A natureza do software 1

Depois de me mostrar a construção mais recente de um dos games de tiro em primeira pessoa mais populares do mundo, o jovem desenvolvedor riu. "Você não joga, né?", ele perguntou.

Eu sorri. "Como adivinhou?"

O jovem estava de bermuda e camiseta. Sua perna balançava para cima e para baixo como um pistão, queimando a tensa energia que parecia ser comum entre seus colegas.

"Porque, se jogasse", ele disse, "estaria muito mais empolgado. Você acabou de ver nosso mais novo produto, algo que nossos clientes matariam para ver... sem trocadilhos".

Estávamos na área de desenvolvimento de uma das empresas de games mais bem-sucedidas do planeta. Ao longo dos anos, as gerações anteriores do game que ele demonstrou venderam mais de 50 milhões de cópias e geraram uma receita de bilhões de dólares.

"Então, quando essa versão estará no mercado?", perguntei.

Ele encolheu os ombros. "Em cerca de cinco meses. Ainda temos muito trabalho a fazer". Ele era responsável pela jogabilidade e pela funcionalidade de inteligência artificial de um aplicativo que abrangia mais de três milhões de linhas de código.

"Vocês usam técnicas de engenharia de software?", perguntei, meio que esperando sua risada e sua resposta negativa.

Conceitos-chave

PANORAMA

O que é? Software de computador é o produto que profissionais de software desenvolvem e ao qual dão suporte no longo prazo. Abrange programas executáveis em um computador de qualquer porte ou arquitetura, conteúdos (apresentados à medida que os programas são executados), informações descritivas tanto na forma impressa (*hard copy*) quanto na virtual, abrangendo praticamente qualquer mídia eletrônica.

Quem realiza? Os engenheiros de software criam e dão suporte a ele, e praticamente todos que têm contato com o mundo industrializado o utilizam, direta ou indiretamente.

Por que é importante? Porque afeta quase todos os aspectos de nossa vida e se difundiu no comércio, na cultura e em nossas atividades cotidianas.

Quais são as etapas envolvidas? Os clientes e outros envolvidos expressam a necessidade pelo software de computador, os engenheiros constroem o produto de software e os usuários o utilizam para resolver um problema específico ou para tratar de uma necessidade específica.

Qual é o artefato? Um programa de computador que funciona em um ou mais ambientes específicos e atende às necessidades de um ou mais usuários.

Como garantir que o trabalho foi realizado corretamente? Se você é engenheiro de software, aplique as ideias contidas no restante deste livro. Se for usuário, conheça sua necessidade e seu ambiente e escolha uma aplicação que seja a mais adequada a ambos.

Ele fez uma pausa e pensou por uns instantes. Então, lentamente, fez que sim com a cabeça. "Adaptamos às nossas necessidades, mas, claro, usamos".

"Onde?", perguntei, sondando. "Geralmente, nosso problema é traduzir os requisitos que os criativos nos dão". "Os criativos?", interrompi. "Você sabe, os caras que projetam a história, os personagens, todas as coisas que tornam o jogo um sucesso. Temos de pegar o que eles nos dão e produzir um conjunto de requisitos técnicos que nos permita construir o jogo."

"E depois os requisitos são fixados?"

Ele encolheu os ombros. "Precisamos ampliar e adaptar a arquitetura da versão anterior do jogo e criar um novo produto. Temos de criar código a partir dos requisitos, testá-lo com construções diárias e fazer muitas coisas que seu livro recomenda."

"Conhece meu livro?" Eu estava sinceramente surpreso. "Claro, usei na faculdade. Há muita coisa lá."

"Falei com alguns de seus colegas aqui e eles são mais céticos a respeito do material de meu livro."

Ele franziu as sobrancelhas. "Olha, não somos um departamento de TI nem uma empresa aeroespacial, então, temos de adaptar o que você defende. Mas o resultado final é o mesmo – precisamos criar um produto de alta qualidade, e o único jeito de conseguirmos isso sempre é adaptar nosso próprio subconjunto de técnicas de engenharia de software."

"E como seu subconjunto mudará com o passar dos anos?"

Ele fez uma pausa como se estivesse pensando no futuro. "Os games vão se tornar maiores e mais complexos, com certeza. E nossos cronogramas de desenvolvimento vão ser mais apertados, à medida que a concorrência surgir. Lentamente, os próprios jogos nos obrigarão a aplicar um pouco mais de disciplina de desenvolvimento. Se não fizermos isso, estaremos mortos."

"Ideias e descobertas tecnológicas são os mecanismos que impulsionam o crescimento econômico."

Wall Street Journal

Software de computador continua a ser a tecnologia mais importante no cenário mundial. E é também um ótimo exemplo da lei das consequências não intencionais. Há 60 anos, ninguém poderia prever que o software se tornaria uma tecnologia indispensável para negócios, ciência e engenharia; que software viabilizaria a criação de novas tecnologias (por exemplo, engenharia genética e nanotecnologia), a extensão de tecnologias existentes (por exemplo, telecomunicações) e a mudança radical nas tecnologias mais antigas (por exemplo, a mídia); que software se tornaria a força motriz por trás da revolução do computador pessoal; que aplicativos de software seriam comprados pelos consumidores com seus smartphones; que o software evoluiria lentamente de produto para serviço, à medida que empresas de software "sob encomenda" oferecessem funcionalidade imediata (*just-in-time*), via um navegador Web; que uma empresa de software se tornaria maior e mais influente do que todas as empresas da era industrial; que uma vasta rede comandada por software evoluiria e modificaria tudo: de pesquisa em bibliotecas a compras feitas pelos consumidores, de discursos políticos a comportamentos de namoro entre jovens e adultos não tão jovens.

Ninguém poderia prever que o software seria incorporado a sistemas de todas as áreas: transportes, medicina, telecomunicações, militar, industrial, entretenimento, máquinas de escritório... a lista é quase infindável. E se você

acredita na lei das consequências não intencionais, há muitos efeitos que ainda não somos capazes de prever.

Também ninguém poderia prever que milhões de programas de computador teriam de ser corrigidos, adaptados e ampliados à medida que o tempo passasse. A realização dessas atividades de "manutenção" absorve mais pessoas e recursos do que todo o esforço aplicado na criação de um novo software.

À medida que aumenta a importância do software, a comunidade da área tenta criar tecnologias que tornem mais fácil, mais rápido e mais barato desenvolver e manter programas de computador de alta qualidade. Algumas dessas tecnologias são direcionadas a um campo de aplicação específico (por exemplo, projeto e implementação de sites); outras são focadas em um campo de tecnologia (por exemplo, sistemas orientados a objetos ou programação orientada a aspectos); e outras ainda são de bases amplas (por exemplo, sistemas operacionais como o Linux). Entretanto, nós ainda temos de desenvolver uma tecnologia de software que faça tudo isso – e a probabilidade de surgir tal tecnologia no futuro é pequena. Ainda assim, as pessoas apostam seus empregos, seu conforto, sua segurança, seu entretenimento, suas decisões e suas próprias vidas em software. Tomara que estejam certas.

Este livro apresenta uma estrutura que pode ser utilizada por aqueles que desenvolvem software – pessoas que devem fazê-lo corretamente. A estrutura abrange um processo, um conjunto de métodos e uma gama de ferramentas que chamaremos de *engenharia de software*.

1.1 A natureza do software

Software é tanto um produto quanto um veículo que distribui um produto.

Hoje, o software tem um duplo papel. Ele é um produto e, ao mesmo tempo, o veículo para distribuir um produto. Como produto, fornece o potencial computacional representado pelo hardware ou, de forma mais abrangente, por uma rede de computadores que podem ser acessados por hardware local. Seja residindo em um celular, seja em um tablet, em um computador de mesa ou em um mainframe, o software é um transformador de informações – produzindo, gerenciando, adquirindo, modificando, exibindo ou transmitindo informações que podem ser tão simples quanto um único bit ou tão complexas quanto uma apresentação multimídia derivada de dados obtidos de dezenas de fontes independentes. Como veículo de distribuição do produto, o software atua como a base para o controle do computador (sistemas operacionais), a comunicação de informações (redes) e a criação e o controle de outros programas (ferramentas de software e ambientes).

O software distribui o produto mais importante de nossa era – a *informação*. Ele transforma dados pessoais (por exemplo, transações financeiras de um indivíduo) de modo que possam ser mais úteis em determinado contexto; gerencia informações comerciais para aumentar a competitividade; fornece um portal para redes mundiais de informação (Internet) e os meios para obter informações sob todas as suas formas. Também propicia um veículo que pode ameaçar a privacidade pessoal e é uma porta que permite a pessoas mal-intencionadas cometer crimes.

"Software é um lugar onde sonhos são plantados e pesadelos são colhidos, um pântano abstrato e místico onde demônios terríveis competem com mágicas panaceias, um mundo de lobisomens e balas de prata."

Brad J. Cox

O papel do software passou por uma mudança significativa no decorrer da metade final do século passado. Aperfeiçoamentos significativos no desempenho do hardware, mudanças profundas nas arquiteturas computacionais, um vasto aumento na capacidade de memória e armazenamento e uma ampla variedade exótica de opções de entrada e saída; tudo isso resultou em sistemas computacionais mais sofisticados e complexos. Sofisticação e complexidade podem produzir resultados impressionantes quando um sistema é bem-sucedido; porém, também podem trazer enormes problemas para aqueles que precisam desenvolver e projetar sistemas robustos.

Atualmente, uma enorme indústria de software tornou-se fator dominante nas economias do mundo industrializado. Equipes de especialistas em software, cada qual concentrando-se numa parte da tecnologia necessária para distribuir uma aplicação complexa, substituíram o programador solitário de antigamente. Ainda assim, as questões levantadas por esse programador solitário continuam as mesmas hoje, quando os modernos sistemas computacionais são desenvolvidos:[1]

- Por que a conclusão de um software leva tanto tempo?
- Por que os custos de desenvolvimento são tão altos?
- Por que não conseguimos encontrar todos os erros antes de entregarmos o software aos clientes?
- Por que gastamos tanto tempo e esforço realizando a manutenção de programas existentes?
- Por que ainda temos dificuldades de medir o progresso de desenvolvimento e a manutenção de um software?

