CAPÍTULO

31 TENDÊNCIAS EMERGENTES NA ENGENHARIA DE SOFTWARE

CONCEITOS-CHAVE

Durante a história relativamente recente da engenharia de software, profissionais e pesquisadores desenvolveram uma série de modelos de processo, métodos técnicos e ferramentas automatizadas em um esforço para apoiar a mudança na maneira como criamos software para computador. Apesar de a experiência indicar o contrário, há um desejo implícito de se encontrar a "solução mágica" — o processo mágico ou tecnologia transcendente que nos permitirá criar facilmente sistemas grandes e complexos, sem confusão, enganos e atrasos — sem os vários problemas que continuam a povoar o trabalho de software.

Mas a história indica que nossa busca pela solução mágica parece estar fadada ao fracasso. Novas tecnologias são introduzidas regularmente, apresentadas como sendo a "solução" para muitos dos problemas que os engenheiros de software enfrentam e incorporadas nos projetos grandes e pequenos. Críticos da indústria exageram a importância dessas "novas" tecnologias de software, os conhecedores (*cognoscenti*) da comunidade do software as adotam com entusiasmo, e, por fim, elas desempenham um papel no mundo da engenharia de software, mas tendem a não produzir o resultado esperado, e, como consequência, a busca continua.

Mili e Cowan [Mil00b] comentam sobre os desafios que enfrentamos ao tentarmos isolar tendências significativas na tecnologia:

> ***Que fatores determinam o sucesso de uma tendência?*** O que caracteriza as tendências bem-sucedidas: Seu mérito técnico? Sua habilidade para abrir novos mercados? Sua habilidade para alterar a economia dos mercados existentes?
>
> ***Qual é o ciclo de vida de uma tendência?*** Embora a visão tradicional de que as tendências evoluem ao longo de um ciclo de vida bem definido e previsível, que vai da pesquisa a um produto acabado, por meio de um processo de transferência, descobrimos que muitas encurtaram esse ciclo ou seguiram outro.
>
> ***Com que antecedência pode uma tendência bem-sucedida ser identificada?*** Se soubermos identificar fatores de sucesso e/ou se entendermos o ciclo de vida de uma tendência, podemos detectar sinais precoces de êxito. Retoricamente, buscamos a habilidade para reconhecer a próxima tendência antes dos outros.

PANORAMA

O que é? Ninguém pode prever o futuro com certeza absoluta. Mas é possível avaliar tendências na área de engenharia de software e a partir dessas tendências sugerir direções possíveis para a tecnologia. É isso que eu tento fazer neste capítulo.

Quem realiza? Qualquer um que deseje dedicar seu tempo para se manter atualizado sobre os problemas da engenharia de software pode tentar prever a direção futura da tecnologia.

Por que é importante? Porque os reis na antiguidade consultavam os adivinhos? Porque grandes corporações multinacionais contratam firmas de consultoria e se esforçam para preparar previsões? Porque uma grande parte do público lê horóscopos? Queremos sempre saber o que vai acontecer para nos preparar.

Quais são as etapas envolvidas? Não há uma fórmula para prever o caminho à frente. Tentamos fazer isso coletando dados, organizando esses dados para que nos forneçam informações úteis, examinando associações sutis para extrair conhecimento, e desse conhecimento sugerir prováveis tendências que preveem como as coisas serão em algum tempo futuro.

Qual é o artefato? Uma visão do futuro próximo que pode ser ou não correta.

Como garantir que o trabalho foi realizado corretamente? Prever o caminho à frente é uma arte, não uma ciência. De fato, é muito raro que uma previsão séria sobre o futuro esteja absolutamente certa ou completamente errada (com exceção, felizmente, das previsões do fim do mundo). Nós procuramos tendências e tentamos extrapolá-las. Podemos avaliar a precisão dessa extrapolação somente com o passar do tempo.

Quais são as "grandes questões" quando consideramos a evolução da tecnologia?

Quais os aspectos controláveis da evolução? Podem as corporações usar sua influência no mercado para impor tendências? Pode o governo usar seus recursos para impor tendências? Qual é o papel das normas e padrões na definição das tendências? Uma cuidadosa análise comparativa entre Ada e Java, por exemplo, poderia ser esclarecedora nesse aspecto?

Não há uma resposta simples a essas questões e não há dúvida de que as tentativas para identificar tecnologias significativas são, no melhor caso, medíocres.

Em edições anteriores deste livro (durante os últimos 30 anos), tenho discutido tecnologias emergentes e seu impacto sobre a engenharia de software. Algumas foram amplamente adotadas, outras nunca alcançaram seu potencial. Minha conclusão: as tecnologias vêm e vão; as tendências reais que devemos explorar são mais flexíveis. Em outras palavras, o progresso na engenharia de software será orientado pelas tendências nos negócios, organizações, mercado e tendências culturais. Estas levam à inovação tecnológica.

Neste capítulo, examinaremos algumas tendências tecnológicas na engenharia de software, mas minha ênfase será sobre algumas tendências nos negócios, nas organizações, no mercado e culturais que podem ter influência importante sobre a tecnologia de engenharia de software durante os próximos 10 ou 20 anos.

31.1 Evolução da Tecnologia

Em uma obra fascinante que oferece uma visão atraente sobre como as tecnologias de computação (e outras tecnologias relacionadas) irão evoluir, Ray Kurzweil [Kur05] afirma que a evolução tecnológica é similar à biológica, mas ocorre a uma velocidade cuja magnitude é muitas vezes maior. A evolução (seja ela biológica, seja tecnológica) ocorre em resultado de uma realimentação positiva — "os métodos mais capacitados resultantes de um estágio do progresso evolucionário são usados para criar o próximo estágio" [Kur06].

As grandes questões para o século 21 são: (1) Quão rápido uma tecnologia evolui? (2) Quão significativos são os efeitos da realimentação positiva? (3) Quão profundas serão as mudanças resultantes?

> "Previsões são muito difíceis de fazer, especialmente quando se trata de futuro."
>
> **Mark Twain**

Quando é introduzida uma tecnologia bem-sucedida, o conceito inicial transforma-se em "ciclo de vida de inovação" razoavelmente previsível [Gai95], ilustrado na Figura 31.1. Na fase *avanço*, um problema é identificado e tentativas repetidas são realizadas em busca de uma solução viável. Em algum ponto, aparece uma promessa de solução. O trabalho inicial de avançar é reproduzido na fase *replicador* e ganha um uso mais amplo. O *empirismo* leva à criação de regras que regem o uso da tecnologia, e o sucesso repetido leva a uma *teoria* de uso mais amplo seguida pela criação de ferramentas automatizadas durante a fase da *automação*. Por fim, a tecnologia amadurece e passa a ser amplamente utilizada.

Devemos observar que muitas pesquisas e tendências tecnológicas nunca atingem a maturidade. Na verdade, a grande maioria das tecnologias "promissoras" no domínio da engenharia de software suscita um grande interesse por alguns anos e depois passa a ser usada por um grupo dedicado de usuários. Isso não significa que não tenham mérito, mas que o caminho através da inovação é longo e difícil.

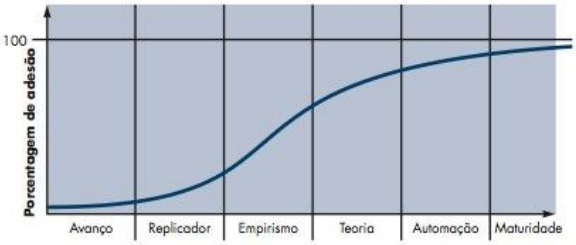
PONTO-CHAVE

A tecnologia da computação está evoluindo a uma taxa exponencial, e seu crescimento pode logo se tornar explosivo.

Kurzweil [Kur05] concorda que as tecnologias de computação evoluem através de uma "curva-S": apresentam um crescimento relativamente lento durante os anos de formação da tecnologia, rápida aceleração durante a fase de crescimento e depois um período de nivelamento quando atinge seus limites. Mas a tecnologia da computação e outras relacionadas têm mostrado um crescimento explosivo (exponencial) durante os estágios centrais (veja a Figura 31.1) e assim continuarão. Além disso, quando uma curva-S termina, outra a substitui com um crescimento ainda mais explosivo durante seu período de crescimento.[1] Hoje, estamos no joelho da curva-S das modernas tecnologias de computação — na transição entre o crescimento inicial e o crescimento explosivo

1 Por exemplo, os limites dos circuitos integrados podem ser esgotados na próxima década, mas essa tecnologia pode ser substituída pelas tecnologias de computação molecular e por outra curva-S acelerada.

FIGURA 31.1
Um ciclo de vida da evolução tecnológica



que deve se seguir. A implicação é que durante os próximos 20 a 40 anos, veremos mudanças significativas (até mesmo impressionantes) na capacidade de computação. As décadas vindouras resultarão em mudanças de grande magnitude na computação em termos de velocidade, tamanho, capacidade e consumo de energia (para citar apenas algumas características).

Kurzweil [Kur05] sugere que, em 20 anos, a evolução tecnológica irá acelerar a um passo cada vez mais rápido, chegando finalmente à era de inteligência não biológica que se combinará com a inteligência humana e a ampliará de maneira fascinante.

E tudo isso, não importa como evolua, necessitará de software e sistemas que na comparação fazem nossos esforços atuais parecerem infantis. Por volta de 2040, uma combinação de computação extrema, nanotecnologia, redes com larguras de banda extremamente altas e robótica nos levará a um mundo diferente.[2] O software, possivelmente na forma em que ainda não podemos compreender, continuará existindo no centro desse mundo novo. A engenharia de software não irá acabar.

