Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

FELIPE FERREIRA CUNHA

RECOMENDAÇÕES PARA O USO DE MÉTODOS ÁGEIS

Belo Horizonte
2012

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática. Ênfase: Engenharia de Software

RECOMENDAÇÕES PARA O USO DE MÉTODOS ÁGEIS

por

FELIPE FERREIRA CUNHA

Monografia de Final de Curso

Prof. Eduardo Magno Lages Figueiredo
Orientador

Belo Horizonte
2012

FELIPE FERREIRA CUNHA

RECOMENDAÇÕES PARA O USO DE MÉTODOS ÁGEIS

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Orientador: Eduardo Magno Lages Figueiredo

Belo Horizonte

2012

AGRADECIMENTOS

A minha família por possibilitar minha existência e formação como pessoa.
A Avenue Code e Macy’s por me possibilitar a experiência profissional para escrever este.
À Universidade Federal de Minas Gerais e também aos professores, em especial ao orientador deste projeto: Eduardo Magno Lages Figueiredo.

RESUMO

Os métodos ágeis surgiram em um contexto onde se observavam atrasos em entregas, custos maiores do que o planejado e software que não satisfazia as necessidades dos clientes. Com o passar do tempo o mercado vem se tornando cada vez mais dinâmico e competitivo, onde os prazos impostos pelos clientes são cada vez mais restritos. Neste ambiente, a escolha de um processo de desenvolvimento de software é de grande importância para a padronização, melhor entendimento dos projetos e gerência da equipe. Com isso, as empresas de desenvolvimento de software vêm buscando métodos ágeis para gestão do desenvolvimento de software. Entretanto, ocorrem casos em que organizações não possuem conhecimento suficiente de como adotar tais métodos ou, por razões de desconfiança, preferem processos tradicionais porque os métodos ágeis são relativamente novos se comparados aos tradicionais. O principal objetivo deste estudo é apresentar alguns métodos ágeis juntamente com suas características e discutir os ambientes mais favoráveis que facilitam sua adoção.

Palavras-chave: Engenharia de Software, Métodos Ágeis, Processos Tradicionais.

ABSTRACT

Agile methods have emerged in a context where it was possible to observe delays in software delivery, higher costs than planned, and software that does not meet the customer requirements. The market has become increasingly dynamic and competitive as well as the deadlines imposed by customers are increasingly restricted. In this environment, the act of choose a software development process is of great importance to standardize, better understand, and manage the project team. As a result, software companies are seeking for agile software development methods in order to manage the software development activities. However, there are cases where organizations do not have enough knowledge of how to adopt a particular agile method. Even worst, they do not trust agile methods because they are relatively new, compared to traditional ones. The main goal of this study is to present some agile methods and their characteristics. We also discuss the most favorable environments to facilitate the decision of adopting an agile method.

Keywords: Software engineering, agile methodology, traditional processes.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAS DE SIGLAS

LPS		Linhas de Produtos de Software
RAD	Rapid Application Development	Desenvolvimento rápido de aplicação
RUP	Rational Unified Process	Processo Unificado Rational
UML	Unified Modeling Language	Linguagem de Modelagem Unificada
XP	Extreme Programming	Programação Extrema

SUMÁRIO

1 INTRODUÇÃO

A indústria de software tem passado por diversas mudanças significativas nos últimos 50 anos, acelerando o desenvolvimento global, a difusão da informação e tornando a sociedade cada vez mais dependente de software. A ausência de ferramentas, métodos e procedimentos no desenvolvimento de software gera um alto índice de projetos não concluídos e projetos que são concluídos, mas que não atendem as necessidades do cliente. Como uma solução para estes problemas, em meados dos anos 70, surgiu uma área de conhecimento denominada de Engenharia de Software que utiliza de princípios de engenharia a fim de obter software confiável e de maneira econômica (Pressman, 2002).

Posteriormente, alguns processos de desenvolvimento de software conhecidos como ágeis, foram criados, devido ao fato de que os processos tradicionais já não estavam sendo bem sucedidos e o software estava em “evolução”, em direções que tornavam ainda mais problemático o desenvolvimento. Os métodos ágeis não são contra as práticas tradicionais. No entanto, seu foco está principalmente na relação entre as pessoas do projeto, e na valorização de seus talentos (Wells, 2002).

Em seu livro, Pressman afirma que a Engenharia de Software deve oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento. A Engenharia de Software pode se dividir em diversas áreas específicas. Algumas delas são enumeradas a seguir:

- Requisitos de software
- Projeto do software
- Teste do software
- Manutenção de software
- Gerência de software
- Processos de software
- Qualidade de software

Dentro das diversas áreas da Engenharia de Software, este trabalho foca em processos de software, que por sua vez, compreende diversas atividades. Tendo em vista os aspectos negativos da falta de um processo de desenvolvimento e as vantagens ao utilizá-lo, este estudo faz uma revisão de alguns métodos ágeis e discute ambientes nos quais estes métodos de desenvolvimento são propícios ou recomendados.

1.1 Objetivos

Este trabalho objetiva analisar alguns processos de desenvolvimento de software comerciais existentes na literatura, juntamente com suas características principais. Espera-se também obter indicações de ambientes onde seja favorável a utilização de métodos ágeis. Com o resultado, este trabalho pode facilitar o entendimento de quando se deve adotar um determinado processo ágil em um ambiente específico.

1.2 Justificativa

Os fundamentos da Engenharia de Software envolvem o uso de modelos de processos que permitem especificar e projetar sistemas favorecendo sua qualidade (Pressman, 2002). Com base nisto, este estudo visa apresentar métodos de desenvolvimento de software de forma a facilitar o entendimento de características presentes em métodos ágeis e a escolha do método de desenvolvimento. Como resultado prático, espera-se apoiar uma organização na adoção do método de desenvolvimento de software mais adequado ao seu ambiente. Desdobramentos futuros do trabalho podem considerar alternativas quem levem ao desenvolvimento de sistemas de software com uma maior qualidade.

1.3 Metodologia

Pesquisa em livros, artigos técnicos e sites especializados a fim de obter um embasamento teórico.

1.4 Estrutura do trabalho

O trabalho está estruturado em seis capítulos dispostos da seguinte forma:

O primeiro capítulo se destina a introdução, incluindo a justificativa, os objetivos do trabalho, além da metodologia e estrutura do trabalho.

O segundo capítulo dedica-se a apresentar alguns dos principais processos de desenvolvimento de software existentes na literatura.

No terceiro capítulo são apresentados alguns processos de desenvolvimento ágil, que são o foco desta monografia.

O quarto capítulo apresenta reflexão de alguns autores sobre a adoção dos métodos ágeis, onde são estudados os métodos ágeis de uma forma a se obter seus ambientes favoráveis, suas características e algumas recomendações.

O quinto capítulo é a conclusão, apresentando uma breve descrição do resultado obtido com esta monografia.

Por fim são apresentadas as referências.

2 PROCESSOS TRADICIONAIS DE DESENVOLVIMENTO DE SOFTWARE

Este projeto de monografia procurou analisar alguns processos tradicionais de desenvolvimento de software existentes na literatura para embasar o trabalho. O objetivo foi obter características e particularidades de cada processo. Este capítulo define o que é processo de desenvolvimento de software e apresenta alguns exemplos de processos. Para o contexto deste estudo, processos de software podem ser divididos em duas categorias: os processos tradicionais e processos ágeis. Os processos ágeis, discutidos no Capítulo 3, são o foco deste trabalho.

2.1 Processos de Desenvolvimento de Software

O conceito de processo de software, segundo a ISO 9000 é o “conjunto de atividades inter-relacionadas ou interativas que transforma insumos (entradas) em produtos (saídas)” (ISO 9000 apud SALVIANO 2005, p. 35). Para se ter garantia de um produto de qualidade, com prazos e custos adequados (objetivo sempre desejado), se faz necessário: (i) pessoas habilitadas e motivadas trabalhando e (ii) processos adequados em projetos bem gerenciados.

