Levantamento de requisitos no desenvolvimento ágil de software

Ricardo Augusto Ribeiro de Mendonça

Coordenação de Pós-Graduação Lato Sensu
Pontifícia Universidade Católica de Goiás (PUC Goiás)
Goiânia – GO – Brasil

ricardoaugusto@gmail.com

Abstract. *This paper presents a study of techniques for requirements gathering and demonstrates how to use them in agile software development methodologies. First, some techniques will be presented to raise requirements, then the key will be known agile methodologies and how the requirements are raised in each.*

Resumo. *Este artigo apresenta um estudo de técnicas de levantamento de requisitos e demonstra como utilizá-las em metodologias de desenvolvimento ágil de software. Primeiro, será apresentado algumas técnicas para levantar requisitos, em seguida, será conhecido as principais metodologias ágeis e a forma como os requisitos são levantados em cada uma delas.*

1. Levantamento de requisitos

O levantamento de requisitos desempenha um papel importante na construção de um sistema de informação, pois é o início para toda a atividade de desenvolvimento de software. É onde o analista faz as primeiras reuniões com os clientes e/ou usuários para conhecer as funcionalidades do sistema que será desenvolvido.

Algumas das razões para o baixo grau de satisfação dos usuários estão na fase de levantamento de requisitos do projeto, onde não é utilizada uma técnica adequada para extrair os requisitos do sistema, além disso, a falha do analista em não descrever os requisitos de modo claro, sem ambiguidades, conciso e consistente com todos os aspectos significativos do sistema proposto [Pompilho 1995].

As técnicas de levantamento de requisitos possuem um conceito próprio e podem ser utilizadas em conjunto pelo analista. A seguir serão apresentadas de forma resumida algumas dessas técnicas.

1.1. Entrevistas

A entrevista é uma das técnicas tradicionais mais simples de utilizar e que produz bons resultados na fase inicial de obtenção de dados. Convém que o entrevistador dê margem ao entrevistado para expor as suas ideias. É necessário ter um plano de entrevista para que não haja dispersão do assunto principal e a entrevista fique longa, deixando o entrevistado cansado e não produzindo bons resultados.

Existem dois tipos de entrevistas:

- **Entrevista fechada**, onde o engenheiro de requisitos tem um conjunto pré-definido de perguntas e está à procura de respostas.
- **Entrevista aberta**, sem perguntas pré-definidas do engenheiro de requisitos, onde há uma discussão de forma aberta com os interessados sobre o que eles esperam do sistema.

Na verdade, muitas vezes não há limite claro entre os dois tipos de entrevistas. Você começa com algumas questões que são discutidas e isso leva a novas questões [Sommerville 1997]. A vantagem de entrevistas é que elas ajudam o desenvolvedor a obter uma rica coleção de informações. Sua desvantagem é que esta quantidade de dados qualitativos pode ser difícil de analisar e poderá haver informações conflitantes das diferentes partes interessadas.

1.2. Questionários

Os questionários são indicados, por exemplo, quando há diversos grupos de usuários que podem estar em diversos locais diferentes. Neste caso, elaboram-se pesquisas específicas de acompanhamento com usuários selecionados, pois não seria prático entrevistar todas as pessoas em todos os locais.

Existem vários tipos de questionários que podem ser utilizados. Entre estes podemos listar: múltipla escolha, lista de verificação e questões com espaços em branco. O questionário deve ser desenvolvido de forma a minimizar o tempo gasto em sua resposta.

Na fase de preparação do questionário deve ser indicado o tipo de informação que se deseja obter. Assim que os requisitos forem definidos o analista deve elaborar o questionário com questões de forma simples, clara e concisa, deixar espaço suficiente para as repostas que forem descritivas e agrupar as questões de tópicos específicos em um conjunto com um título especial. O questionário deve ser acompanhado por uma carta explicativa, redigida por um alto executivo, para enfatizar a importância dessa pesquisa para a organização.

Deve ser desenvolvido um controle que identifique todas as pessoas que receberão os questionários. A distribuição deve ocorrer junto com instruções detalhadas sobre como preenchê-lo e ser indicado claramente o prazo para devolução do questionário. Ao analisar as respostas dos participantes é feito uma consolidação das informações fornecidas no questionário, documentando as principais descobertas e enviando uma cópia com estas informações para o participante como forma de consideração pelo tempo dedicado a pesquisa.

