Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software

Article · June 2004

DOI: 10.21529/RESI.2004.0301006 · Source: DOAJ

1 author:

Michel S. Soares
Universidade Federal de Sergipe
92 PUBLICATIONS **388** CITATIONS

Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software

Michel dos Santos Soares[1]

[1]Universidade Presidente Antônio Carlos, BR 482 Km 3, Gigante, Conselheiro Lafaiete, MG, Brasil
michelssoares@yahoo.com.br

Resumo - Este artigo apresenta algumas vantagens das metodologias ágeis para desenvolver software em relação às metodologias tradicionais. Em particular são apresentadas as principais características e as práticas das metodologias ágeis *Extreme Programming* e Scrum. Também são feitas comparações com as metodologias tradicionais, procurando enfatizar que as metodologias ágeis são baseadas em pessoas e não em processos e planejamentos. Finalmente são apresentadas as principais vantagens e desvantagens da *Extreme Programming* e da Scrum. Também são apresentados alguns resultados empíricos do uso de metodologias ágeis.

Palavras-chave: Engenharia de Software, Metodologias Ágeis, Extreme Programming, Scrum

Abstract – This paper shows some advantages of agile methodologies for software development comparing with traditional ones. In particular Extreme Programming and Scrum are presented with its characteristics and practices. Also are done comparisons with traditional methodologies, emphasizing that agile methodologies are based on people instead of process and planning. Finally the principal advantages and disadvantages of the use of Extreme Programming and Scrum are presented. Also some empirical results of the use of agile methodologies are presented.

Key-words: Software Engineering, Agile Methodologies, Extreme Programming, Scrum

Introdução

As metodologias ágeis para desenvolvimento de software são uma resposta às chamadas metodologias pesadas ou tradicionais. Mesmo com a evolução dos computadores, das técnicas e ferramentas nos últimos anos, a produção de software confiável, correto e entregue dentro dos prazos e custos estipulados ainda é muito difícil. Dados de 1995 [1], usando como base 8380 projetos, mostram que apenas 16,2% dos projetos foram entregues respeitando os prazos e os custos e com todas as funcionalidades especificadas. Aproximadamente 31% dos projetos foram cancelados antes de estarem completos e 52,7% foram entregues, porém com prazos maiores, custos maiores ou com menos funcionalidades do que especificado no início do projeto. Dentre os projetos que não foram finalizados de acordo com os prazos e custos especificados, a média de atrasos foi de 222%, e a média de custo foi de 189% a mais do que o previsto. Considerando todos os projetos que foram entregues além do prazo e com custo maior, na média, apenas 61% das funcionalidades originais foram incluídas. Mesmo os projetos cuja entrega é feita respeitando os limites de prazo e custo possuem qualidade suspeita, uma vez que provavelmente foram feitos com muita pressão sobre os desenvolvedores, o que pode quadruplicar o número de erros de software, segundo a mesma pesquisa. As principais razões destas falhas estavam relacionadas com o processo em Cascata. A recomendação final foi que o desenvolvimento de software deveria ser baseado em modelos incrementais, o que poderia evitar muitas das falhas reportadas.

Processos orientados a documentação para o desenvolvimento de software, como o modelo em Cascata, são de certa forma fatores limitadores aos desenvolvedores. Além disso, muitas organizações não possuem recursos ou inclinação para processos pesados de produção de software. Por esta razão, muitas organizações, particularmente as pequenas, acabam por não usar nenhum processo, o que pode levar a efeitos desastrosos em termos de qualidade de software. Torna-se necessário, então, utilizar metodologias ágeis, que não são orientadas à documentação nem tampouco se preocupam apenas com a codificação.

A maioria das metodologias ágeis nada possuem de novo [2]. O que as diferencia das metodologias tradicionais são o enfoque e os valores. A idéia das metodologias ágeis é o enfoque nas pessoas e não em processos ou algoritmos. Além disso, existe a preocupação de gastar menos tempo com documentação e mais com a implementação. Uma característica das

metodologias ágeis é que elas são adaptativas ao invés de serem preditivas. Com isso, elas se adaptam a novos fatores decorrentes do desenvolvimento do projeto, ao invés de procurar analisar previamente tudo o que pode acontecer no decorrer do desenvolvimento.

