Desenvolvimento e manutenção de software ágil: Um novo processo

Fernando Luiz de Carvalho e Silva (IFF) fernando.carvalho@iff.edu.br

Diego da Silva Sales (UENF) diegosalesbr@gmail.com

Rogério Atem de Carvalho (IFF) ratem@iff.edu.br

Resumo: Neste artigo discute-se um processo de desenvolvimento de software ágil, alternativo ao processo tradicional. Apresenta-se inicialmente um resumo sobre o processo atual de desenvolvimento de software, sendo apresentadas suas características e problemas. Após, são demonstradas as características muito peculiares do novo processo, sob a óptica da engenharia de produção, suas vantagens e desvantagens, sendo ainda descritos em linhas gerais as técnicas ágeis Lean Software Development, Scrum e XP, apresentando um resumo de suas vantagens.
Palavras-chave: Processo; Desenvolvimento de Software ; Agilidade.

1. Introdução

O cenário da economia atual vem apresentando características cada vez mais rápidas e dinâmicas. Como conseqüência, os mercados: fornecedor, concorrente e consumidor, expressam a volatilidade e dinâmica. Este fato exige que as empresas se tornem mais rápidas e apresentem uma capacidade de se adaptarem e de se reconfigurarem para acompanhar o mercado em constante alteração. As empresas precisam se adaptar às demandas de consumidores, ao mercado de fornecedores, às condições geradas pela concorrência e aos eventos emanados pelo Estado na figura de seus órgãos reguladores, fiscalizadores, etc. Assim, a reconfiguração interna das empresas, para que se tornem mais responsivas a estes estímulos (KETTUNEN, 2009), ou seja, responsivas às mudanças, é imprescindível. Neste sentido, projetos de desenvolvimento de software vem sendo cada vez mais demandados para auxiliar no processo de gerenciamento das empresas.

Entretanto, segundo o Standish Group (2004), em seu relatório Chaos Report, os projetos de desenvolvimento de software têm apresentando uma taxa de sucesso muito baixa: a taxa de sucesso média tem ficado em torno de 28% entre os anos de 1994 e 2004.

O objetivo deste trabalho é apresentar as vantagens do uso de técnicas ágeis para gerenciar a manutenção e o desenvolvimento de sistemas de informação por equipes pequenas.

Serão discutidos alguns processos ágeis com suas características, vantagens e viabilidade para Pequenas Empresas. No tópico seguinte, será realizada uma revisão de modelo de Manufatura Enxuta e Manufatura Ágil, para introduzir os Processos Ágeis de Desenvolvimento de software. Por último será justificada a manutenção ou desenvolvimento de sistemas, utilizando técnicas ágeis.

2. Sistemas de Informação

A adoção de sistemas de informação normalmente impacta positivamente a empresa, trazendo tecnologia administrativa, mapeando boas práticas de trabalho implementadas em funcionalidades do software.

Sistemas são desenvolvidos para dar suporte à operacionalização, ao controle gerencial e ao planejamento estratégico da empresa, e por esta razão, precisam estar alinhados às características dos processos da empresa. A empresa, por sua vez, precisa também estruturar-se para reagir a estímulos externos. Neste contexto, o estabelecimento de um processo de manutenção de software ágil torna-se importante, tanto para adaptação dos processos essenciais quanto para realinhar-se continuamente com a empresa que se atualiza.

A finalidade de sistemas de informação, conforme discutido neste artigo, é integrar a maioria das funcionalidades e dos processos de negócio de uma organização em uma única base de dados, fazendo com que a informação obtida em um determinado setor esteja imediatamente disponível para toda a organização.

Segundo Kettunen (2009), muitas empresas de desenvolvimento operam atualmente em um ambiente de incerteza e turbulência no mercado. O mercado consumidor assume posturas diferentes com muita velocidade, obrigando as empresas ao replanejamento de produção e operacionalização de vendas para realizar estoques; o mercado fornecedor pode sofrer com crises, abastecimento de novos concorrentes e causar desabastecimento, podendo resultar em surpresas de custo e *lead-time*; o mercado concorrente age diariamente, buscando crescimento e superação, o que obriga o monitoramento e reação não planejada em curto prazo.