1.3. Brainstorming

Brainstorming é uma técnica para geração de ideias. Ela consiste em uma ou várias reuniões que permitem que as pessoas sugiram e explorem ideias.

Brainstorming contém duas fases - a fase de geração, onde as ideias são coletadas, e a fase de avaliação, onde as ideias coletadas são discutidas. Na fase de geração, as ideias não devem ser criticadas nem avaliadas. Cada ideia pode levar a novas ideias. A técnica de brainstorming leva a uma melhor compreensão do problema para todos e um sentimento de que todos cooperaram para atingir o objetivo.

1.4. Joint Application Design (JAD)

A metodologia JAD foi desenvolvida pela IBM para acelerar o desenvolvimento de sistemas de informação e está embasada em dinâmicas de grupo acompanhadas de planejamento, estruturação e sistematização de reuniões.

O JAD facilita a criação de uma visão compartilhada do que o produto de software deve ser. Através da sua utilização os desenvolvedores ajudam os usuários a formular problemas e explorar soluções. Dessa forma, os usuários ganham um sentimento de envolvimento, posse e responsabilidade com o sucesso do produto.

A técnica JAD tem quatro princípios básicos:

- **Dinâmica de grupo**: são realizadas reuniões com um líder experiente, analista, usuários e gerentes, para despertar a força e criatividade dos participantes. O resultado final será a determinação dos objetivos e requisitos do sistema;
- **Uso de técnicas visuais**: para aumentar a comunicação e o entendimento;
- **Manutenção do processo organizado e racional**: o JAD emprega a análise top down e atividades bem definidas. Possibilita assim, a garantia de uma análise completa reduzindo as chances de falhas ou lacunas no projeto e cada nível de detalhe recebe a devida atenção;
- **Utilização de documentação padrão**: preenchida e assinada por todos os participantes. Este documento garante a qualidade esperada do projeto e promove a confiança dos participantes.

A técnica JAD é composta de duas etapas principais: planejamento, que tem por objetivo elicitar e especificar os requisitos; e projeto, em que se lida com o projeto de software.

Cada etapa consiste em três fases: adaptação, sessão e finalização. A fase de adaptação consiste na preparação para a sessão, ou seja, organizar a equipe, adaptar o processo JAD ao produto a ser construído e preparar o material. Na fase de sessão é realizado um ou mais encontros estruturados, envolvendo desenvolvedores e usuários onde os requisitos são desenvolvidos e documentados. A fase de finalização tem por objetivo converter a informação da fase de sessão em sua forma final (um documento de especificação de requisitos).

Há seis tipos de participantes, embora nem todos participem de todas as fases:

- **Líder da sessão**: é responsável pelo sucesso do esforço, sendo o facilitador dos encontros. Deve ser competente, com bom relacionamento pessoal e qualidades gerenciais de liderança;
- **Engenheiro de requisitos**: é o participante diretamente responsável pela produção dos documentos de saída das sessões JAD. Deve ser um desenvolvedor experiente para entender as questões técnicas e detalhes que são discutidos durante as sessões e ter habilidade de organizar ideias e expressá-las com clareza;
- **Executor**: é o responsável pelo produto sendo construído. Tem que fornecer aos participantes uma visão geral dos pontos estratégicos do produto de software a

ser construído e tomar as decisões executivas, tais como alocação de recursos, que podem afetar os requisitos e o projeto do novo produto;

- **Representantes dos usuários**: são as pessoas na empresa que irão utilizar o produto de software. Durante a extração de requisitos, os representantes são frequentemente gerentes ou pessoas chave dentro da empresa que tem uma visão melhor do todo e de como ele será usado;
- **Representantes de produtos de software**: são pessoas que estão bastante familiarizadas com as capacidades dos produtos de software. Seu papel é ajudar os usuários a entender o que é razoável ou possível que o novo produto faça;
- **Especialista**: é a pessoa que pode fornecer informações detalhadas sobre um tópico específico.

