

**CONCEITOS-  
-CHAVE**

atividades de	
apoio .....	40
campos de	
aplicação .....	34
características	
de software .....	32
engenharia de	
software .....	38
metodologia .....	40
mitos de software ..	45
prática .....	45
princípios .....	44
processo de	
software .....	40
software legado ...	36
WebApps .....	37

**E**le tinha a aparência clássica de um executivo sênior de uma grande empresa de software — cerca de 40 anos de idade, as têmporas levemente grisalhas, elegante e atlético, olhos penetrantes enquanto falava. Mas o que ele disse me deixou em choque: “O software está morto”.

Fiquei surpreso e então sorri. “Você está brincando, não é mesmo? O mundo é comandado pelo software e sua empresa tem lucrado imensamente com isso... Ele não está morto! Está vivo e crescendo.”

Balançando a cabeça enfática e negativamente, acrescentou: “Não, ele está morto... Pelo menos da forma que, um dia, o conhecemos”.

Assenti e disse: “Continue”.

Ele falava batendo na mesa para enfatizar: “A visão da velha escola de software — você o compra, é seu proprietário e é o responsável pelo seu gerenciamento — está chegando ao fim. Atualmente, com a Web 2.0 e a computação pervasiva, que vem surgindo com força, estamos por ver uma geração completamente diferente de software. Ele será distribuído via Internet e parecerá estar residente nos dispositivos do computador de cada usuário... Porém, estará residente em um servidor bem distante”.

**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*) como na virtual, abrangendo praticamente qualquer mídia eletrônica. 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 criam e dão suporte a ele e, praticamente, todos do mundo industrializado o utilizam, direta ou indiretamente.

**Por que ele é importante?** Software é importante porque afeta a quase todos os aspectos de nossas vidas e tornou-se pervasivo (incorporado) no comércio, na cultura e em nossas atividades cotidianas. A engenharia de software é importante

porque ela nos capacita para o desenvolvimento de sistemas complexos dentro do prazo e com alta qualidade.

**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, é um conjunto de programas, conteúdo (dados) e outros artefatos que são software. 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 feito corretamente?** Leia o restante deste livro, selecione as ideias que são aplicáveis ao software que você desenvolver e use-as em seu trabalho.

Eu tive de concordar: "Assim, sua vida será muito mais simples. Vocês, meus amigos, não terão de se preocupar com cinco versões diferentes do mesmo aplicativo em uso por dezenas de milhares de usuários espalhados".

Ele sorriu e acrescentou: "Absolutamente. Apenas a versão mais atual residindo em nossos servidores. Quando fizermos uma modificação ou correção, forneceremos funcionalidade e conteúdo atualizados para todos os usuários. Todos terão isso instantaneamente!".

Fiz uma careta: "Mas, da mesma forma, se houver um erro, todos o terão instantaneamente".

Ele riu: "É verdade, por isso estamos redobrando nossos esforços para aplicarmos a engenharia de software de uma forma ainda melhor. O problema é que temos de fazer isso 'rapidamente', pois o mercado tem se acelerado, em todas as áreas de aplicação".

Recostei-me e, com minhas mãos por trás da cabeça, disse: "Você sabe o que falam por aí: você pode ter algo rápido, você pode ter algo correto ou você pode ter algo barato. Escolha dois!".

"Eu fico com rápido e correto", disse, levantando-se.

Também me levantei, concluindo: "Então, nós realmente precisamos de engenharia de software".

"Sei disso", afirmou, enquanto se afastava. "O problema é: temos de convencer ainda outra geração de 'techies'\* de que isso é verdadeiro!".

O software está *realmente* morto? Se estivesse, você não estaria lendo este livro!

Software de computadores continua a ser a tecnologia única mais importante no cenário mundial. E é também um ótimo exemplo da lei das consequências não intencionais. Há cinquenta anos, ninguém poderia prever que o software iria se tornar uma tecnologia indispensável para negócios, ciência e engenharia; que software iria viabilizar 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, indústria gráfica); que software se tornaria a força motriz por trás da revolução do computador pessoal; que produtos de pacotes de software seriam comprados pelos consumidores em lojas de bairro; que software evoluiria lentamente de produto para serviço, na medida que empresas de software "sob encomenda" oferecessem funcionalidade imediata (*just-in-time*), via um navegador Web; que uma companhia de software iria se tornar a maior e mais influente do que quase todas as companhias da era industrial; que uma vasta rede comandada por software, denominada Internet, iria evoluir e modificar tudo: de pesquisa em bibliotecas a compras feitas pelos consumidores, incluindo discurso político, hábitos de namoros de jovens e de adultos não tão jovens.

Ninguém poderia prever que o software seria incorporado em 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.

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.

Conforme aumenta a importância do software, a comunidade da área tenta desenvolver 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 ainda outras são de bases amplas (por exemplo, sistemas operacionais como o

"Ideias e descobertas tecnológicas são as forças propulsoras do crescimento econômico."

*The Wall Street Journal*

\* N. de R.T.: fanáticos por tecnologia.



Linux). Entretanto, nós ainda temos de desenvolver uma tecnologia de software que faça tudo isso, sendo que 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

### PONTO-CHAVE

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

"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

Hoje, o software assume 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. Independentemente de residir em um celular ou operar dentro de um mainframe, 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 num 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.

O papel desempenhado pelo software tem passado por grandes mudanças ao longo dos últimos cinquenta anos. Aperfeiçoamentos significativos no desempenho do hardware, mudanças profundas nas arquiteturas computacionais, vasto aumento na capacidade de memória e armazenamento, e uma ampla variedade de exóticas 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 sistemas robustos.

