Universidade Estadual de Campinas – UNICAMP

Faculdade de Tecnologia – FT

Métodos Ágeis

Paula L.O. Libardi,
Vladimir Barbosa

LIMEIRA - SP

JUNHO/2010

Universidade Estadual de Campinas – UNICAMP

Faculdade de Tecnologia - FT

Métodos Ágeis

Autores: **Paula L O Libardi,**

Vladimir Barbosa

Orientador: **Prof. Dr. Marcos A Francisco Borges**

Trabalho apresentado a Faculdade de Tecnologia – FT, como requisito para a conclusão da disciplina FT-027 Tópicos em Computação no curso de pós graduação.

LIMEIRA - SP

2010

AGRADECIMENTO

Agradecemos ao Professor Dr. Marcos Augusto Francisco Borges, pelo compartilhamento de informações e experiências profissionais que muito contribuiram para o aprimoramento dos conhecimentos sobre o tema.

Think you know where you are on your well-formed plan
and discover that you are very wrong, very much later.

Ken Schwaber

RESUMO

Este trabalho apresenta um estudo sobre metodologias ágeis de desenvolvimento. É um breve estudo das metodologias ágeis em geral e uma especificação detalhada dos papeis, regras e práticas da metodologia *Scrum* e da linguagem *XP*. Além disso, apresenta uma comparação dos processos propostos pelos métodos ageis XP, SCRUM, FDD e ASD

ABSTRACT

This work presents a study about software agile methodology. It's a brief study of general agile methodology and a detailed specification of roles, practices and rules of Scrum methodology and the XP language. Also, this document presents a comparison of the processes proposed by the agile methods XP, SCRUM, FDD and ASD

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

PO – *Product Owner*

XP – *Extreme Programming*

SUMÁRIO

1. INTRODUÇÃO

A indústria de desenvolvimento de software tem se tornado uma das mais importantes indústrias do nosso tempo. Empregando milhares de trabalhadores em todos os países do mundo, essa indústria cria alguns dos produtos mais essenciais que nós utilizamos para manter o nosso estilo de vida [1].

Utilizado para o controle de produção, controle da segurança dos veículos que nós dirigimos, automação dos nossos negócios e em muitas outras áreas de nossas vidas, o software tem se tornado um dos maiores valores de propriedade intelectual do mundo.

Na intensa competitividade no ambiente de desenvolvimento de software, o que separará os melhores dos piores, os vencedores dos perdedores? Esta resposta é simples: a sua habilidade de criar e entregar mais rapidamente os produtos de software com mais qualidade e que melhor reflitam as reais necessidades dos clientes.

Os métodos ágeis utilizados para desenvolvimento de software é um diferencial que promete aumentar a satisfação do cliente agregando maior valor ao produto final, produzindo software alta qualidade e acelerando os prazos de desenvolvimento de projetos.

Este trabalho visa descrever metodologias de desenvolvimento ágil de software, que está sendo amplamente utilizada em renomadas empresas da área como Google, Yahoo, Globo, Ci&T, ...

2. METODOLOGIA ÁGIL DE DESENVOLVIMENTO

O desenvolvimento de software precisa ser reconhecido como um processo imprevisível e complicado. Reconhecer que um software nunca foi construído da mesma forma, com a mesma equipe, sob as mesmas circunstâncias antes é a grande mudança do pensamento tradicional de desenvolvimento de software. Mas, o mais importante é reconhecê-lo como um processo empírico: que aceita a imprevisibilidade e tem mecanismos de ação corretiva.

Uma característica das metodologias ágeis é que elas são adaptativas ao invés de serem preditivas. Dessa forma, elas se adaptam a novos fatores durante o desenvolvimento do projeto, ao invés de tentar analisar previamente tudo o que pode ou não acontecer no decorrer do desenvolvimento. Essa análise prévia é sempre difícil e apresenta alto custo, além de se tornar um problema quando for necessário fazer alterações nos planejamentos. O problema não é a mudança em si, mesmo porque ela ocorrerá de qualquer forma. O problema é como receber, avaliar e responder às mudanças. Numa metodologia clássica pode acontecer de que um software seja construído por inteiro e depois se descubra que ele não serve mais para o propósito que foi desenvolvido porque as regras mudaram e as adaptações tornem-se complexa demais para que valha a pena desenvolve-las. As metodologias ágeis trabalham com constante *feedback*, o que permite adaptar rapidamente a eventuais mudanças nos requisitos. Alterações essas que são, muitas vezes, críticas nas metodologias tradicionais, que não apresentam meios de se adaptar rapidamente às mudanças. Um outro ponto positivo das metodologias ágeis são as entregas constantes de partes operacionais do software. Desta forma, o cliente não precisa esperar muito para ver o software funcionando e notar que não era bem isso que ele esperava [2].

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

2.1 História

O termo “Metodologias Ágeis” tornou-se popular em 2001 quando um grupo de dezessete especialistas em processos de desenvolvimento de software decidiu se reunir nos EUA, para discutir maneiras de melhorar o desempenho de seus projetos.

