5 Desenvolvimento ágil

Conceitos-chave

agilidade 68
Agile Alliance 70
custo da alteração 68
Método de Desenvolvimento de Sistemas Dinâmicos (DSDM, Dynamic Systems Development Method) . . 79
princípios da agilidade . . . 70
processo ágil 69
Processo Unificado Ágil . . 82
testes de aceitação 75

Em 2001, Kent Beck e outros 16 renomados desenvolvedores, autores e consultores da área de software [Bec01] (batizados de "Agile Alliance" – "Aliança dos Ágeis") assinaram o "Manifesto para o Desenvolvimento Ágil de Software" ("Manifesto for Agile Software Development"). Ele declarava:

> Ao desenvolver e ajudar outros a desenvolver software, desvendamos formas melhores de desenvolvimento. Por meio deste trabalho passamos a valorizar:
>
> *Indivíduos e interações* acima de processos e ferramentas
>
> *Software operacional* acima de documentação completa
>
> *Colaboração dos clientes* acima de negociação contratual
>
> *Respostas a mudanças* acima de seguir um plano
>
> Ou seja, embora haja valor nos itens à direita, valorizaremos os da esquerda mais ainda.

PANORAMA

O que é? A engenharia de software ágil combina filosofia com um conjunto de princípios de desenvolvimento. A filosofia defende a satisfação do cliente e a entrega incremental antecipada; equipes de projeto pequenas e altamente motivadas; métodos informais; artefatos de engenharia de software mínimos; e, acima de tudo, simplicidade no desenvolvimento geral. Os princípios de desenvolvimento priorizam a entrega mais do que a análise e o projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e contínua entre desenvolvedores e clientes.

Quem realiza? Os engenheiros de software e outros envolvidos no projeto (gerentes, clientes, usuários) trabalham conjuntamente em uma equipe ágil – uma equipe que se auto-organiza e que controla seu próprio destino. Uma equipe ágil acelera a comunicação e a colaboração entre todos os participantes (que estão ao seu serviço).

Por que é importante? O ambiente moderno dos sistemas e dos produtos da área é acelerado e está em constante mudança. A engenharia de software ágil constitui uma alternativa razoável para a engenharia convencional voltada para certas classes de software e para certos tipos de projetos. Ela tem se mostrado capaz de entregar sistemas corretos rapidamente.

Quais são as etapas envolvidas? O desenvolvimento ágil poderia ser mais bem denominado "engenharia de software flexível". As atividades metodológicas básicas – comunicação, planejamento, modelagem, construção e entrega – permanecem. Entretanto, elas se transformam em um conjunto de tarefas mínimas que impulsiona a equipe para o desenvolvimento e para a entrega (pode-se levantar a questão de que isso é feito em detrimento da análise do problema e do projeto de soluções).

Qual é o artefato? Tanto o cliente quanto o engenheiro têm o mesmo parecer: o único artefato realmente importante consiste em um "incremento de software" operacional que seja entregue, adequadamente, na data combinada.

Como garantir que o trabalho foi realizado corretamente? Se a equipe ágil concorda que o processo funciona e essa equipe produz incrementos de software passíveis de entrega e que satisfaçam o cliente, então, o trabalho está correto.

Um manifesto normalmente é associado a um movimento político emergente: ataca a velha guarda e sugere uma mudança revolucionária (espera-se que para melhor). De certa forma, é exatamente disso que trata o desenvolvimento ágil.

Embora as ideias fundamentais que norteiam o desenvolvimento ágil tenham estado conosco por muitos anos, apenas há menos de duas décadas se consolidaram como um "movimento". Em essência, métodos ágeis[1] se desenvolveram em um esforço para sanar fraquezas reais e perceptíveis da engenharia de software convencional. O desenvolvimento ágil oferece benefícios importantes; no entanto, não é indicado para todos os projetos, produtos, pessoas e situações. Também *não* é a antítese da prática de engenharia de software confiável e pode ser aplicado como uma filosofia geral para todos os trabalhos de software.

Desenvolvimento ágil não significa que nenhum documento é criado; significa que apenas os documentos que vão ser consultados mais adiante no processo de desenvolvimento são criados.

Na economia moderna, frequentemente é difícil ou impossível prever como um sistema computacional (por exemplo, um aplicativo móvel) vai evoluir com o tempo. As condições de mercado mudam rapidamente, as necessidades dos usuários se alteram, e novas ameaças competitivas surgem sem aviso. Em muitas situações, não se conseguirá definir os requisitos completamente antes que se inicie o projeto. É preciso ser ágil o suficiente para dar uma resposta a um ambiente de negócios fluido.

Fluidez implica mudança, e mudança é cara – particularmente se for sem controle e mal gerenciada. Uma das características mais convincentes da metodologia ágil é sua habilidade de reduzir os custos da mudança no processo de software.

Será que isso significa que o reconhecimento dos desafios apresentados pela realidade moderna faz que valiosos princípios, conceitos, métodos e ferramentas da engenharia de software sejam descartados? Absolutamente não! Como todas as disciplinas de engenharia, a engenharia de software continua a evoluir. Ela pode ser adaptada facilmente aos desafios apresentados pela demanda por agilidade.

"Agilidade: 1. Todo o resto: 0."

Tom DeMarco

Em um texto instigante sobre desenvolvimento de software ágil, Alistair Cockburn [Coc02] argumenta que o modelo de processo prescritivo, apresentado no Capítulo 4, tem uma falha essencial: *esquece-se das fraquezas das pessoas que desenvolvem o software*. Os engenheiros de software não são robôs. Eles apresentam grande variação nos estilos de trabalho; diferenças significativas no nível de habilidade, criatividade, organização, consistência e espontaneidade. Alguns se comunicam bem na forma escrita, outros não. Cockburn afirma que os modelos de processos podem "lidar com as fraquezas comuns das pessoas com disciplina e/ou tolerância" e que a maioria dos modelos de processos prescritivos opta por disciplina. Segundo ele: "Como a coerência nas ações é uma fraqueza humana, as metodologias com disciplina elevada são frágeis".

Para que funcionem, os modelos de processos devem fornecer um mecanismo realista que estimule a disciplina necessária ou, então, devem ter características que apresentem "tolerância" com as pessoas que realizam trabalhos de engenharia de software. Invariavelmente, práticas tolerantes são mais

[1] Os métodos ágeis são, algumas vezes, conhecidos como *métodos light* ou *métodos enxutos* (*lean methods*).

facilmente adotadas e sustentadas pelas pessoas envolvidas, porém (como o próprio Cockburn admite) podem ser menos produtivas. Como a maioria das coisas na vida, deve-se considerar os prós e os contras.

5.1 O que é agilidade?

Afinal, o que é agilidade no contexto da engenharia de software? Ivar Jacobson [Jac02a] apresenta uma discussão útil:

> Atualmente, *agilidade* se tornou a palavra da moda quando se descreve um processo de software moderno. Todo mundo é ágil. Uma equipe ágil é aquela rápida e capaz de responder de modo adequado às mudanças. Mudança tem tudo a ver com desenvolvimento de software. Mudança no software que está sendo criado, mudança nos membros da equipe, mudança devido a novas tecnologias, mudanças de todos os tipos que poderão ter um impacto no produto que está em construção ou no projeto que cria o produto. Suporte à mudança deve ser incorporado a tudo o que fazemos em software, algo que abraçamos porque é o coração e a alma do software. Uma equipe ágil reconhece que o software é desenvolvido por indivíduos trabalhando em equipes e que as habilidades dessas pessoas, suas capacidades em colaborar estão no cerne do sucesso do projeto.

De acordo com Jacobson, a difusão da mudança é o principal condutor para a agilidade. Os engenheiros de software devem ser rápidos, caso queiram assimilar as rápidas mudanças que Jacobson descreve.

Não cometa o erro de supor que a agilidade lhe dará licença para abreviar soluções. Processo é um requisito, e disciplina é essencial.

Entretanto, agilidade é mais do que uma resposta à mudança. Ela abrange também a filosofia proposta no manifesto citado no início deste capítulo. Ela incentiva a estruturação e as atitudes em equipe que tornam a comunicação mais fácil (entre membros da equipe, entre o pessoal ligado à tecnologia e o pessoal da área comercial, entre os engenheiros de software e seus gerentes). Enfatiza a entrega rápida do software operacional e diminui a importância dos artefatos intermediários (nem sempre um bom negócio); aceita o cliente como parte da equipe de desenvolvimento e trabalha para eliminar a atitude de "nós e eles" que continua a impregnar muitos projetos de software; reconhece que o planejamento em um mundo incerto tem seus limites e que o plano (roteiro) de projeto deve ser flexível.