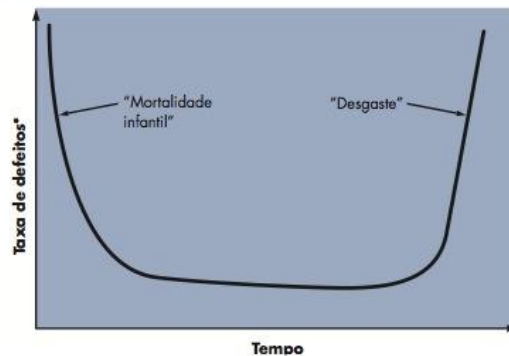
Atualmente, uma enorme indústria de software tornou-se fator dominante nas economias do mundo industrializado. Equipes de especialistas em software, cada qual concentrando-se numa parte da tecnologia necessária para distribuir uma aplicação complexa, substituíram o programador solitário de antigamente.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

- Por que concluir 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 mantendo programas existentes?
- Por que continuamos a ter dificuldade em medir o progresso enquanto o software está sendo desenvolvido e mantido?

<sup>1</sup> Em um excelente livro de ensaio sobre o setor de software, Tom DeMarco (DeM95) contesta. Ele afirma: "Em vez de perguntar por que software custa tanto, precisamos começar perguntando: 'O que fizemos para tornar possível que o software atual custe tão pouco?' A resposta a essa pergunta nos ajudará a continuarmos com o extraordinário nível de realização que sempre tem distinguido a indústria de software".

FIGURA 1.1

Curva de defeitos para hardware



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 Definindo software

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

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

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 como 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, de forma considerável, a compreensão do que é software. Para conseguir isso, é importante examinar as características do software que o tornam diverso de outras coisas que os seres humanos constroem. Software é mais um elemento de sistema lógico do que físico. Dessa forma, tem características que são consideravelmente diferentes daquelas do hardware:

1. *Software é desenvolvido ou passa por um processo de engenharia; ele não é fabricado no sentido clássico.*

Embora existam algumas similaridades entre o desenvolvimento de software e a fabricação de hardware, as duas atividades são fundamentalmente diferentes. Em ambas, alta qualidade é obtida por meio de bom projeto, entretanto, a fase de fabricação de hardware pode gerar problemas de qualidade inexistentes (ou facilmente corrigíveis) para software. Ambas as atividades são dependentes de pessoas, mas a relação entre pessoas envolvidas e trabalho realizado é completamente diferente (ver Capítulo 24). Ambas requerem a construção de um “produto”, entretanto, as abordagens são diferentes. Os custos de software concentram-se no processo de engenharia. Isso significa que projetos de software não podem ser geridos como se fossem projetos de fabricação.

\* N. de R.T.: os defeitos do software nem sempre se manifestam como falha, geralmente devido a tratamentos dos erros decorrentes destes defeitos pelo software. Esses conceitos serão precisamente 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.

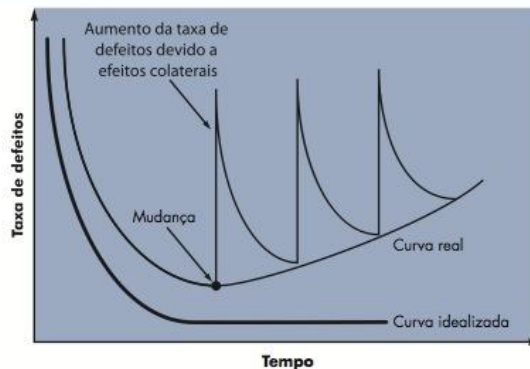
**Como devemos definir software?**

#### PONTO-CHAVE

Software é um processo de engenharia, não é fabricação.

FIGURA 1.2

Curvas de defeitos para software

**PONTO-CHAVE**

Software não se desgasta, mas ele se deteriora.

**AVISO**

Caso queira reduzir a deterioração do software, terá de fazer um projeto melhor de software (Capítulos 8 a 13).

**PONTO-CHAVE**

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.

## 2. 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 males ambientais. Resumindo, o hardware começa a desgastar-se.

Software não é suscetível aos males ambientais 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 sim se deteriora!

Essa aparente contradição pode ser elucidada pela curva real apresentada na Figura 1.2. Durante sua vida<sup>2</sup>, o software passará por alterações. À medida que estas 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 de 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 em complexidade consideravelmente maior do que a de manutenção de hardware.

<sup>2</sup> De fato, desde o momento em que o desenvolvimento começa, e muito antes da primeira versão ser entregue, podem ser solicitadas mudanças por uma variedade de diferentes interessados.



"Ideias são a matéria-prima para construção de ideias."

Jason Zebheazy

3. Embora a indústria caminhe para a construção com base em componentes, a maioria dos softwares continua a ser construída de forma personalizada (sob encomenda).

À medida que a disciplina da engenharia evolui, uma coleção de componentes de projeto padronizados é criada. Parafusos padronizados e circuitos integrados de linha são apenas dois dos milhares de componentes padronizados utilizados por engenheiros mecânicos e elétricos ao projetarem novos sistemas. Os componentes reutilizáveis foram criados para que o engenheiro possa se concentrar nos elementos realmente inovadores de um projeto, isto é, nas partes do projeto que representam algo novo. No mundo do hardware, a reutilização de componentes é uma parte natural do processo de engenharia. No mundo do software, é algo que, em larga escala, apenas começou a ser alcançado.

Um componente de software deve ser projetado e implementado de modo que possa ser reutilizado em muitos programas diferentes. Os modernos componentes reutilizáveis encapsulam tanto dados quanto o processamento aplicado a eles, possibilitando criar novas aplicações a partir de partes reutilizáveis.<sup>3</sup> Por exemplo, as atuais interfaces interativas com o usuário são construídas com componentes reutilizáveis que possibilitam criar janelas gráficas, menus "pull-down" (suspensos e retráteis) e uma ampla variedade de mecanismos de interação. Estruturas de dados e detalhes de processamento necessários para a construção da interface ficam em uma biblioteca de componentes reutilizáveis para a construção de interfaces.