Um modelo de processo de desenvolvimento de software apresenta as atividades que devem ser desempenhadas no desenvolvimento de sistemas. Ele é a definição operacional do processo básico que guia o estabelecimento de um processo comum em uma organização (BORGES; FALBO, 2002). Portanto, um processo é um conjunto ordenado de atividades a fim de atingir um propósito comum. Ou seja, podemos definir processo de software como um roteiro que deve ser seguido para se alcançar o resultado esperado de forma satisfatória.

Considerando a importância da utilização de um processo padrão, considerável esforço tem sido feito para sua modelagem. Como forma de auxiliar a sua criação deste processo padrão, diversas abordagens têm sido propostos na literatura. Estes processos podem ser adotados por completo ou permitir que sejam adaptados de acordo com algumas características da organização. É possível

também que uma organização se baseie em mais de um processo para a definição de seu processo padrão (TEIXEIRA, 2007).

O processo de software é de grande importância no cotidiano de uma empresa que trabalhe com desenvolvimento de software. Para que se possa garantir facilidades nas medições de processos e qualidade de software, o processo de software fornece estabilidade, controle e organização para uma atividade - que se deixada sem controle, pode se tornar bastante caótica (PRESSMAN, 2002).

O desenvolvimento de software inicia com um problema particular que um usuário quer solucionar e termina com um sistema que soluciona este problema. O processo pode ocasionar o desenvolvimento de um novo software. Porém, é comum que um software seja desenvolvido mediante a expansão e a modificação de outros sistemas já existentes (SOMMERVILLE, 2003).

Processos tradicionais também são conhecidos como processos rigorosos, pois eles enfatizam o rigor nas premissas e propostas. Prezam pela documentação detalhada em todas as fases do desenvolvimento. Geralmente são implementados em ambientes onde é necessário exercer controle rígido, como por exemplo, quando é grande o número de indivíduos que participam do projeto. Eles também conhecidos como métodos pesados, devido à quantidade de documentos, papéis, atividades e processos considerados necessários para a implementação. O resultado disso é que os desenvolvedores passam uma parte do tempo realizando tarefas que não são necessariamente ligadas à programação ou seu processo criativo.

2.2 Exemplos de Processos Tradicionais

Os métodos ágeis (detalhados no Capitulo 3) têm sido apontadas como alternativa às abordagens tradicionais para o desenvolvimento de software. Entretanto, os processos tradicionais possuem grande utilização nas situações em que os requisitos do sistema são complexos e estáveis, e requisitos futuros são previsíveis. A tabela 1 apresenta uma breve descrição de alguns processos de desenvolvimento de software existentes na literatura.

Processo	Descrição
Cascata	Chamado também de modelo sequencial linear, teve início em uma abordagem sistemática e sequencial para o desenvolvimento de software. As atividades devem ser executadas de maneira sequencial. Uma atividade só pode começar quando outra termina (PRESSMAN, 2006). Embora seja um modelo muito estudado e simples, não condiz com a realidade dos projetos de software. Na prática, é necessário que em fases posteriores haja revisão e alteração de resultados das fases anteriores (PAULA, 2001). O problema deste processo, também chamado de processo cascata clássico, é que ele não permite a criação de fases intermediárias ou até mesmo revisão de fases anteriores antes do término do processo. Como uma alternativa, uma variação do Cascata foi criado, que também apresenta uma sequência a ser seguida mas com a opção de se retornar a uma etapa anterior. Existem documentos para cada etapa que precisam ser aprovados para que se dê início à etapa seguinte. As etapas deste modelo são: definição de requisitos, projeto do software, implementação e teste de unidade, integração e teste do sistema, operação e manutenção (SOMMERVILLE, 2003)
Incremental	Neste processo, o sistema evolui de maneira incremental ao longo do tempo a cada nova iteração. Os primeiros incrementos são versões simplificadas do produto final. Este modelo tem o objetivo de apresentar um produto operacional a cada incremento até alcançar o produto final (LARMAN, 2002).Neste processo, cada incremento incluiu uma sequência de 5 atividades: comunicação, planejamento, modelagem, construção e implantação.
RAD	O desenvolvimento rápido de aplicação (RAD) é um modelo de processo para desenvolvimento de software incremental que enfatiza um ciclo de desenvolvimento extremamente curto. É uma adaptação do modelo cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes (PRESSMAN, 2006)
Prototipagem	Prototipagem é uma técnica iterativa que tem como premissa que o desenvolvimento de sistemas pode começar com informação incompleta. Assim, requisitos completos são obtidos através de um processo cíclico de reações no usuário ao utilizar o protótipo. O importante neste processo é que o ponto de vista orientado a projeto é enriquecido com o aumento de interesse da participação do usuário final (IBM, 2001). Idealmente, o protótipo serve como um mecanismo para identificação dos requisitos do software e deve ser descartado (PRESSMAN, 2006).
Espiral	É um processo que combina a natureza iterativa da prototipagem com aspectos controlados e sistemáticos do processo cascata. O produto é desenvolvido em uma série de iterações. Cada iteração corresponde a uma volta na espiral. As primeiras iterações podem ser um modelo de papel ou protótipo, durante as últimas iterações são construídas versões cada vez mais completas do software. O risco é considerado à medida que cada evolução é feita, ou seja, este modelo exige que os riscos técnicos sejam considerados em todos os estágios do projeto. Seu principal problema é que requer uma gestão muito sofisticada para ser previsível e confiável. (PAULA, 2001, PRESSMAN 2006)

TABELA 1: Processos de desenvolvimento de software

2.2.1 RUP – *Rational Unified Process*:

O RUP é um processo de desenvolvimento de software desenvolvido pela *Rational Software Corporation*. Sua origem se deu no ano de 1980, e seu sucesso foi comprovado na aplicação de projetos grandes e complexos (R. Dennis Gibbs, 2006). O RUP pode ser aplicado em variados projetos, podendo ser tratado como um arcabouço (*framework*) genérico empenhado para os processos de desenvolvimento, considerando a configuração adequada para o tamanho e a necessidade do projeto e os padrões aplicados na empresa. Associada a este modelo, a Rational Software construiu também um conjunto de ferramentas de suporte ao desenvolvimento (RUP, 2007). A UML (*Unified Modeling Language*) é uma linguagem muito utilizada para apoiar a utilização do RUP, ela oferece um conjunto de modelos para auxiliar na criação de artefatos e modelagem do software.

> O Rational Unified Process (RUP) é um processo de engenharia de software. Ele fornece uma abordagem disciplinada para assumir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de software de alta qualidade que satisfaça as necessidades de seus usuários finais dentro do prazo e orçamento previsíveis. […] O Rational Unified Process é também uma estrutura de processo que pode ser adaptada e estendida para compor as necessidades de uma organização que o esteja adotando.
> (KRUCHTEN, 2003)

O desenvolvimento de software utilizando o RUP é feito em quatro fases, onde cada fase possui o seu ciclo de iteração, suas atividades e geram seus artefatos. Ao final da quarta fase, o produto está em produção. A Figura 1 apresenta as fases de Iniciação, Elaboração, Construção e Transição presente no RUP, além de mostrar uma indicação de esforço de cada disciplina em cada fase. Uma disciplina compreende um conjunto atividades em comum.