Embora cada envolvido tivesse suas próprias práticas e teorias sobre como fazer um projeto de software ter sucesso, utilizando métodos como *Scrum*, *Extreme Programming* (XP) e outros, cada um com as suas particularidades, todos concordavam que, em suas experiências prévias, um pequeno conjunto de princípios sempre parecia ter sido respeitado quando os projetos davam certo.

Foi então criada a Aliança Ágil e o estabelecimento do Manifesto Ágil (conforme anexo 1) [3] contento os conceitos e princípios comuns compartilhados por todos esses métodos. Desde então o termo Desenvolvimento Ágil passou a descrever abordagens de desenvolvimento que seguissem estes conceitos que implicam em valorizar:

- Indivíduos e interação entre eles são 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.

O Manifesto Ágil não rejeita os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária quando comparado com os indivíduos e interações, com o software funcionando, com a colaboração com o cliente e as respostas rápidas a mudanças e alterações. [2]

2.2 Analisando o Manifesto Ágil [1]

É muito importante entender que os conceitos do manifesto ágil definem preferências e não alternativas no desenvolvimento de software, encorajando a focar a atenção em certos conceitos sem eliminar outros. Assim para seguir os conceitos ágeis deve-se valorizar mais a certas coisas do que a outras.

1. **Indivíduos e interação entre eles mais que processos e ferramentas**: os softwares não são construídos por uma única pessoa, eles são construídos por uma equipe, então elas precisam trabalhar juntas (incluindo programadores, testers, projetistas e também o cliente). Processos e ferramentas são importantes, mas não são tão importantes quanto trabalhar juntos.
2. **Software em funcionamento mais que documentação abrangente**: a documentação deve existir para ajudar pessoas a entender como o sistema foi construído, mas é muito mais fácil entender como o funcionamento vendo o sistema funcionar do que através de diagramas que descrevem o funcionamento ou abstraem o uso.
3. **Colaboração com o cliente mais que negociação de contratos**: somente o cliente pode dizer o que ele espera do software e normalmente eles não sabem explicar exatamente o que eles esperam e ainda, eles mudam de idéia ao longo do tempo e conforme eles vêm o software funcionando. Ter um contrato é importante para definir as responsabilidades e direitos mas não deve substituir a comunicação. Trabalhos desenvolvidos com sucesso têm constante comunicação com o cliente para entender suas necessidades e ajuda-los a descobrir a melhor forma de expressa-las.
4. **Responder a mudanças mais que seguir um plano**: mudanças é uma realidade no ambiente de negócios e elas acontecem por inúmeras razões: as regras e leis sofrem alterações, as pessoas mudam de idéia e a tecnologia evolui. O software precisa refletir essas mudanças. Um projeto de software certamente deve ter um plano mas ele deve ser flexível o suficiente para comportar as mudanças quando elas aparecerem, senão ele se torna irrelevante.

[1] Baseado em [10]

3. SCRUM

Scrum trabalha com a complexidade de desenvolvimento de softwares através do controle de inspeção, adaptação e visibilidade de requisitos de um processo empírico fazendo uso de uma série de regras e práticas, onde controle não significa controle para criar o que foi previsto e sim controlar o processo para orientar o trabalho em direção a um produto com o maior valor agregado possível [4].

Para atingir esses objetivos o *Scrum* emprega um estrutura iterativa e incremental da seguinte maneira: no início de cada iteração, a equipe analisa o que deve ser feito e então seleciona aquilo que acreditam poder se tornar um incremento de valor ao produto ao final da iteração. A equipe então faz o seu melhor para realizar o desenvolvimento daquela iteração e ao final apresenta o incremento de funcionalidade construído para que os *stakeholders* possam verificar e requisitar alterações no momento apropriado.

O coração do *Scrum* é a iteração [4]. A cada iteração a equipe analisa os requisitos, a tecnologia e suas habilidades e então se dividem para construir e entregar o melhor software possível adaptando-se diariamente conforme surjam as complexidades, dificuldades e surpresas.

3.1 *Sprint* – Ciclo de desenvolvimento *Scrum*

No *Scrum*, um projeto se inicia com uma visão simples do produto que será desenvolvido. A visão pode ser vaga a principio e ir tornando-se clara aos poucos. O *Product Owner* (PO) então, transforma essa visão em uma lista de requisitos funcionais e não-funcionais que quando forem desenvolvidos reflitam essa visão. Essa lista, chamada de *Product Backlog* é priorizada pelo PO de forma que os itens que gerem maior valor ao produto tenham maior prioridade.

O ponto de partida é dividir o *Product Backlog* em *releases* e é esperado que o conteúdo, a prioridade e agrupamento do *Product Backlog* sofram mudanças a partir do momento que o projeto começa. Essas mudanças refletem mudanças nas regras e requisitos de negócios e no quão rapidamente a equipe pode transformá-lo em produto [4].

Todo o trabalho é feito em *Sprints* que são iterações de 2 a 4 semanas. *Scrum* exige que o *Scrum Team* desenvolva um incremento de produto a cada *Sprint*. Esse incremento de produto precisa ser potencialmente entregável para o PO escolhe-lo para ser desenvolvido. Cada incremento deve ser bem estruturado, codificado e testado.