### 1.1.2 Campos de aplicação de software

Hoje em dia, 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.<sup>4</sup> Outras aplicações de sistema (por exemplo, componentes de sistema operacional, drivers, software de rede, processadores de telecomunicações) processam dados amplamente indeterminados. Em ambos os casos, a área de software de sistemas é caracterizada por "pesada" interação com o hardware do computador; uso intenso por múltiplos usuários; operação concorrente que requer escala da ordem, compartilhamento de recursos e gestão de processo sofisticada; estruturas de dados complexas e múltiplas interfaces externas.

**Software de aplicação** — programas sob medida 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. Além das aplicações convencionais de processamento de dados, o software de aplicação é usado para controlar funções de negócio em tempo real (por exemplo, processamento de transações em pontos de venda, controle de processos de fabricação em tempo real).

**Software científico/de engenharia** — tem sido caracterizado por algoritmos "number crunching" (para "processamento numérico pesado"). As aplicações vão da astronomia à vulcanologia, da análise de tensões na indústria automotiva à dinâmica orbital de ônibus espaciais, e da biologia molecular à fabricação automatizada. Entretanto, aplicações modernas dentro da área de engenharia/científica estão se afastando dos algoritmos numéricos convencionais. Projeto com o auxílio de computador, simulação de sistemas e outras aplicações interativas começaram a ter características de sistemas em tempo real e até mesmo de software de sistemas.

#### WebRef

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

<sup>3</sup> O desenvolvimento com base em componentes é discutido no Capítulo 10.

<sup>4</sup> 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 ordem e *timing* de entradas, processamento e saídas não puderem ser previstos antecipadamente.

**Software embutido** — residente num produto ou sistema e utilizado para implementar e controlar características e funções para o usuário final 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 auto-móveis, tal como controle do nível de combustível, painéis de controle e sistemas de freios).

**Software para linha de produtos** — projetado para prover capacidade específica de utilização por muitos clientes diferentes. Pode focalizar um mercado limitado e particularizado (por exemplo, produtos para controle de estoques) ou direcionar-se para mercados de consumo de massa (por exemplo, processamento de texto, planilhas eletrônicas, computação gráfica, multimídia, entretenimento, gerenciamento de bancos de dados e aplicações financeiras pessoais e comerciais).

**Aplicações para a Web** — chamadas de “WebApps”, essa categoria de software centralizada em redes abarca uma vasta gama de aplicações. Em sua forma mais simples, as WebApps podem ser pouco mais que um conjunto de arquivos de hipertexto interconectados, apresentando informações por meio de texto e informações gráficas limitadas. Entretanto, com o aparecimento da Web 2.0, elas têm evoluído e se transformado em sofisticados ambientes computacionais que não apenas fornecem recursos especializados, funções computacionais e conteúdo para o usuário final, como também estão integradas a bancos de dados corporativos e aplicações comerciais.

**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.

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 para um jovem engenheiro de software trabalhar num programa mais velho 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 essa geração facilite o trabalho de futuros engenheiros de software. Ainda assim, novos desafios (Capítulo 31) têm surgido no horizonte:

**Computação mundial aberta** — o rápido crescimento de redes sem fio pode, em breve, conduzir a uma verdadeira computação distribuída e pervasiva (ampliada, compartilhada e incorporada nos ambientes domésticos e comerciais). O desafio para os engenheiros de software será o de desenvolver sistemas e software aplicativo que permitam que dispositivos móveis, computadores pessoais e sistemas corporativos se comuniquem através de extensas redes.

**Netsourcing (recursos via Internet)** — a Internet está se tornando, rapidamente, tanto um mecanismo computacional, como um provedor de conteúdo. O desafio para os engenheiros de software consiste em arquitetar aplicações simples (isto é, planejamento financeiro pessoal) e sofisticadas que forneçam benefícios aos mercados mundiais de usuários finais visados.

**Software aberto** — uma tendência crescente que resulta na distribuição de código-fonte para aplicações de sistemas (por exemplo, sistemas operacionais, bancos de dados e ambientes de desenvolvimento), de forma que muitas pessoas possam contribuir para seu desenvolvimento. O desafio para os engenheiros de software consiste em construir um código-fonte autodescritivo, porém, mais importante ainda, será desenvolver técnicas que permitam que tanto clientes quanto desenvolvedores saibam quais alterações foram feitas e como se manifestam dentro do software.

“Não existe  
computador que  
tenha bom senso.”  
Marvin Minsky



“Você não pode sempre prever, mas pode sempre se preparar.”

Anônimo

Cada um desses desafios obedecerá, sem dúvida, à lei das consequências não intencionais, produzindo efeitos (para executivos, engenheiros de software e usuários finais) que, hoje, não podem ser previstos. Entretanto, os engenheiros de software podem se preparar, iniciando um processo que seja ágil e suficientemente adaptável para assimilar as profundas mudanças na tecnologia e nas regras comerciais, que certamente virão na próxima década.

### 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 a 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.


Infelizmente, algumas vezes, há uma característica adicional que pode estar presente em um software legado: *baixa qualidade*.<sup>5</sup> Os sistemas legados, algumas vezes, têm projetos não expansíveis, código intrincado, documentação pobre ou inexistente, casos de testes e resultados que nunca foram arquivados, um histórico de modificações mal administrado — a lista pode ser bem longa. Ainda assim, esses sistemas dão suporte a “funções vitais de negócio e são indispensáveis para ele”. O que fazer então?

A única resposta razoável 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 roda de forma confiável, ele não está “quebrado” e não precisa ser “consertado”. Entretanto, com o passar do tempo, esses sistemas, frequentemente, evoluem por uma ou mais das seguintes razões:

- 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 interoperável com outros bancos de dados ou com sistemas mais modernos.
- O software deve ser rearquitetado para torná-lo viável dentro de um ambiente de rede.

Quando essas modalidades de evolução ocorrerem, um sistema legado deve passar por re-engenharia (Capítulo 29) para que permaneça viável no futuro. O objetivo da engenharia de software moderna é o de “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 interoperar e cooperar um com o outro” [Day99].

