JORGE CARLOS ARES GARCIA

ENGENHARIA DE REQUISITOS EM MÉTODOS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para conclusão do Curso de Especialização em Engenharia de Software MBA-USP

São Paulo
2002

JORGE CARLOS ARES GARCIA

ENGENHARIA DE REQUISITOS EM MÉTODOS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para conclusão do Curso de Especialização em Engenharia de Software MBA-USP

Área de Concentração:
Engenharia de Software

Orientadora:
Profª. Drª. Selma Shin Shimizu Melnikoff

São Paulo
2002

"Facts do not cease to exist because they are ignored..."
Aldous Huxley

RESUMO

O desenvolvimento de sistemas, em alguns setores de negócio, é encarado como um diferencial competitivo sob o ponto de vista da velocidade e pela forma como o sistema proporciona capilaridade ao negócio.

Considerados como parte essencial do desenvolvimento, os métodos de Engenharia de Requisitos são avaliados atualmente, na sua forma, conceito e aplicabilidade em tal domínio.

A presente monografia aborda a Engenharia de Requisitos e seus processos sob a perspectiva formal de suas definições e os processos de elicitação de requisitos nos Métodos Ágeis que procuram cumprir as expectativas impostas de velocidade e adaptabilidade.

ABSTRACT

System development in some business markets are seen as competitive differential under the point of view of the speed and for the form as the system provides capillarity to the business.

Considered essential part of the development, the methods of Requirements Engineering are appraised now, in its form, concept and applicability in such domain.

The present work presents the Requirements Engineering and their processes under the formal perspective of their definitions and the requirement elicitation processes in Agile Methods which try to accomplish the imposed expectations of speed and adaptability.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Este capítulo apresenta a abrangência do trabalho, os objetivos, a motivação e as considerações sobre a Engenharia de Requisitos para processos ágeis de desenvolvimento de sistemas de software.

1.1 Considerações Iniciais

O efetivo uso de um processo de Engenharia de Requisitos, pode reduzir custos, implementar qualidade nos produtos e aumentar a satisfação dos clientes.

O processo de Engenharia de Requisitos deve ser encarado como uma atividade que está presente por todo o ciclo de vida de um sistema, onde informações são extraídas, certificadas e testadas objetivando atingir os requisitos reais do domínio envolvido.

De acordo com o relatório CHAOS publicado por Standish Group em 1998, das oito principais razões da falha de um projeto de software, cinco delas são relacionadas com requisitos (Young, 2001):

- Requisitos incompletos
- Falta de envolvimento do usuário
- Expectativa irreal do cliente
- Mudança de requisitos e especificações
- Funções não necessárias entregues.

O resultado disso, segundo o mesmo relatório, indica que: 46 % dos projetos sofrem modificações no escopo inicial, 28 % falharam completamente e 26% dos projetos foram entregues com sucesso. Outros números indicam que o custo final médio é

89% acima do estimado, o cronograma médio é ultrapassado em 122% e 45% da funcionalidade entregue não é utilizada.

Progressos na disciplina de Engenharia de Requisitos estão sendo alcançados e esforços contínuos, visando ganhos de qualidade e custos estão em andamento. Processos e métodos são propostos e estudados para incluir a indústria de software dentro de uma previsibilidade de custos conforme o modelo de outras indústrias tradicionais.

Atualmente deve-se concordar que se vive em um ambiente econômico onde fortunas são criadas e perdidas em questão de meses, denotando a essencialidade da velocidade e adaptabilidade do negócio ao mercado.

Os ciclos de produtos da indústria tradicional encurtaram e o software, como apoio ou base para tais negócios, seguiu o mesmo caminho; conseqüentemente, um equívoco entre a necessidade real e o que o software pode oferecer pode acarretar, no mínimo, uma situação bastante desconfortável para o negócio.

Em ambientes com essas características, muitos dos métodos clássicos de requisitos propostos podem não possuir aderência ou funcionalidade necessária à demanda requerida; em contra partida, a inexistência ou um processo falho de requisitos poderá corroborar os números negativos do relatório do The Standish Group (Young, 2001).

Métodos ágeis começaram ser propostos como resposta a essa necessidade emergente, reduzindo drasticamente as etapas no ciclo de desenvolvimento de software e priorizando o mais rápido possível a entrega do software ou parte dele, de forma funcional, ao cliente, baseando suas atividades e processos característicos nas imprevisibilidades dos requisitos do negócio e sua capacidade de se moldar rapidamente a novas situações não planejadas, prometendo a resposta para os requisitos de velocidade e adaptabilidade emergentes.

1.2 Objetivos

O primeiro objetivo deste trabalho é apresentar a Engenharia de Requisitos relacionada com os processos de software clássicos, através de seus processos, atividades e métodos. Entende-se por processo clássico, aquele em que o software é desenvolvido através das fases de análise, projeto, implementação e testes, apoiando-se na elaboração de modelos.

O segundo objetivo é expor o processo de requisitos dos Métodos Ágeis; para isso, são apresentados os seus processos característicos e os Métodos Ágeis mais utilizados. É feita a discussão de forma geral e sob a óptica do método Extreme Programming, sua aderência ao modelo SW-CMM, no que diz respeito à gerência de requisitos, e um comparativo com o processo unificado da Rational com relação a disciplina de requisitos.

Finalmente, visa coletar as conclusões e as possíveis melhorias das práticas de Engenharia de Requisitos para uma maior abrangência na indústria de software.

1.3 Abrangência

O escopo desta monografia está inserido na pesquisa de livros, artigos impressos e materiais disponíveis eletronicamente. O conteúdo apresentado está baseado nas afirmações e conclusões dos autores citados.

A monografia se restringe ao aspecto do processo da Engenharia de Requisitos, apesar de muitas vezes existir menções ao processo completo de desenvolvimento de software, ocasionado principalmente, pela característica iterativa dos Métodos Ágeis. Não é escopo desta monografia promover ou refutar algum tipo de método em particular, apenas servir como instrumento de informação. Citações favoráveis ou desfavoráveis são referenciadas aos seus respectivos autores.

1.4 Motivação

A motivação desta monografia emergiu pelos assuntos tratados no curso de Engenharia de Software ministrado pela Profª. Drª. Selma Melnikoff e o de Engenharia de Requisitos ministrado pela Profª. Jussara Pimenta Matos, constituintes do curso de especialização MBA – Engenharia de Software da Escola Politécnica da Universidade de São Paulo.

Os assuntos tratavam sobre a especial contribuição da Engenharia de Requisitos no desenvolvimento de um produto de software correto e de qualidade em ambientes caracterizados pela volatilidade e instabilidade de seus requisitos, somados a uma crescente necessidade pela entrega do produto de software em ciclos cada vez mais curtos. O surgimento de um movimento denominado Desenvolvimento Ágil de software, onde seus valores e princípios exploram esses assuntos, acabou impulsionando o presente trabalho.

A análise da presença dos princípios e conceitos da Engenharia de Requisitos em tais métodos tornou-se relevante, exibindo de que forma são tratados e aplicados como uma forma de solucionar a demanda por software em ambientes instáveis e competitivos. Essa análise vai de encontro com a necessidade profissional, exigida pelo mercado de software, sobre questões referentes ao desenvolvimento de requisitos de alta velocidade que normalmente não são amplamente discutidos.

1.5 Estrutura da Monografia

Este material cobre aspectos da Engenharia de Requisitos no seu processo dentro das etapas de desenvolvimento de software e suas características nos Métodos Ágeis. Está estruturalmente composto por cinco capítulos:

- Capítulo primeiro: no presente capítulo são descritos as razões e objetivos para o desenvolvimento desta monografia e a sua estrutura.

- Capítulo segundo: é apresentada a disciplina de Engenharia de Requisitos com relação a seus processos, práticas, características e métodos.

- Capítulo terceiro: apresenta os Métodos Ágeis de desenvolvimento, sua definição, valores e princípios, destacando vários Métodos Ágeis em utilização atualmente.

- Capítulo quarto: apresenta o processo de Engenharia de Requisitos nos Métodos Ágeis de forma geral, um comparativo entre o processo unificado da Rational e método Extreme Programming, além de uma abordagem do método Extreme Programming sobre o modelo SW-CMM sobre a óptica de gerência de requisitos.

- Capítulo quinto: apresenta as conclusões sobre o material apresentado, as contribuições do trabalho e as sugestões para extensão da pesquisa.

2 ENGENHARIA DE REQUISITOS

Neste capítulo é apresentada a disciplina de Engenharia de Requisitos de forma geral, sob perspectiva de seu formalismo, atividades, características e modelos de processos aplicados.

O conteúdo deste capítulo procura ressaltar os pontos principais na Engenharia de Requisitos que são amparados ou refutados pelos métodos de desenvolvimento ágil. O capítulo é estruturado nos seguintes tópicos: introdução à Engenharia de Requisitos, o processo de Engenharia de Requisitos, a qualidade nos requisitos e métodos e técnicas.

2.1 Introdução

A definição de requisitos, sob o ponto de vista da Engenharia de Software, é (Wiegers, 1999):

1. Uma condição ou capacidade necessitada por um usuário para resolver um problema ou alcançar um objetivo.

2. Uma condição ou capacidade que deve ser satisfeita ou possuída por um sistema ou componente do sistema para satisfazer um contrato, um padrão ou uma especificação.

3. Uma representação documentada de uma condição ou capacidade como em (1) ou (2).

A engenharia, de forma genérica, pode ser definida, segundo Holanda (1999), como sendo a "*Arte de aplicar conhecimentos científicos e empíricos e certas habilitações específicas à criação de estruturas, dispositivos e processos...*".

Pode-se definir Engenharia de Requisitos como uma arte de aplicar conhecimentos científicos e empíricos de forma a alcançar a condição ou capacidade necessária de um usuário sobre um determinado problema.

De uma forma mais específica, segundo Kotonya (1997), a Engenharia de Requisitos corresponde a uma disciplina que cobre todas as atividades envolvidas na descoberta, documentação e manutenção de uma coleção de requisitos para um sistema computacional, utilizando técnicas sistemáticas e repetitivas usadas para garantir que os sistemas de requisitos sejam completos, consistentes e relevantes.

A Engenharia Requisitos viabiliza mecanismos adequados para entender e analisar as reais necessidades do cliente, avaliar a viabilidade, negociar soluções, especificar as necessidades sem ambigüidade, validar a especificação e gerenciar os requisitos.

Portanto, um sólido processo apoiando a Engenharia de Requisitos é a melhor solução para assegurar que a especificação do sistema alcance as necessidades e satisfaça as necessidades do cliente (Pressman, 2001).

O cliente, definido como um indivíduo ou organização, que obtém benefícios diretos ou indiretos de um produto de software (Wiegers, 1999), torna-se parte central do processo de requisitos. Desta forma, um processo de Engenharia de Requisitos deve possuir algumas atividades essenciais, que são basicamente focadas no cliente:

- Elicitação dos requisitos;
- Análise e negociação dos requisitos;
- Especificação dos requisitos;
- Verificação dos requisitos;
- Gerência dos requisitos.

Este processo é detalhado a seguir.

2.2 O Processo de Engenharia de Requisitos

Um processo pode ser definido genericamente como "*Maneira pela qual se realiza uma operação segundo determinadas normas, métodos e técnicas*" (Holanda, 1999).

Um processo característico tipicamente envolve pessoas, grupos ou organizações integradas a procedimentos e métodos que definem os relacionamentos das atividades, podendo ter suporte de ferramentas e equipamentos.

Uma estrutura analítica do processo de Engenharia de Requisitos pode ser decomposta, segundo Wiegers (1999), em dois sub-domínios: Desenvolvimento de Requisitos e Gerência de Requisitos (Figura 1). O sub-domínio de desenvolvimento de requisitos divide-se em quatro subgrupos de atividades fundamentais: Descoberta ou Elicitação, Análise e Negociação, Especificação e Verificação.

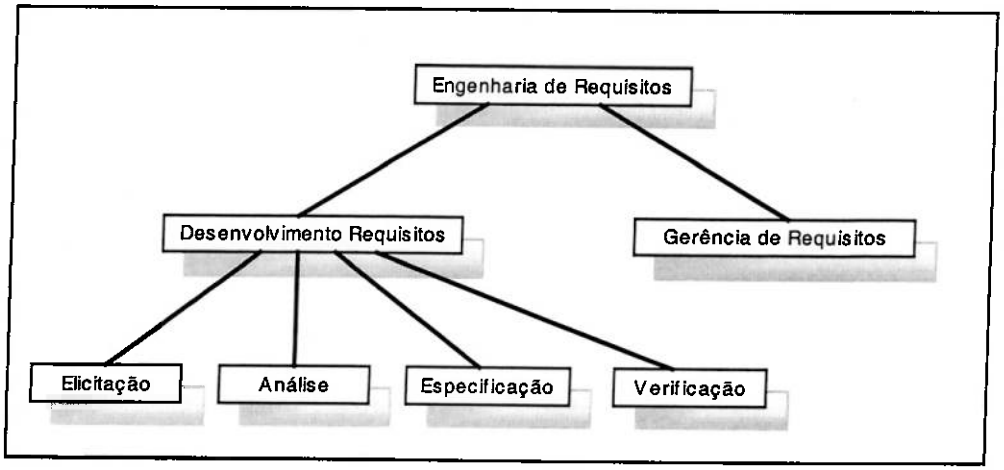

Figura 1 - Estrutura Analítica (Wiegers, 1999)

As atividades de desenvolvimento de requisitos são compatíveis com as apresentadas por Kotonya (1997) e podem ser definidas como:

- Elicitação: Os requisitos do sistema são descobertos através de consultas aos *stakeholders* e a partir de documentos, conhecimento do domínio e estudos de mercado.

- Análise e Negociação: Os requisitos descobertos são analisados em detalhes e os conflitos e as inconsistências detectados são negociados com os *stakeholders* que definem quais requisitos deverão ser priorizados. Este processo é necessário porque os conflitos entre requisitos são inevitáveis, e a informação pode estar incompleta ou incompatível com o custo esperado para o projeto.

- Especificação: Os requisitos resultantes são documentados em um apropriado nível de detalhe, de modo a ser inteligível a todos os *stakeholders* envolvidos.

- Validação: os requisitos documentados são validados antes de serem usados como base do desenvolvimento do sistema, devendo ser cuidadosamente verificados em relação à sua consistência e seu grau de completeza.

A Gerência de Requisitos é ligada ao estabelecimento e manutenção de um acordo com o cliente sobre os requisitos para o projeto de software (Wiegers, 1999). Kotonya (1997) define que as principais funções em um processo de gerência de requisitos são:

- Gerenciar as mudanças dos requisitos contratados.

- Gerenciar o relacionamento entre os requisitos.

- Gerenciar as dependências entre o documento de requisitos e outros documentos de requisitos durante o processo de engenharia de software.

Complementado esta definição, Wiegers (1999), destaca que as atividades do processos de desenvolvimento e gerência de requisitos não devem ser mutuamente exclusivas e muitas vezes são consideradas como atividades sobrepostas. A tabela 1 apresenta estas atividades.

Tabela 1- Atividades Engenharia Requisitos

Desenvolvimento	1) Identificar as classes de usuários esperados para o produto.
	2) Elicitar necessidades de indivíduos que representam cada classe de usuário.
	3) Entender as atividades e objetivos atuais do usuário e necessidades do negócio.
	4) Analisar as informações recebidas dos usuários e classificá-las em requisitos funcionais, regras de negócio, atributos de qualidade, soluções sugeridas e informação extras.
	5) Dividir os requisitos do nível de sistema em subsistemas e alocar os requisitos destinados ao software para componentes de software.
	6) Negociar prioridades de implementação
	7) Entender a importância dos atributos de qualidade
	8) Traduzir as necessidades coletadas dos usuários em especificações escritas e modelos.
	9) Revisar a especificação de requisitos para assegurar um entendimento comum dos usuários para os requisitos contratados e corrigir qualquer problema antes do envio ao grupo de desenvolvimento.
Gerenciamento	1) Definir a *baseline* dos requisitos (Uma fotografia no tempo representando o contrato de requisitos atual).
	2) Revisar as mudanças de requisitos propostos e avaliar o impacto de cada mudança proposta antes de aprová-los.
	3) Incorporar as mudanças de requisitos aprovadas no projeto de forma controlada.