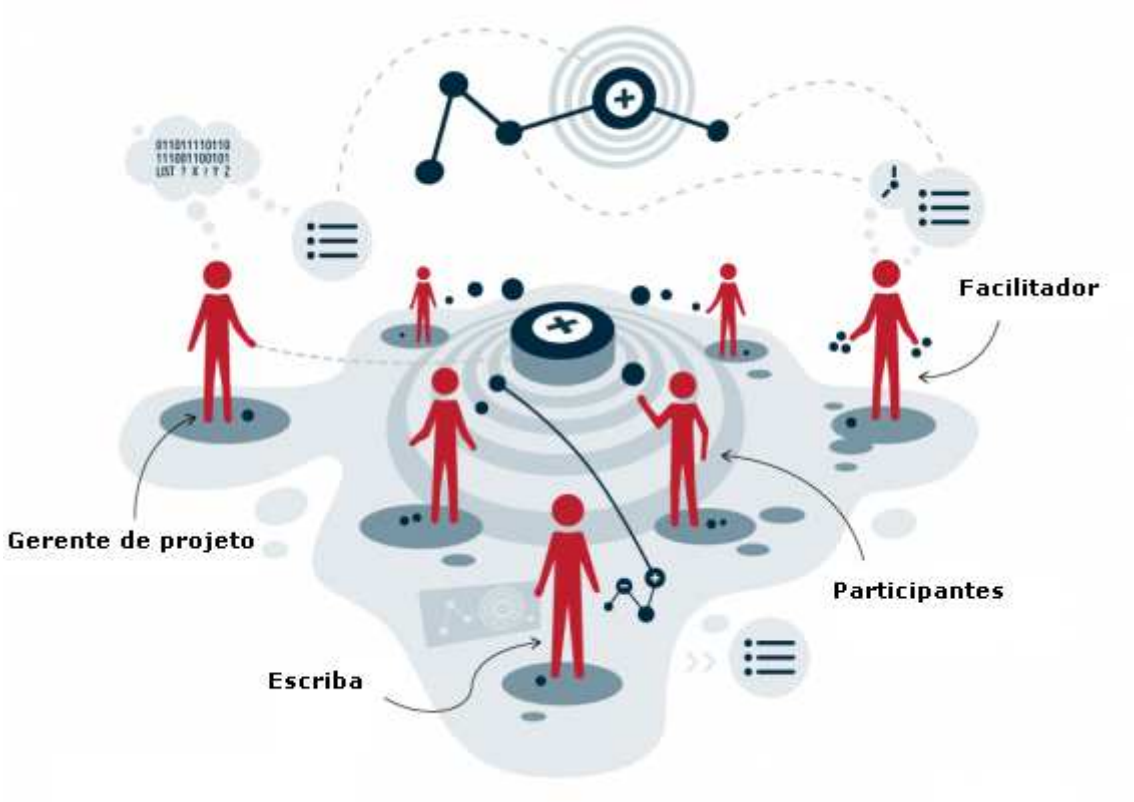

Figura 1. Sessão JAD

O conceito do JAD de abordagem e dinâmica de grupo poderá ser utilizado para diversas finalidades, como: planejamento de atividades técnicas para um grande projeto, discussão do escopo e objetivos de um projeto e estimativa da quantidade de horas necessárias para desenvolver sistemas grandes e complexos.

A maioria das técnicas JAD funciona melhor em projetos pequenos ou médios. Para um sistema grande e complexo podem ser usadas múltiplas sessões JAD para acelerar a definição dos requisitos do sistema.

1.5. Prototipagem

Um protótipo de um sistema é uma versão inicial do sistema que está disponível no início do processo de desenvolvimento. Protótipos de sistemas de software são frequentemente utilizados para ajudar a obter e validar requisitos do sistema.

O protótipo é indicado para estudar as alternativas de interface do usuário; problemas de comunicação com outros produtos; e a viabilidade de atendimento dos requisitos de desempenho. As técnicas utilizadas na elaboração do protótipo são várias: interface de usuário, relatórios textuais, relatórios gráficos, entre outras.

Alguns dos benefícios do protótipo são as reduções dos riscos na construção do sistema, pois o usuário chave já verificou o que o analista captou nos requisitos do produto. Para ter sucesso na elaboração dos protótipos é necessária a escolha do ambiente de prototipagem, o entendimento dos objetivos do protótipo por parte de todos os interessados no projeto, a focalização em áreas menos compreendidas e a rapidez na construção.

