Nesta seção você encontra artigos voltados para as práticas e métodos ágeis.

A necessidade de ser ágil

Uma análise crítica sobre nove métodos ágeis

Alexandre José Henrique de Oliveira Luna
ajhol@cin.ufpe.br
alexluna@mangve.org
Doutorando em Ciência da Computação pelo CIn-UFPE em Governança em TIC (em andamento). Mestre em Ciência da Computação pelo CIn-UFPE em Gerenciamento de Projetos (2009). MBA em Gestão Estratégica de TIC, FACIPE (2003). Engenheiro Químico pela UFPE (2001). É Analista Consultor da ATI-PE, Gestor de Infraestrutura de TIC da Secretaria Estadual de Educação e Pesquisador do NUTES-HC-UFPE e GP2-CIn-UFPE.

Cleyverson Pereira Costa
cpc@cin.ufpe.br
Mestre em Ciência da Computação pelo CIn-UFPE (2009). Especialista em Engenharia de Software com Ênfase em Teste de Software pelo CIn-UFPE (2007). Graduado em Ciência da Computação (2005). Pesquisador do GP2-CIn-UFPE. Possui experiência na área de testes, tendo atuado como Engenheiro de Testes pelo Motorola Brazil Test Center.

Hermano Perrelli de Moura
hermano@cin.ufpe.br
Possui graduação em Engenharia Eletrônica pela Universidade Federal de Pernambuco (1982), Mestrado em Informática pela Universidade Federal de Pernambuco (1989) e PhD in Computing Science pela University of Glasgow (1993). É certificado PMP (2003) pelo Project Management Institute. Atualmente é Professor Adjunto e Vice-Diretor do Centro de Informática da Universidade Federal de Pernambuco.

De que se trata o artigo?
Este artigo tem por objetivo apresentar uma análise comparativa entre nove métodos ágeis, no sentido de instrumentalizar as equipes e organizações a obterem melhores resultados na aplicação de métodos ágeis em seus projetos.

Para que serve?
Os métodos ágeis estão se tornando uma alternativa concreta para ajudar as equipes de desenvolvimento de software a tornarem-se aptas a responder de forma rápida, flexível e eficaz às constantes mudanças que fazem parte da dinâmica natureza dos negócios das organizações. Possuir uma visão crítica sobre o assunto facilita a aplicação destes princípios e práticas no cotidiano das equipes nas empresas.

Em que situação o tema é útil?
A velocidade de resposta às necessidades do negócio está intimamente ligada à competitividade dos times e organizações. A aplicação de métodos ágeis, dentre outros benefícios, geram transparência com usuários, clientes e acionistas. Fatores como integração das pessoas técnicas e de negócio na obtenção de resultados de melhor qualidade de forma rápida, facilita o alcance da estratégia organizacional, o retorno dos investimentos, dentre outros aspectos essenciais à garantia da sobrevivência institucional. Este tema é essencial para desenvolvedores, líderes de equipes e profissionais de TIC que desejam colaborar para a sustentabilidade de suas organizações.

Muitos autores como Roosmalen e Hoppenbrouwers, Cummins e Sloane et al. têm dito que para sobreviver à voracidade do mercado é necessário agilidade nos negócios, porém qual é o significado de tal termo? Segundo o Gartner Group, "agilidade de negócio" é estar apto a responder rapidamente e eficientemente às mudanças no mundo dos negócios e, transformar essas mudanças em vantagem competitiva é o principal motivo para sua adoção.

Neste contexto, observa-se que cada vez mais organizações estão adotando a abordagem ágil como uma tática de sobrevivência nestes tempos economicamente turbulentos. Isto por sua vez levou a uma série de opiniões interessantes examinando quais atitudes e atributos seus times precisam para serem bem sucedidos. Sob esta ótica a agilidade de negócio, reconhecida como a habilidade para "mudar o sentido do ambiente e responder eficientemente e efetivamente a essa mudança", é importante.

Acrescentar agilidade aos processos de gestão, em sua essência já infere um maior nível de convergência entre as iniciativas em TIC (Tecnologias da Informação e Comunicação) e os objetivos do negócio. Contudo, outros benefícios de uma abordagem ágil no contexto de negócios podem ser identificados, como, por exemplo: melhor time-to-market e aumento da velocidade de tomada de decisão, o que acaba refletindo numa maior competitividade organizacional.

Neste contexto, a chave para realizar a verdadeira agilidade de negócio é encorajar os executivos a pensarem nas mudanças de negócio sem se preocupar com as implicações que as mesmas trarão ao legado de TIC existente na organização. Em outras palavras, a empresa precisa se tornar centrada no negócio e não centrada na TIC.

Quando o negócio se torna centrado em si, a organização se torna hábil a definir, criar e construir novos processos ou funções de negócio. Entretanto, para viabilizar esta visão, é essencial que a TIC cumpra o seu papel de se "elevar" de uma abordagem puramente operacional ou tática, para uma participação mais estratégica, colaborando de modo concreto, inclusive, nas definições dos objetivos de negócio.

Desta forma acredita-se que os métodos ágeis têm muito a contribuir nesta direção através da simplificação das iniciativas da TIC, sensibilização e valorização das pessoas, adoção de uma abordagem iterativa e adaptativa, aplicação prática de seus princípios, valores, práticas e orientações sobre sistematização das iniciativas da gestão em TIC.

Engenharia de Software

Engenharia de Software é uma área do conhecimento voltada para a especificação, desenvolvimento e manutenção de sistemas de software, aplicando tecnologias e práticas da ciência da computação, gerência de projetos e outras disciplinas, objetivando organização, produtividade e qualidade.

Pressman destaca que a Engenharia de software abrange três componentes básicos:

• **Métodos**: proporcionam os detalhes de como construir o software. Englobam tarefas como planejamento e estimativa de projeto, análise de requisitos de software e de sistemas, projeto da estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção;

• **Ferramentas**: existem para sustentar cada um dos métodos. Algumas ferramentas existentes para apoio são as Computer-Aided Software Engineering, conhecidas como ferramentas CASE;

• **Procedimentos**: constituem o elo entre métodos e ferramentas. Definem a sequência em que os métodos são aplicados.

Desde então vem prosperando o aparecimento de diversos métodos, técnicas e ferramentas para aperfeiçoar os processos de desenvolvimento de software em todo o mundo. Mesmo com toda esta evolução, a Engenharia de Software há muito vinha enfrentando problemas relativos a atraso na entrega de projetos, orçamento extrapolado, insatisfação de clientes e usuários, além de conflitos e desgastes entre analistas e clientes. Isso se dava, dentre outros fatores, principalmente em função dos métodos disponíveis para o desenvolvimento de software mostrarem-se pesados e burocráticos, ineficientes e improdutivos.

ID	Princípio
P1	A prioridade é a satisfação do cliente, mediante o rápido e contínuo fornecimento de software que agregue um valor ao negócio.
P2	As mudanças são bem-vindas, mesmo no final do desenvolvimento, principalmente se as alterações darão vantagem competitiva para os nossos clientes.
P3	Fazer entregas frequentes de software que funcionem a partir de um par de semanas a um par de meses, sempre procurando o menor intervalo de tempo entre as entregas.
P4	As pessoas de negócio (executivos) e os desenvolvedores devem trabalhar juntos diariamente e ao longo de todo o projeto.
P5	Construir o projeto em torno de indivíduos motivados. Fornecer todo apoio necessário ao ambiente do projeto e confiar plenamente na equipe.
P6	O diálogo face a face é a mais eficiente e eficaz forma de comunicar as informações dentro da equipe de desenvolvimento.
P7	Software que funciona é a principal medida de progresso.
P8	Os processos ágeis promovem um desenvolvimento sustentável. Os promotores, usuários e desenvolvedores devem ser capazes de manter um ritmo de trabalho constante por tempo indeterminado.
P9	A atenção contínua à qualidade técnica e ao bom design melhora a agilidade.
P10	A simplicidade é essencial. É preciso saber maximizar o trabalho que NÃO deve ser feito.
P11	As melhores arquiteturas, requisitos e desenhos surgem a partir da própria Equipe através de sua pró-atividade e auto-organização (inteligência coletiva e colaborativa1).
P12	Em intervalos regulares, a Equipe deve refletir sobre como se tornar mais eficaz, e ajustar o seu comportamento para alcançar este objetivo.