A grande maioria das empresas está submetida aos fatores acima descritos e a velocidade da economia, nos dias de hoje, tem tornado estes fatores mais frequentes, suscitando também eficiência e velocidade de resposta das empresas. A solução para estes problemas é a alta capacidade de resposta e adaptação às condições emergentes, tornando a sobrevivência neste ambiente uma vantagem competitiva. Esta também é a meta da manufatura ágil, das empresas ágeis e do desenvolvimento de software ágil.

Para explicarmos o desenvolvimento de software ágil precisaremos discutir inicialmente o modelo atual de desenvolvimento de software.

3. Processo de Desenvolvimento de Software

Um processo de desenvolvimento de software é um conjunto de atividades com o objetivo de analisar e desenvolver um produto de software (SOMMERVILLE, 1995). Sob a ótica da Engenharia de Produção, desenvolver difere-se de produzir, pois produzir é um esforço repetitivo, sobre algo já conhecido e não exigindo grande conhecimento. Por outro lado, desenvolver é um esforço que envolve aprendizado, análise, criatividade e conhecimento avançado. Desenvolver significa, em última análise, descrever e implementar algo novo. Produzir é um esforço que não exige capacidade analítica, mas sim técnica, podendo ser realizada em segundos em processos automatizados. O desenvolvimento envolve profissionais altamente capacitados em técnicas analíticas, dentro de um processo complexo que pode demorar anos.

3.1 Processo de desenvolvimento Tradicional

No processo de desenvolvimento de software em cascata (PRESSMAN, 1997), tratado aqui como tradicional, os requisitos para o software são elicitados no início do processo, para depois serem processados nas fases de análise, projeto, programação e testes. Ao término do desenvolvimento, o software é avaliado e, se for aceito pelo cliente, é implantado e entra em produção.

Neste caso, geralmente no início do projeto faz-se um contrato de desenvolvimento especificando o escopo que deverá ser tratado, incluindo-se o planejamento do tempo e custo

em função do escopo. Quaisquer novidades que surgirem fora do que estiver previsto no contrato original deverão ser alvo de outro contrato à parte ou de termos aditivos.

Após assinado o contrato é realizado um segundo levantamento mais detalhado dos requisitos. Para tal, o analista de sistemas entrevista os usuários e busca compreender todos os requisitos com o maior nível de detalhes possível para criar a documentação que irá nortear o desenvolvimento do software.

O processo de desenvolvimento passa então para a fase de análise dos problemas levantados, em que são documentados exaustivamente os problemas e soluções propostas.

Em seguida, procede-se à fase de projeto onde a arquitetura da solução também é detalhada.

Estes detalhamentos, de análise e de projeto, servirão de base para a fase de programação. Nesta fase, que também é documentada, utilizam-se os documentos das fases anteriores como guias para o desenvolvimento do código.

A fase seguinte é a de testes, que depende de receber a documentação da análise, do projeto e da programação. Após todos os testes executados e os problemas encontrados resolvidos, é marcada a apresentação para o aceite, quando o usuário avalia o software e o tenderá a aceitá-lo pois o investimento já foi realizado.

Durante este processo sequencial cada passo é documentado visando a comunicação entre as fases e os profissionais envolvidos, o rastreamento das decisões de projeto, os testes planejados e executados, as características internas do software, prevendo também que a manutenção do software pode ser realizada por uma outra equipe em momento futuro.

Este processo apresenta desvantagens, sobretudo em projetos de escopo mais amplo e volátil quando comparado com os processos ágeis, descritos a seguir.

3.2 Processo de desenvolvimento Ágil

O desenvolvimento de software ágil tem seu marco principal na publicação do manifesto ágil (AGILEMANIFESTO) em 2001. Os princípios nele estabelecidos geram um impacto grande sobre o processo de desenvolvimento de software, culminando em um novo processo de desenvolvimento.