31.2 OBSERVANDO TENDÊNCIAS NA ENGENHARIA DE SOFTWARE

"Acredito poder haver um mercado mundial talvez para cinco computadores."

Thomas Watson, presidente da IBM, 1943

A Seção 31.1 considerou brevemente as possibilidades fascinantes que podem surgir das tendências de longo prazo na computação e tecnologias relacionadas. E quanto ao curto prazo?

Barry Boehm [Boe08] sugere que "os engenheiros de software enfrentarão desafios muitas vezes enormes de tratar com rápidas mudanças, incerteza e emergência, confiabilidade, diversidade e interdependência, mas eles também têm oportunidades de fazer contribuições significativas". Mas quais as tendências que nos possibilitarão o enfrentamento desses desafios nos próximos anos?

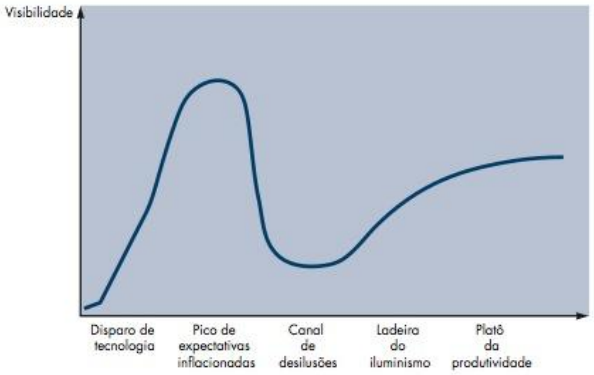
Na introdução deste capítulo, afirmei que "as tendências leves" têm um impacto significativo sobre a direção geral da engenharia de software. Mas outras tendências ("mais pesadas") orientadas para a tecnologia de pesquisa permanecem importantes. As tendências nas pesquisas "são motivadas por percepções gerais do estado da arte e da prática, por percepções do pesquisador sobre as necessidades dos profissionais, por programas de fundos nacionais que buscam estratégias específicas e por puro interesse técnico" [Mil00a]. Tendências tecnológicas ocorrem quando pesquisas são extrapoladas para atender às necessidades da indústria e demandas do marketing.

PONTO-CHAVE
O "ciclo da excelência" apresenta uma visão realística da integração da tecnologia no curto prazo. No entanto, a tendência no longo prazo é exponencial.

Na seção anterior, abordei o modelo da curva-S para a evolução da tecnologia. A curva-S é apropriada para considerar os efeitos de longo prazo das tecnologias básicas à medida que evoluem. Mas o que ocorre com as inovações, ferramentas e métodos mais modestos e de curto prazo? O Gartner Group [Gar08] — uma consultoria que estuda tendências da tecnologia em muitos ramos de atividade — desenvolveu um *ciclo da excelência para tecnologias emergentes*, representado na Figura 31.2. O ciclo Gartner Group apresenta cinco fases:

2 Kurzweil [Kur05] apresenta um argumento técnico razoável que prevê forte tendência da inteligência artificial (que passará pelo teste de Turing) em 2029 e sugere que a evolução de humanos e máquinas começará a se combinar em 2045. A grande maioria dos leitores deste livro viverá para ver se isso de fato ocorrerá.

FIGURA 31.2
O ciclo da excelência do Gartner Group para tecnologias emergentes
Fonte: (Gar08)



- *Disparo da tecnologia* – avanço na pesquisa ou lançamento de um produto inovador que leva a grande cobertura da mídia e entusiasmo popular.
- *Pico de expectativas inflacionadas* – entusiasmo demasiado e projeções extremamente otimistas do impacto com base em um sucesso limitado, mas muito bem publicado.
- *Desilusão* – projeções demasiado otimistas do impacto não se confirmam, e os críticos começam a se manifestar; a tecnologia torna-se sem atrativos entre os conhecedores.
- *Ladeira do iluminismo* – utilização cada vez maior por uma ampla variedade de empresas leva a um melhor entendimento do verdadeiro potencial da tecnologia; aparecem métodos e ferramentas prontos para uso para suportar a tecnologia.
- *Platô de produtividade* – os benefícios do mundo real agora são óbvios, e o uso atinge uma porcentagem significativa do mercado em potencial.

Nem toda a tecnologia de engenharia de software segue esse caminho ao longo do ciclo da excelência. Em alguns casos, a decepção é justificada e a tecnologia fica relegada à obscuridade.

31.3 IDENTIFICANDO AS "TENDÊNCIAS LEVES"

"640K deve ser suficiente para todos."
Bill Gates, presidente da Microsoft, 1981

Quais as tendências leves que afetarão as tecnologias relacionadas à engenharia de software?

Cada país com uma indústria de TI substancial tem um conjunto único de características que definem como os negócios são conduzidos, as dinâmicas organizacionais que surgem dentro de uma empresa, os aspectos de marketing que se aplicam a clientes locais e a cultura dominante que governa toda a interação humana. No entanto, algumas tendências em cada uma dessas áreas são universais e têm tanto a ver com sociologia, antropologia e psicologia de grupo (muitas vezes chamadas de "ciências leves") quanto com pesquisa acadêmica ou industrial.

Conectividade e colaboração (possibilitadas pelas comunicações em banda larga) permitem a existência de equipes de software que não ocupam o mesmo espaço físico (teletrabalho e emprego de tempo parcial em um contexto local). Uma equipe colabora com outras que estão separadas por fuso horário, linguagem e cultura. A engenharia de software deve responder com um modelo de processo abrangente para "equipes distribuídas", que seja ágil o bastante para atender às demandas da urgência, mas disciplinado o suficiente para coordenar diferentes grupos.

A *globalização* leva à força de trabalho diversificada (em termos de linguagem, cultura, solução de problemas, filosofia de administração, prioridades de comunicação e interação entre as pessoas). Isso, por sua vez, exige uma estrutura organizacional flexível. Equipes diferentes (em países diferentes) devem responder a problemas de engenharia de maneira que melhor atenda suas necessidades especiais, enquanto proporciona uniformidade para a execução de um proje-

to global. Esse tipo de organização sugere menos níveis de administração e ênfase maior na tomada de decisões em equipe. Pode levar a uma maior agilidade, mas apenas se mecanismos de comunicação tenham sido estabelecidos para que cada equipe possa entender o projeto e o status técnico (via conexão em rede) a qualquer tempo. Métodos e ferramentas da engenharia de software podem contribuir para a obtenção de algum nível de uniformidade (equipes falam a mesma "linguagem" implementada por métodos e ferramentas específicos). O processo de software pode proporcionar a estrutura para a existência desses métodos e ferramentas.

Em algumas regiões (nos Estados Unidos e na Europa, por exemplo), a população está envelhecendo. Essa inegável tendência demográfica (e cultural) indica que muitos engenheiros de software e gerentes experientes estarão deixando o campo de atividade durante a próxima década. A comunidade de engenharia de software deve responder com mecanismos viáveis que capturem o conhecimento desses gerentes e técnicos [por exemplo, o uso de *padrões* (Capítulo 12) é um passo na direção certa], para que fique à disposição das gerações futuras de profissionais. Em outras regiões do mundo, o número de jovens disponíveis para a indústria de software está explodindo. É a oportunidade de moldar uma cultura de engenharia de software sem o ônus de 50 anos de prejuízos da "velha escola".

Estima-se que mais de um bilhão de novos consumidores entrarão no mercado mundial na próxima década. Os gastos dos consumidores nas "economias emergentes dobrarão, passando de 9 trilhões de dólares" [Pet06]. Não há dúvida de que uma porcentagem significativa desse gasto será com produtos e serviços que tenham um componente digital — que são baseados em software ou controlados por software. A implicação é uma demanda crescente por novos softwares. A questão então é, "Podem ser desenvolvidas novas tecnologias de engenharia de software para atender a essa demanda mundial?". As tendências do mercado moderno muitas vezes são controladas pelo lado do fornecimento.[3] Em outros casos, requisitos no lado da demanda controlam o mercado. Em qualquer das situações, um ciclo de inovação e demanda ocorre de maneira que muitas vezes torna-se difícil determinar quem vem primeiro!

Por fim, a própria cultura humana irá na direção da engenharia de software. Toda geração deixa sua própria marca na cultura local, e a nossa não será diferente. Faith Popcorn [Pop08], um conhecido consultor que se especializou em tendências culturais, caracteriza-as da seguinte maneira: "Nossas Tendências não são modismos. Nossas Tendências permanecem. Nossas Tendências evoluem. Elas representam forças escondidas, causas principais, necessidades humanas básicas, atitudes, aspirações. Elas nos ajudam a navegar pelo mundo, entender o que está acontecendo e por quê, e nos preparar para aquilo que ainda virá". Uma discussão detalhada sobre como as tendências culturais modernas terão um impacto sobre a engenharia de software é mais bem apresentada por aqueles que se especializaram nas "ciências leves".

31.3.1 Administrando a complexidade

Quando foi escrita a primeira edição deste livro (1982), produtos digitais da forma como conhecemos hoje não existiam, e os sistemas baseados em mainframe contendo um milhão de linhas de código-fonte (*lines of source code* — LOC) eram considerados muito grandes. Hoje, não é raro encontrar pequenos dispositivos digitais com 60 mil a 200 mil linhas de código-fonte de software personalizado, com alguns milhões de linhas de código para recursos do sistema operacional. Sistemas modernos baseados em computador contendo de 10 a 50 milhões de linhas de código não são incomuns.[4] Em um futuro relativamente próximo, começarão a surgir sistemas[5] que requerem mais de 1 bilhão de linhas de código.[6]

---

3 O lado fornecedor adota nos mercados uma abordagem do tipo "faça e eles comprarão". Tecnologias especiais são criadas, e os consumidores se aglomeram para adotá-las — às vezes!