A agilidade pode ser aplicada a qualquer processo de software. Entretanto, para alcançá-la, é essencial que o processo seja projetado de modo que a equipe possa adaptar e alinhar (racionalizar) tarefas; possa conduzir o planejamento, compreendendo a fluidez de uma metodologia de desenvolvimento ágil; possa eliminar tudo, exceto os artefatos essenciais, conservando-os enxutos; e enfatize a estratégia de entrega incremental, conseguindo entregar ao cliente, o mais rapidamente possível, o software operacional para o tipo de produto e ambiente operacional.

5.2 Agilidade e o custo das mudanças

O pensamento convencional em desenvolvimento de software (baseada em décadas de experiência) é que os custos de mudanças aumentam de forma não linear conforme o projeto avança (Figura 5.1, curva em preto contínua). É relativamente

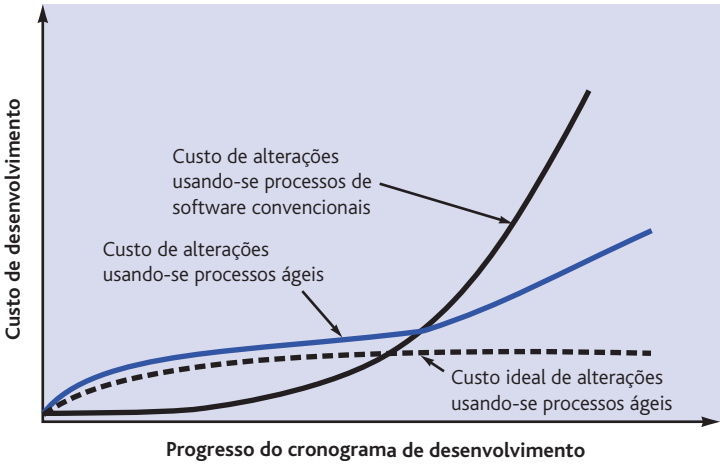

FIGURA 5.1 Custos de alterações como uma função do tempo em desenvolvimento.

fácil acomodar uma mudança quando a equipe de software está reunindo requisitos (no início de um projeto). Talvez seja necessário alterar um detalhamento do uso, ampliar uma lista de funções ou editar uma especificação por escrito. Os custos desse trabalho são mínimos, e o tempo demandado não afetará negativamente o resultado do projeto. Mas, se adiantarmos alguns meses, o que aconteceria? A equipe está em meio aos testes de validação (que ocorrem relativamente no final do projeto), e um importante envolvido está solicitando uma mudança funcional grande. A mudança exige uma alteração no projeto da arquitetura do software, projeto e desenvolvimento de três novos componentes, modificações em outros cinco componentes, projeto de novos testes e assim por diante. Os custos crescem rapidamente, e o tempo e os custos necessários para assegurar que a mudança seja feita sem efeitos colaterais inesperados não serão insignificantes.

"A agilidade é dinâmica, de conteúdo específico, abrange mudanças agressivas e é orientada ao crescimento."

Steven Goldman *et al.*

Os proponentes da agilidade (por exemplo, [Bec00], [Amb04]) argumentam que um processo ágil bem elaborado "achata" o custo da curva de mudança (Figura 5.1, curva em linha azul), permitindo que uma equipe de software assimile as alterações, realizadas posteriormente em um projeto de software, sem um impacto significativo nos custos ou no tempo. Já foi mencionado que o processo ágil envolve entregas incrementais. O custo das mudanças é atenuado quando a entrega incremental é associada a outras práticas ágeis, como testes contínuos de unidades e programação em pares (discutida mais adiante neste capítulo). Há evidências [CocO1a] que sugerem ser possível alcançar redução significativa nos custos de alterações, embora haja um debate contínuo sobre qual o nível em que a curva de custos se torna "achatada".

Um processo ágil reduz o custo das alterações porque o software é entregue (liberado) de forma incremental e as alterações podem ser mais bem controladas dentro de incrementais.

5.3 O que é processo ágil?

Qualquer processo ágil de software é caracterizado de uma forma que trate de uma série de preceitos-chave [Fow02] acerca da maioria dos projetos de software:

1. É difícil prever quais requisitos de software vão persistir e quais sofrerão alterações. É igualmente difícil prever de que maneira as prioridades do cliente sofrerão alterações conforme o projeto avança.

Uma vasta coleção de artigos sobre processo ágil pode ser encontrada em **http://www.agilemodeling.com/**.

2. Para muitos tipos de software, o projeto e a construção são intercalados. Ou seja, ambas as atividades devem ser realizadas em conjunto para que os modelos de projeto sejam provados conforme são criados. É difícil prever quanto de trabalho de projeto será necessário antes que a sua construção (desenvolvimento) seja implementada para avaliar o projeto.
3. Análise, projeto, construção (desenvolvimento) e testes não são tão previsíveis (do ponto de vista de planejamento) quanto gostaríamos que fosse.

Dados esses três preceitos, surge uma importante questão: como criar um processo capaz de administrar a *imprevisibilidade*? A resposta, conforme já observado, está na adaptabilidade do processo (alterar rapidamente o projeto e as condições técnicas). Portanto, um processo ágil deve ser *adaptável*.

Adaptação contínua sem progressos, entretanto, de pouco adianta. Um processo ágil de software deve adaptar *de modo incremental*. Para conseguir uma adaptação incremental, a equipe ágil precisa de feedback do cliente (de modo que as adaptações apropriadas possam ser feitas). Um catalisador eficaz para o feedback do cliente é um protótipo operacional ou parte de um sistema operacional. Dessa forma, deve-se instituir uma *estratégia de desenvolvimento incremental*. Os *incrementos de software* (protótipos executáveis ou partes de um sistema operacional) devem ser entregues em curtos períodos de tempo, de modo que as adaptações acompanhem o mesmo ritmo das mudanças (imprevisibilidade). Essa abordagem iterativa capacita o cliente a avaliar o incremento de software regularmente, fornecer o feedback necessário para a equipe de software e influenciar as adaptações feitas no processo para incluir o feedback adequadamente.

5.3.1 Princípios da agilidade

Embora processos ágeis considerem as alterações, examinar as razões para tais mudanças ainda continua sendo importante.

A Agile Alliance (consulte [Agi03], [Fow01]) estabelece 12 princípios para alcançar a agilidade:

1. A maior prioridade é satisfazer o cliente com entrega adiantada e contínua de software funcionando.
2. Aceite bem os pedidos de alterações, mesmo com o desenvolvimento adiantado. Os processos ágeis se aproveitam das mudanças para a vantagem competitiva do cliente.

Software ativo é importante, mas não se deve esquecer que também deve apresentar uma série de atributos de qualidade, incluindo confiabilidade, usabilidade e facilidade de manutenção.

3. Entregue software em funcionamento frequentemente, de algumas semanas a alguns meses, dando preferência a intervalos mais curtos.
4. O pessoal do comercial e os desenvolvedores devem trabalhar em conjunto diariamente ao longo de todo o projeto.
5. Construa projetos em torno de pessoas motivadas. Dê a elas o ambiente e o apoio necessários e acredite que elas farão o trabalho corretamente.
6. O método mais eficiente e efetivo de transmitir informações para e dentro de uma equipe de desenvolvimento é uma conversa aberta, presencial.
7. Software em funcionamento é a principal medida de progresso.

8. Os processos ágeis promovem desenvolvimento sustentável. Proponentes, desenvolvedores e usuários devem estar aptos a manter um ritmo constante indefinidamente.
9. Atenção contínua para com a excelência técnica e para com bons projetos aumenta a agilidade.
10. Simplicidade – a arte de maximizar o volume de trabalho não realizado – é essencial.
11. As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.
12. Em intervalos regulares, a equipe se avalia para ver como pode se tornar mais eficiente, então, sintoniza e ajusta seu comportamento de acordo.

Nem todo modelo de processo ágil aplica esses 12 princípios atribuindo-lhes pesos iguais, e alguns modelos preferem ignorar (ou pelo menos subestimam) a importância de um ou mais desses princípios. Entretanto, os princípios definem um *espírito ágil* mantido em cada um dos modelos de processo apresentados neste capítulo.

5.3.2 A política do desenvolvimento ágil

Você não tem de escolher entre agilidade ou engenharia de software. Em vez disso, defina uma abordagem de engenharia de software que seja ágil.

