



CAPÍTULO

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 **24** 25 26

Gerenciamento de qualidade

Objetivos

O objetivo deste capítulo é apresentar o gerenciamento de qualidade e a medição de software. Com a leitura deste capítulo você:

- conhecerá o processo de gerenciamento de qualidade e saberá por que o planejamento de qualidade é importante;
- saberá que a qualidade do software é afetada pelo processo usado em seu desenvolvimento;
- estará ciente da importância dos padrões no processo de gerenciamento de qualidade e saberá como os padrões são usados a fim de garantir a qualidade;
- compreenderá de que maneira revisões e inspeções são usadas como um mecanismo de garantia de qualidade de software;
- compreenderá de que maneira a medição pode ser útil na avaliação de alguns atributos de qualidade de software e as limitações atuais da medição de software.

- Conteúdo**
- 24.1** Qualidade de software
 - 24.2** Padrões de software
 - 24.3** Revisões e inspeções
 - 24.4** Medições e métricas de software

Os problemas com a qualidade de software foram inicialmente descobertos na década de 1960 com o desenvolvimento do primeiro grande sistema de software e continuaram a incomodar a engenharia de software ao longo do século XX. O software entregue era lento e pouco confiável, difícil de manter e de reusar. Em resposta à insatisfação com aquela situação, passaram a ser adotadas técnicas formais de gerenciamento de qualidade do software, as quais foram desenvolvidas a partir dos métodos usados na indústria manufatureira. Essas técnicas de gerenciamento de qualidade, em conjunto com novas tecnologias de software e melhores testes de software, conduziram a melhorias significativas no nível geral de qualidade de software.

O gerenciamento de qualidade de software para sistemas de software tem três principais preocupações:

1. No nível organizacional, o gerenciamento de qualidade está preocupado com o estabelecimento de um *framework* de processos organizacionais e padrões que levem a softwares de alta qualidade. Isso significa que a equipe de gerenciamento de qualidade deve assumir a responsabilidade de definir os processos de desenvolvimento do software que serão usados e os padrões que devem ser usados no software, bem como a documentação relacionada, incluindo os requisitos de sistema, projeto e código.
2. No nível de projeto, o gerenciamento de qualidade envolve a aplicação de processos específicos de qualidade, verificando que os processos planejados foram seguidos, e a garantia de que as saídas de projeto estejam em conformidade com os padrões aplicáveis ao projeto.

- 3.** No nível de projeto, o gerenciamento de qualidade também está preocupado com o estabelecimento de um plano de qualidade. O plano de qualidade deve definir as metas de qualidade para o projeto e quais processos e padrões devem ser usados.

Os termos ‘garantia de qualidade’ e ‘controle de qualidade’ são amplamente usados na indústria manufatureira. A garantia de qualidade (QA, do inglês *quality assurance*) é a definição de processos e padrões que devem conduzir a produtos de alta qualidade e a introdução de processos de qualidade na fabricação. O controle de qualidade é a aplicação desses processos de qualidade visando eliminar os produtos que não atingiram o nível de qualidade exigido.

Na indústria de software, diferentes empresas e setores industriais interpretam a garantia de qualidade e controle de qualidade de maneiras diferentes. Às vezes, garantia de qualidade significa simplesmente a definição de procedimentos, processos e padrões que visam reforçar que a qualidade de software seja atingida. Em outros casos, a garantia de qualidade também inclui todo o gerenciamento de configuração, atividades de verificação e validação aplicadas após o produto ter sido entregue por uma equipe de desenvolvimento. Neste capítulo, utilizo o termo ‘garantia de qualidade’ para incluir a verificação e validação e os processos de verificação se os procedimentos de qualidade foram aplicados corretamente. O termo ‘controle de qualidade’ foi evitado, pois não é amplamente usado na indústria de software.

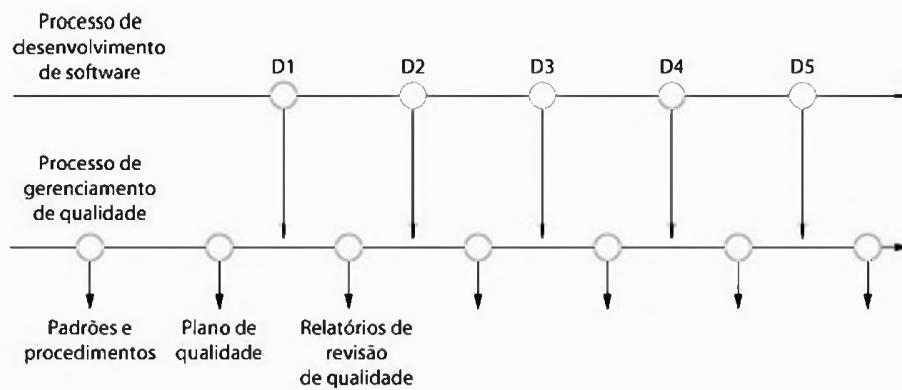
A equipe QA é responsável, na maioria das empresas, por gerenciar o processo de teste de *release*. Conforme discutido no Capítulo 8, isso significa que eles gerenciam os testes de software antes que este seja liberado para os clientes. Eles são responsáveis por verificar se os testes de sistema proporcionam cobertura dos requisitos e mantêm registros adequados do processo de teste. No Capítulo 8 discuti teste de *release* e, portanto, neste capítulo não discuto esse aspecto de garantia de qualidade.

O gerenciamento de qualidade fornece uma verificação independente do processo de desenvolvimento de software. O processo de gerenciamento de qualidade verifica os entregáveis de projeto para garantir que eles sejam consistentes com os padrões e objetivos organizacionais (Figura 24.1). A equipe de QA deve ser independente da equipe de desenvolvimento para que eles tenham uma visão objetiva do software. Isso lhes permite reportar sobre a qualidade de software sem sofrer influências de questões de desenvolvimento de software.

Idealmente, a equipe de gerenciamento de qualidade não deve ser associada a qualquer grupo particular de desenvolvimento, mas deve ter a responsabilidade de toda a organização para o gerenciamento de qualidade. Eles devem ser independentes e reportar para a gerência acima do nível de gerente de projeto. A razão para isso é que os gerentes de projeto precisam manter o cronograma e o orçamento de projeto. Se houver problemas, eles podem correr o risco de comprometer a qualidade do produto, em nome do cumprimento do cronograma, por exemplo. Uma equipe de gerenciamento de qualidade independente garante que os objetivos organizacionais de qualidade não se comprometam com orçamentos curtos e considerações de cronograma. No entanto, em pequenas empresas, isso é praticamente impossível. O gerenciamento de qualidade e desenvolvimento de software está inevitavelmente interligado com pessoas com responsabilidades de desenvolvimento e de qualidade.

O planejamento de qualidade é o processo de desenvolvimento de um plano de qualidade para um projeto. O plano de qualidade deve estabelecer as qualidades desejadas para o software e descrever como elas devem ser avaliadas. Portanto, define o que o software de ‘alta qualidade’ realmente significa para um determinado sistema. Sem essa definição, os engenheiros podem fazer suposições algumas vezes conflitantes sobre quais atributos de produto refletem as mais importantes características de qualidade. O planejamento formal de qualidade é parte integrante dos processos de desen-

Figura 24.1 Gerenciamento de qualidade e desenvolvimento de software



volvimento baseado em planos. No entanto, métodos ágeis adotam uma abordagem menos formal para o gerenciamento de qualidade.

Humphrey (1989), em seu livro clássico sobre o gerenciamento de software, sugere uma estrutura preliminar para um plano de qualidade, a qual inclui:

1. *Introdução ao produto.* Uma descrição do produto, seu mercado pretendido e as expectativas de qualidade do produto.
2. *Planos de produto.* As datas críticas de *release* e responsabilidades para o produto, junto com os planos para a distribuição e prestação de serviço do produto.
3. *Descrições de processo.* Os processos de desenvolvimento e serviço são padrões que devem ser usados para o gerenciamento e desenvolvimento de produto.
4. *Metas de qualidade.* As metas de qualidade e planos para o produto, incluindo uma identificação e uma justificativa para os atributos críticos de qualidade do produto.
5. *Riscos e gerenciamento de riscos.* Os riscos mais importantes que podem afetar a qualidade do produto e as ações que devem ser tomadas ao lidar com eles.

Os planos de qualidade, os quais são desenvolvidos como parte do processo geral de planejamento de projeto, diferem em detalhes, dependendo do tamanho e do tipo de sistema que está sendo desenvolvido. No entanto, ao escrever planos de qualidade, você deve tentar mantê-los o mais breve possível. Se o documento for muito extenso, as pessoas não o lerão, e isso reduz o propósito de se produzir o plano de qualidade.

Algumas pessoas pensam que a qualidade de software pode ser alcançada por meio de processos prescritivos, baseados em padrões organizacionais e procedimentos de qualidade associados que verificam que esses padrões serão seguidos pela equipe de desenvolvimento de software. Seu argumento é que os padrões incorporam as boas práticas de engenharia de software e que seguirlas levará a produtos de alta qualidade. Na prática, contudo, existe muito mais no gerenciamento de qualidade do que apenas padrões e a burocracia associada para garantir que sejam seguidos.

Padrões e processos são importantes, mas os gerentes de qualidade também devem ter como objetivo desenvolver uma ‘cultura de qualidade’ em que todos os responsáveis pelo desenvolvimento de software estejam comprometidos em alcançar um alto nível de qualidade de produto. Eles devem incentivar as equipes a assumirem responsabilidade pela qualidade de seu trabalho e a desenvolverem novas abordagens para a melhoria de qualidade. Apesar de os padrões e procedimentos serem a base do gerenciamento de qualidade, os bons gerentes de qualidade reconhecem haver aspectos intangíveis para a qualidade de software (elegância, legibilidade etc.) que não podem ser incorporados em padrões. Eles devem apoiar as pessoas que estão interessadas em aspectos intangíveis de qualidade e incentivar o comportamento profissional de todos os membros de equipe.

O gerenciamento formal de qualidade é particularmente importante para as equipes que estão desenvolvendo sistemas de longa vida e grande porte que levam vários anos para se desenvolver. A documentação de qualidade é um registro do que foi feito por cada subgrupo no projeto. Isso ajuda as pessoas a verificarem se tarefas importantes não foram esquecidas ou se um grupo não fez suposições incorretas sobre o trabalho de outros grupos. A documentação de qualidade também é um meio de comunicação durante a vida útil de um sistema. Ela permite que os grupos responsáveis pela evolução de sistema possam rastrear os testes e as verificações implementadas pela equipe de desenvolvimento.

