



Métodos Ágeis de Desenvolvimento de Software

Como uma resposta às crescentes pressões por inovação em prazos cada vez mais reduzidos, às necessidades de constantes mudanças de requisitos e ao mau desempenho de grande parte dos projetos de desenvolvimento de software, houve um movimento na comunidade de desenvolvimento de software que deu origem aos Métodos Ágeis. Posteriormente, o conceito-base deste movimento evoluiu, de uma abordagem técnica para o âmbito gerencial, criando um novo enfoque de gerenciamento de projetos – o Gerenciamento Ágil de Projetos.

Neste artigo são apresentados o histórico, o conceito, os valores e os princípios que norteiam os Métodos Ágeis, assim como são descritos os métodos mais frequentemente utilizados pelas organizações. Também são exploradas as suas

De que se trata o artigo?

Neste artigo são apresentados o histórico, o conceito, os valores e os princípios que norteiam os Métodos Ágeis, assim como são descritos os métodos mais frequentemente utilizados pelas organizações. Também são exploradas as suas limitações, as indicações de aplicação e os resultados obtidos por empresas que já os adotaram. Por fim, são discutidos os fatores críticos de sucesso de projetos de desenvolvimento de software conduzidos com o uso desses métodos.

Para que serve?

Apresentar uma visão abrangente sobre o cenário atual envolvendo metodologias ágeis.

Em que situação o tema é útil?

Conhecer metodologias ágeis é cada vez mais importante na medida em que lidamos cada vez mais com projetos de características diferenciadas que exigem, muitas vezes, abordagens não tradicionais de desenvolvimento.

limitações, as indicações de aplicação e os resultados obtidos por empresas que já os adotaram. Por fim, são discutidos os fatores críticos de sucesso de projetos de desenvolvimento de software conduzidos com o uso desses métodos.



Marisa Villas Boas Dias

Definição e Origem dos Métodos Ágeis de Desenvolvimento de Software

Os Métodos Ágeis de Desenvolvimento de Software surgiram como uma reação aos métodos clássicos de desenvolvimento¹ e do reconhecimento da necessidade premente de se criar uma alternativa a estes “processos pesados”, caracterizados pelo foco excessivo na criação de uma documentação completa (BECK, *et al*, 2001). Em meados dos anos 90, integrantes da comunidade de desenvolvimento de software começaram a questionar estes processos, julgando-os pouco efetivos e, muitas vezes, impossíveis de serem colocados em prática (HIGHSMITH, 2002).

Sintetizando o pensamento deste grupo, Highsmith (*Ibid.*) menciona que a indústria e a tecnologia sofrem modificações tão aceleradas que acabam por “atropelar” os métodos clássicos. Highsmith *et al* (2002) ainda acrescentam que os clientes, na maioria das vezes, são incapazes de definir de forma clara e precisa, os requisitos do software, logo no início de um projeto de desenvolvimento, o que inviabiliza a adoção dos métodos clássicos em muitos projetos.

Como resposta a esta situação, muitos especialistas criaram métodos próprios para se adaptar às constantes mudanças exigidas pelo mercado e às indefinições iniciais dos projetos. O agrupamento desses métodos deu origem à família dos Métodos Ágeis de Desenvolvimento de Software. Sendo assim,

“[...] os Métodos Ágeis podem ser considerados uma coletânea de diferentes técnicas e métodos, que compartilham os mesmos valores e princípios básicos, alguns dos quais remontam de técnicas introduzidas em meados dos anos 70, como os desenvolvimentos e melhorias iterativos” (COHEN et al, 2003, p.2).

De fato, Cockburn e Highsmith (2001a) já haviam afirmado que a maioria das práticas propostas pelos Métodos Ágeis não tem nada de novo e que a diferença recai principalmente sobre o foco e os valores que os sustentam.

Segundo Cohen *et al* (2003), um dos primeiros questionamentos aos métodos clássicos de desenvolvimento de software foi feito por Schwaber, criador do *Scrum*. Para entender melhor os métodos clássicos de desenvolvimento de software baseados no SW-CMM, Schwaber (2002) elaborou um estudo junto aos cientistas da DuPont, que tinha por objetivo responder a seguinte pergunta: “Por que os processos definidos e defendidos pelo SW-CMM não promovem entregas consistentes”? Após analisarem seus processos de desenvolvimento de software, os cientistas chegaram à conclusão que, apesar do SW-CMM buscar a consistência, a previsibilidade e a confiabilidade dos processos de desenvolvimento de software, muitos destes processos ainda eram, de fato, imprevisíveis e impossíveis de serem repetidos. A explicação para tal recaía na complexidade dos processos propostos pelo SW-CMM, na consequente

difículdade de aplicação e também na necessidade de mudanças constantes e difíceis de serem antecipadas.

Schwaber (*op. cit.*) percebe que para que o desenvolvimento de software seja realmente ágil, deve-se aceitar as mudanças, ao invés de dar foco extremo à previsibilidade. Quase que simultaneamente, outros especialistas no assunto chegam à conclusão de que métodos que respondam às mudanças, tão rapidamente quanto estas venham a surgir e que incentivem a criatividade, são a única maneira de enfrentar e gerenciar os problemas do desenvolvimento de software em ambientes complexos (COCKBURN; HIGHSMITH, 2001a, SCHWABER, 2002).

Neste mesmo período, modelos de processos aplicados a outras indústrias, começam a ser analisados para servir como fonte de inspiração ao aprimoramento do processo de desenvolvimento de software (POPPENDIECK, 2001). O Modelo Toyota de Produção² foi alvo de atenção especial: enquanto as unidades fabris americanas trabalhavam a 100% de sua capacidade e mantinham grandes volumes de inventário de matérias-primas e de produtos acabados, a fábrica da Toyota mantinha o nível de estoque suficiente para um dia de operação e produzia somente o necessário para atender aos pedidos já colocados. Este modelo traduzido no princípio da *Lean Manufacturing*, visava à utilização mais eficiente dos recursos e a redução de qualquer tipo de desperdício e estava totalmente alinhado à filosofia da Administração da Qualidade Total³, criada pelo Dr. Edwards Deming (POPPENDIECK, 2001; FERREIRA *et al*, 2002). Deming (1990) acreditava que as pessoas desejavam fazer um bom trabalho e que os gerentes deveriam permitir que os trabalhadores do chão de fábrica tivessem autonomia para a tomada de decisões e a resolução de problemas. Além disso, estimulava o estabelecimento de uma relação de confiança com os fornecedores e defendia uma cultura de melhoria contínua dos processos e dos produtos. Enquanto as unidades fabris japonesas geravam produtos cada vez melhores e mais baratos, as fábricas americanas não conseguiam fazer o mesmo.

Com base nesta avaliação, Poppendieck (2001) listou 10 práticas que tornavam a *Lean Manufacturing* tão bem-sucedida e que, em seu entendimento, poderiam ser adaptadas e aplicadas ao desenvolvimento de software:

1. Eliminação de gastos – eliminar ou reduzir diagramas e modelos que não agregam valor ao produto final;
2. Minimização de inventário – minimizar artefatos intermediários, como documentos de requisitos e de desenho;
3. Maximização do fluxo – utilizar o desenvolvimento iterativo para redução do prazo de entrega do software;
4. Atendimento à demanda – atender às mudanças de requisitos;

² Modelo Toyota de Produção – para maiores informações ver Correa e Giansi (1993) e Ferreira *et al* (2002).

³ A Administração da Qualidade Total ou Total Quality Management pode ser entendida como a extensão do planejamento de negócios de uma empresa, abrangendo o planejamento da qualidade (JURAN; GRZYNA, 1991, p. 210 – 214).

¹ Entre os métodos clássicos de desenvolvimento de software podem ser citados o Modelo em Cascata e os modelos iterativos e em Espiral (COHEN *et al*, 2003).

5. Autonomia aos trabalhadores – compartilhar a documentação e dizer aos programadores “o que” precisa ser feito e não “como” deve ser feito;
6. Atendimento aos requisitos dos clientes – trabalhar perto dos clientes, permitindo que eles mudem suas opiniões ou seus desejos;
7. Fazer certo da primeira vez – testar o quanto antes e refazer o código se necessário;
8. Abolição da otimização local – gerenciar o escopo de forma flexível;
9. Desenvolvimento de parceria com os fornecedores – evitar relações conflitantes, tendo em mente que todos devem trabalhar juntos para gerar o melhor software;
10. Cultura de melhoria contínua – permitir que o processo seja melhorado, que se aprenda com os erros e se alcance o sucesso.

Highsmith (2002) afirma, que de forma independente, Kent Beck e Ron Jeffries percebem a importância dos princípios defendidos por Poppendieck (2001) durante um projeto de desenvolvimento de software na Chrysler e criam o projeto *Extreme Programming* (XP), um dos Métodos Ágeis de maior expressão atualmente. Simultaneamente, outras histórias começam a ecoar pelo mundo, como a vivenciada por Alistair Cockburn, que entrevistando profissionais do *IBM Consulting Group*, percebe que equipes de projetos bem-sucedidos se desculparam por não ter seguido os processos formais, por não utilizar as ferramentas de alta tecnologia e por ter “simplesmente” trabalhado de forma próxima e integrada, enquanto membros de projetos malsucedidos afirmavam ter seguido as regras e processos e que não entendiam o que havia dado errado. Com base nesta experiência, Cockburn desenvolveu o *Crystal Method*, outro Método Ágil (HIGHSMITH, 2002).

Face ao exposto, percebe-se que o mundo do desenvolvimento de software passa por uma importante transformação: os métodos clássicos são vistos como não adequados a todas as situações e os especialistas reconhecem a necessidade de criação de novas práticas, orientadas a pessoas e flexíveis o suficiente para fazer frente a um ambiente de negócio dinâmico (COCKBURN; HIGHSMITH, 2001a). Os principais desafios enfrentados e que devem ser endereçados pelos novos métodos de desenvolvimento de software são assim sumarizados por Cockburn e Highsmith (*Ibid.*):

1. A satisfação dos clientes passar a ter precedência frente à conformidade aos planos;
2. As mudanças sempre ocorrem – o foco deixa de ser como evitá-las e passa a ser como abraçá-las e como minimizar o seu custo ao longo do processo de desenvolvimento;

3. A eliminação das mudanças pode significar menosprezar condições importantes do negócio, ou seja, pode levar ao insucesso de uma organização;
4. O mercado espera um software inovador, com alta qualidade, que atenda aos requisitos do negócio e que esteja disponível em prazos cada vez menores.

Manifesto para o Desenvolvimento Ágil de Software


No início de 2001, criadores e representantes dos chamados Métodos Ágeis de Desenvolvimento de Software – Extreme Programming, Scrum, Dynamic Systems Development Method, Adaptive Software Development, Crystal Methods, Feature-Driven Development, Lean Development, entre outros – se reuniram nos Estados Unidos para discutir alternativas aos tradicionais “processos pesados” de desenvolvimento de software (BECK et al, 2001). Estes especialistas foram enfáticos em dizer que não eram contra métodos, processos ou metodologias, sendo que muitos até mencionaram o desejo de resgatar o verdadeiro significado e a credibilidade destas palavras. Defendiam a modelagem e a documentação, mas não em excesso. Planejavam, mas reconheciam os limites do planejamento e da previsibilidade num ambiente turbulento (BECK et al, 2001).

A essência deste movimento é a definição de novo enfoque de desenvolvimento de software, calcado na agilidade, na flexibilidade, nas habilidades de comunicação e na capacidade de oferecer novos produtos

PENSE...