Apesar do uso crescente das metodologias ágeis, ainda falta uma base maior de projetos para verificar suas vantagens. Mesmo assim, os resultados iniciais em termos de qualidade, confiança, datas de entrega e custo são promissores.

Para ser realmente considerada ágil a metodologia deve aceitar a mudança ao invés de tentar prever o futuro. O problema não é a mudança em si, mesmo porque ela ocorrerá de qualquer forma. O problema é como receber, avaliar e responder às mudanças.

Enquanto as metodologias ágeis variam em termos de práticas e ênfases, elas compartilham algumas características, como desenvolvimento iterativo e incremental, comunicação e redução de produtos intermediários, como documentação extensiva. Desta forma existem maiores possibilidades de atender aos requisitos do cliente, que muitas vezes são mutáveis.

Dentre as várias metodologias ágeis existentes, as mais conhecidas são a *Extreme Programming* [3] e a Scrum [4].

Metodologia

Um exemplo de uma metodologia tradicional ou pesada é o modelo em Cascata [5], que é composto basicamente por atividades seqüenciais de levantamento de requisitos, análise, projeto, implementação, teste, implantação e manutenção. Este modelo é derivado de outras engenharias tradicionais (Civil, Elétrica, Naval,...) e foi o primeiro a ser usado pela Engenharia de Software, na década de 70.

O termo Engenharia de Software surgiu em uma conferência no final da década de 60 [6]. A proposta inicial era a sistematização do desenvolvimento de software, que deveria ser tratado com engenharia e não como arte. Desta forma, a idéia foi propor a utilização de métodos, ferramentas e técnicas para a produção de software confiável, correto e entregue respeitando os prazos e custos definidos. Apesar de toda evolução desde que o termo foi criado, o número de fracassos em projetos de software ainda é alto.

O modelo em Cascata dominou a forma de desenvolvimento de software até o início da década de 90, apesar das advertências dos pesquisadores da área e dos desenvolvedores, que identificaram os problemas gerados ao se adotar esta visão seqüencial de tarefas. Por exemplo, Fred Brooks em seu famoso artigo "*No Silver Bullet: Essence and Accidents of Software Engineering*", descreve que a idéia de especificar totalmente um software antes do início de sua implementação é impossível [7]. Outro pesquisador, Tom Gilb, desencoraja o uso do modelo em Cascata para grandes softwares, estimulando o desenvolvimento incremental como um modelo que apresenta menores riscos e maiores possibilidades de sucesso [8].

As metodologias pesadas devem ser aplicadas apenas em situações em que os requisitos do software são estáveis e requisitos futuros são previsíveis. Estas situações são difíceis de serem atingidas, uma vez que os requisitos para o desenvolvimento de um software são mutáveis. Dentre os fatores responsáveis por alterações nos requisitos estão a dinâmica das organizações, as alterações nas leis e as mudanças pedidas pelos *stakeholders*, que geralmente têm dificuldades em definir o escopo do futuro software.

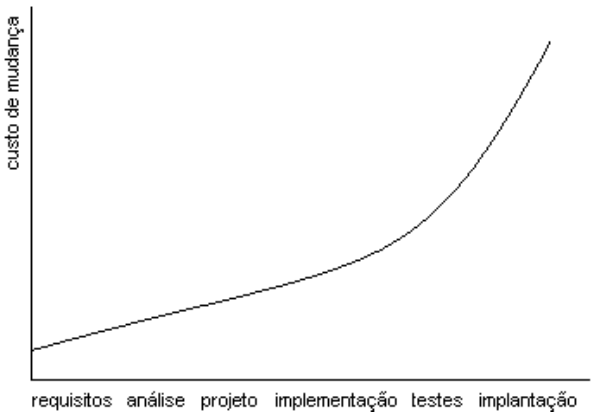

Figura 1 - custo de mudança no projeto no modelo em cascata

O gráfico da Figura 1 apresenta o custo de alterar requisitos de software em várias fases do desenvolvimento ao se usar o modelo em Cascata. Pelo gráfico fica claro que o custo de alterações no modelo em Cascata cresce de forma exponencial de acordo com as fases do projeto. Desta forma, estima-se que caso alguma alteração tenha como custo "1x" quando feita na fase de requisitos, ela terá um custo de "60x a 100x" quando feita na fase de implantação [5]. Portanto, alterações nos requisitos no modelo em Cascata não são desejáveis.