Para sistemas menores, o gerenciamento de qualidade também é importante, mas pode adotar uma abordagem mais informal. Não é necessária muita papelada porque uma equipe pequena de desenvolvimento também pode comunicar-se informalmente. O ponto mais importante do gerenciamento de qualidade para o desenvolvimento de pequenos sistemas é estabelecer uma cultura de qualidade e assegurar que todos os membros de equipe tenham uma abordagem positiva da qualidade de software.

24.1 Qualidade de software

Os fundamentos do gerenciamento de qualidade foram estabelecidos pela indústria manufatureira em um esforço para melhorar a qualidade dos produtos em produção. Como parte disso, eles desenvolveram uma definição de ‘qualidade’, baseada na conformidade com uma especificação detalhada (CROSBY, 1979) e na noção de tolerâncias. A suposição subjacente foi a de que os produtos poderiam ser completamente especificados e poderiam ser estabelecidos procedimentos para avaliar um produto fabricado de acordo com suas especificações. Naturalmen-

te, os produtos nunca cumprirão todas as especificações, então se admitiu alguma tolerância. Se o produto estava 'quase certo', ele era classificado como aceitável.

A qualidade de software não é diretamente comparável à qualidade na manufatura. A ideia de tolerâncias não é aplicável aos sistemas digitais e, pelas razões apresentadas a seguir, pode ser impossível concluir objetivamente se um sistema de software cumpre ou não suas especificações:

1. Conforme discutido no Capítulo 4, no qual abordo a engenharia de requisitos, é difícil escrever especificações de software completas e precisas. Os clientes e desenvolvedores de software podem interpretar os requisitos de maneiras diferentes e pode ser impossível chegar a um acordo sobre se o software cumpre ou não suas especificações.
2. Geralmente, as especificações integram requisitos de várias classes de *stakeholders*. Esses requisitos são, inevitavelmente, um compromisso e podem não incluir os requisitos de todos os grupos de *stakeholders*. Os *stakeholders* excluídos podem perceber o sistema como um sistema de baixa qualidade, mesmo que este implemente os requisitos acordados.
3. É impossível medir determinadas características de qualidade diretamente (por exemplo, manutenibilidade), assim, elas não podem ser especificadas de forma não ambígua. As dificuldades de medição são discutidas na Seção 24.4.

Devido a esses problemas, a avaliação da qualidade de software é um processo subjetivo, em que a equipe de gerenciamento de qualidade precisa usar seu julgamento para decidir se foi alcançado um nível aceitável de qualidade. Ela precisa considerar se o software é adequado para sua finalidade ou não. Trata-se de responder a perguntas sobre as características do sistema. Por exemplo:

1. Durante o processo de desenvolvimento os padrões de programação e documentação foram seguidos?
2. O software foi devidamente testado?
3. O software é suficientemente confiável para ser colocado em uso?
4. O desempenho do software é aceitável para uso normal?
5. O software é útil?
6. O software é bem estruturado e compreensível?

Existe uma suposição geral no gerenciamento de qualidade de software de que os testes de sistema serão baseados em seus requisitos. A decisão sobre se o software oferece ou não a funcionalidade necessária deve basear-se nos resultados desses testes. Por isso, a equipe de QA deve revisar os testes que foram desenvolvidos e analisar os registros de testes para verificar se os testes foram devidamente realizados. Em algumas organizações, a equipe de gerenciamento de qualidade é responsável pelos testes de sistema, mas, às vezes, essa responsabilidade é dada para um grupo separado de testes.

A qualidade subjetiva de um sistema de software baseia-se em grande parte em suas características não funcionais. Isso reflete a experiência prática do usuário — se a funcionalidade do software não é a esperada, os usuários frequentemente apenas contornam esse problema e encontram outras maneiras de fazer o que querem. No entanto, se o software for muito lento ou não confiável, será praticamente impossível aos usuários atingirem seus objetivos.

Portanto, entendemos que a qualidade de software não implica apenas se a funcionalidade de software foi corretamente implementada, mas também depende dos atributos não funcionais de sistema. Boehm et al. (1978) sugeriram que havia quinze atributos importantes de qualidade de software, como mostrado na Tabela 24.1. Esses atributos estão relacionados com a confiança, a usabilidade, a eficiência e a manutenibilidade de software. Como já discutido no Capítulo 11, geralmente os atributos de confiança são os atributos de qualidade mais importantes de um sistema. No entanto, o desempenho do software também é muito importante. Os usuários rejeitarão o software que for muito lento.

É impossível que algum sistema seja otimizado em todos esses atributos — por exemplo, melhorar a robustez pode levar à perda de desempenho. O plano de qualidade, portanto, deve definir os atributos de qualidade mais importantes para o software que está sendo desenvolvido. Pode ser que a eficiência seja crítica e outros fatores tenham de ser sacrificados para obtenção disso. Se no plano de qualidade você estabeleceu que um atributo é crítico, os engenheiros que trabalham no desenvolvimento podem cooperar para esse atributo. O plano deve incluir também uma definição de processo de avaliação de qualidade. Isso deve ser uma maneira de avaliar se alguma qualidade, como manutenibilidade ou robustez, está presente no produto.

Um pressuposto do gerenciamento de qualidade de software é que a qualidade do software é diretamente relacionada à qualidade do processo de desenvolvimento de software. Isso vem novamente da fabricação de sistemas, em que a qualidade de produto é intimamente relacionada ao processo de produção.

Tabela 24.1 Atributos de qualidade de software

Segurança	Compreensibilidade	Portabilidade
Proteção	Testabilidade	Usabilidade
Confiabilidade	Adaptabilidade	Reusabilidade
Resiliência	Modularidade	Eficiência
Robustez	Complexidade	Capacidade de aprendizado

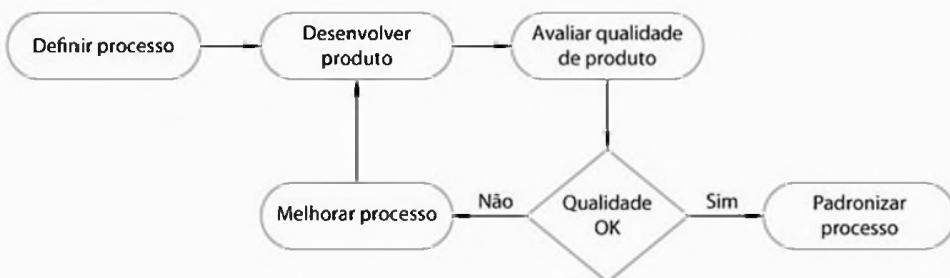
Um processo de fabricação envolve configurar e operar as máquinas envolvidas no processo. Uma vez que as máquinas estão funcionando corretamente, a qualidade do produto segue normalmente. Você mede a qualidade do produto e altera o processo até atingir o nível de qualidade requerida. A Figura 24.2 ilustra essa abordagem baseada em processos para atingir a qualidade do produto.

Existe uma clara ligação entre a qualidade de processo e de produto na manufatura porque o processo é relativamente fácil de ser padronizado e monitorado. Uma vez que sistemas de manufatura sejam calibrados, eles podem ser executados várias vezes para produzir produtos de alta qualidade. No entanto, o software não é manufaturado — ele é criado. No desenvolvimento de software a relação entre a qualidade de processo e de produto é mais complexa. O desenvolvimento de software é um processo criativo, em vez de mecânico, portanto, a influência de competências e experiências individuais é significativa. Os fatores externos, como a novidade de uma aplicação ou pressão comercial para um *release* anterior do produto, também afetam a qualidade do produto, independentemente do processo usado.

Não há dúvida de que o processo de desenvolvimento usado tenha uma influência significativa sobre a qualidade do software e que bons processos são mais suscetíveis de conduzir o software de boa qualidade. O gerenciamento e a melhoria de qualidade de processo podem gerar softwares com menos defeitos. Contudo, é difícil avaliar os atributos de qualidade de software, como manutenibilidade, sem usar o software por um longo período. Portanto, é difícil dizer como as características de processo influenciam esses atributos. Além disso, por causa do papel do projeto e da criatividade no processo de software, a padronização de processos pode, por vezes, sufocar a criatividade, o que pode gerar softwares com menos qualidade.

24.2 Padrões de software

Os padrões de software desempenham um papel muito importante no gerenciamento de qualidade de software. Como já discutido, uma parte importante da garantia de qualidade é a definição ou seleção de padrões que devem ser aplicados no processo de desenvolvimento de software ou produtos de software. Como parte desse processo de QA, devem ser escolhidas ferramentas e métodos para suportar o uso desses padrões. Uma vez que os padrões foram selecionados para uso, processos específicos de projeto devem ser definidos para monitorar o uso dos padrões e verificar se eles foram seguidos.

Figura 24.2 Qualidade baseada em processos

Os padrões de software são importantes por três razões:

1. Capturam sabedoria, que é valiosa para a organização. Eles são baseados em conhecimentos sobre a prática do que é melhor ou mais adequado para a empresa. Muitas vezes, esse conhecimento é adquirido após bastante tentativa e erro. Inseri-lo em um padrão ajuda a empresa a reusar essa experiência e evitar erros anteriormente cometidos.
2. Fornecem um *framework* para a definição do significado de 'qualidade' em uma determinada organização. Como já discutido, a qualidade de software é subjetiva e ao usar padrões você estabelece uma base para decidir se o nível de qualidade exigido foi atingido. Naturalmente, isso depende do estabelecimento de padrões que refletem as expectativas do usuário com relação a confiança, usabilidade e desempenho do software.
3. Ajudam a dar continuidade ao trabalho realizado por uma pessoa, quando retomado e continuado por outra. Os padrões asseguram que todos os engenheiros dentro de uma organização adotem as mesmas práticas. Consequentemente, o esforço de aprendizagem requerido ao iniciar um novo trabalho é reduzido.

Existem dois tipos de padrões de engenharia de software que podem ser definidos e usados no gerenciamento de qualidade de software:

1. *Padrões de produto*. Aplicam-se ao produto de software que está sendo desenvolvido. Eles incluem padrões de documentos — como a estrutura dos documentos de requisitos; padrões de documentação — como um cabeçalho de comentário padrão para uma definição de classe de objeto; e padrões de codificação — os quais definem como uma linguagem de programação deve ser usada.
2. *Padrões de processo*. Definem os processos que devem ser seguidos durante o desenvolvimento de software. Eles devem encapsular as boas práticas de desenvolvimento. Os padrões de processo podem incluir definições de especificação, projeto e processos de validação, ferramentas de suporte do processo e uma descrição dos documentos que devem ser escritos durante esses processos.