Tem havido debates consideráveis (algumas vezes acirrados) sobre os benefícios e a aplicabilidade do desenvolvimento de software ágil, em contraposição aos processos de engenharia de software mais convencionais. Jim Highsmith [Hig02a] (em tom jocoso) estabelece extremos ao caracterizar o sentimento do grupo pró-agilidade ("os agilistas"). "Os metodologistas tradicionais são um bando de 'pés na lama' que preferem produzir documentação sem falhas em vez de um sistema que funcione e atenda às necessidades do negócio". Em um contraponto, ele apresenta (mais uma vez em tom jocoso) a posição do grupo da engenharia de software tradicional: "Os metodologistas de pouco peso, quer dizer, os metodologistas 'ágeis' são um bando de hackers pretensiosos que vão acabar tendo uma grande surpresa ao tentarem transformar seus brinquedinhos em software de porte empresarial".

Como todo argumento sobre tecnologia de software, o debate sobre metodologia corre o risco de descambar para uma guerra santa. Se for deflagrada uma guerra, a racionalidade desaparecerá, e crenças, em vez de fatos, orientarão a tomada de decisão.

Ninguém é contra a agilidade. A verdadeira questão é: qual a melhor maneira de atingi-la? Igualmente importante é: como desenvolver software que atenda às necessidades atuais dos clientes e que apresente características de qualidade que o permitam ser estendido e ampliado para responder às necessidades dos clientes no longo prazo?

Não há respostas absolutas para nenhuma dessas perguntas. Mesmo na própria escola ágil, existem vários modelos de processos propostos (Seção 5.4), cada um com uma abordagem sutilmente diferente a respeito do problema da agilidade. Em cada modelo existe um conjunto de "ideias" (os agilistas relutam em chamá-las "tarefas de trabalho") que representam um afastamento significativo da engenharia de software tradicional. E, ainda assim, muitos conceitos

ágeis são apenas adaptações de bons conceitos da engenharia de software. Conclusão: pode-se ganhar muito considerando o que há de melhor nas duas escolas e praticamente nada denegrindo uma ou outra abordagem.

Caso se interesse mais, consulte [Hig01], [Hig02a] e [DeM02], em que é apresentado um resumo interessante a respeito de outras importantes questões técnicas e políticas.

5.4 Extreme programming – XP (Programação Extrema)

Um premiado "jogo de simulação de processos", que inclui um módulo de processo XP, pode ser encontrado em: **http://www.ics.uci.edu/~emilyo/SimSE/downloads.html**.

Para ilustrar um processo ágil de forma um pouco mais detalhada, vamos dar uma visão geral da *Extreme Programming – XP* (*Programação Extrema*), a abordagem mais amplamente utilizada para desenvolvimento de software ágil. Embora os primeiros trabalhos sobre os conceitos e métodos associados à XP tenham ocorrido no final dos anos 1980, o trabalho seminal sobre o tema foi escrito por Kent Beck [Bec04a]. Uma variante da XP, denominada *Industrial XP* (IXP), refina a XP para aplicar processo ágil especificamente em grandes organizações [Ker05].

5.4.1 O processo XP

O que é uma "história" XP?

A Extreme Programming (Programação Extrema) emprega uma metodologia orientada a objetos (Apêndice 2) como seu paradigma de desenvolvimento e envolve um conjunto de regras e práticas constantes no contexto de quatro atividades metodológicas: planejamento, projeto, codificação e testes. A Figura 5.2 ilustra o processo XP e destaca alguns conceitos e tarefas-chave associados a cada uma das atividades metodológicas. As atividades-chave da XP são sintetizadas nos parágrafos a seguir.

Planejamento. A atividade de planejamento (também chamada de *o jogo do planejamento*) se inicia com *ouvir* – uma atividade de levantamento de requisi-

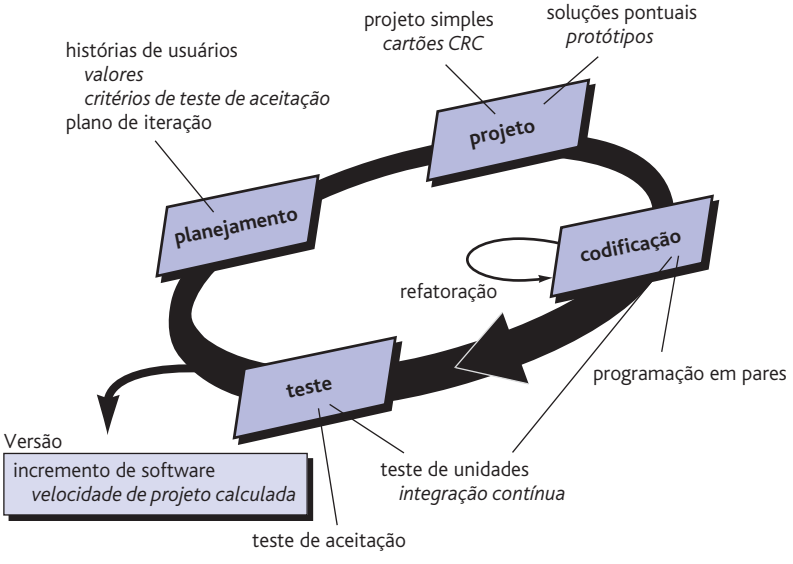

FIGURA 5.2 O processo da Extreme Programming (XP).

tos que capacita os membros técnicos da equipe XP a entender o ambiente de negócios do software e permite obter uma percepção ampla sobre os resultados solicitados, fatores principais e funcionalidade. A atividade de ouvir conduz à criação de um conjunto de "histórias" (também denominadas *histórias de usuários*) que descreve o resultado, as características e a funcionalidade solicitados para o software a ser construído. Cada *história* (similar aos casos de uso descritos no Capítulo 8) é escrita pelo cliente e é colocada em uma ficha. O cliente atribui um *valor* (uma prioridade) à história baseando-se no valor de negócio global do recurso ou função.[2] Os membros da equipe XP avaliam, então, cada história e atribuem um *custo* – medido em semanas de desenvolvimento – a ela. Se a história exigir, por estimativa, mais do que três semanas de desenvolvimento, é solicitado ao cliente que ele a divida em histórias menores, e a atribuição de valor e custo ocorre novamente. É importante notar que podem ser escritas novas histórias a qualquer momento.

Um "jogo de planejamento" XP bastante interessante pode ser encontrado em: **http://csis.pace.edu/~bergin/xp/planninggame.html**.

Clientes e desenvolvedores trabalham juntos para decidir como agrupar histórias para a versão seguinte (o próximo incremento de software) a ser desenvolvida pela equipe XP. Conseguindo chegar a um *compromisso* básico (concordância sobre quais histórias serão incluídas, data de entrega e outras questões de projeto) para uma versão, a equipe XP ordena as histórias a ser desenvolvidas em uma de três formas: (1) todas serão implementadas imediatamente (em um prazo de poucas semanas), (2) as histórias de maior valor serão deslocadas para cima no cronograma e implementadas primeiro ou (3) as histórias de maior risco serão deslocadas para cima no cronograma e implementadas primeiro.

A velocidade do projeto é uma medida sutil da produtividade de uma equipe.

Depois de a primeira versão do projeto (também denominada incremento de software) ter sido entregue, a equipe XP calcula a velocidade do projeto. De forma simples, a *velocidade do projeto* é o número de histórias de clientes implementadas durante a primeira versão. Assim, a velocidade do projeto pode ser utilizada para (1) ajudar a estimar as datas de entrega e o cronograma para versões subsequentes e (2) determinar se foi assumido um compromisso exagerado para todas as histórias ao longo de todo o projeto de desenvolvimento. Se ocorrer um exagero, o conteúdo das versões é modificado – ou as datas finais de entrega são alteradas.

A XP tira a ênfase da importância do projeto. Nem todos concordam. De fato, há ocasiões em que o projeto deve ser enfatizado.

Conforme o trabalho de desenvolvimento prossegue, o cliente pode acrescentar histórias, mudar o valor de uma já existente, dividir algumas ou eliminá-las. Em seguida, a equipe XP reconsidera todas as versões remanescentes e modifica seus planos de forma correspondente.

Projeto. O projeto XP segue rigorosamente o princípio KISS (*keep it simple, stupid!*, ou seja, não complique!). É sempre preferível um projeto simples a uma representação mais complexa. Como acréscimo, o projeto oferece um guia de implementação para uma história à medida que é escrita – nada mais, nada menos. O projeto de funcionalidade extra (pelo fato de o desenvolvedor supor que ela será necessária no futuro) é desestimulado.[3]

[2] O valor de uma história também pode depender da presença de outra história.

[3] Tais diretrizes de projeto devem ser seguidas em todos os métodos de engenharia de software, apesar de ocorrerem situações em que terminologia e notação sofisticadas possam constituir obstáculo para a simplicidade.