Cada *Sprint* inicia-se com uma reunião chamada *Sprint Planning Meeting* na qual o PO e a equipe decidem o que será desenvolvido neste *Sprint.* Nesta reunião o PO apresenta os itens de maior prioridade e o *Scrum team* seleciona os itens que serão entregues ao final do *Sprint* e os dividem em tarefas que compõem então o *Sprint Backlog*. Nessa reunião fica determinado o que fazer no *Sprint*. A equipe se compromete a executar essas tarefas no *Sprint* e o *Product Owner* se compromete a não trazer novos requisitos para a equipe durante o *Sprint.* Requisitos podem mudar (e mudanças são encorajadas), mas apenas fora do *Sprint*. Uma vez que a equipe comece a trabalhar em um Sprint, ela permanece concentrada no objetivo traçado para o *Sprint* e novos requisitos não são aceitos. [6]

Todos os dias o *Scrum Team* se reúne numa reunião de 15 minutos numa reunião chamada *Daily Scrum* para sincronizar o trabalho da equipe toda e informar o *Scrum Master* de eventuais impedimentos no trabalho.

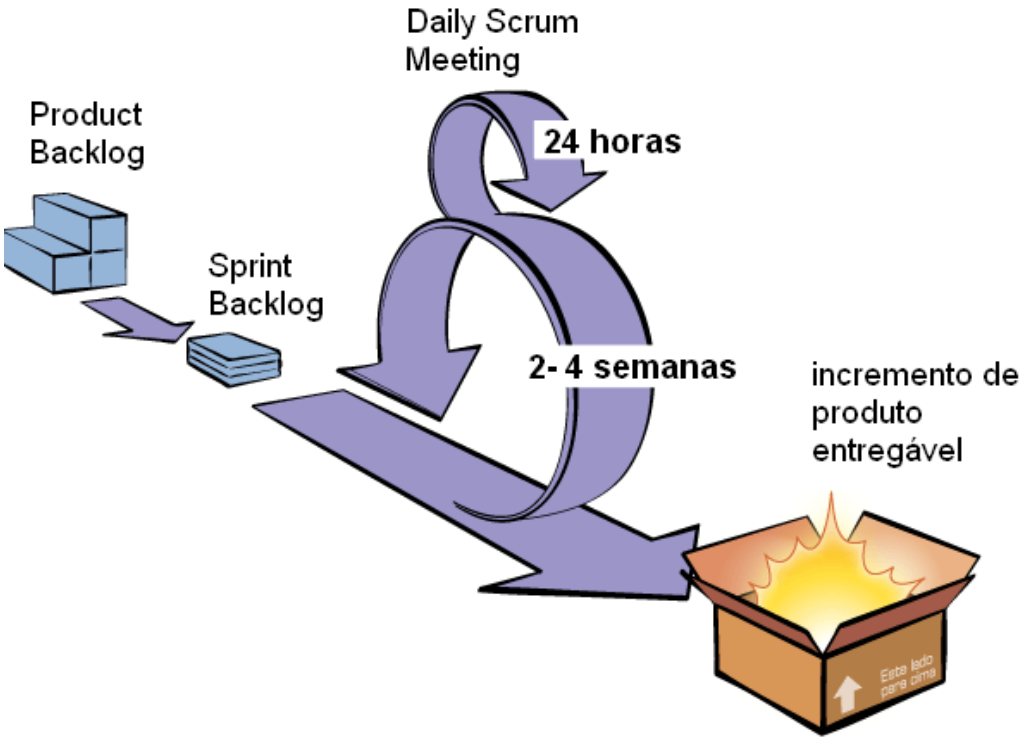

Figura 1 – *Sprint*

No final do *Sprint* ocorre a *Sprint Review Meeting*, reunião na qual o *Scrum Team* apresenta para o PO o que foi desenvolvido durante o *Sprint*.

Depois da *Sprint Review Meeting* e antes do próximo *Sprint* o *Scrum Master* se reúne com o *Scrum Team* numa última reunião: a *Retrospective Meeting*. O *Scrum Master* encoraja o *Scrum Team* a revisar as práticas do *Scrum* e refletir sobre o que precisa ser feito para melhorar no próximo *Sprint*.

Juntas, a Sprint Planning Meeting, a Daily Scrum Meeting, a Sprint Review Meeting e a Sprint Retrospective Meeting implementam as práticas de inspeção e adaptação empíricas. [4]