2. Desenvolvimento Ágil

O Desenvolvimento Ágil é um conjunto de metodologias de desenvolvimento de software. É uma filosofia onde muitas metodologias se encaixam. As metodologias ágeis aplicam uma coleção de práticas guiadas por princípios e valores que podem ser aplicados por profissionais de software no decorrer do trabalho.

Métodos ágeis são adaptativos ao invés de preditivos. Com os métodos tradicionais, o processo de software é planejado em detalhes por um longo período de tempo. Isso funciona bem se não houver grandes mudanças, o domínio da aplicação e as tecnologias de software sejam bem compreendidos pela equipe de desenvolvimento. Os métodos ágeis foram desenvolvidos para se adaptar as mudanças [Fowler 2000].

Métodos ágeis são orientados às pessoas ao invés de processos. Eles confiam na experiência das pessoas, na competência e na colaboração direta, do que em rigor dos processos centrados em documentos, para produzir software de alta qualidade [Fowler 2000].

A seguir, os métodos ágeis mais comuns serão brevemente discutidos.

2.1. 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 ideia 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 [Soares 2004].

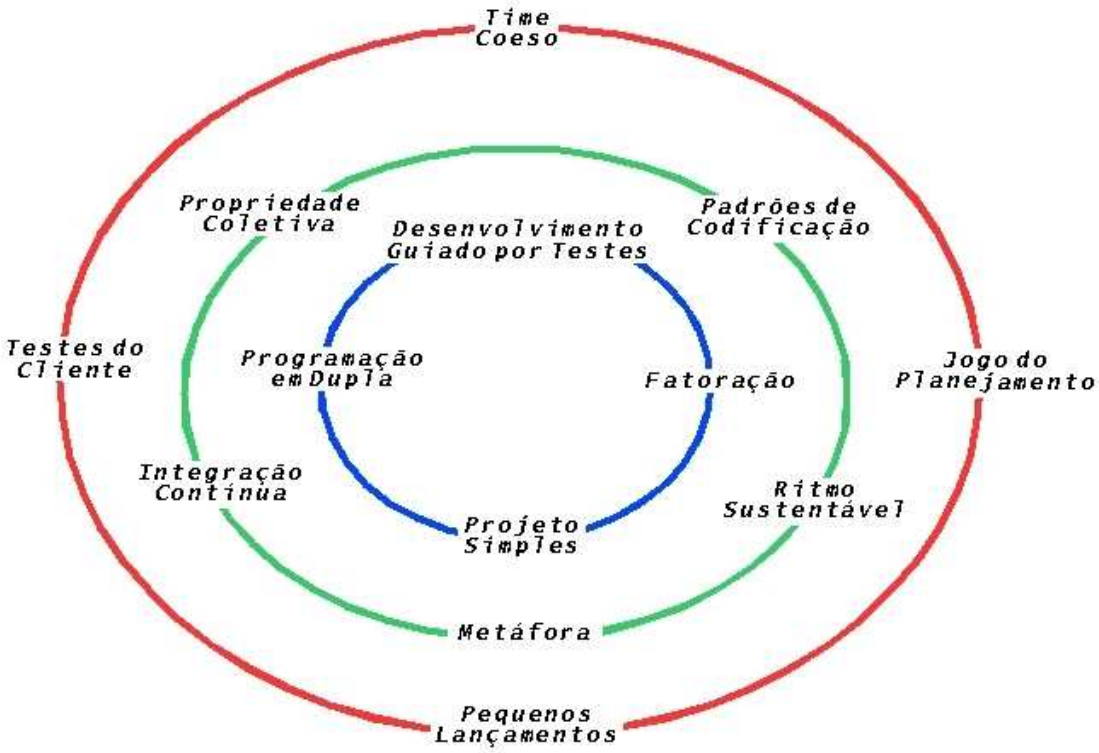

Figura 2. Práticas XP

Durante o planejamento, são usadas as técnicas de elicitação de requisitos, como entrevistas, brainstorming e priorização. A principal ferramenta utilizada para elicitação são os cartões de história em que os usuários escrevem suas histórias de usuário, que é uma descrição de um recurso que fornece o valor de negócio para o cliente [Carvalho Costa 2011].