	4) Manter os planos de projeto atualizados com os requisitos do projeto de software.
	5) Negociar novos comprometimentos baseados na estimativa do impacto em função das mudanças de requisitos.
	6) Rastrear os requisitos em relação aos seus projetos, modelos, código fonte e casos de teste correspondentes.
	7) Apontar o estado dos requisitos e atividades de mudança no andamento do projeto.

O processo e suas atividades descritas exigem uma interação entre os fornecedores e os consumidores de informação no projeto de software, visando um resultado na forma de requisitos reais do sistema a ser desenvolvido.

A figura 2 mostra a interação entre a gerência e o desenvolvimento de requisitos e sua linha divisória sob o ponto de vista de atuação em um domínio de informação.

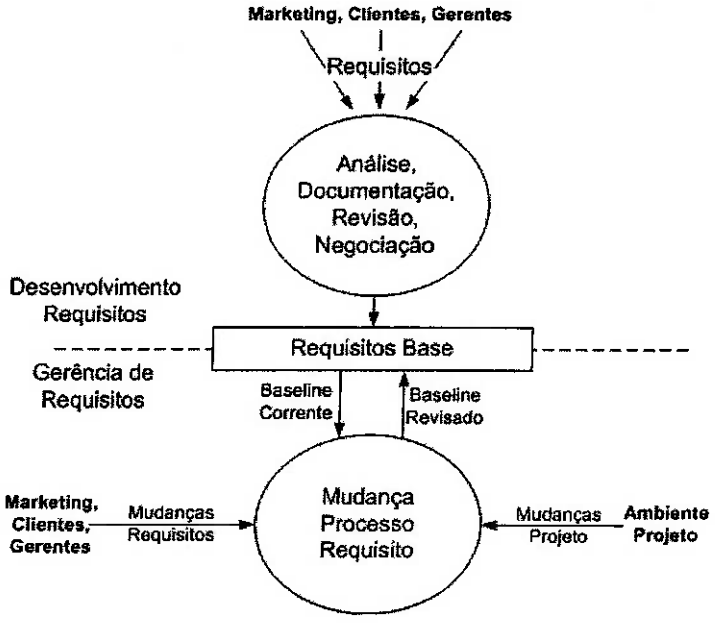

Figura 2 - Gerência e Desenvolvimento de Requisitos

Deve ficar claro que as atividades citadas e relacionadas com requisitos não devem ficar restritas a apenas uma fase do ciclo de vida do processo. As atividades de um processo de requisitos englobam completamente todo o ciclo de vida de um sistema e não apenas atividades necessárias direcionadas ao início de um projeto (Young, 2001); (Wiegers, 1999); (Kotonya, 1997).

Wiegers (1999) destaca como o processo de Engenharia de Requisitos interfere em todos as outras atividades relacionadas a um projeto de desenvolvimento de software (figura 3), concluindo que, mesmo tratando este processo como um evento isolado e estático, sua influência se refletirá diretamente nos outros processos e atividades que compõem o ciclo de vida do projeto de software. A interação da disciplina de Engenharia de Requisitos com os outros processos é inerente ao desenvolvimento de software.

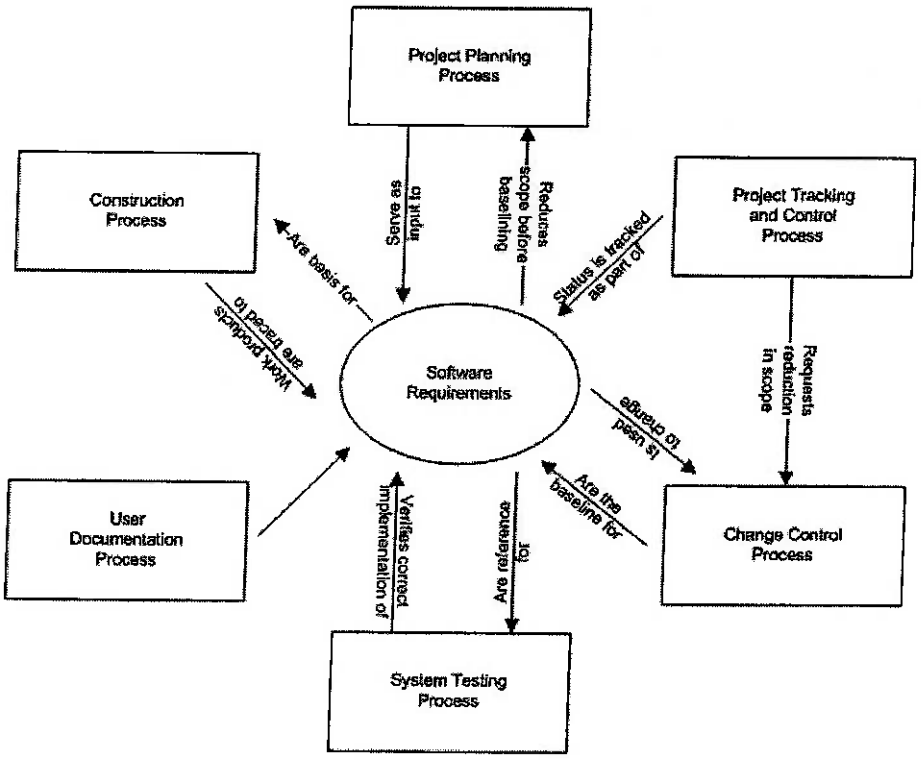

Figura 3 - Interface entre Requisitos de Software e outros Processos (Wiegers, 1999)

A implantação de um processo de Engenharia de Requisitos implica, sem dúvida em custos, mas é necessária, independente do porte dos projetos (Young, 2001); a

sua adoção e a busca por uma melhoria contínua produzem efeitos positivos comprovados na organização. Essa adoção requer fundamentalmente, segundo Young (2001), um comprometimento efetivo da organização e suporte da alta gerência para seu sucesso.

Uma outra característica importante em um processo de Engenharia de Requisitos é a mudança.

As mudanças são comuns a um processo fundamentado em pessoas. Um processo de requisitos deve ser capaz de acomodar mudanças dos requisitos ao longo do ciclo de vida do sistema. As alterações podem ser causadas por vários fatores internos ou externos ao domínio do projeto. Os fatores internos podem ser erros ou não entendimento do problema a ser solucionado, problemas de projeto ou implementação, requisitos emergentes, conforme *stakeholders* desenvolvem um melhor entendimento do sistema. Como fatores externos podem-se citar as estratégias de negócios, as mudanças econômicas e os novos concorrentes.

A capacidade de absorção de mudanças de requisitos em um processo denota sua maturidade e efetividade em relação às expectativas do cliente ou organização (Young, 2001).

2.3 Modelos de Processos de Requisitos

Para solucionar problemas na indústria, um engenheiro ou equipe de engenheiros deve incorporar uma estratégia de desenvolvimento que englobe o processo, métodos e ferramentas. Esta estratégia é muitas vezes conhecida como modelo de processo ou paradigma de engenharia de software (Pressman, 2001). Dorfman (1997) destaca os seguintes modelos como sendo mais significativos do ponto de vista de Engenharia de Requisitos: cascata, prototipação, incremental, evolucionário e espiral. Esses modelos são apresentados a seguir.

2.3.1 Modelo Cascata

Provavelmente é uns dos modelos mais largamente utilizados. Neste tipo de modelo (figura 4), a determinação dos requisitos deve ser completa ou muito próxima disto, antes que qualquer implementação comece. O modelo exige um alto grau de visibilidade gerencial e controle (Dorfman, 1997). A adoção de um modelo seqüencial como o tipo cascata possui alguns problemas comuns, como aqueles citados por Pressman (2001):

- Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. Embora o modelo linear possa trabalhar com iterações, isso deve ser feito de forma indireta podendo causar, no caso de mudanças, confusão na atuação da equipe de projeto.

- Existe uma grande dificuldade para o cliente expor todos os requisitos. O modelo seqüencial necessita isto e tem dificuldade em absorver incertezas naturais que existam no início dos projetos.

- O cliente deve ter paciência. Uma versão funcional do software somente estará disponível nas fases finais do ciclo de vida. Os requisitos não detectados até a avaliação do software funcional, podem ser desastrosos para o projeto.

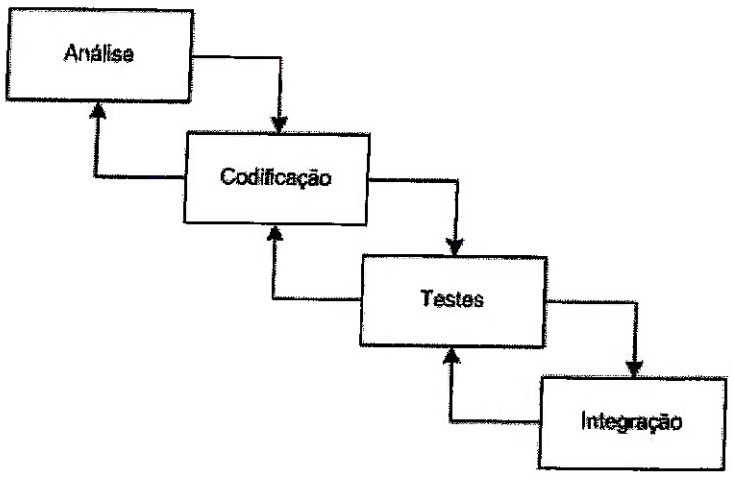

Figura 4 - Modelo Cascata (Dorfmann, 1997)

2.3.2 Modelo Prototipação

O ciclo de vida da prototipação objetiva o uso de um sistema funcional que auxilie a determinação de requisitos (Dorfman, 1997). Neste modelo (figura 5), algumas funções são construídas com controle e formalidade suficientes, para serem manipuladas pelo usuário, de modo a determinar mais requisitos detalhadamente. O protótipo é, portanto, avaliado pelos clientes e utilizado para refinar os requisitos do software a ser desenvolvido. Iterações ocorrem de modo que o protótipo é ajustado para satisfazer as necessidades do cliente, enquanto que, em paralelo, habilita o projetista a obter um melhor entendimento do que é necessário ser feito (Pressman, 2001).

A quantidade de análise de requisitos que precede a prototipação depende da especificação do problema. Normalmente, recomenda-se que o protótipo deva ser usado para ajudar a gerar uma coleção válida de requisitos; após os requisitos serem definidos, eles devem ser documentados e o desenvolvimento deve prosseguir, utilizando os requisitos como *baseline* para a gerência (Dorfman, 1997).

A utilização da prototipação no ciclo de vida do projeto pode ser considerada como uma ferramenta ou um método de apoio à análise de requisitos dentro de um modelo seqüencial tipo cascata.

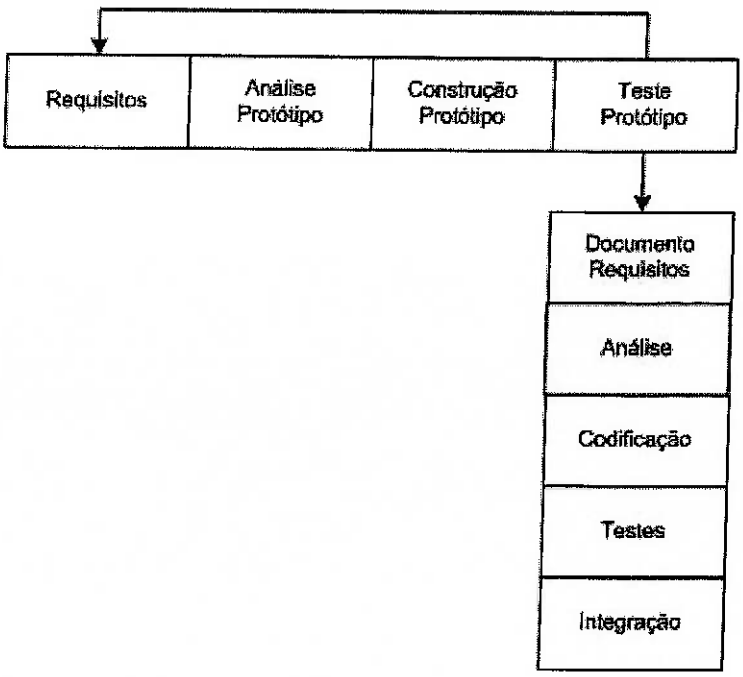

Figura 5 - Modelo Prototipação (Dorfman, 1997)

2.3.3 Modelo Incremental

O modelo de ciclo de vida de desenvolvimento incremental tem, por meta, uma análise de requisitos e esforço de especificação unitários, ou seja, limitados ao escopo de um incremento no ciclo de desenvolvimento por vez; os requisitos e os esforços são distribuídos por uma série de incrementos que são, a princípio, estanques mas podem se sobrepor no ciclo de vida do modelo (figura 6).

Na sua concepção original, os requisitos são considerados estáveis, como no ciclo de vida em cascata, mas na prática os requisitos para incrementos posteriores podem mudar conforme a tecnologia avança ou a experiência com as entregas dos incrementos anteriores aumenta. Segundo Dorfman (1997), este modelo não é efetivamente muito diferente do modelo de desenvolvimento evolucionário.

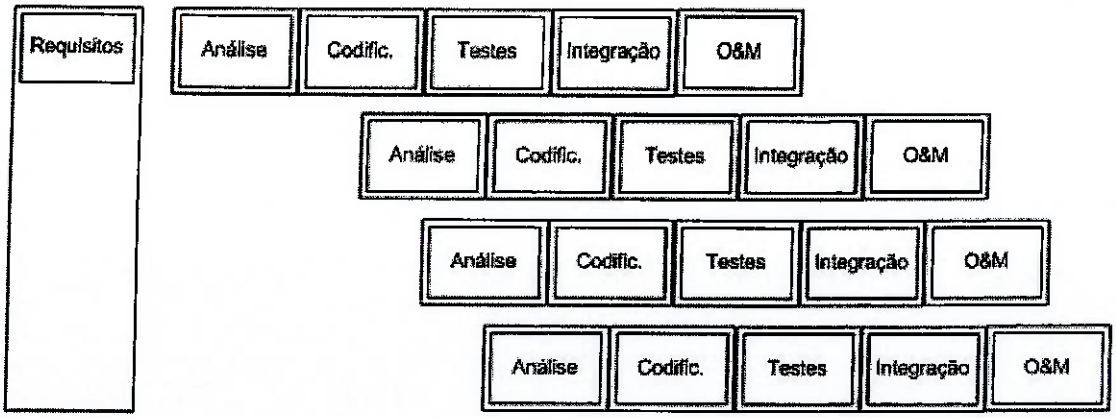

Figura 6 - Modelo Incremental (Dorfman,97)

2.3.4 Modelo Evolucionário

O modelo de ciclo de vida de desenvolvimento evolucionário (figura 7) se baseia em uma seqüência de esforços de desenvolvimento, cujo propósito é, a cada ciclo, entregar um produto a ser utilizado no ambiente operacional do cliente. A diferença com o modelo de prototipação, cujo propósito de cada produto entregue é auxiliar na determinação dos requisitos, no modelo evolucionário, cada ciclo ou evolução fornece um produto com mais alguma capacidade operacional. Contudo, a realimentação dos usuários do sistema funcional afeta os requisitos necessários para entregas posteriores (Dorfman, 1997).

Cada entrega, neste modelo, representa um ciclo de desenvolvimento completo, incluindo a análise de requisitos. O produto de cada fase de análise de requisitos é uma soma ou uma agregação para a fase de análise de requisitos correspondente das entregas anteriores. Cada entrega pode ser encarada como um pequeno exemplo de ciclo de vida cascata, em função de um processo de desenvolvimento rápido, e ciclo de entrega o menor possível, para minimizar os riscos de requisitos instáveis (Dorfman, 1997).