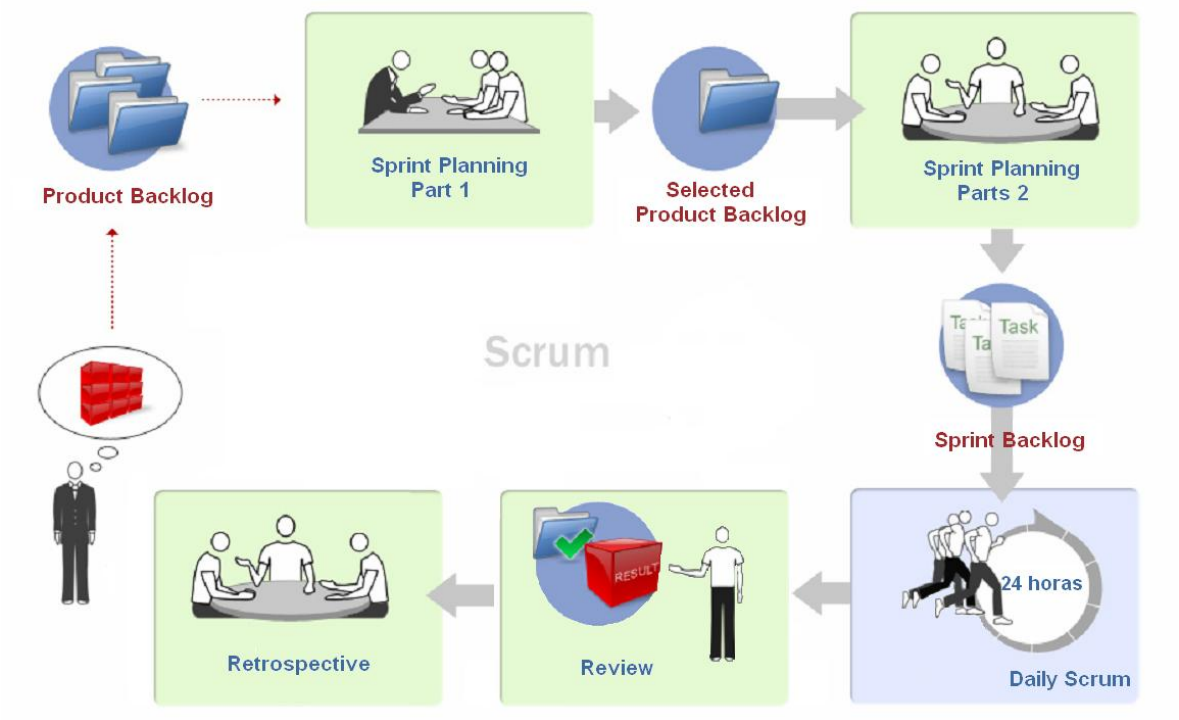

Figura 2 – Ciclo *Scrum* (adaptada de [7])

3.2 Papéis *Scrum*

Scrum implementa sua estrutura iterativa e incremental através de três papéis: o *Product Owner*, o *Team*, e o *ScrumMaster*. Toda responsabilidade no projeto é dividida entre esses três papéis. [4]

3.2.1 *Scrum Master*

O *Scrum Master* é responsável pelo processo *Scrum*, por ensiná-lo a todas as pessoas envolvidas no projeto, por implementá-lo, fazendo dele uma cultura na organização e ainda por garantir que toda a equipe siga as regras e as práticas do *Scrum*.[4]

Ele também protege a equipe assegurando que ela não se comprometa excessivamente com relação àquilo que é capaz de realizar durante um *Sprint*.

O Scrum Master atua como facilitador na Daily Scrum Meeting e torna-se responsável por remover quaisquer obstáculos que sejam levantados pela equipe durante essas reuniões. [6]

O papel de *Scrum Master* pode ser exercido por qualquer pessoa da equipe, entretanto é tipicamente exercido por um gerente de projeto ou um líder técnico.

3.2.2 *Scrum Team*

O *Scrum Team* é a equipe de desenvolvimento.

Um *Scrum Team* típico tem de 6 a 10 pessoas auto-organizáveis, auto-gerenciáveis e multifuncional[6]. Nela, não existe necessariamente uma divisão funcional através de papéis tradicionais, tais como programador, designer, analista de testes ou arquiteto. Todos no projeto trabalham juntos e são responsáveis por completar o conjunto de trabalho com o qual se comprometem a cada iteração. A equipe não pode ser alterada durante o *Sprint*.

Quando é necessário uma equipe maior no *Scrum* é usando o conceito de *Scrum of Scrums*. Cada *Scrum Team* trabalha normalmente, mas cada equipe também contribui com uma pessoa que deverá freqüentar o *Scrum of Scrums Meeting* para coordenar o trabalho de múltiplas equipes *Scrum*. Esses encontros são análogos aos *Daily Scrum*, mas não acontecem necessariamente todos os dias. Fazer essa reunião duas ou três vezes por semana tende a ser suficiente na maioria das organizações. [6]