Técnicas de refatoração e ferramentas podem ser encontradas em: **www.refactoring.com.**

A XP estimula o uso de cartões CRC (Capítulo 10) como um mecanismo eficaz para pensar o software em um contexto orientado a objetos. Os cartões CRC (classe-responsabilidade-colaborador) identificam e organizam as classes orientadas a objetos[4] relevantes para o incremento de software corrente. A equipe XP conduz o exercício de projeto usando um processo semelhante ao descrito no Capítulo 10. Os cartões CRC são o único artefato de projeto produzido como parte do processo XP.

A refatoração aprimora a estrutura interna de um projeto (ou código-fonte) sem alterar sua funcionalidade ou comportamento externos.

Se for encontrado um problema de projeto difícil, como parte do projeto de uma história, a XP recomenda a criação imediata de um protótipo operacional dessa parte do projeto. Denominada *solução pontual*, o protótipo do projeto é implementado e avaliado. O objetivo é reduzir o risco para quando a verdadeira implementação iniciar e validar as estimativas originais para a história contendo o problema de projeto.

A XP estimula a *refatoração* – uma técnica de construção que também é uma técnica de projeto. Fowler [Fow00] descreve a refatoração da seguinte maneira:

> Refatoração é o processo de alterar um sistema de software de modo que o comportamento externo do código não se altere, mas a estrutura interna se aprimore. É uma forma disciplinada de organizar código [e modificar/simplificar o projeto interno] que minimiza as chances de introdução de bugs. Em resumo, ao se refatorar, se está aperfeiçoando o projeto de codificação depois de este ter sido feito.

Como o projeto XP praticamente não usa notação e produz poucos artefatos, quando produz, além dos cartões CRC e soluções pontuais, o projeto é visto como algo transitório que pode e deve ser continuamente modificado conforme a construção prossegue. O objetivo da refatoração é controlar essas modificações, sugerindo pequenas mudanças de projeto "capazes de melhorá-lo radicalmente" [Fow00]. Deve-se observar, no entanto, que o esforço necessário para a refatoração pode aumentar significativamente à medida que o tamanho de uma aplicação cresça.

Um aspecto central na XP é o de que a elaboração do projeto ocorre tanto antes *quanto depois* de se ter iniciado a codificação. Refatoração significa que o "projetar" é realizado continuamente enquanto o sistema estiver em elaboração. Na realidade, a própria atividade de desenvolvimento guiará a equipe XP quanto ao aprimoramento do projeto.

Informações úteis sobre a XP podem ser obtidas em **www.xprogramming.com**.

Codificação. Depois de desenvolvidas as histórias, e de o trabalho preliminar de elaboração do projeto ter sido feito, a equipe *não* passa para a codificação, mas sim desenvolve uma série de testes de unidades que exercitarão cada uma das histórias a ser incluída na versão corrente (incremento de software).[5]

[4] As classes orientadas a objetos são discutidas no Apêndice 2, no Capítulo 10 e ao longo da Parte II deste livro.

[5] Essa abordagem equivale a saber as perguntas de uma prova antes de começar a estudar. Torna o estudo muito mais fácil, permitindo que se concentre a atenção apenas nas perguntas que serão feitas.

Uma vez criado o teste de unidades[6], o desenvolvedor poderá se concentrar melhor no que deve ser implementado para ser aprovado no teste. Nada estranho é adicionado (KISS). Estando o código completo, ele pode ser testado em unidade imediatamente e, dessa forma, fornecer feedback para os desenvolvedores instantaneamente.

O que é programação em pares?

Um conceito-chave na atividade de codificação (e um dos mais discutidos aspectos da XP) é a *programação em pares*. A XP recomenda que duas pessoas trabalhem juntas em uma mesma estação de trabalho para criar código para uma história. Isso fornece um mecanismo para solução de problemas em tempo real (duas cabeças normalmente funcionam melhor do que uma) e garantia da qualidade em tempo real (o código é revisto à medida que é criado). Ele também mantém os desenvolvedores concentrados no problema em questão. Na prática, cada pessoa assume um papel ligeiramente diferente. Por exemplo, uma pessoa poderia pensar nos detalhes da codificação de determinada parte do projeto, enquanto outra assegura que padrões de codificação (uma parte exigida pela XP) sejam seguidos ou que o código para a história passará no teste de unidades desenvolvido para validação do código em relação à história.[7]

Muitas equipes de software são constituídas por individualistas. É preciso mudar tal cultura para que a programação em pares funcione efetivamente.

À medida que a dupla de programadores conclui o trabalho, o código que desenvolveram é integrado ao trabalho de outros. Em alguns casos, isso é realizado diariamente por uma equipe de integração. Em outros, a dupla de programadores é responsável pela integração. A estratégia de "integração contínua" ajuda a evitar problemas de compatibilidade e de interface, além de criar um ambiente "teste da fumaça" (Capítulo 22) que ajuda a revelar erros precocemente.

Como são usados os testes de unidade na XP?

Testes. Os testes de unidades criados devem ser implementados usando-se uma metodologia que os capacite a ser automatizados (assim, poderão ser executados fácil e repetidamente). Isso estimula uma estratégia de testes de regressão (Capítulo 22) toda vez que o código for modificado (o que é frequente, dada a filosofia de refatoração da XP).

Como os testes de unidades individuais são organizados em um "conjunto de testes universal" [Wel99], os testes de integração e validação do sistema podem ocorrer diariamente. Isso dá à equipe XP uma indicação contínua do progresso e também permite lançar alertas logo no início, caso as coisas não andem bem. Wells [Wel99] afirma: "Corrigir pequenos problemas em intervalos de poucas horas leva menos tempo do que corrigir problemas enormes próximo ao prazo de entrega".

Os testes de aceitação da XP são elaborados com base nas histórias de usuários.

Os *testes de aceitação* da XP, também denominados *testes de cliente,* são especificados pelo cliente e mantêm o foco nas características e na funcionalidade do sistema total que são visíveis e que podem ser revistas pelo cliente. Os testes de aceitação são obtidos de histórias de usuários implementadas como parte de uma versão do software.

---

[6] O teste de unidades, discutido detalhadamente no Capítulo 22, concentra-se em um componente de software individual, exercitando a interface, a estrutura de dados e a funcionalidade do componente, em uma tentativa de que se revelem erros pertinentes ao componente.

[7] A programação em pares se tornou tão difundida em toda a comunidade do software, que o tema virou manchete no *The Wall Street Journal* [Wal12].

5.4.2 Industrial XP

Que novas práticas são acrescidas à XP para elaborar a IXP?

Joshua Kerievsky [Ker05] descreve a *Industrial Extreme Programming* (IXP, Programação Extrema Industrial) da seguinte maneira: "A IXP é uma evolução orgânica da XP. Ela é imbuída do mesmo espírito minimalista, centrado no cliente e orientado a testes da XP. Difere da XP original principalmente por sua maior inclusão do gerenciamento, por seu papel expandido para os clientes e por suas práticas técnicas atualizadas". A IXP incorpora seis novas práticas desenvolvidas para ajudar a garantir que um projeto XP funcione com êxito em empreendimentos significativos em uma grande organização:

"Habilidade consiste no que se é capaz de fazer. Motivação determina o que você faz. Atitude determina quão bem você faz."

Lou Holtz

Avaliação imediata. A equipe IXP verifica se todos os membros da comunidade de projeto (por exemplo, envolvidos, desenvolvedores, gerentes) estão a bordo, têm o ambiente correto estabelecido e entendem os níveis de habilidade envolvidos.

Comunidade de projeto. A equipe IXP determina se as pessoas certas, com as habilidades e o treinamento corretos, estão prontas para o projeto. A "comunidade" abrange tecnólogos e outros envolvidos.

Mapeamento do projeto. A própria equipe IXP avalia o projeto para determinar se ele se justifica em termos de negócios e se vai ultrapassar as metas e objetivos globais da organização.

Gerenciamento orientado a testes. A equipe IPX estabelece uma série de "destinos" mensuráveis [Ker05] que avaliam o progresso até a data e, então, define mecanismos para determinar se estes foram atingidos ou não.

Retrospectivas. Uma equipe IXP conduz uma revisão técnica especializada (Capítulo 20) após a entrega de um incremento de software. Denominada *retrospectiva,* a revisão examina "problemas, eventos e lições aprendidas" [Ker05] ao longo do processo de incremento de software e/ou do desenvolvimento da versão completa do software.

Aprendizagem contínua. A equipe IXP é estimulada (e possivelmente incentivada) a aprender novos métodos e técnicas que possam levar a um produto de qualidade mais alta.

CASASEGURA

Considerando o desenvolvimento de software ágil

Cena: Escritório de Doug Miller.