4 Os sistemas operacionais modernos dos PCs (por exemplo, Linux, MacOS e Windows) têm de 30 a 60 milhões de LOC. Sistemas operacionais de dispositivos móveis podem ter mais de 2 milhões de LOC.

5 Na realidade, esse "sistema" será um sistema de sistemas — centenas de aplicativos interoperáveis trabalhando juntos para atingir algum objetivo comum.

6 Nem todos os sistemas complexos são grandes. Um aplicativo relativamente pequeno (digamos, menos de 100 mil LOC) pode ainda ser bastante complexo.

Pense nisso por um instante!

Considere as interfaces para um sistema de 1 bilhão de linhas de código, para o mundo exterior, para outros sistemas interoperáveis, para a Internet (ou seu sucessor), e para os milhões de componentes internos que devem funcionar todos juntos para fazer esse monstro da computação operar corretamente. Há uma maneira confiável de garantir que todas essas conexões permitam que as informações fluam adequadamente?

Considere o próprio projeto. Como administramos o fluxo do trabalho e acompanhamos o progresso? As abordagens convencionais poderão ser escaladas muitas vezes em ordem de grandeza?

"Não há razão para uma pessoa querer ter um computador em casa."

Ken Olson, presidente e fundador da Digital Equipment Corp., 1977

Considere o número de pessoas (e suas localizações) que estarão fazendo o trabalho, a coordenação dos profissionais e da tecnologia, o fluxo ininterrupto de alterações, a possibilidade de ambiente de sistema multiplataforma, multioperacional. Há uma maneira de administrar e coordenar indivíduos que estão trabalhando em um projeto enorme?

Considere o desafio da engenharia. Como podemos analisar dezenas de milhares de requisitos, limitações e restrições de uma forma que garanta que a inconsistência e ambiguidade, omissões e erros imediatamente sejam descobertos e corrigidos? Como podemos criar uma arquitetura de projeto que seja robusta o bastante para lidar com um sistema desse tamanho? Como os engenheiros de software poderão estabelecer um sistema de controle de alterações que terá de manipular centenas de milhares de alterações?

Considere o desafio da garantia da qualidade. Como podemos executar verificação e validação de modo significativo? Como você testa um sistema de 1 bilhão de LOC?

No passado, os engenheiros de software tentaram administrar a complexidade de uma forma que somente pode ser descrita como especial. Hoje, usamos processos, métodos e ferramentas para manter sob controle a complexidade. Mas o que acontecerá no futuro? A nossa abordagem atual estará à altura da tarefa a ser realizada?

31.3.2 Software aberto

Conceitos como inteligência ambiente,[7] aplicações ligadas a contexto, e computação invasiva/onipresente — todos focalizam a integração de sistemas baseados em software em um ambiente mais amplo do que um PC, um dispositivo de computação móvel, ou qualquer outro dispositivo digital. Essas visões separadas do futuro próximo da computação sugerem coletivamente o "software aberto" — software que é projetado para se adaptar a um ambiente em contínua mudança "auto-organizando sua estrutura e auto-adaptando seu comportamento" [Bar06].

Para ajudar a ilustrar os desafios que os engenheiros de software enfrentarão em um futuro previsível, considere a noção de *inteligência ambiente* (*ambient intelligence* — amI). Ducatel [Duc01] define a amI da seguinte maneira: "Pessoas estão rodeadas por interfaces inteligentes e intuitivas que estão embutidas em todos os tipos de objetos. O espaço da inteligência ambiente é capaz de reconhecer e responder à presença de diferentes indivíduos [enquanto trabalham] de uma maneira contínua e sem obstrução".

Examinemos uma visão do futuro próximo na qual a amI se tornou onipresente. Você acaba de comprar um comunicador pessoal (chamado P-com, um dispositivo móvel de bolso) e passou as últimas semanas criando[8] sua "imagem" — tudo, desde sua agenda diária, de coisas a fazer, livro de endereços, registros médicos, informações comerciais, documentos de viagem (coisas que você está procurando, por exemplo, um livro específico, um vinho raro, um curso local de arte em vidraria) e "Digital-Me" (D-Me) que o descreve com um nível de detalhes que possibilita uma apresentação digital aos outros (um tipo de *MySpace* ou *FaceBook* que anda com você). O P-com contém um identificador pessoal denominado "chave das chaves" — um identificador

7 Uma introdução muito boa e detalhada sobre inteligência ambiente pode ser encontrada no endereço **www.emergingcommunication.com/volume6.html.** Mais informações podem ser obtidas em **www.ambientintelligence.org/**.

8 Toda a interação com o P-com ocorre via comandos de reconhecimento de voz e instruções, que evoluíram tornando-se 99% precisos.

pessoal multifuncional que poderia fornecer acesso e habilitar consultas de uma grande variedade de dispositivos amI e sistemas.

É óbvio que entram em jogo aspectos significativos de privacidade e segurança. Um "sistema de gerenciamento de confiança — *trust management system*" [Duc01] será parte integral da amI e controlará os privilégios que possibilitam a comunicação com sistemas de redes, saúde, entretenimento, finanças, emprego e pessoal.

Novos sistemas com capacidade amI serão acrescentados à rede constantemente, cada um deles proporcionando recursos úteis e demandando acesso ao seu P-com. Portanto, o software P-com deve ser projetado de forma que possa se adaptar aos requisitos que surgem sempre que novos sistemas amI estejam on-line. Há muitas maneiras de fazer isso, mas resumindo: o P-com deve ser flexível e robusto em aspectos que o software convencional não consegue atingir.

31.3.3 Requisitos emergentes

No início de um projeto de software, há uma autenticidade que se aplica igualmente a todos os envolvidos: "Você não sabe o que não sabe". Isso significa que os clientes raramente definem requisitos "estáveis". Indica também que os engenheiros de software não podem prever onde ocorrerão as ambiguidades e inconsistências. Os requisitos mudam — mas isso não é novidade.

Em virtude de os requisitos emergentes já serem uma realidade, sua organização deverá pensar em adotar um modelo de processo incremental.

Na medida em que os sistemas se tornam mais complexos, mesmo uma tentativa rudimentar de definir requisitos abrangentes está destinada ao fracasso. Uma definição de objetivos globais pode ser possível, pode ser conseguido um esboço dos objetivos intermediários, mas requisitos estáveis — sem chance! Os requisitos surgirão conforme todos os envolvidos na engenharia e construção de um sistema complexo aprenderem mais sobre esse sistema, sobre o ambiente no qual ele reside e sobre os usuários que vão interagir.

Essa realidade implica uma série de tendências de engenharia de software. Primeiro, devem ser desenhados modelos de processo para aderir à mudança e adotar as crenças básicas da filosofia ágil (Capítulo 3). Em seguida, métodos que resultem em modelos elaborados (por exemplo, modelos de requisito e de desenho) devem ser usados criteriosamente, porque esses modelos mudarão repetidamente à medida que for adquirido mais conhecimento sobre o sistema. Por fim, ferramentas que suportem tanto os processos quanto os métodos devem facilitar a adaptação e mudança.

Mas há outro aspecto dos requisitos emergentes. Para a grande maioria do software desenvolvido até hoje, a fronteira entre o sistema baseado em software e seu ambiente externo é estável. A fronteira pode mudar, mas isso ocorrerá de uma maneira controlada, permitindo que o software seja adaptado como parte de um ciclo de manutenção. Essa opinião está começando a mudar. O software aberto (*open-world software*) (Seção 31.2.2) exige que os sistemas baseados em computador "se adaptem e reajam às mudanças dinamicamente, mesmo que sejam imprevistas" [Bar06].

Pela sua natureza, os requisitos emergentes levam à mudança. Como controlamos a evolução de um aplicativo ou sistema amplamente usado durante toda a sua vida útil, e que efeito isso tem sobre a maneira como projetamos software?

Conforme o número de alterações aumenta, a possibilidade de efeitos colaterais não desejados também aumenta. Isso deverá ser um motivo de preocupação quando sistemas complexos com requisitos emergentes se tornarem comuns. A comunidade de engenharia de software deve desenvolver métodos que ajudem as equipes a prever o impacto das mudanças no sistema inteiro, moderando assim efeitos colaterais indesejados. Hoje, nossa habilidade para tanto é severamente limitada.

31.3.4 O mix de talentos

A natureza de uma equipe de engenharia de software pode mudar à medida que os sistemas baseados em software ficarem mais complexos, as comunicações e colaboração entre equipes globais transformarem-se em lugar-comum e os requisitos emergentes (com o fluxo de mu-

danças resultantes) se tornarem a norma. Cada equipe de software deve contribuir com talento criativo e habilidades técnicas para sua parte de um sistema complexo, e o processo todo deve permitir que o resultado dessas ilhas de talento se combine efetivamente.