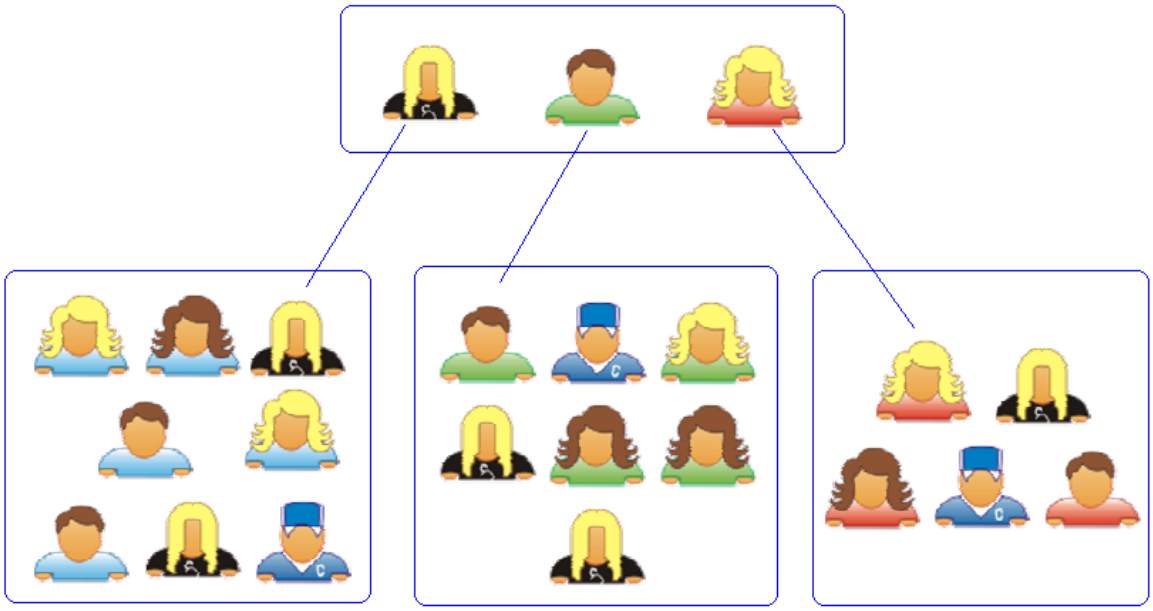

Figura 3 – *Scrum of Scrums*

3.2.3 *Product Owner* (PO)

O *Product Owner* é responsável por representar os interesses dos *stakeholders* no projeto. O *Product Owner* é a pessoa que define todos os itens de requisito do projeto numa lista chamada *Product Backlog*. Utilizando essa lista ele é responsável por garantir que as funcionalidades que agreguem maior valor ao produto sejam desenvolvidas primeiro. Isto é feito através da freqüente priorização do *Product Backlog* para que os itens de maior valor agregado sejam entregues a cada iteração. [4]

3.3 Artefatos

3.3.1 *Product Backlog*

O *Product Backlog* é uma lista contendo todas as funcionalidades desejadas para um produto. O conteúdo desta lista é definido e priorizado pelo *Product Owner*. O *Product Backlog* é totalmente dinâmico, ele é modificado toda vez que se identifica algo que o produto precisa para ser mais apropriado, competitivo ou proveitoso [4], por isso ele nunca está completo, ele sempre contém os requisitos mais conhecidos e melhores entendidos.

Um exemplo de um Product Backlog pode ser visto na tabela 1. Note que os itens são escritos da seguinte maneira: “como um <sujeito> eu gostaria de <ação>”, esse *template* é chamado de *user story* e é utilizado para descrever os itens do backlog.

Backlog item	Estimativa
Como um visitante do site eu gostaria de mandar um e-mail de contato	8
Como um cliente eu gostaria de buscar por um produto	11
Como um cliente eu gostaria de colocar um produto no carrinho de compras	15
Como um cliente eu gostaria de pagar a compra com o cartão de crédito	38
...	...
...	...

Tabela 1 – Exemplo de *Product Backlog*

3.3.2 *Sprint Baclkog*

O *Sprint Backlog* é uma lista de tarefas que o *Scrum Team* se compromete a fazer em um *Sprint* como um potencial incremento de produto entregável[4]. Os itens do *Sprint Backlog* são extraídos do *Product Backlog*, pela equipe, com base nas prioridades definidas pelo *Product Owner* e a percepção da equipe sobre o tempo que será necessário para completar as várias funcionalidades. A quantidade de itens do *Product Backlog* que serão trazidos para o *Sprint Backlog* é definida pelo *Scrum Team* que se compromete a implementá-los durante o *Sprint*.

Os itens do *Sprint Backlog* são divididos em tarefas com detalhes suficientes para que possam ser executadas em até 16 horas [4].

Um exemplo de Sprint Backlog é mostrado na Tabela 2.

Tarefas	Responsável	Estimativa	Status	Dia 1	Dia 2	Dia 3	Dia 4
Criar serviço de email	Paula	5	Complete	5	2	0	0
Implementar envio de email de contato	Marcos	2	In progress	2	2	1	0
Testar o envio de e-mail de contato	Mirian	1	Not Started	1	1	0	0
Criar classe de produto		2	Not Started	2	2	2	2
Criar serviço de busca		4	Not Started	4	4	4	4
...							

Tabela 2 – Exemplo de *Sprint Backlog*

Durante um *Sprint*, o *Scrum Master* mantém o *Sprint Backlog* atualizando-o para refletir que tarefas são completadas e quanto tempo a equipe acredita que será necessário para completar aquelas que ainda não estão prontas. A estimativa do trabalho que ainda resta a ser feito no Sprint é calculada diariamente e colocada em um gráfico, resultando em um Sprint Burndown Chart.[6]

3.3.3 *Burn Down Chart*

O monitoramento do progresso do projeto é realizado através de dois gráficos principais: *Product Burndown Chart* e *Sprint Burndown Chart*. Estes gráficos mostram ao longo do tempo a quantidade de trabalho que ainda resta ser feito, sendo um excelente mecanismo para visualizar a correlação entre a quantidade de trabalho que falta ser feita (em qualquer ponto) e o progresso do time do projeto em reduzir este trabalho [8].

