade, assim como qualquer outra parte da prática de engenharia de software.
2. Um *produto útil* fornece o conteúdo, as funções e os recursos que o usuário final deseja, além disso, e não menos importante, deve fornecer confiabilidade e isenção de erros. Um produto útil sempre satisfaz às exigências definidas explicitamente pelos interessados. Além disso, ele satisfaz a um conjunto de requisitos implícitos (por exemplo, facilidade de uso) que se espera de todo software de alta qualidade.
3. Ao *agregar valor tanto para o fabricante quanto para o usuário* de um produto de software, um software de alta qualidade gera benefícios para a empresa de software bem como para a comunidade de usuários finais. A empresa fabricante do software ganha valor agregado pelo fato de um software de alta qualidade exigir menos manutenção, menos correções de erros e menos suporte ao cliente. Isso permite que os engenheiros de software despendam mais tempo criando aplicações novas e menos tempo em manutenções. A comunidade de usuários ganha um valor agregado, pois a aplicação fornece a capacidade de agilizar algum processo de negócio. O resultado final é: (1) maior receita gerada pelo produto de software, (2) maior rentabilidade quando uma aplicação suporta um processo de negócio, e/ou (3) maior disponibilidade de informações cruciais para o negócio.

14.2.1 Dimensões de qualidade de Garvin

David Garvin [Gar87] sugere que a qualidade deve ser considerada adotando-se um ponto de vista multidimensional que começa com uma avaliação da conformidade e termina com uma

1 Essa definição foi adaptada de [Bes04] e substitui uma visão mais voltada para a fabricação apresentada em edições anteriores deste livro.

visão transcendental (estética). Embora as oito dimensões de qualidade de Garvin não tenham sido desenvolvidas especificamente para software, elas podem ser aplicadas quando se considera qualidade de software:

Qualidade do desempenho. O software fornece todo o conteúdo, funções e recursos que são especificados como parte do modelo de requisitos de forma a gerar valor ao usuário final?

Qualidade dos recursos. O software fornece recursos que surpreendem e encantam usuários finais que os utilizam pela primeira vez?

Confiabilidade. O software fornece todos os recursos e capacidades sem falhas? Está disponível quando necessário? Fornece funcionalidade sem a ocorrência de erros?

Conformidade. O software está de acordo com os padrões de software locais e externos relacionados com a aplicação? Segue as convenções de projeto e codificação de fato? Por exemplo, a interface com o usuário está de acordo com as regras de projeto aceitas para seleção de menus ou entrada de dados?

Durabilidade. O software pode ser mantido (modificado) ou corrigido (depurado) sem a geração involuntária de efeitos colaterais indesejados? As mudanças farão com que a taxa de erros ou a confiabilidade diminuam com o passar do tempo?

Facilidade de manutenção. O software pode ser mantido (modificado) ou corrigido (depurado) em um período de tempo aceitável e curto? O pessoal de suporte pode obter todas as informações necessárias para realizar alterações ou corrigir defeitos? Douglas Adams [Ada93] comenta ironicamente: "A diferença entre algo que pode dar errado e algo que possivelmente não pode dar errado é que quando algo que possivelmente não pode dar errado, dá errado, normalmente acaba sendo impossível acessá-lo ou repará-lo".

Estética. Não há dúvida nenhuma de que cada um de nós tem uma visão diferente e muito subjetiva do que é estética. Mesmo assim, a maioria de nós concordaria que uma entidade estética tem certa elegância, um fluir único e uma "presença" que são difíceis de quantificar mas que, não obstante, são evidentes. Um software estético possui essas características.

Percepção. Em algumas situações, temos alguns preconceitos que influenciarão nossa percepção de qualidade. Por exemplo, se for apresentado um produto de software construído por um fornecedor que, no passado, havia produzido software de má qualidade, ficaremos com a nossa percepção de qualidade do produto de software influenciada negativamente. Similarmente, se um fornecedor tem uma excelente reputação, talvez percebamos qualidade, mesmo quando ela realmente não existe.

As dimensões de qualidade de Garvin nos dão uma visão "indulgente" da qualidade de software. Muitas (mas não todas) dessas dimensões podem ser consideradas apenas subjetivamente. Por tal razão, também precisamos de um conjunto de fatores de qualidade que podem ser classificados em duas grandes categorias: (1) fatores que podem ser medidos diretamente (por exemplo, defeitos revelados durante testes) e (2) fatores que podem ser medidos apenas indiretamente (por exemplo, usabilidade ou facilidade de manutenção). Em cada um dos casos deve ocorrer medição. Devemos comparar o software com algum dado e chegarmos a uma indicação da qualidade.

14.2.2 Fatores de qualidade de McCall

McCall, Richards e Walters [McC77] criaram uma proposta de categorização dos fatores que afetam a qualidade de software. Esses fatores de qualidade de software, apresentados na Figura 14.1, focam em três importantes aspectos de um produto de software: as características operacionais, a habilidade de suportar mudanças e a adaptabilidade a novos ambientes.

Referindo-se aos fatores citados na Figura 14.1, McCall e seus colegas fazem as seguintes descrições:

Correção. O quanto um programa satisfaz a sua especificação e atende aos objetivos da missão do cliente.

Confiabilidade. O quanto se pode esperar que um programa realize a função pretendida com a precisão exigida. [Observe que foram propostas outras definições mais completas de confiabilidade (veja o Capítulo 25).]

Eficiência. A quantidade de recursos computacionais e código exigidos por um programa para desempenhar sua função.