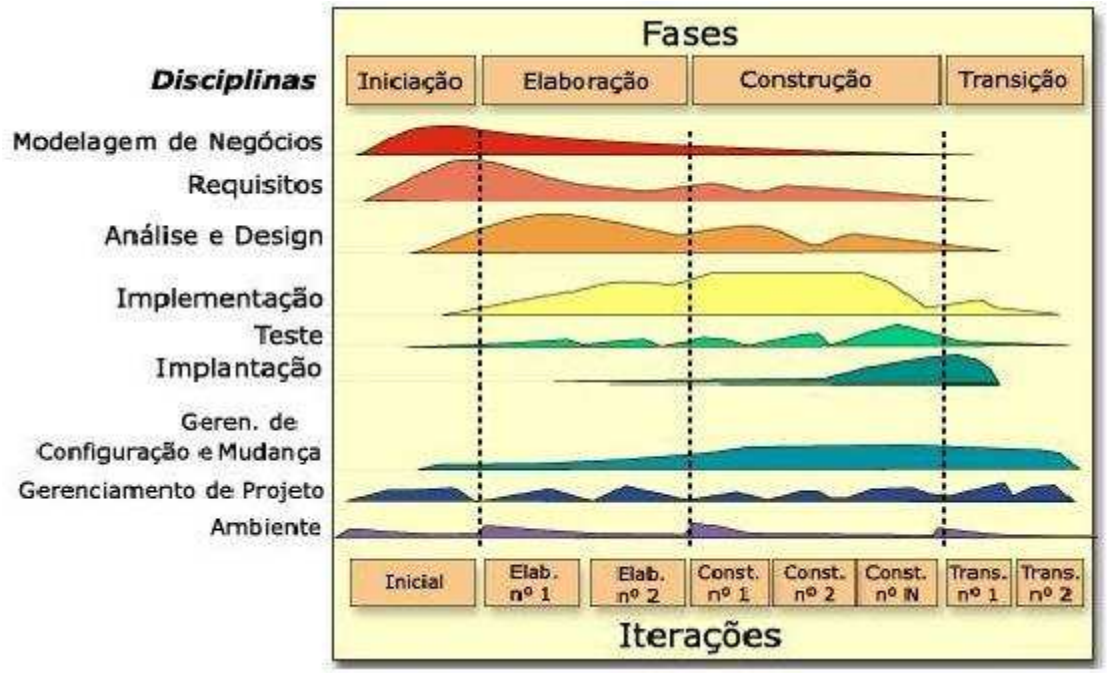

FIGURA 1 – Arquitetura RUP
Fonte: RUP, 2010

O RUP é organizado em fases e disciplinas. A Figura 1, popularmente conhecida como “gráfico das baleias”, ilustra e resume o processo. O eixo horizontal representa o tempo e mostra os aspectos de clico de vida do processo à medida que ele é desenvolvido. Esta primeira dimensão ilustra o aspecto dinâmico do processo e como ele é representado em termos de fases, iterações e marcos (*milestones*). O eixo vertical representa as disciplinas que agrupam logicamente atividades por natureza. Esta segunda dimensão retrata o aspecto estático do processo. Ele é descrito em termos de componentes de processo, disciplinas, atividades, *workflows*, artefatos e papéis (*roles*).

O RUP permite a divisão do projeto em várias partes que serão desenvolvidas através de iterações, ou seja, ocorre um ciclo de repetição para cada uma das partes que serão incrementadas até o produto final. Iterações mostradas na Figura 1 comprovam o compromisso do RUP com o desenvolvimento incremental, que é construir o código e fazer os testes com componentes do sistema, gerando versões em cada fase. No fim da fase de elaboração, o protótipo da arquitetura está disponível para avaliação. Durante as iterações da fase de Construção, componentes terminados são integrados ao projeto. O ponto chave para este elemento é um conjunto de testes que acompanham a construção do produto

(PROBASCO, 2000). A tabela 2 mostra os artefatos de pessoas envolvidas em cada etapa do processo.

Etapa	Artefatos	Papeis Envolvidos
Iniciação	Visão, Caso de Negócio, Lista de Riscos, Plano de Desenvolvimento de Software, Plano de Iteração, Caso de Desenvolvimento, Ferramentas, Glossário, Modelo de Casos de Uso (Atores, Casos de Uso), Repositório do Projeto, Solicitação de Mudanças	Analista de Sistemas, Gerente de Projeto, Engenheiro de Processo, Especialista em Ferramentas, Analista de Sistemas, Gerente de Configuração, Gerente de Controle de Mudanças
Elaboração	Protótipos, Lista de Riscos, Caso de Desenvolvimento, Ferramentas, Documento de Arquitetura de Software, Modelo de Desenho, Modelo de Dados, Modelo de Implementação, Visão, Plano de Desenvolvimento de Software, Guia de Desenho e Guia de Programação, Plano de Iteração, Modelo de Casos de Uso (Atores, Casos de Uso), Especificações Suplementares, Conjunto de Testes, Arquitetura para Automatização de Testes	Gerente de Projetos, Engenheiro de Processo, Especialista em Ferramentas, Arquiteto de Software, Designer de Banco de Dados, Arquiteto de Software, Analista de Sistemas, Gerente de Projetos, Analista de Sistemas, Projetista de Teste
Construção	"O sistema", Plano de Implantação, Modelo de Implementação, Conjunto de Testes, Materiais de Treinamento, Plano de Iteração, Modelo de Desenho, Caso de Desenvolvimento, Ferramentas, Modelo de Dados	Gerente de Implantação, Arquiteto de Software, Designer de Testes, Desenvolvedor do Curso, Gerente de Projetos, Arquiteto de Software, Engenheiro de Processos, Especialista em Ferramentas, Designer de Banco de Dados
Transição	A Construção do Produto, Notas de Liberações, Artefatos de Instalação, Material de Treinamento, Material de Suporte para o Usuário Final	Gerente de Implantação, Gerente de Implantação, Implementados, Desenvolvedor do Curso

TABELA 2: RUP - Fases – Artefatos e Pessoas Envolvidas
Fonte: RUP, 2010

Como pode ser observado na tabela 1, o RUP é um processo burocrático com uma grande diversidade de artefatos e envolvimento de diversos papéis, mas ele também pode ser utilizado no desenvolvimento e manutenção de projetos de pequeno ou de médio porte. Mas, para que isso seja possível, algumas etapas ou passos podem ser eliminados de acordo com as características do projeto, para simplificar ou reduzir as necessidades de documentação, minimizando a formalização (KOHRELL e WONCH ,2005). A variedade de configurações do RUP possibilita o suporte de equipes grandes e pequenas, além de técnicas de desenvolvimento disciplinadas ou menos formais (KRUCHTEN, 2000).

3 MÉTODOS ÁGEIS

Os processos de desenvolvimento de software conhecidos como métodos ágeis foram criados, devido ao fato de que os processos tradicionais já não estavam sendo bem sucedidos e o software estava em “evolução”, em direções que tornavam ainda mais problemático o desenvolvimento. Outro problema é que grande parte do esforço no desenvolvimento tradicional estava sendo usada na parte de documentação e contratos, provocando assim um grande aumento no tempo de desenvolvimento do projeto. Os métodos ágeis não são contra as práticas tradicionais. No entanto, seu foco está principalmente na relação entre as pessoas do projeto e na valorização de seus talentos (Wells, 2002).

O ceticismo por parte dos desenvolvedores a respeito das propostas tradicionais, difíceis de serem compreendidas ou colocadas em prática, favoreceu o aparecimento dos métodos ágeis. Os princípios comuns de métodos ágeis foram estabelecidos pelo Manifesto Ágil (ABRAHAMSSON, 2002). O movimento em prol de métodos ágeis surgiu por meio de um grupo de dezessete pessoas, que sentiram necessidade de divulgar uma forma alternativa para desenvolvimento de software. Este grupo, chamado de Aliança Ágil, argumenta que um método ágil deve atender quatro valores principais:

- Indivíduos e interações acima de processos e ferramentas.
- Software funcionando acima de documentação abrangente.
- Colaboração com o cliente acima de documentação abrangente.
- Responder a mudanças acima de seguir um plano.