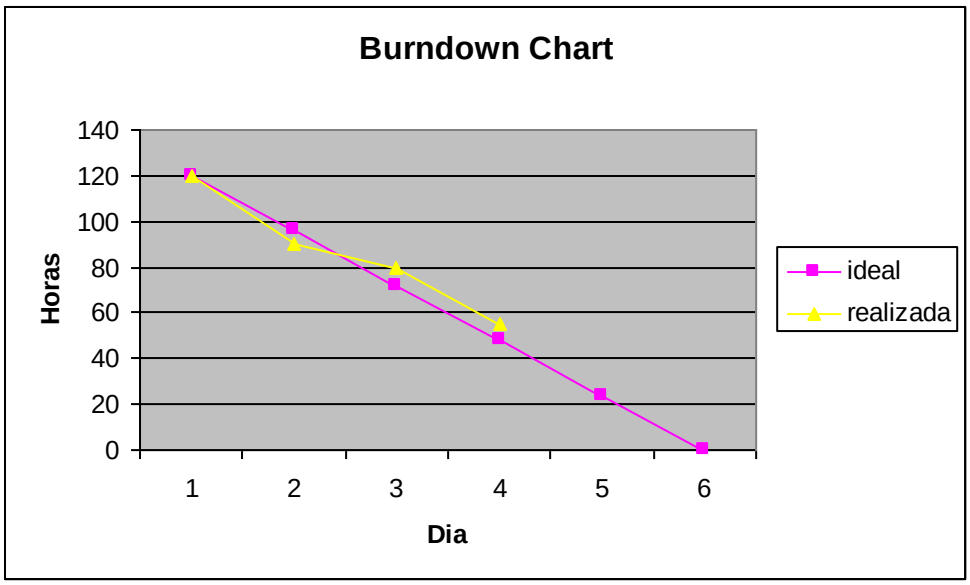

Figura 3 – *Burndown Chart*

3.4 Cerimônias

3.4.1 *Sprint Planning Meeting*

A *Sprint Planning Meeting* é uma reunião na qual estão presentes o PO, o *Scrum Master* e o *Scrum Team*. Durante esta reunião o PO descreve as funcionalidades de maior prioridade para a equipe [6].

Essa reunião, que deve ser de o horas, é dividida em duas partes [4]:

- nas primeiras 4 horas o PO apresenta os itens de maior prioridade do *Product Backlog* para a equipe. O *Scrum Team* faz perguntas para entender melhor as funcionalidades e ser capaz de quebrar as funcionalidades em tarefas técnicas que irão dar origem ao *Sprint Backlog* [6] e então definem colaborativamente o que poderá entrar no desenvolvimento do próximo *Sprint*, considerando o tamanho da equipe, a quantidade de horas disponíveis e a produtividade do *Scrum Team* [5].

- durante as próximas 4 horas o *Scrum Team* planeja seu trabalho, definindo o *Sprint Backlog*, que são as tarefas necessárias para implementar as funcionalidades selecionadas no *Product Backlog* [5].

3.4.2 *Daily Scrum Meeting*

A cada dia do *Sprint* o *Scrum Master* realiza uma reunião de 15 minutos com o *Scrum Team*. Essa reunião, chamada *Daily Scrum Meeting*, tem como objetivo disseminar informações sobre o estado do projeto [5].

As *Daily Scrum Meetings* devem ser realizadas no mesmo lugar, na mesma hora do dia. Idealmente são realizados na parte da manhã, para ajudar a estabelecer as prioridades do novo dia de trabalho [6]. Embora qualquer pessoa possa participar da reunião, somente os membros do *Scrum Team* estão autorizados a falar [5].

Cada membro do *Scrum Team* deve então responder cada uma destas três perguntas:

- O que você fez ontem?
- O que você fará hoje?
- Há algum impedimento no seu caminho?

Concentrando-se no que cada pessoa fez ontem e no que ela irá fazer hoje, a equipe ganha uma excelente compreensão sobre que trabalho foi feito e que trabalho ainda precisa ser feito. A *Daily Scrum Meeting* não é uma reunião de *status* na qual um chefe fica coletando informações sobre quem está atrasado. Ao invés disso, é uma reunião na qual os membros da equipe assumem compromissos perante os demais.

Os impedimentos identificados na *Daily Scrum Meeting* devem ser tratados pelo *Scrum Master* o mais rapidamente possível.

O Daily Scrum não deve ser usado como uma reunião para resolução de problemas. Questões levantadas devem ser levadas para fora da reunião e normalmente tratadas por um grupo menor de pessoas que tenham a ver diretamente com o problema ou possam contribuir para solucioná-lo.

3.4.3 *Sprint Review Meeting*

No final do *Sprint* a *Sprint Review Meeting* é realizada. Essa reunião é planejada para ser realizada em no máximo 4 horas [5]. Nesta reunião o *Scrum Team* mostra o que foi desenvolvido durante o *Sprint*. Tipicamente, isso tem o formato de uma demonstração das novas funcionalidades.