Os padrões devem entregar valor, sob a forma de maior qualidade de produto. Não há razão para definir padrões que sejam caros em termos de tempo e de esforço para serem aplicados e que gerem poucas melhorias na qualidade. Os padrões de produto precisam ser projetados para poderem ser aplicados e verificados de forma efetiva e os padrões de processos devem incluir uma definição de processos que verifique se os padrões de produto foram seguidos.

Geralmente, o desenvolvimento de padrões internacionais de engenharia de software é um processo prolongado, em que os interessados no padrão se encontram, produzem material para discussões e, finalmente, chegam a um acordo sobre o padrão. Organismos nacionais e internacionais, como DoD dos Estados Unidos, ANSI, BSI, OTAN e IEEE apoiam a produção de padrões. Tratam-se de padrões gerais que podem ser aplicados em uma gama de projetos. Organismos como a OTAN e outras organizações de defesa podem exigir que seus próprios padrões sejam usados em seus contratos de desenvolvimento com empresas de software.

Foram desenvolvidos padrões nacionais e internacionais, discutindo a terminologia de engenharia de software e linguagens de programação como Java e C++, notações, como gráficos, símbolos, procedimentos para derivar e escrever requisitos de software, procedimentos de garantia de qualidade e processos de verificação e validação de software (IEEE, 2003). Os padrões mais especializados, como IEC 61508 (IEC, 1998), foram desenvolvidos para sistemas críticos de segurança e proteção.

As equipes de gerenciamento de qualidade que estão desenvolvendo padrões para uma empresa devem, em geral, basear esses padrões em padrões nacionais e internacionais. Ao usar padrões internacionais como ponto de partida, a equipe de garantia de qualidade deve elaborar um 'manual' de padrões. Este deve definir os padrões necessários para sua organização. Exemplos de padrões que poderiam ser incluídos nesse manual são mostrados na Tabela 24.2.

Algumas vezes, os engenheiros de software consideram os padrões demasiadamente prescritivos e não realmente relevantes para a atividade técnica de desenvolvimento de software. Isso é mais provável quando os padrões de projeto exigem documentação tediosa e registros de trabalho. Embora geralmente concordem sobre a necessidade geral de padrões, muitas vezes os engenheiros encontram boas razões pelas quais padrões não se adequam a seus projetos. Portanto, para minimizar o descontentamento e encorajar o uso de padrões, os gerentes de qualidade que definem os padrões devem seguir os seguintes passos:

1. *Envolver os engenheiros de software na seleção de padrões de produto*. Se os desenvolvedores entenderem por que os padrões foram selecionados, eles estarão mais propensos a se comprometerem com esses padrões. Idealmente, além de definir o padrão a ser seguido, o documento de padrões também deve incluir comentários explicando por que essas decisões de padronização foram tomadas.

Tabela 24.2 Padrões de produto e de processo

Padrões de produto	Padrões de processo
Formulário de revisão de projeto	Condução de revisão de projeto
Estrutura de documento de requisitos	Apresentação do novo código para a construção de sistema
Formato de cabeçalho de método	Processo de versão e release
Estilo de programação Java	Processo de aprovação de plano de projeto
Formato de plano de projeto	Processo de controle de mudança
Formulário de solicitação de mudança	Processo de registro de teste

- 2.** *Revisar e modificar regularmente os padrões para refletir mudanças nas tecnologias.* O desenvolvimento de padrões é um processo caro e os padrões tendem a ser consagrados em um manual de padrões de empresa. Por causa dos custos e das discussões necessárias, muitas vezes existe relutância em alterá-los. Um manual de padrões é essencial, mas ele deve evoluir para refletir mudanças nas circunstâncias e tecnologias.
- 3.** *Fornecer ferramentas de software para oferecer suporte aos padrões.* Muitas vezes, os desenvolvedores percebem os padrões como um estorvo, em casos nos quais a conformidade a esses padrões envolve tediosos trabalhos manuais que poderiam ser realizados por uma ferramenta de software. Se o suporte a ferramentas está disponível, é necessário muito pouco esforço para seguir os padrões de desenvolvimento de software. Por exemplo, padrões de documentos podem ser implementados usando-se estilos de processador de texto.

Os diferentes tipos de software precisam de diferentes processos de desenvolvimento, portanto, os padrões precisam ser adaptáveis. Não faz sentido prescrever um modo particular de trabalho se ele é inadequado a um projeto ou a uma equipe de projeto. Cada gerente de projeto deve ter autoridade para modificar padrões de processo de acordo com as circunstâncias individuais. No entanto, quando são feitas alterações, é importante assegurar que essas alterações não ocasionem a perda da qualidade de produto. Isso afetaria o relacionamento da empresa com seus clientes e provavelmente ocasionaria custos maiores para o projeto.

O gerente de projeto e o gerente de qualidade podem evitar os problemas com padrões não apropriados por meio de um planejamento da qualidade cuidadoso logo no início do projeto. Eles devem decidir quais padrões organizacionais devem ser usados sem alterações, quais devem ser modificados e quais devem ser ignorados. Novos padrões podem ser necessários em resposta aos requisitos de clientes e de projeto. Por exemplo, padrões para especificações formais podem ser necessários caso estas não tenham sido usadas em projetos anteriores.



24.2.1 0 framework de normas ISO 9001

Existe uma série de normas que podem ser usadas no desenvolvimento de sistemas de gerenciamento de qualidade em todos os setores, chamada ISO 9000. As normas ISO 9000 podem ser aplicadas a uma variedade de organizações, desde a produção até a indústria de serviços. A ISO 9001, a mais geral desses padrões, aplica-se a organizações que projetam, desenvolvem e mantêm produtos, incluindo software. Originalmente, a norma ISO 9001 foi desenvolvida em 1987, com sua mais recente revisão em 2008.

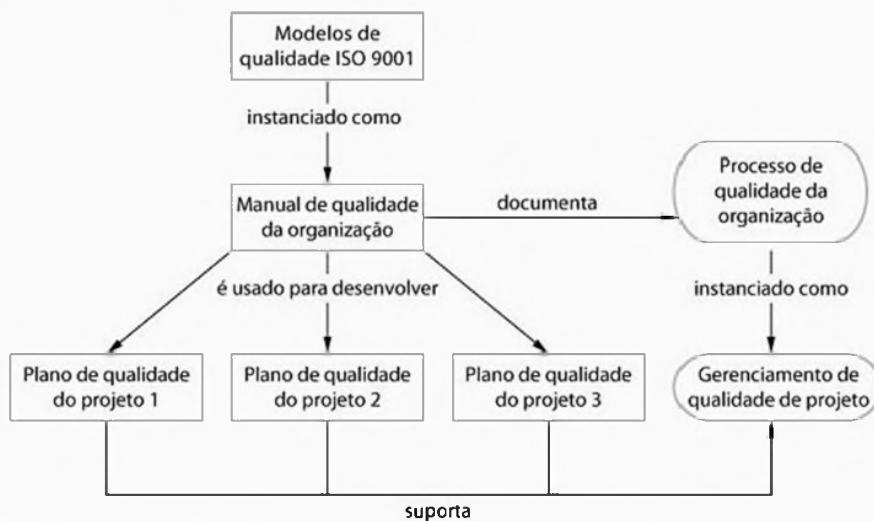
A norma ISO 9001 não é propriamente um padrão para o desenvolvimento de software, mas é um *framework* para o desenvolvimento de padrões de software. Ela define os princípios gerais da qualidade, descreve os processos gerais de qualidade e estabelece os padrões organizacionais e os procedimentos que devem ser definidos. Estes devem ser documentados em um manual de qualidade da organização.

A grande revisão da norma ISO 9001 em 2000 reorientou os processos em torno de nove padrões principais (Figura 24.3). Se uma organização deve adequar-se à ISO 9001, deve documentar como seus processos se referem a esses processos essenciais. Essa organização também deve definir e manter registros que demonstrem que os processos organizacionais definidos foram seguidos. O manual de qualidade da empresa deve descrever os processos relevantes e os dados de processo que devem ser coletados e mantidos.

Figura 24.3 Os processos essenciais da ISO 9001

As normas ISO 9001 não definem ou prescrevem os processos de qualidade específicos que devem ser usados em uma empresa. Para estar adequada à ISO 9001, uma empresa deve ter definidos os tipos de processos mostrados na Figura 24.3, além de ter procedimentos que demonstrem que seus processos de qualidade estão sendo seguidos. Isso permite flexibilidade entre setores industriais e tamanhos de empresa. Podem ser definidos padrões de qualidade apropriados para o tipo de software que está sendo desenvolvido. As pequenas empresas podem ter processos não burocráticos e ainda estarem em conformidade com a ISO 9001. No entanto, essa flexibilidade significa que você não pode fazer suposições sobre as semelhanças ou diferenças entre os processos em diferentes conformidades à ISO 9001. Algumas empresas podem ter processos de qualidade muito rígidos que mantenham registros detalhados, enquanto outras podem ser menos formais, com documentação adicional mínima.

Os relacionamentos entre a ISO 9001, os manuais de qualidade organizacional e os planos individuais de qualidade de projeto são mostrados na Figura 24.4. Esse diagrama foi obtido a partir de um modelo dado por Ince (1994), que explica como a norma geral ISO 9001 pode ser usada como uma base para processos de gerenciamento de qualidade de software. Bamford e Dielbler (2003) explicaram como a norma posterior ISO 9001:2000, pode ser aplicada em empresas de software.

Figura 24.4 ISO 9001 e gerenciamento de qualidade

Alguns clientes de software exigem que seus fornecedores sejam certificados pela ISO 9001. Dessa forma, eles podem ficar confiantes de que a empresa de desenvolvimento de software tem um sistema de gerenciamento de qualidade aprovado. Autoridades independentes de certificação examinam os processos de gerenciamento de qualidade, processam a documentação e decidem se esses processos cobrem todas as áreas especificadas na ISO 9001. Caso isso ocorra, elas certificam que os processos de qualidade da empresa, como definidos no manual de qualidade, estão em conformidade com a norma ISO 9001.

Algumas pessoas pensam que a certificação ISO 9001 significa que a qualidade do software produzido por empresas certificadas será melhor do que o software produzido em empresas não certificadas. Isso não é necessariamente verdade. A norma ISO 9001 assegura que a organização dispõe de procedimentos de gerenciamento de qualidade e segue tais procedimentos. Não há uma garantia de que empresas certificadas com a ISO 9001 usam as melhores práticas de desenvolvimento de software ou que seus processos gerarão software com alta qualidade.