Os princípios do Manifesto Ágil têm a finalidade de moldar um processo de desenvolvimento de software caracterizado por um forte espírito de equipe (cooperativo), simplicidade e velocidade, realizando entregas de versões parciais executáveis e testadas (incremental), em intervalos curtos e freqüentes (iterativo), com documentação em nível apropriado (direto), envolvimento do cliente no processo (interativo) e preparação para mudanças (adaptativo). Esses conceitos,

abordados no manifesto ágil, estão mais relacionados à forma que as pequenas e médias empresas trabalham devido a estas estarem mais habituadas a mudanças.

O desenvolvimento de software no estilo agilista impõe vários desafios às empresas que utilizam métodos tradicionais, já que as mesmas estão fundamentadas em conceitos não muito próximos. Entre esses desafios há a mudança do uso de técnicas centradas em processos para técnicas centradas em características e pessoas e o desenvolvimento iterativo com ênfase na adaptabilidade (NERUR, MAHAPATRA e MANGALARAJ, 2005).

Diversos métodos ágeis focam o desenvolvimento rápido de software, no qual programadores em contato com o cliente desenvolvem múltiplas versões de um sistema, até resultar em um software funcional que o satisfaça. Apesar dos relatos de sucesso, existe um problema nesta abordagem. Um método ágil não vê o software como parte de um sistema funcional mais amplo, que interage com uma variedade de pessoas. Portanto, pela filosofia ágil, sistemas são desenvolvidos de forma isolada, sendo observados somente os pontos de vista de seus programadores e do próprio cliente. Tal filosofia tem sido a causa de muitas falhas em projetos (GOTTERBARN, 2004).

Linhas de Produtos de Software (LPS) constituem uma abordagem emergente para projeto e implementação de sistemas que almeja promover o reuso sistemático de componentes de software. O objetivo dessa abordagem é migrar para uma cultura de desenvolvimento onde novos sistemas são derivados a partir de um conjunto de componentes e artefatos comuns, os quais constituem o núcleo da linha de produtos (CLEMENTS e NORTHROP).

Existe uma aparente incompatibilidade entre práticas ágeis e LPS (Linhas de Produtos de Software). Os princípios ágeis focam a simplicidade e velocidade, com implementação de versões incrementais de requisitos atuais. De forma diferente, a LPS faz um planejamento inicial de todos os produtos que poderão ser derivados dela, e que serão desenvolvidos com base em uma arquitetura comum e um núcleo de artefatos reutilizáveis projetados previamente (PAIGE, 2006).

3.1 XP – eXtreme Programming:

Dos métodos ágeis, o *Extreme Programming* (XP), é um dos métodos mais difundidos. Ele permite atender seus clientes em um prazo mais curto, além de focar sempre nos indivíduos do projeto e contar com a participação ativa do cliente no desenvolvimento do projeto.

Sua primeira aplicação se deu no projeto C3, dirigido pela empresa Chrysler. O sucesso alcançado despertou a curiosidade de outros pesquisadores da área. Com isso, outras empresas adotaram este método como parte dos seus projetos. Depois de anos tentando outros métodos sem resultado, o uso do novo método trouxe bons resultados em pouco mais de um ano (HIGHSMITH, 2000).

3.1.1 Práticas do XP

O XP possui doze práticas que constitui o núcleo principal do processo. Elas foram escolhidas com base em quatro valores que são: comunicação, simplicidade, *feedback* e coragem (XP, 2009). Segundo Kent Beck, citado por Vasconcelos (2005), estas práticas não são novidades, elas já estavam sendo utilizadas há muitos anos, com eficiência em projetos de software. As doze práticas do XP são descritas abaixo:

- Testes constantes – no XP existem dois tipos de testes, os testes de unidade e os testes funcionais. Os testes de unidade são automaticamente executados assim que o programador escreve o código, e os testes funcionais são executados juntamente com o cliente para verificar a conformidade entre os requisitos e o funcionamento do sistema.
- Refatoração – o refatoração é uma técnica que tem por finalidade melhorar o código do sistema, tornando-o mais legível e otimizado, mas sem alterar as funcionalidades do sistema.
- Programação em pares – o desenvolvimento do sistema ocorre em pares, enquanto uma pessoa fica responsável por codificar e pensar nos algoritmos, a outra pessoa fica observando, tentando encontrar melhorias no código, tornando-o mais simples.

- Propriedade coletiva do código – a propriedade coletiva torna a equipe mais unida, e mais focada em seus objetivos. Além de respeitarem padrões de codificação, a integridade do código, pode ser garantida com o uso de ferramentas de controle de versão.
- Integração contínua – A integração é realizada constantemente, assim que uma parte é implementada e testada, ela será integrada ao sistema.
- Semana de quarenta horas – o XP não exige uma carga horária fixa. No entanto é importante obedecer alguns limites, para que não seja afetada a produtividade da equipe.
- Cliente no local – uma pessoa por parte do cliente será integrada a equipe, e participará do desenvolvimento do sistema, respondendo as perguntas e dúvidas. Caso não haja uma pessoa por parte do cliente, será necessário alguém com conhecimento do negócio.
- Padrões de codificação – é imprescindível que se obedeça a certos padrões de codificação, uma vez que o XP realiza programação aos pares.
- O jogo do planejamento – o planejamento é constante no XP, define-se o escopo de versões do sistema, que são implementadas, e novos planejamentos para novas versões serão estudadas. Existe a participação de duas partes pelo lado do cliente, uma pessoa que compreenda o negócio e a pessoa que atuará no sistema.
- Versões pequenas – um release deve ser o quanto menor possível, atendendo aos requisitos mais importantes do sistema, isso garante um maior *feedback* para o cliente.
- Metáfora – uma metáfora consiste em fazer uma analogia entre o sistema que está sendo desenvolvido, e um outro sistema qualquer, não necessariamente um software. O objetivo é tornar mais compreensível o funcionamento do software por parte dos desenvolvedores e cliente.
- Projeto simples – devem-se atender as funcionalidades principais do sistema, e focar nas necessidades imediatas do sistema, não se preocupando em atender requisitos futuros, uma vez que os requisitos são inconstantes e podem ser alterados a qualquer momento.

Aplicar as práticas de forma isolada pode não produzir a agilidade desejada, sendo que as mesmas podem não fazer sentido à primeira vista. As doze práticas do XP apoiam-se umas às outras, devendo ser usadas em conjunto e todas devem ser aplicadas para se ter agilidade no desenvolvimento. A maior preocupação é com desenvolvimento rápido do sistema e a satisfação do cliente, cumprindo as estimativas de tempo e custo. A Figura 2 mostra algumas características do XP:

FIGURA 2 – Características do XP
Adaptação de: XP, 2009.

A comunicação visa criar um vínculo entre o cliente e o provedor de serviços para manter o melhor relacionamento possível, dando preferência, neste caso, a conversas pessoais do que através de outros meios. É encorajada também a comunicação entre desenvolvedores e gerente do projeto. Este se caracteriza como um dos fatores principais na XP: conversar ao máximo pessoalmente, evitando o uso do telefone e mensagens de e-mail.

A simplicidade tem a finalidade de minimizar o código, desconsiderando funções julgadas desnecessárias ou não essenciais. Isto significa que se deve implementar o menor número de classes e métodos. Deve-se também estar sempre

procurando atender os requisitos emergenciais, evitando adicionar funcionalidades ligadas à evolução do produto. O importante é considerar que todos os requisitos são mutáveis. Portanto, o interessante na prática de Extreme Programming é implementar apenas o que é estritamente necessário.

O *feedback* constante se refere ao contato incessante com o cliente a respeito do projeto. Informações sobre o código são passadas por testes periódicos, que indicam erros tanto individuais quanto do software integrado. Além disso, o cliente terá sempre uma parte do software funcional para avaliar. Com isso, novas características e informações são repassadas aos desenvolvedores, que por sua vez devem implementá-las nas próximas versões. Desta maneira, o que se pretende é entregar o software de acordo com as expectativas do cliente.

