



Introdução à engenharia de software

O objetivo nesta parte do livro é fornecer uma introdução geral à engenharia de software. Apresento conceitos importantes, como processos de software e métodos ágeis, e descrevo as atividades essenciais para o desenvolvimento de software, desde a especificação inicial do software até a evolução do sistema. Nesta primeira parte, os capítulos foram concebidos para darem suporte a um curso de engenharia de software de um semestre.

O Capítulo 1 é uma apresentação geral que apresenta a engenharia de software profissional e define alguns conceitos da área. Também escrevi uma breve discussão sobre as questões éticas na engenharia de software. Acho que é importante os engenheiros de software pensarem sobre as implicações mais amplas do seu trabalho. Este capítulo também apresenta três estudos de caso, um sistema de gerenciamento de registros de pacientes em tratamento para problemas de saúde mental, um sistema de controle para uma bomba de insulina portátil e um sistema meteorológico no deserto.

Os capítulos 2 e 3 abrangem os processos de engenharia de software e desenvolvimento ágil. No Capítulo 2, apresento modelos genéricos de processos de software, como o modelo em cascata, e discuto as atividades básicas que são parte desses processos. O Capítulo 3 suplementa esse, com uma discussão sobre métodos ágeis de desenvolvimento de engenharia de software. Uso ainda a Extreme Programming como exemplo de método ágil, mas neste capítulo também faço uma leve introdução ao Scrum.

O restante dos capítulos desta parte são descrições detalhadas das atividades de processo de software, as quais serão introduzidas no Capítulo 2. O Capítulo 4 aborda o tema crítico de importância de engenharia de requisitos, em que são definidos os requisitos que especificam o que um sistema deve fazer. O Capítulo 5 apresenta a modelagem de sistemas usando a UML centrada no uso de diagramas de caso de uso, diagramas de classe, diagramas de sequência e diagramas de estado para a modelagem de um sistema de software. O Capítulo 6 apresenta os projetos de arquitetura, em que se discute a importância da arquitetura e do uso de padrões em projetos de software.

O Capítulo 7 apresenta o projeto orientado a objetos e o uso de padrões de projeto. Apresento também importantes questões de implementação — reuso, gerenciamento de configuração e desenvolvimento *host-target*, além de discutir o desenvolvimento *open source*. O Capítulo 8 se concentra nos testes de software desde os testes unitários durante o desenvolvimento do sistema até o teste de releases de software. Discuto ainda o uso do desenvolvimento dirigido a testes — de uma perspectiva pioneira em métodos ágeis, mas de grande aplicabilidade. Finalmente, o Capítulo 9 apresenta uma visão geral dos assuntos relacionados à evolução de software. Abrange os processos de evolução e manutenção de software, e gerenciamento de sistemas legados.



CAPÍTULO

1

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

Introdução

Objetivos

Os objetivos deste capítulo são fazer uma introdução à engenharia de software e fornecer uma base para a compreensão do restante do livro. Depois de ler este capítulo, você:

- entenderá o que é engenharia de software e por que ela é importante;
- entenderá que o desenvolvimento de diferentes tipos de sistemas de software pode requerer diferentes técnicas de engenharia de software;
- entenderá algumas questões éticas e profissionais importantes para engenheiros de software;
- terá conhecimento de três sistemas de tipos diferentes que serão usados como exemplos neste livro.

- Conteúdo**
- 1.1 Desenvolvimento profissional de software
 - 1.2 Ética na engenharia de software
 - 1.3 Estudos de caso

O mundo moderno não poderia existir sem o software. Infraestruturas e serviços nacionais são controlados por sistemas computacionais, e a maioria dos produtos elétricos inclui um computador e um software que o controla. A manufatura e a distribuição industriais são totalmente informatizadas, assim como o sistema financeiro. A área de entretenimento, incluindo a indústria da música, jogos de computador, cinema e televisão, faz uso intensivo de software. Portanto, a engenharia de software é essencial para o funcionamento de sociedades nacionais e internacionais.

Os sistemas de software são abstratos e intangíveis. Eles não são restringidos pelas propriedades dos materiais, nem governados pelas leis da física ou pelos processos de manufatura. Isso simplifica a engenharia de software, porque não há limites naturais para o potencial do software. No entanto, devido a essa falta de restrições físicas, os sistemas de software podem se tornar extremamente complexos de modo muito rápido, difíceis de entender e caros para alterar.

Existem vários tipos de sistemas de software, desde os simples sistemas embutidos até os sistemas de informações complexos, de alcance mundial. Não faz sentido procurar notações, métodos ou técnicas universais para a engenharia de software, porque diferentes tipos de software exigem abordagens diferentes. Desenvolver um sistema de informações corporativo é totalmente diferente de desenvolver um controlador para um instrumento científico. Nenhum desses sistemas tem muito em comum com um jogo computacional com gráficos intensos. Todas essas aplicações precisam de engenharia de software, embora não necessitem das mesmas técnicas.

Ainda existem muitos relatos e projetos de software que deram errado e resultaram em 'falhas de software'. A engenharia de software é criticada por ser inadequada para o desenvolvimento moderno de software. No entanto, no meu ponto de vista, muitas dessas falhas são consequência de dois fatores:

1. *Aumento de demanda.* Conforme novas técnicas de engenharia de software nos auxiliam a construir sistemas maiores e mais complexos, as demandas mudam. Os sistemas têm de ser construídos e entregues mais rapidamente;

sistemas maiores e até mais complexos são requeridos; sistemas devem ter novas capacidades que antes eram consideradas impossíveis. Como os métodos de engenharia de software existentes não conseguem lidar com isso, novas técnicas de engenharia de software precisam ser desenvolvidas para atender a essas novas demandas.

2. *Expectativas baixas.* É relativamente fácil escrever programas computacionais sem usar técnicas e métodos de engenharia de software. Muitas empresas foram forçadas a desenvolver softwares à medida que seus produtos e serviços evoluíram. Elas não usam métodos de engenharia de software no dia a dia. Consequentemente, seu software é frequentemente mais caro e menos confiável do que deveria ser. Precisamos de educação e treinamento em engenharia de software para solucionar esses problemas.

Engenheiros de software têm o direito de se orgulhar de suas conquistas. É claro que ainda temos problemas em desenvolver softwares complexos, mas, sem a engenharia de software, não teríamos explorado o espaço, não teríamos a Internet ou as telecomunicações modernas. Todas as formas de viagem seriam mais perigosas e caras. A engenharia de software contribuiu muito, e tenho certeza de que suas contribuições no século XXI serão maiores ainda.

1.1

Desenvolvimento profissional de software

Inúmeras pessoas escrevem programas. Pessoas envolvidas com negócios escrevem programas em planilhas para simplificar seu trabalho; cientistas e engenheiros escrevem programas para processar seus dados experimentais; e há aqueles que escrevem programas como *hobby*, para seu próprio interesse e diversão. No entanto, a maior parte do desenvolvimento de software é uma atividade **profissional**, em que o software é desenvolvido para um propósito específico de negócio, para inclusão em outros dispositivos ou como produtos de software como sistemas de informação, sistemas CAD etc. O software profissional, o que é usado por alguém além do seu desenvolvedor, é normalmente criado por equipes, em vez de indivíduos. Ele é mantido e alterado durante sua vida.

A engenharia de software tem por objetivo apoiar o desenvolvimento profissional de software, mais do que a programação individual. Ela inclui técnicas que apoiam especificação, projeto e evolução de programas, que normalmente não são relevantes para o desenvolvimento de software pessoal. Para ajudá-lo a ter uma visão geral sobre o que trata a engenharia de software, listei algumas perguntas comuns na Tabela 1.1.

Muitas pessoas pensam que software é simplesmente outra palavra para programas de computador. No entanto, quando falamos de engenharia de software, não se trata apenas do programa em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse programa operar corretamente. Um sistema de software desenvolvido profissionalmente é, com frequência, mais do que apenas um programa; ele normalmente consiste em uma série de programas separados e arquivos de configuração que são usados para configurar esses programas. Isso pode incluir documentação do sistema, que descreve a sua estrutura; documentação do usuário, que explica como usar o sistema; e sites, para usuários baixarem a informação recente do produto.