Tabela 1. Princípios Ágeis. FONTE: (BECK et al., 2001)

O Manifesto Ágil

Sob este contexto e percepção, em 11 de fevereiro de 2001, um grupo de profissionais e pesquisadores de TI se reuniram com a finalidade de criar uma mobilização em torno de uma série de valores e práticas de desenvolvimento de software que eles intitularam de Manifesto for Agile Software Development.

Assim, os dezessete presentes assinaram o seguinte manifesto:

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar:

- *Indivíduos e interação entre eles mais que processos e ferramentas*
- *Software em funcionamento mais que documentação abrangente*
- *Colaboração com o cliente mais que negociação de contratos*
- *Responder a mudanças mais que seguir um plano*

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda."

Este Manifesto também enuncia os doze princípios de um processo ágil, como podem ser vistos na **Tabela 1**.

Métodos Ágeis

Neste contexto e visando a melhores resultados, as empresas de TIC estão adotando metodologias de desenvolvimento de

software mais flexíveis e propícias às frequentes mudanças, além de mais interação durante todo o projeto entre os usuários e o próprio sistema. Estas metodologias são chamadas de ágeis em contraposição às metodologias pesadas que, tradicionalmente, predominaram na área, mas que se mostraram ineficientes e improdutivas (FERREIRA e LIMA, 2006).

Uma premissa fundamental das metodologias ágeis é o reconhecimento da dificuldade do usuário saber de antemão as funcionalidades que gostaria que o sistema tivesse. Por isso, essas metodologias adotam a estratégia de criar condições favoráveis para as interações e as retroalimentações entre usuários e Equipe durante todo o projeto. Com isso, as metodologias ágeis são estruturadas de modo a atender a natureza mutável e dinâmica do processo de construção de software.

As metodologias ágeis propõem que os projetos devam ser conduzidos de forma adaptativa, isto é, feito através de desenvolvimento iterativo e interativo. A ideia central é trabalhar com iterações curtas. Cada iteração entrega ao seu final um produto completo e pronto para ser usado, que contém a implementação de um novo subconjunto de características. O uso de iterações curtas permite aos usuários e clientes fazerem uma avaliação do software logo que uma versão inicial é colocada em produção.

É importante salientar que uma das vantagens das metodologias ágeis em contraposição às metodologias tradicionais é a flexibilidade que estas possuem quando inseridas em ambientes que têm características como: definição dos requisitos com grande volatilidade (mudanças constantes), onde as equipes são pequenas e os prazos são mais curtos, o que por fim caracteriza a necessidade de um desenvolvimento rápido.

Dentre as metodologias ágeis mais difundidas pode-se citar o XP e o SCRUM. Contudo, podemos citar também: XPM – Extremme Project Management, APM – Agile Project Management, FDD - Feature Driven Development, família Crystal, DSDM - Dynamic System Development Method, ASD - Adaptative Software Development, dentre outras.

Nas seções seguintes, como resultado de um processo de revisão sistemática, serão exploradas cada uma destas metodologias e visitados seus princípios, valores e práticas. No entanto, em função dos modelos SCRUM e XP já serem bastante conhecidos, os demais modelos serão abordados com maior ênfase.

eXtreme Programming – XP

O **eXtreme Programming** ou XP é um modelo ágil de desenvolvimento de software criado em 1996 por Kent Bech no Departamento de Computação da montadora de carros Daimler Chrysler. Ele possui muitas diferenças em relação a outros modelos, podendo ser aplicado a projetos de alto risco e com requisitos dinâmicos (vagos ou em constante mudança), conduzidos por equipes de tamanho médio e pequeno.

Como todo método ágil, o XP procura responder com velocidade às mudanças nas especificações do projeto, com base em princípios, valores e práticas bem definidos. Este método enfatiza o desenvolvimento rápido garantindo a satisfação do cliente e cumprindo as estimativas do projeto. O XP baseia-se em cinco valores para guiar o desenvolvimento: Comunicação, Coragem, Feedback, Respeito e Simplicidade. Segundo Beck, o método oferece ainda 12 práticas, a saber: i) Jogo do planejamento; ii) Versões pequenas; iii) Metáfora; iv) Projeto simples; v) Teste; vi) Refatoração; vii) Programação em pares; viii) Propriedade coletiva do código; ix) Integração contínua; x) 40 horas de trabalho semanal; xi) Cliente presente; e xii) Padrões de codificação. A **Figura 1** demonstra uma representação do processo do XP, incorporando as práticas, princípios e valores.

A comunicação é um dos principais valores do XP que visa incentivar uma melhor integração entre clientes e desenvolvedores, encorajando a interação e o relacionamento interpessoal. Segundo Nawrocki et al. (2002), com o incentivo na participação do cliente no projeto e a liberação de versões frequentes, existe uma probabilidade menor de ocorrência de erros, assim como permite ao time a antecipação à solução de muitos problemas.

A questão da manutenção de sistemas produzidos a partir de um projeto XP é bastante questionada quanto à sua eficácia devido a pouca documentação pregada pela metodologia. Neste contexto, Nawrocki relata que as fontes de conhecimento em projetos XP são: o código fonte, os casos de teste e a memória dos programadores. O risco em relação a pouca documentação está na "perda de memória" que ocorre naturalmente quando há saída de desenvolvedores do time, o que se torna um fator mais crítico no caso da manutenção de projetos mais antigos. Nawrocki afirma, ainda, que neste caso, a única base para a manutenção são o código fonte e os casos de teste.

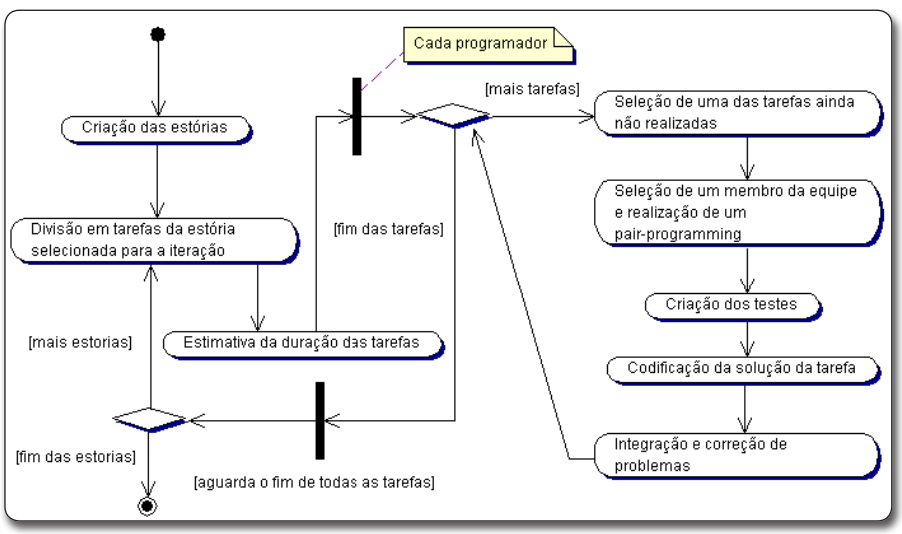

Figura 1. Fluxo de trabalho em um projeto XP. FONTE: (SIQUEIRA, 2002)

SCRUM

O termo **Scrum** vem de um estudo feito por Takeuchi e Nonaka (1986). Como resultado deste estudo, foi percebido

que projetos de curta duração, usando equipes pequenas e multidisciplinares (cross-functional), produzem melhores resultados.

O framework Scrum tem como objetivo, segundo Schwaber, definir um processo para projeto e desenvolvimento de software orientado a objeto, que seja focado nas pessoas e que seja indicado para ambientes em que os requisitos surgem e mudam rapidamente. O Scrum também é considerado um método específico para o gerenciamento do processo de desenvolvimento de software.