A coragem se encaixa na implantação dos três valores anteriores. São necessários profissionais comunicativos e com facilidade de se relacionar auxiliando a simplicidade, quando a possibilidade de simplificar o software é detectada. Por fim, a coragem é necessária para garantir que o *feedback* do cliente ocorra com frequência, pois esta abordagem implicará na possibilidade de mudanças constantes do produto.

A orientação para o perfil das equipes em XP é de programadores experientes, já que é necessária a boa utilização do tempo de desenvolvimento do projeto. Alguns dos papéis identificados em XP são: Programador, Cliente, Testador, Investigador, Orientador, Consultor e Gerente. Ressalta-se que o testador não é necessariamente uma pessoa que realiza somente essa atividade, porém não é apropriado que ele teste um código caso ele o tenha programado (BECK, 2004).

O XP é indicado para equipes pequenas e médias, com até dez integrantes, que desenvolvem software baseado em requisitos que não estejam totalmente definidos e que se modificam de forma rápida. Ele enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As principais diferenças da XP em relação aos outros métodos são *feedback* constante e abordagem incremental (BECK, 2004).

3.1.2 Ciclo de vida de um processo XP

O ciclo de vida de um processo XP é extremamente curto em comparação aos processos tradicionais. Essa questão é muito discutida entre diversos autores da área. No entanto, o XP é um processo que pode se adequar mais facilmente a projetos onde se têm requisitos muito instáveis, quando não há a necessidade de rever requisitos que foram previstos no início do projeto, ou de se acrescentar novos requisitos durante o decorrer do projeto (BONA, 2002).

Segundo Bona (2002) o XP compreende as fases a seguir:

- Exploração – esta é a fase onde se deve entender o propósito do sistema, onde o cliente deve relatar o sistema na forma de histórias, e os programadores por sua vez irão estimar as histórias contadas.
- Planejamento – o objetivo da fase de planejamento é definir o menor cronograma possível e o maior número de histórias. Deve-se em primeiro momento selecionar as histórias essenciais para a primeira versão do sistema. O cliente irá avaliar as historias de acordo com o grau de prioridade e os programadores de acordo com os riscos apresentados. As histórias serão divididas em tarefas e alocadas entre os programadores, que por sua vez irão estimar o tempo previsto de cada tarefa de acordo com a experiência dos mesmos.
- Iteração para a primeira versão – o sistema é desenvolvido ao longo de várias iterações, em cada iteração serão realizadas as tarefas estipuladas, e ao final de cada iteração serão realizados os testes. Na primeira iteração será testada a arquitetura previamente definida, por essa razão convêm-se que seja escolhido um número de tarefas mínimas que consigam representar o sistema como um todo. Ao final das iterações o cliente terá testado todo o sistema e o mesmo estará pronto para ser produzido.
- Produção – a produção inicia-se quando se encerra o feedback por parte do cliente, no entanto, nesta fase novos testes podem ser feitos pela equipe desenvolvedora para validar realmente a funcionalidade do sistema.
- Manutenção - diferentemente de outros processos, esta fase no XP é normalmente executada dentro do ciclo de vida de um projeto. Durante esta fase novas funcionalidades podem ser incorporadas ao sistema, pode-se

questionar pela atualização da tecnologia anteriormente adotada, e é natural que ocorra uma mudança na equipe do projeto, devido ao fato de que agora não serão necessários tantos programadores. É importante ressaltar que o sistema já está em funcionamento, as novas funcionalidades serão desenvolvidas de forma paralela.

- Fim do projeto – o projeto chega ao fim quando o cliente não tem mais histórias para relatar, nesse momento é importante que a equipe desenvolvedora do projeto elabore um documento relatando as funcionalidades do sistema, e algumas informações básicas, caso seja necessário fazer algumas alterações no sistema futuramente. O ponto principal é que se tenha conseguido satisfazer as expectativas do cliente, e também é necessário que a equipe se reúna, para avaliar o trabalho realizado, e assim adquirir novos aprendizados para trabalhos futuros.

3.2 Scrum

Scrum é um método ágil de desenvolvimento de software, criado por Jeff Sutherland e sua equipe no início da década de 90. O seu nome é atribuído a uma atividade que ocorre durante um jogo de rugby, onde vários jogadores se reúnem com a finalidade de mover a bola pelo campo.(PRESSMAN, 2006).

Os princípios de Scrum são consistentes com o manifesto ágil:

- Pequenas equipes, com a finalidade de maximizar a comunicação, minimizar a supervisão, .
- O processo precisa ser adaptável, tanto a nível técnico quanto de negócio para garantir que o melhor produto seja produzido.
- Produzir frequentes incrementos de software que podem ser inspecionados, ajustados, testados, documentados e expandidos.
- O trabalho de desenvolvimento e equipe que participa do mesmo é dividido em participações claras, de baixo acoplamento, ou em pacotes.
- Teste e documentação constantes são produzidas durante o desenvolvimento.

Estes princípios do Scrum são seguidos com a finalidade de direcionar o desenvolvimento seguindo o processo que incluem as atividades de requisitos, análise, projeto, evolução e entrega. Conduzidas por um padrão de processo chamado *Sprint*, em tempo real, estas atividades são conduzidas e adaptadas pela equipe Scrum ao problema que se tem em mãos. Este processo é ilustrado na Figura 3 a seguir.

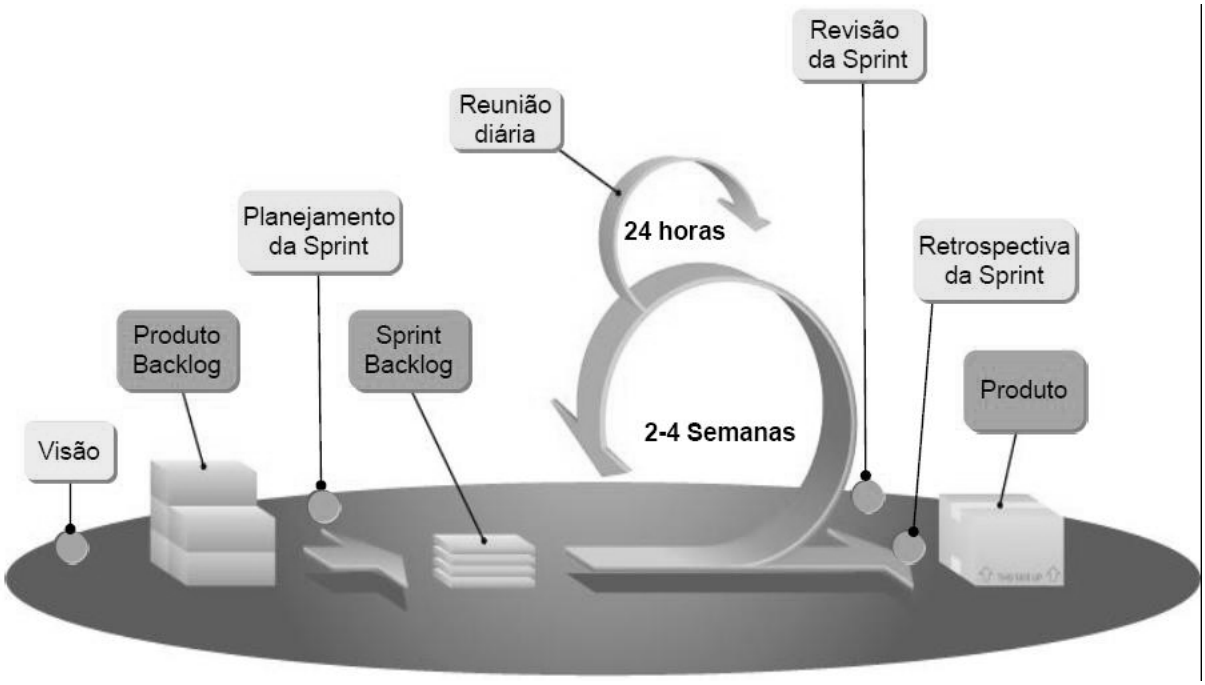

FIGURA 3 – Fluxo do processo Scrum
Fonte: PRESSMAN, 2006.