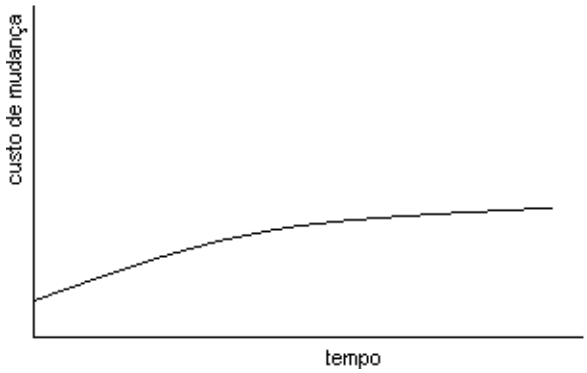

Figura 2 – custo de mudanças no projeto com metodologias ágeis

Como alterações em requisitos são comuns, a curva ideal deveria ser a da Figura 2, em que o custo não cresce muito mesmo quando alterações são feitas em fases avançadas do desenvolvimento. O gráfico da Figura 2 apresenta o custo de mudanças nas metodologias ágeis. Na verdade, as metodologias ágeis incentivam a mudança nos requisitos, pois desta forma é possível realmente entregar ao cliente o produto que ele precisa.

O termo "metodologias ágeis" tornou-se popular em 2001 quando dezessete especialistas em processos de desenvolvimento de software representando os métodos *Extreme Programming* (XP), Scrum, DSDM, Crystal e outros, estabeleceram princípios comuns compartilhados por todos esses métodos. O resultado foi a criação da Aliança Ágil e o estabelecimento do "Manifesto Ágil" (*Agile Manifesto*) [9]. Os conceitos chave do Manifesto Ágil são:

"**Indivíduos e interações** ao invés de processos e ferramentas.

Software executável ao invés de documentação.

Colaboração do cliente ao invés de negociação de contratos.

Respostas rápidas a mudanças ao invés de seguir planos."

O "Manifesto Ágil" não rejeita os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária quando comparado com os indivíduos e interações, com o software estar executável, com a colaboração do cliente e as respostas rápidas a mudanças e alterações. Esses conceitos aproximam-se melhor com a forma que pequenas companhias de Tecnologia da Informação trabalham e respondem a mudanças. Entre as metodologias ágeis a mais conhecida é a *Extreme Programming*.

Extreme Programming

A Extreme Programming (XP) é uma metodologia ágil para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente [3]. Dentre as principais diferenças da XP em relação às outras metodologias estão:

- *Feedback* constante
- Abordagem incremental
- A comunicação entre as pessoas é encorajada.

O primeiro projeto a usar XP foi o C3, da Chrysler. Após anos de fracasso utilizando metodologias tradicionais, com o uso da XP o projeto ficou pronto em pouco mais de um ano [10].

A maioria das regras da XP causa polêmica à primeira vista e muitas não fazem sentido se aplicadas isoladamente. É a sinergia de seu conjunto que sustenta o sucesso de XP, encabeçando uma verdadeira revolução no desenvolvimento de software.

A XP enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As regras, práticas e valores da XP proporcionam um agradável ambiente de desenvolvimento de software para os seus seguidores, que são conduzidos por quatro valores: comunicação, simplicidade, *feedback* e coragem [3].

A finalidade do princípio de comunicação é manter o melhor relacionamento possível entre clientes e desenvolvedores, preferindo conversas pessoais a outros meios de comunicação. A comunicação entre os desenvolvedores e o gerente do projeto também é encorajada.