Alexandra Weber Morales [Mor05] sugere o mix de talentos de uma "equipe ideal de software" (*software dream team*). O *Cérebro* é o arquiteto principal capaz de lidar com as demandas dos interessados e mapeá-las em uma estrutura de tecnologia extensível e implementável. O *Data Grrl* é o guru de base de dados e estrutura de dados que "ataca vigorosamente através de linhas e colunas com profundo conhecimento da lógica de predicados e teoria dos conjuntos no que se refere ao modelo relacional". O *Blocker* é um líder técnico (gerente) que permite que a equipe trabalhe livre de interferências enquanto ao mesmo tempo garante que a colaboração está ocorrendo. O *Hacker* é um programador perfeito que está à vontade com padrões e linguagens e pode usar ambas eficazmente. O *Gatherer* "descobre habilmente requisitos de sistema com visão antropológica" e os expressa com clareza.

31.3.5 Blocos básicos de software

"A resposta criativa para a tecnologia digital é adotá-la como uma nova janela sobre tudo o que é eternamente humano, e usá-la com paixão, sabedoria, bravura e alegria."

Ralph Lombreglia

Todos que já adotamos uma filosofia de engenharia de software já enfatizamos a necessidade de reutilização — de código-fonte, classes orientadas a objeto, componentes, padrões e software de prateleira. Embora a comunidade de engenharia tenha feito progresso na sua tentativa de capturar conhecimento e reutilizar soluções aprovadas, uma porcentagem significativa do software atual continua a ser criada "desde o início". Parte da razão de tal procedimento é um contínuo desejo (por parte dos interessados e profissionais de engenharia de software) de "soluções únicas".

No mundo do hardware, os OEMs (*original equipment manufactures*) dos dispositivos digitais usam produtos-padrão específicos de aplicativo (*application-specific standard products* — ASSPs) produzidos por fabricantes de circuitos integrados quase exclusivamente. Esse "mercador de hardware" fornece os blocos básicos necessários para implementar tudo, desde um telefone móvel até um HD-DVD player. Cada vez mais, os mesmos OEMs usam o "mercador de software" — blocos básicos de software projetados especificamente para um domínio de aplicação único [por exemplo, dispositivos VoIP]. Michael Ward [War07] comenta:

> Uma vantagem do uso de componentes de software é que o OEM pode alavancar a funcionalidade proporcionada pelo software sem ter de desenvolver especialidades internas nas funções específicas ou investir tempo de desenvolvedor no trabalho de implementar e validar os componentes. Outras vantagens incluem a habilidade para adquirir e fornecer apenas o conjunto específico de funcionalidades necessárias ao sistema, bem como a habilidade para integrar esses componentes em uma arquitetura já existente.
>
> No entanto, a abordagem de componente de software tem uma desvantagem porque há certo nível de esforços necessários para integrar os componentes individuais no produto global. Esse desafio pode ser ainda mais complicado se os componentes originam-se de uma variedade de fornecedores, cada um com suas próprias metodologias de interface. Conforme outras fontes de componentes forem usadas, aumenta o trabalho de administrar um número maior de fornecedores, e há um risco maior de se encontrarem problemas com a interação através de componentes de diferentes origens.

Além dos componentes agregados como um mercado de software, há uma tendência crescente em adotar *soluções de plataforma de software* que "incorporam coleções de funcionalidades relacionadas, fornecidas tipicamente em uma estrutura de software integrada" [War07]. Uma plataforma de software libera um OEM do trabalho associado ao desenvolvimento de funcionalidade básica e, em vez disso, permite que o OEM dedique trabalho de software para aquelas características que diferenciam seu produto.

31.3.6 Mudando as percepções de "valor"

Durante os últimos 25 anos do século 20, a pergunta operativa que os homens de negócios faziam ao discutirem software era: "Por que custa tão caro?". Essa pergunta raramente é feita

nos dias de hoje e ela foi substituída por outra: "Por que não podemos obter esse (software e/ou produto baseado em software) mais rapidamente?".

Quando se considera software para computador, a percepção moderna está mudando de valor nos negócios (custo e lucratividade) para valores de clientes que incluem rapidez na entrega, riqueza de funcionalidade e qualidade geral do produto.

31.3.7 Código aberto

Quem é o proprietário do software que você ou sua organização utiliza? Cada vez mais, a resposta é "todos". O movimento "código aberto" (*open source*) tem sido descrito da seguinte maneira [OSO08]: "Código aberto é um método de desenvolvimento de software que utiliza o poder da revisão por pares (*peer review*) distribuída e a transparência do processo. A premissa do código aberto é melhor qualidade, maior confiabilidade, maior flexibilidade, menor custo e o fim do aprisionamento tecnológico (*vendor lock-in*) predatório". O termo *código aberto*, quando aplicado a software de computador, implica que os produtos de engenharia de software (modelos, código-fonte, conjuntos de teste) são abertos ao público e podem ser revistos e ampliados (com controles) por qualquer um com interesse e permissão.

Uma "equipe" de código aberto tem um conjunto de membros, em tempo integral, do tipo "time dos sonhos" (Seção 31.3.4); o número de pessoas trabalhando no software aumenta e diminui conforme a necessidade da aplicação. O poder da equipe de código aberto é derivado da constante revisão por pares e refatoração (*refactoring*) de projeto/código que resulta em uma lenta progressão em direção a uma solução ótima.

Se você tiver mais interesse, Weber [Web05] fornece uma introdução valiosa, e Feller e seus colegas [Fel07] editaram uma antologia abrangente e objetiva que considera os benefícios e problemas associados a código aberto.

INFORMAÇÕES

Tecnologias a ser observadas

Muitas tecnologias emergentes podem ter um impacto significativo sobre os tipos de sistemas baseados em computadores que evoluem. Elas se somam aos desafios que confrontam os engenheiros de software. Vale notar as seguintes tecnologias:

Grid computing – Esta tecnologia (disponível hoje) cria uma rede que aproveita os bilhões de ciclos de CPU não utilizados de cada máquina na rede e permite que a computação de trabalhos extremamente complexos seja feita sem precisar de um supercomputador dedicado. Para um exemplo abrangendo mais de 4,5 milhões de computadores, visite o site http://setiathome.berkeley.edu/.

Computação aberta – "Ambiente, implícito, invisível e adaptável. Situação em que os dispositivos de rede agregados no ambiente proporcionam conectividade e serviços que passam despercebidos todo o tempo" [McC05].

Microcomércio – Novo ramo do comércio eletrônico que cobra quantias muito pequenas para o acesso ou compra de várias formas de propriedade intelectual. O iTunes da Apple é um exemplo bastante usado.

Máquinas cognitivas – O "santo graal" do campo da robótica é desenvolver máquinas que tenham ciência de seu ambiente, que possam "captar informações, responder a situações em contínua mudança e interagir com pessoas naturalmente" [PCM03]. As máquinas cognitivas ainda estão nos primeiros estágios de desenvolvimento, mas o potencial é enorme.

Monitores OLED – O monitor OLED (OLED *display*) "usa uma molécula traçadora à base de carbono que emite luz quando uma corrente elétrica passa através dela. Junte grandes quantidades dessas moléculas e você terá um monitor superfino com uma qualidade assombrosa — sem nenhuma luz de fundo consumindo potência" [PCM03]. O resultado — monitores ultrafinos que podem ser enrolados ou dobrados, posicionados sobre uma superfície curva, ou adaptado de alguma outra forma a um ambiente específico.

RFIDs – Os identificadores por radiofrequência (*radio frequency identification*) trazem a computação aberta para uma base industrial e para o ramo de produtos de consumo. Qualquer coisa, de tubos de pasta de dente a motores de automóveis, pode ser identificada quando se movimenta através de uma esteira até seu destino final.

Web 2.0 – Amplo arranjo de serviços da Web que levará a uma integração ainda maior da Internet tanto no comércio quanto na computação pessoal.

Para mais discussões sobre tecnologias apresentadas em uma combinação especial de vídeo e material impresso, visite o site da Consumer Electronics Association, **www.ce.org/Press/CEA_Pubs/135.asp**.

31.4 DIREÇÕES DA TECNOLOGIA

"Mas para que serve?"

Engenheiro da divisão de computação avançada da IBM, 1968, comentando sobre um microchip

Sempre pensamos que a engenharia de software mudará mais rapidamente do que está ocorrendo. Uma nova tecnologia "excelente" (poderia ser um novo processo, um método especial ou uma ferramenta interessante) é introduzida, e os críticos sugerem que "tudo" mudará. Mas a engenharia de software é muito mais do que tecnologia — ela trata de pessoas e suas habilidades em comunicar necessidades e inovar para tornar aquelas necessidades uma realidade. Sempre que há pessoas envolvidas, as mudanças ocorrem lentamente de maneira impulsiva e irregular. Apenas quando um "ponto de virada" [Gla02] é atingido, que uma tecnologia se propaga através da comunidade de engenharia de software e uma ampla mudança realmente ocorre.

Nesta seção, examinaremos algumas tendências em processos, métodos e ferramentas que podem ter alguma influência sobre a engenharia de software durante a próxima década. Elas conduzirão a um ponto de virada? Temos de esperar para ver.

31.4.1 Tendências de processo

Podemos dizer que todas as tendências de negócios, organizacionais e culturais discutidas na Seção 31.3 reforçam a necessidade de processo. Mas as estruturas abordadas no Capítulo 30 fornecem um roteiro para o futuro? As estruturas de processo irão evoluir para buscar um melhor equilíbrio entre a disciplina e a criatividade? Os processos de software se adaptarão às diferentes necessidades dos interessados que solicitam o software, aqueles que o criam e aqueles que o utilizam? O software pode proporcionar um meio de reduzir o risco dos três componentes ao mesmo tempo?