3.2.1 Papéis do Scrum

O Scrum possui apenas os três papéis principais apresentados a seguir.

Mestre Scrum - Além da responsabilidade de garantir o entendimento do processo Scrum, ele também é responsável por coordenador, acompanhar, monitorar e validar os desenvolvimentos realizados pela equipe. Ele também agenda e participa das reuniões.

Dono do Produto - é o especialista no produto. Além de controlar o *Product Backlog* (requisitos/funcionalidades que o produto possui). Ele é responsável por definir e priorizar os requisitos e ao final validar os itens requisitados.

Time - é a equipe de desenvolvimento que é responsável pela execução e implementação das funcionalidades

3.2.2 Eventos com duração fixa empregados no Scrum

Planejamento do Sprint*:* Fase onde o propósito é expor a equipe informação suficiente para o trabalho. Ele visa dar a equipe informação sobre o trabalho e confiança ao dono do produto para deixá-los fazerem isso, neste planejamento decide quais itens do *product backlog* serão implementados no *sprint* em questão.

> O planejamento de sprint é uma reunião crítica, provavelmente o evento mais importante no Scrum. Um encontro de planejamento de sprint mal feito pode bagunçar totalmente um sprint. (KNIBERG, 2007)

Sprint: Iteração que tem o objetivo de entregar um incremento do produto ao final de um período de 2 a 4 semanas.

Reunião Diária*:* Reunião diária com duração de no máximo 15 minutos. Nesta reunião todos os membros do time devem reportar o que fizeram desde a ultima reunião, o que vão fazer até a próxima reunião e quais o problemas ou impedimentos a realização do seu trabalho. O Mestre Scrum deverá atualizar o Burndown Chart e o quadro de Kanban.

Revisão do *Sprint* e Retrospectiva do *Sprint:* Retrospectiva da ultima *Sprint* com a participação do mestre *Scrum,* dono do produto e o time com o objetivo de rever os procedimentos executados e sugerir melhorias.

3.2.3 Principais artefatos do Scrum

Sprint: É uma interação de duas a quatro semanas, todas as sprint tem como resultado um incremento do produto que pode ser entregue.

Product Backlog: Conhecido também como estórias ou itens de *backlog,* é uma lista priorizada do que o produto precisa ou pode ter, com uma estimativa e normalmente priorizada pelo dono do produto juntamente com o mestre *Scrum*.

> O product backlog é o coração do Scrum. É aqui que tudo começa. O product backlog é basicamente uma lista de requisitos, estórias, coisas que o cliente deseja, descritas utilizando a terminologia do cliente. (KNIBERG, 2007)

Sprint Backlog: Lista de tarefas e sequência criada pelo mestre Scrum para produzir o *Product Backlog* em um incremento entregável no *Sprint* em questão.

Burndown: É uma medida, pode variar dentro do contexto. Podemos ter um

Burndown de *Sprint* para medir o quanto falta das tarefas dentro do tempo do *Sprint*.

3.2.4 Práticas usadas no Scrum

Planning poker: É uma prática utilizada na estimativa de estórias ou tarefas. Uma vez escolhida a estória, os membros deverão estimar a complexidade da estória mostrando uma carta com a estimativa e expor o porquê daquele número. Espera-se obter um consenso sobre a complexidade ao final da estimativa.

Quadro de Kanban: É um quadro que mostra as tarefas que deverão ser executadas pela equipe, ele contem as estorias e seus status possíveis (*Sprint Backlog*, Em execução, Concluído e *BurnDown*). A estória deve mover no quadro conforme seu status.

Gráfico de *Burn-down:* Este diagrama permite a equipe monitorar o quanto de trabalho deve ser implementado para que se possa implementar a incremento da sprint. Este gráfico permite a equipe visualizar a quantidade de pontos realizado e o estimado ao longo do S*print*.

4 FATORES PARA ADOÇÃO DE MÉTODOS ÁGEIS

Decidir qual método de desenvolvimento de software adotar é uma tarefa de grande importância, mas nem sempre fácil. Existem entretanto algumas características, dependendo do tipo de empresa, projeto ou equipe que podem ser mais favoráveis para a adoção de um processo específico de desenvolvimento.

Este capítulo visa pesquisar sobre ambientes favoráveis para a adoção ou execução de um método de desenvolvimento ágil. Com isso, permite-se um melhor entendimento do correto funcionamento de um método ágil e também permite-se indicar situações em que um projeto ou empresa poderia adotá-lo.

4.1 Ambientes favoráveis aos métodos ágeis

Alguns métodos ágeis, afirmam se aplicar a qualquer projeto de desenvolvimento ágil, sem importar suas características (Abrahamsonn et al., 2003). Outros são mais conservadores e provêm algumas variações de um mesmo método para ser selecionado de acordo com algumas características do projeto. A Figura 4 mostra os critérios utilizados por um processo de desenvolvimento ágil que possui uma variação de processos baseada no tamanho do projeto, criticidade e prioridade. Contudo, poucos métodos ágeis fornecem um instrumento explícito para definir um ambiente favorável à sua aplicabilidade.