Antes que as histórias possam ser escritas em cartões, os clientes têm que pensar sobre o que eles esperam do sistema e fazer com que a funcionalidade seja necessária. Este processo é uma espécie de brainstorming.

Desenvolvedores pedem mais detalhes para determinar o que os clientes realmente querem que o sistema faça e estimam o esforço necessário para desenvolvê-la. Com base nestas estimativas e do tempo disponível na próxima iteração, os clientes estão, por sua vez, capazes para escolher as histórias a serem desenvolvidos.

2.1. Scrum

O Scrum é um processo de desenvolvimento iterativo e incremental para gerenciamento de projetos e desenvolvimento ágil de software. Scrum não propõe qualquer técnica de desenvolvimento de software específico para a implementação. Ele se concentra em como uma equipe deve trabalhar em conjunto para produzir um trabalho de qualidade em um ambiente em mudança [Abrahamsson 2002].

As fases do Processo de Scrum são: pré-planejamento, desenvolvimento e pós-planejamento.

Na fase de pré-planejamento (pre-game phase), os requisitos são descritos em um documento chamado backlog. A seguir os requisitos são classificados por prioridade, onde é estimado “o esforço” para o seu desenvolvimento. Nesta fase inclui a definição dos integrantes da equipe, identificação da necessidade de treinamento, as ferramentas que serão utilizadas, como também uma lista com os prováveis riscos de projeto. A fase é concluída com uma proposta de arquitetura de software. As alterações futuras devem ser descritas no backlog.

Na fase de desenvolvimento (game phase), os riscos previamente identificados devem ser mapeados e acompanhados ao longo do projeto para avaliar o seu impacto. Nesta fase, o software é desenvolvido em ciclos interativos (sprints), onde são adicionadas novas funcionalidades. Cada um desses sprints com duração de 2 a 4 semanas são desenvolvidos de forma tradicional (análise, projeto, implementação e testes).

Já na fase de pós-planejamento (post-game phase) é onde acontece a integração do software, os testes finais e a documentação do usuário. A equipe se reúne para analisar o estado do projeto e o software atual é apresentado ao cliente.

As principais técnicas são o scrum product backlog, sprints e scrums. No que diz respeito à engenharia de requisitos, o product backlog desempenha um papel especial no scrum. Todos os requisitos considerados necessários ou úteis para o produto estão listados no product backlog. Ele contém uma lista ordenada de todas as funcionalidades, melhorias e bugs. O product backlog pode ser comparado com um documento de requisitos incompleto e em constante mudança, contendo os requisitos necessários para o desenvolvimento. Cada sprint, iteração com 30 dias de desenvolvimento, está prevista com base nas informações incluídas no product backlog. Tarefas selecionadas a partir do product backlog são movidas para o sprint backlog. Nenhuma alteração será feita para o sprint backlog durante o sprint. Ou seja, não há flexibilidade nos requisitos durante um sprint, mas há total flexibilidade para o cliente escolher os requisitos do próximo sprint.