QUANTO TEMPO
VOCÊ GASTARIA
PARA DESENVOLVER
COBRANÇA COM BOLETOS
BANCÁRIOS PARA
APENAS UM BANCO
NO SEU SOFTWARE

COBREBEMX



56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;

GERAÇÃO DE BOLETOS ON LINE;

GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;

MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

DOWNLOADS E INFORMAÇÕES EM WWW.COUREBEM.COM

Tecnologia

e serviços de valor ao mercado, em curtos períodos de tempo (HIGHSMITH, 2004, p. xix). Como agilidade deve-se entender “a habilidade de criar e responder a mudanças, buscando a obtenção de lucro, em um ambiente de negócio turbulento” (HIGHSMITH, 2004, p. 16); ou ainda, “a capacidade de balancear flexibilidade e estabilidade” (Id., 2002). A agilidade não deve ser vista como falta de estrutura, mas está diretamente relacionada à capacidade de adaptação a situações diversas e inesperadas. Highsmith (2004, p. 16) enfatiza que a ausência de estrutura ou de estabilidade pode levar ao caos, mas que estrutura em demasia gera rigidez.

Como resultado do encontro, foi criada a Agile Alliance⁴, sendo publicado o Manifesto para Desenvolvimento Ágil de Software ou o Manifesto for Agile Software Development (BECK et al, 2001), cujo conteúdo⁵ é apresentado abaixo:

“Nós estamos descobrindo melhores maneiras para desenvolver software, praticando e auxiliando os outros a fazê-lo. Através deste trabalho nós valorizamos:

- *Os indivíduos e suas interações acima de processos e ferramentas;*
- *Software em produção acima da documentação exaustiva;*
- *Colaboração do cliente acima da negociação de contratos;*
- *Respostas às mudanças acima da execução de um plano.*

Ou seja, embora haja valor nos itens à direita, nós valorizamos mais os itens à esquerda.”

Segundo Cohen et al (2003, p. 6), este Manifesto tornou-se a peça-chave do movimento pelo desenvolvimento ágil de software, uma vez que reúne os principais valores dos Métodos Ágeis, que os distingue dos métodos clássicos de desenvolvimento.

Além do Manifesto, foram definidos os princípios que regem a maioria das práticas dos chamados Métodos Ágeis de Desenvolvimento de Software (AGILE ALLIANCE, 2005). Estes princípios são apresentados na **Tabela 1** abaixo, de acordo com a ordem originalmente proposta.

Pode-se dizer, então, que os Métodos Ágeis se caracterizam por serem incrementais, cooperativos, diretos e adaptativos. *Incrementais*, dadas as pequenas versões e ciclos rápidos de desenvolvimento; *cooperativos*, por estimular a proximidade com o cliente e a interação entre os programadores; *diretos*, pela

simplicidade de aprendizado e de documentação; e, finalmente, *adaptativos*, pela habilidade de acomodar mudanças ao longo do projeto (ABRAHAMSSON et al, 2003; FOWLER, 2003).

Como principais diferenças entre os Métodos Ágeis e os clássicos de desenvolvimento de software podem ser citadas (FOWLER, 2003; CHIN, 2004):

- Os Métodos Ágeis são *adaptativos* e não *preditivos*: diferentemente dos enfoques clássicos que defendem o planejamento integral do escopo no início do projeto e um controle rígido de mudanças, os planos dos Métodos Ágeis são elaborados e adaptados ao longo do projeto, permitindo e, algumas vezes estimulando, a incorporação das mudanças requeridas pelo cliente;
- Os Métodos Ágeis são *orientados a pessoas* e não a processos: os processos clássicos têm desenvolvimento de software têm, em geral, a pretensão de funcionar independentemente de quem os executa. Já os Métodos Ágeis levam em consideração os indivíduos, sendo elaborados para auxiliá-las.

Princípios
Nossa maior prioridade é satisfazer o cliente através da entrega rápida e contínua de um software de valor
Pessoas de negócio e programadores devem trabalhar juntos, diariamente, ao longo de todo o projeto
Aceite as mudanças de requisitos, mesmo que numa etapa avançada do desenvolvimento
Entregue novas versões do software frequentemente
O software em funcionamento é a medida primária de progresso do projeto
Construa projetos com pessoas motivadas. Ofereça a elas o ambiente e todo o apoio necessários e acredite em sua capacidade de realização do trabalho
As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizadas
O método mais eficiente e efetivo de distribuir a informação para e entre uma equipe de desenvolvimento é a comunicação face a face
Processos ágeis promovem desenvolvimento sustentado
A atenção contínua na excelência técnica e num bom projeto aprimora a agilidade
Simplicidade é essencial
Equipes de projeto avaliam seu desempenho em intervalos regulares e ajustam seu comportamento de acordo com os resultados

Tabela 1. Princípios dos métodos ágeis de desenvolvimento de software (FONTE: AGILE ALLIANCE, 2005.)

Um ponto interessante a salientar é que enquanto alguns defensores “radicais” dos Métodos Ágeis são categóricos em criticar e apontar as falhas dos métodos clássicos de desenvolvimento de software, outros especialistas, representantes tanto dos métodos clássicos quanto dos ágeis, têm uma postura mais amena, enxergando até mesmo uma possibilidade de integração entre as duas abordagens (GLASS, 2001; COHEN et al, 2003; PAULK, 2002; GLAZER, 2001; HIGHSMITH, 2004). Glass (2001) apresenta uma análise do Manifesto para o Desenvolvimento Ágil de Software e menciona que, apesar de concordar com os pontos defendidos pelos praticantes dos Métodos Ágeis, não se deve desprezar os modelos clássicos e que ambos os lados têm pontos importantes a serem considerados e balanceados. Paulk (2002) defende que os princípios dos Métodos Ágeis devem ser seguidos por todos os profissionais que desenvolvem software,

4 Agile Alliance – Organização sem fins lucrativos criada para auxiliar indivíduos e organizações que utilizam os Métodos Ágeis para o desenvolvimento de software.

5 “We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interaction over process and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

That is, while there is a value in the items on the right, we value the items on the left more.”

mas lembra que formalismo e disciplina devem ser levados em conta, especialmente quando o software a ser desenvolvido envolve severos requisitos de confiabilidade. Paulk (*Id.*, 2001) analisa o Método Ágil *Extreme Programming* (XP) sob a ótica do SW-CMM, apontando como este poderia auxiliar as organizações a alcançar os objetivos propostos pelo SW-CMM.

Principais Métodos Ágeis de Desenvolvimento de Software

Dentre os principais Métodos Ágeis de Desenvolvimento de Software podem ser citados: *Extreme Programming*, *Scrum*, *Dynamic Systems Development Method*, *Adaptive Software Development*, *Crystal Method*, *Feature-Driven Development* e *Lean Development* (COHEN et al, 2003; UDO; KOPPENSTEINER, 2003). A seguir é feita uma breve explanação sobre as suas principais características.

Extreme Programming (XP)

De acordo com Cohen et al (2003, p.12) “o *Extreme Programming* (XP) é, indubitavelmente, o Método Ágil de maior expressão, criado nos últimos anos”. Por se tratar do Método Ágil mais difundido é abordado com mais de detalhe neste artigo.

O XP é um Método ágil para pequenas e médias equipes desenvolverem software, em ambientes com requisitos instáveis. Criado em 1998 por Kent Beck, Ron Jeffries e Ward Cunningham, a partir de um projeto piloto na Chrysler (BECK et al, 1998), o XP vem ganhando cada vez mais adeptos, ampliando sua participação no mercado (HIGHSMITH, 2002). Beck (2000, p.xv) explica que o termo *Extreme* (extremo) é utilizado, dado que o XP reúne um conjunto de práticas de desenvolvimento já existentes e reconhecidas como “boas práticas” no desenvolvimento de software, mas as leva ao extremo, ao limite. Juric (2002) comenta que o XP é uma maneira eficiente, de baixo risco, científica e divertida de desenvolver software.

A premissa básica do XP é que, ao contrário do que se pensava há 30 anos, o custo de mudança de um software não aumenta exponencialmente com o avançar do projeto (BECK, 2004, p. 21-23). As novas técnicas desenvolvidas pelos especialistas em software, como os bancos de dados relacionais e a programação modular, entre outras, permitiram uma redução no custo da mudança (ver Figura 1). Desta forma, não se faz mais necessário evitar mudanças durante o desenvolvimento, sendo possível abandonar o *Modelo em Cascata* e usar o XP.

O XP baseia-se em 12 práticas ou regras concisas e diretas, listadas abaixo (BECK, *Ibid.*; COHEN et al, 2003, p.12, BECK; FOWLER, 2001, p. 72):

1. *Jogo do planejamento*: no início de cada interação, clientes, gerentes e programadores se encontram para definir, estimar e priorizar os requerimentos. A ideia é que se elabore um plano aproximado no início no projeto e se faça um refinamento à medida que as necessidades e requisitos se tornem mais conhecidos;

2. *Programação em pares*: dois programadores utilizando o mesmo equipamento escrevem o código;

3. *Pequenas versões*: as versões devem ser tão pequenas quanto possível e trazerem valor para o negócio. Uma versão inicial do software deve ser colocada em produção após um pequeno número de iterações e, em seguida, outras versões devem ser disponibilizadas tão logo faça sentido;

4. *Metáforas*: clientes, gerentes e programadores criam metáforas ou conjunto de metáforas para modelagem do sistema;

5. *Projeto simples*: os programadores são estimulados a desenvolver o código do software o mais simples possível;

6. *Testes*: os programadores devem criar os testes de unidade antes ou mesmo durante o desenvolvimento do código do sistema. Os clientes, por sua vez, escrevem os testes de aceitação. Ao final de cada iteração a bateria de testes deve ser conduzida;

7. *Refatoração*: técnica que permite a melhoria de código sem a mudança de funcionalidade (BRASIL, 2001b, p. 186), deve ser executada pela equipe do projeto, o tempo todo;

8. *Integração contínua*: os programadores devem integrar os novos códigos ao software tão rapidamente e com a maior frequência possível;

9. *Propriedade coletiva do código*: o código do programa deve ser propriedade de toda a equipe e qualquer integrante pode fazer alterações sempre que for necessário;

10. *Cliente no local*: o cliente deve trabalhar com a equipe de projeto a todo o momento, respondendo perguntas, realizando testes de aceitação e assegurando que o desenvolvimento do software esteja sendo feito a contento;

11. *Semana de 40 horas*: como trabalhar por longos períodos reduz o desempenho, o conteúdo de cada iteração deve ser planejado de forma a não haver necessidade de realização de horas extras, fazendo com que os programadores estejam renovados e ansiosos a cada manhã e cansados e satisfeitos à noite;

12. *Padrão de codificação*: no início do projeto deve ser criado um padrão de codificação, simples e aceito por toda a equipe, que deverá ser seguido de forma a não permitir a identificação de quem desenvolveu determinada parte do código e a auxiliar a condução do trabalho.