Seu ciclo de vida é baseado em três fases principais. A fase de pré-planejamento, desenvolvimento e a fase de pós-planejamento. O pré-planejamento é dividido em duas fases secundárias: a fase de planejamento e a de arquitetura do projeto.

No Scrum, todo o desenvolvimento é feito em iterações: todo o esforço é orientado para que seja apresentado um novo conjunto de funcionalidades ao final de cada iteração, denominada de sprint, para a qual é sugerida uma duração de duas a quatro semanas. A **Figura 2** apresenta, de forma simplificada, o desenvolvimento de um projeto utilizando o Scrum.

O Scrum define três papéis distintos: 1) Equipe – responsável por entregar soluções, geralmente é formada por um grupo pequeno (entre cinco e nove pessoas) e que trabalha de forma auto-gerenciada; 2) Product Owner – responsável pela visão de negócios do projeto, é ele quem define e prioriza o Product Backlog. Geralmente é o papel desempenhado pelo cliente; 3) Scrum Master – é uma mistura de gerente, facilitador e mediador. Seu papel é remover obstáculos da equipe e assegurar que as práticas de Scrum estão sendo executadas com eficiência.

O desenvolvimento do produto, baseado no Scrum inicia com a definição do Backlog pelo Product Owner. Posteriormente o projeto é organizado em Sprints e o Product backlog é dividido em Sprint Backlogs para que os itens selecionados possam ser desenvolvidos. Ao final da Sprint, a Equipe demonstra os resultados para o Product Owner e demais interessados, de forma que os itens do Backlog sejam considerados prontos e então possa se iniciar um novo Sprint.

eXtreme Project Management - XPM

O **XPM**, ou **eXtreme Project Management**, propõe melhorar o gerenciamento de projetos desenvolvidos segundo o paradigma ágil, com ênfase no XP. Este método visa em especial os e-projects, ou seja, projetos de software para os quais o tempo e o custo para tornar o produto disponível no mercado são críticos.

A principal diferença do XPM está na atitude em relação às mudanças. Diferentemente da abordagem tradicional, na qual o planejamento direciona os resultados, no XPM são os resultados que direcionam o planejamento, sendo necessário facilitar a mudança e não desencorajá-la. Segundo BECK, o método é definido por 11 regras, descritas a seguir:

1. A gerência de pessoas e de processos criativos demanda processos de gerenciamento criativos. Tanto gerente quanto a equipe precisam ser criativos no desenvolvimento de um produto inovador, com alto valor para o negócio e maior qualidade;

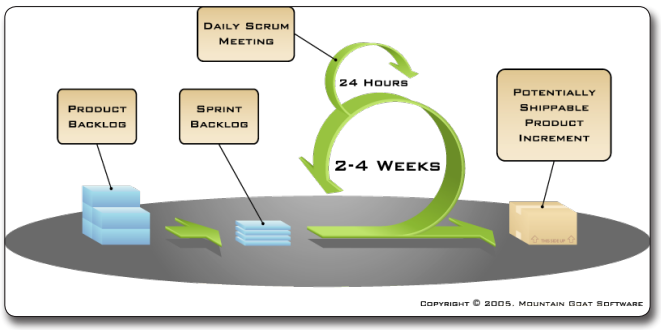

Figura 2. Ciclo de Vida do Scrum. FONTE: (MOUNTAIN GOAT, 2009)

2. O contexto é mais importante do que o conteúdo. O gerente de projetos no XPM deve estar focado no aspecto empresarial do projeto, ou seja, nos objetivos e nos benefícios do projeto, ao invés dos aspectos técnicos do produto ou serviço que está sendo realizado no projeto;

3. Ciclo de vida do projeto inclui período pós-implantação. O que acontece depois que o projeto termina é mais importante do que os problemas que acontecem durante o projeto;

4. O gerente de projetos deve ter o perfil mais facilitador e integrador do que o perfil de gerente. Para aumentar as chances de sucesso de um projeto empresarial, o gerente de projetos deve mudar o foco do planejamento técnico para a facilitação e a integração do processo de planejamento, com a participação efetiva dos stakeholders;

5. Quanto mais tempo o gerente de projeto permanecer com os *stakeholders*, melhor. O XPM está mais relacionado ao contexto do projeto no que diz respeito à disseminação da informação gerencial do negócio, do que com o seu conteúdo no que diz respeito às especificações técnicas e entregas;

***6. Stakeholders* funcionam como Gerente Executivo de Projetos.** No XPM, os stakeholders devem ter as seguintes responsabilidades adicionais: i) participar das sessões de planejamento do projeto que definem escopo, objetivos, stakeholders e benefícios; ii) assistir o Gerente de Projetos nas disputas políticas e de poder na organização que influenciam o projeto; e iii) monitorar indicadores críticos do projeto, de custos e de prazo;

7. Se o sucesso do projeto não foi definido no começo, ele não será alcançado no final. O sucesso em um projeto está geralmente associado a: i) satisfazer os stakeholders; ii) atingir as exigências de escopo; iii) permanecer dentro do orçamento e dos prazos estabelecidos; iv) agregar valor ao negócio; v) assegurar uma boa qualidade ao produto; e vi) deixar os membros da equipe satisfeitos. Os critérios de sucesso devem ser definidos logo de início e acompanhados durante todo o desenvolvimento;

8. Planejamento por cenário ao invés de macroplanejamento. Grande parte dos projetos empresariais possui o nível de incerteza muito alto, assim como o ritmo de mudanças. Por isso, fazer o planejamento detalhado das fases do projeto fornece uma alta probabilidade de retrabalho e constante replanejamento;

9. Lucro ao invés de papelada burocrática. No XPM, a documentação é a mínima necessária para o desenvolvimento e o acompanhamento do projeto. O importante é modelar e

apresentar o valor agregado de acordo com a qualidade requerida e buscar o equilíbrio entre a necessidade de negócio que está sendo atendida e o retorno de investimento que isso traz ao negócio. Mostre os lucros aos stakeholders e nada mais importa;

10. Se o seu projeto não mudou, fique apreensivo. Gerente e equipe devem reunir-se diariamente para avaliar se houve alteração em expectativas de sucesso, escopo, objetivos, riscos, qualidade, stakeholders ou projetos relacionados, bem como verificar se suposições referentes a custo e benefício continuam pertinentes;

11. Em ***e-projects***, um dia é um tempo muito longo. O gerenciamento efetivo de e-projects demanda uma abordagem nova e radical para o gerenciamento de projetos.

Agile Project Management - APM

A abordagem APM - ***Agile Project Management*** foi desenvolvida por um grupo de gerentes de vários projetos XP bem sucedidos da CC Pace Systems. Em sua concepção eles consideraram que a adoção ainda lenta das metodologias ágeis origina-se principalmente da falta de alinhamento entre as suposições fundamentais da gerência tradicional e das metodologias de desenvolvimento ágeis. Também alertaram para a necessidade de mudança em relação a estas suposições, propondo o desenvolvimento de um novo framework para o apoio gerencial ao desenvolvimento ágil.

Na busca deste novo framework, eles passaram a acreditar fortemente na adoção de princípios que explorassem a compreensão do comportamento humano autônomo, adquirida a partir do estudo de sistemas vivos existentes na natureza – como revoadas, cardumes e enxames –, incluindo nas suposições e práticas de gerência a noção de sistemas adaptativos complexos (Complex Adaptive Systems – CAS). Apesar destes sistemas possuírem somente regras e capacidade estratégicas no contexto do próprio sistema (ou grupo em análise), seu comportamento coletivo é caracterizado por uma superposição de ordem, auto-organização e uma inteligência coletiva que é maior que a soma das partes, além de regularmente exibirem uma habilidade notável para se adaptarem a ambientes complexos e dinâmicos.