Essas e muitas outras questões demonstram a preocupação com o software e a maneira como é desenvolvido – uma preocupação que tem levado à adoção da prática da engenharia de software.

1.1.1 Definição de software

Hoje, a maior parte dos profissionais e muitos outros integrantes do público em geral acham que entendem de software. Mas será que entendem mesmo?

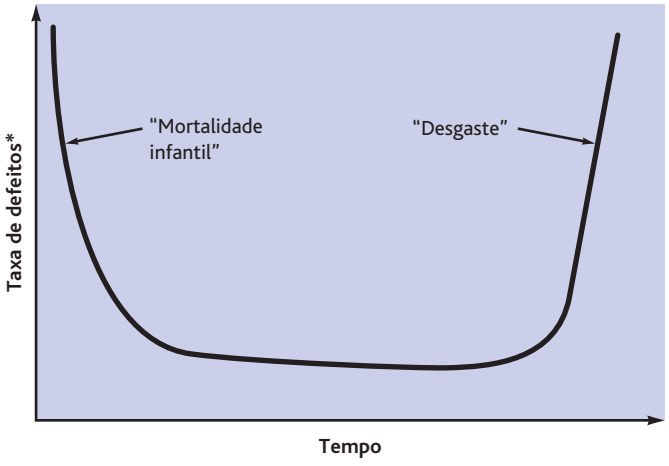
Uma descrição de software em um livro-texto poderia ser a seguinte:

Como devemos definir *software*?

> Software consiste em: (1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados; (2) estruturas de dados que possibilitam aos programas manipular informações adequadamente; e (3) informação descritiva, tanto na forma impressa quanto na virtual, descrevendo a operação e o uso dos programas.

Sem dúvida, poderíamos dar outras definições mais completas, mas, provavelmente, uma definição mais formal não melhoraria, consideravelmente, a compreensão do que é software.

[1] Em um excelente livro de ensaio sobre o setor de software, Tom DeMarco [DeM95] contesta. Segundo ele: "Em vez de perguntar por que software custa tanto, precisamos começar perguntando: 'O que fizemos para que o software atual custe tão pouco?' A resposta a essa pergunta nos ajudará a continuar com o extraordinário nível de realização que tem distinguido a indústria de software".



Caso queira reduzir a deterioração do software, terá de fazer um projeto de software melhor (Capítulos 12 a 18).

FIGURA 1.1 Curva de defeitos para hardware.

Para conseguir isso, é importante examinar as características do software que o tornam diferenciado de outras coisas que os seres humanos constroem. Software é mais um elemento de sistema lógico do que físico. Portanto, o software tem uma característica fundamental que o torna consideravelmente diferente do hardware: *software não "se desgasta"*.

A Figura 1.1 representa a taxa de defeitos em função do tempo para hardware. Essa relação, normalmente denominada "curva da banheira", indica que o hardware apresenta taxas de defeitos relativamente altas no início de sua vida (geralmente, atribuídas a defeitos de projeto ou de fabricação); os defeitos são corrigidos; e a taxa cai para um nível estável (felizmente, bastante baixo) por certo período. Entretanto, à medida que o tempo passa, a taxa aumenta novamente, conforme os componentes de hardware sofrem os efeitos cumulativos de poeira, vibração, impactos, temperaturas extremas e vários outros fatores maléficos do ambiente. Resumindo, o hardware começa a *se desgastar*.

Software não é suscetível aos fatores maléficos do ambiente que fazem com que o hardware se desgaste. Portanto, teoricamente, a curva da taxa de defeitos para software deveria assumir a forma da "curva idealizada", mostrada na Figura 1.2. Defeitos ainda não descobertos irão resultar em altas taxas logo no início da vida de um programa. Entretanto, esses serão corrigidos, e a curva se achata, como mostrado. A curva idealizada é uma simplificação grosseira de modelos de defeitos reais para software. Porém, a implicação é clara: software não se desgasta. Mas *deteriora!*

Essa aparente contradição pode ser elucidada pela curva real apresentada na Figura 1.2. Durante sua vida[2], o software passará por alterações. À

* N. de R.T.: Os defeitos do software nem sempre se manifestam como falha, geralmente devido a tratamentos dos erros decorrentes desses defeitos pelo software. Esses conceitos serão mais detalhados e diferenciados nos capítulos sobre qualidade. Neste ponto, optou-se por traduzir *failure rate* por taxa de defeitos, sem prejuízo para a assimilação dos conceitos apresentados pelo autor neste capítulo.

[2] De fato, desde o momento em que o desenvolvimento começa, e muito antes de a primeira versão ser entregue, podem ser solicitadas mudanças por uma variedade de diferentes envolvidos.

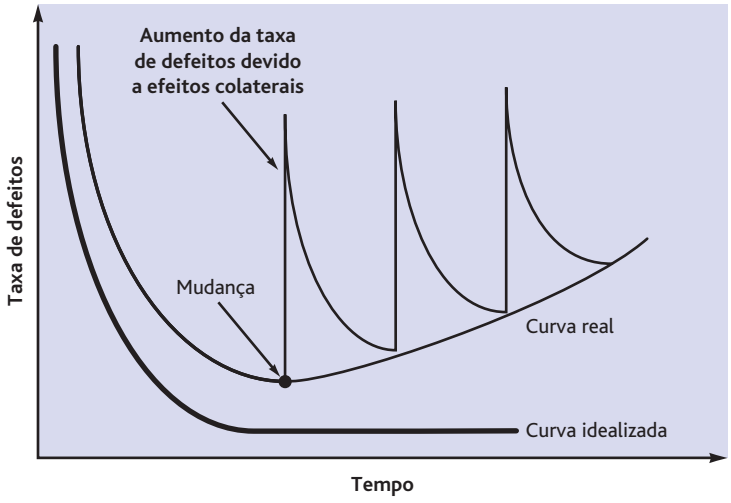


FIGURA 1.2 Curva de defeitos para software.

Os métodos de engenharia de software tentam reduzir ao máximo a magnitude das elevações (picos) e a inclinação da curva real da Figura 1.2.

medida que elas ocorram, é provável que sejam introduzidos erros, fazendo com que a curva de taxa de defeitos se acentue, conforme mostrado na "curva real" (Figura 1.2). Antes que a curva possa retornar à taxa estável original, outra alteração é requisitada, fazendo com que a curva se acentue novamente. Lentamente, o nível mínimo da taxa começa a aumentar – o software está deteriorando devido à modificação.

Outro aspecto do desgaste ilustra a diferença entre hardware e software. Quando um componente de hardware se desgasta, ele é substituído por uma peça de reposição. Não existem peças de reposição de software. Cada defeito de software indica um erro no projeto ou no processo pelo qual o projeto foi traduzido em código de máquina executável. Portanto, as tarefas de manutenção de software, que envolvem solicitações de mudanças, implicam complexidade consideravelmente maior do que a de manutenção de hardware.

1.1.2 Campos de aplicação de software

Atualmente, sete grandes categorias de software apresentam desafios contínuos para os engenheiros de software:

Software de sistema Conjunto de programas feito para atender a outros programas. Certos softwares de sistema (por exemplo, compiladores, editores e utilitários para gerenciamento de arquivos) processam estruturas de informação complexas; porém, determinadas.[3] Outras aplicações de sistema (por exemplo, componentes de sistema operacional, drivers, software de rede, processadores de telecomunicações) processam dados amplamente indeterminados.

[3] Um software é *determinado* se a ordem e o *timing* (periodicidade, frequência, medidas de tempo) de entradas, processamento e saídas forem previsíveis. É *indeterminado* se a ordem e o *timing* de entradas, processamento e saídas não puderem ser previstos antecipadamente.

Software de aplicação Programas independentes que solucionam uma necessidade específica de negócio. Aplicações nessa área processam dados comerciais ou técnicos de uma forma que facilite operações comerciais ou tomadas de decisão administrativas/técnicas.

Uma das mais abrangentes bibliotecas de shareware/freeware (software compartilhado/livre) pode ser encontrada em **shareware.cnet.com**.

Software de engenharia/científico Uma ampla variedade de programas de "cálculo em massa" que abrangem astronomia, vulcanologia, análise de estresse automotivo, dinâmica orbital, projeto auxiliado por computador, biologia molecular, análise genética e meteorologia, entre outros.

Software embarcado Residente num produto ou sistema e utilizado para implementar e controlar características e funções para o usuário e para o próprio sistema. Executa funções limitadas e específicas (por exemplo, controle do painel de um forno micro-ondas) ou fornece função significativa e capacidade de controle (por exemplo, funções digitais de automóveis, tal como controle do nível de combustível, painéis de controle e sistemas de freio).

Software para linha de produtos Projetado para prover capacidade específica de utilização por muitos clientes diferentes. Software para linha de produtos pode se concentrar em um mercado hermético e limitado (por exemplo, produtos de controle de inventário) ou lidar com consumidor de massa.

Aplicações Web/aplicativos móveis Esta categoria de software voltada às redes abrange uma ampla variedade de aplicações, contemplando aplicativos voltados para navegadores e software residente em dispositivos móveis.

Software de inteligência artificial Faz uso de algoritmos não numéricos para solucionar problemas complexos que não são passíveis de computação ou de análise direta. Aplicações nessa área incluem: robótica, sistemas especialistas, reconhecimento de padrões (de imagem e de voz), redes neurais artificiais, prova de teoremas e jogos.

"Para mim, o computador é a ferramenta mais extraordinária que já inventamos. É o equivalente de uma bicicleta para nossas mentes."

Steve Jobs

Milhões de engenheiros de software em todo o mundo trabalham arduamente em projetos de software em uma ou mais dessas categorias. Em alguns casos, novos sistemas estão sendo construídos, mas, em muitos outros, aplicações já existentes estão sendo corrigidas, adaptadas e aperfeiçoadas. Não é incomum um jovem engenheiro de software trabalhar em um programa mais velho do que ele! Gerações passadas de pessoal de software deixaram um legado em cada uma das categorias discutidas. Espera-se que o legado a ser deixado por esta geração facilite o trabalho dos futuros engenheiros de software.