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

 Que tipos de mudanças são feitas em sistemas legados?



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

<sup>5</sup> Nesse caso, a qualidade é julgada pensando-se em termos de 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 ÚNICA DAS WEBAPPS

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

Louis Monier

Nos primórdios da World Wide Web (por volta de 1990 a 1995), os sites eram formados por nada mais que um conjunto de arquivos de hipertexto ligados que apresentavam informações usando texto e gráficos limitados. Com o tempo, o aumento da HTML, via ferramentas de desenvolvimento (por exemplo, XML, Java), tornou possível aos engenheiros da Internet oferecerem capacidade computacional juntamente com as informações. Nasceram, então, os *sistemas e aplicações baseados na Web*<sup>6</sup> (refiro-me a eles coletivamente como aplicações *WebApps*). Atualmente, as *WebApps* evoluíram para sofisticadas ferramentas computacionais que não apenas oferecem funções especializadas (*stand-alone functions*) ao usuário final, como também foram integradas aos bancos de dados corporativos e às aplicações de negócio.

Conforme observado na Seção 1.1.2, *WebApps* são apenas uma dentre uma série de diferentes categorias de software. Ainda assim, pode-se afirmar que elas são diferentes. Powell [Pow98] sugere que sistemas e aplicações baseados na Web "envolvem uma mistura de publicação impressa e desenvolvimento de software, de marketing e computação, de comunicações internas e relações externas e de arte e tecnologia". Os seguintes atributos são encontrados na grande maioria das *WebApps*:

 Que características diferenciam as *WebApps* de outros softwares?

**Uso intensivo de redes.** Uma *WebApp* reside em uma rede e deve atender às necessidades de uma comunidade diversificada de clientes. A rede possibilita acesso e comunicação mundiais (isto é, a Internet) ou acesso e comunicação mais limitados (por exemplo, uma Intranet corporativa).

**Simultaneidade.** Um grande número de usuários pode acessar a *WebApp* ao mesmo tempo. Em muitos casos, os padrões de utilização entre os usuários finais variam amplamente.

**Carga não previsível.** O número de usuários da *WebApp* pode variar, em ordem de grandeza, de um dia para outro. Uma centena de usuários pode conectar-se na segunda-feira e 10.000 na quinta.

**Desempenho.** Se um usuário de uma *WebApp* tiver de esperar muito (para acesso, processamento no servidor, formatação e exibição no cliente), talvez ele procure outra opção.

**Disponibilidade.** Embora a expectativa de 100% de disponibilidade não seja razoável, usuários de *WebApps* populares normalmente exigem acesso 24 horas por dia, 7 dias por semana, 365 dias por ano. Usuários na Austrália ou Ásia podem requerer acesso quando aplicações de software domésticas tradicionais na América do Norte estejam off-line para manutenção.

**Orientadas a dados.** A função principal de muitas *WebApps* é usar hipermídias para apresentar texto, gráficos, áudio e vídeo para o usuário final. Além disso, as *WebApps* são comumente utilizadas para acessar informações em bancos de dados que não são parte integrante do ambiente baseado na Web (por exemplo, comércio eletrônico e/ou aplicações financeiras).

**Sensibilidade no conteúdo.** A qualidade e a natureza estética do conteúdo são fatores importantes que determinam a qualidade de uma *WebApp*.

**Evolução contínua.** Diferentemente de softwares de aplicação convencionais que evoluem ao longo de uma série de versões planejadas e cronologicamente espaçadas, as *WebApps* evoluem continuamente. Não é incomum algumas delas (especificamente seu conteúdo) serem atualizadas segundo uma escala minuto a minuto ou seu conteúdo ser computado independentemente para cada solicitação.

<sup>6</sup> No contexto deste livro, o termo *aplicação Web* (*WebApp*) engloba tudo, de uma simples página Web que possa ajudar um consumidor 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.

**Imediatismo.** Embora *imediatismo* — a imperativa necessidade de colocar rapidamente um software no mercado — seja uma característica de diversos campos de aplicação, as WebApps normalmente apresentam um tempo de colocação no mercado que pode consistir de poucos dias ou semanas.<sup>7</sup>

**Segurança.** Pelo fato de estarem disponíveis via acesso à Internet, torna-se difícil, se não impossível, limitar o número dos usuários finais que podem acessar as WebApps. A fim de proteger conteúdos sensíveis e oferecer modos seguros de transmissão de dados, fortes medidas de segurança devem ser implementadas ao longo da infraestrutura que suporta uma WebApp e dentro da própria aplicação.

**Estética.** Parte inegável do apelo de uma WebApp consiste na sua aparência e na impressão que desperta. Quando uma aplicação for desenvolvida para o mercado ou para vender produtos ou ideias, a estética pode ser tão importante para o sucesso quanto o projeto técnico.

Pode-se argumentar que outras categorias de aplicação discutidas na Seção 1.1.2 podem exibir alguns dos atributos citados. Entretanto, as WebApps quase sempre apresentam todos esses atributos.

### 1.3 ENGENHARIA DE SOFTWARE

Para desenvolver software que esteja preparado para enfrentar os desafios do século vinte e um, devemos perceber uns poucos fatos reais:

#### PONTO-CHAVE

Entenda o problema antes de elaborar uma solução.

#### PONTO-CHAVE

Projetar é uma atividade fundamental na engenharia de software.