Integridade. O quanto o acesso ao software ou dados por pessoas não autorizadas pode ser controlado.

Usabilidade. Esforço necessário para aprender, operar, preparar a entrada de dados e interpretar a saída de um programa.

Facilidade de manutenção. Esforço necessário para localizar e corrigir um erro em um programa. [Trata-se de uma definição muito limitada.]

Flexibilidade. Esforço necessário para modificar um programa em operação.

Testabilidade. Esforço necessário para testar um programa de modo a garantir que ele desempenhe a função destinada.

Portabilidade. Esforço necessário para transferir o programa de um ambiente de hardware e/ou software para outro.

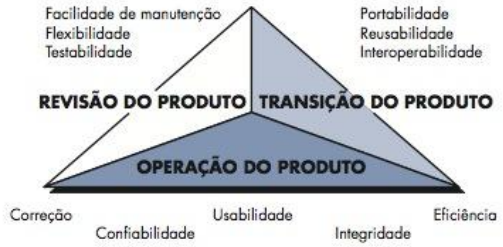
Reusabilidade. O quanto um programa [ou partes de um programa] pode ser reutilizado em outras aplicações — relacionado com o empacotamento e o escopo das funções que o programa executa.

Interoperabilidade. Esforço necessário para integrar um sistema a outro.

> "O amargo da má qualidade permanece muito tempo depois da doçura do cumprimento do prazo ter sido esquecida."
>
> **Karl Weigers (citação não atribuída)**

É difícil, e em alguns casos impossível, desenvolver medidas[2] diretas desses fatores de qualidade. Na realidade, muitas das métricas definidas por McCall et al. podem ser medidas apenas indiretamente. Entretanto, avaliar a qualidade de uma aplicação usando esses fatores possibilitará uma sólida indicação da qualidade de um software.

FIGURA 14.1
Fatores de qualidade de software de McCall



14.2.3 Fatores de qualidade ISO 9126

O padrão ISO 9126 foi desenvolvido como uma tentativa de identificar os atributos fundamentais de qualidade para software de computador. O padrão identifica seis atributos fundamentais de qualidade:

2 Uma *medida direta* implica existir um único valor contável que dê uma indicação direta do atributo que está sendo examinado. Por exemplo, o "tamanho" de um programa pode ser medido diretamente contando-se o número de linhas de código.

Funcionalidade. O grau com que o software satisfaz às necessidades declaradas conforme indicado pelos seguintes subatributos: adequabilidade, exatidão, interoperabilidade, conformidade e segurança.

Confiabilidade. A quantidade de tempo que o software fica disponível para uso conforme indicado pelos seguintes subatributos: maturidade, tolerância a falhas, facilidade de recuperação.

Usabilidade. O grau de facilidade de utilização do software conforme indicado pelos seguintes subatributos: facilidade de compreensão, facilidade de aprendizagem, operabilidade.

Eficiência. O grau de otimização do uso, pelo software, dos recursos do sistema conforme indicado pelos seguintes subatributos: comportamento em relação ao tempo, comportamento em relação aos recursos.

Facilidade de manutenção. A facilidade com a qual uma correção pode ser realizada no software conforme indicado pelos seguintes subatributos: facilidade de análise, facilidade de realização de mudanças, estabilidade, testabilidade.

Portabilidade. A facilidade com a qual um software pode ser transposto de um ambiente a outro conforme indicado pelos seguintes subatributos: adaptabilidade, facilidade de instalação, conformidade, facilidade de substituição.

AVISO

Embora seja tentador criar medidas quantitativas para os fatores de qualidade aqui citados, também podemos criar uma lista de verificação simples dos atributos que dão uma sólida indicação de que o fator está presente.

Assim como outros fatores de qualidade de software discutidos nas subseções anteriores, os fatores da ISO 9126 não levam, necessariamente, à medição direta. Entretanto, eles fornecem uma base razoável para medidas indiretas e uma excelente lista de verificação para avaliar a qualidade de um sistema.

14.2.4 Fatores de qualidade desejados

"Qualquer atividade se torna criativa quando o executor se preocupa em fazê-la de maneira correta ou melhor."

John Updike

As dimensões e os fatores de qualidade apresentados nas Seções 14.2.1 e 14.2.2 concentram-se no software como um todo e podem ser usados como uma indicação genérica da qualidade de uma aplicação. A equipe de software pode desenvolver um conjunto de características de qualidade e questões associadas que investigaria[3] o grau em que cada fator foi satisfeito. Por exemplo, McCall identifica a *usabilidade* como sendo um importante fator de qualidade. Se lhe fosse solicitado para revisar uma interface com o usuário e avaliar sua usabilidade, como você procederia? Você poderia começar com os subatributos sugeridos por McCall — facilidade de compreensão, facilidade de aprendizagem, operabilidade — mas qual seu significado em um sentido prático?

Para conduzir sua avaliação, você precisaria lidar com atributos específicos, mensuráveis (ou pelo menos, reconhecíveis) da interface. Por exemplo [Bro03]:

Intuição. O grau em que a interface segue padrões de uso esperados de modo que até mesmo um novato possa usá-la sem treinamento significativo.

- O layout da interface favorece a fácil compreensão?
- As operações da interface são fáceis de ser localizadas e iniciadas?
- A interface utiliza uma metáfora reconhecível?
- É especificada entrada para economizar toques de teclado ou cliques de mouse?
- A interface segue as três regras de ouro? (Ver o Capítulo 11.)
- A estética ajuda no entendimento e uso?

Eficiência. A facilidade com a qual as operações e informações podem ser localizadas ou iniciadas.