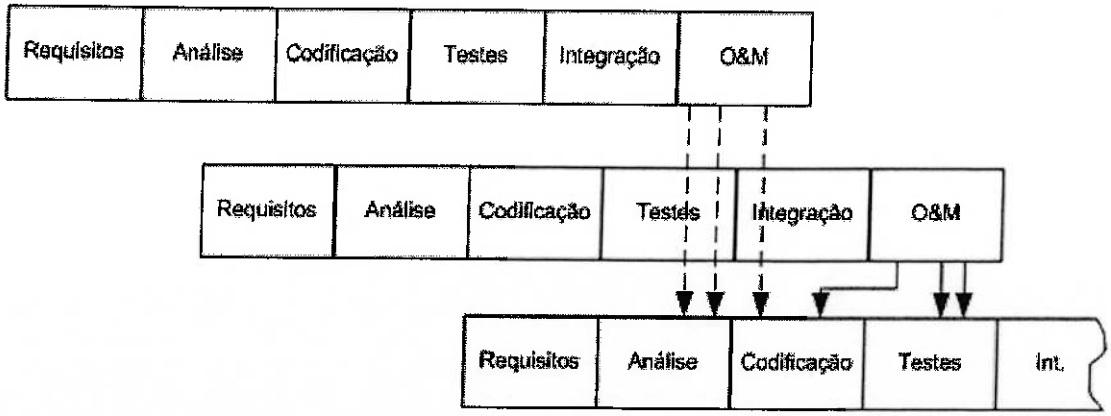

Figura 7 - Modelo Evolucionário - (Dorfman,97)

2.3.5 Modelo Espiral

O modelo Espiral enfatiza a gerência do desenvolvimento do produto e seu risco. Dorfman (1997), descreve o modelo como uma combinação dos modelos cascata, de prototipação e incrementais utilizados por várias fases do desenvolvimento.

O modelo espiral deixa clara a idéia de que a re-avaliação a cada ciclo completo da espiral permite direcionar o comportamento do processo através das realimentações dos clientes, dos resultados dos protótipos, das visões iniciais, dos avanços de tecnologia e da determinação de riscos de projetos e financeiros.

O modelo espiral é caracterizado como um modelo gerador de processo: dada uma gama de condições, a espiral produz um modelo de desenvolvimento mais detalhado.

Por exemplo, na situação onde os requisitos possam ser previsíveis e com baixos riscos, o modelo espiral irá se comportar como uma aproximação do processo em cascata. Se os requisitos são incertos, outros modelos tais como incremental ou prototipação podem ser adaptados ao modelo espiral. E, como mencionado anteriormente, a re-avaliação após cada ciclo de espiral completo permite ajustes no processo, em função das fases concluídas (Dorfman, 1997).

Para um efetivo desenvolvimento de requisitos, as atividades devem se processar de forma iterativa e contínua, com as fronteiras do domínio de cada atividade se

sobrepondo e criando um mecanismo de realimentação de informação de uma atividade para outra (Kotonya, 1997). Estas atividades são realizadas seguindo também um modelo espiral, conforme apresentado na figura 8.

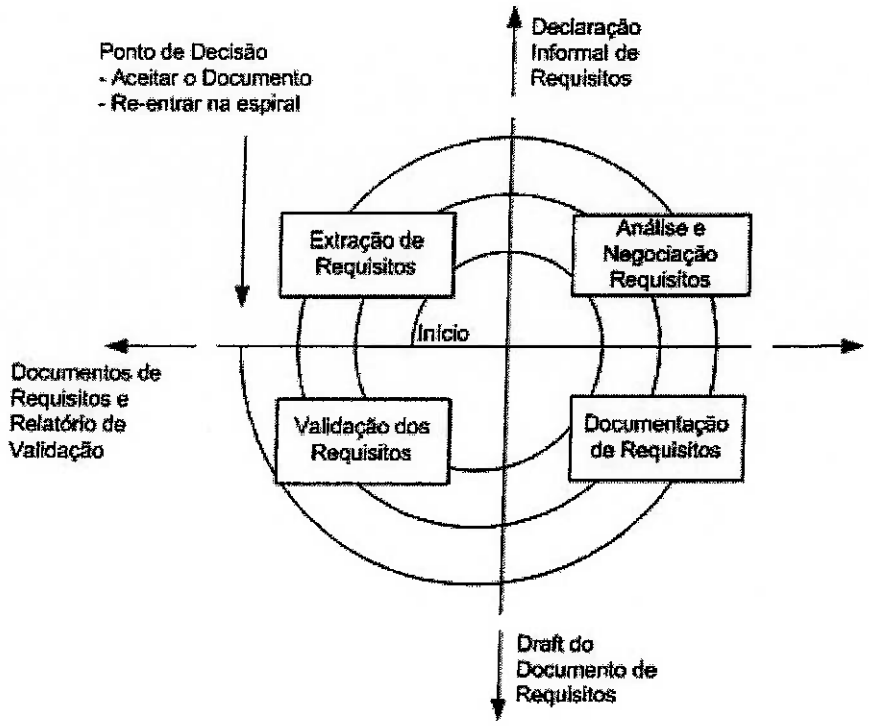

Figura 8 - Modelo Espiral (Kotonya, 1997)

Neste modelo, diferentes atividades de requisitos são repetidas até a decisão de aceitação do documento de requisitos. Se a versão atual do documento de requisitos possui problemas identificados, o desenvolvimento de requisitos (elicitação, análise, documentação e validação) é realimentado no modelo espiral e continua até que o documento de requisitos produzido possua aceitação ou fatores externos como limites de cronograma e falta de recursos indiquem que o desenvolvimento de requisitos deva ser concluído. Quaisquer mudanças dos requisitos tornam-se parte da Gerência de Requisitos (Kotonya, 1997).

2.4 Qualidade em Engenharia de Requisitos

Segundo Wiegers (1999), requisitos com qualidade são aqueles que atingem as necessidades reais do cliente ou organização.

Um processo de requisitos falho ou com pouca efetividade no encontro dos requisitos reais dos usuários pode acarretar sérios prejuízos ao projeto, conforme pode ser observado na Tabela 2.

Tabela 2 - Custo correção erros (Boehm, 1981) apud (Young, 2001)

Fase onde o erro foi encontrado	Custo Relativo
Requisitos	1
Projeto	3-6
Codificação	10
Teste de Desenvolvimento	15-40
Teste de Aceitação	30-70
Operação	40-1000

Uma forma de se obter qualidade nos requisitos é utilizar processos de desenvolvimento de software iterativos, quando possível, como forma para redução dos custos e diminuição de riscos. Através de repetidas interações e com a presença de um processo de requisitos em todas as fases do processo de desenvolvimento, requisitos tornam-se mais consistentes e estáveis, minimizando a possibilidades de mudanças nos estágios finais do desenvolvimento (Young, 2001).

Um processo de Engenharia de Requisitos, em suas atividades de desenvolvimento e gerência de requisitos, deve contemplar certas condições para obter qualidade em seus resultados. Estas condições são apresentadas, agrupadas em termos de:

- verificação de requisitos
- elicitação de requisitos
- especificação de requisitos

- gerência de requisitos.

Em termos de verificação de requisitos, deve-se observar as seguintes características:

- A especificação de requisitos contempla o comportamento e características do software a ser construído.

- Os requisitos são completos.

- Todos os pontos de vista dos requisitos são consistentes.

- Os requisitos proporcionam uma base adequada para continuar com a análise, construção e testes do produto.

Em termos de elicitação de requisitos deve-se verificar se os requisitos são (Wieger, 1999):

- *Completos*: cada requisito deve descrever completamente a funcionalidade a ser entregue.

- *Corretos*: cada requisito deve descrever corretamente a funcionalidade a ser construída. Somente usuários ou seus representantes podem certificar o estado de requisito correto.

- *Decomponíveis*: Devem possibilitar a implementação de cada requisito dentro das capacitações conhecidas e limitações do sistema e seu ambiente.

- *Necessários*: Cada requisito deve ser documentado como algo que o cliente realmente necessita ou algo que é requisitado para complementar um sistema externo ou um padrão.

- *Priorizáveis*: Devem permitir uma hierarquização de prioridade, indicando o quanto é essencial para uma dada versão do produto.

- *Sem ambigüidade*: Todos os leitores dos requisitos devem ter a mesma interpretação consistente dos requisitos.

- *Verificáveis*: Cada requisito deve possuir uma quantidade de testes ou tipos de verificação, como inspeção ou demonstrações.

Em termos de qualidade, uma Especificação de Requisitos deve ser(Kotonya, 1997):

- *Completa*: nenhum requisito ou informação necessária deve estar ausente.

- *Consistente*: Não existe conflito entre os requisitos de software ou com outros requisitos de mais alto nível de negócios ou sistemas.

- *Modificável*: Deve proporcionar revisões, quando necessário, e permitir manter um histórico de mudanças efetuadas de cada requisito.

- *Rastreável*: cada requisito de software que foi derivado deve possuir sua origem referenciada, de forma que os elementos de projeto, código fonte e casos de teste criados possam certificar a correta implementação dos requisitos.

Como documentação em Engenharia de Requisitos, o termo especificação é amplamente citado e definido. Utilizando novamente Holanda (1999), uma especificação significa "*descrição rigorosa e minuciosa das características que um material, uma obra, ou um serviço que deverão apresentar*". No contexto de Engenharia de Requisitos, uma especificação de requisitos de software define precisamente a funcionalidade e as capacidades que um sistema de software deve prover e quais condições, do domínio em que está inserido, deve respeitar (Wiegers, 1999).

Em termos de gerência de requisitos com qualidade, Wiegers (1999) destaca as seguintes: atividades:

- Gerenciar as alterações para os requisitos contratados;
- Gerenciar os relacionamentos entre os requisitos;
- Gerenciar as dependências entre os requisitos documentados e todos os outros documentos produzidos durante o processo.

As atividades de uma gerência efetiva de requisitos necessitam um processo definido para absorver as mudanças propostas e avaliar os custos potenciais dessas alterações e seus riscos (Wiegers, 1999).

Além da busca pelas condições listadas, a qualidade de requisitos está diretamente relacionada com as pessoas envolvidas no processo. Requisitos de alta qualidade emergem de uma efetiva comunicação e colaboração entre projetistas e clientes – uma parceria com seus direitos e deveres estabelecidos (Ambler, 2002).

Wiegers (1999) elege dez direitos e deveres que os clientes de software devem conhecer para atingir a expectativa de requisitos de alta qualidade (Tabela 3).

Tabela 3 - Direitos e Deveres do Cliente para atingir sua expectativa (Wiegers, 1999)

	Direitos		Deveres
1	Os analistas devem falar sua linguagem	1	Educar os analistas sobre seu negócio e jargões.
2	Os Analistas devem aprender sobre seus negócios e seus objetivos para o sistema.	2	Gastar o tempo necessário para expor os requisitos e esclarecê-los.
3	Os Analistas devem estruturar a informação coletada durante a	3	Ser específico e preciso quando fornecer informações sobre os

	elicitação de requisitos na forma de uma especificação de requisitos de software		requisitos do sistema
4	Os desenvolvedores devem explicar todo o produto que é criado a partir do processo de requisitos	4	Tomar decisões ágeis sobre requisitos quando necessário.
5	Os devenvolvedores devem tratar com respeito e manter uma atitude colaborativa e profissional durante as interações.	5	Respeitar as estimativas do desenvolvedor com relação a custos e viabilidade dos requisitos
6	Os desenvolvedores devem compartilhar as idéias e alternativas para a implementação dos requisitos.	6	Priorizar os requisitos e as funções do sistema.
7	Descrever características do produto que o farão de agradável e fácil uso.	7	Revisar requisitos e protótipos.
8	Ser apresentado a oportunidades para ajustar seus requisitos e permitir o reuso de componentes de software existente.	8	Comunicar mudanças nos requisitos de projeto tão logo aconteçam.
9	Ser informado com estimativas reais de custos, impactos e alterações quando da mudança de requisitos.	9	Seguir o processo definido para solicitar a mudança de requisitos.
10	Receber um sistema que vá de encontro às expectativas funcionais e de qualidade desejadas.	10	Respeitar os processos de Engenharia de Requisitos utilizados pelos desenvolvedores.

Nota-se que, seguindo a recomendação de Wiegers (1999), os requisitos de qualidade possuem uma estreita ligação com os atributos comunicação e colaboração.

Produtos de software de excelência constituem o resultado de um projeto bem concebido baseado em requisitos de alta qualidade.

2.5 Métodos e Técnicas em Engenharia de Requisitos

Os métodos e as técnicas fornecem suporte à Engenharia de Requisitos e aos seus processos, através da sistematização das suas atividades. Algumas vantagens da utilização de técnicas e métodos de requisitos são listadas por Young (2001):

- Desenvolve-se maior efetividade na implementação dos requisitos;
- A documentação é mais completa e detalhada;
- O código é testado com maior cuidado;
- O ambiente é mais bem controlado, possibilitando transições suaves de uma atividade para outra;
- Existe o conhecimento sobre onde os problemas estão ocorrendo durante o desenvolvimento.

Existe uma grande quantidade de métodos e técnicas disponíveis para utilização em Engenharia Requisitos. Normalmente tais métodos não são utilizados singularmente mas em conjunção, objetivando ampliar o escopo na obtenção dos requisitos reais necessários e sua manutenção. Um bom número destas técnicas tem sido desenvolvido visando a redução da taxa de mudanças de requisitos, ou pelo menos torná-las menos destrutivas para o projeto (Young, 01).

Os melhores métodos de Engenharia de Requisitos, de acordo com sua efetividade, foram relacionados por Jones (1998) apud Young (2001). Para isso, foram coletados dados de 1984 a 2000 de mais de 650 organizações. Dessas empresas, 150 estão classificadas na Revista Fortune 500, por volta de 30 são grupos governamentais/militares e os dados correspondem a um domínio de aproximadamente 9000 projetos. Estes dados exibem métodos e técnicas que podem ser utilizados em todo o ciclo de desenvolvimento, não sendo apenas na fase de análise de requisitos.

A lista dos melhores métodos de requisitos compreende:

- Inspeções Formais (projeto e código)
- JAD – Joint Application Design
- QFD – Quality Function Deployment
- Métricas de qualidade utilizando ponto de função.
- Métricas de qualidade utilizando classificação ortogonal da IBM
- Ferramentas para rastreamento de defeitos
- Garantia de qualidade efetiva
- Controle de configuração formal
- Levantamento da satisfação do usuário
- Plano de testes formais
- Ferramentas para estimativa de qualidade
- Ferramentas de testes automatizados.

O levantamento de Jones (1998) apud Young (2001) também mostra que os maiores problemas estão relacionados com requisitos incompletos e instáveis. Desta forma, o objetivo dos métodos e técnicas é aumentar a previsibilidade dos requisitos, tentando reduzir a porcentagem de requisitos instáveis em um projeto de software.

Jones (1998) apud (Young, 2001), destaca as seguintes técnicas e métodos como tendo valor positivo para minimizar a pressão de requisitos instáveis sobre o projeto:

- *Joint Application Design*

  JAD (*Joint Application Design*) é um método para desenvolvimento de requisitos de software, onde os representantes dos clientes e dos projetistas trabalham em colaboração com um facilitador, para produzir uma especificação de requisitos comum com a concordância dos dois lados. Comparado com antigos estilos de desenvolvimento de requisitos, onde os lados se encaravam como adversários, o JAD pode reduzir os requisitos instáveis pela metade. É uma excelente escolha para grandes contratos de software que objetivam um efetivo canal de interação entre projetistas e clientes.

- *Prototipação*

  Geralmente, as mudanças não ocorrem até que os clientes visualizem fisicamente o produto; desta forma, a construção de protótipos pode provocar algumas dessas mudanças no início do ciclo de desenvolvimento. Protótipos são, com freqüência, mais efetivos na redução de requisitos instáveis e podem ser combinados com outros métodos como JAD. Somente os protótipos são responsáveis pela redução dos requisitos instáveis a uma taxa entre 10% a 25%.