Durante a *Sprint Review Meeting*, o projeto é avaliado em relação aos objetivos do *Sprint*, determinados durante a *Sprint Planning Meeting*. Idealmente, a equipe completou cada um dos itens do *Sprint Backlog*. [6]

3.4.4 *Sprint Retrospective Meeting*

A *Sprint Retrospective Meeting* é uma reunião de 3 horas [4] que ocorre ao final de um *Sprint* depois da *Sprint Review Meeting* e serve para identificar o que funcionou bem, o que pode ser melhorado e que ações serão tomadas para melhorar.

4. COMPARAÇAO ENTRE OS MÉTODOS ÁGEIS

Algumas outras metodologias ágeis de desenvolvimento de software são:

- *Extreme Programming* (XP)
- *Feature Driven Development* (FDD)
- *Adaptive Software Development* (ASD)

4.1 *Extreme Programming* – XP

A metodologia *Extreme Programming* (XP) enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As regras, práticas e valores da XP proporcionam um agradável ambiente de desenvolvimento de software para os seus seguidores, que são conduzidos por quatro valores: comunicação, simplicidade, *feedback* e coragem. A finalidade do princípio de comunicação é manter o melhor relacionamento possível entre clientes e desenvolvedores, preferindo conversas pessoais a outros meios de comunicação. A comunicação entre os desenvolvedores e o gerente do projeto também é encorajada. A forma de comunicação é um fator chave na XP: procura-se o máximo possível comunicar-se pessoalmente, evitando se o uso de telefone e o envio de mensagens por correio eletrônico. A simplicidade visa permitir a criação de código simples que não deve possuir funções desnecessárias. Por código simples entende-se implementar o software com o menor número possível de classes e métodos. Outra idéia importante da simplicidade é procurar implementar apenas requisitos atuais, evitando-se adicionar funcionalidades que podem ser importantes no futuro. A aposta da XP é que é melhor fazer algo simples hoje e pagar um pouco mais amanhã para fazer modificações necessárias do que implementar algo complicado hoje que talvez não venha a ser usado, sempre considerando que requisitos são mutáveis [2].

Uma característica que podemos destacar nesta metodologia é a programação em pares.

4.2 CRITÉRIOS UTILIZADOS

Os critérios adotados para servirem de base para a identificação das atividades propostas pelos Métodos Ágeis são as atividades sugeridas pelo Desenvolvimento Incremental. Sommerville (2003) afirma que os métodos ágeis são fundamentados no Desenvolvimento Incremental. E conforme puderam ser observado, as atividades sugeridas durante o seu processo de desenvolvimento são bastante semelhantes aos Princípios Ágeis.

No Desenvolvimento Incremental (Figura 4) os clientes inicialmente identificam, em um esboço, os requisitos do sistema e selecionam quais são os mais e os menos importantes. Em seguida é definida uma série de iterações de entrega, onde em cada uma é fornecido um subconjunto de funcionalidades executáveis, dependendo das suas prioridades.

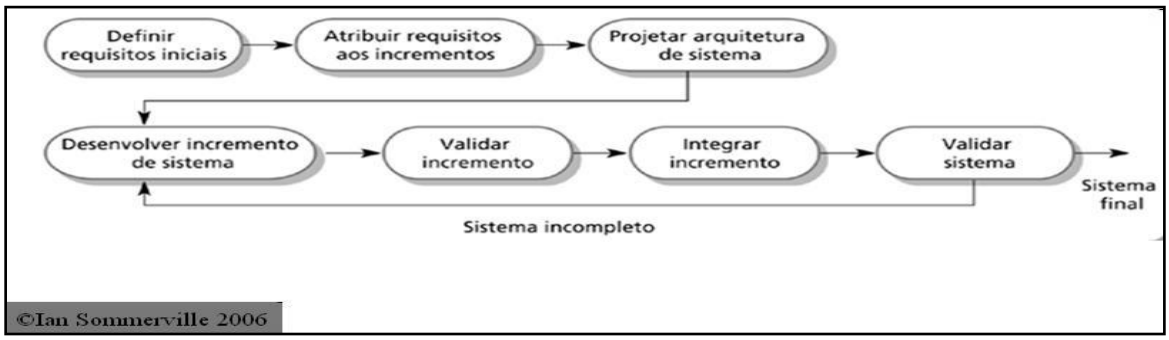

Figura 4 – Desenvolvimento Incremental
Fonte: Adaptado de Sommerville (2003)

4.3 Análise Comparativa

Para facilitar o entedimento da análise comprativa, cada atividade do desenvolvimento incremental e um resumo da atividade correspondente em cada um dos métodos analisados estão elencados na tabela abaixo