- O layout e o estilo da interface permitem a um usuário localizar eficientemente as operações e informações?

3 Essas características e questões podem ser abordadas como parte de uma revisão de software (Capítulo 15).

- Uma sequência de operações (ou entrada de dados) pode ser realizada reduzindo-se o número de movimentos?
- Os dados de saída ou o conteúdo são apresentados de modo a ser imediatamente compreendidos?
- As operações hierárquicas foram organizadas de modo a minimizar o nível em que um usuário deve navegar para realizar algo?

Robustez. O grau com o qual o software trata dados incorretos de entrada ou interação inapropriada com o usuário.

- O software reconhecerá erros caso sejam introduzidos dados dentro ou fora dos limites prescritos? Mais importante ainda, o software continuará a operar sem falha ou degradação?
- A interface reconhece erros cognitivos ou manipuladores comuns e orienta explicitamente o usuário para retomar o caminho certo?
- A interface oferece diagnósticos e orientação úteis quando é descoberta uma condição de erro (associada à funcionalidade do software)?

Riqueza. O grau em que a interface oferece um conjunto rico de recursos importantes.

- A interface pode ser personalizada de acordo com as necessidades específicas de um usuário?
- A interface dispõe de recursos de macros que permitem ao usuário identificar uma sequência de operações comuns por meio de uma única ação ou comando?

Enquanto o projeto de interface é desenvolvido, a equipe de software revisaria o protótipo de projeto e faria as perguntas citadas. Se a resposta a essas perguntas for "sim", é provável que a interface com o usuário apresente alta qualidade. Um conjunto de perguntas similares deveria ser desenvolvido para cada fator de qualidade a ser avaliado.

14.2.5 A transição para uma visão quantitativa

Nas subseções anteriores, apresentamos diversos fatores qualitativos para a "medição" da qualidade de um software. A comunidade da engenharia de software se esforça ao máximo para desenvolver medidas precisas para a qualidade de software e algumas vezes é frustrada pela natureza subjetiva da atividade. Cavano e McCall [Cav78] discutem essa situação:

> A determinação da qualidade é um fator-chave nos eventos do dia a dia — concursos de degustação de vinhos, eventos esportivos (por exemplo, ginástica), concursos de talentos etc. Nessas situações, a qualidade é julgada da forma mais fundamental e direta: comparação entre dois objetos sob condições idênticas e com conceitos predeterminados. O vinho pode ser julgado segundo sua limpidez, cor, buquê, sabor etc. Entretanto, esse tipo de julgamento é muito subjetivo; para ter algum valor, ele precisa ser feito por um especialista.
>
> A subjetividade e a especialização também se aplicam na determinação da qualidade do software. Para ajudar a solucionar esse problema, é preciso uma definição mais precisa da qualidade de software bem como uma maneira de obter medidas quantitativas da qualidade de software para uma análise objetiva... Como não existe conhecimento absoluto sobre isso, não se deve ter a expectativa de medir a qualidade de software com exatidão, já que cada medida é parcialmente imperfeita. Jacob Bronkowski descreve esse paradoxo do conhecimento da seguinte maneira: "Ano após ano, inventamos instrumentos mais precisos com os quais observamos a natureza com maior perfeição. E ao analisarmos essas observações, ficamos desconcertados em ver que elas ainda são distorcidas e ainda incertas".

No Capítulo 23, apresentaremos um conjunto de métricas de software que podem ser aplicadas para a avaliação quantitativa da qualidade de software. Em todos os casos, as métricas representam medidas indiretas; isto é, jamais medimos realmente a *qualidade,* mas sim alguma manifestação dessa qualidade. O fator complicador é a relação precisa entre a variável medida e a qualidade de software.

14.3 O Dilema da Qualidade de Software

Ao deparar com o dilema da qualidade (e todo mundo depara com ele uma hora ou outra), tente alcançar o equilíbrio — esforço suficiente para produzir qualidade aceitável sem "enterrar" o projeto.

Em uma entrevista [Ven03] publicada na Internet, Bertrand Meyer discute o que chamamos de *dilema da qualidade*:

> Se produzimos um sistema de software de péssima qualidade, perdemos porque ninguém irá querer comprá-lo. Se, por outro lado, gastamos um tempo infinito, um esforço extremamente grande e grandes somas de dinheiro para construir um software absolutamente perfeito, então isso levará muito tempo para ser completado, e o custo de produção será tão alto que iremos à falência. Ou perdemos a oportunidade de mercado ou então simplesmente esgotamos todos os nossos recursos. Dessa maneira, os profissionais desta área tentam encontrar aquele meio-termo mágico onde o produto é suficientemente bom para não ser rejeitado logo de cara, como, por exemplo, durante uma avaliação, mas também não é o objeto de tamanho perfeccionismo e trabalho que levaria muito tempo ou que custaria demasiadamente para ser finalizado.

É válido afirmar que os engenheiros de software devem se esforçar para produzir sistemas de alta qualidade. Muito melhor seria aplicar boas práticas na tentativa de obter essa alta qualidade. Porém, a situação discutida por Meyer é a realidade e representa um dilema até mesmo para as melhores organizações de engenharia de software.

14.3.1 Software "bom o suficiente"

Falando claramente, se devemos aceitar o argumento feito por Meyer, é aceitável produzirmos software "bom o suficiente"? A resposta a essa pergunta deve ser "sim", pois atualmente, as principais empresas de software agem dessa forma. Elas criam software com erros (bugs) conhecidos e os entregam a vários usuários finais. Elas reconhecem que algumas funções e características disponibilizadas na Versão 1.0 podem não ser da melhor qualidade e planejam melhoramentos para a Versão 2.0. Elas fazem isso mesmo sabendo que alguns clientes irão reclamar, entretanto, reconhecem que o tempo de colocação do produto no mercado é a melhor cartada de qualidade desde que o produto fornecido seja "bom o suficiente".