1.1.3 Software legado

Centenas de milhares de programas de computador caem em um dos sete amplos campos de aplicação discutidos na subseção anterior. Alguns deles são software de ponta – recém-lançados para indivíduos, indústria e governo. Outros programas são mais antigos – em alguns casos *muito mais* antigos.

Esses programas mais antigos – frequentemente denominados *software legado* – têm sido foco de contínua atenção e preocupação desde os anos 1960. Dayani-Fard e seus colegas [Day99] descrevem software legado da seguinte maneira:

> Sistemas de software legado... foram desenvolvidos décadas atrás e têm sido continuamente modificados para se adequar às mudanças dos requisitos de negócio e a plataformas computacionais. A proliferação de tais sistemas está causando dores de cabeça para grandes organizações que os consideram dispendiosos de manter e arriscados de evoluir.

Liu e seus colegas [Liu98] ampliam essa descrição, observando que "muitos sistemas legados permanecem dando suporte para funções de negócio vitais e são 'indispensáveis' para o mesmo". Por isso, um software legado é caracterizado pela longevidade e criticidade de negócios.

O que faço caso encontre um sistema legado de baixa qualidade?

Infelizmente, de vez em quando uma característica adicional está presente em software legado – a *baixa qualidade*.[4] Às vezes, os sistemas legados têm projetos inextensíveis, código de difícil entendimento, documentação deficiente ou inexistente, casos de teste e resultados que nunca foram documentados, um histórico de alterações mal gerenciado – a lista pode ser bastante longa. Ainda assim, esses sistemas dão suporte a "funções vitais de negócio e são indispensáveis para ele". O que fazer?

A única resposta adequada talvez seja: *não faça nada*, pelo menos até que o sistema legado tenha que passar por alguma modificação significativa. Se o software legado atende às necessidades de seus usuários e funciona de forma confiável, ele não está "quebrado" e não precisa ser "consertado". Entretanto, com o passar do tempo, esses sistemas evoluem devido a uma ou mais das razões a seguir:

Que tipos de mudanças são feitas em sistemas legados?

- O software deve ser adaptado para atender às necessidades de novos ambientes ou de novas tecnologias computacionais.
- O software deve ser aperfeiçoado para implementar novos requisitos de negócio.
- O software deve ser expandido para torná-lo capaz de funcionar com outros bancos de dados ou com sistemas mais modernos.
- O software deve ser rearquitetado para torná-lo viável dentro de um ambiente computacional em evolução.

Todo engenheiro de software deve reconhecer que modificações são naturais. Não tente combatê-las.

Quando essas modalidades de evolução ocorrem, um sistema legado deve passar por reengenharia (Capítulo 36) para que permaneça viável no futuro. O objetivo da engenharia de software moderna é "elaborar metodologias baseadas na noção de evolução"; isto é, na noção de que os sistemas de software modificam-se continuamente, novos sistemas são construídos a partir dos antigos e... todos devem agir em grupo e cooperar uns com os outros" [Day99].

[4] Nesse caso, a qualidade é julgada em termos da engenharia de software moderna – um critério um tanto injusto, já que alguns conceitos e princípios da engenharia de software moderna talvez não tenham sido bem entendidos na época em que o software legado foi desenvolvido.

1.2 A natureza mutante do software

A evolução de quatro categorias amplas de software domina o setor. Ainda assim, há pouco mais de uma década essas categorias estavam em sua infância.

1.2.1 WebApps

Nos primórdios da World Wide Web (por volta de 1990 a 1995), os *sites* eram formados por nada mais do que um conjunto de arquivos de hipertexto linkados e que apresentavam informações usando texto e gráficos limitados. Com o tempo, o crescimento da linguagem HTML, via ferramentas de desenvolvimento (por exemplo, XML, Java), tornou possível aos engenheiros da Internet oferecerem capacidade computacional juntamente com as informações. Nasciam, então, os *sistemas e aplicações baseados na Web*[5] (refiro-me a eles coletivamente como *WebApps*).

Atualmente, as WebApps evoluíram para sofisticadas ferramentas computacionais que não apenas oferecem funções especializadas (*stand-alone functions*) ao usuário, como também foram integradas aos bancos de dados corporativos e às aplicações de negócio.

"Quando notarmos qualquer tipo de estabilidade, a Web terá se transformado em algo completamente diferente."

Louis Monier

Há uma década, as WebApps "envolvem(iam) uma mistura de publicação impressa e desenvolvimento de software, de marketing e computação, de comunicações internas e relações externas e de arte e tecnologia" [Pow98]. Mas, hoje, fornecem potencial de computação total em muitas das categorias de aplicação registradas na seção 1.1.2.

No decorrer da década passada, tecnologias Semantic Web (muitas vezes denominadas Web 3.0) evoluíram para sofisticadas aplicações corporativas e para o consumidor, as quais abrangem "bancos de dados semânticos [que] oferecem novas funcionalidades que exigem links Web, representação [de dados] flexível e APIs de acesso externo" [Hen10]. Sofisticadas estruturas de dados relacionais levarão a WebApps inteiramente novas que permitirão acessar informações díspares de maneiras inéditas.

1.2.2 Aplicativos móveis

O termo *aplicativo* evoluiu para sugerir software projetado especificamente para residir em uma plataforma móvel (por exemplo, iOS, Android ou Windows Mobile). Na maioria dos casos, os aplicativos móveis contêm uma interface de usuário que tira proveito de mecanismos de interação exclusivos fornecidos pela plataforma móvel, da interoperabilidade com recursos baseados na Web que dão acesso a uma grande variedade de informações relevantes ao aplicativo e de capacidades de processamento local que coletam, analisam e formatam as informações de forma mais conveniente para a plataforma. Além disso, um aplicativo móvel fornece recursos de armazenamento persistente dentro da plataforma.

[5] No contexto deste livro, o termo *aplicação Web* (*WebApp*) engloba tudo, de uma simples página Web que possa ajudar um cliente a processar o pagamento do aluguel de um automóvel a um amplo site que fornece serviços de viagem completos para executivos e turistas. Dentro dessa categoria estão sites completos, funcionalidade especializada dentro de sites e aplicações para processamento de informações residentes na Internet ou em uma intranet ou extranet.

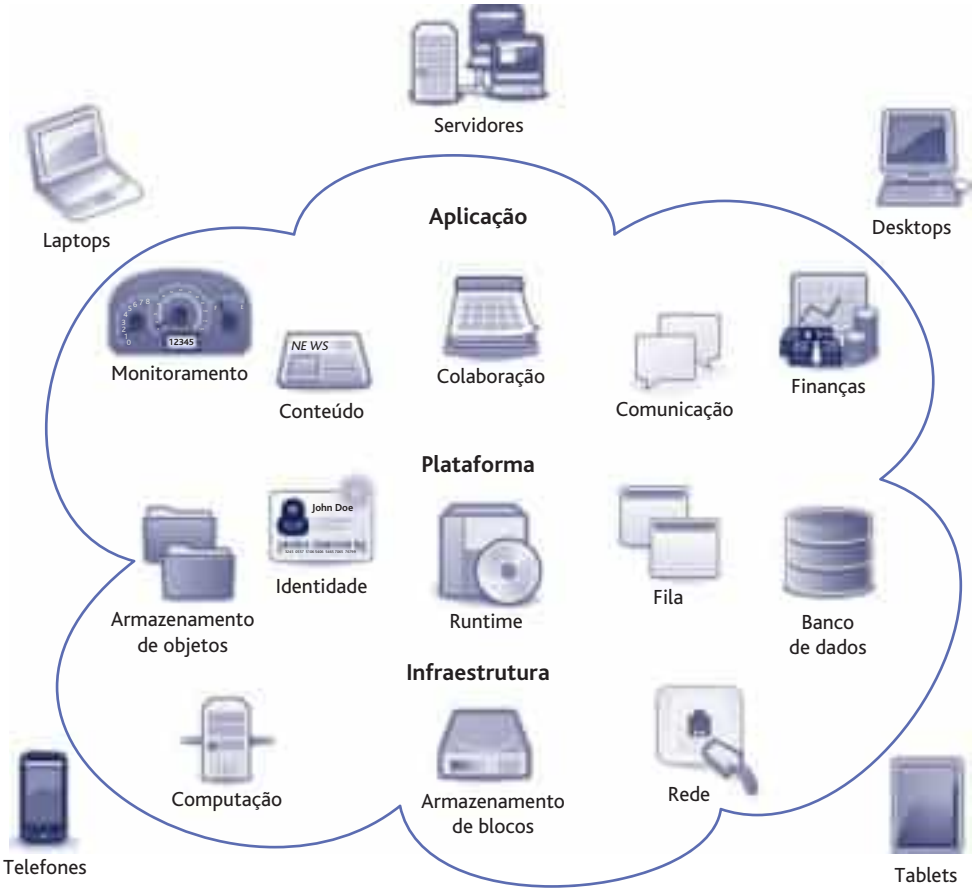


FIGURA 1.3 Arquitetura lógica da computação em nuvem [Wik13].

Qual a diferença entre uma WebApp e um aplicativo móvel?

É importante reconhecer que existe uma diferença sutil entre aplicações web móveis e aplicativos móveis. Uma *aplicação web móvel* (WebApp) permite que um dispositivo móvel tenha acesso a conteúdo baseado na web por meio de um navegador especificamente projetado para se adaptar aos pontos fortes e fracos da plataforma móvel. Um *aplicativo móvel* pode acessar diretamente as características do hardware do dispositivo (por exemplo, acelerômetro ou localização por GPS) e, então, fornecer os recursos de processamento e armazenamento local mencionados anteriormente. Com o passar do tempo, essa diferença entre WebApps móveis e aplicativos móveis se tornará indistinta, à medida que os navegadores móveis se tornarem mais sofisticados e ganharem acesso ao hardware e às informações em nível de dispositivo.