#### PONTO-CHAVE

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

- Software tornou-se profundamente incorporado em praticamente todos os aspectos de nossas vidas e, conseqüentemente, o número de pessoas interessadas nos recursos e nas funções oferecidas por uma determinada aplicação<sup>8</sup> tem crescido significativamente. Quando uma aplicação ou um sistema embutido estão para ser desenvolvidos, muitas vozes devem ser ouvidas. E, algumas vezes, parece que cada uma delas possui uma ideia ligeiramente diferente de quais funções ou recursos o software deve oferecer. Depreende-se, portanto, que se deve fazer *um esforço concentrado para compreender o problema antes de desenvolver uma solução de software*.
- Os requisitos de tecnologia de informação demandados por indivíduos, empresas e órgãos governamentais estão se tornando cada vez mais complexos a cada ano. Atualmente, equipes numericamente grandes desenvolvem programas de computador que antigamente eram desenvolvidos por um único indivíduo. Software sofisticado, outrora implementado em um ambiente computacional independente e previsível, hoje em dia está incorporado em tudo, de produtos eletrônicos de consumo a equipamentos médicos e sistemas de armamentos. A complexidade desses novos produtos e sistemas baseados em computadores demanda uma maior atenção para com as interações de todos os elementos do sistema. Depreende-se, portanto, que *projetar tornou-se uma atividade-chave (fundamental)*.
- Indivíduos, negócios e governos dependem, de forma crescente, de software para decisões estratégicas e táticas, assim como para controle e para operações cotidianas. Se o software falhar, as pessoas e as principais empresas poderão vivenciar desde pequenos inconvenientes a falhas catastróficas. Depreende-se, portanto, que *um software deve apresentar qualidade elevada*.
- À medida que o valor de uma aplicação específica aumente, 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 irá aumentar. Conclui-se, portanto, que *um software deve ser passível de manutenção*.

<sup>7</sup> Com o uso de ferramentas modernas, sofisticadas, páginas de sites podem ser produzidas em apenas algumas horas.

<sup>8</sup> Neste livro, mais à frente, chamarei tais pessoas de “interessados”.



FIGURA 1.3

**Camadas da engenharia de software**



Essas simples constatações nos conduzem a uma única 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 tópico principal deste livro — *engenharia de software*.

Embora centenas de autores tenham criado suas definições pessoais de engenharia de software, uma definição proposta por Fritz Bauer [Nau69] na conferência sobre o tema serve de base para discussão:

[Engenharia de software é] o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais.

Ficamos tentados a acrescentar algo a essa definição.<sup>9</sup> Ela diz pouco sobre os aspectos técnicos da qualidade de software; ela não trata diretamente da necessidade de satisfação do cliente ou da entrega do produto dentro do prazo; ela não faz menção à importância das medições e métricas; ela não declara a importância de um processo eficaz. Ainda assim, a definição de Bauer nos fornece uma base. Quais são os “sólidos princípios de engenharia” que podem ser aplicados ao desenvolvimento de software? Como criar software “economicamente viável” e de modo “confiável”? O que é necessário para desenvolver programas de computador que funcionem “de forma eficiente” não apenas em uma, mas sim em várias e diferentes “máquinas reais”? Essas são as questões que continuam a desafiar os engenheiros de software.

A IEEE [IEE93a] desenvolveu uma definição mais abrangente ao afirmar o seguinte:

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).

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. Referindo-se à Figura 1.3, 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<sup>10</sup> promovem uma cultura de aperfeiçoamento contínuo de processos, e é esta cultura que, no final das contas, leva ao desenvolvimento de abordagens cada vez mais efetivas na engenharia de software. A pedra fundamental que sustenta a engenharia de software é o foco na qualidade.

A base para a 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

“Mais do que uma simples disciplina ou ramo do conhecimento, *engineering* é um verbo [engenhara, engendrar], uma palavra de ação, uma maneira de abordar um problema.”

Scott Whitmir

**Como definimos engenharia de software?**

#### PONTO-CHAVE

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

<sup>9</sup> Para inúmeras definições adicionais de *engenharia de software*, consulte: [www.answers.com/topic/software-engineering#wp-note-13](http://www.answers.com/topic/software-engineering#wp-note-13).

<sup>10</sup> A gestão da qualidade e metodologias relacionadas são discutidas no Capítulo 14 e ao longo da Parte 3 deste livro.

**WebRef**

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](http://www.stsc.hill.af.mil).

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 produtos derivados (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 gama de tarefas, que incluem: comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte.

Os métodos da engenharia de software baseiam-se em um conjunto de princípios básicos que governam cada área da tecnologia e inclui atividades de modelagem e outras técnicas descritivas.

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 usadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de software, denominado *engenharia de software com o auxílio do computador*.

## 1.4 O PROCESSO DE SOFTWARE

**?** Quais são os elementos de um processo de software?

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

Ivar Jacobson,  
Grady Booch  
e James Rumbaugh

*Processo* é um conjunto de atividades, ações e tarefas realizadas na criação de algum produto de trabalho (*work product*). Uma *atividade* esforça-se para atingir um objetivo amplo (por exemplo, comunicar-se com os interessados) e é utilizada independentemente do campo de aplicação, do tamanho do projeto, da complexidade de 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 num artefato de software fundamental (por exemplo, um modelo de projeto de arquitetura). Uma *tarefa* se concentra em um objetivo pequeno, porém, bem definido (por exemplo, realizar um teste de unidades) e produz um resultado tangível.

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 irão utilizá-lo.

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 estruturais* 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 — abertas)* aplicáveis em todo o processo de software. Uma metodologia de processo genérica para engenharia de software compreende cinco atividades:

**Comunicação.** Antes de iniciar qualquer trabalho técnico, é de vital importância comunicar-se e colaborar com o cliente (e outros interessados)<sup>11</sup>. A intenção é compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software.

**Planejamento.** Qualquer jornada complicada pode ser simplificada caso exista 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 ser

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

<sup>11</sup> *Interessado* é qualquer um que tenha interesse no êxito de um projeto — executivos, usuários finais, engenheiros de software, o pessoal de suporte etc. Rob Thomsett ironiza: "Interessado (*stakeholder*, em inglês) é uma pessoa que está segurando (*holding*, em inglês) uma estaca [*stake*, em inglês] grande e pontiaguda. Se você não cuidar de seus interessados, você sabe exatamente onde irá parar essa estaca."



"Einstein argumentou que devia 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

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" da coisa, de modo 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 melhor entender as necessidades do software e o projeto que irá atender a essas necessidades.