Atores: Doug Miller, gerente de engenharia de software; Jamie Lazar, membro da equipe de software; Vinod Raman, membro da equipe de software.

Conversa:
(Batendo à porta, Jamie e Vinod entram na sala de Doug.)

Jamie: Doug, você tem um minuto?

Doug: Com certeza, Jamie, o que há?

Jamie: Estivemos pensando a respeito da discussão de ontem sobre processos... sabe, que processo vamos escolher para o *CasaSegura*.

Doug: E?

Vinod: Eu estava conversando com um amigo de outra empresa e ele me falou sobre Extreme Programming. É um modelo de processo ágil... já ouviu falar?

Doug: Sim, algumas coisas boas, outras ruins.

Jamie: Bem, pareceu muito bom para nós. Permite que se desenvolva software rapidamente, usa algo chamado programação em pares para fazer verificações de qualidade em tempo real... é bem legal, eu acho.

Doug: Realmente, apresenta um monte de ideias muito boas. Gosto do conceito de programação em pares, por exemplo, e da ideia de que os envolvidos devam fazer parte da equipe.

Jamie: Hã? Quer dizer que o pessoal de marketing trabalhará conosco na equipe de projeto?

Doug (confirmando com a cabeça): Eles estão envolvidos, não?

Jamie: Jesus... eles vão solicitar alterações a cada cinco minutos.

Vinod: Não necessariamente. Meu amigo me disse que existem formas de se "abarcar" as mudanças durante um projeto XP.

Doug: Então, meus amigos, vocês acham que deveríamos usar a XP?

Jamie: Definitivamente vale considerar.

Doug: Concordo. E mesmo que optássemos por um modelo incremental, não há razão para não podermos incorporar muito do que a XP tem a oferecer.

Vinod: Doug, mas antes você disse "algumas coisas boas, outras ruins". Quais são as coisas ruins?

Doug: O que não me agrada é a maneira como a XP dá menos importância à análise e ao projeto... diz mais ou menos que a codificação é onde a ação está...

(Os membros da equipe se entreolham e sorriem.)

Doug: Então vocês concordam com a metodologia XP?

Jamie (falando por ambos): Escrever código é o que fazemos, chefe!

Doug (rindo): É verdade, mas eu gostaria de vê-los perdendo um pouco menos de tempo codificando para depois recodificar e dedicando um pouco mais de tempo analisando o que precisa ser feito e projetando uma solução que funcione.

Vinod: Talvez possamos ter as duas coisas, agilidade com um pouco de disciplina.

Doug: Acho que sim, Vinod. Na realidade, tenho certeza disso.

Além das seis novas práticas apresentadas, a IXP modifica várias práticas XP existentes e redefine certas funções e responsabilidades para torná-las mais receptivas para projetos importantes de grandes empresas. Para uma discussão mais ampla sobre a IXP, visite **http://industrialxp.org**.

5.5 Outros modelos de processos ágeis

"Nossa profissão troca de metodologias como uma garota de 14 anos troca de roupas."

Stephen Hawrysh e Jim Ruprecht

A história da engenharia de software é recheada de metodologias e descrições de processos, métodos e notações de modelagem, ferramentas e tecnologias obsoletas. Todas atingiram certa notoriedade e foram ofuscadas por algo novo e (supostamente) melhor. Com a introdução de uma ampla variedade de modelos de processos ágeis – todos disputando aceitação pela comunidade de desenvolvimento de software –, o movimento ágil está seguindo o mesmo caminho histórico.[8]

Conforme citado na última seção, o modelo mais utilizado entre os modelos de processos ágeis é o Extreme Programming (XP). Porém, muitos outros têm sido propostos e encontram-se em uso no setor. Nesta seção, apresentamos um breve panorama de quatro métodos ágeis comuns: Scrum, DSSD, Modelagem Ágil (AM) e Processo Unificado Ágil (AUP).

[8] Isso não é algo ruim. Antes que um ou mais modelos ou métodos sejam aceitos como um padrão, todos devem competir para conquistar os corações e mentes dos engenheiros de software. Os "vencedores" evoluem e se transformam nas boas práticas, enquanto os "perdedores" desaparecem ou se fundem aos modelos vencedores.

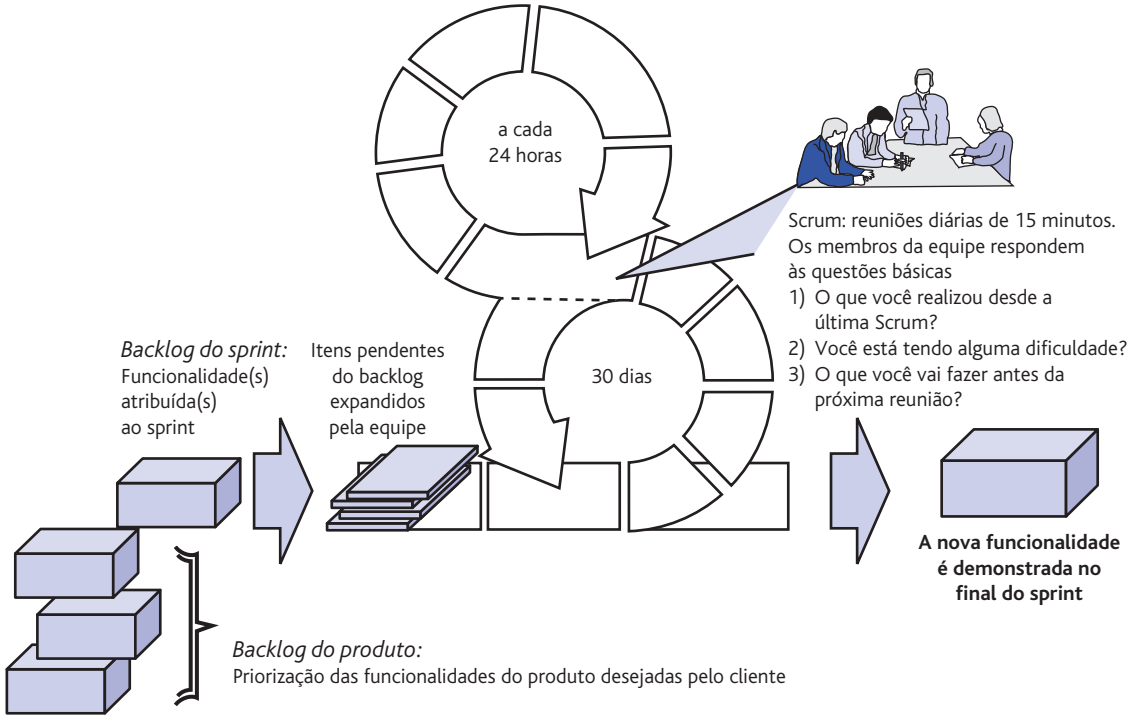

FIGURA 5.3 Fluxo do processo Scrum.

5.5.1 Scrum

Informações e recursos úteis sobre o Scrum podem ser encontrados em **www.controlchaos.com**.

Scrum (o nome provém de uma atividade que ocorre durante a partida de rugby)[9] é um método de desenvolvimento ágil de software concebido por Jeff Sutherland e sua equipe de desenvolvimento no início dos anos 1990. Mais recentemente, Schwaber e Beedle [Sch01b] realizaram desenvolvimentos adicionais nos métodos Scrum.

Os princípios do Scrum são coerentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades metodológicas: requisitos, análise, projeto, evolução e entrega. Em cada atividade metodológica, ocorrem tarefas realizadas dentro de um padrão de processo (discutido no parágrafo a seguir) chamado *sprint*. O trabalho realizado dentro de um sprint (o número de sprints necessários para cada atividade metodológica varia dependendo do tamanho e da complexidade do produto) é adaptado ao problema em questão e definido, e muitas vezes modificado em tempo real, pela equipe Scrum. O fluxo geral do processo Scrum está ilustrado na Figura 5.3.

O Scrum enfatiza o uso de um conjunto de padrões de processos de software [Noy02] que provaram ser eficazes para projetos com prazos de entrega apertados, requisitos mutáveis e urgência do negócio. Cada um desses padrões de processos define um conjunto de atividades de desenvolvimento:

[9] Um grupo de jogadores faz uma formação em torno da bola, e seus companheiros de equipe trabalham juntos (às vezes, de forma violenta!) para avançar com a bola em direção ao fundo do campo.

Backlog – uma lista com prioridades dos requisitos ou funcionalidades do projeto que fornecem valor comercial ao cliente. Os itens podem ser adicionados a esse registro a qualquer momento (é assim que as alterações são introduzidas). O gerente de produto avalia o registro e atualiza as prioridades conforme solicitado.