- *Casos de Uso*

  A técnica de casos de uso se integra normalmente ao modelo mental de *stakeholders* típicos, concentrando-se no conjunto de requisitos relacionados a uma seqüência específica de ações e respostas que um hipotético sistema de software deva executar. A vantagem de utilizar esta técnica é que mantém o processo de requisitos em um nível prático e minimiza a tendência de adicionar funções superficiais que não vão de encontro às expectativas dos *stakeholders*.

- *Conselho de controle de mudanças*

  O conselho de controle de mudanças de requisitos não é exatamente uma técnica, mas um meio que um grupo de gerentes, clientes e pessoal técnico

possuem para se encontrar e decidir quais mudanças devam ser aceitas ou rejeitadas. Trata-se de um canal formal de comunicação e avaliação de mudanças de requisitos.

Estes métodos foram destacados pela sua efetividade devido às seguintes características:

- *Joint Application Design*: tem a característica da conversação face-a-face e a colaboração, como fator para levantamento de requisitos.

- *Prototipação*: fornece, ao cliente, um produto tangível onde pode verificar concretamente seus requisitos e possíveis mudanças.

- *Casos de Uso*: representam a transcrição de um modelo mental do cliente, em termos de eventos que reflitam os requisitos do negócio e suas prioridades.

- *Conselho de controle de mudanças*: comunica a todos os integrantes, de forma colaborativa, o estado dos requisitos do projeto.

Estas características merecem destaque, pois elas estarão presentes, sob do ponto de vista de seu conceito, em todos os Métodos Ágeis, como será exposto nos capítulos seguintes.

3 MÉTODOS ÁGEIS

O objetivo deste capítulo é apresentar o Manifesto do Desenvolvimento de Software Ágil e detalhar seus valores e princípios.

Em seguida, são apresentados os principais Métodos Ágeis em prática e por fim uma proposição de um processo ágil genérico aderente às definições expostas.

3.1 Introdução

Ao final do encontro nos dias de 11 a 13 de Fevereiro de 2001, 17 pessoas envolvidas com desenvolvimento de software, proponentes e praticantes de processos de desenvolvimento leves e rápidos, elaboraram um documento, onde colocaram as expectativas comuns sobre esse novo ponto de vista sobre o processo de desenvolvimento de software. Surgiu o Manifesto para o Desenvolvimento Ágil de Software que é transcrito a seguir:

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

- *Indivíduos e interação 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 tendo valor os itens a direita, valorizamos mais os itens a esquerda.

Nós seguimos os seguintes princípios:

- ✓ *A maior prioridade é satisfazer o cliente, o mais breve possível e continuamente, com software funcional.*

- ✓ *Mudanças de requisitos são bem-vindas. Processos ágeis privilegiam as mudanças como vantagem competitiva para o cliente.*
- ✓ *Entrega de software funcional freqüente, preferencialmente em curtas escalas de tempo.*
- ✓ *Desenvolvedores e pessoas ligadas ao negócio trabalham em conjunto diariamente no projeto.*
- ✓ *Criar projetos cercados por pessoas motivadas, fornecendo ambiente e suporte necessários para acreditarem no trabalho feito.*
- ✓ *O mais eficiente e efetivo método de transmitir informações para e entre uma equipe de desenvolvimento é a comunicação verbal face-a-face.*
- ✓ *Software funcional entregue é a medida básica de progresso do projeto.*
- ✓ *Métodos Ágeis promovem desenvolvimento sustentado. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter uma tranqüilidade constante indefinidamente.*
- ✓ *Cuidado contínuo com a excelência técnica e um bom projeto acentuam a agilidade.*
- ✓ *Simplicidade – a arte de maximizar o quanto de trabalho não deve ser feito – é essencial.*
- ✓ *As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.*
- ✓ *Em intervalos regulares, a equipe reflete em como se tornar mais efetiva, ajustando-se em comum acordo.*

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas " (Beck et al., 2001).

3.2 Definição do Método Ágil de Desenvolvimento de Software

Apesar de grande avanço verificado, processos clássicos de desenvolvimento de software não respondem adequadamente em alguns domínios, onde a velocidade das mudanças de requisitos é uma constante.

Tentando construir uma solução para este tipo de problema, novos métodos começam a nascer, muitas vezes amparados nos métodos clássicos ou como uma particularização de processos atualmente em uso. Há poucos anos tem crescido rapidamente o interesse pela agilidade em processos. Os Métodos Ágeis, caracterizados como um antídoto para a burocracia ou uma licença para a simplificação, tem provocado grande interesse de todos sobre o horizonte do desenvolvimento de software (Highsmith, 2001).

O movimento não repudia processos, ferramentas, documentações, contratos ou planejamento, mas avalia quantos desses artefatos devem ser agregados ao projeto.

Os métodos clássicos impõem um processo disciplinado no desenvolvimento de software, com o objetivo de maior previsibilidade e eficiência, através de um processo detalhado com uma grande ênfase no planejamento, inspirado em outras disciplinas da engenharia (Cockburn, 2001a).

A maior crítica para estes processos, chamados de processos peso pesados ou conforme o termo de Highsmith (2001) - metodologias monumentais, é que eles se tornam burocráticos. A grande quantidade de artefatos e documentações necessária para seguir o processo acaba reduzindo sua velocidade, tornando-os incompatíveis com um domínio onde a agilidade é um requisito importante.

Enquanto os métodos clássicos focam a previsibilidade dos fatos, procurando manter um planejamento detalhado para grande parte do processo de desenvolvimento por um intervalo grande de tempo, os Métodos Ágeis procuram ser altamente adaptáveis e reagir rapidamente às mudanças de requisitos (Fowler, 2002a). Essa característica de buscar a previsibilidade dos requisitos e criar um longo planejamento baseado

nessa previsibilidade acaba provocando uma resistência natural a mudanças e conseqüente falta de qualidade no produto desejado pelo cliente.

Os Métodos Ágeis tentam ser processos que se adaptam e prosperam com as mudanças de requisitos, apoiados em um de seus princípios básicos: as mudanças de requisitos são bem vindas.

Uma outra característica importante que se destaca nos Métodos Ágeis é a sua forte orientação a pessoas e não a processos. Métodos tradicionais buscam o desenvolvimento dos processos, onde as pessoas são partes substituíveis e encaradas como recursos sob o ponto de vista do gerenciamento. Para os Métodos Ágeis, o desenvolvimento de software é um trabalho altamente criativo e profissional, inviabilizando este tipo de fundamentação. Por isso, os Métodos Ágeis rejeitam esta característica dos métodos tradicionais pois, para um processo altamente iterativo, requer uma equipe de desenvolvedores altamente efetivos e em sinergia (Boehm, 2002). Tratar pessoas como componentes do processo é um engano, pois as pessoas são altamente variáveis e não lineares, com particulares modos de reação a sucessos e falhas (Cockburn, 2001a).

O foco nas pessoas torna possível que os Métodos Ágeis sejam altamente adaptáveis; as pessoas decidem como conduzir seu trabalho, avaliar seus resultados e se auto-organizar no processo, não existindo um departamento ou entidade separada decidindo e planejando como o trabalho deve ser feito.

3.2.1 Os Valores

Os quatro valores fundamentais dos Métodos Ágeis, segundo a análise de Cockburn (2001a), são caracterizados como:

a. Indivíduos e interação mais que processos e ferramentas.

Métodos tradicionais normalmente caracterizam, em seu processo de desenvolvimento, pessoas como recursos em suas atividades. Esses

processos enfatizam as atividades, e não o desempenho de indivíduos em uma determinada atividade. Pensar em pessoas como recursos que possam ser somadas ou diminuídas, conforme o andamento do processo, pode se tornar inconveniente em um ambiente de trabalho criativo, como o projeto de software. Capacidade e perfil adequado é realçado em processos ágeis, onde as pessoas são partes fundamentais no processo e não são encaradas como componentes de fácil substituição.

b. *Software em funcionamento mais que documentação abrangente.*

Documentos contendo requisitos, análise ou projeto podem ser muito úteis no auxílio do trabalho dos desenvolvedores e ajudar no delineamento do futuro, mas código funcional, que tenha sido testado e verificado, revela grande quantidade de informações sobre a equipe, o processo de desenvolvimento e a natureza dos problemas e serem resolvidos. Os Métodos Ágeis focam principalmente na implementação, mas também atribui valor a modelagem desde que seja veloz.

c. *Colaboração com o cliente mais que negociação de contratos.*

Este valor enfatiza o sentido da colaboração verdadeira entre a equipe de desenvolvimento e seu cliente. A colaboração objetiva um sentido de comunidade, respeito, decisão conjunta, comunicação ágil e busca da interação entre os indivíduos. Este valor sugere que um processo de colaboração fortificado e construtivo faz dos contratos formais desnecessários. Nos casos em que o risco seja alto, a colaboração efetiva pode ser a solução. A idéia básica por trás deste valor é a satisfação geral do cliente, que é o principal objetivo nos Métodos Ágeis.

d. Responder a mudanças mais que seguir um plano

Requisitos mudam constantemente pela natureza iterativa e de incertezas no desenvolvimento de software, aliado a ambientes altamente competitivos de negócios e de tecnologia. Essas mudanças e instabilidades devem ser contempladas em um processo de desenvolvimento de software. Os Métodos Ágeis consideram que requisitos somados a uma meta rígida de desenvolvimento não podem ser fixados no início, mas que serão naturalmente alterados conforme o andamento do projeto e seu entendimento. O planejamento é utilizado em Métodos Ágeis, como também os mecanismos de contratos de requisitos, desde que sejam flexíveis e não busquem a previsibilidade dos fatos.

Um processo definido, mas flexível é a chave para prover a continuidade, auxiliando a organização a atingir os resultados desejados em ambientes de alta volatilidade.

3.2.2 Os Princípios

1) A maior prioridade é satisfazer o cliente, o mais breve possível e continuamente, com software funcional.

Um desenvolvimento iterativo fornece uma realimentação constante para a equipe de desenvolvimento, promovendo a definição de requisitos, arquitetura e soluções de programação. Schwaber (2002) afirma que incrementos de trabalho, compostos de sistema funcional, criam uma relação direta entre progresso e entrega do produto e provê mecanismos para os *stakeholders* fornecerem e ratificarem os requisitos sobre um produto real.

O software funcional gera dados de comunicação e planejamento especialmente entre desenvolvedores e cliente, no que tange ao entendimento das necessidades, esclarecimento e definição dos requisitos do sistema.

A gerência é auxiliado, pois as entregas fornecem marcos para o controle do estado do projeto e um guia para o planejamento do projeto.

2) *Mudanças de requisitos são bem-vindas. Processos ágeis privilegiam as mudanças como vantagem competitiva para o cliente.*

Requisitos de software tendem a mudar freqüentemente e radicalmente enquanto o esforço de desenvolvimento acontece. Diferentes Métodos Ágeis propõem diferentes modos de reagir à mudança dos requisitos, mas basicamente todos direcionam para a entrega de software funcional, o mais breve possível e dentro de um processo iterativo (Fowler, 2002).
A utilização de uma política rígida de mudanças de requisitos, em função da complexidade de uma mudança requerida pode desencorajar os clientes, agregando requisitos inválidos para o negócio e resultando em produtos sem o valor desejado pelo negócio.
O contrato tradicional de requisitos entre desenvolvedor e cliente tende a criar um relacionamento desequilibrado em termos de vantagens no ciclo de vida do projeto; facilitar a mudança acaba sendo mais efetivo que tentar preveni-la.

3) *Entrega de software funcional freqüente, preferencialmente em curtos intervalos de tempo.*

A entrega de software funcional, já discutida no princípio 1, favorece uma rápida realimentação, fornecendo dados para correção e direcionamento dos requisitos do produto.
A gerência dos requisitos se torna mais eficiente, criando um ambiente para rápidas alterações e testes dos requisitos.
Um ciclo rápido de entregas proporciona, a todos os membros envolvidos no projeto, a avaliar e entender os requisitos de um projeto em crescimento (Astels, 2002).

4) *Desenvolvedores e pessoas ligadas ao negócio trabalham em conjunto diariamente no projeto.*

A colaboração dos *stakeholders*, na integração direta com o projeto, pode apoiar fortemente todo o processo da Engenharia de Requisitos, prevenindo os problemas causados pela falta de comunicação (Rising, 2002).
Os requisitos são analisados e discutidos sob os dois pontos de vista negócio (cliente) e tecnologia (desenvolvedor).
A substituição de um documento formal, constituído de uma coleção de requisitos detalhados, por requisitos com um alto nível de abstração, instáveis e apoiados pela forte integração do cliente no processo de desenvolvimento, enfatiza um contínuo comprometimento e co-autoria no projeto de software.

5) *Criar projetos cercados por pessoas motivadas, fornecendo ambiente e suporte necessários para acreditarem no trabalho feito.*

Sob o ponto de vista de requisitos, existe um benefício indireto, se fatores sociais como amizade, talento, capacidade e comunicação forem enfatizados. Um ambiente com estas características proporciona uma grande fonte de oportunidade de obter produtividade, agregando a utilização e desenvolvimento das capacidades individuais, formando uma comunidade sincronizada na solução dos problemas e focada nos objetivos (Cockburn,01). A qualidade dos requisitos emerge em um ambiente com essas características.

6) *O mais eficiente e efetivo método de transmitir informações para e entre uma equipe de desenvolvimento é a comunicação verbal face-a-face.*

A comunicação face-a-face diminui a necessidade de documentação formal. O conhecimento e as necessidades são transferidos com maior velocidade, facilitando o aprendizado e a realimentação dos requisitos.
O conhecimento tácito não pode ser transferido diretamente da mente de uma pessoa para o papel. Pode ser transferido, com mais efetividade, através de um relacionamento com a pessoa que possui o conhecimento. A razão para isto, é que o conhecimento não é composto apenas pelos fatos em si, mas pelo relacionamento entre os fatos – isto é, como as pessoas combinam certos fatos para demonstrar uma situação específica (Fowler, 2002).
A documentação não deve ser totalmente descartada, mas deve-se procurar o balanceamento entre documentação e conversação para atingir o entendimento.

7) *Software funcional entregue é a medida básica de progresso do projeto.*

O software funcional, como métrica de progresso, apóia principalmente a gerência de projeto, tanto quanto o estado atual do projeto (Astels, 2002). Também pode ser encarado como uma métrica de qualidade para os requisitos, pois permite avaliar se os requisitos do cliente estão sendo cumpridos e implementados.

8) *Processos ágeis promovem desenvolvimento sustentado. Os patrocinadores, os desenvolvedores e os usuários devem ser capazes de manter uma harmonia constante indefinidamente.*

Focar nas pessoas, criando uma responsabilidade social, pode manter a produtividade da equipe afetando positivamente a satisfação no trabalho, proporcionando pessoas alertas e criativas influindo em todo o processo de software.

9) *Cuidado contínuo com a excelência técnica e um bom projeto acentuam a agilidade.*

Métodos Ágeis adotam e encorajam mudanças de requisitos enquanto o código está sendo escrito (Beck, 1999). Portanto, o projeto não pode ser encarado apenas como uma atividade iniciadora para ser complementada com a construção.O projeto é uma atividade contínua que deve ser executada por toda a vida útil do processo. Assim, toda a iteração deve possuir esforço de projeto, através da re-fabricação (*refactoring*) para cumprir os requisitos de qualidade, tornando o projeto e suas implementações melhores, facilitando a manutenção de forma a apoiar a análise, testes e refinamento dos requisitos de software.

10) Simplicidade – a arte de maximizar o quanto de trabalho não deve ser feito – é essencial.

Os desenvolvedores devem somente implementar aquelas funções que foram aceitas pelos clientes e focar em sua real necessidade e nada mais (Beck, 1999). Apesar de o conceito de simplicidade ser um pouco subjetivo, ele deve ser satisfeito sob os dois pontos de vistas envolvidos: o do desenvolvedor e o do cliente.

11) As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.

Equipes que atingem a capacidade de se auto-organizar e, conseqüentemente, se adaptar com equilíbrio a mudanças de requisitos e eventos externos, conseguem atingir uma melhor excelência no produto de software sob as ópticas de projeto, arquitetura e requisitos (Cockburn, 2001).