Essa é uma diferença importante entre desenvolvimento de software profissional e amador. Se você está escrevendo um programa para si mesmo, que ninguém mais usará, você não precisa se preocupar em escrever o manual do programa, documentar sua arquitetura etc. No entanto, se você está escrevendo um software que outras pessoas usarão e no qual outros engenheiros farão alterações, então você provavelmente deve fornecer informação adicional, assim como o código do programa.

Engenheiros de software se preocupam em desenvolver produtos de software (ou seja, software que pode ser vendido para um cliente). Existem dois tipos de produtos de software:

1. *Produtos genéricos.* Existem sistemas *stand-alone*, produzidos por uma organização de desenvolvimento e vendidos no mercado para qualquer cliente que esteja interessado em comprá-los. Exemplos desse tipo de produto incluem software para PCs, como ferramentas de banco de dados, processadores de texto, pacotes gráficos e gerenciamento de projetos. Também incluem as chamadas aplicações verticais projetadas para um propósito específico, como sistemas de informação de bibliotecas, sistemas de contabilidade ou sistemas de manutenção de registros odontológicos.
2. *Produtos sob encomenda.* Estes são os sistemas encomendados por um cliente em particular. Uma empresa de software desenvolve o software especialmente para esse cliente. Exemplos desse tipo de software são sistemas de controle de dispositivos eletrônicos, sistemas escritos para apoiar um processo de negócio específico e sistemas de controle de tráfego aéreo.

Tabela 1.1 Perguntas frequentes sobre software

Pergunta	Resposta
O que é software?	Softwares são programas de computador e documentação associada. Produtos de software podem ser desenvolvidos para um cliente específico ou para o mercado em geral.
Quais são os atributos de um bom software?	Um bom software deve prover a funcionalidade e o desempenho requeridos pelo usuário; além disso, deve ser confiável e fácil de manter e usar.
O que é engenharia de software?	É uma disciplina de engenharia que se preocupa com todos os aspectos de produção de software.
Quais são as principais atividades da engenharia de software?	Especificação de software, desenvolvimento de software, validação de software e evolução de software.
Qual a diferença entre engenharia de software e ciência da computação?	Ciência da computação foca a teoria e os fundamentos; engenharia de software preocupa-se com o lado prático do desenvolvimento e entrega de softwares úteis.
Qual a diferença entre engenharia de software e engenharia de sistemas?	Engenharia de sistemas se preocupa com todos os aspectos do desenvolvimento de sistemas computacionais, incluindo engenharia de hardware, software e processo. Engenharia de software é uma parte específica desse processo mais genérico.
Quais são os principais desafios da engenharia de software?	Lidar com o aumento de diversidade, demandas pela diminuição do tempo para entrega e desenvolvimento de software confiável.
Quais são os custos da engenharia de software?	Aproximadamente 60% dos custos de software são de desenvolvimento; 40% são custos de testes. Para software customizado, os custos de evolução frequentemente superam os custos de desenvolvimento.
Quais são as melhores técnicas e métodos da engenharia de software?	Enquanto todos os projetos de software devem ser gerenciados e desenvolvidos profissionalmente, técnicas diferentes são adequadas para tipos de sistemas diferentes. Por exemplo, jogos devem ser sempre desenvolvidos usando uma série de protótipos, enquanto sistemas de controle críticos de segurança requerem uma especificação analisável e completa. Portanto, não se pode dizer que um método é melhor que outro.
Quais diferenças foram feitas pela Internet na engenharia de software?	A Internet tornou serviços de software disponíveis e possibilitou o desenvolvimento de sistemas altamente distribuídos baseados em serviços. O desenvolvimento de sistemas baseados em Web gerou importantes avanços nas linguagens de programação e reúso de software.

Uma diferença importante entre esses tipos de software é que, em softwares genéricos, a organização que o desenvolve controla sua especificação. Para produtos sob encomenda, a especificação é normalmente desenvolvida e controlada pela empresa que está adquirindo o software. Os desenvolvedores de software devem trabalhar de acordo com essa especificação.

No entanto, a distinção entre esses tipos de produtos de software está se tornando cada vez mais obscura. Mais e mais sistemas vêm sendo construídos tendo por base um produto genérico, que é então adaptado para atender aos requisitos de um cliente. Sistemas ERP (sistema integrado de gestão empresarial, do inglês *enterprise resource planning*), como o sistema SAP, são os melhores exemplos dessa abordagem. Nesse caso, um sistema grande e complexo é adaptado para uma empresa, incorporando informações sobre as regras e os processos de negócio, relatórios necessários etc.

Quando falamos sobre a qualidade do software profissional, devemos levar em conta que o software é usado e alterado pelas pessoas, além de seus desenvolvedores. A qualidade, portanto, implica não apenas o que o software faz. Ao contrário, ela tem de incluir o comportamento do software enquanto ele está executando, bem como a estrutura e a organização dos programas do sistema e a documentação associada. Isso se reflete nos atributos de software chamados não funcionais ou de qualidade. Exemplos desses atributos são o tempo de resposta do software a uma consulta do usuário e a compreensão do código do programa.

Um conjunto específico de atributos que você pode esperar de um software obviamente depende da aplicação. Portanto, um sistema bancário deve ser seguro, um jogo interativo deve ser ágil, um sistema de comutação de telefonia deve ser confiável, e assim por diante. Tudo isso pode ser generalizado em um conjunto de atributos sumarizados na Tabela 1.2, que eu acredito serem as características essenciais de um sistema profissional de software.

Tabela 1.2 Atributos essenciais de um bom software

Características do produto	Descrição
Manutenibilidade	O software deve ser escrito de forma que possa evoluir para atender às necessidades dos clientes. Esse é um atributo crítico, porque a mudança de software é um requisito inevitável de um ambiente de negócios em mudança.
Confiança e proteção	A confiança do software inclui uma série de características como confiabilidade, proteção e segurança. Um software confiável não deve causar prejuízos físicos ou econômicos no caso de falha de sistema. Usuários maliciosos não devem ser capazes de acessar ou prejudicar o sistema.
Eficiência	O software não deve desperdiçar os recursos do sistema, como memória e ciclos do processador. Portanto, eficiência inclui capacidade de resposta, tempo de processamento, uso de memória etc.
Aceitabilidade	O software deve ser aceitável para o tipo de usuário para o qual foi projetado. Isso significa que deve ser compreensível, usável e compatível com outros sistemas usados por ele.



1.1.1 Engenharia de software

Engenharia de software é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado. Há duas expressões importantes nessa definição:

1. *Disciplina de engenharia.* Engenheiros fazem as coisas funcionarem. Eles aplicam teorias, métodos e ferramentas onde for apropriado. No entanto, eles os usam seletivamente e sempre tentam descobrir as soluções para os problemas, mesmo quando não há teorias e métodos aplicáveis. Os engenheiros também reconhecem que devem trabalhar de acordo com as restrições organizacionais e financeiras, então buscam soluções dentro dessas restrições.
2. *Todos os aspectos da produção de software.* A engenharia de software não se preocupa apenas com os processos técnicos do desenvolvimento de software. Ela também inclui atividades como gerenciamento de projeto de software e desenvolvimento de ferramentas, métodos e teorias para apoiar a produção de software.

Engenharia tem a ver com obter resultados de qualidade requeridos dentro do cronograma e do orçamento. Isso frequentemente envolve ter compromissos — engenheiros não podem ser perfeccionistas. Por outro lado, as pessoas que escrevem programas para si mesmas podem gastar o tempo que quiserem com o desenvolvimento do programa.

Em geral, os engenheiros de software adotam uma abordagem sistemática e organizada para seu trabalho, pois essa costuma ser a maneira mais eficiente de produzir software de alta qualidade. No entanto, engenharia tem tudo a ver com selecionar o método mais adequado para um conjunto de circunstâncias, então **uma abordagem mais criativa e menos formal pode ser eficiente em algumas circunstâncias**. Desenvolvimento menos formal é particularmente adequado para o desenvolvimento de sistemas Web, que requerem uma mistura de habilidades de software e de projeto.

Engenharia de software é importante por dois motivos:

1. Cada vez mais, indivíduos e sociedades dependem dos sistemas de software avançados. Temos de ser capazes de produzir sistemas confiáveis econômica e rapidamente.
2. Geralmente é mais barato, a longo prazo, usar métodos e técnicas da engenharia de software para sistemas de software, em vez de simplesmente escrever os programas como se fossem algum projeto pessoal. Para a maioria dos sistemas, a maior parte do custo é mudar o software depois que ele começa a ser usado.

A abordagem sistemática usada na engenharia de software é, às vezes, chamada processo de software. Um processo de software é uma sequência de atividades que leva à produção de um produto de software. Existem quatro atividades fundamentais comuns a todos os processos de software. São elas:

1. **Especificação de software**, em que clientes e engenheiros definem o software a ser produzido e as restrições de sua operação.
2. **Desenvolvimento de software**, em que o software é projetado e programado.

- 3. Validação de software**, em que o software é verificado para garantir que é o que o cliente quer.
- 4. Evolução de software**, em que o software é modificado para refletir a mudança de requisitos do cliente e do mercado.

Tipos diferentes de sistemas necessitam de diferentes processos de desenvolvimento. Por exemplo, um software de tempo real em uma aeronave precisa ser completamente especificado antes de se iniciar o desenvolvimento. Em sistemas de comércio eletrônico, a especificação e o programa são, normalmente, desenvolvidos juntos. Consequentemente, essas atividades genéricas podem ser organizadas de formas diferentes e descritas em nível de detalhamento diferente, dependendo do tipo de software em desenvolvimento. Descrevo processos de software em mais detalhes no Capítulo 2.

Engenharia de software se relaciona tanto com ciência da computação quanto com engenharia de sistemas:

1. A ciência da computação se preocupa com as teorias e métodos que sustentam sistemas computacionais e de software, ao passo que a engenharia de software se preocupa com os problemas práticos de produção de software. Algum conhecimento de ciência da computação é essencial para engenheiros de software, da mesma forma que algum conhecimento de física é essencial para engenheiros elétricos. No entanto, a teoria da ciência da computação é, em geral, mais aplicável para programas relativamente pequenos. Teorias elegantes da ciência da computação nem sempre podem ser aplicadas em problemas grandes e complexos que requerem uma solução através de software.
- 2. A engenharia de sistemas** foca todos os aspectos do desenvolvimento e da evolução de sistemas complexos em que o software tem o papel principal. A engenharia de sistemas se preocupa com desenvolvimento de hardware, projeto de políticas e processos e implantação de sistemas, além de engenharia de software. Engenheiros de sistemas são envolvidos em especificação do sistema, definição da arquitetura geral e integração de diferentes partes para criar o sistema acabado. Eles se preocupam menos com a engenharia dos componentes do sistema (hardware, software etc.).

Conforme discutido na próxima seção, existem muitos tipos de software. Não existe um método ou uma técnica universal de engenharia de software que se aplique a todos. No entanto, há três aspectos gerais que afetam vários tipos diferentes de software:

- 1. Heterogeneidade.** Cada vez mais se requer dos sistemas que operem como sistemas distribuídos através das redes que incluem diferentes tipos de computadores e dispositivos móveis. Além de executar nos computadores de propósito geral, o software talvez tenha de executar em telefones móveis. Frequentemente, você tem de integrar software novo com sistemas mais antigos, escritos em linguagens de programação diferentes. O desafio aqui é desenvolver técnicas para construir um software confiável que seja flexível o suficiente para lidar com essa heterogeneidade.
- 2. Mudança de negócio e social.** Negócio e sociedade estão mudando de maneira incrivelmente rápida, à medida que as economias emergentes se desenvolvem e as novas tecnologias se tornam disponíveis. Deve ser possível alterar seu software existente e desenvolver um novo software rapidamente. Muitas técnicas tradicionais de engenharia de software consomem tempo, e a entrega de novos sistemas frequentemente é mais demorada do que o planejado. É preciso evoluir para que o tempo requerido para o software dar retorno a seus clientes seja reduzido.
- 3. Segurança e confiança.** Pelo fato de o software estar presente em todos os aspectos de nossas vidas, é essencial que possamos confiar nele. Isso se torna verdade especialmente para sistemas remotos acessados através de uma página Web ou uma interface de web service. Precisamos ter certeza de que os usuários maliciosos não possam atacar nosso software e de que a proteção da informação seja mantida.

É claro que essas questões não são independentes. Por exemplo, pode ser necessário fazer mudanças rápidas em um sistema legado para que se possa oferecer o mesmo com uma interface de web service. Para resolver esses desafios precisaremos de novas ferramentas e técnicas, bem como de maneiras inovadoras de combinar e usar métodos de engenharia de software existentes.



1.1.2 Diversidade na engenharia de software

Engenharia de software é uma abordagem sistemática para a produção de software; ela analisa questões práticas de custo, prazo e confiança, assim como as necessidades dos clientes e produtores do software. A forma como essa abordagem sistemática é realmente implementada varia dramaticamente de acordo com a organização que

esteja desenvolvendo o software, o tipo de software e as pessoas envolvidas no processo de desenvolvimento. Não existem técnicas e métodos universais na engenharia de software adequados a todos os sistemas e todas as empresas. Em vez disso, um conjunto diverso de métodos e ferramentas de engenharia de software tem evoluído nos últimos 50 anos.

Talvez o fator mais significante em determinar quais técnicas e métodos de engenharia de software são mais importantes seja o tipo de aplicação a ser desenvolvida. Existem muitos tipos diferentes de aplicações, incluindo:

1. *Aplicações stand-alone*. Essas são as aplicações executadas em um computador local, como um PC. Elas contêm toda a funcionalidade necessária e não precisam estar conectadas a uma rede. Exemplos de tais aplicações são aplicativos de escritório em um PC, programas CAD, software de manipulação de fotos etc.
2. *Aplicações interativas baseadas em transações*. São aplicações que executam em um computador remoto, acessadas pelos usuários a partir de seus computadores ou terminais. Certamente, aqui são incluídas aplicações Web como aplicações de comércio eletrônico em que você pode interagir com o sistema remoto para comprar produtos ou serviços. Essa classe de aplicações também inclui sistemas corporativos, em que uma empresa fornece acesso a seus sistemas através de um navegador Web ou um programa cliente especial e serviços baseados em nuvem, como é o caso de serviços de e-mail e compartilhamento de fotos. Aplicações interativas frequentemente incorporam um grande armazenamento de dados, que é acessado e atualizado em cada transação.
3. *Sistemas de controle embutidos*. São sistemas de controle que controlam e gerenciam dispositivos de hardware. Numericamente, é provável que haja mais sistemas embutidos do que de qualquer outro tipo. Exemplos de sistemas embutidos incluem software em telefone celular, softwares que controlam antitravamento de freios em um carro e software em um micro-ondas para controlar o processo de cozimento.
4. *Sistemas de processamento de lotes*. São sistemas corporativos projetados para processar dados em grandes lotes. Eles processam grande número de entradas individuais para criar as saídas correspondentes. Exemplos de sistemas de lotes incluem sistemas periódicos de cobrança, como sistemas de cobrança telefônica, e sistemas de pagamentos de salário.
5. *Sistemas de entretenimento*. São sistemas cuja utilização principal é pessoal e cujo objetivo é entreter o usuário. A maioria desses sistemas é de jogos de diferentes tipos. A qualidade de interação com o usuário é a característica particular mais importante dos sistemas de entretenimento.
6. *Sistemas para modelagem e simulação*. São sistemas que incluem vários objetos separados que interagem entre si, desenvolvidos por cientistas e engenheiros para modelar processos ou situações físicas. Esses sistemas geralmente fazem uso intensivo de recursos computacionais e requerem sistemas paralelos de alto desempenho para executar.
7. *Sistemas de coleta de dados*. São sistemas que coletam dados de seu ambiente com um conjunto de sensores e enviam esses dados para outros sistemas para processamento. O software precisa interagir com sensores e frequentemente é instalado em um ambiente hostil, por exemplo, dentro de uma máquina ou em um lugar remoto.
8. *Sistemas de sistemas*. São sistemas compostos de uma série de outros sistemas de software. Alguns deles podem ser produtos genéricos de software, como um programa de planilha eletrônica. Outros sistemas do conjunto podem ser escritos especialmente para esse ambiente.

É claro que as fronteiras entre esses tipos de sistema não são claras. Se você desenvolve um jogo para um telefone celular, deve levar em conta as mesmas restrições (energia, interação com hardware) que os desenvolvedores do software do telefone. Sistemas de processamento de lotes são frequentemente usados em conjunto com sistemas Web. Por exemplo, as requisições para reembolso das viagens dentro de uma empresa podem ser submetidas por meio de uma aplicação Web, porém processadas por uma aplicação de processamento de lotes para pagamento mensal.

Utilizamos diferentes técnicas de engenharia de software para cada tipo de sistema, porque cada software tem características bastante diversas. Por exemplo, um sistema de controle embutido em um automóvel é de segurança crítica e é gravado em memória ROM quando instalado no veículo. Por isso, sua alteração é muito cara. Tal sistema necessita de verificação e validação muito extensas para que as chances de ter de fazer um *recall* de carros depois de vendidos para correção de software sejam minimizadas. A interação do usuário, por sua vez, é mínima (ou talvez até inexistente), então não há necessidade de um processo de desenvolvimento que se baseie em prototipação de telas.

Para um sistema Web, uma abordagem baseada em desenvolvimento e entregas iterativas pode ser adequada, com o sistema sendo composto a partir de componentes reusáveis. No entanto, tal abordagem pode ser inviável para um sistema de sistemas, no qual as especificações detalhadas das interações do sistema precisam estar detalhadas antes para que cada sistema possa ser desenvolvido separadamente.

Apesar disso, existem fundamentos de engenharia de software que se aplicam a todos os tipos de sistemas de software:

1. Eles devem ser desenvolvidos em um processo gerenciado e compreendido. A organização que desenvolve o software deve planejar o processo de desenvolvimento e ter ideias claras do que será produzido e quando estará finalizado. É claro que processos diferentes são usados para tipos de software diferentes.
2. Confiança e desempenho são importantes para todos os tipos de sistema. O software deve se comportar conforme o esperado, sem falhas, e deve estar disponível para uso quando requerido. Deve ser seguro em sua operação e deve ser, tanto quanto possível, protegido contra ataques externos. O sistema deve executar de forma eficiente e não deve desperdiçar recursos.
3. É importante entender e gerenciar a especificação e os requisitos de software (o que o software deve fazer). Você deve saber o que clientes e usuários esperam dele e deve gerenciar suas expectativas para que um sistema útil possa ser entregue dentro do orçamento e do cronograma.
4. Você deve fazer o melhor uso possível dos recursos existentes. Isso significa que, quando apropriado, você deve reusar o software já desenvolvido, em vez de escrever um novo.

Essas noções básicas de processo, confiança, requisitos, gerenciamento e reuso são temas importantes desta obra. São refletidas de várias maneiras por diferentes métodos, e são o fundamento de todo o desenvolvimento de software profissional.

Você deve observar que esses fundamentos não cobrem implementação e programação. Eu não cubro técnicas de programação específicas neste livro, porque elas variam dramaticamente de um tipo de sistema para outro. Por exemplo, uma linguagem de *script* como Ruby é usada para programação de sistemas Web, porém seria totalmente inadequada para engenharia de sistemas embutidos.



1.1.3 Engenharia de software e a Internet

O desenvolvimento da Internet teve efeito profundo em nossas vidas. No início, a Internet era basicamente um armazenamento de informações acessível universalmente e tinha pouco efeito nos sistemas de software. Esses sistemas executavam em computadores locais e eram acessíveis apenas dentro da organização. Por volta do ano 2000, a Internet começou a evoluir, e mais e mais recursos passaram a ser adicionados aos navegadores. Isso significa que sistemas Web poderiam ser desenvolvidos e que, em vez de ter uma interface de usuário específica, poderiam ser acessados por um navegador. Isso levou ao desenvolvimento de uma enorme quantidade de novos produtos de software que ofereciam serviços inovadores e que eram acessados através da Internet. Esses produtos eram frequentemente sustentados pela propaganda exibida na tela do usuário e não exigiam pagamento direto.

Assim como esses produtos de software, o desenvolvimento de navegadores Web capazes de executar programas pequenos e fazer algum processamento local levou a uma evolução no software corporativo e organizacional. Em vez de escrever o software e instalá-lo nos computadores dos usuários, o software era implantado em um servidor Web. Isso tornou muito mais barato alterar e atualizar o software, porque não havia necessidade de se instalar o software em cada computador. Isso também reduziu os custos, porque o desenvolvimento de interface de usuário é particularmente caro. Consequentemente, sempre que possível, muitos negócios mudaram para interação Web com os sistemas de software da empresa.

O próximo estágio no desenvolvimento de sistemas Web foi a noção de *web services*. *Web services* são componentes de software acessados pela Internet e fornecem uma funcionalidade específica e útil. Aplicações são construídas integrando esses *web services*, os quais podem ser fornecidos por empresas diferentes. A princípio, essa ligação pode ser dinâmica, para que a aplicação possa usar *web services* diferentes toda vez que é executada. Essa abordagem para desenvolvimento de software é discutida no Capítulo 19.

Nos últimos anos, desenvolveu-se a ideia de 'software como serviço'. Foi proposto que o software normalmente não executará em computadores locais, e sim em 'nuvens computacionais' acessadas pela Internet. Se você usa um serviço como um webmail, está usando um sistema baseado em nuvem. Uma nuvem computacional consiste em um grande número de sistemas computacionais interligados, os quais são compartilhados entre vários usuários.

Os usuários não compram o software, mas pagam de acordo com o uso ou possuem acesso gratuito em troca de propagandas que são exibidas em suas telas.

Portanto, o surgimento da Internet trouxe uma mudança significativa na maneira como o software corporativo é organizado. Antes da Internet, aplicações corporativas eram, na maioria das vezes, monolíticas, programas isolados executando em computadores isolados ou em *clusters* de computadores. Agora, um software é altamente distribuído, às vezes pelo mundo todo. As aplicações corporativas não são programadas do zero; de fato, elas envolvem reúso extensivo de componentes e programas.

Essa mudança radical na organização de software obviamente causou mudanças na maneira como os sistemas Web são projetados. Por exemplo:

1. O reúso de software tornou-se a abordagem dominante para a construção de sistemas Web. Quando construímos esses sistemas, pensamos em como podemos montá-los a partir de componentes e sistemas de software preexistentes.
2. Atualmente, aceita-se que é impraticável especificar todos os requisitos para tais sistemas antecipadamente. Sistemas Web devem ser desenvolvidos e entregues incrementalmente.
3. Interfaces de usuário são restrinidas pela capacidade dos navegadores. Embora tecnologias como AJAX (HOLDENER, 2008) signifiquem que interfaces ricas podem ser criadas dentro de um navegador, essas tecnologias ainda são difíceis de usar. Formulários Web com *scripts* locais são mais usados. Interfaces das aplicações em sistemas Web são normalmente mais pobres do que interfaces projetadas especialmente para produtos de software que executam em PCs.

As ideias fundamentais da engenharia de software discutidas na seção anterior aplicam-se para software baseado em Web da mesma forma que para outros tipos de sistemas de software. A experiência adquirida com o desenvolvimento de grandes sistemas no século XX ainda é relevante para softwares baseados em Web.

1.2 Ética na engenharia de software

Assim como outras disciplinas de engenharia, a engenharia de software é desenvolvida dentro de um framework social e legal que limita a liberdade das pessoas que trabalham nessa área. Como um engenheiro de software, você deve aceitar que seu trabalho envolve maiores responsabilidades do que simplesmente aplicar habilidades técnicas. Você também deve se comportar de forma ética e moralmente responsável se deseja ser respeitado como um engenheiro profissional.

Isso sem falar que você deve manter padrões normais de honestidade e integridade. Você não deve usar suas habilidades e seu conhecimento para se comportar de forma desonesta ou de maneira que possa denegrir a profissão de engenharia de software. No entanto, existem áreas nas quais os padrões de comportamento aceitável não são limitados pelas leis, mas pela mais tênue noção de responsabilidade profissional. Algumas delas são:

1. *Confidencialidade*. Você deve respeitar naturalmente a confidencialidade de seus empregadores ou clientes, independentemente de ter sido ou não assinado um acordo formal de confidencialidade.
2. *Competência*. Você não deve deturpar seu nível de competência. Você não deve aceitar conscientemente um trabalho que esteja fora de sua competência.
3. *Direitos de propriedade intelectual*. Você deve ter conhecimento das leis locais a respeito da propriedade intelectual, como patentes e copyright. Você deve ter cuidado para garantir que a propriedade intelectual dos empregadores e clientes seja protegida.
4. *Mau uso do computador*. Você não deve usar suas habilidades técnicas para fazer mau uso de computadores de outras pessoas. Esse mau uso varia de relativamente trivial (jogar videogames em uma máquina do empregador, por exemplo) até extremamente sério (disseminar vírus ou outros *malwares*).

Sociedades e instituições profissionais têm um papel importante a desempenhar na definição de padrões éticos. Organizações como ACM, IEEE (Institute of Electrical and Electronic Engineers) e British Computer Society publicam um código de conduta profissional ou código de ética. Membros dessas organizações se comprometem a seguir esse código quando se tornam membros. Esses códigos de conduta normalmente se preocupam com o comportamento ético básico.

Associações profissionais, principalmente ACM e IEEE, cooperaram para produzir o código de ética e práticas profissionais. Esse código existe tanto na forma reduzida, mostrada no Quadro 1.1, quanto na forma completa

(GOTTERBARN et al., 1999), a qual acrescenta detalhes e conteúdo à versão resumida. O raciocínio por trás desse código está sumarizado nos dois primeiros parágrafos da versão completa:

Computadores têm um papel central e crescente no comércio, na indústria, no governo, na medicina, na educação, no entretenimento e na sociedade de um modo geral. Os engenheiros de software são aqueles que contribuem com a participação direta, ou lecionando, para análise, especificação, projeto, desenvolvimento, certificação, manutenção e testes de sistemas de software. Por causa de seu papel no desenvolvimento de sistemas de software, os engenheiros de software têm diversas oportunidades para fazer o bem ou para causar o mal, possibilitar que outros façam o bem ou causem o mal ou influenciar os outros a fazer o bem ou causar o mal. Para garantir ao máximo que seus esforços serão usados para o bem, os engenheiros de software devem se comprometer a fazer da engenharia de software uma profissão benéfica e respeitada. De acordo com esse compromisso, os engenheiros de software devem aderir ao Código de Ética e Prática Profissional a seguir.

O Código contém oito princípios relacionados ao comportamento e às decisões dos engenheiros de software profissionais, incluindo praticantes, educadores, gerentes, supervisores e criadores de políticas, assim como trainees e estudantes da profissão. Esses princípios identificam os relacionamentos éticamente responsáveis dos quais cada indivíduo, grupo e organização participa e as principais obrigações dentro desses relacionamentos. As cláusulas de cada princípio são ilustrações de algumas obrigações inclusas nesses relacionamentos. Essas obrigações se baseiam na humanidade do engenheiro de software, especialmente no cuidado devido às pessoas afetadas pelo trabalho dos engenheiros de software e em elementos próprios da prática de engenharia de software. O Código prescreve essas obrigações como de qualquer um que se diz ser ou pretende ser um engenheiro de software.

Em qualquer situação em que pessoas diferentes têm visões e objetivos diferentes, é provável que se enfrentem dilemas éticos. Por exemplo, se você discordar, em princípio, das políticas do gerenciamento de nível mais alto da empresa, como deve agir? É óbvio que isso depende das pessoas envolvidas e da natureza do desacordo. É melhor sustentar sua posição dentro da organização ou demitir-se por princípio? Se você acha que há problemas com um projeto de software, quando deve revelar isso à gerência? Se você discutir isso enquanto existem apenas suspeitas, poderá estar exagerando; se deixar para muito depois, poderá ser impossível resolver as dificuldades.

Tais dilemas éticos acontecem com todos nós em nossas vidas profissionais e, felizmente, na maioria dos casos eles são relativamente pequenos ou podem ser resolvidos sem muitas dificuldades. Quando não podem ser resolvidos, o engenheiro enfrenta, talvez, outro problema. A ação baseada em princípios pode ser pedir demissão, mas isso pode afetar outras pessoas, como seus(suas) companheiros(as) e filhos.

Quadro 1.1 Código de ética da ACM/IEEE (© IEEE/ACM 1999)

Código de ética e práticas profissionais da engenharia de software

Força-tarefa conjunta da ACM/IEEE-CS para ética e práticas profissionais da engenharia de software

Prefácio

Esta versão reduzida do código resume as aspirações em um alto nível de abstração; as cláusulas que estão inclusas na versão completa fornecem exemplos e detalhes de como essas aspirações mudam a forma como agimos enquanto profissionais de engenharia de software. Sem as aspirações, os detalhes podem se tornar legalistas e tediosos; sem os detalhes, as aspirações podem se tornar altissonsantes, porém vazias; juntos, as aspirações e os detalhes formam um código coeso.

Os engenheiros de software devem se comprometer a fazer da análise, especificação, projeto, desenvolvimento, teste e manutenção de software uma profissão benéfica e respeitada. Em conformidade com seu comprometimento com a saúde, a segurança e o bem-estar públicos, engenheiros de software devem aderir a oito princípios:

1. PÚBLICO — Engenheiros de software devem agir de acordo com o interesse público.
2. CLIENTE E EMPREGADOR — Engenheiros de software devem agir de maneira que seja do melhor interesse de seu cliente e empregador e de acordo com o interesse público.
3. PRODUTO — Engenheiros de software devem garantir que seus produtos e modificações relacionadas atendam aos mais altos padrões profissionais possíveis.
4. JULGAMENTO — Engenheiros de software devem manter a integridade e a independência em seu julgamento profissional.
5. GERENCIAMENTO — Gerentes e líderes de engenharia de software devem aceitar e promover uma abordagem ética para o gerenciamento de desenvolvimento e manutenção de software.
6. PROFISSÃO — Engenheiros de software devem aprimorar a integridade e a reputação da profissão de acordo com o interesse público.
7. COLEGAS — Engenheiros de software devem auxiliar e ser justos com seus colegas.
8. SI PRÓPRIO — Engenheiros de software devem participar da aprendizagem continua durante toda a vida, e devem promover uma abordagem ética para a prática da profissão.

Uma situação particularmente difícil para engenheiros profissionais aparece quando seu empregador age de forma antiética. Digamos que a empresa seja responsável por desenvolver um sistema de missão crítica e, por causa da pressão pelos prazos, falsifique os registros de validação de segurança. A responsabilidade do engenheiro é manter a confidencialidade, alertar o cliente ou divulgar, de alguma forma, que o sistema pode não ser seguro?

O problema aqui é que não há valores absolutos quando se trata de segurança. Embora o sistema possa não ter sido validado de acordo com os critérios predefinidos, esses critérios podem ser rígidos demais. O sistema pode, de fato, operar com segurança durante todo seu ciclo de vida. Também ocorre que, mesmo adequadamente validado, o sistema pode falhar e causar um acidente. A divulgação antecipada dos problemas pode resultar em prejuízo para o empregador e outros empregados; não divulgar os problemas pode resultar em prejuízo para outros.

Você deve tomar as próprias decisões em situações como essas. A postura ética adequada aqui depende totalmente dos pontos de vista dos indivíduos envolvidos. Nesse caso, o potencial do prejuízo, sua extensão e as pessoas afetadas por ele devem influenciar a decisão. Se a situação for muito perigosa, pode ser justificável divulgá-la usando a imprensa nacional (por exemplo). No entanto, você sempre deve tentar resolver a situação respeitando os direitos de seu empregador.

Outra questão ética é a participação no desenvolvimento de sistemas militares e nucleares. Algumas pessoas têm sentimentos fortes sobre essas questões e não desejam participar de qualquer desenvolvimento associado a sistemas militares. Outros trabalham em sistemas militares, mas não nos associados a armas. E ainda há aqueles que acham que a segurança nacional é um princípio fundamental e não têm objeções éticas em trabalhar em sistemas de armas.

Nessa situação, é importante que empregadores e empregados exponham sua visão uns aos outros antecipadamente. Quando uma organização está envolvida em um trabalho militar ou nuclear, ela deve ser capaz de deixar claro que os empregados devem estar dispostos a aceitar qualquer atribuição no trabalho. Igualmente, se qualquer empregado deixar claro que não deseja trabalhar em tais sistemas, os empregadores não devem pressioná-lo para fazer isso futuramente.

A área geral de ética e responsabilidade profissional está ficando mais importante à medida que os sistemas que fazem uso intensivo de software se infiltram em cada aspecto do trabalho e da vida cotidiana. Isso pode ser analisado do ponto de vista filosófico, em que os princípios básicos de ética são considerados, e a ética de engenharia de software é discutida com referência a esses princípios. Essa é a abordagem usada por Laudon (1995) e, em extensão menor, por Huff e Martin (1995). O artigo de Johnson sobre ética na computação (2001) também aborda o assunto de uma perspectiva filosófica.

No entanto, eu acho que essa abordagem filosófica é muito abstrata e difícil de ser relacionada com a experiência cotidiana. Eu prefiro uma abordagem mais concreta, baseada em códigos de conduta e práticas. Considero que a ética é mais bem discutida em um contexto de engenharia de software, e não como um assunto à parte. Portanto, não incluí neste livro discussões éticas abstratas, e sim, quando apropriado, exemplos nos exercícios que podem ser o ponto de partida para uma discussão em grupo sobre questões éticas.

1.3 Estudos de caso

Para ilustrar os conceitos de engenharia de software neste livro, uso exemplos de três diferentes tipos de sistemas. O motivo de não ter usado um único estudo de caso é que uma das mensagens-chave desta obra é que a prática da engenharia de software depende do tipo de sistema que está sendo produzido. Dessa forma, posso escolher um exemplo adequado quando discuto conceitos como segurança e confiança, modelagem de sistema, reuso etc.

Os três tipos de sistema que uso como estudos de caso são:

1. *Um sistema embutido.* Trata-se de um sistema no qual o software controla um dispositivo de hardware e é embutido nesse dispositivo. As questões em sistemas embutidos incluem tipicamente o tamanho físico, a capacidade de resposta, o gerenciamento de energia etc. O exemplo de um sistema embutido que uso é um sistema para controlar um dispositivo médico.
2. *Um sistema de informação.* Esse é um sistema cujo principal objetivo é gerenciar e prover acesso a um banco de dados de informações. As questões em sistemas de informação incluem proteção, usabilidade, privacidade

e manutenção da integridade dos dados. O exemplo de um sistema de informação que uso é um sistema de registros médicos.

3. *Um sistema de coleta de dados baseado em sensores.* Esse é um sistema cujo principal objetivo é coletar dados a partir de um conjunto de sensores e processá-los de alguma forma. Os principais requisitos de tais sistemas são confiabilidade, mesmo em condições ambientais hostis, e manutenibilidade. O exemplo de um sistema de coleta de dados que uso é uma estação meteorológica no deserto.

Apresento cada um desses sistemas neste capítulo, com mais informações a respeito de cada um deles disponíveis na Internet.



1.3.1 Sistema de controle de bomba de insulina

Uma bomba de insulina é um sistema médico que simula o funcionamento do pâncreas (um órgão interno). O software que controla o sistema é um sistema embutido, que coleta as informações a partir de um sensor e controla uma bomba que fornece uma dose controlada de insulina para o usuário.

Pessoas que sofrem de diabetes utilizam esse sistema. Diabetes é uma condição relativamente comum, na qual o pâncreas humano é incapaz de produzir quantidade suficiente de um hormônio chamado insulina. A insulina metaboliza glicose (açúcar) no sangue. O tratamento convencional de diabetes envolve injeções regulares de insulina sintetizada. Os diabéticos medem o nível de açúcar no sangue com um medidor externo, e depois calculam a dose de insulina que devem injetar.

O problema com esse tratamento é que o nível requerido de insulina não depende apenas do nível de glicose no sangue, mas também do tempo desde a última injeção. Isso pode levar a níveis muito baixos de glicose no sangue (se houver insulina demais) ou níveis muito altos de açúcar no sangue (se houver muito pouca insulina). Glicose baixa no sangue é, resumidamente, uma condição mais séria, porque pode resultar em mau funcionamento temporário do cérebro e, em casos extremos, inconsciência e morte. A longo prazo, no entanto, níveis altos contínuos de glicose no sangue podem causar prejuízos aos olhos, aos rins e problemas de coração.

Os avanços atuais no desenvolvimento de sensores miniaturizados possibilitaram a criação de sistemas automatizados de fornecimento de insulina. Esses sistemas monitoram o nível de açúcar no sangue e fornecem uma dose adequada de insulina quando necessário. Sistemas de fornecimento de insulina como esse já existem para o tratamento de pacientes hospitalares. No futuro, será possível para muitos diabéticos ter tais sistemas instalados permanentemente no corpo.

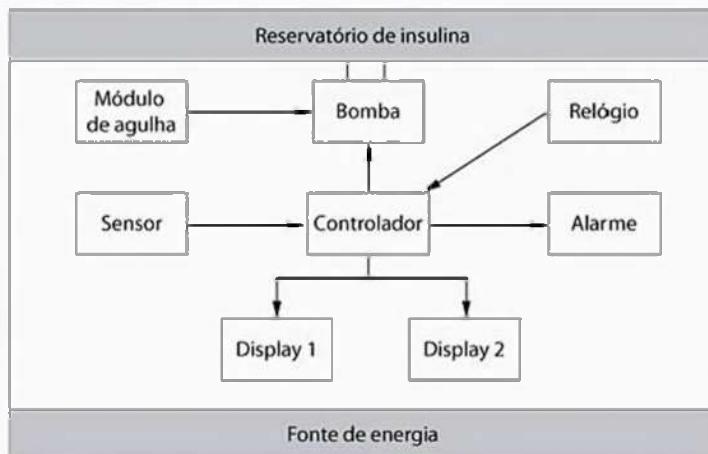
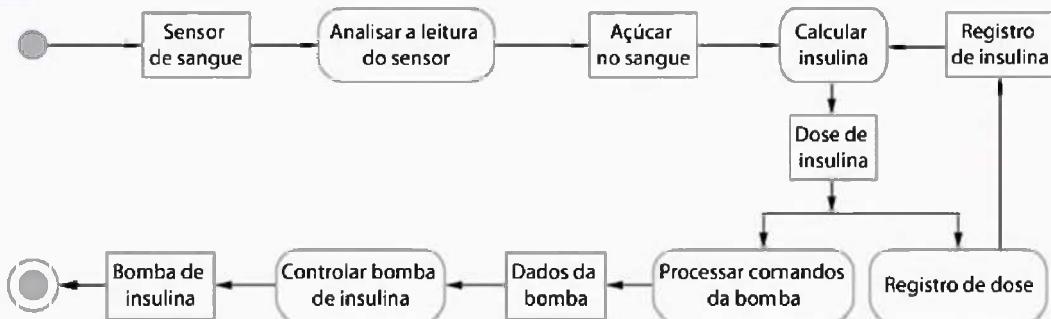
Um sistema de fornecimento de insulina controlado por software pode funcionar com o uso de um microsensor embutido no paciente para medir algum parâmetro do sangue que seja proporcional ao nível de açúcar. A informação coletada é então enviada para o controlador da bomba. Esse controlador calcula o nível de sangue e a quantidade necessária de insulina. Depois, um sinal é enviado para a bomba miniaturizada para fornecer a insulina através de uma agulha permanente.

A Figura 1.1 mostra os componentes de hardware e a organização da bomba de insulina. Para compreender os exemplos deste livro, tudo o que você precisa saber é que o sensor de sangue mede a condutividade elétrica do sangue em diferentes condições, e que esses valores podem ser relacionados ao nível de açúcar no sangue. A bomba de insulina fornece uma unidade de insulina como resposta a um único pulso do controlador. Portanto, para fornecer dez unidades de insulina, o controlador envia dez pulsos à bomba. A Figura 1.2 é um modelo de atividade UML (linguagem de modelagem unificada, do inglês *unified modeling language*), que ilustra como o software transforma uma entrada de nível de açúcar no sangue em uma sequência de comandos que operam a bomba de insulina.

É óbvio que esse é um sistema crítico de segurança. Se a bomba falhar a saúde do usuário pode ser prejudicada ou ele pode entrar em coma, porque o nível de açúcar em seu sangue estará muito alto ou muito baixo. Existem, portanto, dois requisitos essenciais a que esse sistema deve atender:

1. O sistema deve estar disponível para fornecer a insulina quando requerido.
2. O sistema deve executar de forma confiável e fornecer a quantidade correta de insulina para controlar o nível de açúcar no sangue.

Portanto, o sistema deve ser projetado e implementado para garantir que atenda sempre a esses requisitos. Requisitos e discussões mais detalhados sobre como garantir a segurança do sistema são discutidos mais adiante.

Figura 1.1 Hardware de bomba de insulina**Figura 1.2** Modelo de atividade da bomba de insulina

1.3.2 Um sistema de informação de pacientes para cuidados com saúde mental

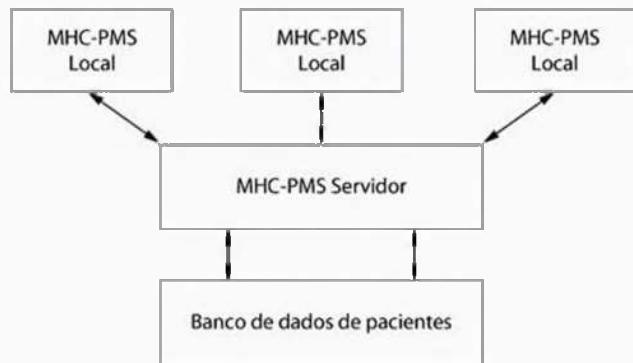
Um sistema de informação de pacientes para apoiar cuidados com saúde mental é um sistema médico de informação que mantém os dados sobre os pacientes que sofrem dos problemas de saúde mental e os tratamentos que eles receberam. A maioria dos pacientes com problemas mentais não requer tratamento hospitalar dedicado; em geral, eles precisam visitar clínicas especializadas regularmente, onde podem encontrar um médico que possua conhecimento detalhado de seus problemas. Para ficar mais fácil atender os pacientes, essas clínicas não existem apenas dentro dos hospitais. Elas também podem ser encontradas em centros comunitários de saúde.

O MHC-PMS (sistema de gerenciamento de pacientes com problemas de saúde mental, do inglês *mental health care-patient management system*) é um sistema de informação utilizado em tais clínicas. Ele usa um banco de dados centralizado de informações dos pacientes. Mas esse sistema também foi projetado para executar em um PC, para poder ser usado em ambientes que não possuem uma conexão de rede segura. Quando o sistema local possui um acesso de rede seguro, a informação sobre os pacientes é usada a partir do banco de dados, mas essa informação pode ser baixada, e uma cópia local de registros de pacientes pode ser usada quando não há conexão disponível. O sistema não é um sistema completo de registros médicos e, por isso, não contém informações sobre outras condições médicas. A Figura 1.3 ilustra a organização do MHC-PMS.

O MHC-PMS tem dois objetivos principais:

1. Gerar informação gerencial que permita aos gestores do serviço de saúde avaliar o desempenho de alvos locais e governamentais.
2. Fornecer ao pessoal médico informação atualizada para apoiar o tratamento dos pacientes.

Figura 1.3 A organização do MHC-PMS



A natureza de problemas de saúde mental tem como característica o fato de os pacientes serem, frequentemente, desorganizados e perderem seus compromissos, proposital ou accidentalmente, bem como receitas e remédios, esquecerem as instruções e demandarem atendimento médico de forma exagerada. Esses pacientes podem aparecer nas clínicas inesperadamente. Na minoria dos casos, podem se tornar um perigo para si mesmos ou para outras pessoas. Podem mudar de endereço regularmente ou estar desabrigados há muito ou pouco tempo. Quando os pacientes são perigosos, podem precisar de internação – em um hospital seguro, para tratamento e observação.

Usuários do sistema incluem o pessoal da clínica, como médicos, enfermeiros e agentes de saúde (enfermeiros que visitam as pessoas em casa para verificar seu tratamento). Usuários que não são da área médica incluem os recepcionistas que agendam as consultas, o pessoal que faz manutenção nos cadastros do sistema e o pessoal administrativo que gera os relatórios.

O sistema é usado para guardar a informação sobre os pacientes (nome, endereço, idade, parente mais próximo etc.), consultas (data, médico que atendeu, impressões subjetivas sobre o paciente etc.), condições e tratamentos. Relatórios são gerados em intervalos regulares para o pessoal médico e gestores de saúde autorizados. Normalmente, os relatórios para o pessoal médico focam a informação sobre pacientes individuais, enquanto relatórios gerenciais são anônimos e se preocupam com condições, custos dos tratamentos etc.

Os principais recursos do sistema são:

1. *Gerenciamento do cuidado individual.* O pessoal clínico pode criar registros dos pacientes, editar as informações no sistema, ver histórico dos pacientes etc. O sistema suporta resumo de dados, para que os médicos que não conheciam o paciente anteriormente possam conhecer rapidamente seus principais problemas e os tratamentos prescritos.
2. *Monitoramento de pacientes.* O sistema regularmente monitora os registros dos pacientes que são envolvidos nos tratamentos e emite alertas quando possíveis problemas são detectados. Portanto, se o paciente não se encontrou com o médico durante algum tempo, um alerta pode ser emitido. Um dos elementos mais importantes do sistema de monitoramento é manter registro dos pacientes que foram internados e garantir que as verificações legais sejam efetuadas em tempo certo.
3. *Relatórios administrativos.* O sistema gera relatórios gerenciais mensais mostrando o número de pacientes tratados em cada clínica, o número de pacientes que entraram e saíram do sistema de saúde, o número de pacientes internados, os remédios prescritos e seus custos etc.

Duas leis diferentes afetam o sistema. A lei de proteção de dados que governa a confidencialidade da informação pessoal e a lei de saúde mental, que governa a detenção compulsória de pacientes considerados perigosos para si mesmos ou para outros. A saúde mental diferencia-se das demais especialidades por ser a única que permite a um médico recomendar a internação de um paciente contra sua vontade. Isso está sujeito a garantias legislativas muito rigorosas. Um dos objetivos do MHC-PMS é garantir que o pessoal sempre aja de acordo com as leis e que suas decisões sejam registradas para fiscalização judicial, caso necessário.

Assim como acontece em todos os sistemas médicos, a privacidade é um requisito crítico do sistema. É essencial que a informação do paciente seja confidencial e que jamais seja revelada a ninguém além do pessoal médico autorizado e dos próprios pacientes. O MHC-PMS é também um sistema de segurança crítica. Algumas doenças

mentais fazem com que os pacientes se tornem suicidas ou perigosos para outras pessoas. Sempre que possível, o sistema deve alertar o pessoal médico sobre pacientes potencialmente suicidas ou perigosos.

O projeto geral do sistema deve levar em conta os requisitos de privacidade e segurança. O sistema deve estar disponível quando necessário; caso contrário, a segurança pode ficar comprometida e pode ficar impossível prescrever a medicação correta para os pacientes. Existe um conflito em potencial aqui — é mais fácil manter a privacidade quando existe apenas uma cópia de dados do sistema; no entanto, para garantir a disponibilidade no caso de falha de servidor ou quando não houver conexão de rede, múltiplas cópias de dados devem ser mantidas. Discuto os compromissos entre esses requisitos nos próximos capítulos.



1.3.3 Uma estação meteorológica no deserto

Para ajudar a monitorar as mudanças climáticas e aprimorar a exatidão de previsões do tempo em áreas distantes, o governo de um país com grandes áreas desertas decide instalar centenas de estações meteorológicas em áreas remotas. Essas estações meteorológicas coletam dados a partir de um conjunto de instrumentos que medem temperatura, pressão, sol, chuva, velocidade e direção do vento.

As estações meteorológicas no deserto são parte de um sistema maior (Figura 1.4), um sistema de informações meteorológicas que coleta dados a partir das estações meteorológicas e os disponibiliza para serem processados por outros sistemas. Os sistemas da Figura 1.4 são:

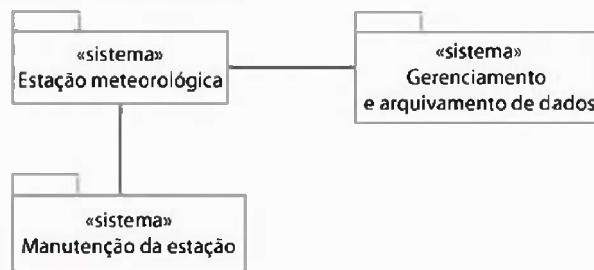
- 1. Sistema da estação meteorológica.** Responsável por coletar dados meteorológicos, efetuar algum processamento inicial de dados e transmiti-los para o sistema de gerenciamento de dados.
- 2. Sistema de gerenciamento e arquivamento de dados.** Esse sistema coleta os dados de todas as estações meteorológicas, executa o processamento e a análise dos dados e os arquiva de forma que possam ser obtidos por outros sistemas, como sistemas de previsões de tempo.
- 3. Sistema de manutenção da estação.** Esse sistema pode se comunicar via satélite com todas as estações meteorológicas no deserto para monitorar as condições desses sistemas e fornecer relatórios sobre os problemas. Ele também pode atualizar o software embutido nesses sistemas. Em caso de problema com o sistema ele pode, ainda, ser usado para controlar remotamente um sistema meteorológico no deserto.

Na Figura 1.4 utilizei o símbolo de pacote da UML para indicar que cada sistema é uma coleção de componentes e identifiquei sistemas separados, usando o estereótipo <<sistema>> da UML. As associações entre os pacotes indicam que há uma troca de informações, mas, nesse estágio, não há necessidade de definir com mais detalhes.

Cada estação meteorológica inclui uma série de instrumentos que medem os parâmetros do tempo, como velocidade e direção do vento, temperatura do solo e do ar, pressão barométrica e chuva em um período de 24 horas. Cada um desses instrumentos é controlado por um sistema de software que obtém periodicamente a leitura dos parâmetros e gerencia os dados coletados a partir desses instrumentos.

O sistema da estação meteorológica opera coletando as observações do tempo em intervalos frequentes — por exemplo, temperaturas são medidas a cada minuto. No entanto, como a largura de banda da conexão por satélite é relativamente insuficiente, a estação meteorológica efetua algum processamento local de agregação de dados. A transmissão desses dados agrupados ocorre a partir da solicitação do sistema coletor. Se, por um motivo qualquer, for impossível estabelecer a conexão, a estação meteorológica mantém os dados localmente até se restabelecer a comunicação.

Figura 1.4 Ambiente de estação meteorológica



Cada estação meteorológica tem uma bateria que a alimenta e deve ser totalmente autocontida — não há energia externa ou cabos de rede disponíveis. Todas as comunicações passam por um link via satélite relativamente lento, e a estação meteorológica deve incluir algum mecanismo (solar ou eólico) para recarregar as baterias. Por serem instaladas em áreas desertas, elas são expostas a diversas condições ambientais e podem ser avariadas por animais. O software da estação não se preocupa apenas com coleção de dados. Ele deve também:

1. Monitorar os instrumentos, energia e hardware de comunicação e reportar defeitos para o sistema de gerenciamento.
2. Gerenciar a energia do sistema, garantindo o carregamento das baterias sempre que as condições ambientais permitirem, bem como o desligamento dos geradores em condições climáticas potencialmente perigosas, como ventos fortes.
3. Permitir reconfiguração dinâmica quando partes do software forem substituídas com novas versões e quando os instrumentos de backup forem conectados ao sistema em caso de falha de sistema.

Como as estações meteorológicas precisam ser autocontidas e independentes, o software instalado é completo, apesar de a funcionalidade de coleta de dados ser bastante simples.

PONTOS IMPORTANTES

- Engenharia de software é uma disciplina de engenharia que se preocupa com todos os aspectos da produção de software.
- Software não é apenas um programa ou programas; ele inclui também a documentação. Os atributos principais de um produto de software são manutenibilidade, confiança, proteção, eficiência e aceitabilidade.
- O processo de software inclui todas as atividades envolvidas no desenvolvimento do software. Atividades de alto nível de especificação, desenvolvimento, validação e evolução são parte de todos os processos de software.
- As ideias fundamentais da engenharia de software são universalmente aplicáveis para todos os tipos de desenvolvimento de sistemas. Esses fundamentos incluem processos de software, confiança, proteção, requisitos e reúso.
- Existem vários tipos diferentes de sistemas, e cada um requer ferramentas e técnicas de engenharia de software adequadas a seu desenvolvimento. Existem poucas, se houver alguma, técnicas específicas de projeto e implementação aplicáveis para todos os tipos de sistemas.
- As ideias básicas da engenharia de software são aplicáveis a todos os tipos de sistemas de software. Esses fundamentos incluem processos de software gerenciados, confiança e proteção de software, engenharia de requisitos e reúso de software.
- Engenheiros de software têm responsabilidades com a profissão de engenharia e com a sociedade. Eles não devem se preocupar apenas com as questões técnicas.
- Sociedades profissionais publicam códigos de conduta que definem os padrões de comportamento esperado de seus membros.

LEITURA COMPLEMENTAR

"No silver bullet: Essence and accidents of software engineering". Apesar de não ser recente, esse artigo é uma boa introdução geral para os problemas da engenharia de software. A mensagem essencial do artigo ainda não mudou. (BROOKS, F. P. *IEEE Computer*, v. 20, n. 4, abr. 1987.) Disponível em: <<http://doi.ieeecomputersociety.org/10.1109/MC.1987.1663532>>.

"Software engineering code of ethics is approved". Esse artigo discute os fundamentos do desenvolvimento do Código de Ética de ACM/IEEE, e inclui ambas as formas do código, resumida e longa. (GOTTERBARN, D.; MILLER, K.; ROGERSON S. *Comm. ACM*, vol. 42, n. 10, out. 1999.) Disponível em: <<http://portal.acm.org/citation.cfm?doid=317665.317682>>.

Professional Issues in Software Engineering. Esse é um excelente livro que discute questões legais e profissionais, assim como as éticas. Eu prefiro sua abordagem prática a textos mais teóricos sobre ética. (BOTT, F.; COLEMAN, A.; EATON, J.; ROWLAND, D. *Professional Issues in Software Engineering*. 3. ed. Londres e Nova York: Taylor and Francis, 2000.)