1.2.3 Computação em nuvem

A *computação em nuvem* abrange uma infraestrutura ou "ecossistema" que permite a qualquer usuário, em qualquer lugar, utilizar um dispositivo de computação para compartilhar recursos computacionais em grande escala. A arquitetura lógica global da computação em nuvem está representada na Figura 1.3.

De acordo com a figura, os dispositivos de computação residem fora da nuvem e têm acesso a uma variedade de recursos dentro dela. Esses recursos abrangem aplicações, plataformas e infraestrutura. Em sua forma mais simples, um dispositivo de computação externa acessa a nuvem por meio de um navegador Web ou software semelhante. A nuvem dá acesso a dados residentes nos bancos de dados e em outras estruturas de dados. Além disso, os dispositivos podem acessar aplicativos executáveis, que podem ser usados no lugar das aplicações residentes no dispositivo de computação.

A implementação da computação em nuvem exige o desenvolvimento de uma arquitetura que contenha serviços de front-end e de back-end. O *front-end* inclui o dispositivo cliente (usuário) e o software aplicativo (por exemplo, um navegador) que permite o acesso ao back-end. O *back-end* inclui servidores e recursos de computação relacionados, sistemas de armazenamento de dados (por exemplo, bancos de dados), aplicativos residentes no servidor e servidores administrativos que utilizam middleware para coordenar e monitorar o tráfego, estabelecendo um conjunto de protocolos de acesso à nuvem e aos seus recursos residentes [Str08].

A arquitetura da nuvem pode ser segmentada para dar acesso a uma variedade de diferentes níveis de acesso público total para arquiteturas de nuvem privadas, acessíveis somente a quem tenha autorização.

1.2.4 Software para linha de produtos (de software)

O Software Engineering Institute define uma *linha de produtos de software* como "um conjunto de sistemas de software que compartilham um conjunto comum de recursos gerenciados, satisfazendo as necessidades específicas de um segmento de mercado ou de uma missão em particular, desenvolvidos a partir de um conjunto comum de itens básicos, contemplando uma forma prescrita" [SEI13]. De certa maneira, a noção de linha de produtos de software relacionados não é nova, mas a ideia de uma linha de produtos de software – todos desenvolvidos com a mesma aplicação subjacente e com as mesmas arquiteturas de dados, sendo todos implementados com um conjunto de componentes de software reutilizáveis em toda a linha de produtos – proporciona um potencial significativo para a engenharia.

Uma linha de produtos de software compartilha um conjunto de itens que incluem requisitos (Capítulo 8), arquitetura (Capítulo 13), padrões de projeto (Capítulo 16), componentes reutilizáveis (Capítulo 14), casos de teste (Capítulos 22 e 23) e outros produtos de trabalho para a engenharia de software. Basicamente, uma linha de produtos de software resulta no desenvolvimento de muitos produtos projetados tirando proveito dos atributos comuns a tudo que é feito dentro da linha de produtos.

1.3 Resumo

Software é o elemento-chave na evolução de produtos e sistemas baseados em computador e é uma das mais importantes tecnologias no cenário mundial. Ao longo dos últimos 50 anos, o software evoluiu de uma ferramenta especializada em análise de informações e resolução de problemas para uma indústria

propriamente dita. Mesmo assim, ainda temos problemas para desenvolver software de boa qualidade dentro do prazo e orçamento estabelecidos.

Software – programas, dados e informações descritivas – contemplam uma ampla gama de áreas de aplicação e tecnologia. O software legado continua a representar desafios especiais àqueles que precisam fazer sua manutenção.

A natureza do software é mutante. As aplicações e os sistemas baseados na Internet passaram de simples conjuntos de conteúdo informativo para sofisticados sistemas que apresentam funcionalidade complexa e conteúdo multimídia. Embora essas WebApps possuam características e requisitos exclusivos, elas não deixam de ser um tipo de software. Os aplicativos móveis apresentam novos desafios, à medida que migram para uma ampla variedade de plataformas. A computação em nuvem transformará o modo de distribuir software e o seu ambiente. O software para linha de produtos oferece eficiências em potencial na maneira de construir software.

Problemas e pontos a ponderar

1.1 Dê no mínimo mais cinco exemplos de como a lei das consequências não intencionais se aplica a software de computador.

1.2 Forneça uma série de exemplos (positivos e negativos) que indiquem o impacto do software em nossa sociedade.

1.3 Dê suas próprias respostas para as cinco perguntas feitas no início da Seção 1.1. Discuta-as com seus colegas.

1.4 Muitas aplicações modernas mudam frequentemente – antes de serem apresentadas ao usuário e depois da primeira versão ser colocada em uso. Sugira algumas maneiras de construir software para impedir a deterioração decorrente de mudanças.

1.5 Considere as sete categorias de software apresentadas na Seção 1.1.2. Você acha que a mesma abordagem em relação à engenharia de software pode ser aplicada a cada uma delas? Justifique sua resposta.

Leituras e fontes de informação complementares[6]

Literalmente milhares de livros são escritos sobre software. A grande maioria trata de linguagens de programação ou aplicações de software, porém poucos tratam do software em si. Pressman e Herron (*Software Shock*, Dorset House, 1991) apresentaram uma discussão preliminar (dirigida ao grande público) sobre software e a maneira como os profissionais o desenvolvem. O best-seller de Negroponte (*Being Digital*, Alfred A. Knopf, 1995) dá uma visão geral da computação e seu impacto global no século 21. DeMarco (*Why Does Software Cost So Much*? Dorset House, 1995) produziu um conjunto de divertidos e perspicazes ensaios sobre software e o processo pelo qual ele é desenvolvi-

[6] A seção *Leitura e fontes de informação complementares* ao final de cada capítulo apresenta uma visão geral de publicações que podem ajudar a expandir o seu entendimento dos principais tópicos apresentados no capítulo. Criamos um site completo (em inglês) para dar suporte a este livro, no endereço **www.mhhe.com/pressman**. Entre os muitos temas tratados no site estão recursos de engenharia de software, capítulo por capítulo, até informações baseadas na Web que podem complementar o material apresentado em cada capítulo. Dentro desses recursos existe um link para a Amazon.com para cada livro mencionado nesta seção.

do. Ray Kurzweil (*How to Create a Mind,* Viking, 2013) discute como em breve o software imitará o pensamento humano e levará a uma "singularidade" na evolução de humanos e máquinas.

Keeves (*Catching Digital,* Business Infomedia Online, 2012) discute como os líderes empresariais devem se adaptar, à medida que o software evolui em um ritmo cada vez maior. Minasi (*The Software Conspiracy: Why Software Companies Put out Faulty Products, How They Can Hurt You, and What You Can Do*, McGraw-Hill, 2000) argumenta que o "flagelo moderno" dos bugs de software pode ser eliminado e sugere maneiras para concretizar isso. Eubanks (*Digital Dead End: Fighting for Social Justice in the Information Age,* MIT Press, 2011) e Compaine (*Digital Divide: Facing a Crisis or Creating a Mith*, MIT Press, 2001) defendem que a "separação" entre aqueles que têm acesso a fontes de informação (por exemplo, a Web) e aqueles que não o têm está diminuindo, à medida que avançamos na primeira década deste século. Kuniavsky (*Smart Things: Ubiquitous Computing User Experience Design,* Morgan Kaufman, 2010), Greenfield (*Everyware: The Dawning Age of Ubiquitous Computing*, New Riders Publishing, 2006) e Loke (*Context-Aware Pervasive Systems*: *Architectures for a New Breed of Applications*, Auerbach, 2006) introduzem o conceito de software "aberto" e preveem um ambiente sem fio no qual o software deve se adaptar às exigências que surgem em tempo real.

Uma ampla variedade de fontes de informação que discutem a natureza do software está disponível na Internet. Uma lista atualizada de referências relevantes (em inglês) para o processo do software pode ser encontrada no site: **www.mhhe.com/pressman.**

2 Engenharia de software

Conceitos-chave

Para desenvolver software que esteja preparado para enfrentar os desafios do século 21, devemos admitir alguns fatos:

- Software está profundamente incorporado em praticamente todos os aspectos de nossas vidas e, consequentemente, o número de pessoas interessadas nos recursos e nas funções oferecidas por determinada aplicação[1] tem crescido significativamente. *Depreende-se, portanto, que é preciso fazer um esforço conjunto para compreender o problema antes de desenvolver uma solução de software.*
- Os requisitos de tecnologia da informação demandados por pessoas, empresas e órgãos governamentais estão mais complexos a cada ano. Atualmente, equipes grandes desenvolvem programas de computador que, antigamente, eram desenvolvidos por um só indivíduo. Software sofisticado, outrora implementado em um ambiente computacional independente e previsível, hoje está incorporado em tudo, de produtos eletrônicos de consumo a equipamentos médicos e sistemas de armamento. *Depreende-se, portanto, que projetar se tornou uma atividade essencial.*

PANORAMA

O que é? A engenharia de software abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade.

Quem realiza? Os engenheiros de software aplicam o processo de engenharia de software.

Por que é importante? A engenharia de software é importante porque nos capacita para o desenvolvimento de sistemas complexos dentro do prazo e com alta qualidade. Ela impõe disciplina a um trabalho que pode se tornar caótico, mas também permite que as pessoas produzam software de computador adaptado à sua abordagem, da maneira mais conveniente às suas necessidades.

Quais são as etapas envolvidas? Cria-se software para computadores da mesma forma que qualquer produto bem-sucedido: aplicando-se um processo adaptável e ágil que conduza a um resultado de alta qualidade, atendendo às necessidades daqueles que usarão o produto. Aplica-se uma abordagem de engenharia de software.

Qual é o artefato? Do ponto de vista de um engenheiro de software, software é um conjunto de programas, conteúdo (dados) e outros artefatos. Porém, do ponto de vista do usuário, o artefato consiste em informações resultantes que, de alguma forma, tornam a vida dele melhor.

Como garantir que o trabalho foi realizado corretamente? Leia o restante deste livro, escolha as ideias aplicáveis ao software que você desenvolver e use-as em seu trabalho.

[1] Neste livro, mais adiante, chamaremos tais pessoas de "envolvidos".