12) Em intervalos regulares, a equipe reflete em como se tornar mais efetiva, ajustando-se em comum acordo.

A efetividade reflete a capacidade de absorção de mudanças e produtividade evidenciadas a cada ciclo iterativo (Cockburn, 2001). Essas qualidades são direcionadas pela característica da adaptabilidade do processo e da equipe à dinâmica das perturbações de origem externa e interna com relação ao processo ideal de desenvolvimento do produto.

3.3 Principais Métodos Ágeis

Uma pesquisa realizada por Charette (2002), com 200 gerentes IS/IT, sobre as utilizações de métodos clássicos e Métodos Ágeis de desenvolvimento de software, revelou que 54% possuem algum tipo de experiência com Métodos Ágeis. A pesquisa apresenta os seguintes números:

- Empresas estão distribuídas na seguinte proporção: 33% América do Norte, 20% Europa, 10% Austrália, 8% Índia, 8% Ásia e 21% restantes divididos pelo Oriente Médio, África e América do Sul.

- Com relação à atividade, ficaram divididas em 39% empresas de software, 11% finanças, 9% consultoria, 6% governamental, 5% telecomunicações, 3% bancos, 2% utilidades, 1% transportes e outras 15 categorias com 24%.

- A pesquisa também categorizou as empresas por receita: 13% com mais de US$ 1 bilhão, 17% entre US$100 milhões e US$ 1 bilhão, 33% entre US$5 milhões e US$100 milhões e 37% com menos de US$5 milhões.

Conforme a figura 9, os Métodos Ágeis mais utilizados são: Extreme Programming 38%, Feature-Driven Development 23%, Adaptive Software Development 22% e Dynamic Systems Development com 19%. Segundo a pesquisa, muitos dos gerentes

responderam que utilizam ou já utilizaram mais de um método nas suas organizações (Charette, 2002).

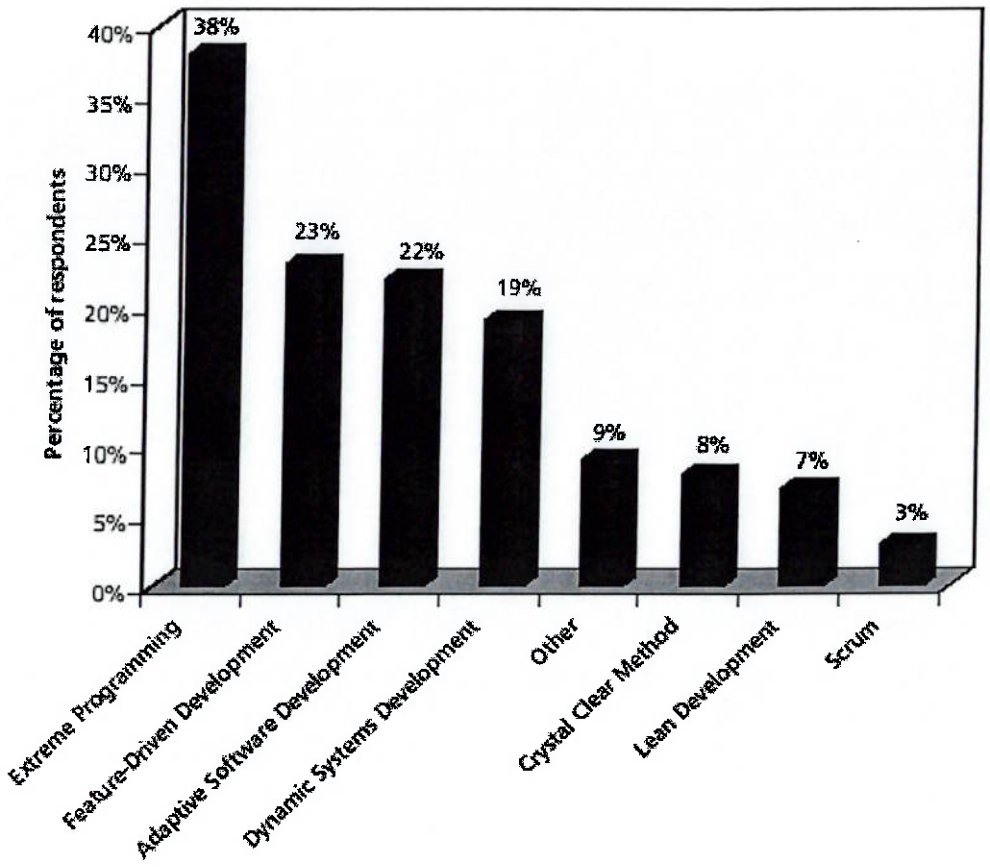

Figura 9 - Métodos Ágeis utilizados na Indústria de Software (Charette, 2002)

Segundo Cockburn (2001), o encontro, onde se definiram os valores e princípios do Manifesto da Agilidade, foi promovido por vários metodologistas que buscavam uma estrutura comum entre os processos que praticavam. Os métodos que estavam formalmente representados no encontro eram: Extreme Programming (XP), Adaptive Software Development (ASD), Dynamic System Development Method (DSDM), Scrum, Crystal e Feature Driven Development (FDD). Esses métodos, por serem precursores e estarem entre os mais utilizados (Charette, 2002), serão detalhados a seguir.

3.3.1 Extreme Programming

Extreme Programming provavelmente hoje é o Método Ágil que possui maior destaque neste tipo de desenvolvimento. Nascido de um projeto para o sistema Chrysler Comprehensive Compensation, codinome C3, pela necessidade de entregar o software o mais rápido possível, Kent Beck, Ward Cunningham e Ron Jeffries criaram um método que explorou os extremos de certas práticas de desenvolvimento, criando o método chamado Extreme Programming (XP).

O XP é baseado em quatro valores fundamentais (Beck, 1999):

- Comunicação
- Avaliação
- Simplicidade
- Coragem

Basicamente, as práticas recomendadas pelo método são baseadas em práticas antigas e testadas, algumas até esquecidas, em planejamento de processos. O XP utiliza-as de uma forma colaborativa e iterativa, centrando as pessoas como foco principal do método.

O XP define como suas atividades principais (Astels, 2002):

- *Trabalhar com seus clientes.* O cliente deve fazer parte da equipe de desenvolvimento, pois a ele é atribuída a capacidade de resolver as dúvidas e tomar decisões relativas à prioridade de recursos e riscos.

- *Metáforas.* São utilizadas para conceitos abstratos e difíceis, fornecendo uma estrutura de conceitos e terminologias comuns para ser compartilhada e entendida por todos os membros da equipe.

Planejamento. É efetuado o planejamento a cada iteração (entre 1 a 3 semanas), evitando intervalos mais longos, diminuindo a imprecisão do planejamento conforme os requisitos são alterados e os riscos vão sendo identificados. Caso existam dúvidas ou inexperiência com relação às estimativas para o planejamento da iteração, recorre-se a uma exploração rápida, em termos de programação, sobre uma ou mais *user stories* que a equipe de desenvolvimento julgue necessária. Essa exploração chamada de *spike*, possui como objetivo principal fornecer uma estimativa mais acertada para o planejamento da iteração e não apenas para o código, sendo encarada como uma rápida prototipação sob a perspectiva da equipe de desenvolvimento.

- *Reuniões Rápidas*. São realizadas reuniões diárias, curtas e produtivas para a avaliação das atividades completadas no dia anterior, além dos problemas e dos obstáculos encontrados.

- *Testes*. É dada uma grande ênfase nos testes. Ao escrever o código de uma função, o seu código de teste deve também ser escrito. Todos os testes devem sempre ser executados para qualquer alteração daquela função.

- *Simplicidade*. Manter o projeto simples e claro fornece condições de deixá-lo ágil e maleável, possibilitando a acomodação de novos requisitos conforme as iterações prosseguem.

- *Programação em Pares*. Toda linha de código deve ser escrita por um par de programadores, agrupados em piloto e parceiro. O piloto é responsável pela escrita dos códigos, testes, integração ou re-construção (*refactoring*) do código, enquanto o parceiro observa o código, tendo em mente os objetivos estratégicos, encontrando erros e oferecendo idéias e sugestões.

- *Utilização de Padrões*. Adotar um conjunto de padrões de código no nível da equipe.

- *Código Coletivo.* O código existente no projeto é da propriedade de todos os membros da equipe e qualquer membro pode manipular um código existente.

- *Integração Contínua.* As funções concluídas ao final de cada tarefa devem ser integradas ao produto de forma contínua, minimizando conflitos e obtendo convergência para um produto funcional.

- *Re-Construção (Refactoring).* Promover a alteração do sistema em sua estrutura interna sem alterar seu comportamento externo, de forma disciplinada, objetivando a clareza do código e seu melhoramento contínuo.

- *Entregas em incrementos pequenos.* Colocar as funções nas mãos dos verdadeiros usuários, buscando avaliação e identificação de novos requisitos o mais breve possível.

- *Semana de 40 horas.* Promover a qualidade de vida dos membros da equipe, evitando grandes jornadas seguidas de trabalho.

- *Adotar as alterações.* A equipe de desenvolvimento deve permanecer flexível, aceitando normalmente as alterações de requisitos.

O método Extreme Programming divide as equipes de trabalho, dentro do processo de desenvolvimento, em duas famílias: a equipe de desenvolvimento e a equipe do cliente.

- *Equipe do cliente*: Os membros definem quais necessidades funcionais devem ser agregadas ao sistema e também são responsáveis pelos seus testes e pela aceitação. A equipe do cliente deve ser mais ampla que um grupo de usuários do sistema, sendo necessárias várias habilidades dos seus integrantes, para que possam atuar de forma constante e cooperativa no processo. Os papéis necessários para essa equipe são (Astels, 2002):

- Contadores de Histórias: Os contadores de histórias são pessoas que possuem especialização no domínio onde o software será construído e são responsáveis por escrever as *user stories* (correspondem aos casos de uso com um escopo simplificado e reduzido). Os contadores de histórias são as pessoas a quem os membros da equipe de desenvolvimento recorrem quando necessitam de algum esclarecimento sobre os requisitos do sistema.

- Aceitantes: Podem ser os contadores de histórias ou pessoas que agem em seu nome para executar os testes de aceitação. Os aceitantes garantem que cada *release* corresponda às expectativas dos contadores de histórias.

- Patrocinadores: Os patrocinadores, ou proprietários do ouro, como Beck (1999) literalmente designa, são os provedores de recursos para o projeto, garantindo que o projeto tenha pessoas, financiamento e equipamentos adequados. Correspondem às pessoas que buscam valor no projeto para seu negócio.

- Planejadores: Os planejadores são as pessoas que vêem as necessidades de distribuir as funções pretendidas do sistema. São as pessoas que conhecem os ciclos do negócio envolvido e os seus relacionamentos. Existe um equivalente na equipe de desenvolvimento que é o acompanhador.

- Chefe: O chefe é a pessoa que lidera o sistema, é responsável por verificar se existe um equilíbrio entre as forças e recursos participantes no projeto. Ele garante que os trabalhos sejam concluídos e os caminhos organizacionais estejam livres. Acaba atuando nas duas equipes, garantindo para o patrocinador que o projeto obtenha o valor esperado para o negócio.

- *Equipe de Desenvolvimento*: É constituído por responsáveis por estimar tarefas, ajudar os clientes a se conscientizarem sobre as conseqüências de suas decisões, adaptar seus processos, construir e entregar o sistema. Os papéis necessários para esta equipe são:

    - Técnico: Com um perfil experiente, ele agrega sua experiência ao projeto e é responsável por garantir que o processo flua normalmente. Ele identifica se existe a necessidade da adaptação do processo e promove a cooperação da equipe nesse intuito.

    - Acompanhador: Responsável por avaliar as estimativas feitas pela equipe de desenvolvimento e medir o tempo real para concluir determinado trabalho, tornando-se uma realimentação para as futuras estimativas. Também monitora a velocidade da entrega de software funcional para o cliente e as mudanças requeridas, sendo responsável pela comunicação das alterações a toda equipe de desenvolvimento.

    - Facilitador: Pessoa com perfil comunicativo e capacidade de relações interpessoais. Seu papel é facilitar o relacionamento entre as duas equipes. Quando existe conflito entre as equipes em função de alguma distorção na comunicação, seu papel é encontrar situações conciliadoras e comunicar de forma eficaz o que é necessário para atingir o resultado.

    - Arquiteto: Produz a arquitetura e faz seu *refactoring* conforme a necessidade. Responsável por escrever os casos de teste de mais alto nível para cobrir a arquitetura necessária para dar suporte à iteração, também observa o código produzido, agregando sua experiência no *refactoring* do código e auxiliando os desenvolvedores.

Uma das principais características do Extreme Programming é definir um alto valor para a satisfação do cliente. Para atingir esse objetivo, o XP busca a integração do

cliente na equipe e proporciona realimentações constantes e freqüentes sobre o produto entregue. Outras fontes importantes de realimentações são os testes, executados já nas fases iniciais da iteração e com a participação ativa do cliente. Desta maneira, entrega contínua de software funcional, colaboração, intensa comunicação e participação efetiva do cliente praticados no XP desenvolvem condições favoráveis que proporcionam os programadores a responderem de forma rápida e efetiva às alterações de requisitos do cliente.

A seguinte figura define o fluxo do processo de desenvolvimento em Extreme Programming (Wells, 2002).

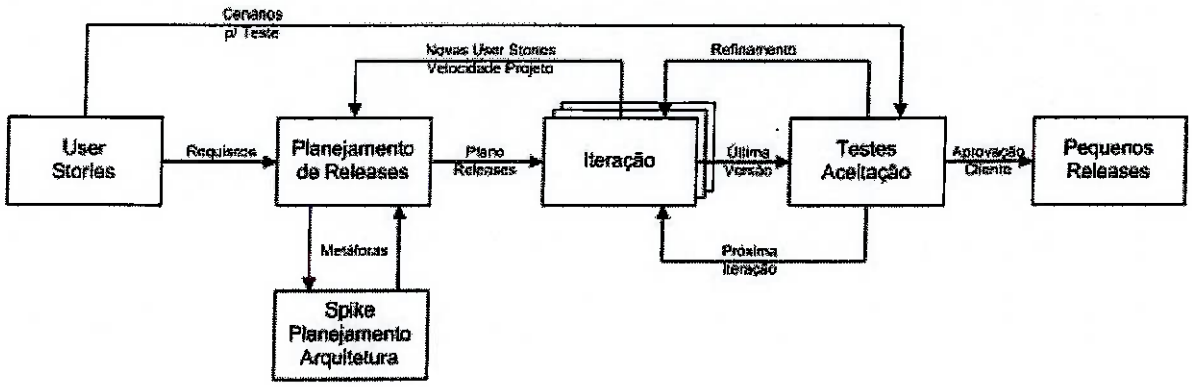

Figura 10 - Processo Extreme Programming - adaptado (Wells, 2002)

3.3.2 Feature Driven Development

O método Feature Driven Development (FDD) foi inicialmente desenvolvido por Jeff De Lucca e Peter Coad e, de forma semelhante a outras metodologias evolucionárias e adaptativas, foca em ciclos rápidos de desenvolvimento com entrega de funções tangíveis para os clientes. Baseia-se em cinco atividades principais (Coad, 1999):

- *Desenvolver um modelo global.* Os clientes apresentam um escopo de alto nível do sistema e seu contexto através de *walk-throughts*. A equipe de

desenvolvimento, em conjunto com os clientes, constrói um esqueleto do modelo do software necessário. Grupos especializados das equipes do cliente e do desenvolvedor adicionam, no modelo, características pertinentes conforme seus pontos de vista, fornecendo uma visão global a todos os envolvidos.

- *Construir uma lista de funções.* As equipes identificam as funções e as organizam, agrupando-as, hierarquizando-as, priorizando-as e atribuindo pesos relativos.

- *Planejar por funções.* Através da lista de funções hierarquizada, priorizada e com pesos atribuídos, são estabelecidos marcos para as atividades iterativas do processo que são: Projetar através das funções e Construir por funções, normalmente compreendidos em ciclos curtos de duas semanas.