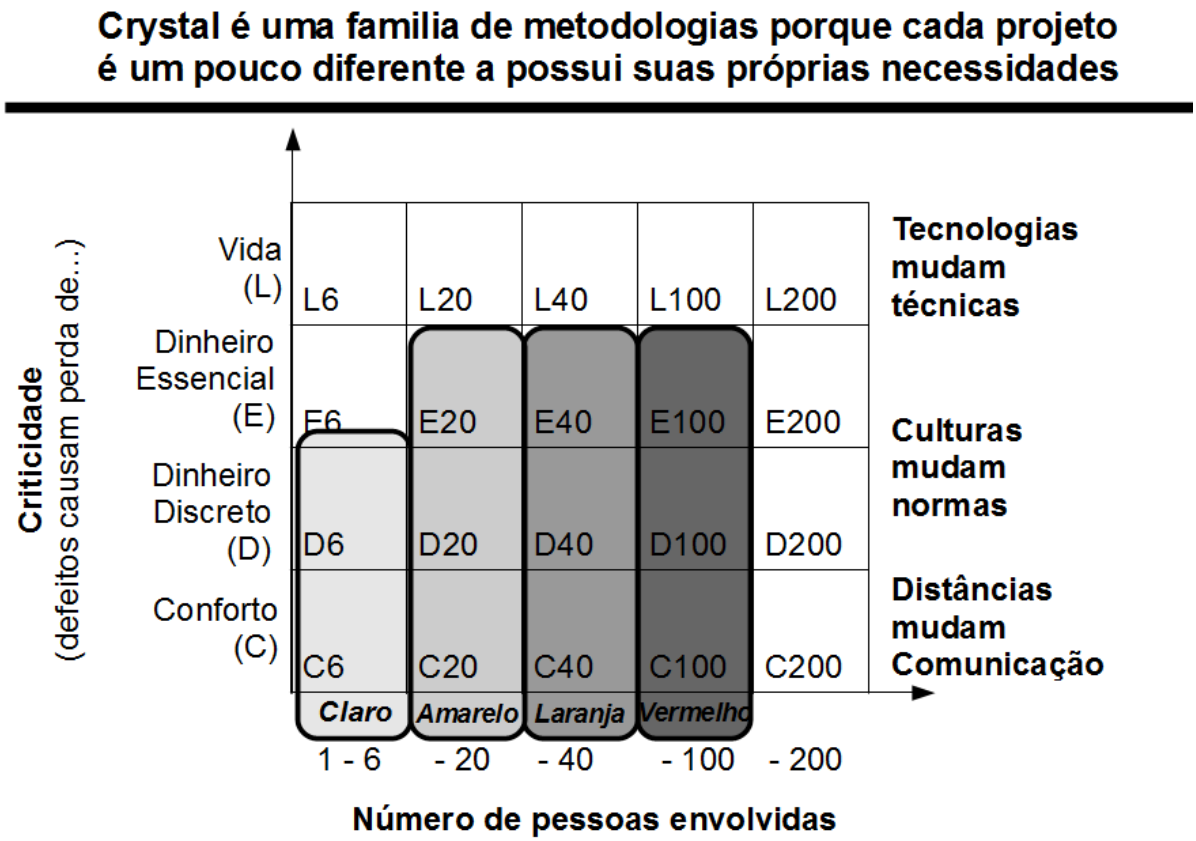

FIGURA 4 - Métodos Crystal por criticidade e número de pessoas envolvidas

A aplicabilidade dos métodos ágeis em geral pode ser examinada de múltiplas perspectivas. Da perspectiva do produto, métodos ágeis são mais adequados quando os requisitos estão emergindo e mudando rapidamente, embora não exista um consenso completo neste ponto. De uma perspectiva organizacional, a aplicabilidade pode ser expressa examinando três dimensões chaves da organização: cultura, pessoal e comunicação. Em relação a estas áreas inúmeros fatores chave do sucesso podem ser identificados (Cohen et al., 2004).

Barry Boehm e Richard Turner sugeriram que um ambiente ideal para o desenvolvimento ágil deve ser:

- Baixa criticidade
- Desenvolvedores seniores
- Mudanças frequentes de requisitos
- Pequeno número de desenvolvedores
- Cultura que tem sucesso no caos.

4.2 Características para um correto funcionamento dos métodos ágeis

Embora os métodos ágeis apresentem diferenças entre suas práticas, eles compartilham inúmeras características em comum, incluindo o desenvolvimento iterativo, foco na comunicação interativa e na redução do esforço empregado em artefatos intermediários (Cohen et al., 2004). Baseado na análise de alguns autores, pode-se afirmar que as características apresentadas nas subseções a seguir são fatores que podem ser decisivos para a adoção de um método ágil.

4.2.1 Projetos pequenos

O fator mais importante é provavelmente o tamanho do projeto (Cohen et al., 2004). Com o aumento do tamanho, a comunicação direta se torna mais difícil, a disponibilidade do cliente para atender a todos os requisitos é menor, as reuniões regulares passam a tomar um tempo maior e se tornam mais difíceis de serem mantidas. Portanto, métodos ágeis são mais adequados para projetos com pequenos times, com no máximo de 20 a 40 pessoas (Cohen et al., 2004).

Um projeto ágil deve ser composto por um número pequeno de desenvolvedores (Boehm, 2003). Um projeto com um número grande de pessoas, afeta diretamente no processo de desenvolvimento, sendo este um dos dois fatores usado pela família Crystal para se escolher entre um método ágil ou rigoroso, conforme demonstrado na Figura 4.

4.2.2 Equipe experiente

Um projeto ágil deve ser composto por desenvolvedores sênior (Boehm, 2003), capazes de manter continua atenção a excelência técnica e bom design, aumentando a agilidade (Manifesto, 2012). É aconselhado possuir uma equipe composta de poucas pessoas, mas confiantes e competentes (Cohen et al., 2004).

As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis. Além disso, é essencial que a equipe seja capaz de manter a simplicidade e maximizar a quantidade de trabalho não realizado (Manifesto, 2012)

4.2.3 Colaboração do cliente

Os métodos de desenvolvimentos ágeis possuem a finalidade de adaptar rapidamente a mudanças do projeto e aceitar mudanças nos requisitos, invés de se manter fiel a documentações entregues ao cliente e renegociações de contrato. É de grande importância portanto, que o cliente esteja colaborando e acompanhando o andamento do projeto. (Manifesto, 2012)

4.2.4 Cultura que tem sucesso no caos

Os métodos ágeis possuem em seus fundamentos a idéia de equipes auto-organizáveis e mudanças frequentes de requisitos, ao invés de se seguir um plano ditado ou tentar controlar todo o processo (Manifesto, 2012). Ao se adotar um processo de desenvolvimento ágil, deve-se estar ciente que o foco do desenvolvimento ágil baseia-se em resposta rápida a mudanças, não sendo aconselhado sua adoção por uma organização que possui uma alta hierarquia de controle e ordem (Boehm, 2003).

4.2.5 Entregas contínuas de software

Diferentemente de alguns modelos de processo de desenvolvimento tradicional, que visa a entrega o software apenas em um estágio avançado, o desenvolvimento ágil visa entrega de software funcional em períodos curtos, de poucas semanas a poucos meses (Manifesto, 2012). A organização deve portanto, estar preparada e possuir uma equipe capaz de percorrer todas as etapas de desenvolvimento do software, desde a fase inicial do processo.

4.2.6 Pessoas do negócio e desenvolvedores trabalham juntos

Ocorrem casos em que uma organização segmenta o desenvolvimento do software em diversas equipes, separando a equipe de desenvolvimento da equipe de negócio. Esta segmentação não é saudável para o desenvolvimento ágil do software, em que deve-se obter um alto nível de integração e comunicação. É interessante que se forme equipes com pessoas de várias áreas, possibilitando que as pessoas do negócios consigam trabalhar junto dos desenvolvedores.

4.2.7 Apoio e confiança na equipe

A organização deve confiar no trabalho realizado pela equipe e fornecer o ambiente e o suporte que a eles precisam de forma que consiga manter os indivíduos motivados (Manifesto, 2012).

4.2.8 Alto nível de comunicação

Métodos ágeis priorizam comunicação em tempo real a documentos escritos, ou seja, enfatizam trabalho no software como uma medida primária do progresso. Segundo o Manifesto Ágil, o método mais eficaz de comunicação é a conversa face a face. A Organização necessita ter um ambiente que facilite a rápida comunicação entre os membros (Cohen et al., 2004).

4.2.9 Reuniões regulares

Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo (Cohen et al., 2004). Reuniões regulares também aumentam o nível de comunicação da equipe, fator muito importante no processo de desenvolvimento ágil.

4.2.10 Projetos de baixa criticidade

Conforme pode ser observado na Figura 4, um dos dois fatores que a família de desenvolvimento Crystal utiliza, é a criticidade do projeto. Podemos observar também que nenhum dos processos Crystal atendem a sistemas críticos onde ocorrem risco de vida. Os métodos ágeis são mais adequados para projetos de baixa criticidade (Boehm, 2003).

4.2.11 A cultura da organização deve apoiar a negociação

Devido a constantes mudanças, esperadas por um processo de desenvolvimento ágil, a organização deve estar culturalmente receptível para as mudanças e as constantes negociações (Cohen et al., 2004).

4.2.12 A organização deve promover as decisões que os desenvolvedores tomam

Em um processo tradicional, muitas vezes as decisões sobre o desenho e arquitetura são tomas sem o conhecimento do desenvolvedor. Em um processo ágil a equipe técnica e do negócio devem trabalhar juntas e os desenvolvedores devem fazer parte das decisões do projeto. Este procedimento pode um muitos casos evitar que barreiras técnicas sejam descobertas em uma fase já avançada do projeto. A organização deve portanto, aceitar ou mesmo incentivar os desenvolvedores a contribuírem com as decisões do projeto (Cohen et al., 2004).

4.3 Recomendações

Baseado nos estudos realizados neste capítulo sobre os ambientes e características importantes no processo de desenvolvimento ágil, podemos observar que o desenvolvimento ágil é particularmente adequado para equipes que têm que lidar com mudanças rápidas ou imprevisíveis nos requisitos. Apesar de apontarem um ambiente ideal para desenvolvimento ágil, conforme descrito no capítulo 4.1, Barry Boehm e Richard Turner também sugerem que a análise de riscos pode ser usada para escolher entre métodos adaptativos ("Ágeis") e preditivos ("dirigidos pelo planejamento").