O Scrum engloba um conjunto de padrões de processos enfatizando prioridades de projeto, unidades de trabalho compartimentalizadas, comunicação e feedback frequente por parte dos clientes.

Sprints – consistem em unidades de trabalho solicitadas para atingir um requisito estabelecido no registro de trabalho (backlog) e que precisa ser ajustado dentro de um prazo já fechado (janela de tempo)[10] (tipicamente 30 dias). Alterações (por exemplo, itens do registro de trabalho – *backlog work items*) não são introduzidas durante execução de urgências (sprint). Portanto, o sprint permite que os membros de uma equipe trabalhem em um ambiente de curto prazo, porém estável.

Reuniões Scrum – são reuniões curtas (tipicamente 15 minutos), realizadas diariamente pela equipe Scrum. São feitas três perguntas-chave que são respondidas por todos os membros da equipe [Noy02]:

- O que você realizou desde a última reunião de equipe?
- Quais obstáculos está encontrando?
- O que planeja realizar até a próxima reunião da equipe?

Um líder de equipe, chamado *Scrum master,* conduz a reunião e avalia as respostas de cada integrante. A reunião Scrum, realizada diariamente, ajuda a equipe a revelar problemas em potencial o mais cedo possível. Ela também leva à "socialização do conhecimento" [Bee99] e, portanto, promove uma estrutura de equipe auto-organizada.

Demos – entrega do incremento de software ao cliente para que a funcionalidade implementada possa ser demonstrada e avaliada por ele. É importante notar que a demo pode não ter toda a funcionalidade planejada, mas sim funções que possam ser entregues no prazo estipulado.

Beedle e seus colegas [Bee99] apresentam uma ampla discussão sobre esses padrões: "O Scrum pressupõe a existência do caos...". Os padrões de processos do Scrum capacitam uma equipe de software a trabalhar com sucesso em um mundo onde é impossível eliminar a incerteza.

5.5.2 Método de Desenvolvimento de Sistemas Dinâmicos (DSDM)

O *Método de Desenvolvimento de Sistemas Dinâmicos* (DSDM, *Dynamic Systems Development Method*) [Sta97] é uma abordagem de desenvolvimento de software ágil que "oferece uma metodologia para construir e manter sistemas que satisfaçam restrições de prazo apertado por meio do uso da prototipação incremental em um ambiente de projeto controlado" [CCS02]. A filosofia DSDM baseia-se em uma versão modificada do princípio de Pareto – 80% de uma aplicação pode ser entregue em 20% do tempo que levaria para entregar a aplicação completa (100%).

Recursos úteis para o DSDM podem ser encontrados em **www.dsdm.org**.

[10] *Janela de tempo* (*time box*) é um termo de gerenciamento de projetos (consulte a Parte IV deste livro) que indica um período de tempo destinado para cumprir alguma tarefa.

O DSDM é um processo de software iterativo em que cada iteração segue a regra dos 80%. Ou seja, somente o trabalho suficiente é requisitado para cada incremento, para facilitar o movimento para o próximo incremento. Os detalhes restantes podem ser concluídos depois, quando outros requisitos do negócio forem conhecidos ou alterações tiverem sido solicitadas e acomodadas.

O DSDM Consortium (**www.dsdm.org**) é um grupo mundial de empresas-membro que assume coletivamente o papel de "mantenedor" do método. Esse consórcio definiu um modelo de processos ágeis, chamado *ciclo de vida DSDM*, que começa com um *estudo de viabilidade*, o qual estabelece os requisitos básicos e as restrições do negócio, e é seguido por um *estudo do negócio*, o qual identifica os requisitos de função e informação. Então, o DSDM define três diferentes ciclos iterativos:

O DSDM é uma metodologia de processos que pode adotar a tática de outra metodologia ágil, como a XP.

Iteração de modelos funcionais – produz um conjunto de protótipos incrementais que demonstram funcionalidade para o cliente. (Observação: todos os protótipos DSDM são feitos com a intenção de que evoluam para a aplicação final entregue ao cliente.) Durante esse ciclo iterativo, o objetivo é reunir requisitos adicionais ao se obter feedback dos usuários, à medida que eles testam o protótipo.

Iteração de projeto e desenvolvimento – revê os protótipos desenvolvidos durante a iteração de modelos funcionais para assegurar-se de que cada um tenha passado por um processo de engenharia para capacitá-los a oferecer, aos usuários, valor de negócio em termos operacionais. Em alguns casos, a iteração de modelos funcionais e a iteração de projeto e desenvolvimento ocorrem ao mesmo tempo.

Implementação – coloca a última versão do incremento de software (um protótipo "operacionalizado") no ambiente operacional. Deve-se notar que: (1) o incremento pode não estar 100% completo ou (2) alterações podem vir a ser solicitadas conforme o incremento seja alocado. Em qualquer um dos casos, o trabalho de desenvolvimento do DSDM continua, retornando-se à atividade de iteração do modelo funcional.

O DSDM pode ser combinado com a XP (Seção 5.4) para fornecer uma abordagem combinada que defina um modelo de processos confiável (o ciclo de vida do DSDM) com as práticas básicas (XP) necessárias para construir incrementos de software.

5.5.3 Modelagem Ágil (AM)

Muita informação sobre a modelagem ágil pode ser encontrada em: **www.agilemodeling.com**.

Existem muitas situações em que engenheiros de software têm de desenvolver sistemas grandes e críticos para o negócio. O escopo e a complexidade desses sistemas devem ser modelados de modo que (1) todas as partes envolvidas possam entender melhor quais requisitos devem ser atingidos, (2) o problema possa ser subdividido eficientemente entre as pessoas que têm de solucioná-lo e (3) a qualidade possa ser avaliada enquanto se está projetando e desenvolvendo o sistema. Porém, em alguns casos pode ser desencorajador gerenciar o volume de notação exigido, o grau de formalismo sugerido, o mero tamanho dos modelos para grandes projetos e a dificuldade em manter o(s) modelo(s) à

medida que ocorrem mudanças. Existe uma metodologia ágil para a modelagem de engenharia de software que possa fornecer algum alívio?

No "The Official Agile Modeling Site", Scott Ambler [Amb02a] descreve *modelagem ágil* (AM) da seguinte maneira:

> Modelagem ágil (AM) consiste em uma metodologia prática, voltada para a modelagem e documentação de sistemas baseados em software. Simplificando, modelagem ágil consiste em um conjunto de valores, princípios e práticas voltados para a modelagem do software que podem ser aplicados a um projeto de desenvolvimento de software de forma leve e eficiente. Os modelos ágeis são mais eficientes do que os tradicionais pelo fato de serem simplesmente bons, pois não têm a obrigação de ser perfeitos.

"Um dia, estava em uma farmácia tentando achar um remédio para resfriado... Não foi fácil. Havia uma parede inteira de produtos. Fica-se lá procurando: 'Bem, este tem ação imediata, mas este outro tem efeito mais duradouro...' O que é mais importante, o presente ou o futuro?"

Jerry Seinfeld

A modelagem ágil adota todos os valores coerentes com o manifesto ágil. Sua filosofia reconhece que uma equipe ágil deve ter a coragem de tomar decisões que possam causar a rejeição de um projeto e sua refatoração. A equipe também deve ter humildade para reconhecer que os profissionais de tecnologia não possuem todas as respostas e que os experts em negócios e outros envolvidos devem ser respeitados e integrados ao processo.

Embora a AM sugira uma ampla variedade de princípios de modelagem "básicos" e "suplementares", os que a tornam única são [Amb02a]:

Modelar com um objetivo. O desenvolvedor que utilizar a AM deve ter um objetivo antes de criar o modelo (por exemplo, comunicar informações ao cliente ou ajudar a compreender melhor algum aspecto do software). Uma vez identificado o objetivo, ficará mais evidente o tipo de notação a ser utilizado e o nível de detalhamento necessário.

Usar vários modelos. Há muitos modelos e notações diferentes que podem ser usados para descrever software. Para a maioria dos projetos, somente um subconjunto é essencial. A AM sugere que, para propiciar a percepção necessária, cada modelo deve apresentar um aspecto diferente do sistema e devem ser usados somente aqueles que valorizem esses modelos para o público pretendido.

"Viajar leve" é uma filosofia apropriada para todo o trabalho de engenharia de software. Construa apenas os modelos que forneçam valor... nem mais, nem menos.