**Construção.** Essa atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros na codificação.

**Emprego.** O software (como uma entidade completa ou como um incremento parcialmente efetivado) 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 grandes 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 um dos casos, mas as atividades metodológicas permanecerão as mesmas.

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

As atividades metodológicas do processo de engenharia de software são complementadas por uma série de *atividades de apoio*; em geral, estas são aplicadas ao longo de um projeto, ajudando a equipe a gerenciar, a controlar o progresso, a qualidade, as mudanças e o risco. As atividades de apoio típicas são:

**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 reusabilidade** — define critérios para o reúso de artefatos (inclusive componentes de software) e estabelece mecanismos para a obtenção de componentes reutilizáveis.

#### PONTO-CHAVE

Atividades de apoio ocorrem ao longo do processo de software e se concentram, principalmente, no gerenciamento, acompanhamento e controle do projeto.

#### PONTO-CHAVE

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 universais será discutida de forma aprofundada mais adiante.

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

- fluxo geral de atividades, ações e tarefas e suas interdependências;
- grau pelo qual ações e tarefas são definidas dentro de cada atividade da metodologia;
- grau pelo qual artefatos de software são identificados e exigidos;
- modo de aplicar as atividades de garantia de 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 interessados);
- nível de autonomia dada à equipe de software;
- grau de prescrição da organização da equipe.

A Parte I deste livro examina o processo de software com grau de detalhamento considerável. Os *modelos de processo prescritivo* (Capítulo 2) abordam detalhadamente a definição, a identificação e a aplicação de atividades e tarefas do processo. A intenção é melhorar a qualidade do sistema, tornar os projetos mais gerenciáveis, tornar as datas de entrega e os custos mais previsíveis e orientar as equipes de engenheiros de software conforme realizam o trabalho de desenvolvimento de um sistema. Infelizmente, por vezes, tais objetivos não são alcançados. Se os modelos prescritivos forem aplicados de forma dogmática e sem adaptações, poderão aumentar a burocracia associada ao desenvolvimento de sistemas computacionais e, inadvertidamente, criarão dificuldades para todos os envolvidos.

Os *modelos ágeis de processo* (Capítulo 3) ressaltam a “agilidade” do projeto, seguindo princípios que conduzam a uma abordagem mais informal (porém, não menos eficiente) para o processo de software. Tais modelos de processo geralmente são caracterizados como “ágeis”, porque enfatizam a flexibilidade e adaptabilidade. Eles são apropriados para vários tipos de projetos e são particularmente úteis quando aplicações para a Internet são projetadas.

**Como os modelos de processo se diferenciam?**

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

Madame Benoit

**O que caracteriza um processo “ágil”?**

## 1.5 A PRÁTICA DA ENGENHARIA DE SOFTWARE

### WebRef

Uma variedade de citações provocativas acerca da prática da engenharia de software pode ser encontrada em [www.literateprogramming.com](http://www.literateprogramming.com)

A Seção 1.4 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 emprego** —, 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 adquirirá um conhecimento básico dos princípios e conceitos genéricos que se aplicam às atividades de uma metodologia.<sup>12</sup>

### 1.5.1 A essência da prática

Em um livro clássico, *How to Solve It*, sobre os modernos computadores, George Polya [Pol45] apontou em linhas gerais a essência da solução de problemas e, conseqüentemente, a essência da prática da engenharia de software:

1. *Compreender o problema* (comunicação e análise).
2. *Planejar uma solução* (modelagem e projeto de software).

<sup>12</sup> Você deve rever seções relevantes contidas neste capítulo à medida que métodos de engenharia de software e atividades de apoio específicas forem discutidos posteriormente neste livro.





Podese afirmar que a abordagem de Polya é simplesmente questão de bom senso. É verdade. Mas é espantoso quão frequentemente o bom senso é incomum no mundo do software.

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

George Polya

3. *Executar o plano* (geração de código).

4. *Examinar o resultado para ter precisão* (testes e garantia da qualidade).

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 nos é apresentado um problema. Ouvimos por 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 despende um pouco de tempo respondendo a algumas questões simples:

- *Quem tem interesse na solução do problema?* Ou seja, quem são os interessados?
- *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 similares anteriormente?* Existem padrões que são reconhecíveis em uma potencial solução? Existe algum software que implemente os dados, as funções e características necessárias?
- *Algum problema similar 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?

**Execute/leve adiante o plano.** 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" nos permitirá que continuemos sem nos perder.

- *A solução se adequa ao plano?* O código-fonte pode ser atribuído ao modelo de projeto?
- *Cada uma das partes componentes da solução está provavelmente correta?* 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 que se adequam aos dados, às funções e características necessários?* O software foi validado em relação a todas as solicitações dos interessados?

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



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

## 1.5.2 Princípios gerais

O dicionário define a palavra *princípio* como “uma importante afirmação ou lei subjacente 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 porque 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.<sup>13</sup>

### Primeiro princípio: a razão de existir

Um sistema de software existe por uma única razão: *gerar valor a seus usuários*. Todas as decisões deveriam ser tomadas tendo esse princípio em mente. Antes de especificar uma necessidade 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.

### Segundo princípio: KISS (Keep It Simple, Stupid!, ou seja: Faça de forma simples, tapado!)

O projeto de software não é um processo casual; há muitos fatores a ser considerados em qualquer esforço de projeto — *todo projeto deve ser o mais simples possível, mas não tão simples assim*. 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, frequentemente os projetos mais elegantes são os mais simples, o que não significa “rápido e malfeito” — na realidade, simplificar exige muita análise e trabalho durante as iterações, sendo que o resultado será um software de fácil manutenção 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 numa colcha de retalhos de projetos incompatíveis, unidos por parafusos inadequados... Comprometer a visão arquitetônica de um sistema de software debilita e até poderá destruir sistemas bem projetados. Ter um arquiteto responsável e capaz de manter a visão clara e de reforçar a adequação ajuda a assegurar o êxito de um projeto.