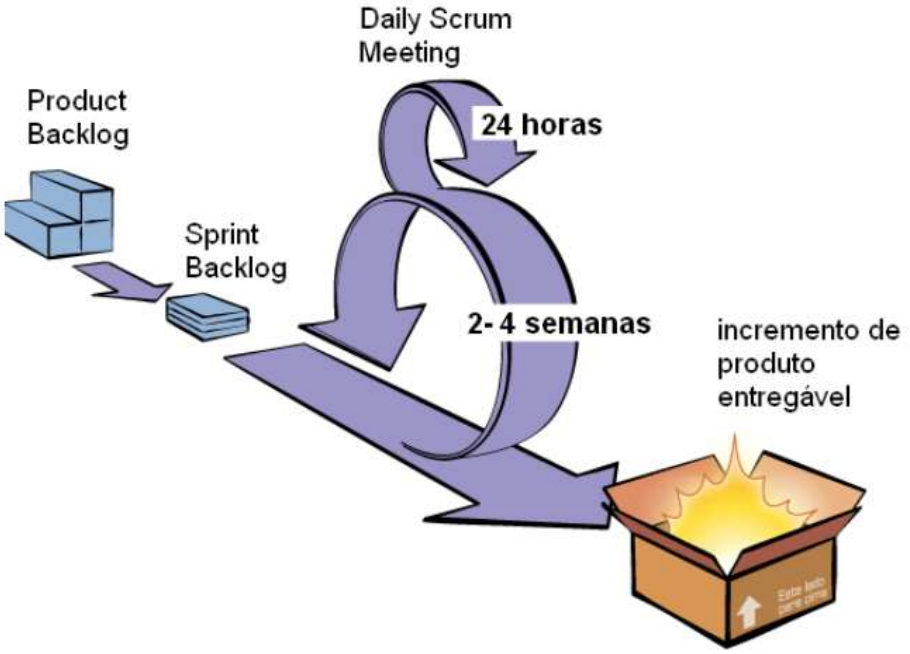

Figura 3. Sprint

Durante uma sprint a equipe de desenvolvimento passa por várias fases (reunião de planejamento da sprint, sprint, revisão da sprint). O objetivo da reunião de revisão da sprint é de que os participantes (clientes e desenvolvedores) tenham entendimento do sistema, sua arquitetura técnica e design, bem como as funcionalidades entregues.

O conhecimento adquirido na reunião de revisão de sprint e o product backlog atual é a base para a próxima reunião de planejamento do sprint. Uma parte da reunião de revisão de sprint, onde os participantes adquirem conhecimentos sobre o sistema, pode ser comparado à revisão de requisitos e a uma apresentação do protótipo evolutivo para o cliente.

2.3. Feature Driven Development (FDD)

O Desenvolvimento Orientado a Funcionalidades (FDD) é um processo de curta iteração para desenvolvimento de software com foco no projeto e na fase de desenvolvimento.

FDD é composto por cinco processos sequenciais. Os três primeiros são executados no início do projeto e os dois últimos durante cada iteração [Fowler 2000].

- Desenvolvimento de um modelo geral;
- Construção de uma lista de funcionalidades;
- Planejamento por funcionalidade;
- Projeto por funcionalidade;
- Desenvolvimento por funcionalidade.

Os dois papéis mais importantes na FDD são o programador-chefe, um desenvolvedor experiente capaz de liderar equipes pequenas de desenvolvimento e de participar da análise de requisitos e da modelagem do projeto, e o proprietário da classe,

que trabalhará com a classe que é atribuída a ele. Na primeira fase, um modelo geral é desenvolvido por membros do domínio e pelos desenvolvedores. A lista de funcionalidades é construída na segunda fase com base no modelo global. O modelo global é composto de diagramas de classe com classes, links, métodos e atributos. As classes e links estabelecem a forma do modelo. Os métodos expressam o comportamento e é a base para a construção da lista de funcionalidades. Uma funcionalidade em FDD é uma função que agrega valor para o cliente. As funcionalidades da lista de funcionalidades são priorizadas pela própria equipe. A lista de funcionalidades é revisada pelos membros do domínio [Lefebvre 1999].

