CAPÍTULO 1

SOFTWARE E ENGENHARIA DE SOFTWARE

CONCEITOS-CHAVE

Ele 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!

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

The Wall Street Journal

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

* 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.

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.

> "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**

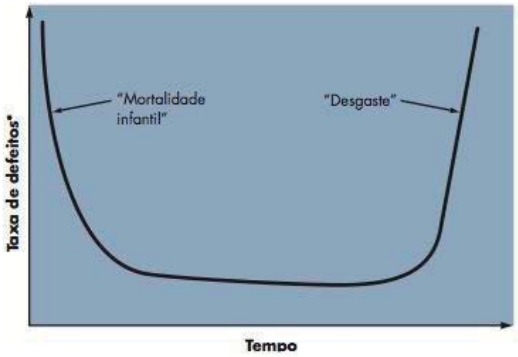
O papel desempenhado pelo software tem passado por grandes mudanças ao longo dos últimos cinquenta anos. Aperfeiçoamentos significativos no desempenho do hardware, mudanças profundas nas arquiteturas computacionais, vasto aumento na capacidade de memória e armazenamento, e uma ampla variedade de exóticas 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:[1]

- 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?

1 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:

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 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:

PONTO-CHAVE
Software é um processo de engenharia, não é fabricação.

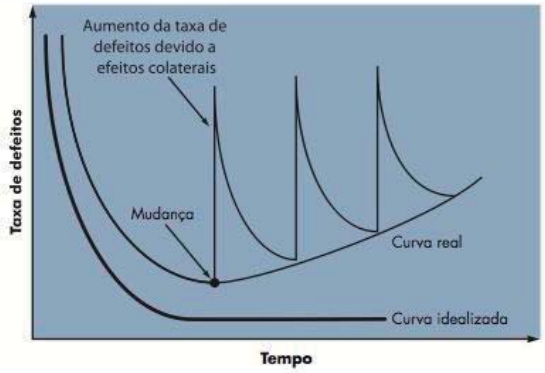
1. *Software é desenvolvido ou passa por um processo de engenharia; ele não é fabricado no sentido clássico.*

   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.

FIGURA 1.2
Curvas de defeitos para software



PONTO-CHAVE
Software não se desgasta, mas ele se deteriora.

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.

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

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[2], 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.

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.

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.

2 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 Zebehazy

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.[3] 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.[4] 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.

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

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.

---

3 O desenvolvimento com base em componentes é discutido no Capítulo 10.

4 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 automóveis, 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.

"Não existe computador que tenha bom senso."
Marvin Minsky

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.

Netsoursing (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.