Por exemplo, uma empresa poderia definir padrões de cobertura de teste especificando que todos os métodos em objetos devem ser chamados ao menos uma vez. Infelizmente, essa norma pode ser atendida por testes de software incompleto, que não executam testes com diferentes parâmetros de método. Desde que sejam seguidos os procedimentos de testes e mantidos os registros dos testes realizados, a empresa poderia ser certificada com a ISO 9001. A certificação ISO 9001 define como qualidade a conformidade com os padrões e não leva em consideração a qualidade vivida pelos usuários do software.

Os métodos ágeis, que evitam a documentação e se concentram no código que está sendo desenvolvido, têm pouco em comum com os processos de qualidade formais discutidos na ISO 9001. Já existe algum trabalho no sentido de reconciliar essas abordagens (STALHANE e HANSEN, 2008), mas a comunidade de desenvolvimento ágil é fundamentalmente contra ao que eles veem como *overhead burocrático* de conformidade a normas. Por essa razão, empresas que usam métodos ágeis de desenvolvimento raramente estão preocupadas com a certificação ISO 9001.

24.3 Revisões e inspeções

Revisões e inspeções são atividades de controle de qualidade que verificam a qualidade dos entregáveis de projeto. Isso envolve examinar o software, sua documentação e os registros do processo para descobrir erros e omissões e verificar se os padrões de qualidade foram seguidos. Conforme discutido nos capítulos 8 e 15, revisões e inspeções são usadas junto com teste de programa, como parte do processo geral de validação e verificação de software.

Durante uma revisão, um grupo de pessoas examina o software e a documentação associada à procura de possíveis problemas e não conformidade com padrões. A equipe de revisão informada sobre o nível de qualidade de um sistema ou entregável de projeto toma decisões. Os gerentes de projeto podem usar essas avaliações para tomar decisões de planejamento e alocar recursos para o processo de desenvolvimento.

As avaliações de qualidade são baseadas em documentos que foram produzidos durante o processo de desenvolvimento de software. Assim como as especificações de software, projetos ou códigos, modelos de processos, planos de testes, procedimentos de gerenciamento de configuração, padrões de processo e manuais de usuário também podem ser revistos. A revisão deve verificar a consistência e a completude dos documentos ou código em revisão e certificar-se de que os padrões de qualidade foram seguidos.

No entanto, as revisões não servem apenas para verificar a conformidade com os padrões. Elas também são usadas para ajudar a descobrir problemas e omissões no software ou na documentação de projeto. As conclusões das revisões devem ser formalmente registradas como parte do processo de gerenciamento de qualidade. Se os problemas forem descobertos, os comentários dos revisores devem ser enviados ao autor do software ou a quem estiver responsável por corrigir os erros ou as omissões.

O objetivo das revisões e inspeções é melhorar a qualidade de software e não avaliar o desempenho das pessoas na equipe de desenvolvimento. A revisão é um processo público de detecção de erros, em comparação com processos de testes de componente mais privados. Inevitavelmente, erros que são cometidos por indivíduos são revelados a toda a equipe de programação. Para garantir que todos os desenvolvedores estejam engajados construtivamente com o processo de revisão, os gerentes de projeto precisam estar sensíveis às preocupações individuais. Eles precisam desenvolver uma cultura de trabalho que forneça suporte sem procurar os culpados pelos erros descobertos.

Ainda que uma revisão de qualidade forneça informações sobre o software que está sendo desenvolvido para gerenciamento, não é igual a uma revisão de progresso de gerenciamento. Conforme discutido no Capítulo 23, as revisões de progresso comparam o andamento real de um projeto de software com o progresso planejado. Sua principal preocupação é o projeto entregar o software útil no prazo e dentro do orçamento. As revisões de progresso levam em conta os fatores externos e as alterações de circunstâncias que podem significar que o software em desenvolvimento não é mais necessário ou precisa ser radicalmente alterado. Projetos que desenvolveram softwares de alta qualidade podem ser cancelados por mudanças nos negócios ou em seu ambiente operacional.



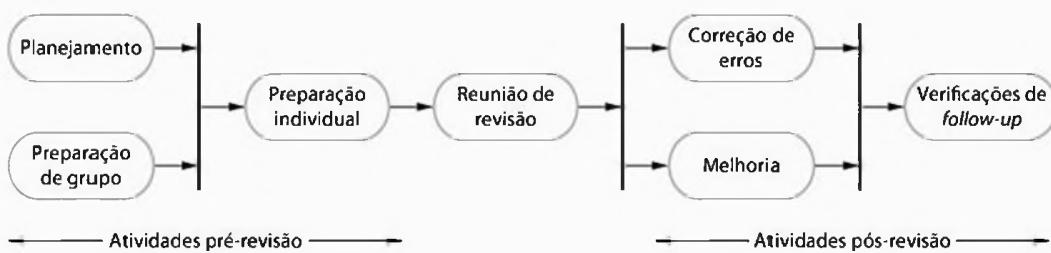
24.3.1 O processo de revisão

Embora existam muitas variações nos detalhes das revisões, o processo de revisão (Figura 24.5) normalmente é estruturado em três fases:

- Atividades pré-revisão.** As atividades preparatórias são essenciais para a eficácia da revisão. Em geral, as atividades de pré-revisão estão relacionadas com o planejamento e a preparação da revisão. O planejamento de revisão envolve a definição de uma equipe de revisão, a organização de um tempo e de um lugar para sua ocorrência e a distribuição de documentos a serem revistos. Durante a preparação da revisão, a equipe pode reunir-se para obter uma visão geral do software a ser revisto. Os membros da equipe de revisão individual precisam ler e entender o software ou os documentos e padrões relevantes. Eles trabalham independentemente para encontrar erros, omissões e desvios de padrões. Os revisores podem fornecer comentários escritos sobre o software, caso eles não possam participar da reunião de revisão.
- Reunião de revisão.** Durante a reunião de revisão, o autor do documento ou do programa a ser revisto deve 'caminhar' pelo documento com a equipe de revisão. A revisão em si deve ser relativamente curta — duas horas, no máximo. Um membro da equipe deve presidir a revisão e outro deve registrar formalmente todas as decisões e ações a serem tomadas. Durante a revisão, o presidente é responsável por garantir que todos os comentários escritos sejam considerados. O presidente da revisão deve escrever um registro dos comentários e ações acordadas durante a revisão.
- Atividades pós-revisão.** Após a reunião de revisão, as questões e os problemas levantados devem ser abordados. Esse processo pode envolver a correção de bugs de software, a refatoração do software para que ele esteja em conformidade com os padrões de qualidade, ou a necessidade de uma nova redação dos documentos. Em alguns casos, os problemas descobertos em uma revisão de qualidade são tais que uma revisão de gerenciamento também é necessária para decidir se mais recursos devem ser disponibilizados para corrigi-los. Após as alterações terem sido feitas, o presidente da revisão pode verificar se todos os comentários de revisão foram levados em consideração. Às vezes, uma nova revisão será necessária para verificar se as alterações efetuadas cobriram todos os comentários da revisão anterior.

Geralmente, as equipes de revisão devem ter um núcleo com três a quatro pessoas, selecionadas como principais revisores. Um membro deve ser um projetista sênior, o qual deve assumir a responsabilidade pela tomada de decisões técnicas significativas. Os revisores principais podem convidar outros membros de projeto, como os projetistas de subsistemas relacionados, para contribuir com a revisão. Eles não podem estar envolvidos na revisão completa do documento, mas devem concentrar-se nas seções que afetam seu trabalho. Como alternativa, a equipe de revisão pode circular o documento e solicitar comentários por escrito de vários membros de projeto. O gerente de projeto não precisa estar envolvido na revisão, a menos que se esperem problemas que exijam alterações no plano de projeto.

Figura 24.5 O processo de revisão de software



O processo de revisão exemplificado depende de todos os membros de uma equipe de desenvolvimento estarem reunidos e disponíveis para uma reunião. No entanto, atualmente, as equipes de projeto costumam ser distribuídas geograficamente, por vezes em países ou continentes diferentes, e, em virtude disso, pode ser inviável a reunião de membros de equipe. Em tais situações, ferramentas de edição de documento podem ser usadas para apoiar o processo de revisão. Os membros de equipe podem usá-las para fazer anotações no documento ou comentários no código-fonte de software. Esses comentários são visíveis a outros membros de equipe, os quais podem, em seguida, aprová-los ou rejeitá-los. Uma discussão por telefone pode ser necessária quando for preciso resolver divergências entre os revisores.

Em desenvolvimento ágil, o processo de revisão de software normalmente é informal. No Scrum, por exemplo, ocorre uma reunião de revisão após a conclusão da cada iteração do software (uma revisão de *sprint*), em que os problemas e as questões de qualidade podem ser discutidos. Em Extreme Programming, conforme discutido na próxima seção, a programação em pares garante que o código esteja sendo examinado e revisto constantemente por outro membro de equipe. As questões de qualidade geral também são consideradas nas reuniões diárias de equipe, mas XP baseia-se em indivíduos que tomam a iniciativa para melhorar e refatorar o código. Abordagens ágeis não costumam ser dirigidas a padrões; desse modo, as questões de conformidade com padrões geralmente não são consideradas.

A falta de procedimentos formais de qualidade em métodos ágeis significa que pode haver problemas ao se usarem as abordagens ágeis em empresas que desenvolveram procedimentos detalhados de gerenciamento de qualidade. Revisões de qualidade podem diminuir o ritmo de desenvolvimento de software e elas são melhor usadas em um processo de desenvolvimento dirigido a planos, no qual as revisões podem ser planejadas e um outro trabalho, programado em paralelo. Isso é impraticável em abordagens ágeis centradas unicamente no desenvolvimento de código.



24.3.2 Inspeções de programa

As inspeções de programa são ‘revisões em pares’ em que os membros da equipe colaboram para encontrar *bugs* no programa que está sendo desenvolvido. Conforme discutido no Capítulo 8, as inspeções podem fazer parte dos processos de verificação e validação de software. Elas complementam os testes, pois não exigem que o programa seja executado. Isso significa que podem ser verificadas versões incompletas do sistema e que representações, tais como modelos UML, podem ser checadas. Gilb e Graham (1993) sugerem que uma das maneiras mais efetivas de se usar as inspeções é revisar os casos de teste para um sistema. As inspeções podem descobrir problemas com testes e, assim, melhorar a eficácia desses testes em detectar *bugs* no programa.

As inspeções de programa envolvem membros de equipe de diferentes origens fazendo uma revisão cuidadosa, linha por linha de código-fonte de programa. Eles procuram defeitos e problemas e os descrevem em uma reunião de inspeção. Os defeitos podem ser erros lógicos, anomalias no código que podem indicar uma condição errada ou recursos que foram omitidos do código. A equipe de revisão examina em detalhes os modelos de projeto ou o código de programa e destaca anomalias e problemas para que sejam reparados.