Os valores expressos no manifesto são:

1. Indivíduos e Interações são mais importantes que processos e ferramentas;

2. Software funcional é mais importante que documentação;

3. Colaboração do usuário é mais importante que contrato; e

4. Responder às mudanças é mais importante que seguir algum plano.

Estes fatores cobrem mudanças não somente técnicas, mas também mudanças políticas no processo de desenvolvimento de software.

O fator “indivíduos e interações” sobre processos e ferramentas recupera a idéia de que o cliente deve ter participação mais ativa no processo de desenvolvimento, em interação direta com os desenvolvedores em todo processo e não somente durante um levantamento preliminar. Segundo as práticas ágeis devem ser alocados usuários técnicos, conhecedores dos problemas a serem desenvolvidos, para auxiliar na definição de testes que servirão para especificar e controlar o processo de desenvolvimento. Deve haver também um responsável administrativo pela definição, priorização e aceite dos vários subprodutos entregues durante o

processo de desenvolvimento.

O software funcional entregue de forma rápida, agregando valor ao cliente, deve ser mais importante do que a documentação. A documentação deverá ser utilizada nos casos em que ela seja explicitamente necessária. Isto indica preocupação com a sobrecarga de documentação gerada no processo de desenvolvimento de software tradicional. Poppendieck (2003) destaca que, sob o ponto de vista da manufatura enxuta, muitas tarefas no processo de desenvolvimento de software tradicional podem ser vistas como desperdícios, daí a justificativa para que a comunicação direta entre o cliente e o desenvolvedor seja valorizada.

O cliente ou usuário deverá ser alocado para trabalho dentro da equipe, com tarefas bem definidas, como especificação dos testes, monitoramento dos resultados, esclarecimento de dúvidas, execução dos produtos intermediários e finais. É da responsabilidade do cliente a contínua redefinição dos requisitos e de sua importância. O objetivo é determinar quais requisitos agregam mais valor ao cliente, gerando o melhor valor agregado, que sejam mais importantes ou críticos. Desta forma, cada entrega de um subproduto justifica rapidamente a contratação. Com o aprendizado e a re-priorização, o cliente tende a deixar os requisitos menos importantes para o final do contrato.

O contrato de desenvolvimento prevê que os requisitos podem e devem mudar dinamicamente, em função do aprendizado do cliente sobre as vantagens e possibilidades de desenvolvimento de software. A equipe de desenvolvimento também aprende cada vez mais sobre o escopo em que está trabalhando, tornando-se cada vez mais hábil e experiente.

Os contratos deixam de ser baseados em escopo fixo e passam a assegurar tempo de disponibilidade de recursos humanos e qualidade de serviços. O tempo de contrato passa a englobar períodos pequenos (2 meses em média), renováveis pelo contratante, ao final do qual um subproduto deve ser entregue (BECK & CLEAL, 1999).

Os métodos ágeis buscam gerar menos software, e que seja útil 100% do tempo para o cliente, eliminando funcionalidades que possam ser utilizadas raramente ou mesmo nunca utilizadas.

Outro objetivo é gerar software pequeno, porém completo e útil. Não se pretende mais entregar um sistema completo ao final do processo de desenvolvimento, mas sim, definir partes pequenas que atendam a um requisito do usuário e possam ser entregues no final de cada iteração. Cada iteração deve durar entre uma e quatro semanas. Ao final de cada iteração, a equipe de desenvolvimento deve ter produzido algo completo e útil ao usuário (o conceito de pronto precisa ser entendido como terminado, testado, aceito pelo usuário, sem mais o que ser feito).

Dentre os processos ágeis de desenvolvimento de software, pode-se destacar o *Lean Software Development (LD)* (POPPENDIECK, 2003), *Scrum* (RISING, 2000) e o *Extreme Programming* (XP) (BECK, 2000).

Cada um destes métodos apresenta vantagens e desvantagens e um conjunto de idéias e princípios próprios.