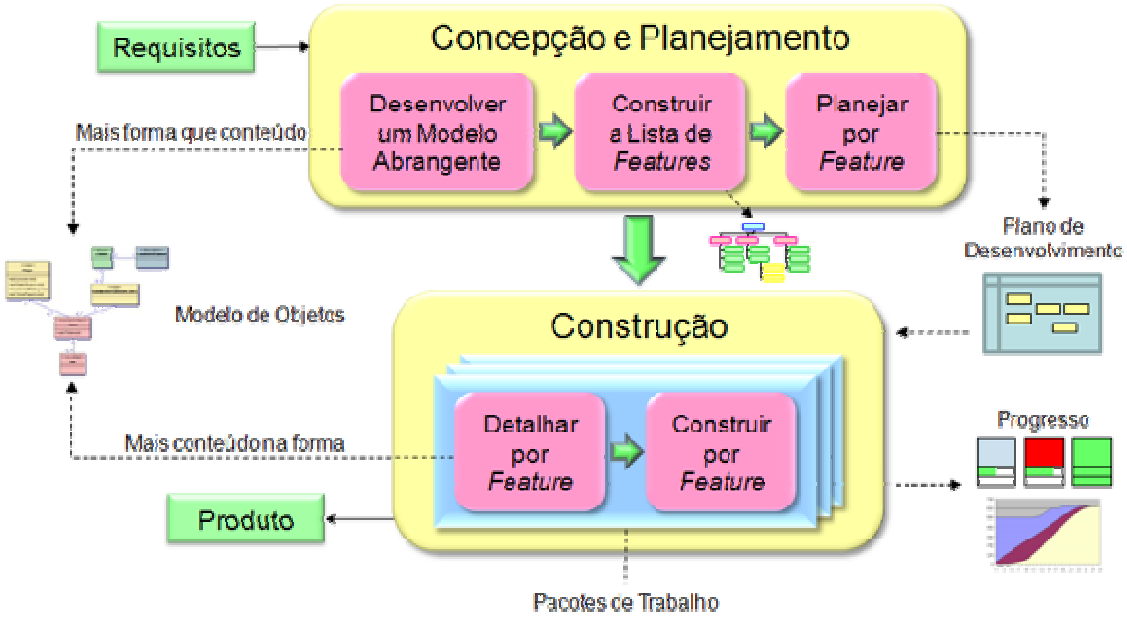

Figura 4. Processos da FDD

Os requisitos em FDD são listados, definidos e documentados através de casos de uso, produzindo diagramas de classe e sequência. Em seguida, os requisitos são agrupados e priorizados conforme importância e dependência.

FDD propõe um encontro semanal de 30 minutos para que a situação das funcionalidades seja discutida e um relatório sobre a reunião seja escrito [Lefebvre 1999]. Esses relatórios podem ser comparados com o monitoramento/gestão dos requisitos.

2.4. Adaptive Software Development (ASD)

Os princípios do Desenvolvimento Adaptável de Software (ASD) são provenientes de pesquisas sobre o desenvolvimento iterativo. ASD fornece uma estrutura para o desenvolvimento de sistemas grandes e complexos com orientação suficiente para impedir os projetos de cair no caos. O método estimula o desenvolvimento iterativo e incremental com prototipagem constante. O processo ASD contém três fases que se sobrepõem: especulação, colaboração e aprendizagem.

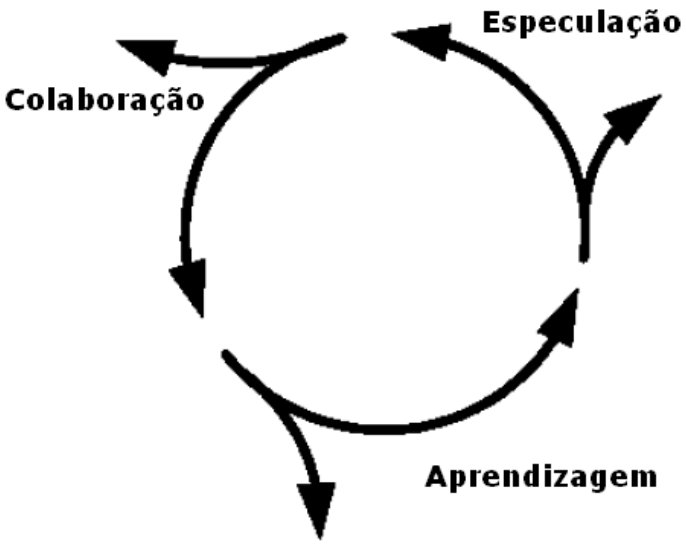

Figura 5. Processo ASD

Os nomes das fases enfatizam a diferença do desenvolvimento de software tradicional. **Especulação** em vez de **planejamento**, porque o planejamento sempre indica que não há incertezas. A importância do trabalho em equipe é destacada pela **colaboração**. Em um ambiente imprevisível, pessoas têm a necessidade de se comunicar para serem capazes de lidar com as incertezas [Highsmith 1996].

Aprendizagem salienta que os requisitos mudam durante o desenvolvimento e existe a necessidade de um conhecimento sobre isso. Na ASD os desvios não são uma falha do sistema, mas levarão para uma solução correta.