Durante uma inspeção, frequentemente se usa um *checklist* de erros comuns de programação para ajudar na busca por *bugs*. Esse *checklist* pode basear-se em exemplos de livros ou no conhecimento de defeitos comuns em um domínio de aplicação específico. Diferentes *checklists* são usados para diferentes linguagens de programação, pois cada linguagem tem seus próprios erros característicos. Humphrey (1989), em uma discussão abrangente de inspeções, dá uma série de exemplos de *checklists* de inspeção.

Algumas possíveis verificações, as quais podem ser feitas durante o processo de inspeção, são mostradas na Tabela 24.3. Gilb e Graham (1993) enfatizam que cada organização deve desenvolver seu próprio *checklist* de inspeção com base em práticas e padrões locais. Esses *checklists* devem ser atualizados regularmente, à medida que novos tipos de defeitos são encontrados. Os itens no *checklist* variam de acordo com a linguagem de programação, em virtude dos diferentes níveis de verificação possíveis em tempo de compilação. Por exemplo, um compilador Java verifica se as funções têm o número correto de parâmetros; um compilador C, por sua vez, não verifica.

A maioria das empresas que introduziu inspeções descobriu que estas são muito eficazes em encontrar *bugs*. Fagan (1986) informou que mais de 60% dos erros em um programa podem ser detectados por meio de inspeções informais de programa. Mills et al. (1987) sugerem que uma abordagem mais formal para inspeção, com base em argumentos de correção, pode detectar mais de 90% dos erros em um programa. McConnell (2004) compara testes de unidade, em que a taxa de detecção de defeitos é de cerca de 25%, com inspeções, em que a taxa de detecção de defeitos era de 60%. Ele também descreve uma série de estudos de caso, incluindo um exemplo em que a introdução de revisões em pares levou a um aumento de 14% na produtividade e diminuiu em 90% os defeitos de programa.

Tabela 24.3 Um checklist de inspeção

Classe de defeito	Verificação de inspeção
Defeitos de dados	<ul style="list-style-type: none"> Todas as variáveis de programa são iniciadas antes que seus valores sejam usados? Todas as constantes foram nomeadas? O limite superior de vetores deve ser igual ao tamanho do vetor ou ao tamanho –1? Se as <i>strings</i> de caracteres são usadas, um delimitador é explicitamente atribuído? Existe alguma possibilidade de <i>overflow de buffer</i>?
Defeitos de controle	<ul style="list-style-type: none"> Para cada instrução condicional, a condição está correta? É certo que cada <i>loop</i> vai terminar? As declarações compostas estão posicionadas corretamente entre colchetes? Em declarações <i>case</i>, todos os <i>cases</i> possíveis são considerados? Se um <i>break</i> é requerido após cada <i>case</i> em declarações <i>case</i>, este foi incluído?
Defeitos de entrada/saída	<ul style="list-style-type: none"> Todas as variáveis de entrada são usadas? Todas as variáveis de saída receberam um valor antes de serem emitidas? Entradas inesperadas podem causar corrupção de dados?
Defeitos de interface	<ul style="list-style-type: none"> Todas as chamadas de funções e métodos têm o número correto de parâmetros? Os parâmetros formais e reais correspondem? Os parâmetros estão na ordem correta? Se os componentes acessam memória compartilhada, eles têm o mesmo modelo de estrutura de memória compartilhada?
Defeitos de gerenciamento de armazenamento	<ul style="list-style-type: none"> Se uma estrutura ligada é modificada, todas as ligações foram corretamente reatribuídas? Se o armazenamento dinâmico é usado, o espaço foi alocado corretamente? O espaço é explicitamente desalocado após não ser mais necessário?
Defeitos de gerenciamento de exceção	<ul style="list-style-type: none"> Foram levadas em consideração todas as condições possíveis de erro?

Apesar da efetividade reconhecida, muitas empresas de desenvolvimento de software são relutantes em usar inspeções ou revisões em pares. Engenheiros de software com experiência em testes de programa, por vezes, não estão dispostos a aceitar que inspeções possam ser mais eficazes que os testes na detecção de defeitos. Gerentes podem ser suspeitos, pois inspeções exigem custos adicionais durante o projeto e desenvolvimento. Eles podem não querer correr o risco de não haver redução nos custos de testes de programa.

Os processos ágeis raramente usam processos formais de inspeção ou de revisão em pares. Em vez disso, eles contam com os membros de equipe que colaboram para verificar o código uns dos outros e as diretrizes informais, como ‘verifique antes do *check-in*’, que sugerem que os programadores devem verificar seu próprio código. Os adeptos de Extreme Programming argumentam que a programação em pares é um substituto eficaz para inspeção, pois é um processo contínuo de inspeção. Duas pessoas olham cada linha de código e verificam antes de este ser aceito.

A programação em pares leva ao profundo conhecimento de um programa, pois ambos os programadores precisam compreender seu funcionamento em detalhes para continuar o desenvolvimento. Esse conhecimento, às vezes, é difícil de alcançar em outros processos de inspeção, assim, a programação em pares pode encontrar bugs que, às vezes, não são descobertos em inspeções formais. No entanto, a programação em pares também pode levar a desentendimentos mútuos de requisitos, em que ambos os membros do par cometem o mesmo erro. Além disso, os pares podem estar relutantes em procurar por erros, por não quererem diminuir o ritmo do andamento do projeto. As pessoas envolvidas não podem ser tão objetivas como uma equipe de inspeção externa e sua capacidade de detectar defeitos pode ser comprometida por seu estreito relacionamento de trabalho.

24.4 Medição e métricas de software

A medição de software preocupa-se com a derivação de um valor numérico ou o perfil para um atributo de um componente de software, sistema ou processo. Comparando esses valores entre si e com os padrões que se

aplicam a toda a organização, você pode ser capaz de tirar conclusões sobre a qualidade do software ou avaliar a eficácia dos métodos, das ferramentas e dos processos de software.

Por exemplo, digamos que uma organização tem a intenção de introduzir uma nova ferramenta de teste de software. Antes de introduzir a ferramenta, em um determinado momento você registra o número de defeitos de software descobertos. Essa é uma *baseline* de avaliação da eficácia da ferramenta. Depois de algum tempo usando a ferramenta, esse processo é repetido. Se mais defeitos forem encontrados durante o mesmo período, depois que a ferramenta foi introduzida, você pode decidir se ela fornece suporte útil para o processo de validação de software.

O objetivo a longo prazo de medição de software é usá-la no lugar de revisões para fazer julgamentos sobre a qualidade de software. Usando a medição de software, um sistema poderia, idealmente, ser avaliado usando uma variedade de métricas e, a partir dessa medição, deduzir um valor para a qualidade do sistema. Se o software atingir o limiar de qualidade requerido, então ele poderia ser aprovado sem revisão. Quando apropriado, as ferramentas de medição também podem realçar áreas do software que poderiam ser melhoradas. No entanto, estamos ainda longe dessa situação ideal e não há sinal de que as avaliações automatizadas de qualidade venham a se tornar realidade em um futuro próximo.

Uma métrica de software é uma característica de um sistema de software, documentação de sistema ou processo de desenvolvimento que pode ser objetivamente medido. Exemplos de métricas incluem: o tamanho de um produto em linhas de código; o índice *Fog* (GUNNING, 1962), que é uma medida da legibilidade de uma passagem de texto escrito; o número de defeitos relatados em um produto de software entregue, e o número de pessoas/dia requerido para desenvolver um componente de sistema.

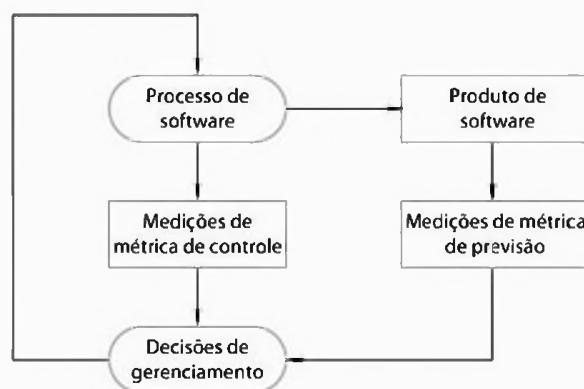
As métricas de software podem ser métricas de controle ou métricas de previsão. Como os nomes sugerem, as primeiras suportam os processos de gerenciamento e as outras o ajudam a prever as características do software. As métricas de controle são geralmente associadas com os processos de software. Exemplos de métricas de controle ou de processos são o esforço médio e o tempo necessário para reparar os defeitos relatados. Métricas de previsão são associadas com o software em si e, por vezes, são conhecidas como 'métricas de produto'. São exemplos de métricas de previsão: a complexidade ciclomática de um módulo (discutida no Capítulo 8), o comprimento médio dos identificadores em um programa e o número de atributos e operações associadas com as classes de objeto em um projeto.

As métricas de controle e de previsão podem influenciar a tomada de decisão de gerenciamento, como mostrado na Figura 24.6. Os gerentes usam métricas de processo para decidir se devem ser feitas alterações no processo; as métricas de previsão são usadas para ajudar a estimar o esforço necessário para fazer as alterações no software. Neste capítulo, discuto principalmente as métricas de previsão, cujos valores são avaliados, analisando o código de um sistema de software. No Capítulo 26, as métricas de controle e como elas são usadas na melhoria de processos são abordadas.

Existem duas maneiras para o uso das medições de um software de sistema:

1. *Para atribuir um valor aos atributos de qualidade de sistema.* Ao medir as características dos componentes de sistema, bem como sua complexidade ciclomática e, em seguida, agregar essas medições, você pode avaliar os atributos de qualidade do sistema, como a manutenibilidade.

Figura 24.6 Medições de previsão e de controle



2. Para identificar os componentes de sistema cuja qualidade não atingiu o padrão. As medições podem identificar componentes individuais com características que se desviam da norma. Por exemplo, você pode medir componentes para descobrir aqueles com a mais alta complexidade. Esses são mais propensos a conter bugs porque a complexidade os torna mais difíceis de entender.

Infelizmente, é difícil fazer medições diretas de muitos dos atributos de qualidade de software mostrados na Tabela 24.1. Os atributos de qualidade como manutenibilidade, compreensibilidade e usabilidade são atributos externos relacionados com os desenvolvedores e usuários que experimentam o software. Eles são afetados por fatores subjetivos, como a experiência e a educação do usuário e, portanto, não podem ser medidos objetivamente. Para fazer um julgamento sobre esses atributos, você deve medir alguns atributos internos do software (como tamanho, complexidade etc.) e assumir que estão relacionados com as características de qualidade com as quais você se preocupa.