A simplicidade visa permitir a criação de código simples que não deve possuir funções desnecessárias. Por código simples entende-se implementar o software com o menor número possível de classes e métodos. Outra idéia importante da simplicidade é procurar implementar apenas requisitos atuais, evitando-se adicionar funcionalidades que podem ser importantes no futuro. A aposta da XP é que é melhor fazer algo simples hoje e pagar um pouco mais amanhã para fazer modificações necessárias do que implementar algo complicado hoje que talvez não venha a ser usado, sempre considerando que requisitos são mutáveis.

A prática do *feedback* constante significa que o programador terá informações constantes do código e do cliente. A informação do código é dada pelos testes constantes, que indicam os erros tanto individuais quanto do software integrado. Em relação ao cliente, o *feedback* constante significa que ele terá freqüentemente

uma parte do software totalmente funcional para avaliar. O cliente então constantemente sugere novas características e informações aos desenvolvedores. Eventuais erros e não conformidades são rapidamente identificados e corrigidos nas próximas versões. Desta forma, a tendência é que o produto final esteja de acordo com as expectativas reais do cliente.

É necessário coragem para implantar os três valores anteriores. Por exemplo, não são todas as pessoas que possuem facilidade de comunicação e têm bom relacionamento. A coragem também dá suporte à simplicidade, pois assim que a oportunidade de simplificar o software é percebida, a equipe pode experimentar. Além disso, é preciso coragem para obter *feedback* constante do cliente.

A XP baseia-se nas 12 práticas [3] a seguir:

- Planejamento: consiste em decidir o que é necessário ser feito e o que pode ser adiado no projeto. A XP baseia-se em requisitos atuais para desenvolvimento de software, não em requisitos futuros. Além disso, a XP procura evitar os problemas de relacionamento entre a área de negócios (clientes) e a área de desenvolvimento. As duas áreas devem cooperar para o sucesso do projeto, e cada uma deve focar em partes específicas do projeto. Desta forma, enquanto a área de negócios deve decidir sobre o escopo, a composição das versões e as datas de entrega, os desenvolvedores devem decidir sobre as estimativas de prazo, o processo de desenvolvimento e o cronograma detalhado para que o software seja entregue nas datas especificadas.
- Entregas freqüentes: visa à construção de um software simples, e conforme os requisitos surgem, há a atualização do software. Cada versão entregue deve ter o menor tamanho possível, contendo os requisitos de maior valor para o negócio. Idealmente devem ser entregues versões a cada mês, ou no máximo a cada dois meses, aumentando a possibilidade de *feedback* rápido do cliente. Isto evita surpresas caso o software seja entregue após muito tempo, melhora as avaliações e o *feedback* do cliente, aumentando a probabilidade do software final estar de acordo com os requisitos do cliente.
- Metáfora: são as descrições de um software sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento do software.
- Projeto simples: o programa desenvolvido pelo método XP deve ser o mais simples possível e satisfazer os requisitos atuais, sem a preocupação de requisitos futuros. Eventuais requisitos futuros devem ser adicionados assim que eles realmente existirem. Esta forma de raciocínio se opõe ao “implemente para hoje e projete para amanhã”.
- Testes: a XP focaliza a validação do projeto durante todo o processo de desenvolvimento. Os programadores desenvolvem o software criando primeiramente os testes.
- Refatoração: focaliza o aperfeiçoamento do projeto do software e está presente em todo o desenvolvimento. A refatoração deve ser feita apenas quando é necessário, ou seja, quando um desenvolvedor da dupla, ou os dois, percebe que é possível simplificar o módulo atual sem perder nenhuma funcionalidade.
- Programação em pares: a implementação do código é feita em dupla, ou seja, dois desenvolvedores trabalham em um único computador. O desenvolvedor que está com o controle do teclado e do *mouse* implementa o código, enquanto o outro observa continuamente o trabalho que está sendo feito, procurando identificar erros sintáticos e semânticos e pensando estrategicamente em como melhorar o código que está sendo implementado. Esses papéis podem e devem ser alterados continuamente. Uma grande vantagem da programação em dupla é a possibilidade dos desenvolvedores estarem continuamente aprendendo um com o outro.
- Propriedade coletiva: o código do projeto pertence a todos os membros da equipe. Isto significa que qualquer pessoa que percebe que pode adicionar valor a um código, mesmo que ele próprio não o tenha desenvolvido, pode fazê-lo, desde que faça a bateria de testes necessária. Isto é possível porque na XP todos são responsáveis pelo software inteiro. Uma grande vantagem desta prática é que, caso um membro da equipe deixe o projeto antes do fim, a equipe consegue continuar o projeto com poucas dificuldades, pois todos conhecem todas as partes do software, mesmo que não seja de forma detalhada.
- Integração contínua: interagir e construir o sistema de software várias vezes por dia, mantendo os programadores em sintonia, além de possibilitar processos rápidos. Integrar apenas um conjunto de modificações de cada vez é uma prática que funciona bem porque fica óbvio quem

