Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software

MICHEL DOS SANTOS SOARES[1]

Unipac - Universidade Presidente Antônio Carlos
Faculdade de Tecnologia e Ciências de Conselheiro Lafaiete
BR 482 Km 3 - Gigante CEP 36.400-000 - Conselheiro Lafaiete
[1]michelssoares@yahoo.com.br

Resumo. Este artigo faz uma comparação entre as metodologias tradicionais para desenvolvimento de software e as metodologias ágeis. Em particular é feita a comparação da *Extreme Programming* (XP), uma metodologia ágil muito usada, e o modelo Clássico ou Sequencial. As práticas da XP são apresentadas, enfatizando que suas características são ideais para projetos que devem ter um desenvolvimento rápido e que podem ter requisitos alterados constantemente pelos clientes.

Palavras-Chave: Metodologias de Desenvolvimento, Extreme Programming, Modelo Clássico

1 Introdução

Metodologias ágeis têm sido apontadas como uma alternativa às abordagens tradicionais para o desenvolvimento de software. As metodologias tradicionais, conhecidas também como pesadas ou orientadas a planejamentos, devem ser aplicadas apenas em situações em que os requisitos do sistema são estáveis e requisitos futuros são previsíveis. Entretanto, em projetos em que há muitas mudanças, em que os requisitos são passíveis de alterações, onde refazer partes do código não é uma atividade que apresenta alto custo, as equipes são pequenas, as datas de entrega do software são curtas e o desenvolvimento rápido é fundamental, não pode haver requisitos estáticos, necessitando então de metodologias ágeis. Além disso o ambiente das organizações é dinâmico, não permitindo então que os requisitos sejam estáticos.

Processos orientados a documentação para o desenvolvimento de software são, de certa forma, fatores limitadores aos desenvolvedores e muitas organizações não possuem recursos ou inclinação para processos pesados de produção de software. Por esta razão, as organizações pequenas acabam por não usar nenhum processo. Isto pode levar a efeitos desastrosos na qualidade do produto final, além de dificultar a entrega do software nos prazos e custos predefinidos. Em particular, o modelo Clássico ou Sequencial será apresentado como exemplo de metodologia tradicional.

Dentre todas as metodologias ágeis existentes, uma que vem se destacando em número de adeptos e projetos é a *Extreme Programming* (XP). As metodologias ágeis surgiram com a proposta de aumentar o enfoque nas pessoas e não nos processos de desenvolvimento. Além disso, existe a preocupação de gastar menos tempo com documentação e mais com resolução de problemas de forma iterativa.

Este artigo apresenta as vantagens e desvantagens do uso de metodologias ágeis em relação às tradicionais. Em particular são feitas comparações entre o modelo Clássico e a *Extreme Programming*.

2 Processos de Software

Um processo de software (ou metodologia de desenvolvimento de software) é um conjunto de atividades e resultados associados que auxiliam na produção de software. Dentre as várias atividades associadas, existem por exemplo a análise de requisitos e a codificação. O resultado do processo é um produto que reflete a forma como o processo foi conduzido.

Embora existam vários processos para o desenvolvimento de software, existem atividades fundamentais comuns a todos eles [Sommerville (2003)]:

Especificação de Software: definição das funcionalidades (requisitos) e das restrições do software. Geralmente é uma fase em que o desenvolvedor conversa com o cliente para definir as características do novo software.

Projeto e Implementação de Software: o software é produzido de acordo com as especificações. Nesta fase são propostos modelos através de diagramas,

e estes modelos são implementados em alguma linguagem de programação.

Validação de Software: o software é validado para garantir que todas as funcionalidades especificadas foram implementadas.

Evolução de Software: o software precisa evoluir para continuar sendo útil ao cliente.

Muitas organizações desenvolvem software sem usar nenhum processo. Geralmente isso ocorre porque os processos tradicionais não são adequados às realidades das organizações. Em particular, as organizações pequenas e médias não possuem recursos suficientes para adotar o uso de processos pesados. Por esta razão, muitas organizações não utilizam nenhum processo. O resultado desta falta de sistematização na produção de software é a baixa qualidade do produto final, além de dificultar a entrega do software nos prazos e custos predefinidos e inviabilizar a futura evolução do software.