- Pessoas, negócios e governos dependem, cada vez mais, de software para a tomada de decisões estratégicas e táticas, assim como para controle e para operações cotidianas. Se o software falhar, as pessoas e as principais empresas poderão ter desde pequenos inconvenientes até falhas catastróficas. *Depreende-se, portanto, que um software deve apresentar qualidade elevada.*

Entenda o problema antes de elaborar uma solução.

- À medida que o valor de uma aplicação específica aumenta, a probabilidade é de que sua base de usuários e longevidade também cresçam. À medida que sua base de usuários e seu tempo em uso forem aumentando, a demanda por adaptação e aperfeiçoamento também vai aumentar. *Depreende-se, portanto, que um software deve ser passível de manutenção.*

Qualidade e facilidade de manutenção são resultantes de um projeto bem feito.

Essas simples constatações nos conduzem a uma só conclusão: *software, em todas as suas formas e em todos os seus campos de aplicação, deve passar pelos processos de engenharia.* E isso nos leva ao tema principal deste livro – *engenharia de software*.

2.1 Definição da disciplina

O IEEE [IEE93a] elaborou a seguinte definição para engenharia de software:

> Engenharia de software: (1) A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software. (2) O estudo de abordagens como definido em (1).

Como definimos engenharia de software?

Entretanto, uma abordagem "sistemática, disciplinada e quantificável" aplicada por uma equipe de desenvolvimento de software pode ser pesada para outra. Precisamos de disciplina, mas também precisamos de adaptabilidade e agilidade.

A engenharia de software é uma tecnologia em camadas. Como ilustra a Figura 2.1, qualquer abordagem de engenharia (inclusive engenharia de software) deve estar fundamentada em um comprometimento organizacional com a qualidade. A gestão da qualidade total Seis Sigma e filosofias similares[2] promovem uma cultura de aperfeiçoamento contínuo de processos, e é essa cultura que, no final das contas, leva ao desenvolvimento de abordagens cada vez mais eficazes na engenharia de software. A pedra fundamental que sustenta a engenharia de software é o foco na qualidade.

A base da engenharia de software é a camada de *processos*. O processo de engenharia de software é a liga que mantém as camadas de tecnologia coesas e possibilita o desenvolvimento de software de forma racional e dentro do prazo. O processo define uma metodologia que deve ser estabelecida para a entrega efetiva de tecnologia de engenharia de software. O processo de software constitui a base para o controle do gerenciamento de projetos de software e estabelece o contexto no qual são aplicados métodos técnicos, são produzidos

A engenharia de software engloba um processo, métodos de gerenciamento e desenvolvimento de software, bem como ferramentas.

[2] A gestão da qualidade e as metodologias relacionadas são discutidas ao longo da Parte III deste livro.

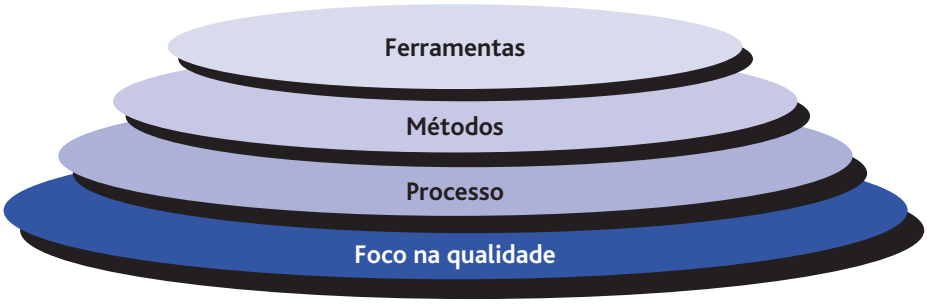


FIGURA 2.1 Camadas da engenharia de software.

artefatos (modelos, documentos, dados, relatórios, formulários etc.), são estabelecidos marcos, a qualidade é garantida e mudanças são geridas de forma apropriada.

Os *métodos* da engenharia de software fornecem as informações técnicas para desenvolver software. Os métodos envolvem uma ampla variedade de tarefas, que incluem: comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte. Os métodos da engenharia de software se baseiam em um conjunto de princípios básicos que governam cada área da tecnologia e incluem atividades de modelagem e outras técnicas descritivas.

CrossTalk é um jornal que divulga informações práticas a respeito de processo, métodos e ferramentas. Pode ser encontrado no endereço: **www.stsc.hill.af.mil**.

As *ferramentas* da engenharia de software fornecem suporte automatizado ou semiautomatizado para o processo e para os métodos. Quando as ferramentas são integradas, de modo que as informações criadas por uma ferramenta possam ser utilizadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de software, denominado *engenharia de software com o auxílio do computador*.

2.2 O processo de software

Quais são os elementos de um processo de software?

Processo é um conjunto de atividades, ações e tarefas realizadas na criação de algum artefato. Uma *atividade* se esforça para atingir um objetivo amplo (por exemplo, comunicar-se com os envolvidos) e é utilizada independentemente do campo de aplicação, do tamanho do projeto, da complexidade dos esforços ou do grau de rigor com que a engenharia de software será aplicada. Uma *ação* (por exemplo, projeto de arquitetura) envolve um conjunto de tarefas que resultam em um artefato de software fundamental (por exemplo, um modelo arquitetural). Uma *tarefa* se concentra em um objetivo pequeno, porém bem-definido (por exemplo, realizar um teste de unidades), e produz um resultado tangível.

"Um processo define quem está fazendo o quê, quando e como para atingir determinado objetivo."

Ivar Jacobson, Grady Booch e James Rumbaugh

No contexto da engenharia de software, um processo *não* é uma prescrição rígida de como desenvolver um software. Ao contrário, é uma abordagem adaptável que possibilita às pessoas (a equipe de software) realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas. A intenção é a de sempre entregar software dentro do prazo e com qualidade suficiente para satisfazer àqueles que patrocinaram sua criação e àqueles que vão utilizá-lo.

2.2.1 A metodologia do processo

Quais são as cinco atividades genéricas de uma metodologia de processo?

Uma *metodologia* (*framework*) *de processo* estabelece o alicerce para um processo de engenharia de software completo por meio da identificação de um pequeno número de *atividades metodológicas* aplicáveis a todos os projetos de software, independentemente de tamanho ou complexidade. Além disso, a metodologia de processo engloba um conjunto de *atividades de apoio* (*umbrella activities*) aplicáveis a todo o processo de software. Uma metodologia de processo genérica para engenharia de software compreende cinco atividades:

Comunicação. Antes que qualquer trabalho técnico possa começar, é de importância fundamental se comunicar e colaborar com o cliente (e outros envolvidos).[3] A intenção é entender os objetivos dos envolvidos para o projeto e reunir requisitos que ajudem a definir os recursos e as funções do software.

"Einstein afirmou que deve haver uma explicação simplificada da natureza, pois Deus não é caprichoso ou arbitrário. Tal fé não conforta o engenheiro de software. Grande parte da complexidade com a qual terá de lidar é arbitrária."

Fred Brooks

Planejamento. Qualquer jornada complicada pode ser simplificada com auxílio de um mapa. Um projeto de software é uma jornada complicada, e a atividade de planejamento cria um "mapa" que ajuda a guiar a equipe na sua jornada. O mapa – denominado plano de projeto de software – define o trabalho de engenharia de software, descrevendo as tarefas técnicas a serem conduzidas, os riscos prováveis, os recursos que serão necessários, os produtos resultantes a ser produzidos e um cronograma de trabalho.

Modelagem. Independentemente de ser um paisagista, um construtor de pontes, um engenheiro aeronáutico, um carpinteiro ou um arquiteto, trabalha-se com modelos todos os dias. Cria-se um "esboço" para que se possa ter uma ideia do todo – qual será o seu aspecto em termos de arquitetura, como as partes constituintes se encaixarão e várias outras características. Se necessário, refina-se o esboço com mais detalhes, numa tentativa de compreender melhor o problema e como resolvê-lo. Um engenheiro de software faz a mesma coisa, criando modelos para entender melhor as necessidades do software e o projeto que vai atender a essas necessidades.

Construção. O que se projeta deve ser construído. Essa atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros na codificação.

Entrega. O software (como uma entidade completa ou como um incremento parcialmente concluído) é entregue ao cliente, que avalia o produto entregue e fornece feedback, baseado na avaliação.

Essas cinco atividades metodológicas genéricas podem ser utilizadas para o desenvolvimento de programas pequenos e simples, para a criação de aplicações para a Internet e para a engenharia de grandes e complexos sistemas baseados em computador. Os detalhes do processo de software serão bem diferentes em cada caso, mas as atividades metodológicas permanecerão as mesmas.

[3] *Envolvido* é qualquer pessoa que tenha interesse no êxito de um projeto – executivos, usuários, engenheiros de software, pessoal de suporte etc. Rob Thomsett ironiza: "Envolvido (*stakeholder*) é uma pessoa que segura (*hold*) uma estaca (*stake*) grande e pontiaguda... Se você não cuidar de seus envolvidos, sabe bem onde essa estaca vai parar".

Para muitos projetos de software, as atividades metodológicas são aplicadas iterativamente conforme o projeto se desenvolve. Ou seja, comunicação, planejamento, modelagem, construção e entrega são aplicados repetidamente, sejam quantas forem as iterações do projeto. Cada iteração produzirá um *incremento de software* que disponibilizará uma parte dos recursos e das funcionalidades do software. A cada incremento, o software se torna cada vez mais completo.

2.2.2 Atividades de apoio

As atividades metodológicas do processo de engenharia de software são complementadas por diversas *atividades de apoio*. De modo geral, as atividades de apoio são aplicadas por todo um projeto de software e ajudam uma equipe de software a gerenciar e a controlar o andamento, a qualidade, as alterações e os riscos. As atividades de apoio típicas são:

Atividades de apoio ocorrem ao longo do processo de software e se concentram, principalmente, em gerenciamento, acompanhamento e controle do projeto.

Controle e acompanhamento do projeto – possibilita que a equipe avalie o progresso em relação ao plano do projeto e tome as medidas necessárias para cumprir o cronograma.