deve fazer as correções quando os testes falham: a última equipe que integrou código novo ao software. Esta prática é facilitada com o uso de apenas uma máquina de integração, que deve ter livre acesso a todos os membros da equipe.

- 40 horas de trabalho semanal: a XP assume que não se deve fazer horas extras constantemente. Caso seja necessário trabalhar mais de 40 horas pela segunda semana consecutiva, existe um problema sério no projeto que deve ser resolvido não com aumento de horas trabalhadas, mas com melhor planejamento, por exemplo. Esta prática procura ratificar o foco nas pessoas e não em processos e planejamentos. Caso seja necessário, os planos devem ser alterados, ao invés de sobrecarregar as pessoas.
- Cliente presente: é fundamental a participação do cliente durante todo o desenvolvimento do projeto. O cliente deve estar sempre disponível para sanar todas as dúvidas de requisitos, evitando atrasos e até mesmo construções erradas. Uma idéia interessante é manter o cliente como parte integrante da equipe de desenvolvimento.
- Código padrão: padronização na arquitetura do código, para que este possa ser compartilhado entre todos os programadores.

Scrum

Outra metodologia ágil que apresenta uma comunidade grande de usuários é a Scrum [4]. Seu objetivo é fornecer um processo conveniente para projeto e desenvolvimento orientado a objeto. A Scrum apresenta uma abordagem empírica que aplica algumas idéias da teoria de controle de processos industriais para o desenvolvimento de softwares, reintroduzindo as idéias de flexibilidade, adaptabilidade e produtividade. O foco da metodologia é encontrar uma forma de trabalho dos membros da equipe para produzir o software de forma flexível e em um ambiente em constante mudança.

A idéia principal da Scrum é que o desenvolvimento de softwares envolve muitas variáveis técnicas e do ambiente, como requisitos, recursos e tecnologia, que podem mudar durante o processo. Isto torna o processo de desenvolvimento imprevisível e complexo, requerendo flexibilidade para acompanhar as mudanças. O resultado do processo deve ser um software que é realmente útil para o cliente [11].

A metodologia é baseada em princípios semelhantes aos da XP: equipes pequenas, requisitos pouco estáveis ou desconhecidos e iterações curtas para promover visibilidade para o desenvolvimento. No entanto, as dimensões em Scrum diferem de XP.

A Scrum divide o desenvolvimento em iterações (*sprints*) de trinta dias. Equipes pequenas, de até dez pessoas, são formadas por projetistas, programadores, engenheiros e gerentes de qualidade. Estas equipes trabalham em cima de funcionalidades (os requisitos, em outras palavras) definidas no início de cada *sprint*. A equipe é responsável pelo desenvolvimento desta funcionalidade.

Na Scrum existem reuniões de acompanhamento diárias. Nessas reuniões, que são preferencialmente de curta duração (aproximadamente quinze minutos), são discutidos pontos como o que foi feito desde a última reunião e o que precisa ser feito até a próxima. As dificuldades encontradas e os fatores de impedimento (*bottlenecks*) são identificados e resolvidos.