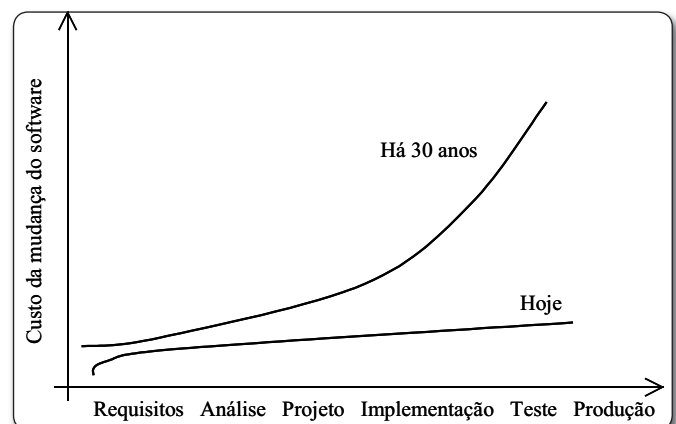


Figura 1. Custo da mudança do software (FONTE: BECK, 2004, p. 21-23)

Especialistas concordam que o XP não é decorrência da aplicação destas práticas isoladamente, mas sim do resultado de sua combinação (COHEN *et al*, 2003, p.13). HIGHSMITH (2002) ainda ressalta que cinco princípios constituem a base do XP: comunicação, simplicidade, *feedback*, coragem e qualidade de trabalho.

Cohen *et al* (2003, p.13) apresentam um resumo de características principais que norteiam a aplicação do *Extreme Programming*, retratado na **Tabela 2**.

Característica	Valores sugeridos
Tamanho da Equipe	Equipes formadas por 2 a 10 integrantes
Duração das iterações	Duração usual de duas semanas por iteração
Equipes distribuídas	Dada que a equipe deve trabalhar preferencialmente no mesmo local físico, o XP não é indicado para equipes distribuídas
Aplicações de alta criticidade	Pode ser utilizado no desenvolvimento de software de baixa, média ou alta criticidade

Tabela 2. Características principais para utilização do XP (FONTE: COHEN *et al*, 2003, p.13.)

Scrum

Criado por Ken Schwaber e Jeff Sutherland em 1996, como um método que aceita que o desenvolvimento de software é imprevisível e formaliza a abstração, o *Scrum* é aplicável a ambientes voláteis (SCHWABER, 2002). É uma abordagem empírica baseada na flexibilidade, adaptabilidade e produtividade, em que a escolha das técnicas de desenvolvimento fica a cargo dos programadores. Segundo Udo e Koppensteiner (2003), o *Scrum* se destaca dos demais Métodos Ágeis pela maior ênfase dada ao gerenciamento do projeto. Há atividades específicas de monitoramento e *feedback*, em geral, reuniões rápidas e diárias com toda a equipe, visando à identificação e à correção de quaisquer deficiências e/ou impedimentos no processo de desenvolvimento (SCHWABER; BEEDLE, 2001).

Schwaber (2002) sumariza assim os princípios-chave do *Scrum*:

- Equipes pequenas de trabalho, buscando a maximização da comunicação e da troca de conhecimento tácito e informal e minimização de *overhead*;
- Adaptação às solicitações de mudanças técnicas ou do cliente / usuário, assegurando a entrega do melhor software possível;
- Entregas frequentes de versões que podem ser testadas, ajustadas, executadas, documentadas e liberadas para produção;
- Divisão do trabalho e das responsabilidades da equipe de projeto em pequenas entregas;
- Habilidade em entregar um software pronto quando da necessidade do cliente ou do negócio.

As características principais que norteiam a aplicação do *Scrum* são apresentadas na **Tabela 3** (COHEN *et al*, 2003, p.15).

Característica	Valores sugeridos
Tamanho da Equipe	O trabalho é dividido em equipes de até sete pessoas
Duração das iterações	Duração usual de quatro semanas por iteração
Equipes distribuídas	Como o projeto pode ser constituído por várias pequenas equipes, há a possibilidade de que estas estejam distribuídas (descentralizadas)
Aplicações de alta criticidade	Não há menção específica quanto à aplicabilidade do método para desenvolvimento de software de alta criticidade

Tabela 3. Características principais para utilização do Scrum (FONTE: COHEN *et al*, 2003, p.16-17.)

Crystal Methods

Criado por Alistair Cockburn no início dos anos 90, a partir da crença de que os principais obstáculos enfrentados no desenvolvimento de produtos recaíam sobre os problemas de comunicação, os *Crystal Methods* dão grande ênfase às pessoas, à comunicação, às interações, às habilidades e aos talentos individuais, deixando os processos em segundo plano (UDO; KOPPENSTEINER, 2003; COCKBURN, 2004, COHEN *et al*, 2003). Correspondem a uma família de métodos organizados por cores, de acordo com o número de pessoas envolvidas (tamanho do projeto x necessidade de comunicação), com as prioridades do negócio e com a complexidade e a criticidade do software a ser desenvolvido, conforme mostra a **Figura 2**.

Apesar da estrutura proposta servir como um guia dos processos mais adequados a uma determinada situação, nos *Crystal Methods*, a definição final dos processos a serem utilizados é responsabilidade da equipe de projeto. Mas duas regras principais são sempre seguidas: ciclos de desenvolvimento incrementais com duração de no máximo quatro meses e reuniões de reflexão que estimulam a colaboração entre integrantes da equipe de projeto (UDO; KOPPENSTEINER, 2003; COCKBURN, 2004; COHEN *et al*, 2003).

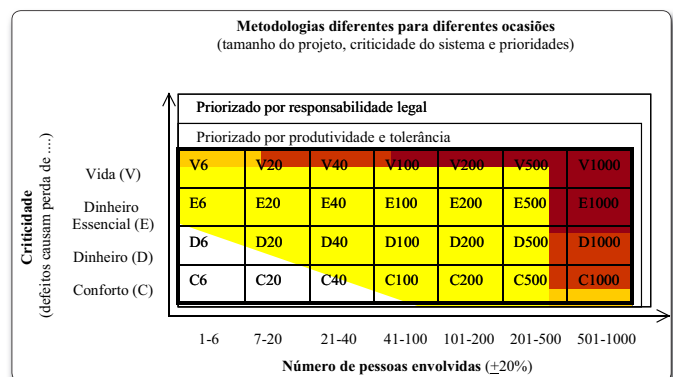


Figura 2. Esquema dos Crystal Methods (FONTE: COHEN *et al*, 2003, p. 16)

Cohen *et al* (2003, p.15) apontam as características principais de projetos que utilizam os *Crystal Methods* (**Tabela 4**).

Dynamic Systems Development Method (DSDM)

Originário da Inglaterra, em meados dos anos 90, o *Dynamic Systems Development Method* é controlado por um consórcio de empresas. Criado a partir do RAD – *Rapid Application Development*, o DSDM é o único método ágil compatível com a ISO 9000. Seu ciclo de vida é dividido nos seguintes estágios:

a) Pré-projeto; b) Análise de Aderência; c) Estudo de Negócio; d) Modelagem Funcional; e) Projeto e Desenvolvimento; f) Implementação, e, g) Pós-implementação. A ideia central do DSDM é que se deve primeiramente fixar o prazo e os recursos para, em seguida, definir e ajustar o número de funcionalidades a serem desenvolvidas (COHEN *et al*, 2003; UDO; KOPPENSTEINER, 2003).

Característica	Valores sugeridos
Tamanho da Equipe	A família dos Crystal Methods acomoda equipes de qualquer tamanho, preferencialmente compostas por pessoas talentosas
Duração das iterações	Até 4 meses para projetos grandes e altamente críticos
Equipes distribuídas	Os Crystal Methods foram concebidos para atender ao conceito de equipes distribuídas
Aplicações de alta criticidade	Os Crystal Methods atendem a projetos críticos, incluindo aqueles que envolvem risco de vida e de valores monetários

Tabela 4. Características principais para utilização dos Crystal Methods (FONTE: COHEN *et al*, 2003, p.15.)

Dadas a sua natureza, o DSDM não endereça um tamanho de equipe específico e não possui durações pré-determinadas para suas iterações (COHEN *et al*, *op. cit.*). Não foram encontradas na literatura recomendações específicas para utilização no desenvolvimento de aplicações de determinada criticidade ou para equipes centralizadas ou descentralizadas.

Feature-Driven Development (FDD)

O *Feature-Driven Development*, criado por Peter Coad e Jeff DeLuca em 1999, é um método de desenvolvimento de software específico para aplicações críticas de negócio (PALMER; FELSING, 2001). Diferentemente de outros Métodos Ágeis, o FDD se baseia em processos bem definidos e que podem ser repetidos. Sua abordagem se concentra nas fases de projeto e construção, com maior ênfase na modelagem, em um ciclo de vida iterativo e também em atividades de gerenciamento de projetos (UDO; KOPPENSTEINER, 2003). Os princípios-base do FDD são apontados abaixo (HIGHSMITH, 2002):

- Necessidade de se automatizar a geração de software para projetos de grande escala;
- Um processo simples e bem definido é fundamental;
- As etapas de um processo devem ser lógicas e óbvias para cada integrante da equipe de desenvolvimento;
- Bons processos atuam na retaguarda, permitindo que a equipe se dedique ao alcance dos resultados;
- Ciclos de vida curtos e iterativos são mais indicados.

Um projeto conduzido pelo método FDD possui as seguintes etapas: a) Desenvolvimento de um modelo geral; b) Construção da lista de funcionalidades; c) Planejamento por funcionalidades; d) Projeto e desenvolvimento por funcionalidades.

A **Tabela 5** apresenta as principais características de um projeto desenvolvido pelo método FDD (COHEN *et al*, 2003, p.18).

Característica	Valores sugeridos
Tamanho da Equipe	Variável de acordo com a complexidade das funcionalidades a serem desenvolvidas
Duração das iterações	Até 2 semanas
Equipes distribuídas	O FDD foi criado para trabalhar com equipes múltiplas e, apesar de não haver indicação formal para equipes distribuídas, é passível de adaptação
Aplicações de alta criticidade	Indicado para desenvolvimento de software crítico

Tabela 5. Características principais para utilização do FDD (FONTE: COHEN *et al*, 2003, p.15.)

Lean Development (LD)

Com raízes na indústria automotiva dos anos 80, em especial no Modelo Toyota de Produção, o *Lean Development* é considerado o Método Ágil com maior foco estratégico. Iniciado por Bob Charette, o LD tem como principais objetivos reduzir em um terço o prazo, o custo e o nível de defeitos no desenvolvimento de software. Para tanto, requer um grande comprometimento da alta administração e uma predisposição a mudanças radicais (COHEN *et al*, 2003; UDO; KOPPENSTEINER, 2003). HIGHSMITH (2002) aponta como princípios fundamentais do *Lean Development*:

- A satisfação do cliente é a prioridade principal;
- Prover sempre o maior valor possível para o dinheiro;
- O sucesso depende da participação ativa dos clientes;
- Todo projeto baseado no LD requer um esforço conjunto de toda a equipe;
- Tudo pode ser mudado;
- Domine, não aponte as soluções;
- Complete, não desenvolva;
- Prefira uma solução a 80% hoje, a uma solução a 100% amanhã;
- O minimalismo é essencial;
- A necessidade determina a tecnologia;
- O incremento do produto corresponde a um incremento de funcionalidade e não de tamanho;
- Nunca force o LD além de seus limites.

Cohen *et al* (2003, p. 19) afirma que “[...] como o *Lean Development* é mais uma filosofia de gerenciamento que um processo de desenvolvimento”, os itens referentes ao tamanho da equipe, à duração das iterações, ao tratamento de equipes centralizadas ou distribuídas e à criticidade da aplicação não são diretamente endereçados pelo método.

Adaptive Software Development (ASD)

Criado por Highsmith como uma evolução do RAD – *Rapid Application Development* em 1992, o *Adaptive Software Development* propõe uma forma alternativa de se enxergar o desenvolvimento de software nas organizações (HIGHSMITH, 2002). O ASD foi projetado para lidar com ambientes repletos de incertezas e complexos. Segundo Udo e Koppensteiner (2003), o método estimula a aprendizagem durante o processo de desenvolvimento e a adaptação constante às novas realidades do negócio e do projeto. Além disso, encoraja o desenvolvimento

iterativo e incremental, com a liberação constante de novas versões. O ASD oferece estrutura e orientação suficiente para evitar que os projetos se tornem caóticos, sem, entretanto, trazer uma rigidez indesejada que venha a suprimir a criatividade. Neste método, o papel do gerente de projetos é favorecer a colaboração entre a equipe de desenvolvimento e o cliente.