- *Projetar através das funções.* As classes comuns do projeto são identificadas e os responsáveis pelas classes designados; as equipes, de desenvolvedor e cliente, detalham as funções através de diagramas de seqüência. Os responsáveis pelas classes, que são membros da equipe de desenvolvimento, escrevem seus modelos e seus métodos, com a inspeção do modelo sendo efetuada pela equipe de desenvolvimento, de forma geral.

- *Construir por funções.* Os responsáveis pelas classes geram os casos de teste unitários e das classes. Após o código ser implementado com sucesso e inspecionado, a equipe faz a inspeção das classes envolvidas. Caso sejam aprovadas, todas as classes de uma determinada função são promovidas para o produto principal, focando a entrega de versões de software funcionais de valor para o cliente.

O FDD é um processo altamente iterativo e orientado a resultado que objetiva habilitar equipes a desenvolver software de valor para o usuário, rapidamente sem

comprometer a qualidade do produto. Trata-se de um processo que busca focar as pessoas em detrimento da documentação.

O FDD foi projetado para equipes pequenas, mas pode ser escalável para grandes equipes. A equipe consiste de pessoas que possuem experiências e competências distintas de modo que se completem de forma colaborativa. Nesse sentido, pessoas de desenvolvimento e pessoas ligadas ao negócio devem trabalhar em equipe (Coad, 1999).

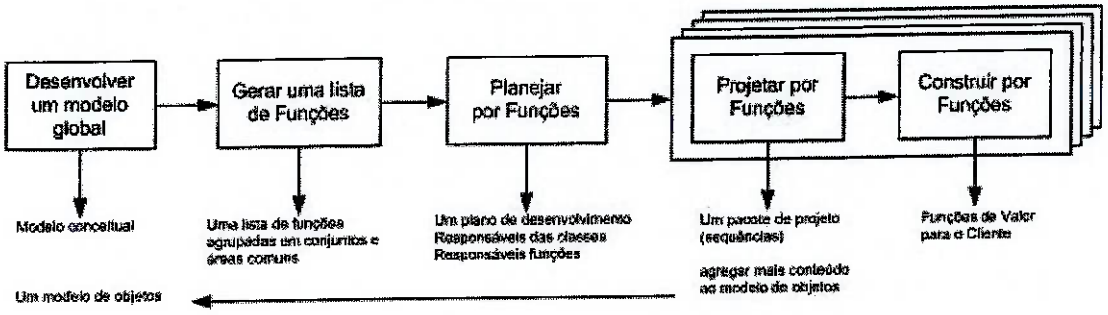

Figura 11 - Processo Feature Driven Development - adaptado de (Coad, 1999)

3.3.3 Adaptive Software Development

Desenvolvido por Jim Higsmith, o método Adaptive Software Development (ASD) foi criado para projetos que são caracterizados pela sua alta velocidade de entrega de produtos funcionais, mudanças freqüentes e incerteza de requisitos. O foco requerido no produto necessita de um método iterativo, porque o resultado de cada iteração contribui fortemente para o direcionamento do processo (Cockburn, 2001). O método segue um ciclo de vida dinâmico baseado em três principais atividades:

- *Especular:* Um planejamento requer um certo grau de certeza para atingir os objetivos desejados. Como Highsmith (2000) afirma, adaptação é necessária no novo mundo, onde mudanças, improvisação e inovação reinam. Em tal ambiente é difícil planejar para gerenciar projetos em direções inovadoras.

Esta é a razão pela qual Highsmith (2000) prefere o conceito especulação. A especulação consiste da iniciação do projeto e do planejamento do ciclo. A especulação não elimina o planejamento, mas as incertezas são agregadas, através do encorajamento da exploração e experimentação. Por isso, adotam-se ciclos curtos de entrega de software funcional e iteração das atividades para atingir esse objetivo.

- *Colaborar:* Quando aplicações complexas são desenvolvidas em um ambiente turbulento, o volume do fluxo de informação e a demanda pelo conhecimento dos requisitos são altos. Este tipo de ambiente necessita de alta colaboração para o gerenciamento efetivo do ciclo de vida do projeto. Em ASD, a colaboração e o paralelismo são as palavras chave para desenvolver componentes funcionais.

- *Aprender:* Aprendizado vem como resultado da realimentação capturada das revisões de qualidade do produto. Os resultados levantados de cada iteração realinham as atividades, objetivos e expectativas das equipes.

O ASD é caracterizado pela capacidade de absorção da mudança de requisitos, reavaliação contínua do processo e intensa colaboração entre desenvolvedores, testadores e clientes. Em ambiente de incertezas, pessoas colaborativas são necessárias, sendo a comunicação o canal mais importante para identificar problemas e encorajar as equipes na solução (Highsmith, 2000)

Cada iteração, chamada de ciclo adaptativo, deve possuir as seguintes propriedades:

- Baseada na visão do projeto e guiada pela missão, não como um destino fixo, mas como limite do projeto;

- Dirigida a resultado, por exemplo, um componente construído no lugar de tarefa cumprida;

- Limitada no tempo;

- Considerada como uma pequena parte de um conjunto maior de iterações;

- Direcionada pelo risco;

- Tolerante a mudanças, encaradas como uma oportunidade para aprender o obter vantagem competitiva.

A figura12 exemplifica o processo em ASD.

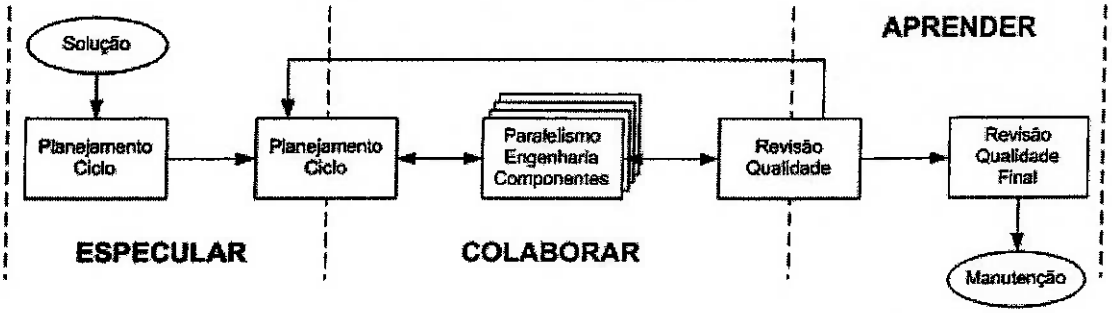

Figura 12 - Processo Adaptive Software Development - adaptado de (Highsmith, 2000)

3.3.4 Dynamic System Development Method

O Dynamic Systems Development Method (DSDM) surgiu na Inglaterra em 1994, em um consórcio de companhias do Reino Unido que visavam construir um RAD agregado ao desenvolvimento iterativo. O ciclo de vida do Dynamic System Development Model (DSDM) é iterativo e incremental, consistindo de cinco fases (Easton, 2002):

- Estudo de viabilidade

- Estudo do negócio

- Iteração do modelo funcional

- Iteração do projeto e construção

- Implementação.

O método inicia com estudo de viabilidade e do negócio, verificando se o DSDM é apropriado ao projeto. O estudo do negócio é uma coleção de *workshops* curtos para entender a área de negócio onde o desenvolvimento acontecerá. Como resultado, propõe o esboço da arquitetura do sistema e de um plano de projeto geral.

O restante do método é formado por três ciclos sobrepostos (Easton, 2002):

- Ciclo do modelo funcional, que produz documentação de análise e protótipos;

- Ciclo de projeto e construção, que cria o sistema funcional para o usuário;

- Ciclo de implementação, que controla a implantação do sistema para uso operacional.

As atividades nas iterações dos ciclos são baseadas nos seguintes princípios:

- O envolvimento do cliente é imperativo.

- As equipes devem possuir capacidade de decisão.

- A entrega de produtos para o cliente é freqüente.

- Critério essencial de aceitação de produtos é o valor para o cliente sob o ponto de vista do propósito do negócio.

- Desenvolvimento iterativo e incremental é necessário para atingir os objetivos da solução para o negócio.

- Todas as mudanças durante o desenvolvimento são aceitas.

- Requisitos de alto nível são delineadores para o projeto.

- Testes e integrações ocorrem por todo o ciclo de vida.

- É essencial o clima de cooperação e colaboração entre todos os *stakeholders* envolvidos no projeto.

A figura 13, exibe o processo em DSDM.

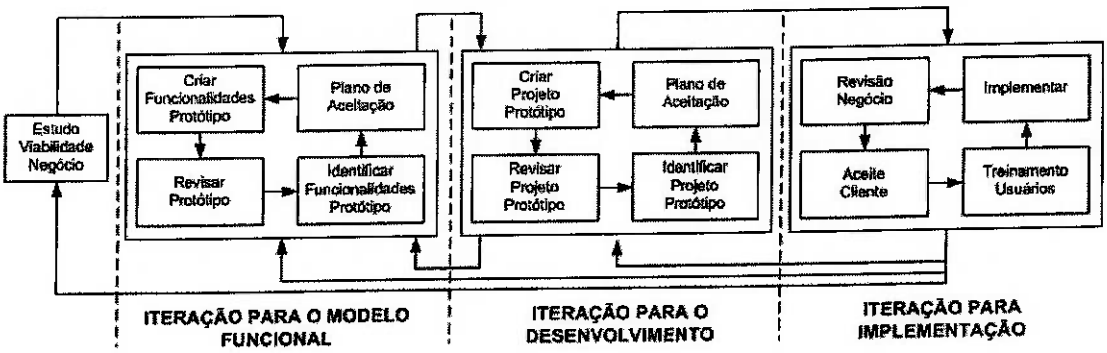

Figura 13 - Dynamic Systems Development Method - adaptado de (DSDM, 2001)

Seus ciclos são focados em princípios de desenvolvimentos ágeis, como a participação ativa do cliente, entregas freqüentes, programa de testes nas iterações, ciclos curtos nas iterações (duas a seis semanas). O método busca dar ênfase na qualidade e adaptatividade do método para requisitos variáveis e inconstantes.

O DSDM se destaca por basear-se na infra-estrutura de metodologias clássicas e mais maduras, porém seguindo os princípios e características dos Métodos Ágeis.

3.3.5 Crystal Family

Diferentes projetos requerem diferentes métodos e Crystal pretende fornecer uma solução para essas diferentes necessidades.

Crystal é um processo orientado a pessoas para desenvolvimento de software, focado na idéia de que são as pessoas que fazem o sucesso do projeto. Ferramentas, produtos e processos existem apenas para apoiar o componente humano e as pessoas e os métodos devem ser centrados na comunicação.

Outra característica do conjunto de métodos Crystal é o reconhecimento das diferenças culturais humanas, encorajando o aspecto da alta tolerância, que pode proporcionar a equipe a incorporar partes de outros métodos que julgue necessário, por exemplo Extreme Programming (XP) (Cockburn, 2001a).

Crystal Family classifica seus métodos em Clear, Yellow, Orange, Red, Maroon e Violet, em função do tamanho e da criticidade do sistema a ser desenvolvido. Projetos de grande porte necessitam mais coordenação e metodologias mais pesadas do que projetos de pequeno porte. Cada um desses métodos é projetado para um determinado tamanho de equipe, de modo a orientar sobre suas necessidades de organização e estratégias de comunicação.

A figura 14 indica o potencial de perda causado por uma falha de sistema, classificando em: Confortável (C), Custo Moderado (D), Custo Essencial (E) e Custo de Vidas (L). Para cada nível de criticidade existe um número máximo de pessoas recomendado no processo. Assim, por exemplo, D40 representa um projeto com um máximo de 40 pessoas envolvidas entregando, um sistema com uma criticidade máxima de um custo moderado.

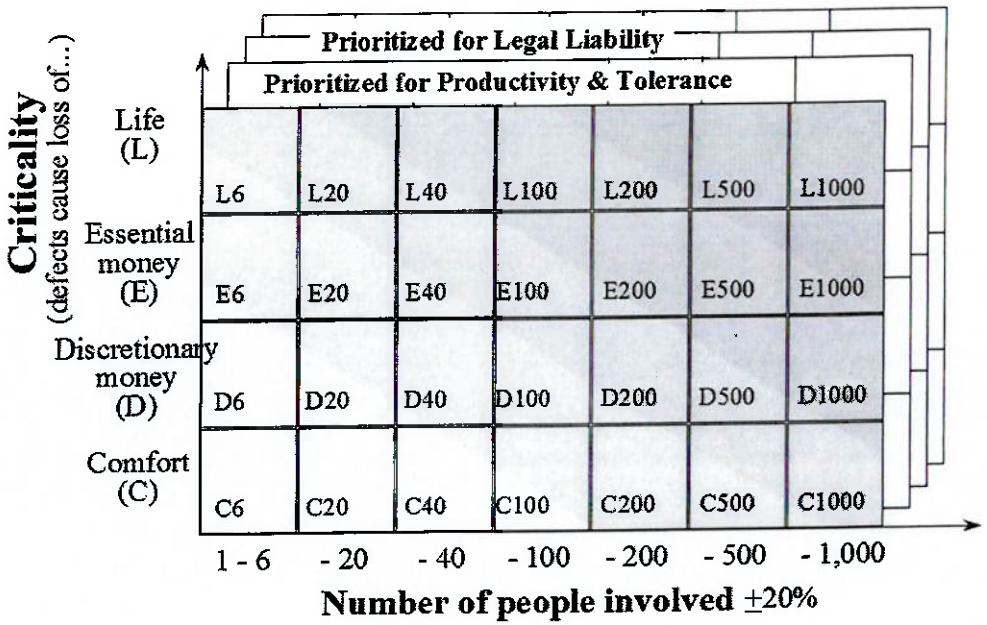

Figura 14 - Classificação Crystal Family (Cockburn, 2001a)

Apesar de todos os métodos que compõe a Cristal Family seguirem os mesmos valores e princípios, será destacado o método Clear pela sua maior similaridade com os Métodos Ágeis listados, no presente trabalho, e em função de ser talhado para pequenas projetos e equipes concentradas em um mesmo ambiente de trabalho.

As seguintes práticas são aplicadas durante o processo (Cockburn, 2001a).

- Produto funcional em entregas incrementais regulares
- Medição de progresso do projeto baseado no software entregue
- Envolvimento direto do cliente
- Automação de testes de funções
- *Workshops* sobre o produto e refinamento do método no início e na metade de cada incremento.

As seguintes atividades estão envolvidas no processo (Cockburn, 2001a):

- *Simulação (Staging):* Corresponde ao planejamento da próxima iteração do sistema. Deve ser concebido para produzir *releases* do produto funcional em cada três ou quatro meses no máximo. A equipe seleciona os requisitos a serem implementados nos incrementos e planejam suas entregas.

- *Revisão e Investigação.* Cada incremento contém várias iterações. Cada iteração inclui as seguintes atividades: construção, demonstração e investigação dos objetivos do incremento.

- *Monitoração.* O progresso é medido conforme as entregas de produtos funcionais durante o processo de desenvolvimento.

- *Paralelismo.* Múltiplas equipes podem proceder com o paralelismo das atividades desde que a monitoração da estabilidade do processo forneça resultados para isso.

- *Refinamento do Processo.* A cada incremento, a equipe de projeto pode aprender e utilizar o conhecimento obtido para refinar o processo para o próximo incremento.

Pessoas com perfis distintos são necessárias nos vários papéis que o método exige: o patrocinador, o analista-programador sênior, o analista programador e o cliente, que deve ser um membro da equipe. A esses papéis são atribuídas responsabilidades como: elicitação de requisitos, análise, programação e coordenação do projeto, que normalmente fica a cargo de um especialista no domínio do negócio, . Tanto pessoas com perfil técnico como de negócios devem estar presentes nestes papéis e na equipe.

O método busca a entrega freqüente de software funcional através de um desenvolvimento incremental. O intervalo de tempo de um incremento recomendado

é entre um a três meses. As revisões ao final das iterações são fortemente enfatizadas no Crystal.