O que é exatamente "bom o suficiente"? Software bom o suficiente fornece funções e características de alta qualidade que os usuários desejam, mas, ao mesmo tempo, fornece outras funções e características mais obscuras ou especializadas e ainda contendo erros conhecidos. O fornecedor de software espera que a grande maioria dos usuários ignore os erros pelo fato de estarem muito satisfeitos com as outras funcionalidades oferecidas pela aplicação.

Essa ideia pode ter um significado especial para muitos leitores. Se você for um deles, pedimos para considerar alguns dos argumentos contra software "bom o suficiente".

É verdade que software "bom o suficiente" pode funcionar em alguns domínios de aplicação e para algumas grandes empresas de software. Afinal de contas, se uma empresa tiver um grande orçamento para o marketing e conseguir convencer um número suficiente de pessoas a comprar a versão 1.0, ela será bem-sucedida. Conforme citado anteriormente, pode-se argumentar que a qualidade será melhorada nas próximas versões. Ao entregar a versão 1.0 boa o suficiente, a empresa já monopolizou o mercado.

Caso trabalhe em uma pequena empresa, não confie nessa filosofia. Ao entregar um produto bom o suficiente (com erros), você corre o risco de arruinar permanentemente a reputação da empresa. Talvez jamais tenha a chance de entregar a versão 2.0 pois, devido à má propaganda resultante dessa filosofia, as vendas podem despencar e, consequentemente, a empresa falir.

Caso trabalhe em certos domínios de aplicação (por exemplo, software embarcado para aplicações em tempo real) ou crie software de aplicação integrado com o hardware (por exemplo, software para a indústria automotiva, software para telecomunicações), entregar software com erros conhecidos pode ser negligente e expõe sua empresa a litígios custosos. Em alguns casos, pode constituir crime. Ninguém quer software bom o suficiente e inútil!

Portanto, proceda com cautela caso acredite que "bom o suficiente" seja um atalho capaz de resolver seus problemas de qualidade de software. Pode ser que funcione, mas apenas para poucos casos e em um conjunto limitado de domínios de aplicação.[4]

14.3.2 Custo da qualidade

A discussão prossegue mais ou menos assim — *sabemos que a qualidade é importante, mas ela nos custa tempo e dinheiro — tempo e dinheiro em demasia para obter o nível de qualidade de software que realmente desejamos.* Dessa forma, o argumento parece razoável (veja os comentários de Meyer no início desta seção). Não há dúvida nenhuma de que a qualidade tem um preço, mas a falta de qualidade também tem um preço — não apenas para os usuários finais, que terão de conviver com um software com erros, mas também para a organização de software que o criou e, além de tudo, terá de fazer a manutenção. A verdadeira questão é a seguinte: *com qual custo deveríamos nos preocupar?* Para responder a essa pergunta, devemos entender tanto o custo para atingir alta qualidade quanto o custo de software de baixa qualidade.

O *custo da qualidade* inclui todos os custos necessários para a busca de qualidade ou para a execução de atividades relacionadas à qualidade, assim como os custos causados pela falta de qualidade. Para compreender esses custos, uma organização deve reunir métricas para prover uma base para o custo corrente da qualidade, identificar oportunidades para reduzir esses custos e fornecer uma base normalizada de comparação. O custo da qualidade pode ser dividido em custos associados à prevenção, avaliação e falhas.

Não tenha medo de incorrer em custos significativos de prevenção. Esteja seguro que esse investimento possibilitará um excelente retorno.

Os custos de prevenção incluem: (1) o custo de atividades de gerenciamento necessárias para planejar e coordenar todas as atividades de controle e garantia da qualidade, (2) o custo de atividades técnicas adicionais para desenvolver modelos completos de requisitos e de projeto, (3) custos de planejamento de testes e (4) o custo de todo o treinamento associado a essas atividades.

Os custos de avaliação incluem atividades para a compreensão aprofundada da condição do produto "a primeira vez através de" cada processo. Entre os exemplos de custos de avaliação, temos:

- Custo para realização de revisões técnicas (Capítulo 15) para produtos resultantes de engenharia de software.
- Custo para coleta de dados e avaliação de métricas (Capítulo 23).
- Custo para testes e depuração (Capítulos 18 a 21).

Os custos de falhas são aqueles que desapareceriam caso nenhum erro tivesse surgido antes ou depois da entrega de um produto a clientes. Esses custos podem ser subdivididos em custos de falhas internas e custos de falhas externas. Os custos de falhas internas ocorrem quando se detecta um erro em um produto antes de ele ser entregue e abrangem:

- Custo necessário para realizar retrabalhos (reparos) para corrigir um erro.
- Custo que ocorre quando retrabalhos geram, inadvertidamente, efeitos colaterais que devem ser reduzidos.
- Custos associados à reunião de métricas de qualidade que permitem a uma organização avaliar os modos de falha.

"Leva menos tempo para fazer algo certo do que explicar porque o fez errado."