O ciclo de vida da Scrum é baseado em três fases principais, divididas em sub-fases:

1. **Pré-planejamento** (*Pre-game phase*): os requisitos são descritos em um documento chamado *backlog*. Posteriormente eles são priorizados e são feitas estimativas de esforço para o desenvolvimento de cada requisito. O planejamento inclui também, entre outras atividades, a definição da equipe de desenvolvimento, as ferramentas a serem usadas, os possíveis riscos do projeto e as necessidades de treinamento. Finalmente é proposta uma arquitetura de desenvolvimento. Eventuais alterações nos requisitos descritos no *backlog* são identificadas, assim como seus possíveis riscos.
2. **Desenvolvimento** (*game phase*): as muitas variáveis técnicas e do ambiente identificadas previamente são observadas e controladas durante o desenvolvimento. Ao invés de considerar essas variáveis apenas no início do projeto, como no caso das metodologias tradicionais, na Scrum o controle é feito continuamente, o que aumenta a flexibilidade para acompanhar as mudanças. Nesta fase o software é desenvolvido em ciclos (*sprints*) em que novas funcionalidades são adicionadas. Cada um desses ciclos é desenvolvido de forma tradicional, ou seja, primeiramente faz-se a análise, em seguida o projeto, implementação e testes. Cada um desses ciclos é planejado para durar de uma semana a um mês.
3. **Pós-planejamento** (*post-game phase*): após a fase de desenvolvimento são

feitas reuniões para analisar o progresso do projeto e demonstrar o software atual para os clientes. Nesta fase são feitas as etapas de integração, testes finais e documentação.

Ferramentas de apoio

Por serem metodologias relativamente novas, existem poucas ferramentas disponíveis que suportem o processo ágil de desenvolvimento. Dentre as existentes, a maioria suporta apenas a *Extreme Programming* e ainda estão em fase de pesquisa e desenvolvimento. A mais conhecida é a XPlanner [12], que é uma ferramenta de código livre usada para auxiliar o planejamento da *Extreme Programming*. O desenvolvimento desta ferramenta ainda está em progresso, mas já existe uma versão estável para o gerenciamento de estórias do usuário, gerenciamento de tarefas, verificação de progresso do projeto e gerenciamento das métricas individuais e da equipe. Atualmente existe suporte para vários idiomas, dentre eles o francês, o espanhol, o chinês, o italiano, o alemão e o português. Dentre as funcionalidades futuras da ferramenta estão a integração com outras metodologias ágeis, em especial a Scrum.

Resultados

As metodologias ágeis ainda estão em sua infância, mas já apresentam resultados efetivos. Por exemplo, um artigo [13] comparando métodos ágeis com as metodologias tradicionais pesadas mostrou que os projetos usando os métodos ágeis obtiveram melhores resultados em termos de cumprimento de prazos, de custos e padrões de qualidade. Além disso, o mesmo estudo mostra que o tamanho dos projetos e das equipes que utilizam as metodologias ágeis têm crescido. Apesar de serem propostas idealmente para serem utilizadas por equipes pequenas e médias (até 12 desenvolvedores), aproximadamente 15% dos projetos que usam metodologias ágeis estão sendo desenvolvidos por equipes de 21 a 50 pessoas, e 10% dos projetos são desenvolvidos por equipes com mais de 50 pessoas, considerando um universo de 200 empresas usado no estudo.

Em relação à qualidade do software, de acordo com [14] e [15] o uso correto da XP pode levar a organização a atingir os níveis CMM [16] 2 e 3, apesar do CMM ser orientado a metodologias tradicionais e ter como foco “o quê” fazer, enquanto a XP enfoca o “como fazer”. Juntos estes métodos formam um *framework* para estruturar o desenvolvimento de software de uma organização. Na Boeing a XP foi usada antes de serem implementadas as idéias de CMM. Verificou-se que não foram necessárias muitas alterações nos processos para que a Boeing fosse certificada com o nível 5 da CMM [17].

A XP é ideal para ser usada em projetos em que os *stakeholders* não sabem exatamente o que desejam e podem mudar muito de opinião durante o desenvolvimento do projeto. Com *feedback* constante, é possível adaptar rapidamente eventuais mudanças nos requisitos.