De acordo com Highsmith (2002), o ASD é indicado para equipes pequenas, mas pode ser adaptado para equipes maiores. Com relação aos demais atributos – duração das iterações, apoio a equipes distribuídas e desenvolvimento de aplicações de alta criticidade – não foi encontrada uma indicação precisa na literatura.

Resumo das Características Principais dos Métodos Ágeis

Nos tópicos anteriores foram apresentadas as principais características de alguns dos chamados Métodos Ágeis de Desenvolvimento de Software. É importante mencionar que estes métodos foram selecionados por serem os mais citados na literatura.

Como visto, apesar dos Métodos Ágeis terem uma essência ou valores em comum, há algumas diferenças significativas entre eles. A **Tabela 6** apresenta um resumo das principais características dos Métodos Ágeis analisados, com uma linha específica destinada à análise da incorporação ou não de práticas relacionadas ao gerenciamento de projetos.

Aplicação dos Métodos Ágeis nas Organizações

Há uma grande variedade de Métodos Ágeis de Desenvolvimento de Software, cada qual sugerindo práticas específicas, cuja adoção traz mudanças às rotinas das organizações. De acordo com Nerur *et al* (2005, p.74), “[...] as alterações nos processos de desenvolvimento de software representam um fenômeno complexo de mudança organizacional que não pode ser alcançado pela mera substituição de ferramentas e tecnologias”. Estas mudanças têm impacto significativo em vários aspectos da organização: na estrutura, na cultura e nas práticas gerenciais. Conhecer a amplitude deste fenômeno é fator crítico para o planejamento e o gerenciamento destas mudanças. Sendo assim, antes de uma organização adotar e implementar qualquer um dos Métodos Ágeis é fundamental que avalie a dimensão do impacto e a sua prontidão para tal (AMBLER, 2002c; COHEN *et al*, 2003; FOWLER, 2003; NERUR *et al*, 2005).

Para facilitar o entendimento e retomar as principais características e diferenças entre os enfoques clássico e ágil de desenvolvimento de software, é traçado paralelo

entre eles na **Tabela 7**. Estas diferenças entre as abordagens sugerem que as organizações devem repensar suas metas, seus objetivos e reconfigurar seus componentes humanos, gerencial e tecnológico, de forma a alcançar uma implementação bem-sucedida dos métodos ágeis (NERUR *et al*, 2005, p.75). Enfocando o componente gerencial, estas diferenças podem demandar até mesmo uma alteração no enfoque de gerenciamento de projetos utilizado para as iniciativas de desenvolvimento de software.

Ambler (2002c) discute os fatores que influenciam a adoção bem-sucedida de Métodos Ágeis, enfatizando que o ponto mais importante para que isto aconteça é a existência de uma compatibilidade entre os conceitos e valores da organização e dos Métodos Ágeis, ou seja, o autor discute claramente a questão da cultura da organização. Na mesma linha, Nerur *et al* (2005) destacam que os valores, as normas e os padrões estabelecidos e reforçados pelas organizações ao longo do tempo se refletem nas políticas e nas rotinas da empresa e exercem considerável influência nos processos de tomada de decisão, nas estratégias de solução de problemas, nas práticas voltadas à inovação, nos filtros de informação, nos relacionamentos interpessoais e nas negociações. Como a cultura e a forma de pensar das pessoas não são facilmente modificáveis, pode-se dizer que a adoção de Métodos Ágeis seja indicada apenas a algumas organizações (AMBLER, 2002; NERUR *et al*, 2005).

Um segundo fator importante a ser considerado quando da decisão pela implementação de Métodos Ágeis, de acordo com Ambler (*op. cit.*) é o projeto e as características do negócio. Questões como – Os trabalhos têm sido conduzidos de forma incremental? Qual é a motivação da equipe de projeto? Qual o nível de apoio que a equipe de desenvolvimento pode esperar? Os recursos adequados estão disponíveis? Quão voláteis são os requisitos do projeto? – são fundamentais para esta avaliação. Com relação ao último ponto, Bohem (2002) chega a sugerir que os métodos clássicos deveriam ser preferidos quando o índice de alteração de requisitos do projeto for inferior a 1% ao mês.

Um terceiro aspecto destacado por Ambler (*op. cit.*) é a necessidade de definição de um *champion*, ou seja, de um profissional que assuma os desafios do projeto, de forma que a equipe de desenvolvimento possa trabalhar com tranquilidade. Ampliando esta análise, uma vez que os Métodos Ágeis valorizam e depositam elevado grau de confiança no conhecimento tácito e na capacidade de tomada de decisões de cada indivíduo, Bohem (*op. cit.*) enfatiza a importância de se ter também uma equipe de projeto bem treinada e composta por especialistas. Com relação a este aspecto, apesar de concordar com a

Característica	XP	Scrum	Crystal Methods	FDD	ASD	LD	DSDM
Tamanho da Equipe	2-10	1-7	Variável	Variável	Variável	Não definido	
Duração das iterações	2 semanas	4 semanas	< 4 meses	< 2 semanas			
Equipes distribuídas	Não	Adaptável	Sim	Adaptável			
Aplicações de alta criticidade	Adaptável	Adaptável	Todos os tipos	Adaptável			
Práticas de gerenciamento de projetos	Poucas	Muitas	Poucas	Muitas	Muitas		

Tabela 6. Características principais dos métodos ágeis selecionados (FONTE: Adaptado de COHEN *et al*, 2003, p. 23.)

	Enfoque Clássico	Enfoque Ágil (Métodos Ágeis)
Premissa Fundamental	O software é totalmente especificável e previsível e pode ser construído através de um planejamento meticuloso e extensivo	Software adaptativo e de alta qualidade pode ser construído por pequenas equipes, com o uso de princípios como a melhoria contínua do projeto e o desenvolvimento e testes baseados no rápido feedback e na aceitação de mudanças
Estilo de Gerenciamento	Comando e controle	Liderança e colaboração
Distribuição de Papéis	Individual, favorecendo a especialização	Equipes auto-organizadas, encorajando o intercâmbio de papéis
Papel do Cliente	Importante	Crítico
Modelo de Desenvolvimento	Modelo de ciclo de vida – Modelo em Cascata, Espiral e iterativos	Métodos Ágeis
Tecnologia	Sem restrições	Favorece a tecnologia orientada a objetos

Tabela 7. Comparação entre os enfoques clássico e ágil de desenvolvimento de software (FONTE: ADAPTADO DE NERUR et al, 2005, p. 75.)

necessidade de formação de uma equipe especializada, Nerur *et al* (2005) chama a atenção para que não se crie uma cultura de elitismo dentro do grupo de desenvolvimento, que pode afetar o moral e o comprometimento de outros profissionais da empresa, não integrantes da equipe.

É inegável também que os Métodos Ágeis trazem consigo uma mudança radical na relação cliente – fornecedor. Cockburn e Hignsmith (2001a), Bohem (2002) e Nerur *et al* (2005) apontam o envolvimento e a participação dos clientes como fatores imprescindíveis para o sucesso da implementação dos métodos ágeis. Os clientes devem estar totalmente comprometidos com o projeto, trabalhar com espírito de colaboração, possuir o conhecimento necessário, ter autonomia para a tomada de decisões, além de estar disponíveis para sanar as dúvidas quando necessário. Entretanto, por se tratar de um item extremamente crítico no processo e que demanda tempo, paciência e esforço consideráveis para o estabelecimento de um ambiente de respeito e confiança mútua entre clientes e fornecedores, é mencionado por alguns autores, entre eles Nerur *et al* (*Ibid.*), como um dos obstáculos à utilização dos Métodos Ágeis.

Compartilhando os pontos discutidos nos parágrafos anteriores, Cohn e Ford (2003, p. 74) acrescentam que transição de um processo clássico com ênfase no “planejamento – execução – controle”, para um processo ágil, afeta não apenas a equipe de desenvolvimento de software, mas também outras equipes, departamentos, assim como o corpo gerencial da organização. Tomando por base a experiência empírica de implementações de Métodos Ágeis em várias organizações, abrangendo tanto projetos simples como projetos complexos, equipes centralizadas ou distribuídas, os autores sugerem uma abordagem diferenciada junto aos principais envolvidos nos processos – programadores, alta gerência e a área de Recursos Humanos – para que se garanta uma implementação bem-sucedida deste novo enfoque de desenvolvimento de software.

Segundo Cohn e Ford (2003, p. 74 - 75), usualmente os programadores respondem à implementação de Métodos Ágeis com um misto de ceticismo, entusiasmo, otimismo, ou mesmo, resistência. Como este novo enfoque, o desenvolvimento de software normalmente prioriza a entrega do código à geração de uma documentação extensiva, a adaptação ao planejamento completo e, as pessoas e suas interações aos processos e ferramentas (BECK *et al*, 2001). Cohn e Ford (*op.*

cit.) afirmam que há sentimentos controversos entre os integrantes da equipe durante o processo de transição: alguns se sentem perdidos ou sem um direcionamento, pela ausência de um cronograma formal ou de uma documentação completa de requisitos; outros empolgados, pelo reconhecimento de sua capacidade e pela autonomia concedida; alguns creem erroneamente que a adoção dos Métodos Ágeis tem o intuito de estimular a microgerência, dados os encontros e as reuniões constantes entre gerentes e equipe, em métodos como o *Scrum* e XP.

Em face desta situação, Cohn e Ford (2003, p. 75), assim como Ambler (2002c), defendem uma passagem gradual dos processos clássicos de desenvolvimento para os Métodos Ágeis, tornando o período de transição mais tranquilo.

A idéia de uma equipe formada por talentos, citada por Boehm (2002) é ratificada por Cohn e Ford (2003, p. 75) e por Ambler (2002c), que explicam ainda que diferenças grandes de produtividade numa equipe podem trazer impactos negativos ao projeto, uma vez que num Método Ágil, a equipe se dedica apenas às atividades essenciais para a entrega do software. Não se deve esquecer, contudo, que uma pequena queda de produtividade é esperada durante a transição, até que toda a equipe esteja ambientada com a nova dinâmica de trabalho. Fowler (2003) destaca que a equipe técnica não pode ser responsável por todo o trabalho por si só, sendo fundamental o papel dos gerentes, fornecendo o direcionamento, auxiliando a definição das prioridades de negócio, removendo obstáculos e negociando os prazos de entrega. Com esta colocação, Fowler (*Ibid.*) enfatiza a importância do gerenciamento de projetos mesmo no desenvolvimento de software conduzido com o uso de Métodos Ágeis.

Ao considerar a questão de equipes distribuídas, Cohn e Ford (*op. cit.*) são enfáticos em dizer que, mesmo havendo a necessidade de se organizar um projeto de forma descentralizada, deve-se fazer o possível para que nas primeiras semanas de trabalho, a maior parte da equipe esteja reunida e somente após este período as equipes sejam distribuídas. Os autores justificam este ponto ao afirmar que equipes que trabalham segundo os Métodos Ágeis necessitam tomar decisões de forma mais rápida que nos processos tradicionais, mas para tanto, recorrem a uma comunicação mais frequente e normalmente informal. Sem esta proximidade inicial, a comunicação pode ser comprometida.