### Quarto princípio: o que um produz outros consomem

Raramente um sistema de software de força industrial é construído e utilizado de forma isolada. De uma maneira ou de outra, alguém mais irá usar, manter, documentar ou, de alguma forma, dependerá da capacidade de entender seu sistema. Portanto, *sempre especifique, projete e implemente ciente de que alguém mais terá de entender o que você está fazendo*. O público para qualquer produto de desenvolvimento de software é potencialmente grande. Especifique tendo em vista os usuários; projete, tendo em mente os implementadores; e codifique considerando aqueles que terão de manter e estender o sistema. Alguém terá de depurar o código que você escreveu e isso o faz um usuário de seu código; facilitando o trabalho de todas essas pessoas você agrega maior valor ao sistema.

“Há uma certa majestade na simplicidade que está muito acima de toda a excentricidade do saber.”

Papa Alexandre  
(1688-1744)

## PONTO-CHAVE

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

<sup>13</sup> Reproduzido com a permissão do autor [Hoo96]. Hooker define padrões para esses princípios em <http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment>.



**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. Entretanto, verdadeiros sistemas de software com força industrial devem durar um período muito maior — e, para isso, devem estar preparados para se adaptar a 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 possíveis respostas, criando sistemas que resolvam o problema geral, não apenas aquele específico.*<sup>14</sup> Isso muito provavelmente conduziria à reutilização de um sistema inteiro.

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

A reutilização economiza tempo e esforço;<sup>15</sup> alcançar um alto grau de reúso 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 o maior benefício do uso de tecnologias orientadas a objetos, entretanto, o retorno desse investimento não é automático. Alavancar as possibilidades de reutilização (oferecida pela programação orientada a objetos [ou convencional]) requer 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 antecedência para o reúso 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, aquele que é mais menosprezado. *Pensar bem e de forma clara antes de agir quase sempre produz melhores resultados.* Quando se analisa alguma coisa, provavelmente esta 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 fez parte de um sistema, seu valor aflora. Aplicar os seis primeiros princípios requer intensa reflexão, para a qual as recompensas potenciais 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.

## 1.6 MITOS RELATIVOS AO SOFTWARE

“Na falta de padrões significativos substituindo o folclore, surge uma nova indústria como a do software.”

Tom DeMarco

Os mitos criados em relação ao software — crenças infundadas sobre o software e sobre o processo usado 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 razoáveis (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.

<sup>14</sup> Esse conselho pode ser perigoso se levado a extremos. Projetar para o “problema geral” algumas vezes requer compromissos de desempenho e pode tornar ineficientes as soluções específicas.

<sup>15</sup> Embora isso seja verdade para aqueles que reutilizam o software em futuros projetos, a reutilização pode ser cara para aqueles que precisam projetar e desenvolver componentes reutilizáveis. Estudos indicam que o projeto e o desenvolvimento de componentes reutilizáveis pode custar de 25 a 200% mais que o próprio software. Em alguns casos, o diferencial de custo não pode ser justificado.

**WebRef**

Software Project Managers Network, no endereço [www.spmn.com](http://www.spmn.com), pode ajudá-lo a dissipar esses e outros mitos.

**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 num mito do software, para aliviar a pressão (mesmo que temporariamente).

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

**Realidade:** O livro com padrões pode muito bem existir, mas ele é usado? 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 alinhado 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 ficarmos em dia (algumas vezes denominado conceito da “horda mongol”).

**Realidade:** O desenvolvimento de software não é um processo mecânico como o de fábrica. Nas palavras de Brooks [Bro95]: “acrescentar pessoas num projeto de software atrasado só o tornará mais atrasado ainda”. A princípio, essa afirmação pode parecer um contrassenso, no entanto, o que ocorre é que, quando novas pessoas entram, as que já estavam terão de gastar tempo situando os recém-chegados, reduzindo, conseqüentemente, 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 essa empresa realizá-lo.

**Realidade:** Se uma organização não souber gerenciar e controlar projetos de software, ela irá, 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 encomendou o projeto por contrato. 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.

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

**Realidade:** Embora nem sempre seja possível uma definição ampla e estável dos requisitos, uma definição de objetivos ambígua é receita para um desastre. Requisitos não ambíguos (normalmente derivados da iteratividade) 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 ela foi introduzida. Quando as mudanças dos requisitos são solicitadas cedo (antes do projeto ou da codificação terem começado), o impacto sobre os custos é relativamente pequeno. 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.



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





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

**Mitos dos profissionais da área.** Mitos que ainda sobrevivem nos profissionais da área têm resistido por mais de 50 anos de cultura de programação. Durante seus primórdios, a programação era vista como uma forma de arte. Modos e atitudes antigos dificilmente morrem.

- Mito:** Uma vez feito um programa e o 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 terminá-lo". 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 entre em funcionamento, não há maneira de avaliar sua qualidade.
- Realidade:** Um dos mecanismos de garantia da qualidade de software mais eficaz pode ser aplicado desde a concepção de um projeto — a revisão técnica. Revisores de software (descritos no Capítulo 15) são um "filtro de qualidade" que mostram ser mais eficientes do que testes 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 produtos derivados (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, irá nos retardar.
- Realidade:** A engenharia de software não trata de criação de documentos, trata da criação de um produto de qualidade. Melhor qualidade conduz à redução do retrabalho, e menos retrabalho resulta em maior rapidez na entrega.

Muitos profissionais de software reconhecem a falácia dos mitos que acabamos de descrever. Lamentavelmente, métodos e atitudes habituais fomentam tanto gerenciamento quanto medidas técnicas deficientes, mesmo quando a realidade exige uma abordagem melhor. Ter ciência das realidades do software é o primeiro passo para buscar soluções práticas na engenharia de software.

## 1.7 COMO TUDO COMEÇOU

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 estender 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.