H. W. Longfellow

Os custos de falhas externas estão associados a defeitos encontrados após o produto ter sido entregue ao cliente. Exemplos de custos de falhas externas são resolução de reclamações, devolução e substituição de produtos, suporte telefônico/via e-mail e custos de mão de obra associados à garantia do produto. Uma má reputação e a consequente perda de negócios é outro custo de falhas externas difícil de ser quantificado, mas bastante real. Coisas negativas acontecem quando se produz software de baixa qualidade. Em uma censura aos

4 Uma interessante discussão dos prós e contras de software "bom o suficiente" pode ser encontrada em [Bre02].

desenvolvedores de software que se recusam a considerar os custos de falhas externas, Cem Kaner [Kan95] afirma:

> Muitos dos custos de falhas externas, assim como sua reputação no mercado, são difíceis de ser quantificados e, consequentemente, muitas empresas os ignoram no cálculo das relações custo-benefício. Outros custos de falhas externas podem ser reduzidos (por exemplo, o fornecimento de suporte pós-venda mais barato e de menor qualidade ou cobrando o suporte dos clientes) sem aumentar a satisfação do cliente. Ao ignorar os custos de produtos ruins para nossos clientes, os engenheiros de qualidade encorajam tomadas de decisão relacionadas à qualidade que penalizam nossos clientes em vez de satisfazê-los.

Como era de esperar, os custos relativos para descobrir e reparar um erro ou defeito aumentam drasticamente à medida que avançamos dos custos de prevenção para custos de detecção de falhas internas e para custos de falhas externas. A Figura 14.2, fundamentada em dados coletados por Boehm e Basili [Boe01b] e ilustrada pela Cigital Inc. [Cig07], exemplifica este fenômeno.

O custo médio da indústria de software para corrigir um defeito durante a geração de código é aproximadamente US$ 977 por erro. O custo médio para corrigir o mesmo erro caso ele tenha sido descoberto durante os testes do sistema passa a ser de US$ 7.136 por erro. A Cigital Inc. [Cig07] considera uma grande aplicação em que foram introduzidos 200 erros durante a codificação:

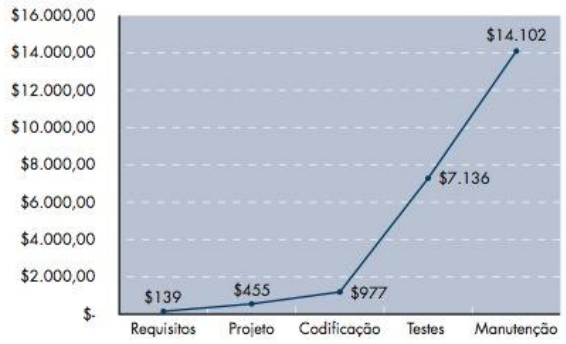
> De acordo com os dados médios do setor, o custo para descobrir e corrigir defeitos durante a fase de codificação é de US$ 977 por defeito. Portanto, o custo total para corrigir os 200 defeitos "críticos" durante essa fase (200 x US$ 977) é de aproximadamente US$ 195.400.
>
> Os dados médios do setor mostram que o custo para descobrir e corrigir defeitos durante a fase de testes é de US$ 7.136 por defeito. Nesse caso, supondo que a fase de testes do sistema tenha revelado aproximadamente 50 defeitos críticos (ou apenas 25% daqueles encontrados pela Cigital na fase de codificação), o custo para descobrir e corrigir esses defeitos (50 x US$ 7.136) teria sido de aproximadamente US$ 356.800. Isso teria resultado em 150 erros críticos sem serem detectados e corrigidos. O custo para descobrir e corrigir esses 150 defeitos remanescentes na fase de manutenção (150 x US$ 14.102) teria sido de US$ 2.115.300. Portanto, o custo total para descobrir e corrigir os 200 defeitos após a fase de codificação teria sido de US$ 2.472.100 (US$ 2.115.300 + US$ 356.800).

Mesmo que sua organização de software possua custos equivalentes à metade da média do setor (a maioria não tem a mínima ideia de quanto são seus custos!), a economia de custos associados a atividades iniciais de controle com garantia da qualidade (conduzidas durante a análise de requisitos e projeto) é considerável.

FIGURA 14.2

Custo relativo para correção de erros e defeitos

***Fonte:* Adaptado de [Boe01b]**



14.3.3 Riscos

No Capítulo 1 foi escrito que "as pessoas apostam seus empregos, comodidades, segurança, entretenimento, decisões e as próprias vidas em software. Esperamos que elas estejam certas". A implicação disso é que software de baixa qualidade aumenta os riscos tanto para o desenvolvedor quanto para o usuário final. Na subseção anterior, discutimos um desses riscos (custo). Mas o lado negativo de aplicações mal projetadas e implementadas nem sempre resulta apenas em altos custos e mais tempo. Um exemplo [Gag04] extremo pode servir como ilustração.

Ao longo do mês de novembro de 2000, em um hospital no Panamá, 28 pacientes receberam doses maciças de raio gama durante tratamento para uma série de tipos de câncer. Nos meses seguintes, 5 desses pacientes morreram por contaminação radioativa e outros 15 desenvolveram sérias complicações. O que provocou essa tragédia? Um pacote de software, desenvolvido por uma companhia americana, foi modificado por técnicos do hospital para calcular doses de radiação para cada paciente.

Os três médicos panamenhos com especialização em física, que "ajustaram" o software para aumentar a capacidade funcional, foram processados por homicídio doloso, mas sem premeditação. A empresa americana está enfrentando litígios graves em dois países. Gage e McCormick comentam:

> Não se trata de um conto com fundo moral para técnicos em medicina, muito embora eles possam lutar para manter-se fora da prisão caso tenham interpretado ou usado uma tecnologia de forma errada. Não se trata também de um conto de como seres humanos podem ser feridos ou sofrer algo ainda mais grave por software mal projetado ou documentado, embora existam muitos exemplos que o comprovem. Trata-se de um alerta para qualquer criador de programas de computador: a qualidade de software é importante, as aplicações têm de ser infalíveis e — independentemente de estarem embutidas no motor de um carro, em um braço mecânico em uma fábrica ou em um aparelho médico em um hospital — código mal empregado pode matar.