Analisando outro grupo de interessados, Cohn e Ford (2003, p.76) apontam que normalmente a alta gerência representa um desafio singular à adoção dos Métodos Ágeis. Basicamente as suas preocupações recaem sobre quatro pontos fundamentais, extremamente vinculados à visão do Gerenciamento Clássico de Projetos:

1. Como é possível prometer novas funcionalidades aos clientes?
2. Como mensurar o progresso do projeto?
3. Quando o projeto acaba?
4. Como a utilização de um Método Ágil afetará outros grupos?

Apesar de pertinentes, a origem destes questionamentos está na dificuldade que muitos gerentes têm em abrir mão do processo clássico de planejamento e controle, da solicitação de compromissos formais de entrega, mesmo sabendo que raramente estes objetivos são cumpridos pelas equipes de desenvolvimento⁶. Cohn e Ford (2003, p.76) e Highsmith (2004) salientam que gerentes que temem que com os Métodos Ágeis não seja possível a fixação de datas de entrega de produtos a clientes, deveriam lembrar que, no passado, os planos formais provedores de “garantias” de prazo, custo e qualidade estavam usualmente errados, superestimados, ou ambos. Em organizações com histórico de estimativas incorretas de projetos, convencer a alta gerência sobre os benefícios dos Métodos Ágeis pode não ser algo difícil, bastando uma avaliação dos resultados passados frente aos objetivos iniciais de prazo, custo e qualidade e a apresentação dos benefícios potenciais da nova abordagem. Já nos casos em que a equipe de desenvolvimento entrega seus projetos sistematicamente no prazo e no custo, a utilização dos Métodos Ágeis pode ser justificada com vistas à redução de prazos de entrega, à redução das equipes de trabalho, o que significaria um ganho para a organização.

Apesar dos Métodos Ágeis proporem uma mudança do paradigma de “comando e controle” para “liderança e colaboração”, conforme explica Nerur *et al* (2005, p.75), isto não significa que sejam alheios à necessidade de mensuração do progresso dos projetos. Segundo Cohn e Ford (2003, p.77), os Métodos Ágeis podem oferecer um acompanhamento adequado do projeto, por meio da utilização de relatórios específicos a cada iteração, que contêm datas-chave, comparação de resultados reais *versus* planejados, métricas principais e até mesmo a identificação dos riscos do projeto. Estas são práticas do gerenciamento de projetos, mas que neste caso, são aplicadas a cada interação e não ao projeto como um todo.

Outra preocupação bastante comum à alta gerência, refletida na terceira questão, é que aparentemente um projeto realizado segundo um Método Ágil tende a não ter fim, uma

vez que as iterações podem continuar enquanto houver itens prioritários, definidos pelo cliente, a serem desenvolvidos. Cohn e Ford (2003, p.77) apontam como solução para esta questão, a definição de um intervalo de prazo e custo, vinculado a um escopo macro, que servem como uma estimativa preliminar do projeto, a ser considerada durante a execução do projeto. A equipe deve buscar entregar uma versão básica e funcional do software, no prazo mínimo deste intervalo e, a partir daí, são negociadas entregas complementares (outras iterações). Desta forma, Cohn e Ford (2003, p. 77), afirmam que é possível minimizar o desconforto quanto à finalização do projeto.

Considerando a quarta questão, a implementação de um Método Ágil pode afetar além da equipe de desenvolvimento, os gerentes, os clientes e, até mesmo, áreas ou departamentos internos à empresa que, numa análise inicial, não têm nenhum vínculo aparente com o projeto. Sendo assim, é fundamental que a alta gerência tenha ciência de quais grupos ou departamentos serão afetados pela adoção dos Métodos Ágeis, para que se estabeleça um consenso quanto à forma de trabalho e se evite desgastes ou problemas futuros. Neste contexto, deve-se destacar a importância da participação da área ou departamento de Recursos Humanos no processo de transição para a utilização dos Métodos Ágeis (COHN; FORD, 2003, p. 78). Representantes desta área devem ser envolvidos logo no início da implementação, para que tenham ciência da nova forma de trabalho e forneçam todo o apoio necessário, sanando dúvidas e amenizando ansiedades dos envolvidos no processo de mudança.

Complementando esta análise, COHEN *et al* (2003, p. 31) selecionam um conjunto de lições aprendidas, que consideram úteis quando da decisão pela adoção de um Método Ágil, algumas das quais rebatem as críticas mais comuns atribuídas a este novo enfoque de desenvolvimento de software:

- Os Métodos Ágeis podem ser aplicados a equipes de qualquer tamanho, entretanto, deve-se ter em mente que quanto maior a equipe, maior a dificuldade de comunicação;
- Experiência é importante para que um projeto ágil seja bem-sucedido, mas a experiência técnica em desenvolvimento de software é muito mais importante que a experiência prévia com os Métodos Ágeis;
- Os Métodos Ágeis requerem menos treinamento formal que os métodos clássicos. Técnicas como a programação em pares minimizam a necessidade de treinamento, pois as pessoas aprendem umas com as outras. Os treinamentos formais podem ser realizados por meio de auto-estudo;
- Um software crítico e de alta confiabilidade pode ser construído com a utilização de Métodos Ágeis. Neste caso, é fundamental que os requisitos de desempenho sejam explicitados logo no início do projeto e os níveis adequados de teste planejados. Ao solicitar *feedback* constante, incentivar a participação dos clientes e a definição de prioridade por eles, os Métodos Ágeis tendem a antecipar resultados e minimizar os riscos do projeto;

⁶ Pesquisa publicada pelo STANDISH GROUP INTERNATIONAL (2003), referente a projetos de desenvolvimento de software, aponta que usualmente os projetos ultrapassam em 82% os prazos inicialmente previstos. Em 1999, este índice de atraso chegou ao valor aproximado de 222%.

- Os três principais fatores críticos de sucesso para um projeto ágil são: cultura, pessoas e comunicação. Os Métodos Ágeis exigem uma compatibilidade cultural para serem bem-sucedidos; profissionais competentes são fundamentais: os Métodos Ágeis usam menos técnicos, porém mais qualificados; equipes centralizadas facilitam a comunicação e a interação próxima com cliente é fator base para seu funcionamento;
- Os Métodos Ágeis permitem a identificação rápida de eventuais problemas no projeto, através de pequenos sinais, como a queda da motivação da equipe nas reuniões diárias, atrasos nas entregas das iterações e a geração de documentação desnecessária;
- A documentação deve ser considerada um custo e sua extensão deve ser determinada pelo cliente. Deve-se buscar desenvolver uma comunicação mais efetiva, mantendo a documentação formal em um patamar mínimo necessário.

A forma como um Método Ágil é introduzido em uma organização e os cuidados tomados durante este processo podem determinar o sucesso ou o fracasso da iniciativa. Outro importante desafio enfrentado pelos gerentes das organizações refere-se à escolha do Método Ágil mais apropriado para o momento e o projeto em questão, em face da variedade de métodos atualmente disponíveis no mercado (NERUR *et al*, 2005, p.77). Ambler (2002c) e Cohn e Ford (*op. cit.*) mencionam que, se após uma análise detalhada, os Métodos Ágeis não se apresentarem totalmente compatíveis com o projeto ou com os princípios da organização, mas suas idéias ainda assim despertarem interesse, pode-se partir para uma adoção parcial. Nesta estratégia, deve-se identificar os pontos de melhoria prioritários no processo clássico de desenvolvimento de software e aplicar algumas práticas dos Métodos Ágeis, visando ao aprimoramento. Obtendo-se um resultado positivo, procede-se a uma nova seleção de áreas de melhoria e implantação de novas técnicas, até que todo o Método Ágil tenha sido adotado.

Por fim, Highsmith (2004), Cohn e Ford (2003) e Nerur *et al* (2005) ressaltam que muitas das inseguranças e incertezas inerentes a este processo de transição podem ser minimizadas com a adoção de um enfoque de gerenciamento de projetos adequado e compatível com o desenvolvimento de software conduzidos com o uso de Métodos Ágeis.

Resultados da Aplicação dos Métodos Ágeis

Como os Métodos Ágeis de Desenvolvimento de Software são relativamente recentes, poucos são os dados empíricos disponíveis referentes aos resultados de sua aplicação nas organizações, alguns dos quais são apresentados a seguir.

Pesquisa realizada por Reifer (2002, p. 14-17) em 32 organizações, representando 28 empresas de dez segmentos de mercado distintos, apontou que dentre elas, 14 organizações adotavam Métodos Ágeis, todas motivadas por um histórico de baixo desempenho dos projetos de desenvolvimento de software quanto ao cumprimento dos objetivos de prazo e custo. Surpreendentemente, a maioria das empresas analisadas era classificada como nível dois ou superior, na escala de maturidade nos processos de desenvolvimento de software proposta pelo SW-CMM, ou seja, possuíam processos relativamente maduros. No entanto, estas empresas buscavam algo novo, que pudesse reverter o quadro de mau desempenho consistente de seus projetos. A distribuição destas empresas entre os segmentos de atuação, assim como alguns detalhes quanto ao início da prática, à quantidade de projetos realizados com o uso dos Métodos Ágeis e ao estágio de implementação do novo método são apresentados na **Tabela 8**.

Os projetos pesquisados são qualificados como de pequeno porte (com menos de 10 participantes), em geral, internos, de baixo risco, com duração menor que 12 meses, mas que exigiam elevado grau de flexibilidade.

Com relação aos resultados, 50% das organizações que responderam a pesquisa conseguiram mensurar os ganhos obtidos com a utilização de Métodos Ágeis de Desenvolvimento de Software de forma concreta, sendo que muitas, por possuírem métricas anteriores, realizaram comparações bastante efetivas. Reifer (2002, p.15) aponta como principais ganhos, já normalizados entre todos os participantes, os seguintes itens:

- Incremento de produtividade: ganhos de 15% a 23% de produtividade frente aos indicadores da indústria;
- Redução de custos: 5% a 7% de redução de custos quando comparado aos indicadores da indústria;
- Redução do tempo de entrega: 25% a 50% de redução de prazo comparando-se com projetos anteriores realizados pelas empresas;
- Melhoria de qualidade: cinco empresas que possuíam dados concretos apontaram que a taxa de defeitos estava no mesmo nível dos projetos anteriores quando da liberação de produtos e aplicações.

Indústria	# Empresas que utilizam Métodos Ágeis	# de Projetos	Início da Utilização de Métodos Ágeis	Estágio da Implementação
Aeroespacial	1	1	2001	Descoberta
Computação	2	3	2000	Piloto
Consultoria	1	2	2000	Piloto
Comércio Eletrônico	5	15	2000	Produção
Pesquisa	1	1	2000	Piloto
Software	2	4	2000	Produção
Telecomunicações	2	5	2000	Produção
Total	14	31	n/a	n/a

Tabela 8. Características das empresas pesquisadas (FONTE: ADAPTADO DE REIFER, 2002, p. 15.)

As demais organizações analisadas, que não possuíam indicadores formais, mensuraram seus ganhos através da opinião dos principais interessados nos projetos, listando como benefícios da utilização dos Métodos Ágeis, o alto grau de motivação dos profissionais envolvidos, a elevação da moral da equipe e alguns outros argumentos intangíveis.

De forma unânime, todas as organizações afirmaram sua intenção de continuar ou mesmo de estender a utilização dos Métodos Ágeis, considerando-a uma experiência bastante proveitosa e bem-sucedida. Mas apesar dos resultados animadores, Reifer (2002, p.15), chama a atenção para que não sejam tiradas conclusões precipitadas, nem se faça uma generalização dos resultados, dados o tamanho da amostra (14 organizações) e o tipo de projetos envolvidos (projetos pequenos, de baixo risco, em ambiente relativamente controlado).