O LD funciona como um conjunto de princípios que permeia todos os outros métodos ágeis. Esta aderência do LD aos método ágeis é fruto tanto da interação entre os autores, como da assimilação dos conceitos de agilidade presentes no mercado na década de 90 (KETTUNEN, 2009).

Em relação ao processo XP, a disciplina de desenvolvimento baseia-se nos valores:

simplicidade, comunicação, *feedback* e coragem. Sua visão é de manter a equipe coesa, buscando a simplicidade e a adaptação dinâmica do uso de ferramentas necessárias a cada situação encontrada.

O processo *Scrum*, mais utilizado para gerenciamento do processo de desenvolvimento, utiliza as técnicas do XP como a programação em pares, integração contínua, especificação por testes (TDD) como parte operacional do processo.

3.2.1. *Lean Software Development*

Após o sucesso do sistema de produção "*Just in Time*", posteriormente tratado como Enxuto (Lean), Walmack & Jones (1996) publicaram um aprofundamento do estudo sobre este sistema citando os princípios por traz do sistema, o Pensamento Enxuto.

A partir do pensamento enxuto pode-se perceber que existe na verdade um conjunto de princípios que rege tanto o sistema de produção como a programação de produção, mas também possibilita gerenciar cadeia de fornecedores, desenvolvimento, serviços, entre outras possibilidades.

Segundo os autores, a aplicação dos princípios enxutos possibilita a criação de processos eficientes, entre eles, em especial processos de desenvolvimento de produtos.

O caso de desenvolvimento de software é, sob a ótica da engenharia de produção um caso de desenvolvimento de produtos.

Processos similares são comuns na industria onde se desenvolve produtos em uma vasta variedade desde carros até doces. O processo de desenvolvimento de um produto passa por vários estágios dentre eles pesquisa de risco, pesquisa de mercado, levantamento de requisitos, análise de viabilidade, projeto do produto, projeto da fabricação entre outras fases.

Cabe ressaltar que um processo de desenvolvimento de quaisquer destes produtos envolve uma imersão nas características não só do produto mas também do mercado onde este será inserido e depois da forma que ele deverá ser produzido. As características de tempo para execução deste tipo de processo não são simples de serem mensuradas em função da complexidade do processo.

O risco de se apressar um processo de desenvolvimento é o de fazê-lo sem a qualidade necessária e inviabilizar a produção de uma indústria, ou empresa.

De forma similar, parece estranho à um engenheiro de produção, acostumado a trabalhar em processos de desenvolvimento, que o desenvolvimento de software esteja sujeito a mensuração dos tempos operacionais, medidos através de métricas do tipo: quantas horas demora para se desenvolver um requisito de uma empresa.

O risco do desenvolvimento não agregar a qualidade necessária pode ser o da parada da empresa ou indústria.

O software é um produto único, com características de alta tecnologia e que envolve trabalhadores altamente capacitados em uma tarefa de alto grau de abstração exigindo grande capacidade técnica e criativa.

Segundo Poppendieck (2001), o desenvolvimento de software deve ser definido a partir dos princípios estabelecidos no pensamento enxuto. Esta autora estabeleceu um paralelo destes princípios e em sua obra oferece princípios adaptados para o desenvolvimento de software. Estes princípios para o desenvolvimento de software não são um roteiro ou sequenciamento, mas sim dicas que o engenheiro de software deve respeitar para que o

processo adotado possua adequação a teoria administrativa enxuta.

Várias vantagens podem ser apontadas para se adotar a teoria enxuta, entre elas:

1. Utilizar um processo em fluxo contínuo;
2. Mais eficiente;
3. Garante a qualidade do produto;
4. Possibilita reatividade as mudanças do ambiente;
5. Emprega a melhoria contínua.