A baixa qualidade induz a riscos, alguns muito sérios.

14.3.4 Negligência e responsabilidade civil

A história é bastante comum. Um órgão do governo ou empresa contrata uma grande consultoria ou uma importante empresa desenvolvedora de software para analisar os requisitos e então projetar e construir um "sistema" baseado em software para apoiar alguma atividade de importante. O sistema poderia oferecer suporte a uma importante função corporativa (por exemplo, administração de aposentadorias) ou alguma função governamental (por exemplo, administração do sistema de saúde ou de segurança do território nacional).

O trabalho inicia com as melhores das intenções de ambas as partes, mas na época em que o sistema é entregue, as coisas não andam bem. O sistema é lento, não fornece os recursos e funções desejados, é suscetível a erros e não tem a aprovação do usuário. Segue-se um litígio. Na maioria dos casos, o cliente alega que o desenvolvedor foi negligente (na maneira de aplicação de práticas de software) e, portanto, não tem direito a receber seu pagamento. Normalmente, o desenvolvedor alega que o cliente mudou repetidamente os requisitos e subverteu a parceria de desenvolvimento de outras formas. Em todos os casos, a questão é a qualidade do sistema entregue.

14.3.5 Qualidade e segurança

À medida que a criticabilidade das aplicações e dos sistemas baseados na Internet aumenta, a segurança da aplicação tem se tornado cada vez mais importante. Ou seja, software que não apresente alta qualidade é mais fácil de ser copiado ("pirateado") e, como consequência, o software de baixa qualidade pode aumentar indiretamente o risco de segurança assim como todos os problemas e custos associados.

Numa entrevista à revista *ComputerWorld,* o autor e especialista em segurança Gary McGraw comenta [Wil05]:

> A segurança de software se relaciona inteira e completamente à qualidade. Devemos nos preocupar com a segurança, a confiabilidade, a disponibilidade e a dependência — nas fases inicial, de projeto, de arquitetura, de testes e de codificação, ao longo de todo o ciclo de vida [qualidade] de um software. Até mesmo pessoas cientes do problema de segurança têm se concentrado em coisas relacionadas ao ciclo de vida do software. O quanto antes descobrirmos um problema de software, melhor. E existem dois tipos de problemas de software. O primeiro são os bugs, que são problemas de implementação. Os demais são falhas de software — problemas arquiteturais do projeto. As pessoas prestam muita atenção aos bugs, mas não o suficiente às falhas.

Para construir um sistema seguro, temos de focar na qualidade, e esse foco deve iniciar durante o projeto. Os conceitos e os métodos discutidos na Parte 2 deste livro nos conduzem a uma arquitetura de software que reduz "falhas". Ao eliminar falhas arquiteturais (melhorando, consequentemente, a qualidade do software), faremos com que fique muito mais difícil a cópia ilegal do software.

14.3.6 O impacto das ações administrativas

A qualidade de software é normalmente tão influenciada pelas decisões administrativas quanto pelas decisões técnicas. Até mesmo as melhores práticas de engenharia de software podem ser subvertidas por decisões de negócios inadequadas e ações questionáveis de gerenciamento de projeto.

Na Parte 4 deste livro discutiremos o gerenciamento de projetos no contexto da gestão da qualidade. Quando cada tarefa de projeto é iniciada, um líder de projeto tomará decisões que podem ter um impacto significativo sobre a qualidade do produto.

Decisões de estimativas. Conforme citamos no Capítulo 26, raramente uma equipe de software pode se dar ao luxo de fornecer uma estimativa para um projeto *antes* das datas de entrega serem estabelecidas e um orçamento geral ser especificado. Em vez disso, a equipe realiza um "exame de sanidade" para garantir que as datas de entrega são racionais. Em muitos casos, existe uma enorme pressão de colocação do produto no mercado que força uma equipe a aceitar as datas impraticáveis de entrega. Como resultado, são realizados atalhos, as atividades que produzem um software de maior qualidade talvez sejam deixadas de lado e, assim, a qualidade do produto sofre as consequências. Se uma data de entrega for absurda, é importante manter-se firme. Explique por que você precisa de mais tempo ou, alternativamente, sugira um subconjunto de funcionalidades que possa ser entregue (com alta qualidade) no tempo alocado.

Decisões de cronograma. Quando um cronograma de projeto de software é estabelecido (Capítulo 27), as tarefas são sequenciadas tomando-se como base as dependências. Por exemplo, como o componente **A** depende do processamento que ocorre nos componentes **B, C** e **D**, o componente **A** não pode ser agendado para testes até que os componentes **B**, **C** e **D** sejam completamente testados. O cronograma de um projeto deve refletir essa situação. Mas se o tempo for muito curto e **A** tem de estar disponível para outros testes críticos, talvez se opte por testar **A** sem seus componentes subordinados (que estão ligeiramente atrasados em relação ao cronograma), de modo a torná-lo disponível para outros testes que devem ser feitos antes da entrega. Afinal de contas, o prazo final se aproxima. Dessa forma, **A** pode ter defeitos que estão ocultos e que seriam descobertos muito mais tarde. A qualidade sofre as consequências.