Por exemplo, em uma equipe XP, os gerentes de projeto também precisam de um conjunto de práticas simples que os guiem, que forneçam um framework dentro do qual possam administrar, e não de um conjunto de instruções rígidas. Seguindo estas práticas, o gerente torna-se um líder com capacidade de adaptação, capaz de fixar uma direção, estabelecer regras simples e geradoras do sistema, bem como encorajar uma constante avaliação (feedback), adaptação e colaboração.

O framework para gerência de projeto ágil baseado em CAS é composto por seis práticas-chave que, juntas, ajudam a administrar equipes de desenvolvimento como sistemas adaptáveis complexos, ao mesmo tempo em que proporcionam liberdade para sobrepor estilos próprios de liderança pessoal. Uma descrição resumida destas práticas é apresentada a seguir:

1. Visão Direcionada – Estabeleça uma visão direcionadora para o projeto e reforce-a continuamente, por meio de palavras e ações. É importante transmitir e manter no time sempre a visão do todo, para que seja mais fácil para cada um compreender, transmitir e mesmo supervisionar a produção das partes e a sua integração na formação do todo. A visão precisa ser sempre difundida, atualizada e preservada de agentes internos ou externos ao time;

2. Trabalho e Colaboração em Equipe – Facilite a colaboração e o trabalho em equipe reforçando relacionamentos. Quando o trabalho conjunto aprimora e reforça os potenciais individuais complementares, os resultados obtidos podem ser excepcionais. Contudo, conseguir que as pessoas trabalhem de forma colaborativa é um desafio e não pode acontecer por "imposição";

3. Regras Simples – Estabeleça e apoie um conjunto de práticas-chave (guias). Estabeleça e fomente um conjunto de práticas simples que possam fornecer suporte para um comportamento complexo, permitindo à equipe trabalhar dentro de uma estrutura flexível;

4. Informação Aberta – Forneça acesso aberto à informação. O compartilhamento da informação e o sentimento de propriedade coletiva potencializam os resultados alcançados pela equipe;

5. Toque leve – Aplique somente o controle suficiente para manter a ordem emergente. O controle inteligente de equipes requer uma sutil combinação de ordem imposta e emergente;

6. Vigilância Ágil – Aplique um contínuo monitoramento, aprendizado e adaptação ao ambiente. O trabalho mais criativo e ágil ocorre no limiar entre a ordem e o caos – imprevisível o bastante para ser desafiador, mas ordenado o suficiente para não sair de controle.

easY Process - YP

O Departamento de Sistemas e Computação da Universidade Federal de Campina Grande (UFCG) criou em maio de 2003 o ***easY Process - YP***, um processo de software mais simplificado que se apoia em práticas do XP, RUP e Agile Modeling.

A necessidade de se criar um novo processo surgiu devido às dificuldades encontradas em se adaptar os processos já existentes para o uso na academia. O Fluxo básico está ilustrado na **Figura 3**. Os tempos apresentados nos retângulos desta figura denotam os tempos estimados pelo YP para o avanço de cada etapa do processo.

A primeira etapa do processo consiste na **definição de papéis.** O YP sugere os seguintes papéis: cliente, usuário, testador, desenvolvedor e gerente; podendo uma mesma pessoa desempenhar mais de um papel dentro do processo, principalmente quando se tratam de equipes de desenvolvimento pequenas. Em seguida deve ser realizada uma conversa com o cliente, onde informações sobre o escopo do problema são capturadas. A partir de então, a equipe encontra-se apta a gerar o documento de visão, que após ser validado pelo cliente, funciona como um acordo de trabalho entre cliente e equipe de desenvolvimento.

Na fase de inicialização o cliente define as User Stories e são elaborados o projeto arquitetural e o modelo lógico de dados – este último apenas se necessário. O cliente deve priorizar as User Stories e a equipe deve fazer uma estimativa inicial do tempo para implementação de cada uma delas. Baseado nessa estimativa pode-se então verificar a viabilidade de

desenvolvimento do projeto no escopo e tempo definidos.

Parte-se então para o Planejamento, fase composta por dois planos, o de release e o de iteração. Ambos possuem tempo fixo com variação de escopo permitida. Tratando-se do ambiente acadêmico são sugeridos três releases, cada um com duas iterações de duas semanas, por semestre letivo. O planejamento de um release só ocorre após o término do anterior, e da mesma forma para as iterações. No planejamento de release alocaram-se as User Stories de acordo com a priorização do cliente. No planejamento de iteração as User Stories alocadas são quebradas em tarefas, e o cliente deve definir os testes de aceitação para cada User Story. Para auxílio na gerência o processo faz uso da Tabela de Alocação de Tarefas (TAT), na qual se especificaram as User Stories envolvidas, tarefas, responsáveis, estimativas de tempo, tempo real consumido e status da tarefa. Estes dois últimos atributos são preenchidos apenas no fechamento da iteração.

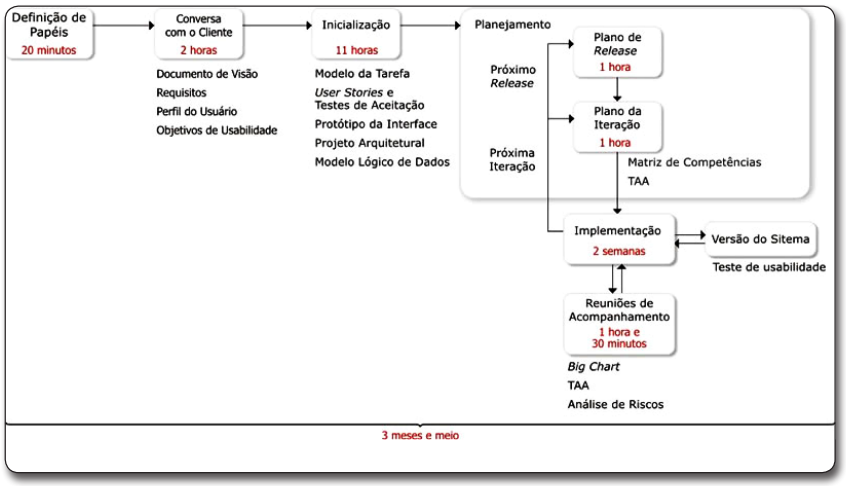

Figura 3. Fluxo do YP. FONTE: (YP, 2006)

Segundo Garcia, algumas características importantes do YP são:

• Participação efetiva do cliente: fator imprescindível para o sucesso do projeto;

• Papéis diferentes desempenhados pela mesma pessoa: necessário quando se trabalha com equipes pequenas;

• Releases e iterações curtas: tratando-se do ambiente acadêmico são sugeridos três releases, cada um com duas iterações de duas semanas;

• Variação no escopo e não no tempo, tanto no release quanto na iteração. Este é um princípio baseado no paradigma ágil que considera fixos o tempo e os recursos e permite variar dentro de uma iteração o escopo;

• Forte enfoque nos testes, em boas práticas de programação, propriedade coletiva de código e refatoramento;

• Acompanhamento do progresso do projeto através de métricas pré-definidas (user stories alcançadas, classes produzidas, testes realizados) reunidas no Big Chart;

• Manutenção do repositório de código com ferramentas de controle de versão.

Feature Driven Development – FDD

Criado por Jeff de Luca e Peter Code, o FDD é um método ágil e adaptável ao sistema. Este método não cobre o processo inteiro de desenvolvimento do software, mas focaliza-o particularmente no projeto e nas fases de construção.

O FDD incorpora o desenvolvimento iterativo e as melhores práticas da modelagem ágil. Os aspectos de qualidade são enfatizados durante todo o processo de desenvolvimento, incluindo entregas frequentes e tangíveis, bem como monitoração do progresso do projeto no período de desenvolvimento.

FDD possui cinco processos sequenciais durante o projeto e o desenvolvimento do sistema, como ilustrado na **Figura 4.** A parte iterativa dos processos de FDD ("projetar por característica" e "construir por característica") suporta o desenvolvimento ágil com adaptações rápidas às mudanças, de acordo com as exigências e as necessidades do negócio. As iterações do projeto e a construção de uma característica (feature) seguem por um período de uma a três semanas de trabalho.