Essas e outras questões permanecem pendentes. No entanto, algumas tendências começam a surgir. Conradi e Fuggetta [Con02] sugerem seis "teses sobre como aperfeiçoar e melhor aplicar as estruturas SPI". Eles iniciam a discussão com a seguinte afirmação:

> A meta de um promovedor de software é selecionar o melhor prestador de serviço objetivamente e racionalmente. A meta de uma empresa de software é sobreviver e crescer em um mercado competitivo. A meta de um usuário final é adquirir o produto que pode resolver o problema certo, na hora certa, a um preço aceitável. Não podemos esperar que uma mesma abordagem e consequente esforço de SPI possa acomodar todos esses diferentes pontos de vista.

Nos próximos parágrafos, adaptarei as teses propostas por Conradi e Fuggetta [Con02] para sugerir possíveis tendências de processo durante a próxima década.

Quais são as tendências de processo mais prováveis na próxima década?

1. *À medida que as estruturas SPI evoluem, darão ênfase a "estratégias que focalizam a orientação e inovação de produto"* [Con02]. Em um mundo de rápidas mudanças no desenvolvimento de software, estratégias SPI de longo prazo raramente sobrevivem em um ambiente de negócios dinâmicos. Há muitas mudanças ocorrendo rapidamente. Isso significa que um roteiro estável, passo a passo para SPI, pode precisar ser substituído por uma estrutura que enfatize os objetivos de curto prazo com uma orientação para produto. Se os requisitos para uma nova linha de produto baseado em software surgirem no decorrer de uma série de versões de produto incrementais (a ser fornecidas ao usuário final via Web), a organização de software pode precisar melhorar sua habilidade em administrar a mudança. Melhoras de processo associadas a gerenciamento de mudança devem ser coordenadas com o ciclo de versões do produto, para que melhore o gerenciamento de mudanças e ao mesmo tempo não seja confuso.
2. *Pelo fato de os engenheiros de software terem uma boa noção do ponto frágil do processo, as mudanças em geral deverão ser motivadas por suas necessidades e começar de baixo para cima.* Conradi e Fuggetta [Con02] sugerem que as atividades de SPI no futuro deverão "usar um placar simples e focalizado para começar, não uma ampla avaliação". Concentrando-se nos esforços de SPI de forma restrita e trabalhando de baixo para cima, os profissionais começarão logo a ver mudanças substanciais em como é conduzido o trabalho de engenharia de software.

3. *A tecnologia de processo automatizado de software (automated software process technology — SPT) se distanciará do gerenciamento global de processo (suporte com base ampla de todo o processo de software) para concentrar-se nos aspectos que podem se beneficiar mais da automação.* Ninguém é contra ferramentas e automação, mas em muitas situações, a SPT não atingiu seu objetivo (veja a Seção 31.2). Para maior eficácia, ela deverá focalizar atividades de apoio (Capítulo 2) — os elementos mais estáveis do processo de software.
4. *Haverá mais ênfase ao retorno sobre o investimento das atividades de SPI.* No Capítulo 30, você aprendeu que o retorno sobre o investimento (*return on investment* — ROI) pode ser definido como:

$$\text{ROI} = \left[\frac{\Sigma(\textit{benefícios}) - \Sigma(\textit{custos})}{\Sigma(\textit{custos})}\right] \times 100\%$$

   Até hoje, as organizações de software têm se empenhado para delinear claramente os "benefícios" de maneira quantitativa. Pode-se afirmar [Con02] que "precisamos, portanto, de um modelo padronizado de valor de mercado, como o empregado na Cocomo II (veja o Capítulo 26) para levar em conta as iniciativas de melhorias de software".
5. *À medida que o tempo passa, a comunidade do software pode vir a entender que a especialização em sociologia e antropologia pode ter tanto ou muito mais a ver com uma SPI bem-sucedida, quanto outras disciplinas, mais técnicas.* Além disso, a SPI muda a cultura organizacional, e a mudança cultural envolve indivíduos e grupos de profissionais. Conradi e Fuggetta [Con02] observam corretamente que "os desenvolvedores de software são trabalhadores de conhecimento. Tendem a responder negativamente a ordem de cima sobre como fazer o trabalho ou como mudar processos". Pode-se aprender muito examinando a sociologia de grupos para melhor entender maneiras eficazes de introduzir uma mudança.
6. *Novos modos de aprender podem facilitar a transição para um processo de software mais eficaz.* Nesse contexto, "aprender" implica aprender com sucessos e erros. Uma organização de software que coleta métricas (Capítulos 23 e 25) permite a si mesma entender como os elementos de um processo afetam a qualidade do produto final.

31.4.2 O grande desafio

Há uma tendência inegável — os sistemas baseados em software sem dúvida se tornarão maiores e mais complexos com o passar do tempo. É a engenharia desses sistemas grandes e complexos, independentemente da plataforma ou domínio de aplicação, que apresenta o "grande desafio" [Bro06] para os engenheiros de software. Manfred Broy [Bro06] afirma que os engenheiros de software podem enfrentar "o desafio assustador do desenvolvimento de sistemas complexos de software", criando novas abordagens para entender modelos de sistema e usando esses modelos como base para a construção de software de alta qualidade da nova geração.

Conforme a comunidade de engenharia de software desenvolve novas abordagens motivadas por modelo (assunto discutido mais adiante nesta seção) para a representação de requisitos de sistema e projeto, devem ser tratadas as seguintes características [Bro06]:

Que características de sistema os analistas e projetistas devem considerar para futuras aplicações?

- *Multifuncionalidade* – Depois que os dispositivos digitais evoluíram para sua segunda e terceira geração, começaram a apresentar um amplo conjunto de funções às vezes não relacionadas. O telefone móvel, considerado um dispositivo de comunicação, agora é usado para fotos, calendário, navegação e como reprodutor de música. Se as interfaces abertas vieram para ficar, esses dispositivos móveis serão usados para muitas outras funções nos próximos anos. Conforme observa Broy [Bro06], "os engenheiros devem descrever o contexto detalhado no qual as funções serão fornecidas e, mais importante, devem identificar as interações potencialmente perigosas entre diferentes características do sistema".
- *Reatividade e temporização* (*timeliness*) – os dispositivos digitais interagem cada vez mais com o mundo real e devem reagir aos estímulos externos no tempo. Eles devem estabe-

lecer interface com um amplo conjunto de sensores e devem responder em um intervalo de tempo apropriado para a tarefa em questão. Devem ser desenvolvidos novos métodos que (1) ajudem os engenheiros de software a prever a cadência das várias características reativas e (2) implementem características menos dependentes da máquina e mais portáteis.

- *Novos modos de interação de usuário* – o teclado e o mouse funcionam muito bem em um ambiente de PC, mas as tendências abertas para software significam que novos modos de interação devem ser modelados e implementados. Independentemente do fato de essas novas abordagens usarem interfaces multitoque, de reconhecimento de voz ou interfaces mentais diretas,[9] as novas gerações de software para dispositivos digitais devem modelar as novas interfaces homem-computador.
- *Arquiteturas complexas* – um automóvel de luxo tem mais de 2 mil funções controladas por software que reside em uma arquitetura de hardware complexa, incluindo múltiplas CPUs, estrutura de barramento sofisticada, atuadores, sensores e uma interface humana cada vez mais sofisticada, e muitos componentes relacionados com a segurança. Sistemas ainda mais complexos estão no horizonte próximo, apresentando desafios significativos para os projetistas de software.
- *Sistemas heterogêneos distribuídos* – os componentes de tempo real de qualquer sistema embutido moderno podem ser conectados através de um barramento interno, uma rede sem fio ou da Internet (ou as três coisas).
- *Criticidade* – o software tornou-se o componente pivô em todos os sistemas críticos nos negócios e em muitos sistemas em termos de segurança. Contudo, a comunidade de engenharia de software apenas começou a aplicar os princípios mais básicos de segurança de software.
- *Variabilidade de manutenção* – a vida do software em um dispositivo digital raramente dura além de 3 a 5 anos, mas os sistemas complexos de aviônica instalados em uma aeronave têm uma vida útil de pelo menos 20 anos. O software dos automóveis fica a meio-termo. Isso deverá ter um impacto sobre o projeto?

Broy [Bro06] afirma que essas e outras características do software podem ser administradas somente se a comunidade desenvolver uma filosofia de engenharia de software distribuída e colaborativa mais eficaz, melhores abordagens de requisitos de engenharia, uma abordagem mais robusta do desenvolvimento motivado por modelo e melhores ferramentas de software. Nas próximas seções, exploraremos rapidamente cada uma dessas áreas.

31.4.3 Desenvolvimento colaborativo

PONTO-CHAVE

A colaboração envolve a disseminação ao longo do tempo e um processo eficaz para comunicação e criação de projeto.

Parece muito óbvio, mas direi mesmo assim: *a engenharia de software é uma tecnologia de informação.* Desde o princípio de qualquer projeto de software, todos os interessados devem compartilhar informações — sobre metas e objetivos básicos, sobre requisitos básicos de sistema, sobre problemas de projeto de arquitetura, sobre quase todos os aspectos do software a ser criado.

Hoje, os engenheiros de software colaboram em diferentes fusos horários e diferentes países, e todos eles devem compartilhar informações. O mesmo vale para projetos abertos nos quais centenas ou milhares de desenvolvedores de software trabalham para criar uma aplicação aberta. Uma vez mais, as informações devem ser disseminadas para que possa ocorrer a colaboração aberta.