Um outro ponto positivo são as entregas constantes de partes operacionais do software. Desta forma, o cliente não precisa esperar muito para ver o software funcionando, como nas metodologias tradicionais.

A integração e o teste contínuos também possibilitam a melhora na qualidade do software. Não é mais necessário existir uma fase de integração de módulos, uma vez que eles são continuamente integrados e eventuais problemas são resolvidos constantemente.

A XP apresenta algumas desvantagens e problemas identificados, e tem recebido algumas críticas. Muitos acreditam que a XP é a volta ao processo caótico de desenvolvimento de software, conhecido também como codifica-remenda [5]. Este modelo caótico existe principalmente em pequenas e médias organizações que não podem suportar os altos custos de desenvolvimento ao se usar metodologias tradicionais.

A XP possui a tendência de eliminar várias boas práticas que existiram durante vários anos do desenvolvimento de software. Por exemplo, a análise do problema por meio de diagramas. Obviamente não se deve projetar vários diagramas, muitos dos quais nunca serão consultados. Mas é importante projetar alguns modelos que ajudarão no entendimento do problema.

A análise de requisitos parece ser muito informal, e em alguns casos dificilmente funcionaria assim. A informalidade na captura de requisitos pode não ser bem vista pelos clientes, que podem se sentir inseguros. Outra possibilidade de insegurança é a refatoração de código, que pode ser vista pelos clientes como amadorismo e incompetência.

Na XP não existe a preocupação formal em fazer a análise e o planejamento de riscos. Como riscos acontecem normalmente em projetos de desenvolvimento de software, este é um ponto negativo da XP. Deve-se, portanto, procurar implementar uma estratégia de gestão de riscos sem tornar a metodologia muito complexa.

De forma geral, as 12 práticas da XP apóiam-se mutuamente. Portanto, deve-se procurar aplicá-las totalmente, ou a maior parte possível. Por exemplo, não usar a programação em dupla pode atrapalhar a prática do código ser de propriedade coletiva. Apesar disso, elas podem ser de certa forma adaptadas à realidade das organizações. A implantação de todas as

práticas pode ser confusa, levando ao abandono prematuro da XP. Segundo Beck [3], as práticas da XP podem ser implantadas uma a uma para que sejam evitadas confusões, desentendimentos e pressões, pois a equipe sobre pressão tem a tendência de voltar a aplicar as práticas anteriores.

Em relação à metodologia Scrum, existem vários casos de sucesso relatados [18].

Segundo Schwaber e Beedle [4], a Scrum deve ser usada em equipes de até 10 pessoas. Em projetos que precisam de equipes maiores, deve-se dividir as pessoas em equipes de até 10 pessoas. Isto é necessário para melhorar a comunicação entre as pessoas da equipe.

Discussão e Conclusões

As Metodologias Ágeis têm sido bem aceitas pela indústria de software e por pesquisadores da Engenharia de Software. Apesar de não haver ainda uma grande base de comparações os primeiros resultados têm sido satisfatórios.

De certa forma, apesar da XP ser uma metodologia nova e ser considerada por muitos como uma revolução, ela não apresenta muitos pontos revolucionários. Na verdade, a XP agrupa uma série de práticas que têm sido usadas desde o início da computação eletrônica, como a programação em duplas e a propriedade coletiva do código.

É possível fazer uma analogia do futuro das metodologias ágeis com as metodologias para desenvolvimento de software orientadas a objeto. Da mesma forma que várias metodologias de desenvolvimento orientadas a objeto surgiram e competiram entre si, surgiram também várias metodologias ágeis. No caso da orientação a objetos houve um esforço de padronização que resultou na notação UML [19], que atualmente é um padrão na indústria de software mundial. Da mesma forma, futuramente é possível que as várias metodologias ágeis unam-se, principalmente em torno da XP, por ser a mais aceita. Existe, por exemplo, um movimento para o uso conjunto da XP com a Scrum. A XP seria usada para a fase de desenvolvimento e a Scrum para o planejamento e gerenciamento do projeto. A integração das duas metodologias seria relativamente simples, uma vez que elas compartilham algumas características, como a necessidade da presença do cliente, pequenas liberações e o encorajamento em fazer as mudanças necessárias para atender requisitos reais dos *stakeholders*. Existe inclusive a proposta da metodologia híbrida XP@Scrum [20]. Apesar de não existirem estudos empíricos suficientes, alguns autores recomendam o uso em conjunto da Scrum e da XP para grandes projetos [20].