A seguir estão descritos sucintamente os cinco processos ilustrados na **Figura 4**.

• **Processo 1:** "Desenvolver um modelo abrangente" – Os membros de um projeto devem estar cientes do contexto e das exigências do sistema a ser construído logo no início do desenvolvimento do projeto. Isso é alcançado por meio de casos de uso ou especificações funcionais exigidos neste processo;

• **Processo 2:** "Construir uma lista de Características" – A equipe identifica as características, agrupa-as hierarquicamente e atribuem prioridades e tamanho. Entre as tarefas deste processo incluem a formação da equipe que irá projetar a lista de características;

• **Processo 3:** "Planejar por Características" – Um plano de projeto é construído e usado nos processos seguintes, determinando a sequência de desenvolvimento com as prioridades e as datas que cada característica deve ser completada;

• **Processo 4:** "Projetar por Características" – Um pequeno grupo de características é selecionado do conjunto de características. Deste grupo são identificadas as classes que estão envolvidas e os seus respectivos proprietários. Cada característica selecionada irá passar por esta etapa, em que a equipe de características define um diagrama de sequência detalhado para ela. Os proprietários das classes estruturam suas classes e métodos. No final a equipe faz uma inspeção no projeto. Entre as tarefas deste processo incluem a formação da equipe de projeto e a definição de um guia de domínio, a construção do diagrama de sequência, a estruturação das classes e métodos e a inspeção do projeto;

• **Processo 5:** "Construir por Características" – Neste processo são realizados a implementação das classes e métodos, a inspeção do código, os testes de unidade e o desenvolvimento de cada característica ou conjunto delas.

Em relação às práticas definidas no FDD, elas não são extremamente rígidas, pregando a adaptação ao ambiente de desenvolvimento. No entanto, existe um conjunto de práticas que são fundamentais e que definem o FDD:

• **Modelagem em objetos de domínio:** construir um diagrama de classes básico com os objetos de domínio e suas relações, definindo assim uma arquitetura básica para o modelo do sistema;

• **Desenvolvimento por características:** a implementação deve ser orientada pelas características;

• **Autoria individual:** o código é de autoria de um "dono" da classe, o que permite uma maior rapidez na implementação das tarefas associadas;

• **Times da característica:** para a implementação de uma determinada característica, o chefe programador recruta os "donos" das classes que serão usadas. Esse grupo de pessoas é o time da característica;

• **Inspeções:** é o meio através do qual ocorrem as verificações de qualidade do código e do projeto;

• **Integração (*build*) regular:** em um determinado período de tempo fixo devem ser integradas as características já terminadas, permitindo a verificação de erros e também criando uma versão atual que pode ser demonstrada ao cliente;

• **Gerência de configuração:** visa gerenciar todo o ciclo de vida dos itens de configuração do projeto, realizando o controle de versões de todos os artefatos criados;

• **Reportar/Visibilidade dos resultados:** permitir que se conheça o progresso do projeto.

Família Crystal

Criado por Alistair Cockburn, a família de métodos Crystal prioriza a comunicação entre os participantes do projeto e inclui um número diferente de métodos que atendem projetos com características distintas. A família Crystal é formada por: i) Crystal Clear; ii) Crystal Yellow; iii) Crystal Orange; iv) Crystal Red; e v) Crystal Orange/Web. A escolha de um método deve ser baseada no tipo de projeto. A dimensão e o tamanho de um projeto são representados por símbolos, onde cada um denota uma categoria que especifica o tipo de projeto com relação ao tamanho e a complexidade.

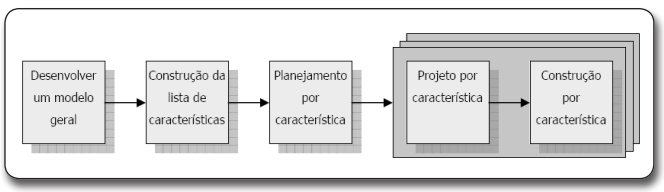

Figura 4. Processos FDD. FONTE: adaptado de (ABRAHAMSSON et al., 2002)

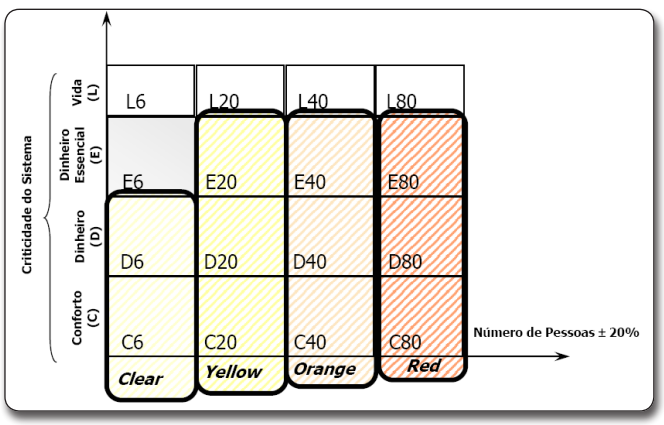

Figura 5. A distribuição dos métodos da família Crystal a partir de duas dimensões. FONTE: Adaptado de (COCKBURN, 2000)

A **Figura 5** apresenta a distribuição dos métodos da família Crystal a partir da análise de duas dimensões: a criticidade do produto a ser construído pelo projeto versus a quantidade de colaboradores envolvidos.

Conforme as cores dos membros da família Crystal se tornam mais escuras, tem-se um maior peso dos métodos, o que é necessário devido à complexidade dos projetos. Esse peso é representado pela quantidade de artefatos e a rigidez da gerência, itens que são absorvidos entre os 13 elementos definidos para cada método: papéis, habilidades, times, técnicas, atividades, processos, artefatos, produtos de trabalho, padrões, ferramentas, personalidades, qualidade e valores da equipe.

As regras, características e valores são comuns em todos os métodos da família Crystal. Segundo Cockburn, dois valores próprios da família Crystal são a alta tolerância e o fato de ser centrada em comunicação e pessoas. A tolerância relaciona-se ao comportamento humano com relação às ferramentas e produtos de trabalho utilizados em um projeto Crystal.

Segundo Cockburn, os princípios do Método da Família Crystal são os seguintes:

• O time pode reduzir o trabalho em produtos intermediários, para produzir código rodando mais frequentemente, através do uso de canais de comunicação mais ricos entre as pessoas, como o contato face-a-face;

• Como cada projeto é diferente e evolui ao longo do tempo, o conjunto de práticas que a equipe adota também deve ser adaptado e evoluído;

• Os gargalos de mudança no software determinam a utilização do trabalho sobreposto (aquele que é realizado por pessoas distintas do time) e apontam os detentores de informação sobre o projeto. Ou seja, fica fácil identificar quais pessoas do time detêm informações essenciais sobre o projeto e gerenciá-los.

As duas regras comuns à Família Crystal, são:

• O projeto precisa usar desenvolvimento incremental, com incrementos de quatro meses ou menos (com forte preferência a incrementos de um a três meses);

• O time precisa realizar oficinas de reflexão pré e pós-incremento (com forte preferência para a realização de oficinas no meio do incremento, também). Em outras palavras, antes, durante e após a finalização de cada iteração.

As duas técnicas base em Crystal são:

• A metodologia é refinada pela técnica: usando entrevistas de projeto e oficinas com a equipe para converter e aperfeiçoar a metodologia de referência em um conjunto de regras práticas para a condução do projeto. Estas regras deverão ser aperfeiçoadas pelo time na medida em que o projeto avança;

• A técnica usada para aplicar as oficinas de reflexão, que devem ser refinadas pelo próprio time.

Ainda de acordo com Cockburn, o time pode substituir as técnicas acima por outras que julgar melhor no apoio para o cumprimento das suas metas.