Viajar leve. Conforme o trabalho de engenharia de software prossegue, conserve apenas aqueles modelos que terão valor no longo prazo e desfaça-se do restante. Todo artefato mantido deve sofrer manutenção à medida que mudanças ocorram. Isso representa trabalho que retarda a equipe. Ambler [Amb02a] observa que "Toda vez que se opta por manter um modelo, troca-se a agilidade pela conveniência de ter aquela informação acessível para a equipe de uma forma abstrata (já que, potencialmente, aumenta a comunicação dentro da equipe, assim como com os envolvidos no projeto)".

Conteúdo é mais importante do que a representação. A modelagem deve transmitir informações para seu público pretendido. Um modelo sintaticamente perfeito que transmita pouco conteúdo útil não possui tanto valor quanto aquele com notações falhas que, no entanto, fornece conteúdo valioso para seu público-alvo.

Conhecer os modelos e as ferramentas utilizadas para criá-los. Compreenda os pontos fortes e fracos de cada modelo e as ferramenta usadas para criá-lo.

Adaptar localmente. A modelagem deve ser adaptada às necessidades da equipe ágil.

Um segmento de vulto da comunidade da engenharia de software adotou a linguagem de modelagem unificada (Unified Modeling Language, UML)[11] como o método preferido para análise representativa e para modelos de projeto. O Processo Unificado (Capítulo 4) foi desenvolvido para fornecer uma metodologia para a aplicação da UML. Scott Ambler [Amb06] desenvolveu uma versão simplificada do UP que integra sua filosofia de modelagem ágil.

5.5.4 Processo Unificado Ágil

O *Processo Unificado Ágil* (AUP, *Agile Unified Process*) adota uma filosofia "serial para o que é amplo" e "iterativa para o que é particular" [Amb06] no desenvolvimento de sistemas computadorizados. Adotando as atividades em fases UP clássicas – concepção, elaboração, construção e transição –, o AUP fornece uma camada serial (isto é, uma sequência linear de atividades de engenharia de software) que permite à equipe visualizar o fluxo do processo geral de um projeto de software. Entretanto, dentro de cada atividade, a equipe itera para alcançar a agilidade e entregar incrementos de software significativos para os usuários o mais rápido possível. Cada iteração AUP trata das seguintes atividades [Amb06]:

- *Modelagem*. Representações UML do universo do negócio e do problema são criadas. Entretanto, para permanecerem ágeis, esses modelos devem ser "suficientemente bons e adequados" [Amb06] para possibilitar que a equipe prossiga.
- *Implementação*. Os modelos são traduzidos em código-fonte.
- *Testes*. Como a XP, a equipe projeta e executa uma série de testes para descobrir erros e assegurar que o código-fonte se ajuste aos requisitos.
- *Entrega*. Como a atividade de processo genérica discutida no Capítulo 3, neste contexto a entrega se concentra no fornecimento de um incremento de software e na obtenção de feedback dos usuários.
- *Configuração e gerenciamento de projeto*. No contexto do AUP, gerenciamento de configuração (Capítulo 29) refere-se a gerenciamento de alterações, de riscos e de controle de qualquer artefato[12] persistente que sejam produzidos por uma equipe. O gerenciamento de projeto monitora e controla o progresso de uma equipe e coordena suas atividades.
- *Gerenciamento do ambiente*. Coordena a infraestrutura de processos que inclui padrões, ferramentas e outras tecnologias de suporte disponíveis para a equipe.

[11] Um breve tutorial sobre a UML é apresentado no Apêndice 1.

[12] *Artefato persistente* é um modelo ou documento ou pacote de testes produzido pela equipe que será mantido por um período de tempo indeterminado. *Não* será descartado quando o incremento de software for entregue.

Embora o AUP tenha conexões históricas e técnicas com a linguagem de modelagem unificada, é importante notar que a modelagem UML pode ser usada com qualquer modelo de processo ágil descrito neste capítulo.

FERRAMENTAS DO SOFTWARE

Engenharia de requisitos

Objetivo: O objetivo das ferramentas de desenvolvimento ágil é auxiliar em um ou mais aspectos do desenvolvimento ágil, com ênfase em facilitar a geração rápida de software operacional. Essas ferramentas também podem ser usadas quando forem aplicados modelos de processos prescritivos (Capítulo 4).

Mecanismos: O mecanismo das ferramentas é variado. Em geral, conjuntos de ferramentas ágeis englobam suporte automatizado para o planejamento de projetos, desenvolvimento de casos de uso, reunião de requisitos, projeto rápido, geração de código e testes.

Ferramentas representativas:[13]

Observação: como o desenvolvimento ágil é um tópico importante, a maioria dos fornecedores de ferramentas de software tende a vender ferramentas que aceitam a metodologia ágil. As ferramentas aqui mencionadas têm características que as tornam particularmente úteis para projetos ágeis.

OnTime, desenvolvida pela Axosoft (**www.axosoft.com**), fornece suporte para gerenciamento de processo ágil para uma variedade de atividades técnicas dentro do processo.

Ideogramic UML, desenvolvida pela Ideogramic (**http://ideogramic-uml.software.informer.com/**), é um conjunto de ferramentas UML desenvolvido para uso em processo ágil.

Together Tool Set, distribuída pela Borland (**www.borland.com**), fornece uma mala de ferramentas que dão suporte para muitas atividades técnicas na XP e em outros processos ágeis.

5.6 Um conjunto de ferramentas para o processo ágil

Alguns proponentes da filosofia ágil argumentam que as ferramentas de software automatizadas (por exemplo, ferramentas para projetos) deveriam ser vistas como um suplemento secundário para as atividades, e não como fundamental para o sucesso da equipe. Entretanto, Alistair Cockburn [Coc04] sugere que ferramentas podem trazer vantagens e que "equipes ágeis enfatizam o uso de ferramentas que permitem o fluxo rápido de compreensão. Algumas dessas ferramentas são sociais, iniciando-se até no estágio de contratação de pessoal. Algumas são tecnológicas, auxiliando equipes distribuídas a simular sua presença física. Muitas são físicas, permitindo sua manipulação em workshops."

O "conjunto de ferramentas" que suporta os processos ágeis se concentra mais nas questões pessoais do que nas questões tecnológicas.

"Ferramentas" voltadas para a comunicação e para a colaboração são, em geral, de baixa tecnologia e incorporam qualquer mecanismo ("proximidade física, quadros brancos, papéis para pôster, fichas e lembretes adesivos" [Coc04] ou modernas técnicas de rede social) que forneça informações e coordenação entre desenvolvedores ágeis. A comunicação ativa é obtida por meio de dinâmicas de grupo (por exemplo, programação em pares), enquanto a comunicação passiva é obtida por meio dos "irradiadores de informações" (por exemplo, um display de um painel fixo que apresente o status geral dos diferentes componentes de um incremento). As ferramentas de gerenciamento de projeto dão pouca ênfase ao diagrama de Gantt e o substituem por gráficos de valores ganhos ou "gráficos de testes criados e cruzados com os anteriores... outras ferramentas

[13] 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.

ágeis são utilizadas para otimizar o ambiente no qual a equipe ágil trabalha (por exemplo, áreas de reunião mais eficientes), ampliar a cultura da equipe promovendo interações sociais (por exemplo, equipes próximas umas das outras), dispositivos físicos (por exemplo, lousas eletrônicas) e melhoria do processo (por exemplo, programação em pares ou janela de tempo)" [Coc04].

Algumas dessas coisas são realmente ferramentas? Serão, caso facilitem o trabalho desenvolvido por um membro da equipe ágil e venham a aprimorar a qualidade do produto final.

5.7 Resumo

Em uma economia moderna, as condições de mercado mudam rapidamente, as necessidades do cliente e do usuário evoluem e novos desafios competitivos surgem sem aviso. Os profissionais têm de assumir uma abordagem de engenharia de software que permita que permaneçam ágeis – definindo processos que sejam manipuláveis, adaptáveis e sem excessos, somente com o conteúdo essencial que possa se adequar às necessidades do mundo dos negócios moderno.

Uma filosofia ágil para a engenharia de software enfatiza quatro elementos-chave: a importância das equipes que se auto-organizam, que têm controle sobre o trabalho por elas realizado; a comunicação e a colaboração entre os membros da equipe e entre os desenvolvedores e seus clientes; o reconhecimento de que as mudanças representam oportunidades; e a ênfase na entrega rápida do software para satisfazer o cliente. Os modelos de processos ágeis foram feitos para tratar de todas essas questões.

Extreme Programming (XP) é o processo ágil mais amplamente utilizado. Organizada em quatro atividades metodológicas – planejamento, projeto, codificação e testes – a XP sugere várias técnicas poderosas e inovadoras que possibilitam a uma equipe ágil criar versões de software com frequência, propiciando recursos e funcionalidade descritos previamente e priorizados pelos envolvidos.