O desafio para a próxima década é desenvolver métodos e ferramentas que facilitem essa colaboração. Hoje, continuamos nos esforçando para facilitar a colaboração. Eugene Kim [Kim04] comenta:

9 Uma breve discussão sobre as interfaces mentais diretas pode ser encontrada em **http://en.wikipedia.org/wiki/Brain computer_interface**, e um exemplo comercial é descrito no site **http://au.gamespot.com/news/6166959.html**.

> Considere uma tarefa colaborativa básica: compartilhamento de documento. Muitos aplicativos (tanto comerciais quanto abertos) prometem resolver o problema do compartilhamento de documento e, ainda mais, o método predominante para compartilhar arquivos é enviá-los por e-mail de um lado para outro. Esse é o equivalente computacional de transferência eletrônica de informações. Se as ferramentas que pretendem resolver esse problema são boas, por que não utilizá-las?
>
> Vemos problemas similares em outras áreas básicas. Posso entrar em qualquer reunião em qualquer lugar do mundo com uma folha de papel na mão, e posso estar seguro de que todas as pessoas poderão ler, fazer marcações, passar adiante e arquivar. Não posso dizer o mesmo dos documentos eletrônicos. Não posso fazer anotações em uma página da Web ou usar o mesmo sistema de arquivamento para meus e-mails e meus documentos do Word, pelo menos não de uma maneira que seja interoperável com aplicações em minha própria máquina e nas máquinas de outras pessoas. Por que não?
>
> ...Para causar um impacto real no espaço colaborativo, as ferramentas precisam não apenas ser boas, mas devem ser interoperáveis.

Mas a falta de ferramentas colaborativas abrangentes é apenas uma parte do desafio enfrentado por aqueles que devem desenvolver software de forma colaborativa.

Hoje, uma porcentagem significativa[10] dos projetos de TI são terceirizados internacionalmente, e o número crescerá substancialmente durante a próxima década. Não é surpresa que Bhat e seus colegas [Bha06] afirmam que a engenharia de requisitos é a atividade crucial em um projeto de terceirização. Eles identificam um conjunto de fatores de sucesso que levam a esforços colaborativos bem-sucedidos:

- *Metas compartilhadas* – as metas de projeto devem ser claramente enunciadas, e todos os interessados devem entendê-las e concordar com seu intento.
- *Cultura compartilhada* – diferenças culturais devem ser claramente definidas, e deve ser desenvolvida uma abordagem educacional (que ajudará a moderar essas diferenças) e uma abordagem de comunicação (que facilitará a transferência de conhecimentos).
- *Processo compartilhado* – de alguma maneira, o processo serve de esqueleto de um projeto colaborativo, proporcionando um meio uniforme para avaliar o progresso e a direção e introduzir uma "linguagem" técnica comum para todos os membros da equipe.
- *Responsabilidade compartilhada* – todo membro de equipe deve reconhecer a importância da engenharia de requisitos e trabalhar para proporcionar a melhor definição possível do sistema.

Quando combinados, esses fatores de sucesso levam à "confiança" — uma equipe global que pode confiar em grupos diversos para executar a tarefa que lhes foi atribuída.

31.4.4 Engenharia de requisitos

As ações básicas de engenharia de requisitos (*requirements engineering* — RE) — evocação, elaboração, negociação, especificação e validação — foram apresentadas nos Capítulos 5 a 7. O sucesso ou fracasso dessas ações têm forte influência sobre o sucesso ou fracasso do processo inteiro de engenharia de software. Contudo, a RE tem sido comparada com "tentar colocar uma braçadeira de mangueira" [Gon04]. Conforme já notamos em várias partes deste livro, os requisitos de software tendem a continuar mudando e, com o surgimento dos sistemas abertos, requisitos emergentes (e mudanças quase contínuas) podem se tornar coisa normal.

Hoje, muitas abordagens "informais" de RE começam com a criação de cenários de usuário (por exemplo, casos de uso). Abordagens mais informais criam um ou mais modelos de requisitos e os utilizam como base para o projeto. Métodos formais permitem a um engenheiro de software representar requisitos usando uma notação matemática que pode ser verificada. Tudo pode funcionar razoavelmente bem quando os requisitos são estáveis, mas não resolve imediatamente o problema dos requisitos dinâmicos ou emergentes.

---

10 Aproximadamente 20% de um orçamento típico de TI para grandes empresas é dedicado atualmente à terceirização, e essa porcentagem está aumentando a cada ano. (Fonte: www.logicacmg.com/page/400002849.)

PONTO-CHAVE

"Novos subprocessos de RE incluem: (1) melhor aquisição de conhecimento, (2) iteração ainda melhor, e (3) ferramentas mais eficazes de comunicação e coordenação."

Há uma série de direções distintas de pesquisa de engenharia de requisitos, incluindo processamento de linguagem natural com base em descrições textuais traduzidas para representações mais estruturadas , uma maior confiabilidade em bases de dados para estruturação e entendimento de requisitos de software, o uso de padrões de RE para descrever problemas típicos e soluções quando são executadas as tarefas de engenharia de requisitos e engenharia de requisitos orientada para metas. No entanto, no nível industrial, ações de RE permanecem relativamente informais e surpreendentemente básicas. Para melhorar a maneira como os requisitos são definidos, a comunidade de engenharia de software provavelmente implementará três subprocessos distintos enquanto a RE é executada [Gli07]: (1) melhora na aquisição de conhecimentos e compartilhamento de conhecimentos que possibilita entendimento mais completo das restrições do domínio de aplicação e necessidades dos interessados, (2) maior ênfase sobre a iteração quando os requisitos são definidos, e (3) ferramentas mais eficazes de comunicação e coordenação que permitem a todos os interessados colaborar eficazmente.

Os subprocessos de RE descritos no parágrafo anterior só terão sucesso se forem integrados adequadamente em uma abordagem evolutiva da engenharia de software. Como a solução de problemas baseados em padrões e as soluções baseadas em componentes começam a conquistar muitos domínios de aplicação, a RE deve conciliar o desejo de agilidade (entrega incremental rápida) e os requisitos emergentes inerentes resultantes. A natureza concorrente de muitos modelos de engenharia de software significa que a RE será integrada a atividades de projeto e construção. Como consequência, a noção de uma "especificação de software" estática está começando a desaparecer, para ser substituída por "requisitos motivados por valor" [Som05] derivados quando os interessados respondem às características e funções fornecidas nos primeiros incrementos de software.

31.4.5 Desenvolvimento de software controlado por modelo

PONTO-CHAVE

Abordagens controladas por modelo tratam de um desafio contínuo para todos os desenvolvedores de software — como representar o software em um nível mais alto de abstração do que o código.

Os engenheiros de software lidam com abstração em quase todas as etapas no processo de engenharia de software. Quando o projeto começa, as abstrações em nível de arquitetura e de componente são representadas e julgadas. Elas devem então ser traduzidas para uma representação de linguagem de programação que transforma o projeto (uma abstração de nível relativamente alto) em um sistema operável com um ambiente de computação específico (baixo nível de abstração). O *desenvolvimento de software controlado por modelo*[11] acopla linguagens de modelagem específicas de domínio com mecanismo de transformação e geradores para facilitar a representação da abstração em altos níveis e, então, a transforma em níveis mais baixos [Sch06].

Linguagens de modelagem específicas de domínio (domain-specific modeling languages – DSMLs) representam "estrutura de aplicativo, comportamento e requisitos dentro de domínios de aplicação particulares" e são descritas com metamodelos que "definem as relações entre conceitos no domínio e especificam precisamente as semânticas fundamentais e as restrições associadas a esses conceitos de domínio" [Sch06]. A principal diferença entre uma DSML e uma linguagem de modelagem de uso geral como a UML (Apêndice 1) é que a DSML está sintonizada com conceitos de projeto inerentes ao domínio de aplicação e pode, portanto, representar relações e restrições entre elementos de projeto de modo eficiente.

31.4.6 Projeto pós-moderno

Em um artigo interessante sobre projeto de software na "era pós-moderna", Philippe Kruchten [Kru05] faz a seguinte observação:

> A ciência da computação não atingiu ainda o contexto geral que consiga explicar sua totalidade — ainda não descobrimos as leis fundamentais do software que teriam o mesmo papel das leis fundamentais da física em outras disciplinas de engenharia. Ainda sentimos o gosto amargo do estouro da bolha da Internet e do pesadelo Y2K. Assim, nesta era pós-moderna, em que parece que tudo importa um pouco, embora quase nada importe, quais são os próximos rumos para o projeto de software?

11 O termo *model-driven engineering* (MDE) também é usado.

Parte de qualquer tentativa de entender tendências em projeto de software é estabelecer fronteiras para o projeto. Onde termina a engenharia de requisitos e começa o projeto? Onde termina o projeto e começa a geração de código? A resposta a essas questões não é tão fácil como pode parecer. Embora o modelo de requisitos devesse concentrar-se "no que", não em "como", todo analista faz um pouco de projeto e quase todos os projetistas fazem um pouco de análise. De forma semelhante, conforme o projeto de componentes de software se aproxima do detalhe algorítmico, um projetista começa a representar o componente em um nível de abstração que se aproxima do código.

O projeto pós-moderno continuará a enfatizar a importância da arquitetura de software (Capítulo 9). O projetista deve definir um problema arquitetônico, tomar uma decisão relacionada com esse problema e então definir claramente as suposições, restrições e implicações que a decisão impõe ao software como um todo. Mas há uma estrutura na qual os problemas podem ser descritos e a arquitetura pode ser definida? O desenvolvimento de software orientado a aspecto (Capítulo 2) ou desenvolvimento de software controlado por modelo (Seção 31.4.4) podem se tornar abordagens de projeto importantes nos próximos anos, mas ainda é muito cedo para dizer. Pode ocorrer de grandes avanços em desenvolvimento baseado em componente (Capítulo 10) levar a uma filosofia de projeto que enfatiza a montagem de componentes existentes. Se o passado é um prólogo, é muito provável que muitos métodos "novos" de projeto surgirão, mas poucos percorrerão a curva da excelência (Figura 31.2) muito além do "canal da desilusão".