O cerne da filosofia Crystal é considerar o desenvolvimento de software como um jogo cooperativo de invenção e comunicação, com o objetivo principal de fornecimento útil de software. Duas consequências dessa filosofia são que diferentes projetos precisam ser executados de forma diferente, e a quantidade de modelagem e de comunicação que as pessoas precisam fazer é apenas a quantidade de que necessitam para fazer o "jogo" progredir. Os membros da família Crystal compartilham: i) valores e princípios; e ii) adaptação "on-the-fly", ou seja, ajuste da forma de trabalho durante a execução do projeto.

Uma crítica comum a esta metodologia é o excesso de simplicidade, de forma que para grandes projetos recomenda-se o uso de métodos ágeis com mais recursos, como por exemplo, o ASD (*Adaptative Software Development*).

Dynamic Systems Development Method – DSDM

O *Dynamic Systems Development Method* - DSDM é uma formulação dos métodos RAD (*Rapid Application Development*) organizada por um consórcio de companhias membros que, além de fornecer serviços e treinamentos, também cuida do licenciamento de uso do método.

As ideias principais do DSDM podem ser observadas no conjunto de princípios que foram definidos para nortear o método:

• O envolvimento ativo do usuário é imperativo;
• O time deve ter o poder para tomar decisões;
• O foco é na entrega frequente de produtos;
• O encaixe ao propósito do negócio é o critério essencial para a aceitação das entregas;
• O desenvolvimento iterativo e incremental é necessário para convergir com precisão às soluções do negócio;
• Todas as mudanças durante o desenvolvimento são reversíveis;
• Requisitos são alinhados em um alto nível;
• O teste é integrado por todo o ciclo de vida;
• Uma abordagem colaborativa e cooperativa entre as partes envolvidas é essencial.

Além desses princípios, existem algumas técnicas principais que são usadas durante a execução de um projeto usando DSDM:

• **Time-box:** definição de um período fixo para a execução do projeto, colocando até datas de entrega. Com isso, caso haja alguma funcionalidade que não possa ser implementada durante o período estipulado, ela deve ser feita após o desenvolvimento em si (antes da fase de pós-projeto);
• **MoSCoW:** regra básica para a priorização de requisitos durante o período de desenvolvimento. A ideia fundamental é priorizar e implementar os requisitos que sejam considerados principais, deixando os menos importantes para depois;
• **Modelagem:** não deve ser uma atividade burocrática, sendo usada para prover um melhor entendimento do problema e da solução;
• **Prototipação:** forma de verificar a adequação dos requisitos e facilitar as discussões com o cliente. O protótipo criado deve evoluir juntamente com o projeto;
• **Teste:** essa atividade deve ser executada sistematicamente e de forma contínua durante o projeto;
• **Gerência de configuração:** essencial, visto que os produtos são entregues com uma grande frequência.

Em relação ao processo do DSDM, existem cinco fases básicas, que podem ser vistas na Figura 6, antecedidas por uma fase de pré-projeto e precedidas pelo pós-projeto. No pré-projeto, tem-se como objetivo definir se o projeto deve ou não ser implementado, observando aspectos gerenciais básicos, como questões financeiras e um plano para o estudo de viabilidade. O estudo de viabilidade em si é feito na etapa seguinte, em que se verifica se o DSDM é a solução mais adequada, além das atividades tradicionais em um estudo desse tipo. Na etapa seguinte, de estudo do negócio, são observados "os processos que serão afetados e as suas necessidades de informação" (DSDM, 2003), definindo o escopo do projeto.

Posteriormente é iniciado o desenvolvimento em si, que é executado de forma interativa em cada uma das três fases seguintes: modelagem funcional, projeto e construção e implementação. Como a transição entre essas fases é algo bastante complicado, a decisão de quando e como isso deve acontecer acaba sendo feita de projeto a projeto, podendo haver sobreposição e mescla entre elas. Além disso, a qualquer momento pode haver um refinamento do projeto, fazendo com que se volte a fases anteriores para corrigir problemas, solucionar dúvidas, etc.

Na primeira fase de desenvolvimento, que cuida do modelo funcional, os requisitos (funcionais e não funcionais) são obtidos, montando uma lista de prioridades e colocando-os no protótipo. Em seguida é documentada a maioria dos requisitos dessa forma (visual), através de esboços de tela, ao invés da especificação textual. Na fase de implementação é feita a transição do sistema do ambiente de desenvolvimento para o operacional, cuidando do treinamento e outras tarefas que sejam necessárias.

Ao finalizar as etapas de desenvolvimento com um resultado satisfatório na realização dos requisitos, chega-se a fase de pós-projeto. Nela é feita a manutenção do sistema, realizando as tarefas de alteração praticamente da mesma forma que foi feito o desenvolvimento.

A equipe em um projeto DSDM, segundo Abrahamsson et al. (2002), pode variar entre duas a seis pessoas, podendo existir várias equipes pequenas em um projeto. Em uma equipe de duas pessoas deve existir pelo menos um usuário e um colaborador.

Adaptative Software Development – ASD

Criado por Jim Highsmith, o método *Adaptative Software Development* baseia-se também em práticas derivadas do RAD, orientando o desenvolvimento para aceitar as mudanças. Este método tem seu foco voltado principalmente para resolver problemas no desenvolvimento de sistemas grandes e complexos.

O método incentiva fortemente o desenvolvimento incremental, iterativo e com prototipação constante.

Sob esse panorama, o ASD propõe atualizar o ciclo de desenvolvimento baseado em planejar, projetar e construir, trocando-o por um com as fases de especular, colaborar e aprender. Essa mudança seria necessária devido ao enfoque diferente dos dois ciclos: o primeiro considera a estabilidade no ambiente de negócios, enquanto o segundo foca em ambientes de incerteza e de grande mudança – visão comum a todos os métodos ágeis.

O processo de desenvolvimento é guiado por meio de ciclos, compostos por três fases: especulação, colaboração e aprendizado. Segundo Highsmith, o ASD permite mudanças no projeto, não visualizando as mudanças como um problema, mas sim como uma vantagem. A não resistência a mudanças enfatiza uma característica dos métodos ágeis, que é ser adaptativo.

Na **Figura 7** apresenta-se o ciclo de vida do método ASD. Na fase de especulação é realizado o planejamento do projeto, a fase de colaboração apoia a equipe de trabalho nas mudanças do projeto e a fase de aprendizado representa o conhecimento envolvido no projeto, enfatizando o reconhecimento de erros e mudanças durante o desenvolvimento.

O enfoque no ciclo ASD é mais voltado aos resultados com qualidade, do que às tarefas a serem desempenhadas. As tarefas representam as atividades existentes para o desenvolvimento das funcionalidades.

Na **Figura 7** apresentam-se as fases do ciclo de vida do método ASD. A primeira fase define a atividade para a iniciação do projeto e é a atividade responsável pelo ciclo de desenvolvimento adaptável (fase de especulação). A fase de colaboração possui uma atividade utilizada para o desenvolvimento do componente (módulo) e a fase de aprendizado engloba as atividades de revisão, qualidade e liberação de versões. Na fase de aprendizado existe um retorno que vai da atividade de revisão de qualidade e volta para a atividade de ciclo adaptável da fase de especulação, o que representa o ciclo de aprendizado do método. Os ciclos duram em média de quatro a oito semanas.