Decisões orientadas a riscos. A administração de riscos (Capítulo 28) é um dos atributos fundamentais de um projeto de software bem-sucedido. Precisamos realmente saber o que poderia dar errado e estabelecer um plano de contingência caso isso aconteça. Um número muito grande de equipes de software prefere um otimismo cego, estabelecendo um cronograma de desenvolvimento sob a hipótese de que nada vai dar errado. Pior ainda, eles não têm um método para lidar com as coisas que dão errado. Consequentemente, quando um risco se torna realidade, reina o caos e, à medida que o grau de loucura aumenta, o nível de qualidade invariavelmente cai.

O dilema da qualidade de software pode ser mais bem sintetizado enunciando-se a Lei de Meskimen — *Nunca há tempo para fazer a coisa certa, mas sempre há tempo para fazê-la de novo.* Meu conselho: tomar o tempo necessário para fazer certo da primeira vez quase nunca é uma decisão errada.

14.4 Alcançando a Qualidade de Software

A qualidade de software não aparece simplesmente do nada. Ela é o resultado de um bom gerenciamento de projeto e uma prática consistente de engenharia de software. O gerenciamento e a prática são aplicados no contexto de quatro grandes atividades que ajudam uma equipe de software a atingir alto padrão de qualidade de software: métodos de engenharia de software, técnicas de gerenciamento de projeto, ações de controle de qualidade e garantia da qualidade de software.

14.4.1 Métodos de engenharia de software

O que é preciso fazer para afetar a qualidade de forma positiva?

Se nossa expectativa é construir software de alta qualidade, temos de entender o problema a ser resolvido. Temos também de ser capazes de criar um projeto que seja adequado ao problema e, ao mesmo tempo, apresente características que levem a um software com as dimensões e fatores de qualidade discutidos na Seção 14.2.

Na Parte 2 deste livro, apresentamos uma ampla gama de conceitos e métodos capazes de levar a um entendimento relativamente completo do problema e a um projeto abrangente que estabeleça uma base sólida para a atividade de construção. Se aplicarmos esses conceitos e adotarmos os métodos de análise e projeto apropriados, a probabilidade de criarmos software de alta qualidade aumentará substancialmente.

14.4.2 Técnicas de gerenciamento de software

O impacto de decisões de gerenciamento inadequadas sobre a qualidade de software foi discutido na Seção 14.3.6. As implicações são claras: se (1) um gerente de projeto usar estimativas para verificar que as datas de entrega são plausíveis, (2) as dependências de cronograma forem entendidas e a equipe resistir à tentação de usar atalhos, (3) o planejamento de riscos for conduzido de modo que problemas não gerem caos, a qualidade do software será afetada de forma positiva.

Além disso, o plano de projeto deve incluir técnicas explícitas para gerenciamento de mudanças e qualidade. Técnicas que levam a práticas ótimas de gerenciamento de projeto são discutidas na Parte 4 do livro.

14.4.3 Controle de qualidade

O que é controle de qualidade de software?

O controle de qualidade engloba um conjunto de ações de engenharia de software que ajudam a garantir que cada produto resultante atinja suas metas de qualidade. Os modelos são revistos de modo a garantir que sejam completos e consistentes. O código poderia ser inspecionado de modo a revelar e corrigir erros antes de os testes começarem. Aplica-se uma série de etapas de teste para descobrir erros na lógica de processamento, na manipulação de dados e na comunicação da interface. Uma combinação de medições e realimentação (feedback) permite a uma equipe de software ajustar o processo quando qualquer um desses produtos resultantes deixe de atender às metas estabelecidas para a qualidade. As atividades de controle de qualidade são discutidas em detalhe ao longo do restante da Parte 3 deste livro.

14.4.4 Garantia da qualidade

A garantia da qualidade estabelece a infraestrutura que suporta métodos sólidos de engenharia de software, gerenciamento racional de projeto e ações de controle de qualidade — todos fundamentais para a construção de software de alta qualidade. Além disso, a garantia

WebRef

Links úteis para recursos sobre garantia da qualidade de software podem ser encontrados em **www.niwotridge.com/Resources/PMSWEResources/SoftwareQualityAssurance.htm**.

da qualidade consiste em um conjunto de funções de auditoria e de relatórios que possibilita uma avaliação da efetividade e da completude das ações de controle de qualidade. O objetivo da garantia da qualidade é fornecer ao pessoal técnico e administrativo os dados necessários para ser informados sobre a qualidade do produto, ganhando, portanto, entendimento e confiança de que as ações para atingir a qualidade desejada do produto estão funcionando. Obviamente, se os dados fornecidos pela garantia da qualidade identificarem problemas, é responsabilidade do gerenciamento tratar esses problemas e aplicar os recursos necessários para resolver os problemas de qualidade. A garantia da qualidade de software é discutida em detalhe no Capítulo 16.

14.5 Resumo

A preocupação com a qualidade de sistemas de software cresceu à medida que o software passou a se tornar cada vez mais integrado em cada aspecto da vida cotidiana. Mas é difícil desenvolver uma descrição completa sobre qualidade de software. Neste capítulo, a qualidade foi definida como uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam.

Foi proposta uma grande variedade de dimensões e fatores para qualidade de software ao longo dos anos. Todos tentam definir um conjunto de características que, se atingidas, levarão a um software de alta qualidade. Os fatores de qualidade de McCall e da ISO 9126 estabelecem características como confiabilidade, usabilidade, facilidade de manutenção, funcionalidade e portabilidade como indicadores de que a qualidade existe.