Existem vários processos de software definidos na literatura da Engenharia de Software. É comum mesmo algumas organizações criarem seu próprio processo ou adaptar algum processo à sua realidade. Dentre os vários processos existentes, existem as metodologias tradicionais, que são orientadas a documentação, e as metodologias ágeis, que procuram desenvolver software com o mínimo de documentação.

3 Metodologias Tradicionais

As metodologias tradicionais são também chamadas de pesadas ou orientadas a documentação. Essas metodologias surgiram em um contexto de desenvolvimento de software muito diferente do atual, baseado apenas em um *mainframe* e terminais burros [Royce (1970)]. Na época, o custo de fazer alterações e correções era muito alto, uma vez que o acesso aos computadores eram limitados e não existiam modernas ferramentas de apoio ao desenvolvimento do software, como depuradores e analisadores de código. Por isso o software era todo planejado e documentado antes de ser implementado. A principal metodologia tradicional e muito utilizada até hoje é o modelo Clássico.

3.1 Modelo Clássico

O modelo Clássico ou Sequencial [Pressman (2001)] foi o primeiro processo publicado de desenvolvimento de software. Desde sua introdução tem sido muito utilizado. É um modelo em que existe uma sequência a ser seguida de uma etapa a outra. Cada etapa tem associada ao seu término uma documentação padrão que deve ser aprovada para que se inicie a etapa imediatamente posterior. De uma forma geral fazem parte do modelo Clássico as etapas de definição de requisitos, projeto do software, implementação e teste unitário, integração e teste do sistema, operação e manutenção. O problema do modelo em Cascata é sua inflexível divisão do projeto em fases distintas, o que dificulta possíveis alterações que são comuns no desenvolvimento de um projeto. É um modelo que deve ser usado somente quando os requisitos forem bem compreendidos. A figura 1 ilustra graficamente o modelo Clássico.

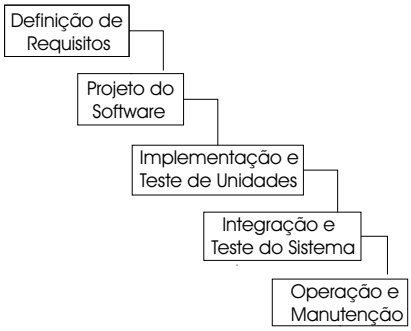

Figura 1: Modelo Clássico

O modelo Clássico 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 [Brooks (1987)]. Outro pesquisador, Tom Gilb, desencoraja o uso do modelo Clássico em grandes softwares, estimulando o desenvolvimento incremental como um modelo que apresenta menores riscos e maiores possibilidades de sucesso [Gilb (1999)].

Dados de 1995 [Standish Group, (1995)], utilizando 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 modelo Clássico. A recomendação final foi que o desenvolvimento de software deveria ser baseado em modelos incrementais, o que poderia evitar muitas das falhas reportadas.

4 Metodologias Ágeis

O termo "Metodologias Ágeis" tornou-se popular em 2001 quando dezessete especialistas em processos de desenvolvimento de software representando os métodos Scrum [Schwaber e Beedle (2002)], *Extreme Programming* (XP) [Beck (1999)] e outros, estabeleceram princípios comuns compartilhados por todos esses métodos. Foi então criada a Aliança Ágil e o estabelecimento do "Manifesto Ágil" [Agile Manifesto (2004)]. 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 e médias organizações trabalham e respondem a mudanças. Entre as metodologias ágeis a mais conhecida é a *Extreme Programming*.

4.1 *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 [Beck (1999)]. 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 [Highsmith et al., (2000)].

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 [Beck (1999)]. 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 forma de comunicação é um fator chave na XP: procura-se o máximo possível comunicar-se pessoalmente, evitando-se o uso de telefone e o envio de mensagens por correio eletrônico. 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 tendên-

cia 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 [Beck (1999)] 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 e melhora as avaliações 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.

Testes: a XP focaliza a validação do projeto durante todo o processo de desenvolvimento. Os programadores desenvolvem o software criando primeiramente os testes.

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.

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.

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: é a prática de 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 sé-

rio 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.

5 Comparativo entre as metodologias

A maioria das metodologias ágeis nada possuem de novo [Cockburn et al., (2001)]. 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. 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.