3.2.2. *SCRUM*

O processo *Scrum* (RISING, 2000) se baseia em processos da engenharia industrial para trazer luz ao ciclo de vida de desenvolvimento de software (SDLC). O nome *Scrum* vem da formação em *Rugby* em que um time age de forma coesa e compacta para forçar uma jogada. Este processo foi criado para gerenciar o processo de desenvolvimento e manutenção de software em ambientes em que os requisitos estejam em constante mudança. A produção de produtos entregáveis é baseada nos seguintes requisitos: tempo, competitividade, qualidade, visão e recursos. Também é levada em consideração a natureza evolucionária do método.

O método *Scrum* é baseado em princípios como: a) produtos deverão se tornar subprodutos gerenciáveis; b) progresso pode ocorrer mesmo com requisitos instáveis; c) tudo tem que ser visível para todos d) a comunicação interfere positivamente na qualidade; e) o time é responsável solidariamente por tudo que é feito no processo, compartilhando sucesso, fracasso e autoria de tudo até o fim; f) clientes e usuários devem participar ao vivo do processo, interferindo e sabendo como funciona e com está o desenvolvimento; g) o relacionamento e o conhecimento devem ser ampliados durante o processo e h) deve ser estimulada a expectativa pelo sucesso do projeto.

Todo o processo descrito deve ser de conhecimento de toda a equipe que conta com um elemento viabilizador e condutor do processo chamado de *Scrum Master* ou SM. Seu papel é de conduzir a equipe para as regras do *Scrum*, viabilizando e facilitando o processo.

O *Scrum* parte da intenção de resolver problemas utilizando soluções em software. Tais problemas são descritos em termos de estórias. Estas estórias serão os requisitos a serem desenvolvidos, e são de responsabilidade do contratante ou administrador, aqui chamado de dono do produto *(Product Owner* ou PO).

O cliente ou usuário precisa estar também representado por técnicos que deverão participar intensamente do processo, na especificação de testes junto a programadores, na execução e validação de produtos, na definição de telas e relatórios e na resolução de dúvidas.

As equipes devem ser formadas de profissionais com alta capacitação e multifuncionais que deverão ser capazes de realizar todas as tarefas dentro do desenvolvimento. Nessas equipes deverão constar também os clientes ou usuários operacionais da equipe.

No início, cada estória definida pelo cliente é analisada pela equipe e são definidas tarefas essenciais para as estórias. O conjunto das estórias e tarefas será chamado de *Product Backlog,* simplesmente *backlog* ou tarefas do produto. Essas estórias precisam ser pequenas e coesas, possibilitando a geração de um produto de software como resultado.

O PO deverá dar importância às estórias de forma que as mais importantes, que mais agreguem valor ao usuário sejam desenvolvidas inicialmente e as menos importantes fiquem para o final do processo de desenvolvimento. Esta repriorização deverá ser constante em função do aprendizado e da expectativa do cliente. Mudanças (ou mesmo o aparecimento de novas estórias) são esperadas e bem vindas. O PO deverá aprender durante o processo; dessa forma, deverão emergir novas soluções, nova priorização, mudança de conceitos, gerando o que se pode considerar um aumento na qualidade do produto; por isso, estas mudanças são bem vindas.

As estórias e tarefas pendentes e em produção deverão estar agrupadas e ser fisicamente visíveis a todos, preferencialmente no estilo *Kanban*, utilizando notas e cores.

O processo é realizado em ciclos ou iterações chamados *sprints*, que normalmente duram entre uma e quatro semanas, preferencialmente duas, e toda a equipe se responsabiliza solidariamente ao desenvolvimento de uma ou mais estórias.

No início de cada *sprint* é realizada uma reunião de planejamento do *sprint*. O PO já deverá ter a lista de estórias repriorizada e deverá explicar quais as características da estória que deseja ver desenvolvida. A equipe deve discutir e planejar o tempo e as tarefas necessárias ao desenvolvimento da estória. Este prazo deve ser determinado pela equipe e não será negociável e isto configura a qualidade interna do processo. O PO somente pode interferir na qualidade externa, mudando as características do que deseja.