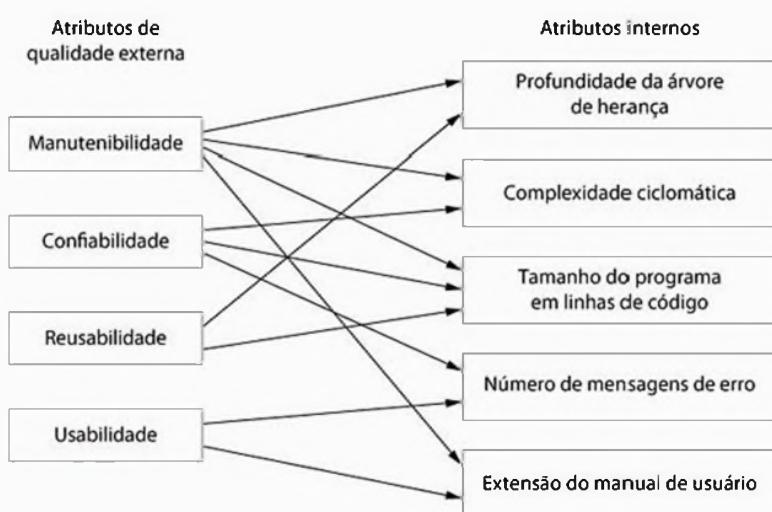
A Figura 24.7 mostra alguns atributos externos de qualidade dos softwares e atributos internos que poderiam, intuitivamente, ser relacionados a eles. O diagrama sugere que pode haver relacionamentos entre atributos internos e externos, mas não diz como esses atributos estão relacionados. Se a medida do atributo interno deve ser uma previsão útil de características externas de software, três condições devem ser mantidas (KITCHENHAM, 1990):

1. O atributo interno deve ser medido com precisão. Isso nem sempre é simples e pode exigir ferramentas específicas para fazer as medições.
2. Deve existir um relacionamento entre o atributo que pode ser medido e o atributo de qualidade externa de interesse. Ou seja, o valor do atributo de qualidade deve ser relacionado, de alguma forma, com o valor do atributo que pode ser medido.
3. Esse relacionamento entre os atributos internos e externos deve ser compreendido, validado e expresso em termos de uma fórmula ou modelo. A formulação de modelo envolve a identificação da forma funcional do modelo (linear, exponencial etc.) pela análise de dados coletados, identificando os parâmetros que devem ser incluídos no modelo e calibrando esses parâmetros com o uso dos dados existentes.

Os atributos internos de software, como a complexidade ciclomática de um componente, são medidos com o uso de ferramentas de software que analisam o código-fonte do software. As ferramentas *open source* disponíveis podem ser usadas para fazer essas medições. Embora a intuição sugira que poderia haver um relacionamento entre a complexidade de um componente de software e o número de falhas observadas em uso, é difícil demonstrar objetivamente que é este o caso. Para testar essa hipótese, você precisa de dados de falha de um grande número de componentes e de acesso ao código-fonte do componente para análise. Poucas empresas fizeram um compromisso a longo prazo para a coleta de dados sobre seu software, portanto, dados de falha raramente são disponibilizados para análise.

Na década de 1990, grandes empresas, como a Hewlett-Packard (GRADY, 1993), a AT&T (BARNARD e PRICE, 1994) e a Nokia (KILPI, 2001) introduziram programas de métricas. Elas fizeram medições de seus

Figura 24.7 Relacionamentos entre os atributos internos e externos de software



produtos e processos e usaram-nas em seus processos de gerenciamento de qualidade. Elas se centraram em coletar métricas de defeitos de programa, bem como em processos de verificação e validação. Offen e Jeffrey (1997) e Hall e Fenton (1997) discutem a introdução de programas de métricas na indústria com mais detalhes.

Existem poucas informações publicamente disponíveis sobre o uso atual da medição sistemática de software na indústria. Muitas empresas coletam informações sobre seu software, como o número de solicitações de mudança de requisitos ou o número de defeitos descobertos nos testes. No entanto, não está claro se, em seguida, usam essas medições sistematicamente para comparar produtos e processos de software ou avaliar o impacto das mudanças nos processos e ferramentas de software. Existem várias razões pelas quais isso é difícil:

1. É impossível quantificar o retorno sobre o investimento da introdução de um programa de métricas em organizações. Durante os últimos anos, houve melhorias significativas na qualidade de software sem o uso de métricas. Por isso é difícil justificar os custos iniciais de introdução de medição e avaliação sistemática de software.
2. Não existe um padrão para as métricas de software ou processos padronizados para medição e análise. Muitas empresas relutam em introduzir programas de medição até que estes e as ferramentas de suporte estejam disponíveis.
3. Em muitas empresas, os processos de software não são padronizados e são mal definidos e mal controlados. Além disso, existe muita variabilidade em processos da mesma empresa para que as medições sejam usadas de forma significativa.
4. Grande parte da pesquisa sobre medição e métricas de software centra-se nas métricas baseadas em códigos e processos de desenvolvimento dirigidos a planos. No entanto, cada vez mais o software é desenvolvido configurando sistemas ERP ou COTS ou usando os métodos ágeis. Não sabemos, portanto, se a pesquisa anterior é aplicável a essas técnicas de desenvolvimento de software.
5. A introdução da medição acrescenta *overhead* aos processos. Estes contradizem os objetivos dos métodos ágeis, os quais recomendam a eliminação das atividades de processos que não estão diretamente relacionadas ao desenvolvimento de programa. Portanto, as empresas que adotaram os métodos ágeis geralmente não adotam um programa de métricas.

As métricas e medições de software são as bases da engenharia de software empírica (ENDRES e ROMBACH, 2003). Essa é uma área de pesquisa em que as experiências em sistemas de software e a coleta de dados sobre projetos reais foram usadas para formar e validar hipóteses sobre os métodos e as técnicas de engenharia de software. Os pesquisadores que trabalham nessa área argumentam que podemos confiar no valor dos métodos e das técnicas de engenharia de software apenas se pudermos fornecer evidências concretas de que eles realmente oferecem os benefícios que seus inventores sugerem.

Infelizmente, mesmo quando é possível fazer medições objetivas e tirar conclusões a partir delas, isso pode não convencer os tomadores de decisões necessariamente. Em vez disso, as tomadas de decisões muitas vezes são influenciadas por fatores subjetivos, como a novidade ou a extensão em que as técnicas são de interesse para os profissionais. Portanto, penso que ainda se passarão muitos anos antes que a engenharia de software empírica tenha efeitos significativos nas práticas de engenharia de software.



24.4.1 Métricas do produto

As métricas de produto são métricas de previsão usadas para medir atributos internos de um sistema de software. O tamanho de sistema, medido em linhas de código, ou o número de métodos associados a cada classe de objeto são exemplos de métricas de produto. Contudo, como já foi explicado nesta seção, as características de software que podem ser facilmente medidas, como tamanho e complexidade ciclomática, não têm uma relação clara e consistente com atributos de qualidade tais como capacidade de compreensão e manutenibilidade. Os relacionamentos variam de acordo com os processos de desenvolvimento e tecnologia usada, bem como o tipo de sistema que está sendo desenvolvido.

As métricas de produto se dividem em duas classes:

1. Métricas dinâmicas, as quais são coletadas por meio de medições efetuadas de um programa em execução. Essas métricas podem ser coletadas durante o teste de sistema ou após o sistema estar em uso. Um exemplo pode ser o número de relatórios de *bugs* ou o tempo necessário para concluir uma computação.

2. Métricas estáticas, que são coletadas por meio de medições feitas de representações do sistema, como o projeto, o programa ou a documentação. Exemplos de métricas estáticas são o tamanho de código e o comprimento médio de identificadores usados.

Esses tipos de métrica estão relacionados com diferentes atributos de qualidade. Métricas dinâmicas ajudam a avaliar a eficiência e a confiabilidade de um programa. Métricas estáticas ajudam a avaliar a complexidade, a comprehensibilidade e a manutenibilidade de um sistema ou componentes de um sistema de software.

Geralmente, existe um relacionamento claro entre as métricas dinâmicas e as características de qualidade de software. É bastante fácil medir o tempo de execução necessário para funções particulares e avaliar o tempo necessário para iniciar um sistema. Eles se relacionam diretamente com a eficiência do sistema. Da mesma forma, o número de falhas de sistema e o tipo de falha podem ser registrados e relacionados diretamente com a confiabilidade do software, conforme vimos no Capítulo 15.

Como já discutido, métricas estáticas, tais como as mostradas na Tabela 24.4, têm um relacionamento indireto com atributos de qualidade. Um grande número de métricas diferentes foi proposto e muitos experimentos tentaram derivar e validar os relacionamentos entre essas métricas e atributos, como a complexidade de sistema e manutenibilidade. Nenhum desses experimentos foi conclusivo, mas o tamanho de programa e a complexidade de controle parecem ser os mais confiáveis mecanismos de previsão de comprehensibilidade, complexidade e manutenibilidade de sistema.

As métricas na Tabela 24.4 são aplicáveis a qualquer programa, mas métricas orientadas a objetos (OO) mais específicas também foram propostas. A Tabela 24.5 resume o conjunto de Chidamber e Kemerer (1994), às vezes, chamado *suite CK*, de seis métricas orientadas a objetos. Embora elas tenham sido originalmente propostas na década de 1990, ainda são as métricas OO mais amplamente usadas. Algumas ferramentas de projeto de UML coletam valores automaticamente para essas métricas quando são criados diagramas UML.

El-Amam (2001), em uma excelente revisão de métricas orientadas a objetos, discute as métricas CK e outras métricas OO e conclui que nós ainda não temos provas suficientes para compreender como essas e outras métricas orientadas a objetos se relacionam com as qualidades externas de software. Essa situação não mudou muito desde sua análise em 2001. Ainda não sabemos como usar as medições de programas orientados a objeto para tirar conclusões confiáveis sobre sua qualidade.