É recomendável que uma organização interessada em utilizar um método ágil, atenda a todos os princípios ágeis, visto que um princípio completa outro, aumentando assim, a probabilidades de sucesso do método. De uma perspectiva organizacional, a aplicabilidade de um método ágil pode ser expressa examinando algumas características do projeto, da cultura, do pessoal e da comunicação.

- Da perspectiva do projeto, não é recomendável que se utilize um processo ágil para sistemas críticos, projetos grandes ou com mais de 40 pessoas. Sendo entretanto, mais eficiente para sistemas em que não se possui os requisitos bem definidos já no início do projeto e que podem mudar rapidamente.
- De uma perspectiva de pessoal, a equipe deve ser experiente e capaz de percorrer todas as etapas do processo em um menor período de

tempo, de forma iterativa, sendo necessário possuir uma equipe composta de pessoas de várias áreas desde o início do projeto.

- De uma perspectiva ágil, a comunicação é mais importante do que documentação, preferivelmente comunicação face a face. Reuniões regulares e a colaboração do cliente é a garantia de que o projeto esta caminhando de acordo com as expectativas.
- A cultura pode ser considerada um fator altamente importante, a organização deve então, estar culturalmente preparada para trabalhar de forma iterativa, em um ambiente incerto e com constantes mudanças. A organização deve apoiar e confiar na equipe para realizar o trabalho com pouca ou nenhuma supervisão, buscando integrar pessoas do negócio a trabalhar com os desenvolvedores desde o início do projeto.

5 CONCLUSÃO

É difícil de se apontar uma “receita de bolo” confiável para a escolha de um processo de desenvolvimento de software, ou de se garantir a correta adequação de um método ágil à filosofia da empresa. Entretanto, podemos observar através deste estudo que existem algumas características que viabilizam saber, com antecedência, se uma organização deve ou não adotar um método de desenvolvimento ágil. Além disso, estas características podem indicar mudanças na organização necessárias para esta melhor se enquadrar à filosofia ágil. Pode-se observar também que os métodos ágeis e processos tradicionais possuem diferentes focos. Com isso, baseando-se nas características da organização, do projeto e da equipe, é possível apontar um processo de desenvolvimento que melhor se enquadre para aquele ambiente.

Como trabalho futuro, seria interessante encontrar soluções ou ferramentas que apóiem a adoção de um método ágil pela empresa baseando-se nas características levantadas no Capítulo 4. A idéia da ferramenta seria indicar um método mais adequado para os tipos de ambiente em que ele é aconselhado. Além disso, poderia ser feito um estudo sobre a perda das características ágeis, que costuma ocorrer em projetos com uma grande equipe. Futuros desdobramentos do trabalho poderiam oferecer apoio de ferramentas de gerenciamento de equipes e fragmentação de projetos, tornando os projetos que seguem métodos ágeis mais simples.

REFERÊNCIAS

ABRAHAMSSON, P.; WARSTA, J.; SIPONEN, M.T.; RONKAINEN J. New directions on agile methods: a comparative analysis. 2003. proceedings of ICSE'03, 244-254.

BONA C. Avaliação de processos de software: um estudo de caso em XP e Iconix. 2002 . Disponível em: <http://teses.eps.ufsc.br/defesa/pdf/10816.pdf>. Acesso em: 11 dez. 2011.

BOEHM, B.W.; Turner, R. Balancing Agility and Discipline: a guide for the perplexed. 2003

BORGES, L. M. S.; FALBO R. A. Uma ferramenta de apoio à instanciação de processos de software com gerência de conhecimento, I Concurso de teses e dissertações em qualidade de software, Anais do I Simpósio brasileiro de qualidade de software, 2002.

BROOKS, F. No silver bullet: essence and accidents of software engineering. Proc. IFIP, IEEE CS Press, p. 1069-1076, Abril 1987.

CLEMENTS, Paul; NORTHROP, L. M. Software product lines: Practices and Patterns. Addison-Wesley, 2002.

GOTTERBARN D. UML and agile methods: in support of irresponsible development. In: ACM SIGCSE Bulletin, Column Thinking Professionally, v. 36, n. 2, p. 11-13, jun. 2004.

HIGHSMITH, J. Dinosaurs retiring lifecycle. Revista Software Testing& Quality Engineering. Disponível em: <http://www.adaptivesd.com/articles/Dinosaurs.pdf>. Acessado em: 1 Abril 2012.

HUMPHREY, W. S. A discipline for software engineering. Addison-Wesley. Reading-MA, 1995.

HUMPHREY, W. S. The personal software process. Software Process Newsletter, Technical Council on Software Engineering, IEEE Computer Society, Volume 13, N. 1. Setembro 1994. pp SPN 1-3. Disponível em: <http://www.sei.cmu.edu/pub/documents/articles/pdf/psp.pdf>.

KNIBERG, Henrik. Scrum e XP Direito das Trincheiras. C4Media Inc, 2007. Disponível em <http://www.infoq.com/br/minibooks/scrum-xp-from-the-trenches>, 15 Abril 2012

KOHRELL, D.; WONCH, B. Using RUP to manage small projects and teams. Rational Software White Paper. 2005.

KRUCHTEN, Philippe. Introdução ao RUP – Rational Unified Process: Rio de Janeiro, Editora Ciência Moderna, 2003.

LARMAN, C. Utilizando UML e padrões: uma introdução à análise e ao projeto orientado a objetos e ao Processo Unificado. Trad. Luiz Augusto Meirelles Salgado e João Tortello. 2 ed. Porto Alegre: Bookman, 2004.

MANIFESTO para Desenvolvimento Ágil de Software. Disponível em: <http://agilemanifesto.org/iso/ptbr/> Acesso em: 15 Abril 2012.

Nerur S., Mahapatra R., Mangalaraj G. Challenges of migrating to agile methodologies. Communications of the ACM, v. 48, n. 5, p. 72-78, 2005.

PAIGE R.F.; Wang X.; Stephenson Z.R.; Brooke P.J. Towards an agile process for building software product Lines. XP2006, LNCS 4044, p. 198-199, Springer-Verlag Berlin Heidelberg, 2006.

PAULA, W. P. Engenharia de Software: Fundamentos, métodos e padrões. LTC Editora. Rio de Janeiro - RJ, 2001.

PRESSMAN, R. S. Engenharia de Software. 5º ed. Rio de Janeiro: McGraw-Hill, 2002.

PRESSMAN, R. S. Engenharia de Software. 6º ed. Rio de Janeiro: McGraw-Hill, 2006.

PROBASCO, L. The ten essentials of RUP: the essence of an effective development process. Rational Software White Paper. 2000.

RUP – Rational Unified Process: visão geral. disponível em: <http://www.wthreex.com/rup/>, acesso em: 05 jan. 2010.

SALVIANO, C. F.; TSUKUMO, A. Introdução aos modelos de capacidade de processo do CMMI, MPS.BR, ISO/IEC 15504 e outros. Disponível em: <http://www.simpros.com.br/upload/T1_tutorial.pdf>, Acesso em: 15 Abril 2012.

SOMMERVILLE, I. Engenharia de Software. Editora Addison-Wesley, 2003.

SOMMERVILLE, I. Engenharia de Software. São Paulo: Pearson - Prentice Hall, 2007

TEIXEIRA, J. M. Modelagem de um ambiente de gerenciamento de desenvolvimento distribuído de software baseado em workflow. Pelotas/RS: UFPel. Disponível em: <www.ufpel.tche.br/prg/sisbi/bibct/acervo/info/2007/mono_juliano_teixeira.pdf> Acesso em: 15 Abril 2012.

VASCONCELOS A. Processo de desenvolvimento de software 1, Textos Acadêmicos UFLA/FAEPE, 2005 .

WELLS, Joseph T. Encyclopedia of fraud. Salem: Obsidian, 2002.

XP. Extreme Programming. Disponível em: <http://www.extremeprogramming.org/> Acesso em: 15 Abril 2012.