Diariamente serão conduzidas reuniões em pé, buscando a sincronia da equipe e dos trabalhos realizados, onde deverão ser relatados, o mais brevemente possível, os trabalhos finalizados, impedimentos encontrados e trabalhos futuros.

O processo de desenvolvimento é preferencialmente especificado por testes com a participação do cliente. Estes testes realizam-se de forma incremental, inicialmente simples e aos poucos agregando complexidade até atingir eficiência sob o ponto de vista do usuário. Estes testes serão integrados ao ambiente de integração contínua que será responsável pela execução automática de todos os testes já produzidos, pela construção dos produtos e pela notificação a todos os membros da equipe a respeito das alterações produzidas nos produtos. O usuário será responsável pela execução e validação dos produtos integrados por esta ferramenta.

O processo termina quando o PO estiver satisfeito.

De uma forma geral, nos processos ágeis, os requisitos do software a ser desenvolvido ou mantido são determinados pelo cliente, conforme descrito no processo *Scrum*.

3.2.3. Extreme Programming (XP)

A Programação Extrema (XP) (BECK, 2000) é um conjunto de práticas para o desenvolvimento de software com o objetivo de produzir software de qualidade para atender as necessidades do cliente ou usuário.

Um dos aspectos principais no XP é o conceito de time. O time no XP é um grupo coeso, que assume a responsabilidade por todos o projeto. Neste grupo existirão os desenvolvedores e representantes do cliente, porém todos deverão dividir responsabilidade sob o produto a ser desenvolvido. O cliente também é responsável solidariamente, porém terá atribuições específicas sob sua responsabilidade. Outros membros deverão ser capazes de realizar todas as outras tarefas, não existindo o papel de especialista idealmente.

Durante o processo de desenvolvimento o time replaneja o projeto a cada iteração,

revendo os requisitos restantes, medindo a velocidade do time, prevendo o término do projeto, e revendo os erros e acertos até então.

O trabalho é desenvolvido visando a entrega de subprodutos pequenos. Estes produtos são trabalhados durante uma iteração, que tem como foco o valor do produto para o cliente. Dentro da iteração o cliente deverá participar da especificação de testes para o produto a ser desenvolvido. Durante o desenvolvimento este produto será continuamente submetido aos testes planejados até que passe em todos. Ao passar nos testes iniciais, estes deverão ser refinados, retrabalhando o produto até que se atinja os objetivos do cliente. Cabe observar que o produto deverá passar não somente pelos seus testes, mas o conjunto do software também deverá passar por todos os outros testes já desenvolvidos desde o início do processo.

É recomendada a programação em pares visando o aprendizado dos desenvolvedores e a simplicidade do código. A arquitetura das soluções deverá evoluir aos poucos, mantendo a simplicidade. Cada membro da equipe pode alterar o código livremente. O código deverá ser produzido tendo em vista padrões definidos pela equipe visando a compreensão de todos, pois todos serão considerados autores de todo o código.

O produto desenvolvido deverá estar sempre pronto para testes pelo cliente, e para tanto, o ambiente de desenvolvimento deverá contar com o recurso de integração contínua (DUVALL, 2002), que realiza os testes e prepara o produto para execução.

Todos devem trabalhar e agir com respeito incondicionalmente.

É comum a presença de um facilitador ou concelheiro na equipe, para orientar, viabilizar as tarefas e trazer a equipe para os objetivos do XP e do produto.

Em resumo, pode-se observar que o método XP provê ferramentas e práticas de programação e desenvolvimento de software não previstas nas práticas *Scrum* e *Lean*, tais como: integração contínua e programação em pares, por exemplo. *Scrum* possibilita uma visão gerencial e leve para o desenvolvimento de software, enquanto *Lean* preconiza um conjunto de princípios que influencia a maioria das metodologias ágeis.