Uma das vantagens de um método com iterações curtas é a possibilidade do surgimento das dúvidas e dos problemas nas fases mais prematuras do projeto, facilitando a correção e o ajuste do projeto na direção correta. Para que isto ocorra, o método necessita que os participantes monitorem e melhorem o processo, atribuindo ao processo a qualidade da adaptabilidade.

O Crystal Clear descreve o acompanhamento do progresso através de marcos que consiste de entrega de software ao invés de algum tipo de documentação formal. A documentação de projeto é definida como necessária no método, mas seu conteúdo e forma não são detalhados no método, atribuindo aos membros das equipes a decisão sobre a documentação que acharem necessária ao projeto.

O usuário também é envolvido no método Crystal Clear; seus pontos de vista na entrega do sistema funcional a cada iteração são encarados como atividades de revisão no projeto.

O método também utiliza *workshops* e reuniões rápidas para o refinamento do produto e do processo, que normalmente são executadas no início e na metade de cada iteração.

A figura 15 apresenta o processo Crystal de forma genérica.

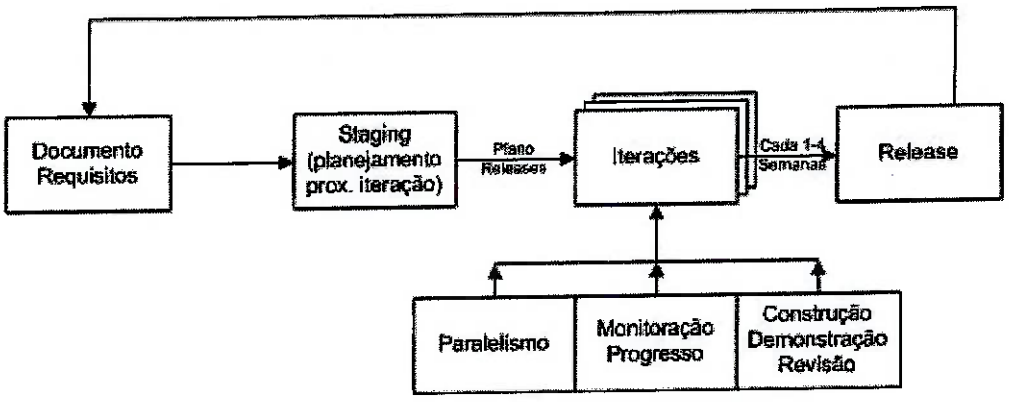

Figura 15 - Processo Crystal Family - adaptado de (Cockburn, 2001a)

3.3.6 Scrum

O método Scrum foi referenciado pela primeira vez como um método desenvolvimento de um produto no artigo "The New Product Development Game (Harvard Business Review 86116: 137-146, 1986)" e mais tarde detalhado em "The Knowlodge Creating Company (Oxford University Press, 1995) Ikujiro Nokata, Hirotaka Takeuchi". Baseado em suas idéias e com a colaboração do Departamento de Pesquisas Avançadas da DuPont, Scrum foi oficialmente descrito e apresentado no encontro do grupo de Gerenciamento de Objetos em 1995 (Beedle, 2001).

O Scrum possui dois conceitos básicos: adaptabilidade e a autonomia da equipe. A adaptabilidade refere-se à capacidade de equilíbrio que a equipe demonstra com relação a distúrbios de origem externa ao processo. A autonomia se refere às equipes capazes de decidir seus rumos no projeto. Os gerentes do projeto determinam o trabalho a ser feito, mas as equipes decidem de que forma será executado o trabalho a cada incremento. A tarefa do gerente é proporcionar liberdade necessária para que as equipes possam trabalhar, identificar e remover os pontos de ineficiência que restrinjam suas capacidades na produtividade no projeto. As equipes praticantes do Scrum fazem diariamente reuniões curtas e rápidas, proporcionando uma realimentação direta a todos sobre o andamento do projeto, reduzindo uma grande quantidade de burocracia gerencial (Janoff, 2000).

Na opinião de Schuwaber (2001), observar a interação entre os membros da equipe é a melhor forma para entender a complexa interação entre pessoas, tecnologia, necessidades, interesses e satisfação. A união das pessoas acaba promovendo a colaboração e participação efetiva no projeto, promovendo tanto o grupo como o indivíduo a trabalhar em equipe.

O Scrum apóia-se no desenvolvimento iterativo e adaptável. Os projetos são divididos em *sprints* que formam as iterações, cada uma com aproximadamente um mês de duração.

Após cada *sprint*, tanto a equipe de projeto como os *stakeholders* envolvidos reúnem-se para avaliar a iteração. As atividades e os resultados da iteração são revisados e aquilo a ser feito no próximo *sprint* é planejado.Para cada novo *sprint*, a equipe identifica as atividades prioritárias e as reclassifica, iniciando um novo *sprint*.

A cada *sprint* deve ser entregue, pelo menos, uma funcionalidade de valor ao cliente para sua avaliação.

Durante o *sprint*, reuniões diárias são efetuadas com durações de 15 a 30 minutos objetivando:

- Focar o esforço nos itens das atividades priorizadas.
- Comunicar os itens das atividades aos membros das equipes.
- Informar a todos sobre os avanços e obstáculos.
- Remover os obstáculos o mais breve possível.
- Apontar o progresso na entrega das funcionalidades, direcionar e minimizar o risco de projeto.

A realização diária de reuniões ao longo do projeto aponta o sucesso e a transparência do projeto, proporcionando benefícios como deixar claro, para todos os membros da equipe dentro do contexto do projeto, as metas gerais e individuais de cada um. A transparência, também proporciona uma redução na quantidade de tempo gasto em atividades que se preocupam com o jogo político dentro do projeto (Janoff, 2000).

O método, de forma geral, procura:

- Dividir o projeto em pequenas séries adaptáveis e maleáveis.
- Progredir enquanto os requisitos não são estáveis.
- Deixar toda informação visível para todos.
- Ter a comunicação como chave do sucesso para a equipe.
- Entregar, aos clientes, produtos tangíveis em incrementos.
- Desenvolver uma cultura de confiança e engajamento do cliente junto com a equipe de desenvolvimento, buscando o sucesso do projeto.

O Scrum requer indivíduos comprometidos com a equipe e o projeto, enquanto que dos gerentes são esperados a demonstração de confiança e o respeito para com os indivíduos. Assim, os indivíduos podem se concentrar em seu trabalho, necessitando menor quantidade de reuniões, relatórios e autorizações. Um ambiente de trabalho aberto e democrático onde a realimentação é encorajada é uma das características do Scrum. Assim, a tolerância a enganos também é praticada e absorvida. O objetivo é utilizar a capacidade, inteligência e experiência de todos (Beedle, 2001).

A figura 16 exibe o processo do método Scrum.

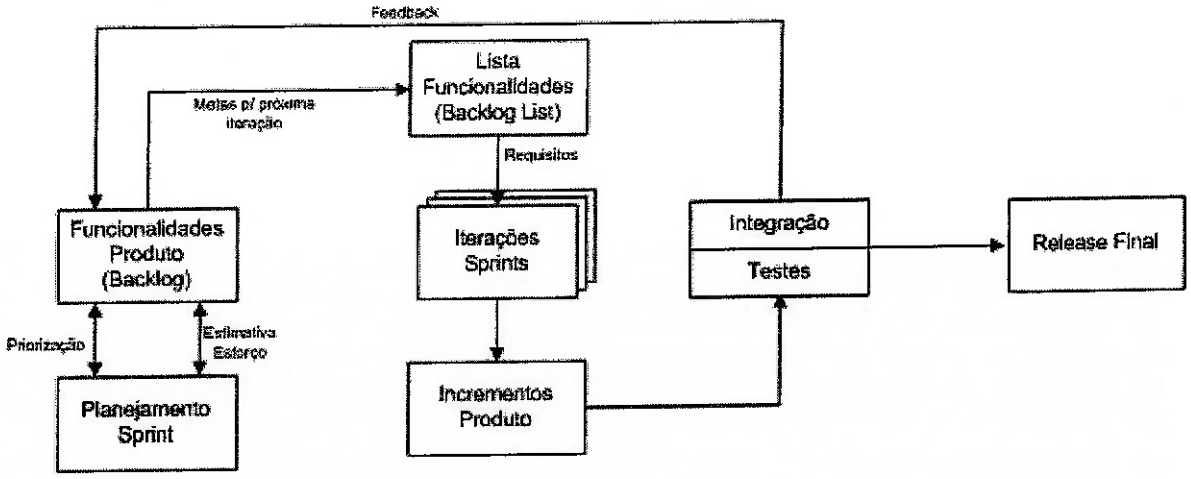

Figura 16 - Processo Scrum - adaptado de (Beedle, 2001)

3.4 O Processo Geral do Método Ágil

Apesar de terem os mesmos objetivos de agilizar o desenvolvimento de software, os métodos apresentados nas seções 3.3.1 a 3.3.6 mostram diferenças entre si. Foi feita, neste trabalho, uma síntese de um Método Ágil genérico, caracterizado através dos princípios e dos valores, considerando os Métodos Ágeis definidos anteriormente.

Um Método Ágil é caracterizado por 10 práticas, segundo Astels (2002):

- *Modular*: O processo é dividido em atividades distintas. As atividades podem ser incluídas no processo modular e podem ser removidas caso não sejam necessárias. A modularidade não é um movimento na direção dos processos de desenvolvimento personalizados. Em vez disso, ela promove a definição concreta das novas atividades para que não se tenha um processo inconsistente que tome direções que vão contra o bom senso.

- *Iterativo*: Através de um desenvolvimento iterativo se reconhece a existência de problemas e erros que deverão ser solucionados. Como resultado, o foco não é a perfeição absoluta, mas a aprendizagem. Os erros são tolerados e se permite aprender com eles.

- *Incremental*: Os projetos são criados em pequenas partes, evitando algum tipo de solução radical. Como resultado, se adquire uma realimentação sobre as partes pequenas, o que permite refinar a entrega do produto principal.

- *Limitado no tempo*: Cada incremento tem uma duração fixa. A mudança das datas não é aceita. É preferível reduzir o escopo da iteração, em vez de permitir que a data de conclusão seja alterada.

- *Parcimonioso*: Os processos ágeis exigem um número mínimo de atividades necessárias para diminuir os riscos e atingir seus objetivos. Exigindo um mínimo de atividades, um processo ágil permite que os desenvolvedores se concentrem e atendam os cronogramas rígidos sem desgaste.

- *Adaptável*: À medida que o desenvolvimento continua, novos riscos podem ser encontrados. Para lidar com esses riscos, talvez sejam necessárias novas atividades. Um processo ágil permite tanto que essas novas atividades sejam incluídas como permite que as atividades existentes sejam modificadas conforme a necessidade.

- *Convergente*: Essa prática declara que todos os riscos necessários foram assumidos. Após cada incremento, o sistema está mais próximo do seu objetivo final, convergindo para uma solução, focando a arquitetura e o projeto como dois pontos-chave para manter a convergência.

- *Orientado a Pessoas*: Os processos de software ágeis funcionam melhor com equipes pequenas. Uma equipe pequena promove um forte espírito de grupo e a sensação de que cada pessoa é importante e não apenas uma peça descartável em um grande processo. Isso fornece poder aos desenvolvedores para que eles façam parte da evolução do processo e melhorem sua própria produtividade, qualidade e desempenho. Uma equipe pequena torna as comunicações mais fáceis. À medida que a equipe cresce, a comunicação se

torna um desafio maior. Quando isso acontece, a equipe precisa se dividir em equipes menores. No entanto, a abordagem orientada a pessoas é mais do que simplesmente criar equipes pequenas; é fundamental entender que pessoas competentes e motivadas em um ambiente produtivo entregam bom software.

- *Colaborativo:* A comunicação é de importância vital para todo o processo de desenvolvimento de software e é incentivada pelos Métodos Ágeis. Todos os participantes têm de entender como as diversas partes se completam, qual o significado dos requisitos, e trabalhar de forma cooperativa para atingir o objetivo.

- *Complementar*: Determinadas atividades fornecem realimentação quando combinadas com outras em um processo. Para atingir a complementação nos processos de desenvolvimento de software, as atividades que atuam bem unidas quase sempre são uma parte fundamental para criar a dinâmica que leva ao sucesso.

As dez práticas descritas estão presentes em cada um dos Métodos Ágeis descritos com maior ou menor destaque, conforme suas características de desenvolvimento. Nota-se que todos possuem algum tipo de ciclo de vida em que estão baseados, e possuem muitas semelhanças e características comuns, apesar de utilizarem taxonomias diferentes para atividades e produtos afins. Da análise destes métodos, pode-se identificar as seguintes características nos seus processos:

- Existência de uma fase de iniciação do projeto

- Existência de planejamento do projeto, para definir a liberação das partes do sistema
- Entrega de produtos funcionais consecutivos em períodos de tempo fixados

- Ciclos curtos para a entrega dos produtos funcionais, com adaptabilidade do processo

- Redução de riscos para o projeto, através das realimentações fornecidas pelas entregas de produtos funcionais.

Com base nestes dados, pode-se definir um processo genérico de um Método Ágil, ilustrado pela figura 17, com as seguintes fases:

- Fase de Concepção
- Fase de Elaboração
- Fase de Construção
- Fase de Transição

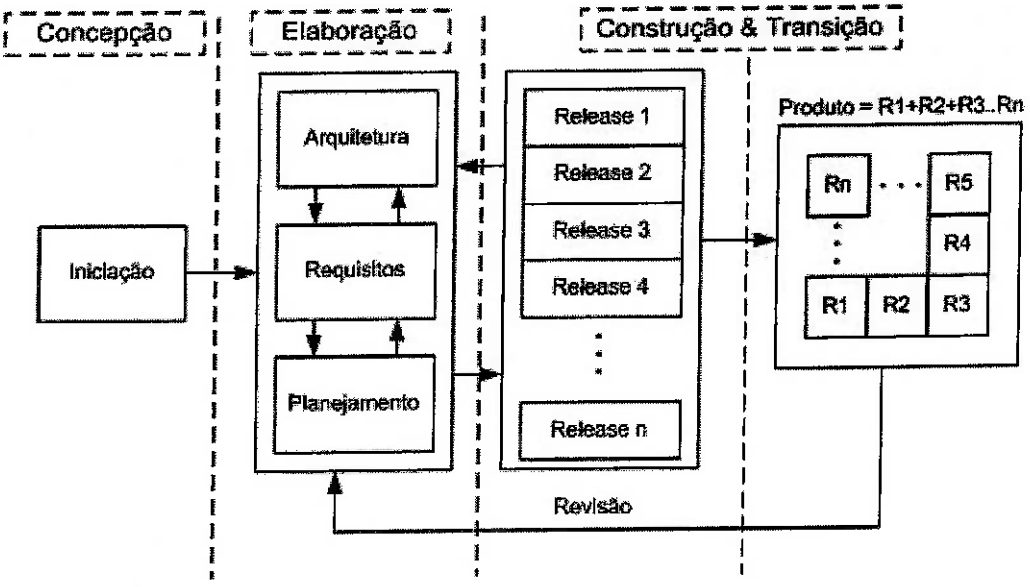

Figura 17 - Processo Ágil Genérico

Tem-se, a seguir, a descrição das fases do processo genérico:

Concepção: Seu objetivo é dimensionar e justificar o projeto. Os requisitos de mais alto nível são estabelecidos, e normalmente definem o escopo do projeto.

Elaboração: O detalhamento dos requisitos, o planejamento e a definição da arquitetura são executados de forma paralela e colaborativa.

- Os requisitos de alto nível são detalhados pelos clientes, normalmente com pouca profundidade, e os aspectos do negócio são transmitidos através da verbalização.

- O planejamento segue os requisitos definidos pelos clientes e a sua priorização, considerando aquilo que é de mais valor para o negócio sob o ponto de vista do cliente.

- A definição da arquitetura segue em paralelo, pela equipe técnica, conforme os requisitos emergem e as necessidades do sistema são identificadas.

Construção e Transição: são fases do processo em que uma parte do software funcional é construída e entregue para o cliente, para avaliação e realimentação. O resultado deverá realimentar a fase de elaboração para a adaptação do processo e avaliação dos objetivos alcançados. Essa entrega é limitada por um escala de tempo que deve ser suficientemente curta para que a realimentação seja efetiva.