Tabela 24.4 Métricas estáticas de produto de software

Métrica de software	Descrição
<i>Fan-in/Fan-out</i>	<i>Fan-in</i> é a medida do número de funções ou métodos que chamam outra função ou método (digamos X). <i>Fan-out</i> é o número de funções que são chamadas pela função de X. Um valor alto para <i>fan-in</i> significa que X está fortemente acoplado ao resto do projeto e alterações em X terão repercussões extensas. Um valor alto para <i>fan-out</i> sugere que a complexidade geral do X pode ser alta por causa da complexidade da lógica de controle necessário para coordenar os componentes chamados.
Comprimento de código	Essa é uma medida do tamanho de um programa. Geralmente, quanto maior o tamanho do código de um componente, mais complexo e sujeito a erros o componente é. O comprimento de código tem mostrado ser uma das métricas mais confiáveis para prever a propensão a erros em componentes.
Complexidade ciclomática	Essa é uma medida da complexidade de controle de um programa. Essa complexidade de controle pode estar relacionada à comprehensibilidade de programa. A complexidade ciclomática é discutida no Capítulo 8.
Comprimento de identificadores	Essa é uma medida do comprimento médio dos identificadores (nomes de variáveis, classes, métodos etc.) em um programa. Quanto mais longos os identificadores, mais provável que sejam significativos e, portanto, mais comprehensível o programa.
Profundidade de aninhamento condicional	Essa é uma medida da profundidade de aninhamento de declarações <i>if</i> em um programa. Declarações <i>if</i> profundamente aninhadas são difíceis de entender e potencialmente sujeitas a erros.
Índice Fog	Essa é uma medida do comprimento médio de palavras e sentenças em documentos. Quanto maior o valor de um índice <i>Fog</i> de um documento, mais difícil a sua comprehensão.

Tabela 24.5 O conjunto de métricas de CK orientadas a objetos

Métrica orientada a objeto	Descrição
Métodos ponderados por classe (WMC, do inglês <i>weighted methods per class</i>)	É o número de métodos em cada classe, ponderados pela complexidade de cada método. Portanto, um método simples pode ter uma complexidade de 1 e um método grande e complexo pode ter um valor muito superior. Quanto maior o valor para essa métrica, mais complexa a classe de objeto. Geralmente, os objetos complexos são mais difíceis de compreender. Eles podem não ser logicamente coesos, portanto, não podem ser reusados efetivamente como superclasses em uma árvore de herança.
Árvore de profundidade de herança (DIT, do inglês <i>depth Of inheritance tree</i>)	Representa o número de níveis discretos na árvore de herança em que as subclasses herdam atributos e operações (métodos) de superclasses. Quanto mais profunda a árvore de herança, mais complexo o projeto. Muitas classes de objeto podem precisar ser compreendidas para que as classes de objeto nas folhas da árvore sejam entendidas.
Número de filhos (NOC, do inglês <i>number of children</i>)	É uma medida do número de subclasses imediatas em uma classe. Ele mede a largura de uma hierarquia de classe, considerando que DIT mede sua profundidade. Um valor alto para NOC pode indicar um maior reuso. Isso pode significar que mais esforço deve ser dispensado na validação de classes de base por causa do número de subclasses que dependem delas.
Acoplamento entre classes de objeto (CBO, do inglês <i>coupling between object classes</i>)	Classes são acopladas quando métodos em uma classe usam métodos ou variáveis de instância definidas em uma classe diferente. CBO é uma medida de quanto acoplamento existe. Um valor alto para o CBO significa que as classes são altamente dependentes e, portanto, é mais provável que a mudança em uma classe afete outras classes do programa.
Resposta para uma classe (RFC, do inglês <i>response for a class</i>)	RFC é a medida do número de métodos que poderiam ser executados em resposta a uma mensagem recebida por um objeto dessa classe. Mais uma vez, RFC está relacionada com a complexidade. Quanto maior o valor de RFC, mais complexa é a classe e, portanto, mais provável que inclua erros.
Falta de coesão em métodos (LCOM, do inglês <i>lack of cohesion in methods</i>)	LCOM é calculada considerando os pares de métodos em uma classe. LCOM é a diferença entre o número de pares de métodos sem atributos compartilhados e o número de pares de métodos com atributos compartilhados. O valor dessa métrica tem sido amplamente discutido, e ele existe em diversas variações. Não está claro se realmente adiciona qualquer informação útil além das que já são fornecidas por outras métricas.

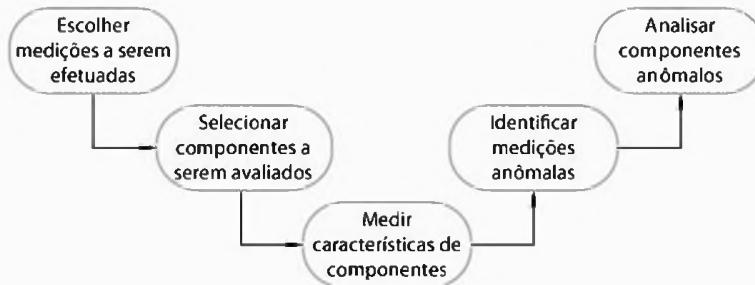


24.4.2 Análise de componentes de software

Um processo de medição que pode ser parte de um processo de avaliação de qualidade de software é mostrado na Figura 24.8. Cada componente de sistema pode ser analisado separadamente, usando uma variedade de métricas. Os valores dessas métricas podem ser comparados com diferentes componentes e, talvez, com dados históricos de medição obtidos em projetos anteriores. Medições anômalas, que diferem significativamente da norma, podem significar que existem problemas com a qualidade desses componentes.

Os estágios principais nesse processo de medição de componente são:

1. *Escolher medições a serem efetuadas.* As questões que a medição está destinada a responder devem ser formuladas e as medições necessárias para responder a essas questões devem ser definidas. As medições que não são

Figura 24.8 O processo de medição de produto

diretamente relevantes para essas questões não necessitam ser coletadas. O paradigma Meta-Questão-Métrica (GQM, do inglês *Goal-Question-Metric*) de Basili (BASILI e ROMBACH, 1988), discutido no Capítulo 26, é uma boa abordagem para se usar ao decidir quais dados devem ser coletados.

2. *Selecionar componentes a serem avaliados.* Você pode não precisar avaliar valores de métricas para todos os componentes de um sistema de software. Às vezes, pode selecionar um conjunto representativo de componentes para medição, que lhe permita fazer uma avaliação global da qualidade de sistema. Outras vezes, você pode se centrar nos componentes principais do sistema que estão em uso quase constante. A qualidade desses componentes é mais importante do que a qualidade de componentes raramente usados.
3. *Medir características de componentes.* Os componentes selecionados são medidos e os valores e as métricas associados, computados. Normalmente, isso envolve o processamento da representação de componente (projeto, código etc.) usando uma ferramenta automatizada de coleta de dados. Essa ferramenta pode ser especialmente escrita ou pode ser um recurso de ferramentas de projeto que já esteja em uso.
4. *Identificar medições anômalas.* Após terem sido realizadas as medições de componentes, você deve compará-las uns com os outros e com as medições anteriores que foram registradas em um banco de dados de medições. Você deve procurar valores anormalmente altos ou baixos para cada métrica, pois estes sugerem que pode haver problemas com o componente que exibe tais valores.
5. *Analizar componentes anômalos.* Ao identificar os componentes que têm valores anômalos para sua métrica escolhida, você deve examiná-los para decidir se esses valores de métricas anômalos significam que a qualidade do componente está comprometida. Um valor anômalo de métrica de complexidade (por exemplo) não significa, necessariamente, um componente de má qualidade. Pode haver alguma outra razão para o alto valor, por isso pode não significar que existam problemas de qualidade de componente.

Você sempre deve manter os dados coletados como um recurso organizacional e manter registros históricos de todos os projetos, mesmo quando dados não tenham sido usados em um determinado projeto. Uma vez estabelecido um banco de dados de medições suficientemente grande, você pode comparar a qualidade do software entre projetos e validar as relações entre os atributos de componentes internos e as características de qualidade.



24.4.3 Ambiguidade de medições

Quando você coleta dados quantitativos sobre software e processos de software, precisa analisar esses dados para entender seu significado. É fácil interpretar dados erroneamente e fazer inferências incorretas. Você não pode simplesmente olhar os dados *per se* — você também deve considerar o contexto em que eles são coletados.

Para ilustrar como os dados coletados podem ser interpretados de maneiras diferentes, considere o cenário adiante, relacionado com o número de solicitações de mudança feitas pelos usuários de um sistema:

Um gerente decide monitorar o número de solicitações de mudança enviadas pelos clientes com base em uma pré-suposição de que há um relacionamento entre essas solicitações de mudança e a usabilidade e adequabilidade do produto. Ele parte do princípio de que quanto maior for o número de solicitações, menos o software atenderá às necessidades do cliente.

Tratar as solicitações de mudança de software é caro. Portanto, a organização decide modificar seu processo com o objetivo de aumentar a satisfação do cliente e, ao mesmo tempo, reduzir os custos de mudanças. A intenção é que as mudanças de processo resultarão em melhores produtos e menos solicitações de mudança.

As mudanças de processo são iniciadas para aumentar o envolvimento do cliente no processo de projeto de software. São introduzidos testes Beta de todos os produtos e as solicitações dos clientes por modificações são incorporadas no produto entregue. Novas versões de produtos, desenvolvidas com esse processo modificado, são entregues. Em alguns casos, o número de solicitações de mudança é reduzido. Em outros, ele aumenta. O gerente fica desconcertado e considera impossível avaliar os efeitos das mudanças nos processos sobre a qualidade do produto.

Para entender por que esse tipo de ambiguidade pode ocorrer, você deve compreender as razões pelas quais os usuários podem fazer solicitações de mudança:

1. O software não é bom o suficiente e não faz o que os clientes querem que ele faça. Portanto, eles solicitam mudanças ofereçam a funcionalidade exigida.

2. Alternativamente, o software pode ser muito bom e por isso é ampla e fortemente usado. As solicitações de mudança podem ser geradas porque existem muitos usuários de software que pensam criativamente em novas coisas que poderiam ser feitas com o software.

Portanto, aumentar o envolvimento do cliente no processo pode reduzir o número de solicitações de mudança de produtos com os quais os clientes estavam descontentes. As mudanças de processo têm sido eficazes e tornaram o software mais usável e adequado. No entanto, as mudanças de processo podem não ter funcionado e os clientes podem decidir buscar um sistema alternativo. O número de solicitações de mudança pode diminuir porque o produto perdeu parte do mercado para um produto rival e, consequentemente, existem menos usuários.

Por outro lado, as mudanças de processo poderiam trazer muitos clientes novos, felizes, que desejem participar no processo de desenvolvimento de produto. Portanto, eles geram mais solicitações de mudança. As mudanças no processo de tratamento de solicitações de mudança podem contribuir com esse aumento. Se a empresa é mais sensível aos clientes, estes podem gerar um número maior de solicitações de mudança, pois sabem que essas solicitações serão levadas a sério. Eles acreditam que suas sugestões provavelmente serão incorporadas em versões posteriores do software. Então, o número de solicitações de mudança pode ter aumentado porque os sites de beta-teste não eram típicos do maior uso do programa.