31.4.7 Desenvolvimento baseado em teste

Os requisitos controlam o projeto, e o projeto estabelece uma fundação para a construção. Essa simples realidade de engenharia de software funciona razoavelmente bem e é essencial quando se cria uma arquitetura de software. No entanto, uma mudança sutil pode trazer benefícios significativos quando são considerados projeto e construção em nível de componente.

PONTO-CHAVE

"TDD é uma tendência que enfatiza o projeto de casos de teste *antes* da criação de código-fonte".

Em *desenvolvimento baseado em teste* (*test-driven development* — TDD), requisitos para um componente de software servem de base para a criação de uma série de casos de teste que exercitam a interface e tentam encontrar erros nas estruturas de dados e funcionalidade fornecida pelo componente. A TDD não é realmente uma nova tecnologia, mas sim uma tendência que enfatiza o projeto de casos de teste *antes* da criação do código-fonte.[12]

O processo TDD segue o fluxo procedural simples ilustrado na Figura 31.3. Antes de ser criado o primeiro segmento de código, um engenheiro de software cria um teste para exercitar o código (tentando fazer o código falhar). É então escrito o código para satisfazer ao teste. Se passar, um novo teste é criado para o próximo segmento de código a ser desenvolvido. O processo continua até que o componente esteja completamente codificado e todos os testes executam sem erro. No entanto, se algum teste conseguir descobrir um erro, o código existente é refeito (corrigido) e todos os testes criados até aquele ponto são executados novamente. Esse fluxo iterativo continua até que não haja mais teste a ser criado, implicando que o componente satisfaz a todos os requisitos definidos para ele.

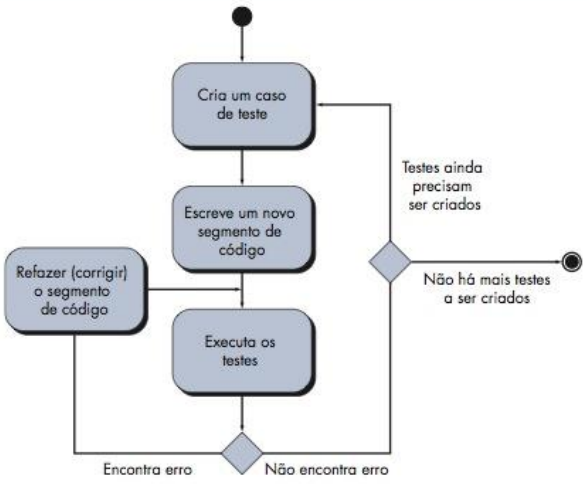
Durante o TDD, o código é desenvolvido em incrementos muito pequenos (uma subfunção de cada vez), e nenhum código é escrito enquanto não houver um teste para experimentá-lo. Você deve observar que cada iteração resulta em um ou mais novos testes que são acrescentados a um conjunto de testes de regressão que roda com cada mudança. Isso é feito para garantir que o novo código não tenha gerado efeitos colaterais que causam erros no código anterior.

Em TDD, testes controlam o projeto detalhado de componente e o código-fonte resultante. Os resultados desses testes causam modificações imediatas no projeto do componente (via código), e o que é mais importante, o componente resultante (quando completado) foi verificado em condição isolada (*stand-alone*). Se você tiver mais interesse em TDD, veja [Bec04b] ou [Ast04].

---

12 Recorde-se que a *Extreme Programming* (Capítulo 3) enfatiza essa abordagem como parte de seu modelo de processo ágil.

FIGURA 31.3
O fluxo de desenvolvimento controlado por teste



31.5 TENDÊNCIAS RELACIONADAS COM FERRAMENTAS

Centenas de ferramentas de engenharia de software de nível industrial são introduzidas a cada ano. A maioria é fornecida por empresas que afirmam que aquela ferramenta irá melhorar o gerenciamento de projeto, ou a análise de requisitos, ou a modelagem do projeto, ou geração de código, ou o teste, ou gerenciamento de mudanças, ou qualquer uma das muitas atividades de engenharia de software, ações e tarefas discutidas neste livro. Outras foram desenvolvidas como ofertas de código aberto. A maioria das ferramentas de código aberto concentra-se nas atividades de "programação" com ênfase específica na construção (particularmente a geração de código). Há ainda outras que resultam de esforços de pesquisas em laboratórios de universidades ou do governo. Embora tenham um atrativo em aplicações bastante limitadas, grande parte não está pronta para a aplicação mais ampla da indústria.

Em nível industrial, os pacotes mais abrangentes formam os *ambientes de engenharia de software* (*software engineering environments* — SEE)[13] que integram uma coleção de ferramentas individuais ao redor de uma base de dados central (repositório). Quando considerado um todo, um SEE integra informações por meio do processo de software e auxilia na colaboração necessária para muitos sistemas grandes e complexos baseados em software. Mas os ambientes atuais não são facilmente extensíveis (é difícil integrar uma ferramenta COTS que não faz parte do pacote) e tendem a ser de uso geral (não são específicos de domínio de aplicação). Há também um retardo de tempo substancial entre a introdução de novas soluções de tecnologia (por exemplo, desenvolvimento de software controlado por modelo) e a disponibilidade de SEEs viáveis que suportam a nova tecnologia.

As tendências futuras em ferramentas de software seguirão dois caminhos distintos — um *caminho focalizado no homem* que responde a algumas das "tendências leves" discutidas na Seção 31.3, e o caminho centrado na tecnologia que trata de novas tecnologias (Seção 31.4) à medida que são introduzidas e adotadas. Nas próximas seções, examinaremos rapidamente cada um desses caminhos.

13 É usado também o termo *ambiente de desenvolvimento integrado* (*integrated development environment* — IDE).

31.5.1 Ferramentas que respondem a tendências leves

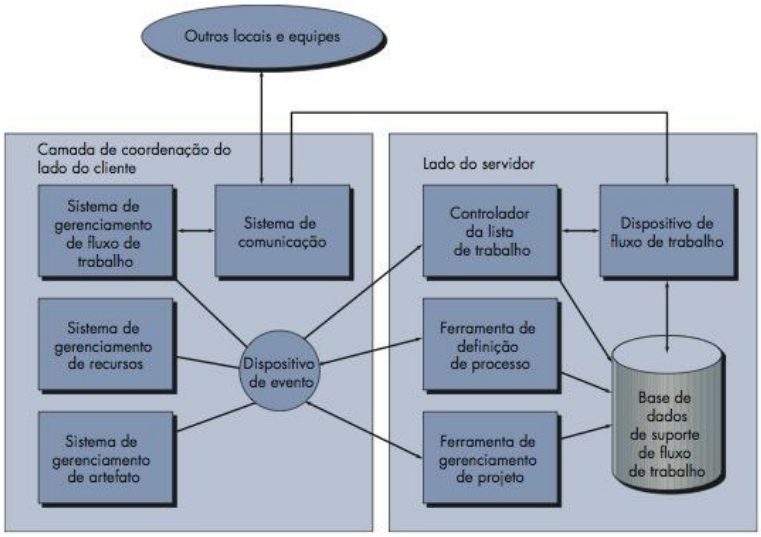
As tendências leves discutidas na Seção 31.3 — a necessidade de gerenciar a complexidade, acomodar requisitos emergentes, estabelecer modelos de processo que aceitam mudanças, coordenar equipes globais com um *mix* de talentos que varia, entre outras coisas — sugere uma nova era em que suporte de ferramentas para colaboração de interessados se tornará tão importante quanto o suporte de ferramentas para tecnologia. Mas que tipo de conjunto de ferramentas suporta essas tendências?

Um exemplo de pesquisa nessa área é o GENESIS — um ambiente generalizado, de código aberto, criado para suportar trabalho de engenharia de software colaborativo [Ave04]. O ambiente GENESIS pode ou não ter um uso difundido, mas seus elementos básicos são representativos da direção dos SEEs colaborativos que irão evoluir para suportar as tendências leves descritas neste capítulo.

Um SEE colaborativo "suporta cooperação e comunicação entre engenheiros de software pertencentes a equipes de desenvolvimento distribuído envolvidas em modelar, controlar e medir processos de desenvolvimento de software e de manutenção. E ainda mais, inclui uma função de gerenciamento de artefato que armazena e controla artefatos de software (produtos acabados) produzidos por diferentes equipes no decorrer de seu trabalho" [Bol02].

A Figura 31.4 ilustra uma arquitetura para um SEE colaborativo. A arquitetura, baseada no ambiente GENESIS [Ave04], é construída com subsistemas integrados dentro de um cliente Web comum e é complementada por componentes baseados em servidor que proporcionam suporte para todos os clientes. Cada organização de desenvolvimento tem seus próprios subsistemas no lado do cliente que se comunica com outros clientes. De acordo com a Figura 31.4, um *subsistema de gerenciamento de recursos* (*resource management subsystem*) controla a alocação de recursos humanos para diferentes projetos ou subprojetos; um *sistema de gerenciamento de artefatos* (*work product management system*) é "responsável pela criação, modificação, exclusão",

FIGURA 31.4
Arquiterura SEE colaborativa
Fonte: **Adaptado de [Ave04]**



indexação, pesquisa e armazenamento de todos os artefatos de engenharia de software [Ave04]; um *subsistema de gerenciamento de fluxo de trabalho* (*workflow management subsystem*) coordena a definição, a ocorrência (*instantiation*) e implementação das atividades, ações e tarefas de processo de software; um dispositivo de evento "coleta eventos" que ocorrem durante o processo de software (por exemplo, uma revisão bem-sucedida de um artefato, o término de um teste de unidade de um componente) e notifica os outros; um *sistema de comunicações* (*communication system*) suporta comunicação síncrona e assíncrona entre as equipes distribuídas.