Maurer e Martel (2002) relataram o caso de uma companhia – Bitonic Solutions Inc. – sediada no Canadá, cuja equipe de sistemas, composta por nove profissionais, desenvolveu uma aplicação para comércio eletrônico. O processo de desenvolvimento foi iniciado segundo uma abordagem orientada a objetos *ad hoc*, sendo adotado o método ágil *Extreme Programming* (XP), depois de determinado período. Para a mensuração da produtividade do desenvolvimento, os autores definiram três indicadores principais, cujos valores foram divididos pelo esforço despendido para execução (em horas), a saber:

- Novas linhas de código (NLOC)
- Número de novos métodos (# métodos);
- Número de novas classes (# classes).

Maurer e Martel (2002) coletaram métricas do desenvolvimento nos dois períodos (anterior e posterior ao uso do XP) e reportaram ganhos de produtividade que variaram de 66,3% a 302,1%, conforme mostra a **Tabela 9**.

Indústria	NLOC / esforço	# Métodos / esforço	# Classes / esforço
Média anterior ao uso do XP	10,2	0,36	0,05
Média com uso do XP	17	1,45	0,21
% de Variação	66,3%	302,1%	282,6%

Tabela 9. Ganhos de produtividade com XP em projetos Web (FONTE: ADAPTADO DE MAURER; MANTEL, 2002.)

Poole e Huisman (2001) relataram o uso bem-sucedido do *Extreme Programming* (XP) na companhia Iona Technologies, no desenvolvimento de um *middleware*. Os autores mencionaram que a empresa não possuía um processo de desenvolvimento e manutenção de software definido, sendo que a falta de qualidade no processo acabava por se refletir na qualidade do produto. Um processo de reengenharia de software foi iniciado em 1997 sem, entretanto, alcançar os resultados desejados. Em 2000, a companhia decidiu implementar, de forma gradativa, as práticas do XP em seu processo de manutenção de software. A Iona Technologies alcançou excelentes resultados, reportando ganhos de produtividade na ordem de 67%, o que possibilitou uma redução no seu quadro de pessoal de 36

para 25 profissionais. A empresa obteve também ganhos de qualidade, com uma queda bastante significativa do número de erros encontrados pelos clientes, que passaram de 33 erros ao mês para quatro erros ao mês. Estes resultados foram atribuídos, principalmente, à adoção da prática de programação em pares.

Em 2001, pesquisa conduzida pelo Cutter Consortium (COCKBURN; HIGHSMITH, 2001b), com mais de 200 profissionais de diferentes empresas dos Estados Unidos, Europa, Índia e Austrália, sobre o uso dos métodos clássicos e ágeis de desenvolvimento de software chegou a três conclusões interessantes:

- Um aumento do número de empresas que utilizavam algum Método Ágil de Desenvolvimento de Software, quando comparado à pesquisa similar realizada em 2000;
- Os projetos de desenvolvimento realizados segundo o enfoque ágil apresentaram melhor desempenho em termos de atendimento aos requisitos do negócio, satisfação do cliente e qualidade, que os projetos conduzidos nos moldes clássicos;
- Os Métodos Ágeis receberam melhor pontuação no quesito “moral dos profissionais”, resultado este considerado, na época, surpreendente pelo fato de apenas 12% dos respondentes serem analistas ou programadores.

Similarmente a Reifer (2002), Cockburn e Highsmith (2001b) não generalizam estas conclusões, mas consideram, em especial o terceiro item, um indício bastante positivo da contribuição dos Métodos Ágeis em relação à motivação das equipes de projeto.

A aplicação bem-sucedida do método *Extreme Programming* (XP) no desenvolvimento de um sistema de missão crítica⁷, voltado à coordenação de equipes da Hewlett-Packard distribuídas ao redor do mundo, foi relatada por Gawlas (2004, p. 18-24). O autor aponta que o processo de desenvolvimento, pelo método XP, foi dividido em sete etapas, cada qual com um resultado e um prazo inicial pré-determinados. Várias adaptações foram feitas ao longo do projeto. Além de destacar práticas relacionadas ao XP, Gawlas (2004, p. 18-24) chama a atenção para a abordagem de gerenciamento de projetos utilizada, caracterizada como uma combinação entre o conhecimento e a experiência da equipe de projeto e a necessidade de balanceamento entre o enfoque ágil e o Gerenciamento Clássico de Projetos adotado pela companhia. Após nove meses e dentro do prazo previsto, o software entrou em produção com poucos defeitos, que puderam ser corrigidos de forma rápida e tranquila. Gawlas (2004, p. 18-24) salienta também que um ponto de destaque foi a motivação da equipe de projeto.

Bonato (2004, p. 107-172), por sua vez, apresenta um caso de adoção do *Extreme Programming* (XP) em renomada empresa jornalística brasileira, para o desenvolvimento de uma aplicação que visava à reformulação do pagamento de bonificação às

⁷ Sistema (ou software) de missão crítica é aquele em que a vida, a saúde ou a segurança das pessoas ou organizações podem ser comprometidas se a qualidade do software não for extremamente alta (TURK et al, 2005, p.29).

empresas publicitárias, garantindo maior controle ao processo. O projeto foi iniciado segundo o enfoque clássico de desenvolvimento de software, utilizado pela companhia. Contudo, logo no início do projeto, a equipe se deparou com problemas como a impossibilidade de detalhamento prévio de requisitos e com a dificuldade de entregar o software no prazo, dada a documentação bastante extensiva exigida pelo processo. Neste momento, houve uma decisão pela adoção do XP, buscando aliar qualidade e produtividade diante de requisitos instáveis. Ao final, o projeto foi considerado um sucesso, com ganhos de qualidade (mensurados por uma redução de 62,9% no número de erros reportados pelos usuários após a implementação, quando comparados com outros projetos conduzidos pelo Jornal) e com ganhos de produtividade estimados em 4%, quando comparados à média da indústria (BONATO, 2004, p. 131-135). Assim como nos outros relatos, Bonato (*Ibid.*) destaca o elevado grau de motivação da equipe de projeto.

Apesar dos vários estudos expostos, não foi encontrada na literatura uma pesquisa extensa o suficiente, que permita a generalização dos benefícios resultantes do emprego dos Métodos Ágeis. No entanto, não se pode negar que os estudos realizados até o momento, alguns dos quais aqui retratados, apontam que determinadas organizações estão de fato auferindo resultados positivos, quanto à qualidade, à produtividade e à motivação de seus profissionais, com a adoção dos Métodos Ágeis para o desenvolvimento de software.

Limitações dos Métodos Ágeis

Neste trabalho, tão importante quanto listar a aplicação e os benefícios obtidos com o uso dos Métodos Ágeis, é apresentar as limitações e obstáculos à sua utilização. Dentre os autores que escreveram sobre os Métodos Ágeis de Desenvolvimento de Software, alguns apresentaram e discutiram os principais conceitos e valores que regem este novo enfoque, como Abrahamsson *et al* (2003), Bohem e Turner (2003), Glass (2001), Cohen *et al* (2003), Cockburn e Highsmith (2001a; 2001b), Udo e Koppensteiner (2003), Fowler (2003); outros, como Turk *et al* (2003; 2005), Nerur *et al* (2005), Boger *et al* (2001) e McBreen (2003), introduziram uma visão mais crítica, apontando limitações ao emprego dos Métodos Ágeis.

Dentre as obras com enfoque crítico acima citadas, a de Turk *et al* (2005) se destaca das demais por sua ampla abordagem, voltada à identificação tanto dos pressupostos básicos sobre os quais os princípios e práticas dos Métodos Ágeis estão ancorados, quanto das limitações, decorrentes de sua aceitação, conforme mostra a **Figura 3**. Com base nestes pressupostos, identificados após análise detalhada de publicações referentes aos diferentes Métodos Ágeis e dos valores e princípios defendidos pela *Agile Alliance* e retratados no documento *Manifesto para o Desenvolvimento Ágil de Software* (BECK *et al*, 2001), Turk *et al* (2003, p. 43-46) pontuam as limitações dos Métodos Ágeis.

Sendo assim, para que se possa entender tais limitações, deve-se primeiramente visualizar os pressupostos identificados pelos autores (TURK *et al*, 2005, p. 22), que se encontram retratados na **Tabela 10**.

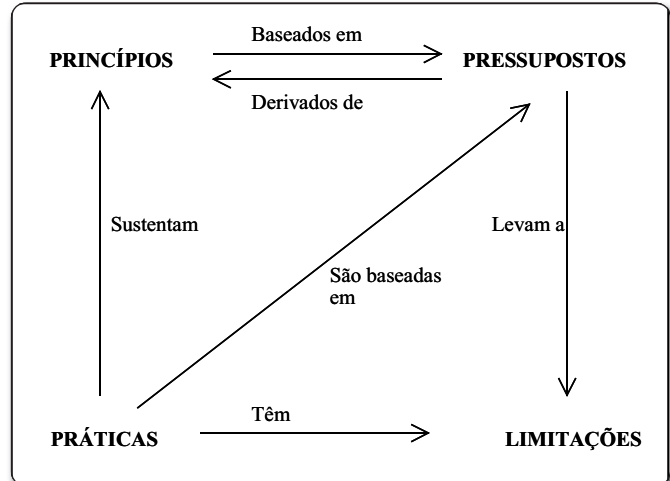


Figura 3. Relacionamento entre princípios, práticas, pressupostos e limitações (FONTE: TURK *et al*, 2005, p. 8.)

Ao abordar as limitações dos Métodos Ágeis, Turk *et al* (2005, p. 23), iniciam a discussão afirmando que “[...] nenhum processo ágil é uma bala de prata (apesar dos altos brados dos entusiastas)”. Os autores defendem que, uma vez que os pressupostos apresentados acima não são atendidos por todas as organizações ou por todos os ambientes de desenvolvimento, os Métodos Ágeis em seus formatos atuais apresentam sim limitações. Para minimizá-las, a depender do contexto, determinados métodos necessitam de extensões ou incorporações de práticas e princípios, usualmente associados os enfoques preditivos e clássicos de desenvolvimento de software. As principais limitações apontadas por Turk *et al* (2005, p. 23-34) são apresentadas nos parágrafos a seguir.

A primeira limitação identificada diz respeito ao suporte a equipes distribuídas. A dispersão geográfica acarreta vários problemas que inexistem quando os integrantes da equipe de desenvolvimento e do cliente encontram-se próximos. A comunicação se torna mais difícil, há necessidade de ferramentas e tecnologia especiais. Em ambientes distribuídos, os pressupostos de interação com o cliente, comunicação da equipe, comunicação face a face ficam comprometidos. O uso de documentação formal pode se fazer necessário, para organizar o trabalho realizado em várias localidades por diferentes equipes (TURK *et al*, 2005, p. 25-26).