4. Vantagens do Processo de Desenvolvimento Ágil

Como explicitado anteriormente, no processo tradicional um contrato de prestação de serviços apresenta seu escopo fechado, determinando exatamente o que será feito e entregue após o desenvolvimento do software. Diferentemente, em um contrato para desenvolvimento ágil, o contratante tem a oportunidade de fazer um contrato por um prazo curto, renovável(eis) enquanto o processo tiver resultados satisfatórios, ou terminar ao final de cada etapa, normalmente a cada dois meses, obtendo, porém, produtos prontos a cada iteração. É importante observar que é sempre o cliente quem define qual sub-produto é o mais importante, qual agrega mais valor para a empresa e que, por isso, deseja-se receber a cada iteração.

No processo tradicional, o produto é entregue somente no final do processo, que pode demorar de seis meses até anos. Este tempo de desenvolvimento com o escopo fechado pode gerar envelhecimento, desatualização ou mesmo perecimento dos requisitos e o software, ao final do processo de desenvolvimento, muitas vezes pode tornar-se inútil. Em contraste, os métodos ágeis oferecem entregas rápidas, de subprodutos os mais importantes possíveis, definíveis dinamicamente pelo cliente.

No processo tradicional, o cliente que já investiu no produto desejado inicialmente, pode perceber que não tem mais interesse no produto, e neste caso, perderá seu investimento.

Nos métodos ágeis, os resultados de cada iteração poderão compor ferramental do cliente, sendo produtos prontos e entregues. E ainda, a interrupção de um contrato ágil pode ser feita a cada renovação, implicando um risco financeiro menor.

O desenvolvimento ágil prevê a presença constante do cliente ou usuário na equipe de desenvolvimento. O processo de desenvolvimento tradicional prevê o levantamento minucioso de requisitos que farão parte de documentos. Estes documentos servirão de interface entre diferentes profissionais especializados, que irão criar novos documentos e assim por diante. Segundo PRESSMAN (2001), esses documentos tendem a não ser lidos após o término do software. Existe também uma tendência de tornarem-se defasados ao longo do tempo pela falta de atualização após as mudanças no projeto inicial do software. A grande maioria dos softwares atualmente desenvolvidos apresentam-se sem documentação ou defasados em relação a esta (PRESSMAN, 2001).

O processo de desenvolvimento tradicional de software baseia-se em uma sequencia de etapas praticadas exaustivamente até a mudança para a etapa posterior, p.e. análise de todos os requisitos, projeto de todo o sistema, programação de todo o sistema e teste de todo o sistema. Na prática ágil, a cada requisito, os testes servem como especificação do problema e são desenvolvidos junto ao usuário no início de cada iteração. Esta especificação de testes é técnica, executável e adicionada a um ambiente automatizado de integração, que mantém o sistema continuamente testado e integrado, anunciando a todos qualquer problema detectado.

Desta forma, o processo ágil parte de requisitos que agreguem mais valor ao produto sob o ponto de vista do cliente, minimizando o problema de integração tardia.

5. Riscos e Desvantagens

Ao desenvolver software utilizando técnicas ágeis, alguns efeitos colaterais podem ser detectados.

No desenvolvimento em incrementos, cada módulo entregue possui qualidade, baixa taxa de erros, atinge as espectativas do cliente ou usuário, porém pode ocorrer um fenômeno descrito por CARVALHO *et al.* (in pres) como *Late Integration Problem* (Problema de Integração Tardia). Este problema ocorre quando um dado processo de negócio é implementado em um software sem identificar todas as relações com os demais processos. Este problema pode desencadear manutenção e retrabalho, o que foge aos princípios do desenvolvimento ágil. Em seu trabalho, os autores (op. cit) afirmam que o framework do software deve ser capaz de promover a integração dos processos de maneira rápida, preferencialmente via configuração, desde que identificados os pontos de integração entre os processos de negócio. Esta integração se dará através de refatoração das funcionalidades desenvolvidas, a cada iteração, após estabelecer resultados satisfatórios ao cliente.