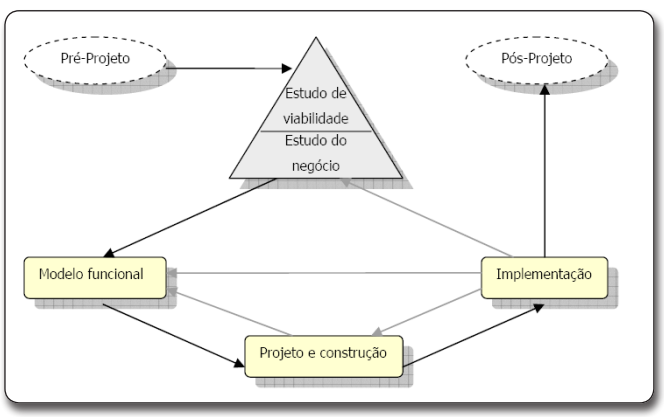

Figura 6. O Processo DSDM. FONTE: Adaptado de (DSDM, 2003)

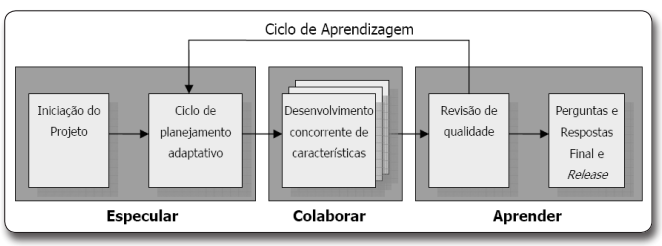

Figura 7. Ciclo de vida do ASD. FONTE: Adaptado de (HIGHSMITH, 2002b).

O ciclo ASD possui seis características básicas que devem ser seguidas em um projeto. Essas características são apresentadas a seguir:

- **Conduzido à missão** - as atividades em cada ciclo de desenvolvimento devem ser ajustadas de acordo com o projeto;
- **Baseado em módulos** – as atividades não devem ser orientadas a tarefas, mas de preferência ao desenvolvimento do software, construindo pequenas versões em pequenos períodos;
- **Iterativo** – o desenvolvimento deve ser bem compreendido e bem definido;
- **Quadro do tempo** – a ambiguidade em projetos complexos com relação a prazos pode ser evitada com o uso de históricos de projetos anteriores. A gerência do projeto força os participantes a tomarem decisões inevitáveis no início do projeto;
- **Dirigido aos riscos** – as mudanças são frequentes no desenvolvimento do software e devem ser avaliadas constantemente para sua adaptação;
- **Tolerante a mudanças** – as mudanças que proporcionam risco ao projeto devem começar o mais rápido possível.

O ASD propõe poucas práticas para o trabalho de desenvolvimento do software, sendo basicamente três: desenvolvimento iterativo, planejamento (baseado em módulos) e revisões de grupo voltadas para o cliente. Neste método existem poucas práticas para o dia-a-dia do trabalho do time. Basicamente é sugerido: i) desenvolvimento iterativo; ii) desenvolvimento e planejamento baseado em funcionalidades (e componentes); iii) avaliações de grupo com foco no cliente. De fato, talvez o mais significativo problema com ASD seja que suas práticas se tornam difíceis de identificar e deixam muitos detalhes em aberto.

Análise Comparativa

Cada método ágil possui características que influenciam no funcionamento e no desenvolvimento do projeto de software. Algumas características podem ser encontradas em vários métodos e outras são específicas de cada um.

Desenvolvimento dirigido por planejamento	Abordagem Ágil
Desenvolvedor com habilidades variadas	Desenvolvedor ágil, educado, disposto e colaborador
Níveis de capacidade do cliente podem variar	Clientes mais representativos e autorizados
Confiança em conhecimento documentado, explícito	Confiança em conhecimento interpessoal, tácito
Requisitos conhecidos e altamente estáveis	Requisitos emergentes, mudança rápida
Projetado para requisitos atuais e previsíveis	Projetado para requisitos atuais
Retrabalho e reestruturações de código são caros	Retrabalho e reestruturações de código são baratos
Equipes e produtos maiores	Equipes e produtos menores
Premia a garantia da qualidade obtida	Premia o valor rápido obtido

Tabela 2. Comparação entre os pressupostos do desenvolvimento dirigido por planejamento e da abordagem ágil. FONTE: Adaptado de (MAGALHÃES et al., 2005)

Métodos	Pontos chave	Principais Características	Limitações/ Falhas
XP	Desenvolvimento dirigido pelo cliente, equipes pequenas e versões frequentes.	Refatoração do software melhora o desempenho e é responsável pelas mudanças.	Pouca atenção no uso de prática de gerenciamento.
SCRUM	Enxuto, auto-organizável, ciclo de desenvolvimento de até 15 dias.	Visão do produto bem definida e repetível.	Carente de testes de integração e omisso em relação a aspectos de implementação.
XPM	Complementa a carência de abordagem gerencial do XP. Recomendada aplicação conjunta.	Voltado para e-Projects. Os resultados direcionam o planejamento e as mudanças são encorajadas.	As práticas são muito subjetivas. É necessário alto grau de maturidade do Gerente de Projeto para colocá-las em prática.
APM	Acredita fortemente na adoção de princípios que explorem a compreensão do comportamento humano autônomo.	Considera que o comportamento coletivo é caracterizado por uma superposição de ordem, auto-organização e uma inteligência coletiva que é maior que a soma das partes.	Requer muita experiência do Gerente na liderança de pessoas para se extrair o melhor resultado do método. Não recomendado para Equipes pouco maduras.
YP	Processo simplificado que se apoia em práticas do XP, RUP e Agile Modeling.	Visa aplicação em projetos acadêmicos, ou comerciais de pequeno ou médio porte.	Recomendado para projetos de escopo pequeno, que possam ser concluídos em até quatro meses.
FDD	Formado por cinco processos e iterações curtas.	Método simples, desenvolvimento por características e modelagem de objeto.	Foco restrito ao projeto e à implementação.
Crystal	Vários métodos com características diferentes.	Capacidade de selecionar o método mais adaptável ao projeto.	Dificuldade no uso de estimativas.
DSDM	Uso do RAD, equipe com autonomia para tomar decisões.	Utiliza a prototipação e possui vários papéis (responsáveis) para execução de uma atividade no método.	Somente os membros da equipe têm acesso aos procedimentos do método, não envolvendo o cliente.
ASD	Foca no ciclo adaptável, colaborativo e no desenvolvimento iterativo.	Oriundo da filosofia de Sistemas Adaptativos.	Baseia-se mais nos conceitos e na cultura ágil do que em práticas ágeis.

Tabela 3. Comparação entre os métodos ágeis revisados. FONTE: Adaptado de (ABRAHAMSSON et al., 2002)

Na **Tabela 2** apresenta-se um estudo comparativo entre os pressupostos do desenvolvimento dirigido por planejamento (tradicional) e da abordagem ágil realizado por Magalhães et al. (2005).

Na **Tabela 3** apresenta-se um estudo comparativo dos métodos ágeis, partindo do estudo realizado por Abrahamsson et al. (2002) e complementado por este trabalho, apontando os pontos chaves, as principais características e as falhas entre os métodos aqui apresentados.

A contribuição acrescentada ao trabalho comparativo de Abrahamsson et al. (2002) diz respeito mais precisamente à complementação da análise com o acréscimo dos métodos: XPM, APM e YP, bem como a uma revisão da análise original sob a ótica dos objetivos deste artigo.

Com base na **Tabela 2** ficam perceptíveis as vantagens resultantes da abordagem ágil no que diz respeito à capacidade do time em responder de forma rápida e precisa às mudanças naturais que fazem parte do ambiente dos negócios, onde estão inseridos os projetos. Contudo, para adoção de uma abordagem ágil, as organizações precisam estar dispostas a mudar a sua percepção em relação a seus clientes, a reavaliar a forma como encaram seus projetos e a assumir alguns riscos. Além disso, os times também precisam estar dispostos a aprender e amadurecer durante o processo, refinando suas práticas e aumentando o grau de integração e comunicação com o cliente.

Conclusões

Neste artigo foram apresentadas reflexões sobre porque adotar uma "abordagem ágil" para os processos de negócio das organizações. Foram apresentadas algumas definições sobre processo de desenvolvimento de software, assim como a origem da formação da Aliança Ágil, através do Manifesto Ágil, seus princípios e valores comuns a todos os métodos ágeis.

No decorrer deste artigo foram apresentados nove métodos ágeis, abordando seu processo de desenvolvimento e as práticas existentes em cada um. Além disso, foi apresentado também um estudo comparativo dos processos de desenvolvimento ágil abordados. A revisão sistemática realizada neste artigo a respeito de metodologias ágeis está diretamente associada com a necessidade de identificação dos princípios, valores e boas práticas ágeis que possam ser adequados e aplicados aos negócios de cada organização.