Turk *et al* (2005, p. 26-27) apontam a questão da subcontratação como outra limitação dos Métodos Ágeis, sendo que Nerur *et al* (2005, p. 76) estendem a discussão para a negociação de contratos de forma geral. A transferência do desenvolvimento de software a um subcontratado pressupõe o estabelecimento de um contrato de prestação de serviços bem definido. No entanto, as bases de um contrato tradicional não são as mesmas utilizadas pelos Métodos Ágeis. Mesmo seguindo um enfoque iterativo e incremental, os contratos de forma geral prevêem marcos, um número fixo de iterações, com durações e entregáveis pré-definidos e cláusulas restritivas a mudanças, o que não se adequa a vários pressupostos dos Métodos Ágeis. Turk *et al* (2005, p. 26-27) apontam como saída, a criação de

Pressupostos	Detalhamento
Pressuposto da visibilidade	A visibilidade do projeto só pode ser alcançada através da entrega de um software funcionando
Pressuposto da iteração	Um projeto pode ser sempre estruturado em iterações curtas e de períodos fixos
Pressuposto da interação com o cliente	A equipe do cliente está disponível para interação frequente, quando solicitado pelos programadores
Pressuposto da comunicação da equipe	Os programadores estão localizados em local que permita a comunicação frequente e intensiva entre si
Pressuposto da comunicação face a face	A comunicação face a face é o método mais produtivo de comunicação com o cliente e entre os programadores
Pressuposto da documentação	Desenvolver uma documentação extensiva e consistente (relativamente completa) e modelos de software é contraproducente
Pressuposto da mudança dos requerimentos	Requisitos sempre sofrem modificações, devido a mudanças de tecnologia, necessidades dos clientes, domínios de negócio ou mesmo aquisição de novos clientes
Pressuposto do custo da mudança	O custo da mudança não aumenta dramaticamente com o passar do tempo
Pressuposto da experiência da equipe	Os programadores têm a experiência necessária para definir e adaptar seus processos apropriadamente
Pressuposto da auto-avaliação	As equipes almejam a auto-avaliação
Pressuposto da auto-organização	As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizadas
Pressuposto da garantia de qualidade	A avaliação dos artefatos de software (produtos e processos) pode se restringir a entrevistas frequentes e informais, revisões e testes de codificação
Pressuposto do desenvolvimento da aplicação específica	O reuso e a generalidade não devem ser objetivos do desenvolvimento de uma aplicação específica.
Pressuposto do redesenho contínuo	O software pode ser continuamente redesenhado e assim mesmo, manter sua estrutura e integridade conceitual.

Tabela 10. Sumário dos pressupostos relativos aos princípios propostos pela Agile Alliance (FONTE: TURK et al, 2005, p. 22.)

um contrato contendo uma parte fixa e outra variável para acomodar ambas as expectativas.

A dificuldade de apoio a grandes projetos e grandes equipes é outra limitação apontada por alguns autores. Vários autores apontam que quanto maior a equipe, maior a complexidade da comunicação e menos ágil se torna o processo (HIGHSMITH, 2004; COHEN *et al*, 2003, TURK *et al*, 2005, p. 27-28; NERUR *et al*, 2005, p.76). O tamanho das equipes restringe a eficiência e a frequência das interações face a face, havendo necessidade de uma maior estruturação e organização da equipe e da definição de várias linhas e formas de comunicação. O volume de documentação requerido é maior e, de forma geral, a agilidade tende a diminuir. Isto não quer dizer que grandes projetos não possam utilizar Métodos Ágeis, mas há que se adotar práticas complementares para garantir o bom andamento dos trabalhos.

Discutindo outro item, a geração de componentes total ou parcialmente reutilizáveis (códigos de programas, documentos de análise e desenho, entre outros) pressupõe a visão completa da solução (e não apenas do software a ser desenvolvido naquele momento) e a ênfase no controle de qualidade. Os Métodos Ágeis têm por premissa a manutenção de uma documentação mínima, o que dificulta a determinação do potencial de reuso de um determinado artefato, além de ter por foco o desenvolvimento de soluções para problemas específicos e não o desenvolvimento de códigos mais genéricos e reutilizáveis. Desta forma, os Métodos Ágeis de Desenvolvimento de Software não são os mais adequados à geração de artefatos reutilizáveis (TURK *et al*, 2005, p. 28-29).

A adoção de Métodos Ágeis no desenvolvimento de sistemas de missão crítica é outro ponto comentado por Turk *et al* (2005, p. 29-30). Aplicações desta natureza requerem que a qualidade

de todos os seus componentes seja intensivamente testada e que estes sejam projetados de forma a não haver falhas que impossibilitem a correção ou o uso seguro de um determinado equipamento. Neste contexto, os pressupostos relacionados à documentação, à garantia da qualidade e ao aprimoramento contínuo dos Métodos Ágeis não são mais válidos. Uma especificação formal, testes intensivos e abrangentes e outras técnicas de análise e avaliação dos métodos clássicos de desenvolvimento de software se tornam necessárias, apesar de os autores ressaltarem que a adoção de algumas práticas dos Métodos Ágeis seja interessante, para aumentar a qualidade e a confiabilidade do produto final (TURK *et al*, *Ibid.*).

O desenvolvimento de um software grande e complexo, com numerosas linhas de códigos (milhares ou milhões) e alto grau de integração entre seus vários componentes é outra situação em que a aplicabilidade dos Métodos Ágeis é contestada. Segundo Turk *et al* (2005, p. 30-31), projetos deste porte requerem elevado esforço de gerenciamento e controle, processos estruturados e formais, para garantir o perfeito entendimento do software e a integração de todas as suas partes. Os testes também devem ser cuidadosamente planejados. De forma geral, não é possível o desenvolvimento deste tipo de software de forma incremental e a adoção da premissa de desenvolvimento contínuo pode ser comprometida, haja vista a complexidade e a extensão do código.

Complementando a análise, Nerur *et al* (2005, p.76) e TURK *et al* (2005, p. 13-15) chamam a atenção à questão da dependência entre as organizações e suas equipes ágeis de desenvolvimento, em especial no que diz respeito à gestão do conhecimento, vital nos dias de hoje, e à relação de poder. Os Métodos Ágeis de Desenvolvimento de Software encorajam o pensamento direto e o corte em todo e qualquer esforço desnecessário, inclusive

na documentação. Sendo assim, no desenvolvimento ágil, o conhecimento é tácito e reside apenas na mente de cada integrante de equipe. Em alguns casos, as organizações tornam-se dependentes desta equipe e muitas vezes há uma mudança no balanço do poder, entre os gerentes e os programadores, sendo os últimos os detentores das informações. Para minimizar este impasse, Nerur *et al* (*op. cit.*) e TURK *et al* (*op. cit.*) sugerem que seja claramente definido o que deve ser documentado e o que pode ser mantido como conhecimento tácito.

O pressuposto de interação com os clientes é outro ponto amplamente discutido e criticado por vários autores. Turk *et al* (2005, p. 12) esclarece que os Métodos Ágeis pressupõem que os clientes estão sempre disponíveis para participar, esclarecer dúvidas e tomar decisões junto à equipe de desenvolvimento, o que na prática nem sempre se mostra viável. Nerur *et al* (2005, p. 76) complementam que o ambiente pluralista de tomada de decisões (que envolve clientes e equipe de desenvolvimento), estimulado pelos Métodos Ágeis, torna o processo decisório mais lento e difícil, quando comparado ao enfoque clássico, em que o gerente do projeto é o responsável por tal. Nerur *et al* (*Ibid.*) destacam que o sucesso dos Métodos Ágeis depende de se encontrar clientes que almejem e tenham disponibilidade para uma participação ativa no processo de desenvolvimento, que sejam colaborativos, representativos e comprometidos e que disponham da autonomia e do conhecimento adequados ao projeto.

Há que se mencionar ainda, a questão da manutenção de software, qualificada como algo tão problemática quanto o desenvolvimento propriamente dito e ainda pouco abordada pelos Métodos Ágeis (RUS *et al*, 2002; COHEN *et al*, 2003).

Todas estas limitações apontadas por autores e estudiosos do assunto (TURK *et al*, 2003 e 2005; NERUR *et al*, 2005; BOGER *et al*, 2001; McBRENN, 2003) devem ser avaliadas quando da definição método de desenvolvimento de software a ser utilizado. Contudo, apesar de sua importância, não podem ser consideradas como única verdade, uma vez que a literatura aponta exemplos de projetos conduzidos com o uso de Métodos Ágeis que foram bem-sucedidos, apesar das condições teoricamente adversas à sua utilização. Entre estes exemplos podem ser citados: o desenvolvimento de um software de missão crítica com o uso do XP (GAWLAS, 2003) e a utilização do XP e do *Scrum* em projetos com mais de 100 pessoas distribuídas em mais de um continente (Fowler, 2003).

Fatores Críticos de Sucesso de Projetos de Desenvolvimento de Software Realizados com o Uso dos Métodos Ágeis

Poucas são as referências na literatura que discutem a questão dos fatores críticos de sucesso de projetos conduzidos com o uso de Métodos Ágeis. Cohen *et al* (2003, p. 31) destacam que são três os principais fatores críticos de sucesso para um projeto conduzido segundo um enfoque ágil, a saber: cultura, pessoas e comunicação.

Cockburn e Highsmith (2001b), autores que defendem uma visão dos Métodos Ágeis centrada nas pessoas, identificam a

competência individual como o fator primordial para o sucesso de projetos conduzidos de acordo com os Métodos Ágeis. Mencionam que “[...] se as pessoas forem boas o suficiente, podem usar praticamente qualquer processo e atingir seus objetivos. Se as pessoas não forem boas o bastante, não há processo que possa reparar esta inadequação” (COCKBURN; HIGHSMITH, 2001b, p. 131). A falta de apoio executivo e dos principais usuários, por outro lado, é apontada como um grande empecilho para o alcance do sucesso de um projeto, levando até mesmo excelentes profissionais ao não cumprimento de seus objetivos.

Em 2003, um estudo de caráter exploratório desenvolvido por Lazarevic (classificado pelo próprio autor como “sem similar” na área até aquele momento) identificou os principais fatores críticos de sucesso de projetos de desenvolvimento de software conduzidos com o uso de Métodos Ágeis (LAZAREVIC, 2003). A pesquisa foi realizada por meio da aplicação de um questionário a profissionais que haviam gerenciado ou participado de projetos com tais características, integrantes de cinco grupos da internet especializados na troca de experiência e na discussão dos principais Métodos Ágeis. Apesar de obter uma taxa de resposta de apenas 4% (o autor acredita que este número deva ser maior, dado que muitos dos integrantes dos referidos grupos encontravam-se inativos), a pesquisa permitiu a identificação de fatores críticos de sucesso para tais projetos que se mostraram bastante alinhados às colocações de autores como Cockburn e Highsmith (2001b), Cohen *et al* (2003) e Reifer (2002).

A existência de programadores motivados foi identificada como a chave para o sucesso dos projetos pesquisados, sendo considerado o fator crítico mais importante para projetos conduzidos com o uso de Métodos Ágeis. O segundo fator crítico de sucesso identificado foi o grau de propriedade coletiva do código. A propriedade coletiva pressupõe que qualquer integrante da equipe pode alterar o código desenvolvido por outrem e contribuir para a solução e encoraja a sinergia, a colaboração e o trabalho em equipe. A descoberta deste segundo fator está totalmente alinhada ao primeiro item. Lazarevic (2003, p. 28) destaca que “[...] atingir um estágio em que os resultados do grupo sejam melhores que a contribuição de cada indivíduo é o coração de muitos Métodos Ágeis”. Ainda, de acordo com Sutherland (2001), quando os indivíduos se ajudam mutuamente e contribuem para o todo, a equipe de desenvolvimento atinge o estado de hiper-produtividade. O terceiro fator encontrado diz respeito à entrega de um protótipo logo no início do projeto. Para Lazarevic (*Ibid.*), a maioria dos respondentes interpretou esta questão à luz das necessidades de mudanças de requisitos, ou seja, da necessidade de um desenvolvimento incremental do software.