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. Essa análise prévia é difícil e apresenta alto custo, além de tornar-se um problema caso não se queira fazer alterações nos planejamentos. Por exemplo, para seguir estritamente o planejamento, pode ser necessário que a equipe trabalhe sobre pressão e faça muitas horas extras, o que prejudica a qualidade do software.

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. Como exemplo, as aplicações baseadas em Web são melhor modeladas usando metodologias ágeis, uma vez que o ambiente Web é muito dinâmico.

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. 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 [Pressman (2001)], ao se usar o modelo Clássico. Portanto, alterações nos requisitos no modelo Clássico não são desejáveis.

Por serem relativamente novas, existem poucas ferramentas disponíveis que suportam 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 XPlanner é uma ferramenta de código livre muito conhecida que suporta a XP, principalmente auxiliando a fase de planejamento [XPlanner, (2004)]. 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. Dentre as funcionalidades futuras da ferramenta estão a integração com outras metodologias ágeis, em especial a Scrum.

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. Estas alterações nos requisitos são muitas vezes críticas nas metodologias tradicionais, que não apresentam meios de se adaptar rapidamente às mudanças.

Um outro ponto positivo das metodologias ágeis 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.

As metodologias ágeis ainda estão em sua infância, mas já apresentam resultados efetivos. Por exemplo,

um artigo [Charette, R., (2001)] 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.

6 Conclusão

O desafio futuro das metodologias ágeis é encontrar maneiras de eliminar alguns de seus pontos fracos, como a falta de análise de riscos, sem torná-las metodologias pesadas. 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. Outro desafio é como usar essas metodologias ágeis 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 crescente no uso das metodologias ágeis, ainda faltam casos de sucesso de seu uso em projetos grandes e críticos. Quanto mais organizações usem as metodologias ágeis, melhores serão os resultados empíricos em termos de vantagens, desvantagens, riscos e procedimentos para sua adoção nas organizações. Mesmo assim, os resultados iniciais em termos de qualidade, confiança, datas de entrega e custo são promissores.

Atualmente o autor está coordenando um projeto de pesquisa com o uso de metodologias ágeis para desenvolvimento de sistemas baseados na plataforma 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 planejamento e gerenciamento de projeto da Scrum serão integrados com as práticas da XP.

Referências

[Agile Manifesto (2004)] Disponível em http://agilemanifesto.org/, acessado em 25 de Setembro de 2004.

[Beck (1999)] Beck, K. Programação Extrema Explicada. Bookman, 1999.

[Brooks (1987)] Brooks, F. *No Silver Bullet: Essence and Accidents of Software Engineering*. Proc. IFIP, IEEE CS Press, pp. 1069-1076; reprinted in IEEE Computer, pp. 10-19, Apr. 1987.

[Charette, R., (2001)] Charette, R. "Fair Fight? Agile Versus Heavy Methodologies", Cutter Consortium E-project Management Advisory Service, 2, 13, 2001.

[Cockburn et al., (2001)] Cockburn, A. e Highsmith, J. "Agile Software Development: The Business of Innovation", IEEE Computer, Sept., pp. 120-122, 2001.

[Gilb (1999)] Gilb, T. *Principles of Software Engineering Management*. Addison-Wesley, 1988.

[Highsmith et al., (2000)] Highsmith, J. Orr, K. Cockburn, A. *Extreme Programming*. E-Business Application Delivery, Feb., pp. 4-17, 2000.

[Pressman (2001)] Pressman, R. Engenharia de Software. McGraw-Hill, 2001.

[Royce (1970)] Royce, W.W. *Managing the development of large software systems: concepts and techniques*. Proc. IEEE Westcon, Los Angeles, CA.

[Schwaber e Beedle (2002)] Schwaber, K. and Beedle, M., Agile Software Development with Scrum, NJ, Prentice-Hall, 2002.

[Sommerville (2003)] Sommerville, I. Engenharia de Software. Editora Addison-Wesley. 592p, 2003.

[Standish Group, (1995)] CHAOS report, 586 Olde Kings Highway. Dennis, MA 02638, USA, 1995.

[XPlanner, (2004)] XPlanner, disponível em http://www.xplanner.org/, acessado em 20 de Outubro de 2004.