A resistência ao reuso de software também pode ser considerado um desperdício. De acordo com POULIN (1997), na teoria de reutilização se prega que todo software deve ser desenvolvido visando à possibilidade de reutilização futura, no mesmo projeto ou em outros. Não foram identificadas ações direcionadas à reutilização na literatura relacionada a processos ágeis. POULIN (op-cit) critica a falta de trabalhos científicos que abordem os aspectos da reutilização de código e prega a disseminação do conhecimento por parte dos pesquisadores para que as empresas comecem a conhecer e aplicar estes princípios. Segundo este autor, enquanto não pudermos quantificar os benefícios do reuso em termos concretos como tempo e dinheiro economizados, a prática sistemática de reuso simplesmente não irá acontecer.

6. Considerações Finais

Na literatura, alguns trabalhos iniciais observavam incorretamente que as técnicas ágeis deveriam ser aplicadas somente a trabalhos pequenos (QURESHI, 2008). Atualmente há o entendimento de que tais técnicas podem e devem ser utilizadas em trabalhos maiores, justamente por suas vantagens em suavizar os problemas advindos da volatilidade de requisitos.

Este trabalho trouxe para discussão um tema ainda pouco tratado na literatura, em especial no Brasil. Seu objetivo, portanto, foi apontar as vantagens e desvantagens dos métodos ágeis para avaliação dos leitores.

Como estudo futuro, pretende-se acompanhar a aplicação do processo ágil em um caso real, avaliando a velocidade, satisfação do usuário e ocorrência dos problemas apontados. Pretende-se também testar o fenômeno da resistência de profissionais de Tecnologia de Informação à apreciação dos princípios da engenharia de produção que fundamentam os processos ágeis. Para tanto está sendo desenvolvido um curso à distância, em ambiente Moodle, sobre processo de desenvolvimento ágil onde são aplicados questionários qualitativos que possibilitam uma análise estatística entre as impressões iniciais e as posteriores ao estudo de técnicas ágeis.

Pretende-se também estudar o alinhamento do processo ágil com técnicas de gerenciamento de projeto, como o PMBOK (PMI, 2000), visando projetizar o desenvolvimento de software sob o paradigma ágil.

Referências

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR 14724*: Informação e Documentação - Trabalhos acadêmicos - Apresentação. Rio de Janeiro: ABNT, 2001.

AGILEMANIFESTO (2001). <http://www.agilemanifesto.org>. acessado em 10/05/2009.

BECK, K. (2000) Extreme Programming Explained: Embrace Change. Addison-Wesley.

BECK, K., & CLEAL, D. (1999). Optional Scope Contracts. Xprogramming.com Publications. Acessado em 21 de agosto de 2009. <http://www.jarn.com/about/OptionalScopeContracts.pdf>

CARVALHO, R. A. ; JOHANSSON, B & MANHÃES, R. S. Lean Software Development for Customizing ERPs. In Press.

DUVALL, Paul M (2002). Continuous Integration. Boston: Addison - Wesley.

KETTUNEN, P. (2009) Adopting key lessons from agile manufacturing to agile software product development-A comparative study. Technovation. Vol. 29, n. 6-7, p. 408-422.

POPPENDIECK, M. & POPPENDIECK, T. (2003) Lean Software Development: An Agile Toolkit for Software Development Managers. ed. New York: Addison-Wesley Professional.

POULIN, J. S. (1997) On the Contributions of Reuse Research and Development to the State-of-the-Practice in reuse. In. SSR'97 - ACM SYMPOSIUM ON SOFTWARE REUSABILITY.

PRESSMAN, R. S. (1997) Software Engineering, A Practitioner's Approach. ed. McGraw-Hill, n.4.

PROJECT MANAGEMENT INSTITUTE (2000). A Guide to the Project Management Body of Knowledge (PMBOK Guide): Edição 2000.

RISING, L. & JANNOF, N. (2000) Scrum software development process for small teams. IEEE Software, Vol. 17, p. 26-32.

SOMMERVILLE, I. (1995) Software engineering. ed. Addison-Wesley. n. 5.

WOMACK, J.P., JONES, D.T. (1996). Lean Thinking. Simon & Schuster.