Exceto por uma rápida referência, o software mal foi mencionado como parte da conversação. Ainda assim, o software irá decretar o sucesso ou o fracasso da linha de produtos *CasaSegura*. A empreitada de engenharia terá êxito apenas se o software para a linha *CasaSegura* tiver êxito; e o mercado irá aceitar o produto apenas se o 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.

**CASA SEGURA<sup>16</sup>****Como começa um projeto**

**Cenar:** 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 construindo 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, é conectável a quase tudo. Usa o protocolo sem fio 802.11g, permitindo que acessemos saídas de dispositivos sem o emprego de fios. Acreditamos que nos levará a uma geração de produtos inteiramente nova.

**Joe:** Você concorda, Mal?

**Mal:** Sim. Na verdade, com as vendas tão em baixa quanto 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 a ele sobre nossa ideia, Lisa.

**Lisa:** Trata-se de uma geração completamente nova na linha de "produtos de gerenciamento doméstico". Chamamos esses produtos que criamos de CasaSegura. Eles usam uma nova interface sem fio e oferecem a pequenos empresários e proprietários de casas 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 um 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 sobre o levantamento final.

**Mal:** Os PCs estão em mais de 70% dos lares, se formos capazes de estabelecer um preço baixo para essa coisa, ela poderia se tornar um produto "revolucionário". Ninguém mais tem nosso dispositivo sem fio... Ele é exclusivo! Estaremos 2 anos à frente de nossos concorrentes... E as receitas? Algo em torno de 30 a 40 milhões de dólares no segundo ano...

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

**1.8 RESUMO**

Software é o elemento-chave na evolução de produtos e sistemas baseados em computador e 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.

Softwares — 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.

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.

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 emprego; e elas se aplicam a todos os projetos de software. A prática

<sup>16</sup> 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 inteiramente fictícias, porém as situações e os problemas são reais.



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 de software e das tecnologias necessárias para construí-los. À medida que for aprendendo mais sobre a engenharia de software, você começará a compreender porque esses mitos devem ser derrubados toda vez que se deparar com eles.

### PROBLEMAS E PONTOS A PONDERAR

- 1.1. Cite pelo menos cinco outros exemplos de como a lei das consequências não intencionais se aplica ao software.
- 1.2. Forneça uma série de exemplos (positivos e negativos) que indiquem o impacto do software em nossa sociedade.
- 1.3. Desenvolva suas próprias respostas às cinco perguntas colocadas no início da Seção 1.1. Discuta-as com seus colegas.
- 1.4. Muitas aplicações modernas mudam com frequência — antes de serem apresentadas ao usuário final e só então a 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.
- 1.6. A Figura 1.3 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.
- 1.7. A engenharia de software é aplicável quando as WebApps são construídas? Em caso positivo, como poderia ser modificada para atender às características únicas das WebApps?
- 1.8. 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, cuja falha de um programa de computador poderia causar um grande dano (em termos econômico ou humano).
- 1.9. Descreva uma estrutura de processos 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? Justifique.
- 1.10. 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 de metodologia?
- 1.11. Acrescente dois outros mitos à lista apresentada na Seção 1.6 e declare a realidade que acompanha tais mitos.

### LEITURAS E FONTES DE INFORMAÇÃO COMPLEMENTARES<sup>17</sup>

Há literalmente milhares de livros sobre o tema software. A grande maioria trata de linguagens de programação ou aplicações de software, porém poucos tratam do software em si.

<sup>17</sup> A seção *Leitura e fontes de informação adicionais* apresentada no final de cada capítulo apresenta uma breve visão geral de publicações que podem ajudar a expandir o seu entendimento dos principais tópicos apresentados no capítulo. Há um site bem abrangente para dar suporte ao livro *Engenharia de Software: uma abordagem prática* no endereço [www.mhhe.com/pressman](http://www.mhhe.com/pressman). Entre os diversos tópicos contemplados no site estão recursos de engenharia de software capítulo a capítulo e dicas de sites que poderão complementar o material apresentado em cada capítulo.

Pressman e Herron (*Software Shock*, Dorset House, 1991) apresentaram uma discussão preliminar (dirigida ao grande público) sobre software e a maneira pela qual os profissionais o desenvolvem. O best-seller de Negroponte (*Being Digital*, Alfred A. Knopf, Inc., 1995) dá uma visão geral da computação e seu impacto global no século XXI. 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 desenvolvido.

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. Compaine (*Digital Divide: Facing a Crisis or Creating a Myth*, MIT Press, 2001) defende 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. Livros de 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.

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 sites são dedicados à engenharia de software e ao processo de software.

Foram publicados vários livros sobre o processo de desenvolvimento de software e sobre a engenharia de software nos últimos anos. 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:

- Abran, A. e J. Moore, *SWEBOK: Guide to the Software Engineering Body of Knowledge*, IEEE, 2002.
- Andersson, E. et al., *Software Engineering for Internet Applications*, The MIT Press, 2006.
- 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.
- Jacobson, I., *Object-Oriented Software Engineering: A Use Case Driven Approach*, 2.<sup>a</sup> ed., Addison-Wesley, 2008.
- Jalote, P., *An Integrated Approach to Software Engineering*, Springer, 2006.
- Pfleeger, S., *Software Engineering: Theory and Practice*, 3.<sup>a</sup> ed., Prentice-Hall, 2005.
- Schach, S., *Object-Oriented and Classical Software Engineering*, 7.<sup>a</sup> ed., McGraw-Hill, 2006.
- Sommerville, I., *Software Engineering*, 8.<sup>a</sup> ed., Addison-Wesley, 2006.
- Tsui, F. e O. Karam, *Essentials of Software Engineering*, Jones & Bartlett Publishers, 2006.

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

Uma ampla gama de fontes de informação sobre engenharia de software e processo de software se encontra à disposição na Internet. Uma lista atualizada de referências relevantes para o processo para o software pode ser encontrada no site [www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm](http://www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm).