Aspectos humanos da engenharia de software

6

Em uma edição especial da *IEEE Software*, os editores convidados [deS09] fizeram a seguinte observação:

> A engenharia de software tem uma fartura de técnicas, ferramentas e métodos projetados para melhorar tanto o processo de desenvolvimento de software quanto o produto final. Aprimoramentos técnicos continuam a surgir e a gerar resultados animadores. No entanto, software não é simplesmente um produto de soluções técnicas adequadas aplicadas a hábitos técnicos inadequados. Software é desenvolvido por pessoas, usado por pessoas e dá suporte à interação entre pessoas. Assim, características, comportamento e cooperação humanos são fundamentais no desenvolvimento prático de software.

Ao longo dos capítulos posteriores a este, vamos discutir as "técnicas, ferramentas e métodos" que resultarão na criação de um produto de software bem-sucedido. Mas, antes disso, é fundamental entender que, sem pessoas habilitadas e motivadas, o sucesso é improvável.

Conceitos-chave

PANORAMA

O que é? Todos nós temos a tendência de nos dedicarmos à linguagem de programação mais recente, aos melhores e novos métodos de projeto, ao processo ágil mais moderno ou à impressionante ferramenta de software recém lançada. Mas no frigir dos ovos são *pessoas* que constroem software de computador. E, por isso, os aspectos humanos da engenharia de software frequentemente têm tanto a ver com o sucesso de um projeto quanto a melhor e mais recente tecnologia.

Quem realiza? Indivíduos e equipes realizam o trabalho de engenharia de software. Em alguns casos, apenas uma pessoa é responsável pela maior parte do trabalho, mas no caso de produção de software em nível industrial, uma equipe de pessoas o realiza.

Por que é importante? Uma equipe de software só será bem-sucedida se sua dinâmica estiver correta. Às vezes, os engenheiros de software têm a reputação de não trabalhar bem com outras pessoas. Na verdade, é fundamental que os engenheiros de software de uma equipe trabalhem bem com seus colegas e com outros envolvidos no produto a ser construído.

Quais são as etapas envolvidas? Primeiramente, é preciso entender as características pessoais de um engenheiro de software bem-sucedido e, então, tentar imitá-las. Em seguida, você deve compreender a complexa psicologia do trabalho de engenharia de software para que possa navegar por um projeto sem riscos. Então, precisa entender a estrutura e a dinâmica de uma equipe de software, pois a engenharia de software baseada no trabalho em equipe é comum em um cenário industrial. Por fim, você deve compreender o impacto das mídias sociais, da nuvem e de outras ferramentas colaborativas.

Qual é o artefato? Uma melhor compreensão das pessoas, do processo e do produto final.

Como garantir que o trabalho foi realizado corretamente? Passe um tempo observando como os engenheiros de software bem-sucedidos fazem seu trabalho e ajuste sua abordagem para tirar proveito dos pontos positivos do projeto deles.

6.1 Características de um engenheiro de software

"A maioria dos bons programadores faz seu trabalho não porque espera pagamento ou bajulação pública, mas porque é divertido programar."

Linus Torvalds

Então você quer ser engenheiro de software? Obviamente, precisa dominar o material técnico, aprender e aplicar as habilidades exigidas para entender o problema, projetar uma solução eficaz, construir o software e testá-lo com a finalidade de produzir a mais alta qualidade possível. Você precisa gerenciar mudanças, comunicar-se com os envolvidos e usar ferramentas adequadas nas situações apropriadas. Tudo isso é discutido com detalhes mais adiante neste livro.

Quais são as características pessoais de um engenheiro de software competente?

Mas existem outras coisas igualmente importantes – os aspectos humanos que o tornarão um engenheiro de software competente. Erdogmus [Erd09] identifica sete características pessoais que estão presentes quando um engenheiro de software demonstra comportamento "super profissional".

Um engenheiro de software competente tem um senso de *responsabilidade individual*. Isso implica a determinação de cumprir suas promessas para colegas, para os envolvidos e para a gerência. Significa que ele fará o que precisar ser feito, quando for necessário, executando um esforço adicional para a obtenção de um resultado bem-sucedido.

Um engenheiro de software competente tem *consciência aguçada* das necessidades dos outros membros de sua equipe, dos envolvidos que solicitaram uma solução de software para um problema existente e dos gerentes que têm controle global sobre o projeto que vai gerar essa solução. É capaz de observar o ambiente em que as pessoas trabalham e de adaptar seu comportamento a ele e às próprias pessoas.

Um engenheiro de software competente é *extremamente honesto*. Se ele vê um projeto falho, aponta os defeitos de maneira construtiva, mas honesta. Se instado a distorcer fatos sobre cronogramas, recursos, desempenho ou outras características do produto ou projeto, opta por ser realista e sincero.

Um engenheiro de software competente mostra *resiliência sob pressão*. Conforme mencionamos anteriormente no livro, a engenharia de software está sempre à beira do caos. A pressão (e o caos que pode resultar) vem em muitas formas – mudanças nos requisitos e nas prioridades, envolvidos ou colegas exigentes, um gerente irrealista ou autoritário. Mas um engenheiro de software competente é capaz de suportar a pressão de modo que seu desempenho não seja prejudicado.

Um engenheiro de software competente tem *elevado senso de lealdade*. De boa vontade, compartilha os créditos com seus colegas. Tenta evitar conflitos de interesse e nunca age no sentido de sabotar o trabalho dos outros.

Um engenheiro de software competente mostra *atenção aos detalhes*. Isso não significa obsessão com a perfeição, mas sugere que ele considera atentamente as decisões técnicas que toma diariamente, em comparação com critérios mais amplos (por exemplo, desempenho, custo, qualidade) que foram estabelecidos para o produto e para o projeto.

Por último, um engenheiro de software competente é pragmático. Reconhece que a engenharia de software não é uma religião na qual devem ser seguidas regras dogmáticas, mas sim uma disciplina que pode ser adaptada de acordo com as circunstâncias.

6.2 A psicologia da engenharia de software

Em um artigo seminal sobre a psicologia da engenharia de software, Bill Curtis e Diane Walz [Cur90] sugerem um modelo comportamental em camadas para desenvolvimento de software (Figura 6.1). No nível individual, a psicologia da engenharia de software se concentra no reconhecimento do problema a ser resolvido, nas habilidades exigidas para solucionar o problema e na motivação necessária para concluir a solução dentro das restrições estabelecidas pelas camadas externas do modelo. Nos níveis da equipe e do projeto, dinâmicas de grupo se tornam o fator dominante. Aqui, a estrutura da equipe e fatores sociais governam o sucesso. A comunicação, a colaboração e a coordenação do grupo são tão importantes quanto as habilidades dos membros individuais da equipe. Nas camadas externas, o comportamento organizacional governa as ações da empresa e sua resposta para o meio empresarial.

No nível das equipes, Sawyer e seus colegas [Saw08] sugerem que elas frequentemente estabelecem fronteiras artificiais que reduzem a comunicação e, como consequência, a eficácia da equipe. Sugerem ainda um conjunto de "papéis que ultrapassam fronteiras", que permite que os membros de uma equipe de software transponham eficazmente suas fronteiras. Os papéis a seguir podem ser atribuídos explicitamente ou evoluir naturalmente.

Quais papéis os membros de uma equipe de software desempenham?

- *Embaixador* – representa a equipe para a clientela de fora, com o objetivo de negociar tempo e recursos e obter o retorno dos envolvidos.
- *Patrulha* – cruza a fronteira da equipe para reunir informações organizacionais. "O patrulhamento pode incluir o mapeamento de mercados externos, busca de novas tecnologias, identificação de atividades relevantes fora da equipe e descoberta de focos de concorrência em potencial" [Saw08].

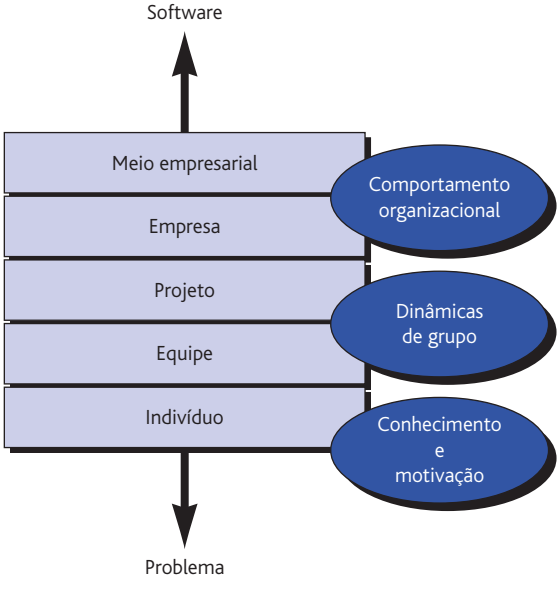

FIGURA 6.1 Um modelo comportamental em camadas para engenharia de software (adaptado de [Cur90]).

- *Guarda* – protege o acesso aos artefatos e outras informações da equipe.
- *Sentinela* – controla o fluxo de informações enviadas para a equipe pelos envolvidos e por outros.
- *Coordenador* – concentra-se na comunicação horizontal entre a equipe e dentro da organização (por exemplo, discutindo um problema de projeto específico com um grupo de especialistas da organização).

6.3 A equipe de software

Em seu livro clássico, *Peopleware,* Tom DeMarco e Tim Lister [DeM98] discutem a coesão de uma equipe de software:

> Há uma tendência em se utilizar a palavra *equipe* de forma constante e vaga na área de negócios, denominando qualquer grupo de profissionais designados para trabalharem juntos de "equipe". Entretanto, muitos deles não se assemelham a equipes. Não há uma definição comum de sucesso nem um espírito de equipe identificável. O que falta é um fenômeno que se denomina *consistência*.

O que é uma equipe "consistente"?

> Uma equipe consistente é um grupo de pessoas tão coesas, que o todo é maior que a soma das partes...
>
> Quando uma equipe começa a ser consistente, a probabilidade de sucesso aumenta muito. A equipe pode se tornar imbatível, um rolo compressor de sucesso... Não é preciso gerenciá-la do modo tradicional e, com certeza, não precisará ser motivada. Ela adquire velocidade e ímpeto.

DeMarco e Lister sustentam que os membros de equipes consistentes são significativamente mais produtivos e mais motivados do que a média. Compartilham de um objetivo comum, de uma cultura comum e, em muitos casos, de um senso de pertencimento a uma equipe de elite que os torna únicos.

Não existe nenhum método infalível para se criar uma equipe consistente. Porém, existem atributos normalmente encontrados em equipes de software eficazes.[1] Miguel Carrasco [Car08] sugere que uma equipe de software eficiente deve estabelecer um *senso de propósito*. Por exemplo, se todos os membros da equipe concordam que o objetivo dela é desenvolver software que vai transformar uma categoria de produto e, como consequência, transformar sua empresa em líder do setor, eles têm um forte senso de propósito. Uma equipe eficaz também deve incorporar um *senso de envolvimento* que permita a cada membro sentir que suas qualidades e contribuições são valiosas.

Uma equipe eficaz deve promover um *senso de confiança*. Os engenheiros de software da equipe devem confiar nas habilidades e na competência de seus colegas e gerentes. A equipe deve estimular um *senso de melhoria,* refletindo periodicamente em sua abordagem de engenharia de software e buscando maneiras de melhorar seu trabalho.

[1] Bruce Tuckman observa que as equipes bem-sucedidas passam por quatro fases (Formação, Ataque, Regulamentação e Execução) no caminho para se tornarem produtivas (**http://www.realsoftwaredevelopment.com/7-key-attributes-of-high-performance-software-development-teams/**).

Uma equipe de software eficaz é diversificada, preenchida por pessoas que têm senso de propósito, envolvimento, confiança e melhoria.

As equipes de software mais eficazes são diversificadas, no sentido de combinarem uma variedade de diferentes qualidades. Técnicos altamente capacitados são complementados por membros que podem ter menos base técnica, mas compreendem melhor as necessidades dos envolvidos.

Por que as equipes não conseguem ser consistentes?

Porém, nem todas as equipes são eficazes e nem todas são consistentes. Na verdade, muitas sofrem do que Jackman [Jac98] denomina de "toxicidade de equipe". Ela define cinco fatores que "promovem um ambiente em equipe potencialmente tóxico": uma atmosfera de trabalho frenética; alto grau de frustração que causa atrito entre os membros da equipe; um processo de software fragmentado ou coordenado de forma deficiente; uma definição nebulosa dos papéis dentro da equipe de software; e contínua e repetida exposição a falhas.

Para evitar um ambiente de trabalho frenético, a equipe deve ter acesso a todas as informações exigidas para cumprir a tarefa. As principais metas e objetivos, uma vez definidos, não devem ser modificados, a menos que seja absolutamente necessário. Uma equipe pode evitar frustrações se lhe for oferecida, tanto quanto possível, responsabilidade para tomada de decisão. Um processo inapropriado (por exemplo, tarefas onerosas ou desnecessárias ou artefatos mal selecionados) pode ser evitado por meio da compreensão do produto a ser desenvolvido, das pessoas que realizam o trabalho e pela permissão para que a equipe selecione o modelo do processo. A própria equipe deve estabelecer seus mecanismos de responsabilidades (revisões técnicas[2] são excelentes meios para conseguir isso) e definir uma série de abordagens corretivas quando um membro falhar em suas atribuições. E, por fim, a chave para evitar uma atmosfera de derrota consiste em estabelecer técnicas baseadas no trabalho em equipe voltadas para realimentação (feedback) e solução de problemas.

"Nem todo grupo é uma equipe, nem toda equipe é eficaz."

Glenn Parker

Somando-se às cinco toxinas descritas por Jackman, uma equipe de software frequentemente despende esforços com as diferentes características de seus membros. Uns são extrovertidos; outros, introvertidos. Uns coletam informações intuitivamente, destilando conceitos amplos de fatos disparatados. Outros processam informações linearmente, coletando e organizando detalhes minuciosos dos dados fornecidos. Alguns se sentem confortáveis tomando decisões apenas quando um argumento lógico e ordenado for apresentado. Outros são intuitivos, acostumados a tomar decisões baseadas em percepções. Certos desenvolvedores querem um cronograma detalhado, preenchido por tarefas organizadas que os tornem aptos a ter proximidade com elementos do projeto. Outros, ainda, preferem um ambiente mais espontâneo, no qual resultados e questões abertas são aceitáveis. Alguns trabalham arduamente para conseguir que as etapas sejam concluídas bem antes da data estabelecida, evitando, portanto, estresse na medida em que a data-limite se aproxima, enquanto outros são energizados pela correria em fazer até o último minuto da data-limite. Reconhecer as diferenças humanas, junto com outras diretrizes apresentadas nesta seção, proporciona uma maior probabilidade de se criar equipes consistentes.

[2] Revisões técnicas são tratadas em detalhes no Capítulo 20.

6.4 Estruturas de equipe

A melhor estrutura de equipe depende do estilo de gerenciamento das organizações, da quantidade de pessoas na equipe e seus níveis de habilidade e do grau de dificuldade geral do problema. Mantei [Man81] descreve vários fatores que devem ser considerados ao planejarmos a estrutura da equipe de engenharia de software: dificuldade do problema a ser resolvido; "tamanho" do programa (ou programas) resultante em linhas de código ou pontos de função;[3] tempo que a equipe irá permanecer reunida (tempo de vida da equipe); até que ponto o problema pode ser modularizado; qualidade e confiabilidade exigidas do sistema a ser construído; rigidez da data de entrega; e grau de sociabilidade (comunicação) exigida para o projeto.

Constantine [Con93] sugere quatro "paradigmas organizacionais" para equipes de engenharia de software:

Quais opções temos ao definir a estrutura de uma equipe de software?

1. O *paradigma fechado* estrutura uma equipe em termos de uma hierarquia de autoridade tradicional. Tais equipes podem trabalhar bem em produção de software bastante similar a esforços já feitos no passado, mas se mostrarão menos propícias a ser inovadoras trabalhando sob o paradigma fechado.
2. O *paradigma randômico* estrutura uma equipe de forma mais livre e depende da iniciativa individual de seus membros. Quando for necessária uma inovação ou um avanço tecnológico, as equipes que seguem o paradigma randômico se destacarão. Mas essas equipes podem brigar quando for exigido um "desempenho ordenado".

Quais fatores devem ser considerados ao se escolher a estrutura de uma equipe de software?

3. O *paradigma aberto* procura estruturar a equipe de maneira que consiga alguns dos controles associados ao paradigma fechado, mas também muito da inovação que ocorre ao se usar o paradigma randômico. O trabalho é feito de forma colaborativa, com forte comunicação e tomada de decisão baseada no consenso – características marcantes das equipes de paradigma aberto. As estruturas das equipes de paradigmas abertos são bem adequadas para a solução de problemas complexos, mas não conseguem desempenhar tão eficientemente quanto outras equipes.
4. O *paradigma sincronizado* baseia-se na compartimentalização natural de um problema e organiza os membros da equipe para trabalhar nas partes do problema com pouca comunicação entre si.

"Se deseja ser incrementalmente melhor, seja competitivo. Se deseja ser exponencialmente melhor, seja cooperativo."

Autor desconhecido

Como um comentário histórico final, uma das mais antigas organizações de equipe de software foi uma estrutura de paradigma fechado, denominada originalmente *equipe com um programador-chefe* (principal). Essa estrutura foi primeiramente proposta por Harlan Mills e descrita por Baker [Bak72]. O núcleo da equipe era composto de um *engenheiro sênior* (o programador-chefe), que planejava, coordenava e fazia a revisão de todas as atividades técnicas da equipe, o *pessoal técnico* (normalmente de duas a cinco pessoas), que conduzia as atividades de análise e de desenvolvimento, e um *engenheiro re-*

[3] Linhas de código (LOC, lines of code) e pontos de função são medidas do tamanho de um programa de computador e são discutidas no Capítulo 33.

serva que dava suporte ao engenheiro sênior em suas atividades e que podia substituí-lo com perdas mínimas de continuidade. O programador-chefe (principal) podia ter a seu dispor um ou mais especialistas (por exemplo, perito em telecomunicações, desenvolvedor de banco de dados), uma equipe de suporte (por exemplo, codificadores técnicos, pessoal de escritório) e um bibliotecário de software.

Como contraponto à estrutura da equipe de programadores-chefe, o paradigma randômico de Constantine [Con93] sugere a primeira equipe criativa, cuja abordagem de trabalho poderia ser mais bem denominada *anarquia inovadora*. Embora a abordagem de espírito livre para o trabalho de software seja atraente, a energia da criatividade direcionada para uma equipe de alta performance deve ser o objetivo central de uma organização de engenharia de software.

CASASEGURA

Estrutura da equipe

Cena: Escritório de Doug Miller antes do início do projeto do software *CasaSegura*.

Atores: Doug Miller (gerente da equipe de engenharia de software do *CasaSegura*) e Vinod Raman, Jamie Lazar e outros membros da equipe.

Conversa:

Doug: Vocês deram uma olhada no informativo preliminar do *CasaSegura* que o departamento de marketing preparou?

Vinod (balançando afirmativamente a cabeça e olhando para seus companheiros de equipe): Sim, mas temos muitas dúvidas.

Doug: Vamos deixar isso de lado por um momento. Gostaria de conversar sobre como vamos estruturar a equipe, quem será responsável pelo quê...

Jamie: Estou totalmente de acordo com a filosofia ágil, Doug. Acho que devemos ser uma equipe auto-organizada.

Vinod: Concordo. Devido ao cronograma apertado e ao grau de incertezas e pelo fato de todos sermos realmente competentes (risos), parece ser o caminho certo a tomar.

Doug: Tudo bem por mim, mas vocês conhecem o procedimento.

Jamie (sorridente e falando ao mesmo tempo): Tomamos decisões táticas sobre quem faz o que e quando, mas é nossa responsabilidade ter o produto pronto sem atraso.

Vinod: E com qualidade.

Doug: Exatamente. Mas lembrem-se de que há restrições. O marketing define os incrementos de software a serem desenvolvidos – consultando-nos, é claro.

Jamie: E...?

6.5 Equipes ágeis

Ao longo da última década, o desenvolvimento de software ágil (Capítulo 5) tem sido indicado como o antídoto para muitos problemas que se alastraram nas atividades de projeto de software. Relembrando, a filosofia ágil enfatiza a satisfação do cliente e a entrega prévia incremental de software, pequenas equipes de projeto altamente motivadas, métodos informais, mínimos artefatos de engenharia de software e total simplicidade de desenvolvimento.

6.5.1 A equipe ágil genérica

A pequena e altamente motivada equipe de projeto, também denominada *equipe ágil*, adota muitas das características das equipes de software bem-sucedidas, discutidas na seção anterior, e evita muito das toxinas geradoras

Uma equipe ágil é auto-organizada e tem autonomia para planejar e tomar decisões técnicas.

de problemas. Entretanto, a filosofia ágil enfatiza a competência individual (membro da equipe) combinada com a colaboração em grupo como fatores críticos de sucesso para a equipe. Cockburn e HighSmith [Coc01a] observam isso ao escreverem:

> Se as pessoas do projeto forem boas o suficiente, podem usar praticamente qualquer processo e cumprir sua missão. Se não forem boas o suficiente, nenhum processo irá reparar a sua inadequação. "Pessoas são o trunfo do processo" é uma forma de dizer isso. Entretanto, falta de suporte ao desenvolvedor e ao usuário pode acabar com um projeto – "política é o trunfo de pessoas". Suporte inadequado pode fazer com que até mesmo os bons fracassem na realização de seus trabalhos.

"Propriedades coletivas nada mais são do que uma instância da ideia de que os produtos deveriam ser atribuídos à equipe (ágil), não a indivíduos que compõem a equipe."

Jim Highsmith

Para uso das competências de cada membro da equipe, para fomentar colaboração efetiva ao longo do projeto, equipes ágeis são auto-organizadas. Uma equipe auto-organizada não mantém, necessariamente, uma estrutura de equipe única, mas usa elementos da aleatoriedade de Constantine, paradigmas abertos e de sincronicidade discutidos na Seção 6.2.

Muitos modelos ágeis de processo (por exemplo, Scrum) dão à equipe ágil autonomia para gerenciar o projeto e tomar as decisões técnicas necessárias à conclusão do trabalho. O planejamento é mantido em um nível mínimo, e a equipe tem a permissão para escolher sua própria abordagem (por exemplo, processo, método, ferramentas), limitada somente pelos requisitos de negócio e pelos padrões organizacionais. Conforme o projeto prossegue, a equipe se auto-organiza, concentrando-se em competências individuais para maior benefício do projeto em um determinado ponto do cronograma. Para tanto, uma equipe ágil pode fazer reuniões de equipe diariamente a fim de coordenar e sincronizar as atividades que devem ser realizadas naquele dia.

Com base na informação obtida durante essas reuniões, a equipe adapta sua abordagem para incrementar o trabalho. A cada dia que passa, auto-organizações contínuas e colaboração conduzem a equipe em direção a um incremento de software completo.

6.5.2 A equipe XP

Simplifique sempre que puder, mas reconheça que uma "refabricação" (retrabalho, redesenvolvimento) contínua pode absorver tempo e recursos significativos.

Beck [Bec04a] define um conjunto de cinco *valores* que estabelecem as bases para todo trabalho realizado como parte da programação extrema (XP) – comunicação, simplicidade, feedback (realimentação ou retorno), coragem e respeito. Cada um desses valores é usado como um direcionamenro para as atividades, ações e tarefas específicas da XP.

Para conseguir a *comunicação* efetiva entre a equipe ágil e outros envolvidos (por exemplo, estabelecer as funcionalidades necessárias para o software), a XP enfatiza a colaboração estreita, embora informal (verbal), entre clientes e desenvolvedores, o estabelecimento de metáforas[4] eficazes para comunicar conceitos importantes, feedback (realimentação) contínuo e evita documentação volumosa como um meio de comunicação.

[4] No contexto da XP, *metáfora* é "uma história que todos – clientes, programadores e gerentes – podem contar sobre como o sistema funciona" [Bec04a].

Para obter a *simplicidade,* a equipe ágil projeta apenas para as necessidades imediatas, em vez de considerar as necessidades futuras. O objetivo é criar um projeto simples que possa ser facilmente implementado em código. Se o projeto tiver que ser melhorado, ele poderá ser *refabricado*[5] mais tarde.

O feedback provém de três fontes: do próprio software implementado, do cliente e de outros membros da equipe de software. Por meio da elaboração do projeto e da implementação de uma estratégia de testes eficaz (Capítulos 22 a 26), o software (via resultados de testes) propicia um feedback para a equipe ágil. A equipe faz uso do *teste unitário* como tática de teste principal. À medida que cada classe é desenvolvida, a equipe desenvolve um teste unitário para testar cada operação de acordo com a funcionalidade especificada. À medida que um incremento é entregue a um cliente, as *histórias de usuários* ou *casos de uso* (Capítulo 9) implementados pelo incremento são utilizados para realizar os testes de aceitação. O grau em que o software implementa o produto, a função e o comportamento do caso em uso é uma forma de feedback. Por fim, conforme novas necessidades surgem como parte do planejamento iterativo, a equipe dá ao cliente um rápido feedback referente ao impacto nos custos e no cronograma.

"A XP é a resposta para a pergunta: 'Qual é o mínimo possível que se pode realizar e mesmo assim desenvolver um software excelente?'."

Anônimo

Beck [Bec04a] afirma que a obediência estrita a certas práticas da XP exige *coragem*. Uma palavra melhor poderia ser *disciplina*. Por exemplo, frequentemente, há uma pressão significativa para a elaboração do projeto pensando em futuros requisitos. A maioria das equipes de software sucumbe, argumentando que "projetar para amanhã" poupará tempo e esforço no longo prazo. Uma equipe XP deve ter disciplina (coragem) para projetar para hoje, reconhecendo que as necessidades futuras podem mudar significativamente, exigindo, consequentemente, um retrabalho substancial em relação ao projeto e ao código implementado.

Ao buscar cada um desses valores, a equipe XP fomenta *respeito* entre seus membros, entre outros envolvidos e os membros da equipe e, indiretamente, para o próprio software. Conforme consegue entregar com sucesso incrementos de software, a equipe desenvolve cada vez mais respeito pelo processo XP.

6.6 O impacto da mídia social

E-mail, mensagens de texto e videoconferência se tornaram atividades onipresentes no trabalho de engenharia de software. Mas, na verdade, esses mecanismos de comunicação nada mais são do que substitutos ou suplementos modernos para o contato face a face. A mídia social é diferente.

[5] A refabricação permite que o engenheiro de software aperfeiçoe a estrutura interna de um projeto (ou código-fonte) sem alterar sua funcionalidade ou comportamento externos. Basicamente, a refabricação pode ser usada para melhorar a eficiência, a legibilidade ou o desempenho de um projeto ou o código que implementa um projeto.

Begel [Beg10] e seus colegas tratam do crescimento e da aplicação de mídia social na engenharia de software ao escreverem:

> Os processos sociais em torno do desenvolvimento de software são... altamente dependentes da capacidade dos engenheiros de encontrar e associar-se a pessoas que compartilhem objetivos semelhantes e habilidades complementares para harmonizar a comunicação e as preferências de cada membro da equipe, colaborar e coordenar durante todo o ciclo de vida do software e defender o sucesso de seu produto no mercado.

De certa forma, essa "associação" pode ser tão importante quanto a comunicação face a face. O valor da mídia social cresce à medida que o tamanho da equipe aumenta, e é ainda mais ampliado quando a equipe está geograficamente dispersa.

Primeiro, é definida uma rede social para um projeto de software. Usando a rede, a equipe de software pode se basear na experiência coletiva de seus membros, dos envolvidos, dos técnicos, especialistas e outros executivos que tenham sido convidados a participar da rede (se for privada) ou de qualquer parte interessada (se for pública). E isso pode acontecer quando surgir uma questão, uma dúvida ou um problema. Existem várias formas diferentes de mídia social, e cada uma ocupa seu lugar no trabalho de engenharia de software.

"Se conteúdo é rei, a conversa é rainha."

John Munsell

Um *blog* pode ser usado para postar uma série de artigos breves, descrevendo aspectos importantes de um sistema ou expressando opiniões sobre recursos ou funções que ainda precisam ser desenvolvidos. Também é importante observar que "as empresas de software frequentemente usam blogs para compartilhar informações técnicas e opiniões com seus funcionários e, muito proveitosamente, com seus clientes, tanto internos quanto externos" [Beg10].

Microblogs (como o Twitter) permitem que um membro de uma rede de engenharia de software poste mensagens breves para seus seguidores. Como as mensagens são instantâneas e podem ser lidas a partir de todas as plataformas móveis, a dispersão da informação se dá quase em tempo real. Isso permite a uma equipe de software convocar uma reunião inesperada caso surja uma questão, solicitar ajuda especializada se ocorrer um problema ou informar os envolvidos sobre algum aspecto do projeto.

Fóruns online dirigidos permitem aos participantes postar perguntas, opiniões, estudos de caso ou qualquer outra informação relevante. Uma pergunta técnica pode ser postada e, em poucos minutos, frequentemente várias "respostas" estão disponíveis.

Sites de rede social (como Facebook, LinkedIn) permitem ligações por laços de amizade entre desenvolvedores de software e técnicos próximos. Isso permite aos "amigos" em um site de rede social conhecer amigos de amigos que podem ter conhecimento ou especialidade relacionada ao domínio de aplicação ou problema a ser resolvido. Redes privadas especializadas, baseadas no paradigma das redes sociais, podem ser usadas dentro de uma organização.

Grande parte das mídias sociais permite a formação de "comunidades" de usuários com interesses semelhantes. Por exemplo, uma comunidade de engenheiros de software especializados em sistemas embarcados em tempo real poderia ser uma maneira interessante para uma pessoa ou equipe que esteja

trabalhando nessa área estabelecer relações que melhorariam seu trabalho. À medida que uma comunidade cresce, os participantes discutem tendências tecnológicas, cenários de aplicação, novas ferramentas e outros conhecimentos da engenharia de software. Por fim, os *sites de social bookmarking* (como Delicious, Stumble, CiteULike) permitem a um engenheiro ou equipe de software recomendar recursos baseados na Web que podem ser interessantes para uma comunidade de mídia social de pessoas com a mesma opinião.

É muito importante mencionar que questões de privacidade e segurança não devem ser desprezadas ao se usar mídia social no trabalho de engenharia de software. Grande parte do trabalho realizado por engenheiros de software pode ser propriedade de seus empregadores, e a divulgação poderia ser muito prejudicial. Por isso, os benefícios da mídia social devem ser ponderados em relação à possibilidade de revelação descontrolada de informações privativas.

6.7 Engenharia de software usando a nuvem

"Eles não a chamam mais de Internet, chamam de computação em nuvem. Não vou mais me opor ao nome. Chame-a como quiser."

Larry Ellison

A computação em nuvem oferece um mecanismo de acesso a todos os produtos, artefatos e informações relacionados ao projeto da engenharia de software. Ela funciona em qualquer lugar e elimina a dependência de dispositivos, o que já foi uma restrição para muitos projetos de software. Também permite que os membros de uma equipe de software façam testes de baixo risco e independentes de plataforma de novas ferramentas de software e forneçam feedback sobre elas. Permite ainda novas possibilidades para a distribuição e testes de software beta. Ela oferece o potencial de abordagens aprimoradas de gerenciamento de conteúdo e configuração (Capítulo 29).

Como a computação em nuvem pode fazer essas coisas, ela tem o potencial de influenciar o modo como os engenheiros de software organizam suas equipes, como realizam seu trabalho, como se comunicam e se associam e o modo de gerenciar projetos de software. As informações de engenharia de software desenvolvidas por um membro da equipe podem estar instantaneamente disponíveis para todos os membros, independentemente da plataforma que os outros estejam usando ou de sua localização.

A nuvem é um poderoso repositório de informações sobre engenharia de software, mas você deve considerar as questões de controle de alteração discutidas no Capítulo 29.

Basicamente, a dispersão da informação é significativamente acelerada e ampliada. Isso muda a dinâmica da engenharia de software e pode ter um impacto profundo em seus aspectos humanos.

Porém, a computação em nuvem em um ambiente de engenharia de software tem seus riscos [The13]. A nuvem é dispersa por muitos servidores, e a arquitetura e os serviços frequentemente estão fora do controle de uma equipe de software. Como consequência, existem vários pontos de falha, apresentando riscos à confiabilidade e à segurança. À medida que o número de serviços fornecidos pela nuvem aumenta, a complexidade relativa do ambiente de desenvolvimento de software também aumenta. Cada um desses serviços funciona bem com outros serviços, possivelmente de outros fornecedores? Isso apresenta um risco à capacidade de operação conjunta para serviços da nuvem. Por último, se a nuvem se torna o ambiente de desenvolvimento, os serviços devem enfatizar a usabilidade e o desempenho. Às vezes, esses atributos entram em conflito com a segurança, privacidade e confiabilidade.

Porém, do ponto de vista humano, a nuvem oferece bem mais benefícios do que riscos para os engenheiros de software. Dana Gardner [Gar09] resume os benefícios (com um alerta):

> Tudo que está relacionado aos aspectos sociais ou colaborativos do desenvolvimento de software serviu bem para a nuvem. Gerenciamento de projeto, cronogramas, listas de tarefas (checklists), requisitos e gerenciamento de defeitos; tudo convém, pois estão no grupo principal das funções, no qual a comunicação é essencial para manter os projetos sincronizados e todos os membros da equipe – onde quer que estejam – literalmente no mesmo canal. Evidentemente, há uma importante advertência aqui – se sua empresa projeta software embarcado nos produtos, ele não é um bom candidato para a nuvem: imagine pôr as mãos nos planos de projeto da Apple para a próxima versão do iPhone.

Como Gardner declara, uma das principais vantagens da nuvem é sua capacidade de melhorar os "aspectos sociais e colaborativos do desenvolvimento de software". Na próxima seção, você vai saber um pouco mais sobre ferramentas colaborativas.

6.8 Ferramentas de colaboração

Fillipo Lanubile e seus colegas [Lan10] sugerem que os ambientes de desenvolvimento de software (SDEs, software development environments) do último século se transformaram em *ambientes de desenvolvimento colaborativo* (CDEs; *collaborative development environments*).[6] Eles declaram:

> Ferramentas são essenciais para a colaboração entre os membros da equipe, permitindo a facilitação, automação e controle do processo de desenvolvimento inteiro. O suporte de ferramentas adequado é particularmente necessário na engenharia de software global, pois a distância agrava os problemas de coordenação e controle, direta ou indiretamente, por meio de seus efeitos negativos sobre a comunicação.

Muitas das ferramentas usadas em um CDE não são diferentes das usadas para ajudar nas atividades de engenharia de software discutidas nas Partes II, III e IV do livro. Mas um CDE digno de consideração também fornece um conjunto de serviços especificamente projetados para melhorar o trabalho colaborativo [Fok10]. Esses serviços incluem:

Quais serviços genéricos são encontrados em ambientes de desenvolvimento colaborativo?

- Um repositório com a identificação clara do projeto que permita a equipe de projeto armazenar todos os artefatos e outras informações de um modo que melhore a segurança e a privacidade, permitindo o acesso apenas a pessoas autorizadas.
- Um *calendário* para coordenar reuniões e outros eventos do projeto.
- *Modelos* que permitam aos membros da equipe criar artefatos com aparência e estrutura padronizados.
- *Suporte de métricas* que monitorem as contribuições de cada membro da equipe de maneira quantitativa.

[6] O termo *ambiente de desenvolvimento colaborativo* (CDE) foi cunhado por Grady Booch [Boo02].

- *Análise de comunicação* que monitore a comunicação entre a equipe e isole padrões que possam significar problemas ou questões que precisam ser resolvidos.
- *Agrupamento de artefatos* que organize os artefatos e outros produtos do projeto de uma maneira que responda a perguntas como: "O que causou uma mudança em particular, quem conhece um produto específico que possivelmente deve ser consultado sobre alterações nele e como o trabalho de um membro [equipe] poderia afetar o de outras pessoas?" [Fok10].

FERRAMENTAS DO SOFTWARE

Ambientes de desenvolvimento colaborativo

Objetivo: À medida que o desenvolvimento de software se torna global, as equipes de software precisam de mais ferramentas. Elas precisam de um conjunto de serviços que permitam aos membros da equipe colaborar de forma local e em longas distâncias.

Mecanismos: Ferramentas e serviços dessa categoria permitem a uma equipe estabelecer mecanismos para trabalho colaborativo. Um CDE implementará muitos ou todos os serviços descritos na Seção 6.6, ao mesmo tempo fornecendo acesso às ferramentas de engenharia de software convencionais para gerenciamento de processo (Capítulo 4) discutidas ao longo deste livro.

Ferramentas representativas:[7]

GForge – um ambiente colaborativo que contém recursos de gerenciamento de projeto e código **(http://gforge.com/gf/)**.

OneDesk – oferece um ambiente colaborativo que cria e gerencia uma área de trabalho de projetos para desenvolvedores e envolvidos **(www.onedesk.com)**.

Rational Team Concert – um sistema detalhado de gerenciamento de ciclo de vida colaborativo **(http://www-01.ibm.com/software/rational/products/rtc/)**.

6.9 Equipes globais

"Cada vez mais, em qualquer empresa, os gerentes lidam com diferentes culturas. As empresas estão se tornando globais, mas as equipes estão sendo divididas e espalhadas por todo o planeta."

Carlos Ghosn, Nissan

No campo do software, globalização significa mais do que a transferência de bens e serviços entre fronteiras internacionais. Nas últimas décadas, foi construído um crescente número de importantes produtos de software, por equipes muitas vezes sediadas em diferentes países. Essas equipes de desenvolvimento de software global (GSD, global software development) têm muitas das características de uma equipe de software convencional (Seção 6.4), mas uma equipe GSD tem outros desafios exclusivos, que incluem coordenação, colaboração, comunicação e tomada de decisão especializada. Abordagens de coordenação, colaboração e comunicação foram discutidas anteriormente neste capítulo. A tomada de decisão em todas as equipes de software é complicada por quatro fatores [Gar10]:

- Complexidade do problema.
- Incerteza e risco associados à decisão.
- A lei das consequências não intencionais (isto é, uma decisão associada ao trabalho tem um efeito colateral sobre outro objetivo do projeto).
- Diferentes visões do problema que levam a diferentes conclusões sobre o caminho a seguir.

[7] A citação de ferramentas aqui não representa um endosso, mas sim uma amostragem das ferramentas nessa categoria. Na maioria dos casos, seus nomes são marcas registradas pelos respectivos desenvolvedores.

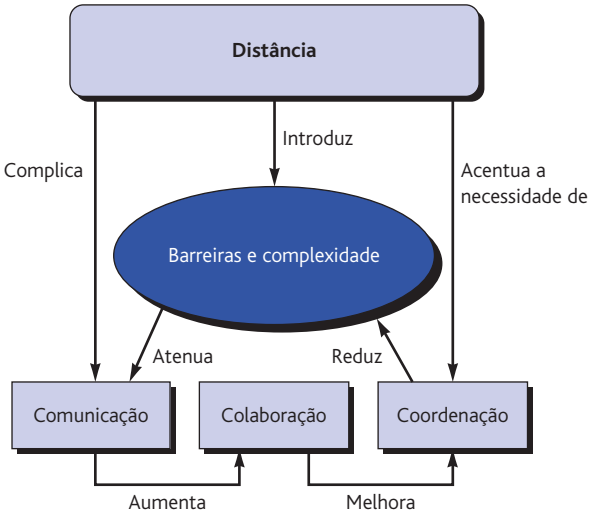

FIGURA 6.2 Fatores que afetam uma equipe GSD (adaptado de [Cas06]).

Para uma equipe GSD, os desafios associados à coordenação, colaboração e comunicação podem ter um efeito profundo na tomada de decisão. A Figura 6.2 ilustra o impacto da distância sobre os desafios enfrentados por uma equipe GSD. A distância complica a comunicação, mas, ao mesmo tempo, acentua a necessidade de coordenação. A distância também introduz barreiras e complexidade, motivadas por diferenças culturais. Barreiras e complexidade enfraquecem a comunicação (isto é, a relação sinal-ruído diminui). Os problemas inerentes a essa dinâmica podem resultar em um projeto instável.

Embora não exista nenhuma solução milagrosa que possa corrigir totalmente as relações indicadas na Figura 6.2, o uso de CDEs eficazes (Seção 6.6) pode ajudar a reduzir o impacto da distância.

6.10 Resumo

Um engenheiro de software de sucesso precisa ter habilidades técnicas. Além disso, deve assumir a responsabilidade por seus compromissos, estar ciente das necessidades de seus colegas, ser honesto em sua avaliação do produto e do projeto, mostrar resiliência sob pressão, tratar seus colegas de modo correto e mostrar atenção aos detalhes.

A psicologia da engenharia de software engloba o conhecimento e a motivação individuais, a dinâmica de grupo de uma equipe de software e o comportamento organizacional da empresa. Para melhorar a comunicação e a colaboração, os membros de uma equipe de software podem assumir papéis que ultrapassam fronteiras.

Uma equipe de software bem-sucedida (“consistente”) é mais produtiva e motivada do que a média. Para ser eficaz, uma equipe de software deve ter senso de propósito, envolvimento, confiança e melhoria. Além disso, a equipe deve evitar a “toxicidade” caracterizada por uma atmosfera de trabalho frenética e frustrante, um processo de software inadequado, uma definição nebulosa dos papéis na equipe e contínua exposição a falhas.

Existem muitas estruturas de equipe diferentes. Algumas equipes são organizadas hierarquicamente, enquanto outras preferem uma estrutura livre que conte com a iniciativa individual. As equipes ágeis endossam a filosofia ágil e geralmente têm mais autonomia do que as equipes de software mais convencionais. As equipes ágeis enfatizam a comunicação, simplicidade, feedback, coragem e respeito.

A mídia social está se tornando parte de muitos projetos de software. Blogs, microblogs, fóruns e recursos de rede social ajudam a formar uma comunidade de engenharia de software que se comunica e é coordenada de modo mais eficaz.

A computação em nuvem tem o potencial de influenciar o modo como os engenheiros de software organizam suas equipes, como realizam seu trabalho, como se comunicam e se associam e o modo de gerenciar projetos de software. Em situações nas quais a nuvem pode melhorar os aspectos sociais e colaborativos do desenvolvimento de software, seus benefícios superam em muito os riscos.

Os ambientes de desenvolvimento colaborativo contêm vários serviços que melhoram a comunicação e a colaboração de uma equipe de software. Esses ambientes são particularmente úteis para desenvolvimento de software global, em que a separação geográfica pode criar barreiras para uma engenharia de software bem-sucedida.

Problemas e pontos a ponderar

6.1 Com base em sua própria observação de pessoas que são excelentes desenvolvedoras de software, cite três qualidades da personalidade que pareçam ser comuns entre elas.

6.2 Como você pode ser "extremamente honesto" e ainda não ser percebido (pelos outros) como insultante ou agressivo?

6.3 Como uma equipe de software constrói "fronteiras artificiais" que reduzem sua capacidade de se comunicar com outros?

6.4 Escreva um breve cenário descrevendo cada um dos "papéis que ultrapassam fronteiras" estudadas na Seção 6.2.

6.5 Na Seção 6.3, mencionamos que sensos de propósito, envolvimento, confiança e melhoria são atributos fundamentais para equipes de software eficazes. Quem é responsável por instilar esses atributos quando uma equipe é formada?

6.6 Qual dos quatro paradigmas organizacionais para equipes (Seção 6.4) você acha o mais eficaz (a) para o departamento de TI em uma grande companhia de seguros; (b) para um grupo de engenharia de software em um importante fornecedor militar; (c) para um grupo de software que constrói jogos de computador; (d) para uma grande empresa de software? Explique o motivo das escolhas que você fez.

6.7 Se você tivesse de escolher um atributo de uma equipe ágil que a tornasse diferente de uma equipe de software convencional, qual seria?

6.8 Das formas de mídia social descritas para o trabalho de engenharia de software na Seção 6.6, qual você acha a mais eficaz e por quê?

6.9 Escreva um cenário no qual os membros da equipe do *CasaSegura* utilizam uma ou mais formas de mídia social como parte de seu projeto de software.

6.10 Atualmente, a nuvem é um dos conceitos mais badalados no mundo da computação. Descreva como ela pode agregar valor para uma organização de engenharia de software, com referência específica aos serviços especialmente destinados a melhorar o trabalho de engenharia de software.

6.11 Pesquise uma das ferramentas de CDE mencionadas no quadro da Seção 6.8 (ou uma ferramenta designada por seu professor) e prepare uma breve apresentação de seus recursos para sua classe.

6.12 Com referência à Figura 6.2, por que a distância complica a comunicação? Por que acentua a necessidade de coordenação? Por que tipos de barreiras e complexidade são introduzidos pela distância?

Leituras e fontes de informação complementares

Embora muitos livros tenham tratado dos aspectos humanos da engenharia de software, dois deles podem ser legitimamente chamados "clássicos". Jerry Weinberg (*The Psychology of Computer Programming,* Silver Anniversary Edition, Dorset House, 1998) foi o primeiro a considerar a psicologia das pessoas que constroem software de computador. Tom DeMarco e Tim Lister (*Peopleware: Productive Projects and Teams,* 2ª ed., Dorset House, 1999) argumentam que os principais desafios no desenvolvimento de software são humanos, não técnicos.

Observações interessantes sobre os aspectos humanos da engenharia de software também foram feitas por Mantle e Lichty (*Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams*, Addison-Wesley, 2012), Fowler (*The Passionate Programmer,* Pragmatic Bookshelf, 2009), McConnell (*Code Complete,* 2ª ed., Microsoft Press, 2004), Brooks (*The Mythical Man-Month,* 2ª ed., Addison-Wesley, 1999) e Hunt e Thomas (*The Pragmatic Programmer,* Addison-Wesley, 1999). Tomayko e Hazzan *(Human Aspects of Software Engineering,* Charles River Media, 2004) tratam da psicologia e da sociologia da engenharia de software, com ênfase em XP.

Os aspectos humanos do desenvolvimento ágil foram tratados por Rasmussen (*The Agile Samurai,* Pragmatic Bookshelf, 2010) e Davies (*Agile Coaching,* Pragmatic Bookshelf, 2010). Importantes aspectos das equipes ágeis são considerados por Adkins (*Coaching Agile Teams,* Addison-Wesley, 2010) e Derby, Larsen e Schwaber (*Agile Retrospectives: Making Good Teams Great,* Pragmatic Bookshelf, 2006).

A solução de problemas é uma atividade exclusivamente humana e é tratada em livros de Adair (*Decision Making and Problem Solving Strategies,* Kogan Page, 2010), Roam (*Unfolding the Napkin,* Portfolio Trade, 2009) e Wananabe (*Problem Solving 101,* Portfolio Hardcover, 2009).

Diretrizes para facilitar a colaboração dentro de uma equipe de software são apresentadas por Tabaka (*Collaboration Explained,* Addison-Wesley, 2006). Rosen (*The Culture of Collaboration,* Red Ape Publishing, 2009), Hansen (*Collaboration*, Harvard Business School Press, 2009) e Sawyer (*Group Genius: The Creative Power of Collaboration,* Basic Books, 2007) apresentam estratégias e diretrizes práticas para melhorar a colaboração em equipes técnicas.

Promover a inovação humana é o tema de livros de Gray, Brown e Macanufo (*Game Storming,* O-Reilly Media, 2010), Duggan (*Strategic Intuition,* Columbia University Press, 2007) e Hohmann (*Innovation Games,* Addison-Wesley, 2006).

Uma visão geral do desenvolvimento de software global é apresentada por Ebert *(Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing,* Wiley-IEEE Computer Society Press, 2011). Mite e seus colegas (*Agility Across Time and Space: Implementing Agile Methods in Global Software Projects,* Springer, 2010) editaram uma antologia que trata do uso de equipes ágeis no desenvolvimento global.

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