Administração de riscos – avalia riscos que possam afetar o resultado ou a qualidade do produto/projeto.

Garantia da qualidade de software – define e conduz as atividades que garantem a qualidade do software.

Revisões técnicas – avaliam artefatos da engenharia de software, tentando identificar e eliminar erros antes que se propaguem para a atividade seguinte.

Medição – define e coleta medidas (do processo, do projeto e do produto). Auxilia na entrega do software de acordo com os requisitos; pode ser usada com as demais atividades (metodológicas e de apoio).

Gerenciamento da configuração de software – gerencia os efeitos das mudanças ao longo do processo.

Gerenciamento da capacidade de reutilização – define critérios para a reutilização de artefatos (inclusive componentes de software) e estabelece mecanismos para a obtenção de componentes reutilizáveis.

A adaptação do processo de software é essencial para o sucesso de um projeto.

Preparo e produção de artefatos de software – engloba as atividades necessárias para criar artefatos como, por exemplo, modelos, documentos, logs, formulários e listas.

Cada uma dessas atividades de apoio será discutida em detalhes mais adiante.

2.2.3 Adaptação do processo

Anteriormente, declaramos que o processo de engenharia de software não é rígido nem deve ser seguido à risca. Mais do que isso, ele deve ser ágil e adaptável (ao problema, ao projeto, à equipe e à cultura organizacional). Portanto, o processo adotado para determinado projeto pode ser muito diferente daquele adotado para outro. Entre as diferenças, temos:

- Fluxo geral de atividades, ações e tarefas e suas interdependências.
- Até que ponto as ações e tarefas são definidas dentro de cada atividade da metodologia.
- Até que ponto artefatos de software são identificados e exigidos.
- Modo de aplicar as atividades de garantia da qualidade.
- Modo de aplicar as atividades de acompanhamento e controle do projeto.
- Grau geral de detalhamento e rigor da descrição do processo.
- Grau de envolvimento com o projeto (por parte do cliente e de outros envolvidos).
- Nível de autonomia dada à equipe de software.
- Grau de prescrição da organização da equipe.

"Sinto que uma receita consiste em apenas um tema com o qual um cozinheiro inteligente pode brincar, cada vez com uma variação."

Madame Benoit

A Parte I deste livro examina o processo de software com um grau de detalhamento considerável.

2.3 A prática da engenharia de software

A Seção 2.2 apresentou uma introdução a um modelo de processo de software genérico, composto por um conjunto de atividades que estabelecem uma metodologia para a prática da engenharia de software. As atividades genéricas da metodologia – **comunicação, planejamento, modelagem, construção** e **entrega** –, bem como as atividades de apoio, estabelecem um esquema para o trabalho da engenharia de software. Mas como a prática da engenharia de software se encaixa nisso? Nas seções seguintes, você vai adquirir um conhecimento básico dos princípios e conceitos genéricos que se aplicam às atividades de uma metodologia.[4]

Diversas citações instigantes sobre a prática da engenharia de software podem ser encontradas em **www.literate-programming.com**.

2.3.1 A essência da prática

No livro clássico *How to Solve It*, escrito antes de os computadores modernos existirem, George Polya [Pol45] descreveu em linhas gerais a essência da solução de problemas e, consequentemente, a essência da prática da engenharia de software:

1. *Compreender o problema* (comunicação e análise).
2. *Planejar uma solução* (modelagem e projeto de software).
3. *Executar o plano* (geração de código).
4. *Examinar o resultado para ter precisão* (testes e garantia da qualidade).

Pode-se afirmar que a abordagem de Polya é simplesmente uma questão de bom senso. É verdade. Mas é espantoso como o bom senso é tão pouco usado no mundo do software.

No contexto da engenharia de software, essas etapas de bom senso conduzem a uma série de questões essenciais [adaptado de Pol45]:

Compreenda o problema. Algumas vezes é difícil de admitir; porém, a maioria de nós é arrogante quando um problema nos é apresentado. Ouvimos por

O elemento mais importante para se entender um problema é escutar.

[4] Você deve rever seções relevantes contidas neste capítulo à medida que discutirmos os métodos de engenharia de software e as atividades de apoio específicas mais adiante neste livro.

alguns segundos e então pensamos: "*Ah, sim, estou entendendo, vamos começar a resolver este problema*". Infelizmente, compreender nem sempre é assim tão fácil. Vale a pena despender um pouco de tempo respondendo a algumas perguntas simples:

- *Quem tem interesse na solução do problema?* Ou seja, quem são os envolvidos?
- *Quais são as incógnitas?* Que dados, funções e recursos são necessários para resolver apropriadamente o problema?
- *O problema pode ser compartimentalizado?* É possível representá-lo em problemas menores que talvez sejam mais fáceis de ser compreendidos?
- *O problema pode ser representado graficamente?* É possível criar um modelo analítico?

Planeje a solução. Agora você entende o problema (ou assim pensa) e não vê a hora de começar a codificar. Antes de fazer isso, relaxe um pouco e faça um pequeno projeto:

- *Você já viu problemas semelhantes anteriormente?* Existem padrões que são reconhecíveis em uma possível solução? Existe algum software que implemente os dados, as funções e as características necessárias?
- *Algum problema semelhante já foi resolvido?* Em caso positivo, existem elementos da solução que podem ser reutilizados?
- *É possível definir subproblemas?* Em caso positivo, existem soluções aparentes e imediatas para eles?
- *É possível representar uma solução de maneira que conduza a uma implementação efetiva?* É possível criar um modelo de projeto?

"Há um grão de descoberta na solução de qualquer problema."

George Polya

Leve o plano adiante. O projeto elaborado que criamos serve como um mapa para o sistema que se quer construir. Podem surgir desvios inesperados, e é possível que se descubra um caminho ainda melhor à medida que se prossiga; porém, o "planejamento" permitirá que continuemos sem nos perder.

- *A solução é adequada ao plano?* O código-fonte pode ser atribuído ao modelo de projeto?
- *Todas as partes componentes da solução estão provavelmente corretas?* O projeto e o código foram revistos ou, melhor ainda, provas da correção foram aplicadas ao algoritmo?

Examine o resultado. Não se pode ter certeza de que uma solução seja perfeita; porém, pode-se assegurar que um número de testes suficiente tenha sido realizado para revelar o maior número de erros possível.

- *É possível testar cada parte componente da solução?* Foi implementada uma estratégia de testes razoável?
- *A solução produz resultados adequados aos dados, às funções e às características necessários?* O software foi validado em relação a todas as solicitações dos envolvidos?

Não é surpresa que grande parte dessa metodologia consista no bom senso. De fato, é possível afirmar que uma abordagem de bom senso à engenharia de software jamais o levará ao mau caminho.

2.3.2 Princípios gerais

O dicionário define a palavra *princípio* como "uma importante afirmação ou lei básica em um sistema de pensamento". Ao longo deste livro serão discutidos princípios em vários níveis de abstração. Alguns se concentram na engenharia de software como um todo, outros consideram uma atividade de metodologia genérica específica (por exemplo, **comunicação**) e outros ainda destacam as ações de engenharia de software (por exemplo, projeto de arquitetura) ou tarefas técnicas (por exemplo, redigir um cenário de uso). Independentemente do seu nível de enfoque, os princípios ajudam a estabelecer um modo de pensar para a prática segura da engenharia de software. Esta é a razão por que são importantes.

David Hooker [Hoo96] propôs sete princípios que se concentram na prática da engenharia de software como um todo. Eles são reproduzidos nos parágrafos a seguir:[5]

Antes de iniciar um projeto, certifique-se de que o software tem um propósito para a empresa e de que seus usuários reconhecem seu valor.

Primeiro princípio: ***a razão de existir***

Um sistema de software existe por um motivo: *agregar valor para seus usuários*. Todas as decisões devem ser tomadas com esse princípio em mente. Antes de especificar um requisito de um sistema, antes de indicar alguma parte da funcionalidade de um sistema, antes de determinar as plataformas de hardware ou os processos de desenvolvimento, pergunte a si mesmo: "Isso realmente agrega valor real ao sistema?". Se a resposta for "não", não o faça. Todos os demais princípios se apoiam neste primeiro.

"Há certa majestade na simplicidade, que está muito acima de toda a excentricidade do saber."

Alexandre Pope (1688-1744)

Segundo princípio: ***KISS (Keep It Simple, Stupid!, ou seja: não complique!)***

O projeto de software não é um processo casual. Existem muitos fatores a considerar em qualquer trabalho de projeto. *Todo projeto deve ser o mais simples possível, mas não simplista*. Esse princípio contribui para um sistema mais fácil de compreender e manter. Isso não significa que características, até mesmo as internas, devam ser descartadas em nome da simplicidade. De fato, os projetos mais elegantes normalmente são os mais simples. Simples também não significa "gambiarra". Na verdade, muitas vezes são necessárias muitas reflexões e trabalho em várias iterações para simplificar. A contrapartida é um software mais fácil de manter e menos propenso a erros.

Terceiro princípio: ***mantenha a visão***

Uma visão clara é essencial para o sucesso. Sem ela, um projeto se torna ambíguo. Sem uma integridade conceitual, corre-se o risco de transformar o projeto em uma colcha de retalhos de projetos incompatíveis, unidos por parafusos inadequados... Comprometer a visão arquitetural de um sistema de software debilita e até poderá destruir sistemas bem projetados. Ter um ar-

[5] Reproduzido com a permissão do autor [Hoo96]. Hooker define padrões para esses princípios em **http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment**.

quiteto responsável e capaz de manter a visão clara e de reforçar a adequação ajuda a assegurar o êxito de um projeto.

Um software de valor mudará ao longo de sua vida útil. Por essa razão, ele deve ser desenvolvido para fácil manutenção.