Para analisar os dados de solicitação de mudança, não basta saber o número de solicitações de mudança. Você precisa saber quem fez a solicitação, como essas pessoas usam o software e por que a solicitação foi feita. Também precisa de informações sobre fatores externos, como modificações nos procedimentos de solicitação de mudança ou alterações de mercado que podem ter efeito. Com essas informações, é possível descobrir se as mudanças de processo foram eficazes no aumento da qualidade de produto.

Isso ilustra as dificuldades de compreensão dos efeitos das mudanças e a abordagem 'científica' para esse problema é reduzir o número de fatores que possam afetar as medições feitas. No entanto, processos e produtos que estão sendo medidos não são isolados de seu ambiente. O ambiente de negócios está mudando constantemente e é impossível evitar mudanças nas práticas de trabalho só porque eles podem fazer comparações de dados inválidos. Assim, dados quantitativos sobre as atividades humanas não podem sempre ser tomados pelo valor de face. Muitas vezes, as razões pelas quais um valor medido se altera são ambíguas. Essas razões devem ser pesquisadas em detalhes antes de se tirarem conclusões a respeito de quaisquer medições efetuadas anteriormente.

PONTOS IMPORTANTES

- O gerenciamento de qualidade de software visa assegurar que o software tenha um pequeno número de defeitos e que se adeque aos padrões de manutenibilidade, confiabilidade, portabilidade etc. em vigor. Inclui a definição de padrões para produtos e processos e o estabelecimento de processos para verificar se tais padrões foram seguidos.
- Os padrões de software são importantes para a garantia de qualidade, pois representam uma identificação das 'melhores práticas'. Durante o desenvolvimento de software, os padrões fornecem uma base sólida para a construção de software de boa qualidade.
- Você deve documentar um conjunto de procedimentos de garantia de qualidade em um manual de qualidade organizacional. Este pode ser baseado no modelo genérico para um manual de qualidade sugerido na norma ISO 9001.
- Revisões dos entregáveis de processos de software envolvem uma equipe de pessoas que verifica se os padrões de qualidade estão sendo seguidos. Revisões são técnicas largamente usadas para avaliação de qualidade.
- Em uma inspeção de programa ou revisão em pares, uma pequena equipe verifica o código sistematicamente. Ela lê o código em detalhes procurando por possíveis erros e omissões. Em seguida, os problemas detectados são discutidos em uma reunião de revisão de código.
- A medição de software pode ser usada para coletar dados quantitativos sobre o software e o processo de software. Você pode ser capaz de usar os valores das métricas de software que são coletadas para fazer inferências sobre a qualidade de produto e de processo.
- Métricas de qualidade de produto são particularmente úteis para realçar componentes anômalos que podem ter problemas de qualidade. Em seguida, esses componentes devem ser analisados em mais detalhes.

 LEITURA COMPLEMENTAR 

Metrics and Models for Software Quality Engineering. 2nd edition. Discussão muito abrangente sobre as métricas de software que abrange o processo, o produto e as métricas orientadas a objetos. Ela também inclui algumas informações básicas sobre a matemática necessária para desenvolver e entender modelos baseados em medição de software. (KAN, S. H. *Metrics and Models for Software Quality Engineering*. 2. ed. Addison-Wesley, 2003.)

Software Quality Assurance: From Theory to Implementation. Um olhar excelente, atualizado sobre os princípios e as práticas de garantia de qualidade de software. Inclui uma discussão sobre normas tais como a ISO 9001. (GALIN, D. *Software Quality Assurance: From Theory to Implementation*. Addison-Wesley, 2004.)

'A Practical Approach for Quality-Driven Inspections'. Muitos artigos sobre inspeções são bastante antigos e não consideram as práticas modernas de desenvolvimento de software. Esse artigo relativamente recente descreve um método de inspeção que aborda alguns dos problemas de uso de inspeção e sugere ela pode ser usada em um ambiente de desenvolvimento moderno. (DENCER, C.; SHULL, F. *IEEE Software*, v. 24, n. 2, mar.-abr. 2007.) Disponível em: <<http://dx.doi.org/10.1109/MS.2007.31>>.

'Misleading Metrics and Unsound Analyses'. Excelente artigo dos principais pesquisadores de métricas que aborda as dificuldades de compreender o que as medições realmente significam. (KITCHENHAM, B.; JEFFREY, R.; CONNAUGHTON, C. *IEEE Software*, v. 24, n. 2, mar.-abr. 2007.) Disponível em: <<http://dx.doi.org/10.1109/MS.2007.49>>.

'The Case for Quantitative Project Management'. Introdução para uma seção especial na revista que inclui dois outros artigos sobre gerenciamento de projetos quantitativos. Argumenta acerca de mais pesquisas em métricas e medições para melhorar o gerenciamento de projetos de software. (CURTIS, B. et al *IEEE Software*, v. 25, n. 3, mai.-jun. 2008.) Disponível em: <<http://dx.doi.org/10.1109/MS.2008.80>>.

 EXERCÍCIOS 

- 24.1** Explique por que um processo de software de alta qualidade deve levar a produtos de software de alta qualidade. Discuta possíveis problemas com esse sistema de gerenciamento de qualidade.
- 24.2** Explique como os padrões podem ser usados para capturar a sabedoria organizacional a respeito de métodos eficazes de desenvolvimento de software. Sugira quatro tipos de conhecimentos que possam ser capturados em normas organizacionais.
- 24.3** Discuta a avaliação de qualidade de software de acordo com os atributos de qualidade mostrados na Tabela 24.1. Você deve considerar cada atributo e explicar como ele pode ser avaliado.
- 24.4** Projete um formulário eletrônico que possa ser usado para registrar comentários de revisão e que poderia ser usado para enviar por e-mail os comentários para os revisores.
- 24.5** Descreva rapidamente possíveis padrões que possam ser usados para:
 - O uso de construções de controle em C, C# ou Java;
 - Relatórios que possam ser submetidos a um projeto de formatura em uma universidade;
 - O processo de fazer e aprovar mudanças de programa (ver Capítulo 26);
 - O processo de comprar e instalar um novo computador.
- 24.6** Suponha que você trabalhe para uma organização que desenvolve produtos de banco de dados para indivíduos e empresas de pequeno porte. Essa organização está interessada na quantificação de seu desenvolvimento de software. Escreva um relatório sugerindo métricas adequadas e sugira como estas podem ser coletadas.
- 24.7** Explique por que inspeções de programa são uma técnica eficaz para descobrir erros em um programa. Que tipos de erros são improváveis de serem descobertos por meio de inspeções?
- 24.8** Explique por que as métricas de projeto são, por si só, um método inadequado de previsão de qualidade de projeto.
- 24.9** Explique por que é difícil validar os relacionamentos entre os atributos internos de produto, como complexidade ciclomática e atributos externos, como a manutenibilidade.
- 24.10** Um colega, que é um excelente programador, produz softwares com poucos defeitos, mas constantemente ignora os padrões de qualidade da organização. Como seus gerentes deveriam reagir a esse comportamento?

 REFERÊNCIAS 

- BAMFORD, R.; DEIBLER, W. J. (Orgs.). ISO 9001:2000 for Software and Systems Providers: An Engineering Approach. Boca Raton, Flórida: CRC Press, 2003.
- BARNARD, J.; PRICE, A. Managing Code Inspection Information. *IEEE Software*, v. 11, n. 2, 1994, p. 59-69.
- BASILI, V. R.; ROMBACH, H. D. The TAME project: Towards Improvement-Oriented Software Environments. *IEEE Trans. on Software Eng.*, v. 14, n. 6, 1988, p. 758-773.
- BOEHM, B. W.; BROWN, J. R.; KASPAR, H.; LIPOW, M.; MacLEOD, G.; MERRIT, M. *Characteristics of Software Quality*. Amsterdam: North-Holland, 1978.
- CHIDAMBER, S.; KEMERER, C. A Metrics Suite for Object-Oriented Design. *IEEE Trans. on Software Eng.*, v. 20, n. 6, 1994, p. 476-493.
- CROSBY, P. *Quality is Free*. Nova York: McGraw-Hill, 1979.
- EL-AMAM, K. Object-oriented Metrics: A Review of Theory and Practice. National Research Council of Canada. 2001. Disponível em: <<http://seg.iit.nrc.ca/English/abstracts/NRC44190.html>>.
- ENDRES, A.; ROMBACH, D. *Empirical Software Engineering*: A Handbook of Observations, Laws and Theories. Harlow, Reino Unido: Addison-Wesley, 2003.
- FAGAN, M. E. Design and code inspections to reduce errors in program development. *IBM Systems J.*, v. 15, n. 3, 1976, p. 182-211.
- _____. Advances in Software Inspections. *IEEE Trans. on Software Eng.*, v. SE-12, n. 7, 1986.
- GILB, T.; GRAHAM, D. *Software Inspection*. Wokingham: Addison-Wesley, 1993.
- GRADY, R. B. Practical Results from Measuring Software Quality. *Comm. ACM*, v. 36, n. 11, 1993, p. 62-68.
- GUNNING, R. *Techniques of Clear Writing*. Nova York: McGraw-Hill, 1962.
- HALL, T.; FENTON, N. Implementing Effective Software Metrics Programs. *IEEE Software*, v. 14, n. 2, 1997, p. 55-64.
- HUMPHREY, W. *Managing the Software Process*. Reading, Mass.: Addison-Wesley, 1989.
- IEC. Standard IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission: Geneva, 1998.
- IEEE. *IEEE Software Engineering Standards Collection on CD-ROM*. Los Alamitos, Ca.: IEEE Computer Society Press, 2003.
- INCE, D. *ISO 9001 and Software Quality Assurance*. Londres: McGraw-Hill, 1994.
- KILPI, T. Implementing a Software Metrics Program at Nokia. *IEEE Software*, v. 18, n. 6, 2001, p. 72-77.
- KITCHENHAM, B. Measuring Software Development. In: *Software Reliability Handbook*. ROOK, P. (Org.). Amsterdam: Elsevier, 1990, p. 303-331.
- McCONNELL, S. *Code Complete: A Practical Handbook of Software Construction*. 2. ed. Seattle: Microsoft Press, 2004.
- MILLS, H. D.; DYER, M.; LINGER, R. Cleanroom Software Engineering. *IEEE Software*, v. 4, n. 5, 1987, p. 19-25.
- OFFEN, R. J.; JEFFREY, R. Establishing Software Measurement Programs. *IEEE Software*, v. 14, n. 2, 1997, p. 45-54.
- STALHANE, T.; HANSSEN, G. K. The application of ISO 9001 to agile software development. *9th International Conference on Product Focused Software Process Improvement*. PROFES 2008, Monte Porzio Catone, Itália: Springer, 2008.