Nesse mesmo estudo, Lazarevic (2003, p. 24) menciona que os projetos desenvolvidos segundo os métodos *Extreme Programming* (XP) e *Dynamic Software Development Model* (DSDM) foram mais bem-sucedidos, mas este resultado não se mostrou estatisticamente significativo.

Como informação adicional desta pesquisa, tem-se que a maioria dos respondentes possuía de um a quatro anos de experiência em Métodos Ágeis, o que confirma que a utilização destes métodos no desenvolvimento de software é um fenômeno relativamente novo. O tamanho das equipes dos projetos analisados variava entre quatro e vinte integrantes, sendo que apenas 10% dos projetos tiveram mais de 50 participantes (LAZAREVIC, 2003, p. 26).

Como muitos outros autores (REIFER, 2002; COCKBURN; HIGHSMITH, 2001b), Lazarevic (2003, p. 29) não considera seu estudo representativo, não devendo assim ser generalizado. Mas o enfoque diferenciado de seu trabalho, em especial, sua abrangência (obtenção de dados de diferentes projetos, de profissionais provenientes de distintas organizações), certamente contribui para a ampliação do conhecimento na área.

Ampliando a visão dos fatores críticos de sucesso, Chin (2004), assim como Cohn e Ford (2003), Highsmith (2004) e Nerur *et al* (2005), ressaltam que não se pode esquecer o inegável valor do gerenciamento de projetos na entrega de um projeto de desenvolvimento de software bem-sucedido. Apesar de alguns Métodos Ágeis já possuírem componentes de gerenciamento de projetos inseridos em suas práticas, como por exemplo o

Scrum, o FDD e o ASD, Thomsett (2002, p. 17) indica a necessidade de uma abordagem mais ampla e, preferencialmente, em separado das questões técnicas.

Analisando os fatores críticos de sucesso identificados, percebe-se que estão estritamente relacionados à valorização da competência individual e à comunicação (REIFER, 2002; COCKBURN; HIGHSMITH, 2001b; LAZAREVIC, 2003) e à entrega rápida de valor (LAZAREVIC, 2003). Todos estes itens correspondem a características marcantes dos Métodos Ágeis de Desenvolvimento de Software e o também do Gerenciamento Ágil de Projetos, o que pode sugerir que talvez este seja o enfoque de gerenciamento de projetos mais apropriado para o desenvolvimento de software conduzido com o uso de um Método Ágil. ●

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine

tem que ser feita ao seu gosto.

Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback



Referências

ABRAHAMSSON, P et al. New directions on agile methods: a comparative analysis. In: Proceedings of the 25th International Conference on Software Engineering. [S.l.]. IEEE Software Society, 2003, p. 244-254.

AGILE ALLIANCE. Manifesto for agile software development. Disponível em <<http://www.agilemanifesto.org/>>. Acesso em janeiro, 2005.

AMBLER, S. W. Agile modeling: effective practices for extreme programming and the unified process. John Wiley & Sons, Inc., 2002a.

AMBLER, S.W. Lessons in agility from internet-based development. IEEE Software, [S.l.], v. 19, n. 2, 2002b, p. 66-73.

AMBLER, S.W. When does (n't) agile modeling make sense. Apr, 2002c. Disponível em <<http://www.agilemodeling.com/essays/whendoesAmWork.htm>>. Acesso em julho de 2005.

AMBLER, S.W. Quality in an agile world. Software Quality Professional, [S.L.], v. 7, n. 4, sep. 2005.

BECK, K. Embrace Change with Extreme Programming. IEEE Computer Magazine, [S.l.], Oct 1999, p. 70-77.

_____. Extreme programming explained. Boston: Addison-Wesley, 2000.

BECK, Kent. et al. Chrysler goes to "extremes". Oct, 1998. Disponível em <<http://www.xprogramming.com/publications/dc9810cs.pdf>>. Acesso em julho, 2005.

BECK, Kent et al. Manifesto for agile software development. Feb. 2001. Disponível em <<http://www.agilemanifesto.org/>>. Acesso em janeiro, 2005.

BECK, Kent; FOWLER, M. Extreme programming applied. Boston: Addison-Wesley, 2001.

BECK, Kent; MEE, R. Enhancing brazilian software's global competitiveness. Jan, 2003. Disponível em: <<http://www.xispe.com.br/wiki/wiki.jsp?topic=EnhancingBrazilsSoftwareGlobalCompetitiveness>>. Acesso em novembro, 2004.

BOGER, M. et al. Extreme modeling. In: SUCCI, G.; Marchesi, M. (eds). Extreme programming examined. Boston: Addison-Wesley, 2001.

BOHEM, Barry. Get ready for agile methods, with care. IEEE Computer Magazine, [S.l.], Jan. 2002, p. 64-69.

BOHEM, Barry.; TURNER, Richard. Balancing discipline and agility: evaluating and integrating plan-driven methods. In: Proceedings of the 26th Conference on Agile Development. IEEE COMPUTER SOCIETY, [S.l.], May 2004, p. 718-719.

BONATO, A. S. F. Uma experiência de aplicação do processo Extreme Programming em pequenos projetos. São Paulo, 2004. Dissertação (Mestrado em Engenharia) - Programa de Pós-Graduação em Engenharia, Escola Politécnica da Universidade de São Paulo.

BRASIL. Ministério da Ciência e Tecnologia. Secretaria de Política de Informática. Levantamento do universo de Associadas Softex. Brasília - DF, 2001a.

_____. Ministério da Ciência e Tecnologia. Secretaria de Política de Informática. Qualidade e Produtividade no Setor de Software Brasileiro. Brasília - DF, 2001b.

CHIN, Gary. Agile project management: how to succeed in the face of changing project requirements. NY: Amacon, 2004.

COCKBURN, A. Crystal clear: a human-powered methodology for small teams. Boston: Addison-Wesley, 2004.

_____. Agile software development. Boston: Addison-Wesley, 2001.

_____. Learning from agile software development - part one. Crosstalk, The Journal of Defense Software Engineering, [S.l.], October 2002.

COCKBURN, A.; HIGHSMITH, J. Agile software development: the business of innovation. IEEE Computer Magazine, [S.l.], p. 131-133, sep 2001a.

_____. Agile software development: the people factor. IEEE Computer Magazine, [S.l.], p. 131-133, nov 2001b.

COHEN, David et al. Agile software development: a DACS state of art report. NY: Data Analysis Center for Software - Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland, 2003. Disponível em <<http://www.thedacs.com/techs/agile/>>. Acesso em abril, 2005.

Continuação: Referências

- COHEN, Dennis. J.; GRAHAM, Robert. J. Gestão de projetos: MBA executivo. Trad. Afonso Celso da Cunha Serra. Rio de Janeiro: Campos, 2002.
- COHN, Martin. The scrum development process. Disponível em <www.mountaingoatsoftware.com/scrum> Acesso em julho, 2005.
- COHN, Martin; FORD, Doris. Introducing an agile process to an organization. IEEE Computer Magazine, June 2003, [S.l.], p. 74-78.
- DEMING, W. E. Qualidade: a revolução da administração. Rio de Janeiro: Marques Saraiva, 1990.
- FERREIRA, A. A.; REIS, A. C. F.; PEREIRA, M. I. Gestão empresarial: de Taylor aos nossos dias. São Paulo: Thomson, 2002, p. 146-164.
- FOWLER, Martin. The new methodology. April, 2003. Disponível em <<http://www.martinfowler.com/articles/newMethodology.html>>. Acesso em julho, 2005.
- GAWLAS, J. Mission critical development with XP & agile process: common code ownership and lots of testing. Dr. Dobbs' Journal, [S.l.], p. 21-24, Jan. 2004.
- GLASS, Robert. Extreme programming: the good, the bad and the bottom line. IEEE Software, [S.l.], Nov. 2001, p. 111-112.
- GLAZER, H. Dispelling the process myth: having a process does not mean sacrificing agility or creativity. Crosstalk, The Journal of Defense Software Engineering, [S.l.], Nov. 2001.
- HIGHSMITH, Jim. Adaptive management: patterns for the e-business era. Cutter IT Journal, [S.l.], v. XII, n. 9, sep. 1999.
- HIGHSMITH, Jim. Agile software development ecosystems. Boston: Addison-Wesley, 2002.
- _____. Agile project management: creating innovative products. Boston: Addison-Wesley, 2004.
- HIGHSMITH, Jim et al. Extreme programming: e-business application delivery. Feb. 2002. Disponível em <<http://www.cutter.com/freestuff/ead0002.pdf>>. Acesso em julho, 2004.
- JURAN, J. M.; GRZYNA, F. M. Controle da qualidade – handbook: conceitos, políticas e filosofia da qualidade. São Paulo: Makron, McGraw-Hill, 1991
- JURIC, R. Extreme Programming and its Development Practices. In: Proceedings of 22nd international conference on information technology interfaces. 2002, p. 97-104.
- LAZAREVIC, George. An exploratory study of the new product development utilized by software companies using agile product development approach. Oct. 2003. Disponível em <<http://www.agilealliance.org/articles/articles/agileOct.pdf>>. Acesso em agosto, 2005.
- MAURER, Frank.; MARTEL, Sebastien. Extreme programming: rapid development for web-based applications. IEEE Internet Computing, [S.l.], v. 6, n. 1, p. 86-90, Jan. 2002.
- _____. On the productivity of agile software practices: an industrial case study. Abr. 2004. Disponível em: <<http://seml.ucalgary.ca/milos/papers/2002/MaurerMartel2002c.pdf>>. Acesso em Agosto, 2005.
- McBREEN, P. Questioning extreme programming. Boston: Addison-Wesley, 2003.
- NERUR, S. et al. Challenges of migrating to agile methodologies: organizations must carefully assess their readiness before treading the path of agility. Communication of the ACM, [S.l.], v. 48, n. 5, p. 73-78, May 2005.
- PALMER, S. R.; FELSING, M. A practical guide to feature-driven development. [S.l.]: Pearson Education, 2001.
- PAULK, Mark. C. Extremeprogramming from a SW-CMM perspective. IEEE Software, [S.l.], v. 18, n. 6, p. 19-26, Nov. 2001.
- _____. Agile methodologies and process discipline. Crosstalk - The Journal of Defense Software Engineering, [S.l.], v. 15, n. 10, p. 15-28, Oct. 2002.
- POPPENDIECK, M. Lean programming. Software Development Magazine. May-June, 2001 Disponível em <<http://www.agilealliance.org/articles/articles/LeanProgramming.htm>>. Acesso em julho, 2005.
- POPPENDIECK, M.; POPPENDIECK, T. Lean software development. Boston: Addison-Wesley, 2003.
- REIFER, Donald J. How good are agile methods? IEEE Software, [S.l.], p. 14-17, jul-ago, 2002.
- RUS, I. et al. Process diversity in software maintenance guest editors' introduction. Software Maintenance Research and Practice, Dec. 2002.
- SCHWABER, K.; BEEDLE, M. Agile software development with scrum. NJ: Prentice Hall, 2001.
- SCHWABER, K. Controlled chaos: living on the edge. 2002. Disponível em <<http://www.agilealliance.org/articles/articles/ap.pdf>>. Acesso em junho, 2005.
- THOMSETT, R. Radical Project Management. NJ: Prentice Hall, 2002.
- TURK, D. et al. Limitations of agile software processes. Jan, 2003. Disponível em <<http://www.agilealliance.org/articles/articles/LimitationsofAgile.pdf>> Acesso em Agosto, 2005.
- _____. Assumptions underlying agile software development process. Colorado State University, 2005.
- UDO, N.; KOPPENSTEINER, S. Will agile change the way we manage software projects? Agile from a PMBoK guide perspective. Projectway, [S.l.], 2003.