Quarto princípio: *o que um produz outros consomem*

Raramente um sistema de software de qualidade industrial é construído e utilizado de forma isolada. De uma maneira ou de outra, alguém mais vai usar, manter, documentar ou, de alguma forma, depender da capacidade de entender seu sistema. Portanto, *sempre especifique, projete e implemente ciente de que mais alguém terá de entender o que você está fazendo*. O público para qualquer produto de desenvolvimento de software é potencialmente grande. Especifique tendo como objetivo os usuários. Projete tendo em mente os implementadores. Codifique se preocupando com aqueles que deverão manter e ampliar o sistema. Alguém terá de depurar o código que você escreveu, e isso o torna um usuário de seu código. Facilitando o trabalho de todas essas pessoas, você agrega maior valor ao sistema.

Quinto princípio: *esteja aberto para o futuro*

Um sistema com tempo de vida mais longo tem mais valor. Nos ambientes computacionais de hoje, em que as especificações mudam de um instante para outro, e as plataformas de hardware se tornam rapidamente obsoletas, a vida de um software, em geral, é medida em meses. Contudo, os verdadeiros sistemas de software com "qualidade industrial" devem durar muito mais. Para serem bem-sucedidos nisso, esses sistemas precisam estar prontos para se adaptar a essas e outras mudanças. Sistemas que obtêm sucesso são aqueles que foram projetados dessa forma desde seu princípio. *Jamais faça projetos limitados*. Sempre pergunte "e se" e prepare-se para todas as respostas possíveis, criando sistemas que resolvam o problema geral, não apenas o específico.[6] Isso muito provavelmente conduziria à reutilização de um sistema inteiro.

Sexto princípio: *planeje com antecedência, visando a reutilização*

A reutilização economiza tempo e esforço.[7] Alcançar um alto grau de reutilização é indiscutivelmente a meta mais difícil de ser atingida ao se desenvolver um sistema de software. A reutilização de código e projetos tem sido proclamada como uma grande vantagem do uso de tecnologias orientadas a objetos. Contudo, o retorno desse investimento não é automático. Aproveitar as possibilidades de reutilização – oferecidas pela programação orientada a objetos (ou convencional) – exige planejamento e capacidade de fazer previsões. Existem várias técnicas para levar a cabo a reutilização em cada um dos níveis do processo de desenvolvimento do sistema... *Planejar com antecedên-*

[6] Esse conselho pode ser perigoso se levado ao extremo. Projetar para o "problema geral" algumas vezes exige comprometer o desempenho e pode tornar ineficientes as soluções específicas.

[7] Embora isso seja verdade para aqueles que reutilizam o software em projetos futuros, a reutilização poderá ser cara para aqueles que precisarem projetar e desenvolver componentes reutilizáveis. Estudos indicam que o projeto e o desenvolvimento de componentes reutilizáveis podem custar de 25 a 200% mais do que o próprio software. Em alguns casos, o diferencial de custo não pode ser justificado.

cia para a reutilização reduz o custo e aumenta o valor tanto dos componentes reutilizáveis quanto dos sistemas aos quais eles serão incorporados.

Sétimo princípio: *pense!*

Este último princípio é, provavelmente, o mais menosprezado. *Pensar bem e de forma clara antes de agir quase sempre produz melhores resultados.* Quando se analisa alguma coisa, provavelmente ela sairá correta. Ganha-se também conhecimento de como fazer correto novamente. Se você realmente analisar algo e mesmo assim o fizer da forma errada, isso se tornará uma valiosa experiência. Um efeito colateral da análise é aprender a reconhecer quando não se sabe algo, e até que ponto poderá buscar o conhecimento. Quando a análise clara faz parte de um sistema, seu valor aflora. Aplicar os seis primeiros princípios exige intensa reflexão, para a qual as recompensas em potencial são enormes.

Se todo engenheiro de software e toda a equipe de software simplesmente seguissem os sete princípios de Hooker, muitas das dificuldades enfrentadas no desenvolvimento de complexos sistemas baseados em computador seriam eliminadas.

2.4 Mitos do desenvolvimento de software

Os mitos criados para o desenvolvimento de software – crenças infundadas sobre o software e sobre o processo utilizado para criá-lo – remontam aos primórdios da computação. Os mitos possuem uma série de atributos que os tornam insidiosos. Por exemplo, eles parecem ser, de fato, afirmações sensatas (algumas vezes contendo elementos de verdade), têm uma sensação intuitiva e frequentemente são promulgados por praticantes experientes "que entendem do riscado".

Atualmente, a maioria dos profissionais versados na engenharia de software reconhece os mitos por aquilo que eles representam – atitudes enganosas que provocaram sérios problemas tanto para gerentes quanto para praticantes da área. Entretanto, antigos hábitos e atitudes são difíceis de ser modificados, e resquícios de mitos de software permanecem.

Mitos de gerenciamento. Gerentes com responsabilidade sobre software, assim como gerentes da maioria das áreas, frequentemente estão sob pressão para manter os orçamentos, evitar deslizes nos cronogramas e elevar a qualidade. Como uma pessoa que está se afogando e se agarra a uma tábua, um gerente de software muitas vezes se agarra à crença em um mito do software para aliviar a pressão (mesmo que temporariamente).

Software Project Managers Network, em www.spmn.com, pode ajudá-lo a refutar esses e outros mitos.

Mito: *Já temos um livro cheio de padrões e procedimentos para desenvolver software. Ele não supriria meu pessoal com tudo que precisam saber?*

Realidade: O livro com padrões pode muito bem existir, mas ele é realmente utilizado? Os praticantes da área estão cientes de que ele existe? Esse livro reflete a prática moderna da engenharia de software? É completo? É adaptável? Está otimi-

zado para melhorar o tempo de entrega, mantendo ainda o foco na qualidade? Em muitos casos, a resposta para todas essas perguntas é "não".

Mito: *Se o cronograma atrasar, poderemos acrescentar mais programadores e ficar em dia (algumas vezes denominado conceito da "horda mongol").*

Realidade: O desenvolvimento de software não é um processo mecânico como o de uma fábrica. Nas palavras de Brooks [Bro95]: "acrescentar pessoas em um projeto de software atrasado só o tornará mais atrasado ainda". A princípio, essa declaração pode parecer absurda. No entanto, o que ocorre é que, quando novas pessoas entram, as que já estavam terão de gastar tempo situando os recém-chegados, reduzindo, consequentemente, o tempo destinado ao desenvolvimento produtivo. Pode-se adicionar pessoas, mas somente de forma planejada e bem coordenada.

Mito: *Se eu decidir terceirizar o projeto de software, posso simplesmente relaxar e deixar a outra empresa realizá-lo.*

Realidade: Se uma organização não souber gerenciar e controlar projetos de software, ela vai, invariavelmente, enfrentar dificuldades ao terceirizá-los.

Mitos dos clientes. O cliente solicitante do software computacional pode ser uma pessoa na mesa ao lado, um grupo técnico do andar de baixo, de um departamento de marketing/vendas ou uma empresa externa que contratou o projeto. Em muitos casos, o cliente acredita em mitos sobre software porque gerentes e profissionais da área pouco fazem para corrigir falsas informações. Mitos conduzem a falsas expectativas (do cliente) e, em última instância, à insatisfação com o desenvolvedor.

Esforce-se ao máximo para compreender o que deve ser feito antes de começar. Você pode não chegar a todos os detalhes, mas, quanto mais souber, menor será o risco.

Mito: *Uma definição geral dos objetivos é suficiente para começar a escrever os programas – podemos preencher os detalhes posteriormente.*

Realidade: Embora nem sempre seja possível uma definição ampla e estável dos requisitos, uma definição de objetivos ambígua é a receita para um desastre. Requisitos não ambíguos (normalmente derivados iterativamente) são obtidos somente pela comunicação contínua e eficaz entre cliente e desenvolvedor.

Mito: *Os requisitos de software mudam continuamente, mas as mudanças podem ser facilmente assimiladas, pois o software é flexível.*

Realidade: É verdade que os requisitos de software mudam, mas o impacto da mudança varia dependendo do momento em que foi introduzida. Quando as mudanças dos requisitos são solicitadas cedo (antes de o projeto ou de a codificação te-

rem começado), o impacto sobre os custos é relativamente pequeno.[8] Entretanto, conforme o tempo passa, ele aumenta rapidamente – recursos foram comprometidos, uma estrutura de projeto foi estabelecida e mudar pode causar uma revolução que exija recursos adicionais e modificações fundamentais no projeto.

Mitos dos profissionais da área. Mitos que ainda sobrevivem entre os profissionais da área têm resistido por mais de 60 anos de cultura da programação. Durante seus primórdios, a programação era vista como uma forma de arte. Hábitos e atitudes antigos são difíceis de perder.

Toda vez que pensar "não temos tempo para engenharia de software", pergunte a si mesmo: "teremos tempo para fazer de novo?".

Mito: *Uma vez que o programa foi feito e colocado em uso, nosso trabalho está terminado.*

Realidade: Uma vez alguém já disse que "o quanto antes se começar a codificar, mais tempo levará para terminar". Levantamentos indicam que entre 60 e 80% de todo o esforço será despendido após a entrega do software ao cliente pela primeira vez.

Mito: *Até que o programa esteja "em execução", não há como avaliar sua qualidade.*

Realidade: Um dos mecanismos de garantia da qualidade de software mais eficientes pode ser aplicado a partir da concepção de um projeto – *a revisão técnica*. Os revisores de software (descritos no Capítulo 20) são "filtros de qualidade", considerados mais eficientes do que os testes feitos para encontrar certas classes de defeitos de software.

Mito: *O único produto passível de entrega é o programa em funcionamento.*

Realidade: Um programa funcionando é somente uma parte de uma configuração de software que inclui muitos elementos. Uma variedade de artefatos (por exemplo, modelos, documentos, planos) constitui uma base para uma engenharia bem-sucedida e, mais importante, uma orientação para suporte de software.

Mito: *A engenharia de software nos fará criar documentação volumosa e desnecessária e, invariavelmente, vai nos retardar.*

Realidade: O objetivo da engenharia de software não é criar documentos. É criar um produto de qualidade. Uma qualidade melhor leva à redução do retrabalho. E retrabalho reduzido resulta em tempos de entrega menores.