Cada projeto começa com a definição da missão do projeto, uma breve declaração indicando o curso do projeto. Durante a fase de iniciação do projeto, o cronograma geral bem como os prazos e os objetivos para os ciclos de desenvolvimento são definidos. Um ciclo em ASD dura entre quatro e oito semanas. Como ASD é orientada a componentes em vez de orientado a tarefas, ela foca nos resultados e sua qualidade. A próxima fase é o planejamento do ciclo adaptativo que contém a fase de colaboração onde o ASD aborda a visão orientada a componentes. Ciclos do planejamento são repetidos quando ocorrem novas exigências e os componentes têm de ser refinados. Revisões com a presença do cliente demonstram a funcionalidade do software durante o ciclo. A fase de release é a última fase de um projeto ASD. Não há recomendações de como esta fase deve ser realizada, mas a ênfase está na importância de capturar as lições aprendidas.

Os requisitos em ASD são definidos através de sessões de JAD com a presença de representantes do cliente.

Como em outros processos ágeis, ASD é baseada em ciclos de desenvolvimento iterativo. ASD destaca que um método cascata só funciona em um ambiente bem compreendido e bem definido. Mas as mudanças são frequentes no desenvolvimento de software e é importante ser capaz de reagir às mudanças e utilizar um método de tolerância às mudanças.

3. Conclusão

As metodologias de desenvolvimento ágil podem ser consideradas novas, porém vêm acompanhando as necessidades de desenvolver softwares com qualidade, no menor tempo possível e que atendam as necessidades dos clientes.

Profissionais da tecnologia da informação vêm aprimorando as concepções destas metodologias para atender as necessidades do mercado e principalmente das pessoas.

Os métodos ágeis são baseados em um forte envolvimento do cliente durante todo o processo de desenvolvimento. Porém, os métodos pedem vários clientes com diferentes experiências, mas às vezes apenas um cliente está disponível. Isso significa que nem todas as questões levantadas podem ser respondidas com detalhes suficientes. Isso pode resultar em atrasos no desenvolvimento ou em tomada de decisões potencialmente incorretas. Além disso, algumas áreas de negócio não podem ser suficientemente compreendidas pelo cliente, de modo que ele não pode dizer a equipe de desenvolvimento sobre os requisitos relacionados. Mesmo que os requisitos sejam levantados em sessões de grupo não é garantido que os usuários ou clientes com a base necessária estejam presentes.

Contudo, o desenvolvimento ágil de software leva a uma melhor compreensão dos requisitos e do processo de desenvolvimento, pois envolve os clientes, que são capazes de tomar as decisões corretas e ter sua própria visão a respeito do sistema a ser desenvolvido.

Uma empresa ao utilizar estes métodos, estará agregando valor aos seus negócios e melhorando o ambiente de seus colaboradores e clientes, tratando-os como pessoas e parceiros. Está é chave no mundo dos negócios, o bem estar de seus colaboradores e a parceria entre o fornecedor e seus clientes, criando um laço de confiança.

Referências

ABRAHAMSSON, Pekka, SALO, Outi, RONKAINEN, Jussi e WARSTA, Juhani. Agile software developlment methods. Review and analysis. Espoo 2002. VTT Publications 478. 107 p.

CARVALHO COSTA, Gustavo Henrique de. Engenharia de Requisitos no Desenvolvimento de Software Ágil. Universidade Federal de Pernambuco, Centro de Informática, 2011.

FOWLER, Martin. The new methodology. http://www.martinfowler.com/ articles/newMethodology.html, 2000.

HIGHSMITH, James A. Adaptive Software Development. Dorset House Publishing, 1996.

LEFEBVRE, Eric, COAD, Peter e DE LUCA, Jeff. Java Modeling in Color with UML. Prentice Hall PTR, 1999.

POMPILHO, S. Análise Essencial Guia Prático de Análise de Sistemas. Rio de Janeiro: Ed. Ciência Moderna Ltda, 1995.

PRESSMAN, Roger S. Engenharia de Software. São Paulo: Ed. Markon Books, 1995.

SOMMERVILLE, Ian e KONTONYA, Gerald. Requirements Engineering. John Wiley & Sons, 1997.

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 Lafaiet, 2004.