Espera-se que os resultados da análise comparativa, bem como o conjunto de conhecimentos explorados neste artigo sirvam de base para a formação de uma visão crítica sobre a aplicação de métodos ágeis na indústria de software, assim como no comportamento das equipes envolvidas neste contexto. Possibilitando, desta forma, a geração de um senso analítico sobre em que circunstâncias cada um dos métodos aqui explorados melhor se adéquam à natureza dos projetos em andamento em nossas organizações. E permitindo, inclusive, uma reflexão sobre a pertinência de utilização combinada de alguns métodos ágeis, em função das características das variáveis envolvidas em cada projeto, como: cliente, equipe, restrições e premissas, dentre outras.

Referências

ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. (2002). Agile Software Development Methods. Review and analysis. ESPOO (Technical Research Centre of Finland)' 2002. VTT Publications n. 478, 112p, 2002.

APM (2003). CC PACE Systems. "Agile Project Management Explained – White paper". Disponível em: < http://www.ccpace.com/Resources/documents/AgileProjectManagement.pdf >.

BECK, K ; FOWLER, M. (2000). Planning extreme programming. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000. Disponível em: < http://www.mip.sdu.dk/~brianj/Extreme%20Programming%20Explained%20-%20Kent%20Beck%3B%20Addison-Wesley,%201999.pdf >.

BECK, KENT et al. (2001). Agile Manifesto, 2001. Disponível em: <http://www.agilemanifesto.org>.

BECK K. (1999). Programação Extrema Explicada. Bookman, 1999.

COCKBURN, A. (2000). Agile Software Development Draft version: 3b. Highsmith Series Editors, 2000. Disponível em: <http://zsiie.icis.pcz.pl/ksiazki/Agile%20Software%20Development.pdf>.

CUMMINS, F.A (2008). Building the Agile Enterprise: With SOA, BPM and MBM, 2008. Paperback, 336 pages, publication date: SEP-2008. ISBN-13: 978-0-12-374445-6. Disponível em: < http://books.google.com/books?hl=pt-BR e lr= e id=S6bla9Oy7SYC e oi=fnd e pg=PR13 e dq=%22agile+governance%22 e ots=k05jBK84BQ e sig=Yy6lpvSQ9TNKELMr3Ohv3dR_7UA >.

DSDM (2003). DYNAMIC SYSTEMS DEVELOPMENT METHOD LTD. DSDM Consortium, 2003. Site do consórcio responsável pelo DSDM e onde estão disponíveis diversas informações sobre o método. Disponível em: <http://www.dsdm.org/>.

FERREIRA, RB; LIMA, FPA (2006). Metodologias Ágeis: Um Novo Paradigma de Desenvolvimento de Software. II Workshop Um Olhar Sociotécnico sobre a Engenharia de Software – WOSES, 2006, Vila Velha - ES. Disponível em: <http://www.cos.ufrj.br/~handrade/woses/woses2006/pdfs/10-Artigo10WOSES-2006.pdf>.

GARCIA, F. P. et al. (2004). easYProcess: Um Processo de Desenvolvimento para Uso no Ambiente Acadêmico. XII WEI-Workshop de Educação em Computação. Campina Grande: UFCG, 2004. Disponível em: < http://www.dsc.ufcg.edu.br/~yp/Download/ArtigoYPWEI.pdf>.

HIGHSMITH, J. (2000). Retiring Lifecycle Dinosaurs. Software Testing e Quality Engineering 2, n.4, July/August 2000.

HIGHSMITH, J. (2002a). Agile Software Development Ecosystems. Addison Wesley, 2002.

HIGHSMITH, J. (2002b).What Is Agile Software Development? Agile Software Development. CROSSTALK. The Journal of Defense Software Engineering. Outubro, 2002. p. 4-9.

JACOBSEN, CATRINE M. (2001). "XPM – from idea to realization - critical approach to the concept of XPM". Disponível em: <http://www.glyn.dk/download/synopsisXPM.pdf >.

LOJKINE, J. (1996). "A revolução informacional". São Paulo, Editora Cortez, 1996.

LUFTMAN, J.N.; LEWIS, P.R. e OLDACH, S.H. (1993). "Transforming The Enterprise: The Alignment Of Business And Information Technology Strategies". IBM Systems Journal, v.32, n.1, p.198-221, 1993.

LUNA, Alexandre J. H. DE O.; COSTA, Cleyverson P. and NOVAES, Magdala A. (2010). Desenvolvimento Distribuído de uma Aplicação de Telessaúde com a Metodologia Ágil SCRUM. In Revista Científica Tecnologus 4, no. 1: 6. Disponível em: <http://www.unibratec.com.br/revistacientifica/n4_artigos.html>.

MAGALHÃES, ANA L. C. DE C.; ROUILLER, ANA C.; VASCONCELOS, ALEXANDRE M. L. (2005). O Gerenciamento de Projetos de Software Desenvolvidos à Luz das Metodologias Ágeis: Uma Visão Comparativa. Revista ProQuality – Qualidade na Produção de Software – vol. 1, n. 1 – Lavras, Universidade Federal de Lavras, maio de 2005. Disponível em: < http://www.proqualiti.org.br/revista/revista_ProQualiti_maio2005.pdf#page=29 >.

NAWROCKI, J.; JASINSKI, M.; WALTER, B.; WOJCIECHOWSKI. (2002). Extreme Programming Modified: Embrace Requirements Engineering Practices. In: RE' 2002, International Conference on Requirements Engineering, IEEE, 8 p., 2002.

OLIVEIRA, E. S. (2003). "Uso de Metodologias Ágeis no Desenvolvimento de Software". Monografia apresentada no Programa de Pós-Graduação em Engenharia de Software da UFMG, 2003.
PALMER, S. R.; FELSING, J. M. A (2002). Practical Guide to Feature-Driven Development. Prentice Hall, 2002.

PRESSMAN, Roger S. (2006). Engenharia de Software: Uma abordagem prática. Mc Graw Hill - 6ªedição, 2006.

ROOSMALEN, MW VAN e HOPPENBROUWERS, S (2008). Supporting Corporate Governance with Enterprise Architecture and Business Rule Management: A Synthesis of Stability and Agility. Proceedings of ReMoD, 2008. Disponível em: < http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-342/paper2.pdf >.

SCHWABER, Ken e BEEDLE, Mike (2002). Agile Software Development with SCRUM. Prentice Hall, PTR Upper Saddle River, NJ, USA, 2002.

SCOTT, DONNA (2000). Operation Zero Downtime, a Gartner Group Report, Donna Scott, 2000. Disponível em: <http://www.gartner.com/>.

SLOANE, E; BECK, R e METZGER, S (2008). AGSOA - Agile Governance for Service Oriented Architecture (SOA) Systems: A Methodology to Deliver 21st Century Military Net-Centric Systems of Systems. Systems Conference, 2008 2nd Annual IEEE, 2008. Disponível em: < http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4518995 >.

SOARES, M. S. (2004). Comparação entre Metodologias Ágeis e Tradicionais para Desenvolvimento de Software. INFOCOMP Journal of Computer Science, Vol. 3, n.º 2, p. 8-13, 2004.

SOMMERVILLE, IAN (2007). Engenharia de Software. Pearson Education - 8ª Edição, São Paulo, 2007.

STAPLETON, JENNIFER (1997). DSDM: The Method in Practice. Pages: 192. Medium: Hardcover, 1997. ISBN:0201178893.

TAKEUCHI, H. AND I. NONAKA (1986). The New New Product Development Game. Harvard Business Review, 1986 (January-February).

THOMSETT, ROB (2002). Radical Project Management. Prentice Hall, 2002. ISBN-13: 978-0-13-009486-5. Disponível em: <http://my.safaribooksonline.com/0130094862>.

YP (2006). easY Process. Disponível em: <http://www.dsc.ufcg.edu.br/~yp/>.