Atualmente, muitos profissionais de software reconhecem a falácia dos mitos que acabamos de descrever. Estar ciente das realidades do software é o primeiro passo para buscar soluções práticas na engenharia de software.

[8] Muitos engenheiros de software têm adotado uma abordagem "ágil" que favorece as alterações incrementais, controlando, com isso, seu impacto e seu custo. Os métodos ágeis são discutidos no Capítulo 5.

2.5 Como tudo começa

Todo projeto de software é motivado por alguma necessidade de negócios – a necessidade de corrigir um defeito em uma aplicação existente; a necessidade de adaptar um "sistema legado" a um ambiente de negócios em constante transformação; a necessidade de ampliar as funções e os recursos de uma aplicação existente ou a necessidade de criar um novo produto, serviço ou sistema.

No início de um projeto de software, a necessidade do negócio é, com frequência, expressa informalmente como parte de uma simples conversa. A conversa apresentada no quadro a seguir é típica.

CASASEGURA[9]

Como começa um projeto

Cena: Sala de reuniões da CPI Corporation, empresa (fictícia) que fabrica produtos de consumo para uso doméstico e comercial.

Atores: Mal Golden, gerente sênior, desenvolvimento do produto; Lisa Perez, gerente de marketing; Lee Warren, gerente de engenharia; Joe Camalleri, vice-presidente executivo, desenvolvimento de negócios.

Conversa:

Joe: Lee, ouvi dizer que o seu pessoal está trabalhando em algo. Do que se trata? Um tipo de caixa sem fio de uso amplo e genérico?

Lee: Trata-se de algo bem legal... aproximadamente do tamanho de uma caixa de fósforos, conectável a todo tipo de sensor, como uma câmera digital – ou seja, se conecta a quase tudo. Usa o protocolo sem fio 802.11n. Permite que acessemos saídas de dispositivos sem o emprego de fios. Acreditamos que nos levará a uma geração inteiramente nova de produtos.

Joe: Você concorda, Mal?

Mal: Sim. Na verdade, com as vendas tão baixas neste ano, precisamos de algo novo. Lisa e eu fizemos uma pequena pesquisa de mercado e acreditamos que conseguimos uma linha de produtos que poderá ser ampla.

Joe: Ampla em que sentido?

Mal (evitando comprometimento direto): Conte sobre nossa ideia, Lisa.

Lisa: Trata-se de uma geração completamente nova na linha de "produtos de gerenciamento doméstico". Chamamos esses produtos de *CasaSegura*. Eles usam uma nova interface sem fio e oferecem a pequenos empresários e proprietários de residências um sistema que é controlado por seus PCs, envolvendo segurança doméstica, sistemas de vigilância, controle de eletrodomésticos e dispositivos. Por exemplo, seria possível diminuir a temperatura do aparelho de ar condicionado enquanto você está voltando para casa, esse tipo de coisa.

Lee (reagindo sem pensar): O departamento de engenharia fez um estudo de viabilidade técnica dessa ideia, Joe. É possível fazê-lo com baixo custo de fabricação. A maior parte dos componentes do hardware é encontrada no mercado. O software é um problema, mas não é nada que não possamos resolver.

Joe: Interessante. Mas eu perguntei qual é o ponto principal.

Mal: PCs e tablets estão em mais de 70% dos lares nos EUA. Se pudermos acertar no preço, essa aplicação poderá ser excepcional. Ninguém mais tem nosso dispositivo sem fio... ele é exclusivo! Estaremos dois anos à frente de nossos concorrentes... e as receitas? Algo em torno de 30 a 40 milhões no segundo ano...

Joe (sorrindo): Vamos levar isso adiante. Estou interessado.

Exceto por uma rápida referência, o software mal foi mencionado como parte da conversa. Ainda assim, o software vai decretar o sucesso ou o fracasso da linha de produtos *CasaSegura*. O trabalho de engenharia só terá êxito se o software do *CasaSegura* tiver êxito. O mercado só vai aceitar o produto se o

[9] O projeto *CasaSegura* será usado ao longo deste livro para ilustrar o funcionamento interno de uma equipe de projeto à medida que ela constrói um produto de software. A empresa, o projeto e as pessoas são fictícios, porém as situações e os problemas são reais.

software incorporado atender adequadamente às necessidades (ainda não declaradas) do cliente. Acompanharemos a evolução da engenharia do software *CasaSegura* em vários dos capítulos que estão por vir.

2.6 Resumo

A engenharia de software engloba processos, métodos e ferramentas que possibilitam a construção de sistemas complexos baseados em computador dentro do prazo e com qualidade. O processo de software incorpora cinco atividades estruturais: comunicação, planejamento, modelagem, construção e entrega, e elas se aplicam a todos os projetos de software. A prática da engenharia de software é uma atividade de resolução de problemas que segue um conjunto de princípios básicos.

Inúmeros mitos em relação ao software continuam a levar gerentes e profissionais para o mau caminho, mesmo com o aumento do conhecimento coletivo sobre software e das tecnologias necessárias para construí-los. À medida que for aprendendo mais sobre a engenharia de software, você começará a compreender por que esses mitos devem ser derrubados toda vez que nos depararmos com eles.

Problemas e pontos a ponderar

2.1. A Figura 2.1 coloca as três camadas de engenharia de software acima de uma camada intitulada "foco na qualidade". Isso implica um programa de qualidade organizacional como o de gestão da qualidade total. Pesquise um pouco a respeito e crie um sumário dos princípios básicos de um programa de gestão da qualidade total.

2.2. A engenharia de software é aplicável na construção de WebApps? Em caso positivo, como poderia ser modificada para atender às características únicas das WebApps?

2.3. À medida que o software invade todos os setores, riscos ao público (devido a programas com imperfeições) passam a ser uma preocupação cada vez maior. Crie um cenário o mais catastrófico possível, porém realista, em que a falha de um programa de computador poderia causar um grande dano em termos econômicos ou humanos.

2.4. Descreva uma metodologia de processo com suas próprias palavras. Ao afirmarmos que atividades de modelagem se aplicam a todos os projetos, isso significa que as mesmas tarefas são aplicadas a todos os projetos, independentemente de seu tamanho e complexidade? Explique.

2.5. As atividades de apoio ocorrem ao longo do processo de software. Você acredita que elas são aplicadas de forma homogênea ao longo do processo ou algumas delas são concentradas em uma ou mais atividades da metodologia?

2.6. Acrescente mais dois mitos à lista apresentada na Seção 2.4. Declare também a realidade que acompanha o mito.

Leituras e fontes de informação complementares

O estado atual da engenharia de software e do processo de software pode ser mais bem determinado a partir de publicações como *IEEE Software*, *IEEE Computer*, *CrossTalk* e *IEEE*

Transactions on Software Engineering. Periódicos do setor, como *Application Development Trends* e *Cutter IT Journal*, normalmente contêm artigos sobre tópicos da engenharia de software. A disciplina é "sintetizada" todos os anos no *Proceeding of the International Conference on Software Engineering*, patrocinado pelo *IEEE* e *ACM*, e é discutida de forma aprofundada em periódicos como *ACM Transactions on Software Engineering and Methodology*, *ACM Software Engineering Notes* e *Annals of Software Engineering*. Dezenas de milhares de páginas Web são dedicadas à engenharia de software e ao processo de software.

Nos últimos anos, foram publicados vários livros sobre o processo de desenvolvimento de software e sobre a engenharia de software. Alguns fornecem uma visão geral de todo o processo, ao passo que outros se aprofundam em tópicos específicos importantes, em detrimento dos demais. Entre as ofertas mais populares (além deste livro, é claro!), temos:

SWEBOK: Guide to the Software Engineering Body of Knowledge, [10] IEEE, 2013, consulte: http:// www.computer.org/portal/web/swebok

Andersson, E., *et al*., *Software Engineering for Internet Applications,* MIT Press, 2006.

Braude, E. e M. Bernstein, *Software Engineering: Modern Approaches,* 2ª ed., Wiley, 2010.

Christensen, M. e R. Thayer, *A Project Manager's Guide to Software Engineering Best Practices,* IEEE-CS Press (Wiley), 2002.

Glass, R., *Fact and Fallacies of Software Engineering,* Addison-Wesley, 2002.

Hussain, S., *Software Engineering,* I K International Publishing House, 2013.

Jacobson, I., *Object-Oriented Software Engineering: A Use Case Driven Approach,* 2ª ed., Addison-Wesley, 2008.

Jalote, P., *An Integrated Approach to Software Engineering,* 3ª ed., Springer, 2010.

Pfleeger, S., *Software Engineering: Theory and Practice,* 4ª ed., Prentice Hall, 2009.

Schach, S., *Object-Oriented and Classical Software Engineering,* 8ª ed., McGraw-Hill, 2010.

Sommerville, I., *Software Engineering,* 9ª ed., Addison-Wesley, 2010.

Stober, T. e U. Hansmann, *Agile Software Development: Best Practices for Large Development Projects,* Springer, 2009.

Tsui, F. e O. Karam, *Essentials of Software Engineering,* 2ª ed., Jones & Bartlett Publishers, 2009.

Nygard (*Release It!: Design and Deploy Production-Ready Software,* Pragmatic Bookshelf, 2007), Richardson e Gwaltney (*Ship it! A Practical Guide to Successful Software Projects,* Pragmatic Bookshelf, 2005) e Humble e Farley (*Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010) apresentam uma ampla coleção de diretrizes úteis, aplicáveis à atividade de entrega.

Ao longo das últimas décadas foram publicados diversos padrões de engenharia de software pelo IEEE, pela ISO e suas organizações de padronização. Moore (*The Road Map to Software Engineering: A Standards-Based Guide,* IEEE Computer Society Press [Wiley], 2006) disponibiliza uma pesquisa útil sobre padrões relevantes e como aplicá-los em projetos reais.

Uma ampla variedade de fontes de informação sobre engenharia de software e o processo de software está disponível na Internet. Uma lista atualizada de referências relevantes (em inglês) para o processo para o software pode ser encontrada no site: **www.mhhe.com/pressman.**

[10] Disponível gratuitamente em **<http://www.computer.org/portal/web/swebok/htmlformat>.**