Atualmente o autor está coordenando um projeto de pesquisa em que a XP é usada para o desenvolvimento de softwares baseados em Web. Como a Web é um ambiente de desenvolvimento dinâmico e com mudanças constantes, as metodologias tradicionais orientadas a documentação são menos adequadas que as metodologias ágeis. Uma idéia a ser implantada futuramente neste projeto de pesquisa é a integração da XP com outras metodologias ágeis, como a Scrum. Neste caso os aspectos de gerenciamento de projetos da Scrum serão integrados com as práticas da XP.

O desafio futuro das Metodologias Ágeis é encontrar meios de eliminar alguns de seus pontos fracos, como a falta de análise de riscos, sem torná-las metodologias pesadas. Outro desafio é como usar essas metodologias em grandes empresas e equipes, uma vez que normalmente essas metodologias são baseadas em equipes pequenas. Neste caso, pelo menos é necessário resolver os problemas de comunicação internos na equipe, uma vez que é comum em grandes empresas os funcionários estarem separados geograficamente. Apesar do interesse nas metodologias ágeis, ainda faltam casos de sucesso de seu uso em projetos grandes e críticos.

Referências

1. Standish Group, “CHAOS report”, 586 Olde Kings Highway, Dennis, MA 02638, USA, (1995)
2. Cockburn, A. e Highsmith, J. "Agile Software Development: The Business of Innovation", IEEE Computer, Sept., (2001), pp. 120-122
3. Beck, K., “Programação Extrema Explicada” , Bookman, (1999)
4. Schwaber, K. e Beedle, M. "Agile Software Development with SCRUM", Prentice-Hall, (2002)
5. Pressman, R. “Engenharia de Software” McGraw-Hill, (2001)
6. Naur, P. e Randell, B. “Software Engineering: Report on a Conference sponsored by the NATO science committee”. 7-11 October, Garmisch, Germany, (1968)
7. Brooks, F., “No Silver Bullet: Essence and Accidents of Software Engineering” Proc. IFIP, IEEE CS Press, 1987, pp. 1069-1076; reprinted in IEEE Computer, Apr, (1987), pp. 10-19
8. Gilb, T., “Principles of Software Engineering Management”, Addison-Wesley, (1988)

9. Agile Manifesto, http://agilemanifesto.org/, acessado em 25 de Setembro de 2004
10. Highsmith, J. Orr, K. Cockburn, A. "Extreme Programming", E-Business Application Delivery, Feb., (2000), pp. 4-17
11. Schwaber, K. "Scrum Development Process", OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag. (1995)
12. XPlanner,http://www.xplanner.org/, acessado em 24 de Setembro de 2004
13. Charette, R. ”Fair Fight? Agile Versus Heavy Methodologies” Cutter Consortium E-project Management Advisory Service, 2, 13, (2001)
14. Paulk, M. C. “Extreme Programming from a CMM perspective”, IEEE Software, vol. 18, n. 6, (2001), pp 19-26
15. Glazer, H. “Dispelling the process myth: having a process does not mean sacrificing agility or creativity”, Crosstalk (2001)
16. “The Capability Maturity Model: Guidelines for Improving the Software Process”, Addison-Wesley, (1995)
17. Abrhamsson, P., Salo, O., Ronkainen, J., “Agile software development methods - review and analysis”, VTT (2002)
18. Rising, L. e Janof, N.S. "The Scrum software development process for small teams". IEEE Software 17(4): (2002), pp. 26-32
19. Booch, G., Jacobson, I., Rumbaugh, J. UML - Guia do Usuário. Campus, São Paulo, 476p, (2000)
20. XP@Scrum, http://controlchaos.com/about/xp.php acessado em 24 de Setembro de 2004