No lado do servidor, quatro componentes compartilham uma base de dados de suporte de fluxo de trabalho. Os componentes implementam as seguintes funções:

- *Definição de processo* – ferramenta que permite à equipe definir novas atividades de processo, ações ou tarefas e define as regras que governam como esses elementos de processo interagem uns com os outros e os produtos acabados que eles geram.
- *Gerenciamento de projeto* – conjunto de ferramentas que permite à equipe criar um plano de projeto e coordenar o plano com outras equipes ou projetos.
- *Dispositivo de fluxo de trabalho* (*workflow engine*) – "interage com o dispositivo de evento para propagar eventos que são relevantes para a execução de processos cooperativos executados em outros locais" [Ave04].
- *Manipulador de lista de trabalhos* – interage com a base de dados do lado do servidor para fornecer ao engenheiro de software informações sobre a tarefa que está em execução no momento ou qualquer tarefa futura que seja derivada do trabalho que esteja em execução no momento.

Embora a arquitetura de um SEE colaborativo possa variar consideravelmente em relação àquele discutido nesta seção, os elementos funcionais básicos (sistemas de gerenciamento e componentes) parecerão atingir o nível de coordenação necessário para um projeto distribuído de engenharia de software.

31.5.2 Ferramentas que lidam com as tendências tecnológicas

A agilidade na engenharia de software (Capítulo 3) é obtida quando os interessados trabalham em equipe. Portanto, a tendência para os SEEs colaborativos (Seção 31.5.1) produzirá benefícios mesmo quando o software é desenvolvido localmente. Mas qual das ferramentas de tecnologia complementa o sistema e componentes que fortalecem uma melhor colaboração?

Uma das tendências dominantes é a criação de um conjunto de ferramentas que suporta desenvolvimento controlado por modelo (Seção 31.4.4) com ênfase no projeto controlado por arquitetura. Oren Novotny [Nov04] sugere que o modelo e não o código-fonte se torne o foco da engenharia de software:

> Modelos independentes de plataforma são criados em UML e então passam por vários níveis de transformação para eventualmente se transformarem em código-fonte para uma plataforma específica. É razoável concluir que o modelo, não o arquivo, deverá se tornar a nova unidade de saída. Um modelo tem muitas visões diferentes em diferentes níveis de abstração. No nível mais alto, componentes independentes de plataforma podem ser especificados em análise; no nível mais baixo há uma implementação específica de plataforma que se reduz a um conjunto de classes no código.

Novotny afirma que uma nova geração de ferramentas funcionará em conjunto com um repositório para criar modelos em todos os níveis necessários de abstração, estabelecer relações entre os vários modelos, transformar modelos de um nível de abstração para outro (por exemplo, transformar um modelo de projeto em código-fonte), gerenciar alterações e versões e coordenar ações de controle e garantia de qualidade nos modelos de software.

Além dos ambientes completos de engenharia de software, ferramentas de solução pontual que resolvem tudo, desde reunião de requisitos até refatoração de projeto/código e teste, con-

tinuarão a evoluir e se tornarão mais capazes em funcionalidade. Em algumas situações, ferramentas de modelagem e teste voltadas para um domínio de aplicação específico proporcionarão um melhor benefício quando comparadas com seus equivalentes genéricos.

31.6 Resumo

As tendências que têm um efeito sobre a tecnologia de engenharia de software muitas vezes originam-se de cenários de negócios, organizacionais, mercado e cultural. Essas "tendências leves" podem influir na direção da pesquisa e da tecnologia derivada em consequência da pesquisa.

Quando uma nova tecnologia é introduzida, ela passa por um ciclo de vida que nem sempre leva a uma aceitação ampla, apesar de as expectativas originais serem elevadas. O grau segundo o qual qualquer tecnologia de engenharia de software ganha uma aceitação ampla está ligado a sua habilidade para resolver os problemas apresentados pelas tendências tanto leves (*soft*) quanto pesadas (*hard*).

Tendências leves — a necessidade cada vez maior de conectividade e colaboração, projetos globais, transferência de conhecimento, o impacto das economias emergentes e a influência da própria cultura humana levam a uma série de desafios que abrangem desde o gerenciamento de complexidade e requisitos emergentes até a manipulação de um *mix* de talentos sempre em mudanças entre equipes de software geograficamente dispersas.

Tendências pesadas — o ritmo sempre acelerado da mudança da tecnologia — surgem do âmbito das tendências leves e afetam a estrutura do software e o escopo dos processos e a maneira pela qual uma estrutura de processo é caracterizada. Desenvolvimento colaborativo, novas formas de engenharia de requisitos, desenvolvimento baseado em modelo e controlado por teste, e projeto pós-moderno mudarão o cenário de métodos. Os ambientes de ferramentas responderão a uma necessidade cada vez maior de comunicação e colaboração e ao mesmo tempo integram soluções pontuais específicas de domínio que podem mudar a natureza atual das tarefas de engenharia de software.

Problemas e Pontos a Ponderar

31.1. Obtenha uma cópia do *best-seller The tipping point*, de Malcolm Gladwell (disponível via Google Book Search) e discuta como suas teorias se aplicam à adoção das novas tecnologias de engenharia de software.

31.2. Por que o software aberto apresenta um desafio às abordagens convencionais de engenharia de software?

31.3. Reveja o *ciclo da excelência para tecnologias emergentes* do Gartner Group. Selecione um produto de tecnologia bem conhecido e apresente um breve histórico que ilustre como ele se comportou ao longo da curva. Selecione outro produto de tecnologia bem conhecida que não seguiu o caminho sugerido pela curva da excelência.

31.4. O que é uma "tendência leve"?

31.5. Você enfrenta um problema extremamente complexo que vai requerer uma solução demorada. Como trata a complexidade desse problema e como propõe uma solução?

31.6. O que são "requisitos emergentes" e por que eles representam um desafio para os engenheiros de software?

31.7. Selecione um trabalho de desenvolvimento de código aberto (que não o Linux) e apresente um breve histórico de sua evolução e sucesso relativo.

31.8. Descreva como o processo de software mudará durante a próxima década.

31.9. Você trabalha em Los Angeles e participa de uma equipe global de engenharia de software. Você e colegas em Londres, Mumbai, Hong Kong e Sydney devem editar uma especifi-

cação de requisitos de 245 páginas para um grande sistema. A primeira edição deve ser feita em três dias. Descreva o conjunto ideal de ferramentas on-line que lhe possibilitaria colaborar eficazmente.

31.10. Descreva o desenvolvimento de software controlado por modelo. Faça o mesmo para o desenvolvimento controlado por teste.

Leituras e Fontes de Informação Complementares

Livros que discutem o caminho para software e computação abrangem uma grande variedade de assuntos técnicos, científicos, econômicos, políticos e sociais. Kurweil (*The Singularity Is Near,* Penguin Books, 2005) apresenta uma visão atraente de um mundo que mudará de formas muito profundas nos meados deste século. Sterling (*Tomorrow Now,* Random House, 2002) lembra-nos de que o progresso real raramente é ordenado e eficiente. Teich (*Technology and the Future,* Wadworth, 2002) apresenta ensaios sobre o impacto social da tecnologia e como a cultura em mudança dá forma à tecnologia. Naisbitt, Philips e Naisbitt (*High Tech/High Touch,* Nicholas Brealey, 2001) observam que muitos de nós se tornaram "intoxicados" com alta tecnologia e que a "grande ironia da era de alta tecnologia é que nos tornamos escravos de dispositivos que se destinavam a nos dar a liberdade". Zey (*The Future Factor,* McGraw-Hill, 2000) discute cinco forças que definirão o destino humano durante este século. A obra de Negroponte *(Being Digital,* Alfred A. Knopf, 1995) foi um *best-seller* nos meados da década de 1990 e continua a fornecer uma visão esclarecedora da computação e seu impacto geral.

À medida que o software se torna parte do cenário de quase todas as facetas de nossas vidas, a "cyber ética" evoluiu como um importante tópico de discussão. Livros de Spinello (*Cyberethics: Morality and Law in Cyberspace*, Jones & Bartlett Publishers, 2002), Halbert e Ingulli (*Cyberethics,* South-Western College Publishers, 2001) e Baird e seus colegas (*Cyberethics: Social and Moral Issues in the Computer Age*, Prometheus Books, 2000) consideram o tópico em detalhe. O governo dos Estados Unidos publicou um volumoso relatório em CD-ROM (*21st Century Guide to Cybercrime,* Progressive Management, 2003) que considera todos os aspectos do crime no computador, problemas de propriedade intelectual e o National Infrastructure Protection Center (NIPC).

Uma grande variedade de fontes de informação sobre os rumos futuros das tecnologias relacionadas ao software e engenharia de software está disponível na Internet. Uma lista atualizada das referências da Web relevantes sobre as futuras tendências na engenharia de software pode ser encontrada no site **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**.