Métodos / Fases	XP	SCRUM	FDD	ASD
Requisitos Iniciais	User stories escrita pelo cliente	Os requisitos são listados originando o Product Backlog	Os requisitos são listados, definidos e documentados através de casos de uso (diagramas de classe e sequência)	Sessões de JAD com a presença de representantes dos clientes
Atribuir Requisitos aos incrementos	Definição das "User Stories" que serão desenvolvidas a cada iteração	Os requisitos definidos no Product Backlog são alocados às Sprints durante a Reunião de Planejamento da Sprint	Requisitos agrupados e priorizados conforme importância e dependência	É definido a quantidade de iterações e requisitos que serão implementados em cada uma delas
Projetar Arquitetura do Sistema	Paralelo ao desenvolvimento da User Stories	Projeto geral baseado no Product Log	Diagrama de Classes para representar a arquitetura do sistema	Nenhuma atividade relacionada ao Projeto de Arquitetura.
Desenvolver Incremento de Sistema	Implementação das user stories que fazem parte da iteração corrente por dupla de programadores	Implementação dos requistos contemplados no Sprint Backlog Reuniões diárias de 15 – 30 minutos	Acorre após análise da documentação existente, criação de diagrama de sequência e revisão do modelo gerado durante levantamento de requisitos e projeto de arquitetura	Requisitos desenvolvidos dentro de suas respectivas iterações e simultaneamente

Métodos / Fases	XP	SCRUM	FDD	ASD
Validar Incremento	Programadores: testes de unidade Cliente: testes de aceitação Ambos os testes são escritos antes da codificação e executados após	Acontece ao final da Sprint	Testes efetuados pelos programadores ao final de cada iteração	Grupo de clientes são definidos para revisar e testar a aplicação. Programadores verificam a qualidade do código.
Integrar Incremento	Código é integrado a medida que vai sendo desenvolvido	Acontece ao final de cada Sprint	Ocorre no final de cada iteração após os testes e inspeções	Nenhuma atividade sugerida
Validar Sistema	Sistema é validado pelo Cliente	Sistema é validado no último dia de cada Sprint (Revisão da Sprint)	Validação acontece através das inspeções e dos testes de integração	Nenhuma atividade específica é sugerida
Entrega Final	Cliente Satisfeito	Todos os items do Product Backlog desenvolvidos	Todos os requisitos devem passar por todas as fases de Projeto e construção	Todos os requisitos desenvolvidos

5. CONCLUSÃO

A intensiva competitividade na área de desenvolvimento de software faz com que as empresas busquem sempre o aperfeiçoamento de seus serviços para poder vencer a concorrência. Prazo e qualidade, além é claro de melhor aceitação e adaptação a mudanças são importantes diferenciais que podem ser atingidos utilizando-se metodologias ágeis de desenvolvimento. Embora não seja a solução para todos os problemas, a metodologia ágil mostra uma maneira de trabalhar bastante organizada e iterativa, podendo inclusive contribuir para um ambiente de trabalho mais amigável, portanto é uma boa opção para se obter os diferencias desejados.

No que diz respeito a análise comparativa de métodos ageis, podemos concluir que existe bastante semelhanças entre si, entretanto, algumas particularidades foram notadas como dupla de programadores, user stories e escrita dos testes antes da implementação no XP, reuniões de planejamento, diárias e de revisão no SCRUM, inspeções de código no FDD e sessões de JAD no ASD.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] LEFFINGWELL, Dean and MUIRHEAD, Dave, **Tactical Management of Agile Development: Achieving Competitive Advantage**. 2004. Boulder, Colorado

[2] SOARES, Michel dos Santos, **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software.** Unipac - Universidade Presidente Antônio Carlos, Faculdade de Tecnologia e Ciências de Conselheiro Lafaiete

[3] Agile Manifesto, Disponível em http://agilemanifesto.org/,

[4] SCHWABER , Ken, **What Is Scrum?**

[5] www.scrumalliance.org

[6] www.improveit.com.br

[7] GLOGER, Boris, **Scrum Checklists.**

[8] Ana Sofia Cysneiros Marçal et all, **Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI**

[9] Bob Galen Spring, **SCRUM Product Ownership: From the Inside Out**. 2009. v1.4

[10] AMBLEW, Scott W., **Examining the Agile Manifesto**. 2009.

[11] Fagundes, Priscila Basto; Santos, Sandro da Silva dos e Deters, Janice Inês**, Comparação entre os processos dos métodos ágeis, E-Tech**: Atualidades Tecnológicas para Competitividade Industrial, Florianópolis, v. 1, n. 1, p. 37-46, 1º. sem., 2008.

ANEXO 1 - Manifesto for Agile Software Development

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on
the right, we value the items on the left more.

ANEXO 2 - Principles behind the Agile Manifesto

Principles behind the Agile Manifesto

We follow these principles:
Our highest priority is to satisfy the customer
through early and continuous delivery
of valuable software.
Welcome changing requirements, even late in
development. Agile processes harness change for
the customer's competitive advantage.
Deliver working software frequently, from a
couple of weeks to a couple of months, with a
preference to the shorter timescale.
Business people and developers must work
together daily throughout the project.
Build projects around motivated individuals.
Give them the environment and support they need,
and trust them to get the job done.
The most efficient and effective method of
conveying information to and within a development
team is face-to-face conversation.
Working software is the primary measure of progress.
Agile processes promote sustainable development.
The sponsors, developers, and users should be able
to maintain a constant pace indefinitely.
Continuous attention to technical excellence
and good design enhances agility.
Simplicity--the art of maximizing the amount
of work not done--is essential.
The best architectures, requirements, and designs
emerge from self-organizing teams.

At regular intervals, the team reflects on how
to become more effective, then tunes and adjusts
its behavior accordingly.