Todas as organizações envolvidas com software deparam com o dilema da qualidade de software. Em essência, todos querem construir sistemas de alta qualidade, mas o tempo e o esforço necessários para produzir um software "perfeito" simplesmente não existem em um mundo orientado ao mercado. A pergunta passa a ser: devemos construir software " bom o suficiente"? Embora muitas empresas façam exatamente isso, há um grande porém que deve ser considerado.

Independentemente da abordagem escolhida, a qualidade tem, efetivamente, um custo que pode ser discutido em termos de prevenção, avaliação e falha. Os custos de prevenção incluem todas as ações de engenharia de software que são desenvolvidas para, em primeiro lugar, evitar defeitos. Os custos de avaliação estão associados àquelas ações que avaliam os artefatos resultantes para determinar sua qualidade. Os custos de falhas englobam o preço de falhas internas e os efeitos externos que a má qualidade gera.

A qualidade de software é atingida por meio da aplicação de métodos de engenharia de software, práticas administrativas consistentes e controle de qualidade completo — todos suportados por uma infraestrutura de garantia da qualidade de software. Nos capítulos seguintes, o controle e a garantia da qualidade são discutidos com maior nível de detalhamento.

Problemas e Pontos a Ponderar

14.1. Descreva como você avaliaria a qualidade de uma universidade antes de se candidatar a ela. Quais fatores seriam importantes? Quais seriam críticos?

14.2. Garvin [Gar84] descreve cinco visões diferentes de qualidade. Dê um exemplo de cada uma delas usando um ou mais produtos eletrônicos conhecidos com os quais você esteja familiarizado.

14.3. Usando a definição de qualidade de software proposta na Seção 14.2, você acredita que seja possível criar um produto útil que gere valor mensurável sem usar um processo eficaz? Justifique sua resposta.

14.4. Acrescente duas outras perguntas a cada uma das dimensões de qualidade de Garvin apresentadas na Seção 14.2.1.

14.5. Os fatores de qualidade de McCall foram desenvolvidos durante a década de 1970. Praticamente todos os aspectos da computação mudaram drasticamente desde essa época e, mesmo assim, os fatores de McCall ainda se aplicam a software moderno. Você seria capaz de tirar alguma conclusão com base nesse fato?

14.6. Usando os subatributos citados para o fator de qualidade da ISO 9126, "facilidade de manutenção", na Seção 14.2.3, desenvolva um conjunto de perguntas que explore se esses atributos estão presentes ou não. Siga o exemplo mostrado na Seção 14.2.4.

14.7. Descreva o dilema da qualidade de software com suas próprias palavras.

14.8. O que é software "bom o suficiente"? Cite uma empresa específica e produtos específicos que você acredita terem sido desenvolvidos usando essa filosofia.

14.9. Considerando cada um dos quatro aspectos do custo da qualidade, qual você acredita ser o mais caro e a razão para sua resposta?

14.10. Faça uma busca na Internet e encontre três outros exemplos de "riscos" para o público que podem ser diretamente atribuídos à baixa qualidade de software. Considere a opção de iniciar sua pesquisa em **http://catless.ncl.ac.uk/risks.**

14.11. *Qualidade* e *segurança* são a mesma coisa? Explique.

14.12. Explique por que muitos de nós continuamos a viver da lei de Meskimen. Como isso se aplica a um negócio de software?

Leituras e Fontes de Informação Complementares

Conceitos básicos sobre qualidade de software são considerados em livros como os de Henry e Hanlon (*Software Quality Assurance,* Prentice-Hall, 2008), Khan e seus colegas (*Software Quality: Concepts and Practice,* Alpha Science International, Ltd., 2006), O'Regan (*A Practical Approach to Software Quality,* 2002) e Daughtrey (*Fundamental Concepts for the Software Quality Engineer,* ASQ Quality Press, 2001).

Duvall e seus colegas (*Continuous Integration: Improving Software Quality and Reducing Risk,* Addison-Wesley, 2007), Tian (*Software Quality Engineering,* Wiley-IEEE Computer Society Press, 2005), Kandt (*Software Engineering Quality Practices,* Auerbach, 2005), Godbole (*Software Quality Assurance: Principles and Practice,* Alpha Science International, Ltd., 2004) e Galin (*Software Quality Assurance: From Theory to Implementation,* Addison-Wesley, 2003) apresentam tratados detalhados sobre SQA. A garantia de qualidade no contexto do processo ágil é considerada por Stamelos e Sfetsos (*Agile Software Development Quality Assurance,* IGI Global, 2007).

Projeto consistente leva a qualidade de software elevada. Jayasawal e Patton (*Design for Trustworthy Software,* Prentice-Hall, 2006) e Ploesch (*Contracts, Scenarios and Prototypes,* Springer, 2004) discutem ferramentas e técnicas para desenvolver software "robusto".

A medição é um importante componente da engenharia de qualidade de software. Ejiogu (*Software Metrics: The Discipline of Software Quality,* BookSurge Publishing, 2005), Kan (*Metrics and Models in Software Quality Engineering,* Addison-Wesley, 2002) e Nance e Arthur (*Managing Software Quality,* Springer, 2002) discutem importantes métricas e modelos relacionados com a qualidade. Os aspectos da qualidade de software orientados a equipes são considerados por Evans (*Achieving Software Quality through Teamwork,* Artech House Publishers, 2004).

Uma ampla gama de fontes de informação sobre qualidade de software se encontra à disposição na Internet. Uma lista atualizada de referências relevantes para a qualidade de software pode ser encontrada no site **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**.