Outros modelos de processos ágeis também enfatizam a colaboração humana e a auto-organização das equipes, mas definem suas próprias atividades metodológicas e selecionam diferentes pontos de ênfase. Por exemplo, o Scrum enfatiza o uso de um conjunto de padrões de software que se mostrou eficaz para projetos com cronogramas apertados, requisitos mutáveis e aspectos críticos de negócio. Cada padrão de processo define um conjunto de tarefas de desenvolvimento e permite à equipe Scrum construir um processo que se adapte às necessidades do projeto. O método de desenvolvimento de sistemas dinâmicos (DSDM) defende o uso de um cronograma de tempos definidos (janela de tempo) e sugere que apenas o trabalho suficiente seja requisitado para cada incremento de software, para facilitar o movimento em direção ao incremento seguinte. A modelagem ágil (AM) afirma que modelagem é essencial para todos os sistemas, mas a complexidade, tipo e tamanhos de um modelo devem ser balizados pelo software a ser construído. O processo unificado ágil (AUP) adota a filosofia do "serial para o que é amplo" e "iterativa para o que é particular" para o desenvolvimento de software.

Problemas e pontos a ponderar

5.1. Releia o "Manifesto for Agile Software Development" no início deste capítulo. Você consegue exemplificar uma situação em que um ou mais dos quatro "valores" poderiam levar a equipe a ter problemas?

5.2. Descreva agilidade (para projetos de software) com suas próprias palavras.

5.3. Por que um processo iterativo facilita o gerenciamento de mudanças? Todos os processos ágeis discutidos neste capítulo são iterativos? É possível concluir um projeto com apenas uma iteração e ainda assim permanecer ágil? Justifique suas respostas.

5.4. Cada um dos processos ágeis poderia ser descrito usando-se as atividades metodológicas genéricas citadas no Capítulo 3? Construa uma tabela que associe as atividades genéricas às atividades definidas para cada processo ágil.

5.5. Tente elaborar mais um "princípio de agilidade" que ajudaria uma equipe de engenharia de software a se tornar mais adaptável.

5.6. Escolha um princípio de agilidade citado na Seção 5.3.1 e tente determinar se cada um dos modelos de processos apresentados neste capítulo demonstra o princípio. (Observação: apresentamos apenas uma visão geral desses modelos de processos; portanto, talvez não seja possível determinar se um princípio foi ou não tratado por um ou mais dos modelos, a menos que você pesquise mais a respeito, o que não é exigido neste problema).

5.7. Por que os requisitos mudam tanto? Afinal de contas, as pessoas não sabem o que elas querem?

5.8. A maior parte dos modelos de processos ágeis recomenda comunicação face a face. Mesmo assim, hoje em dia os membros de uma equipe de software e seus clientes podem estar geograficamente separados uns dos outros. Você acredita que isso implique que a separação geográfica seja algo a ser evitado? Você é capaz de imaginar maneiras de superar esse problema?

5.9. Escreva uma história de usuário XP que descreva o recurso "sites favoritos" ou "favoritos" disponível na maioria dos navegadores Web.

5.10. O que é uma solução pontual na XP?

5.11. Descreva com suas próprias palavras os conceitos de refatoração e programação em pares da XP.

5.12. Usando a planilha de padrões de processos apresentada no Capítulo 3, desenvolva um padrão de processo para qualquer um dos padrões Scrum da Seção 5.5.1.

5.13. Visite o site Official Agile Modeling e faça uma lista completa de todos os princípios básicos e complementares do AM.

5.14. O conjunto de ferramentas proposto na Seção 5.6 oferece suporte a muitos dos aspectos "menos prioritários" dos métodos ágeis. Como a comunicação é tão importante, recomende um conjunto de ferramentas real que poderia ser usado para melhorar a comunicação entre os envolvidos de uma equipe ágil.

Leituras e fontes de informação complementares

A filosofia geral e os princípios subjacentes do desenvolvimento de software ágil são considerados em profundidade em muitos dos livros citados neste capítulo. Além disso, livros de Pichler (*Agile Project Management with Scrum: Creating Products that Customers Love,* Addison-Wesley, 2010), Highsmith (*Agile Project Management: Creating Innovative*

Products, 2ª ed. Addison-Wesley, 2009), Shore e Chromatic (*The Art of Agile Development,* O'Reilly Media, 2008), Hunt (*Agile Software Construction,* Springer, 2005) e Carmichael e Haywood (*Better Software Faster,* Prentice Hall, 2002) trazem discussões interessantes sobre o tema. Aguanno (*Managing Agile Projects,* Multi-Media Publications, 2005) e Larman (*Agile and Iterative Development: A Manager's Guide,* Addison-Wesley, 2003) apresentam uma visão geral sobre gerenciamento e consideram as questões envolvidas no gerenciamento de projetos. Highsmith (*Agile Software Development Ecosystems,* Addison-Wesley, 2002) retrata uma pesquisa de princípios, processos e práticas ágeis. Uma discussão que vale a pena sobre o delicado equilíbrio entre agilidade e disciplina é fornecida por Booch e seus colegas (*Balancing Agility and Discipline,* Addison-Wesley, 2004).

Martin (*Clean Code: A Handbook of Agile Software Craftsmanship,* Prentice-Hall, 2009) enumera os princípios, padrões e práticas necessários para desenvolver "código limpo" em um ambiente de engenharia de software ágil. Leffingwell (*Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*, Addison-Wesley, 2011; e *Scaling Software Agility: Best Practices for Large Enterprises,* Addison-Wesley, 2007) discute estratégias para dar maior corpo às práticas ágeis para poderem ser usadas em grandes projetos. Lippert e Rook (*Refactoring in Large Software Projects: Performing Complex Restructurings Successfully,* Wiley, 2006) discutem o uso da refatoração quando aplicada a sistemas grandes e complexos. Stamelos e Sfetsos (*Agile Software Development Quality Assurance,* IGI Global, 2007) trazem técnicas SQA que estão em conformidade com a filosofia ágil.

Foram escritos dezenas de livros sobre Extreme Programming ao longo da última década. Beck (*Extreme Programming Explained: Embrace Change,* 2ª ed., Addison-Wesley, 2004) ainda é o tratado de maior autoridade sobre o tema. Além disso, Jeffries e seus colegas (*Extreme Programming Installed,* Addison-Wesley, 2000), Succi e Marchesi (*Extreme Programming Examined,* Addison-Wesley, 2001), Newkirk e Martin (*Extreme Programming in Practice,* Addison-Wesley, 2001) e Auer e seus colegas (*Extreme Programming Applied: Play to Win,* Addison-Wesley, 2001) fornecem uma discussão básica da XP, juntamente com uma orientação sobre como melhor aplicá-la. McBreen (*Questioning Extreme Programming,* Addison-Wesley, 2003) adota uma visão crítica em relação à XP, definindo quando e onde ela é apropriada. Uma análise aprofundada da programação em pares é apresentada por McBreen (*Pair Programming Illuminated,* Addison-Wesley, 2003).

Kohut (*Professional Agile Development Process: Real World Development Using SCRUM,* Wrox, 2013), Rubin (*Essential Scrum: A Practical Guide to the Most Popular Agile Process*, Addison-Wesley, 2012), Larman e Vodde (*Scaling Lean and Agile Development: Thinking and Organizational Tools for Large Scale Scrum,* Addison-Wesley, 2008) e Schwaber (*The Enterprise and Scrum,* Microsoft Press, 2007) discutem o uso de Scrum para projetos que têm grande impacto comercial. Os detalhes práticos do Scrum são debatidos por Cohn (*Succeeding with Agile*, Addison-Wesley, 2009) e Schwaber e Beedle (*Agile Software Development with SCRUM,* Prentice-Hall, 2001). Tratados úteis sobre o DSDM foram escritos pelo DSDM Consortium (*DSDM: Business Focused Development,* 2ª ed., Pearson Education, 2003) e Stapleton (*DSDM: The Method in Practice,* Addison-Wesley, 1997).

Livros de Ambler e Lines (*Disciplined Agile Delivery: A Practitioner's Guide to Agile Delivery in the Enterprise*, IBM Press, 2012) e Poppendieck e Poppendieck (*Lean Development: An Agile Toolkit for Software Development Managers,* Addison-Wesley, 2003) dão diretrizes para gerenciar e controlar projetos ágeis. Ambler e Jeffries (*Agile Modeling,* Wiley, 2002) discutem a AM com certa profundidade.

Uma grande variedade de fontes de informação sobre desenvolvimento de software ágil está disponível na Internet. Uma lista atualizada de referências relevantes (em inglês) para o processo ágil pode ser encontrada no site: **www.mhhe.com/pressman**.