Todas as fases deverão estar imersas em um ambiente de comunicação efetiva e ágil, proporcionando um alto grau de participação e colaboração.

4 ENGENHARIA DE REQUISITOS NOS MÉTODOS ÁGEIS

Neste capítulo são discutidos o tratamento dos requisitos nos Métodos Ágeis de desenvolvimento de software e o processo característico de Engenharia de Requisitos nestes métodos.

Em seguida, é feita uma discussão mais detalhada do método Extreme Programming, por ser ele um dos métodos mais utilizados. É apresentada a disciplina de Engenharia de Requisitos no contexto do Extreme Programming e a sua comparação com duas abordagens bastante relevantes no momento: a disciplina de requisitos do Rational Unified Process (referenciar) e a KPA de gerência de requisitos do modelo SW-CMM.

4.1 Requisitos nos Métodos Ágeis

Na construção de software voltado para negócios, as mudanças de requisitos são comuns (Fowler, 2002). Em ambientes onde existe a instabilidade e a mudança freqüente de requisitos, a Engenharia de Requisitos deve ser capaz de estabelecer os processos que garantam a efetividade do projeto. A utilização de processos obsoletos ou incompatíveis com o domínio de requisitos do negócio pode, invariavelmente, obrigar a disciplina de Engenharia de Requisitos a efetuar um grande esforço na captação de todos os requisitos na fase inicial do projeto, resultando em requisitos repletos de omissões e enganos de concepções, ocasionando grandes problemas nos projetos de software.

A solução tem sido um movimento na direção de modelos com perfis iterativos (Tomayko, 2002) e, via de regra, esses processos são baseados em modelos evolucionários como definidos em 3.3.4.

Construir sistemas competitivos com tecnologia avançada é uma tarefa complexa. Segundo Schwaber (2002), o grau de complexidade, como demonstrado na figura 18, aumenta conforme os requisitos são pouco conhecidos e a tecnologia é pouco dominada. Se incluir uma terceira dimensão - as pessoas, o grau de complexidade da

maioria dos projetos atinge pelo menos o grau de dificuldade que aponta normalmente para o complexo.

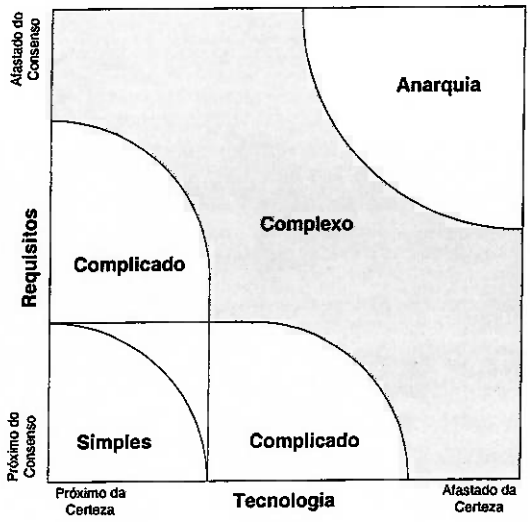

Figura 18 - Complexide Desenvolvimento Software - (Schwaber,02)

Em ambientes complexos, o fator risco se torna preponderante e é imperativo que o processo de desenvolvimento esteja focado em sua minimização. Riscos, segundo Boehm (2000), são situações ou possíveis eventos que podem levar um projeto a não alcançar seus objetivos.

Um modelo de processo de baixo risco é aquele que seja direcionado pelo risco, de modo a determinar o processo e o produto através de uma construção incremental do sistema. O modelo que melhor se encaixa nesse requisito é o modelo espiral de desenvolvimento, pois é um modelo gerador de processos direcionado pelo risco (Boehm, 2000), possuindo duas características principais:

- O ciclo incremental de crescimento do sistema e diminuição gradual do risco.

- Um conjunto de marcos para o projeto, de modo a assegurar a viabilidade e o alcance das expectativas dos *stakeholders* na solução construída para o sistema.

O processo de Engenharia de Requisitos apóia-se nesse modelo como disciplina, para desenvolver e gerenciar os requisitos de software a ser desenvolvido.

A freqüente inspeção e adaptação imediata, conforme os resultados ocorrem, são os mecanismos básicos para se trabalhar com requisitos em projetos complexos. A melhor forma de lidar com esse ambiente é utilizar implementações das seguintes práticas (Schwaber, 2002):

- *Desenvolvimento Iterativo*: Freqüentes iterações geram incrementos de requisitos que podem ser inspecionados para determinar o estado do projeto e serve como base para a adaptação do processo.

- *Incrementos de Trabalho*: Correspondem às versões de sistema funcional, ao invés de artefatos e documentações. Esses incrementos criam uma relação 1:1 entre progresso e produto entregue, e proporciona um mecanismo para a realimentação do cliente sobre o produto real e seus requisitos.

- *Colaboração*: Clientes e desenvolvedores formam equipes que trabalham em conjunto, aumentando a velocidade e a qualidade da informação do projeto.

- *Reuniões diárias*: Revelam a situação diária do estado do projeto, dos problemas e dos obstáculos a serem vencidos.

- *Adaptação*: As equipes de desenvolvedores se auto-organizam diariamente, baseadas nas reuniões diárias, e os clientes e os desenvolvedores se auto-organizam como equipe ao final de cada incremento, movendo o projeto no sentido de agregar o máximo valor de conhecimento e produtividade.

- *Emergência*: a arquitetura, a estrutura da equipe e a descoberta dos requisitos ocorrem durante o projeto. As equipes são direcionadas através de visões esquemáticas e preliminares dos requisitos e da arquitetura, que são

confirmadas ou adaptadas conforme a realimentação do processo colaborativo envolvido.

Além do risco, um outro fator importante nos Métodos Ágeis é tornar a mudança dos requisitos como uma conseqüência para o aumento do valor do software do ponto de vista do negócio.

Quando um projeto é iniciado, a organização normalmente se baseia na expectativa do retorno do investimento, ou seja, o custo do projeto deve-se transformar em algum benefício ao negócio ou em um valor que seja maior que o investido. Métodos Ágeis objetivam que, ao invés dos requisitos como direcionadores do valor do negócio, o valor possa emergir através de incrementos de software funcional (Schwaber, 2002). Na definição de Fowler (2002), o software é naturalmente intangível, é muito difícil visualizar que o valor de uma função possui, até que possa ser vista na forma real como produto. Somente quando se utiliza uma versão inicial de algum software é que se começa a entender quais funções possuem valor e quais não possuem.

O foco no valor, e não nos requisitos, pode ser expresso como (Schwaber, 2002):

$$\text{Valor Negócio} = f(\text{custo, tempo, funcionalidade, qualidade})$$

Para que isso ocorra, dois ciclos de trabalhos colaborativos de requisitos devem ocorrer em processos ágeis:

- A equipe de desenvolvimento, responsável pelos requisitos de tecnologia, gera com freqüência novos incrementos de funcionalidade, com base nas necessidades levantadas pela equipe do cliente.

- A equipe do cliente, responsável pelo domínio do negócio, gera novos requisitos e prioridades das funções do sistema, necessidades de cronograma e expectativa de qualidade, baseados na dinâmica do negócio, baseados em um produto tangível, entregue pela equipe de desenvolvimento.

Ao final de cada iteração, clientes e equipes de desenvolvedores planejam, de forma colaborativa, o próximo ciclo de desenvolvimento, baseados naquilo que já foi desenvolvido e nos novos requisitos de negócio que emergiram. Quanto mais rápida for a entrega de software funcional ao cliente em seus ciclos iterativos, mais vantagens serão proporcionadas em reconhecer os eventos inesperados e as oportunidades de negócio, refinando os requisitos na direção de um sistema de qualidade. A figura 19 apresenta a dinâmica do processo.

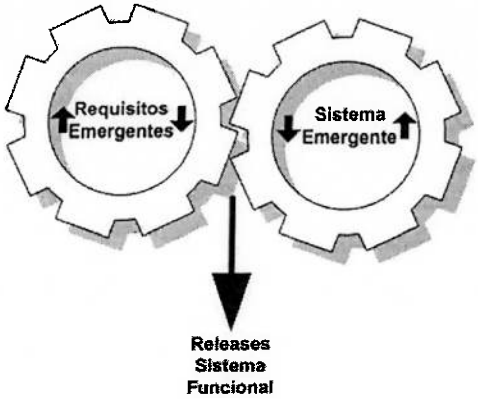

Figura 19 - Processo Requisitos Métodos Ágeis - (Schwaber, 2002)

4.2 O Processo de Engenharia de Requisitos nos Métodos Ágeis

O objetivo da Engenharia de Requisitos não é apenas escrever um documento de requisitos extenso e detalhado mas transferir, de forma efetiva, idéias e necessidades do cliente para o desenvolvedor (Young, 2001). Nos Métodos Ágeis, a transferência de idéias é um dos maiores parâmetros de efetividade. Segundo Highsmith (2000), essa efetividade é basicamente evidenciada através de um dos seus princípios – *A maior prioridade é satisfazer o cliente, o mais breve possível e continuamente, com software funcional.*

Para o cliente transferir idéias, ele deve explorar o universo do discurso, a fim de construir um modelo mental de uma situação futura e comunicar aquela realidade para o desenvolvedor. O desenvolvedor, ao entender o que é comunicado, deve adaptar a sua realidade à realidade do cliente tão próximo quanto possível. O

processo de captura de requisitos, a partir do ponto de vista do usuário, é a melhor maneira de resolver o problema de maneira correta (Clavadetscher, 1998). Como esses dois modelos mentais não podem ser idênticos, a tarefa principal é diminuir a distância entre eles. Os Métodos Ágeis não declaram explicitamente a utilização de Engenharia de Requisitos em seus ciclos de vida, mas propõem princípios extremos para diminuir a distância entre os dois modelos mentais e construir a ponte de comunicação com velocidade.

A Engenharia de Requisitos pode ser encarada como uma tentativa de construir a comunicação entre esses modelos. O desafio em um mercado altamente dinâmico é construí-la de forma rápida (Goetz, 2002).

A figura 20 estabelece uma proposição para um processo iterativo e incremental de captação de requisitos no Método Ágil, indicando as tarefas pelas quais os *stakeholders* e os desenvolvedores são responsáveis. O processo sugere o nível de responsabilidade que *stakeholders* e desenvolvedores estejam envolvidos de forma colaborativa e interativa na identificação de idéias ou sugestões, na discussão dos requisitos potenciais, na sua modelagem e na documentação necessária, para que os envolvidos adquiram conjuntamente o entendimento do sistema e suas reais necessidades.

Como responsabilidade integral, cabe aos *stakeholders* a priorização dos requisitos e aos desenvolvedores a estimativa para sua implementação. A transferência do aprendizado, das experiências e das necessidades se dá em forma da verbalização entre desenvolvedores e *stakeholders* (Ambler, 2002).

A sugestão de Ambler (2002) é que os *stakeholders* devam ser envolvidos na modelagem e documentação de seus requisitos e não somente como provedores de informação, trabalhando de forma ativa no projeto, tornando as necessidades de uma documentação formal uma prioridade a ser estabelecida pelo próprio cliente.

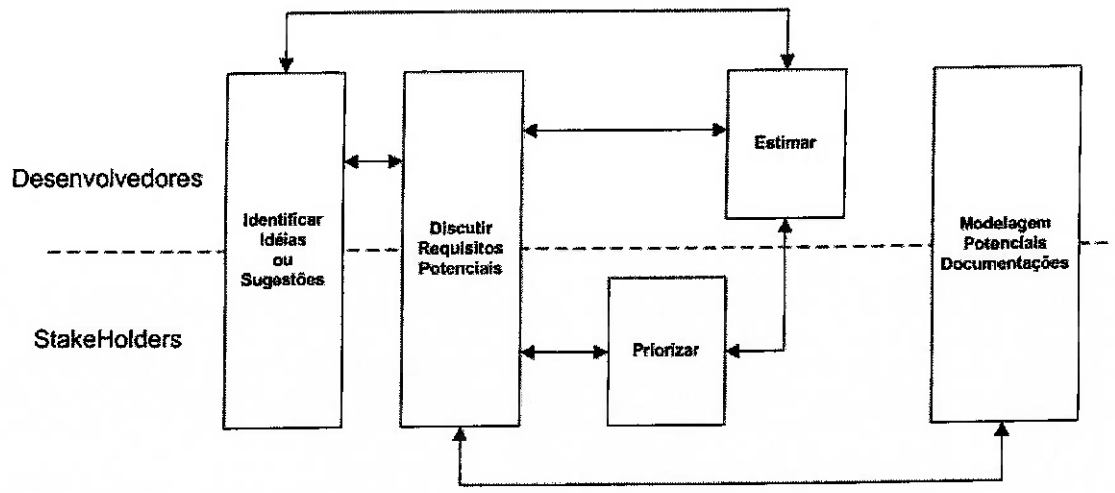

Figura 20 - Processo de Requisitos em Métodos Ágeis – (Ambler, 2002)

Aplicando este modelo de esforço na modelagem de requisitos, acaba-se proporcionando um engajamento dos *stakeholders*, com participação no projeto e aumentando as chances de uma colaboração mais efetiva.

Uma das características dos Métodos Ágeis é a preferência por comunicação face-a-face. Este processo é considerado como um canal de alta velocidade em termos de captura de requisitos, se comparado com documentação formal de requisitos, cuja obtenção é caracterizada como muito lenta, na transferência de conhecimento para uma equipe de projeto de software.

A figura 21 (Cockburn, 2001a) exibe a efetividade do tipo de comunicação através de alguns meios comuns em projeto de software. O pico de efetividade de comunicação é para duas pessoas em um Quadro Branco (**whiteboard*). A comunicação face-a-face possui características que, segundo Cockburn (2001a), incrementam a efetividade deste canal de comunicação, destacando a proximidade física, gestos, inflexão vocal e perguntas e respostas em tempo real que enfatizam e agregam o canal de comunicação. O documento não é capaz de possuir essas características, tornando-se muito pouco eficaz na comunicação entre pessoas.

---

* *Whiteboard* ou quadro branco para escrita informal, segundo Cockburn (2001a), é um ponto de concentração onde a efetividade da comunicação é a maior possível, através do seu conteúdo informal, perguntas e respostas em tempo real e compartilhamento de informação entre várias pessoas.

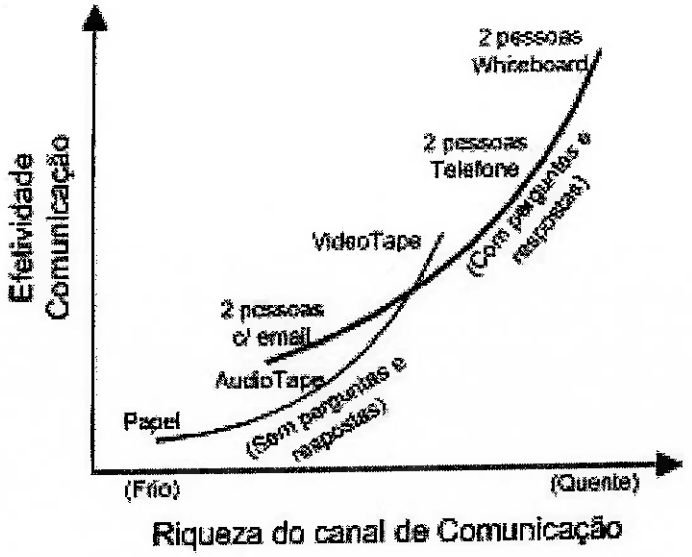

Figura 21 - Efetividade de Comunicação (Cockburn, 2001a)

A comunicação efetiva é um ingrediente chave para atingir o sucesso de um projeto (Young, 2001). O processo de requisitos deve estar imerso nessa efetividade da comunicação entre os membros participantes do projeto. O modelo de processo da figura 20 pode ser complementado com a atividade de comunicação face-a-face por todo o processo, como mostrado na figura 22.

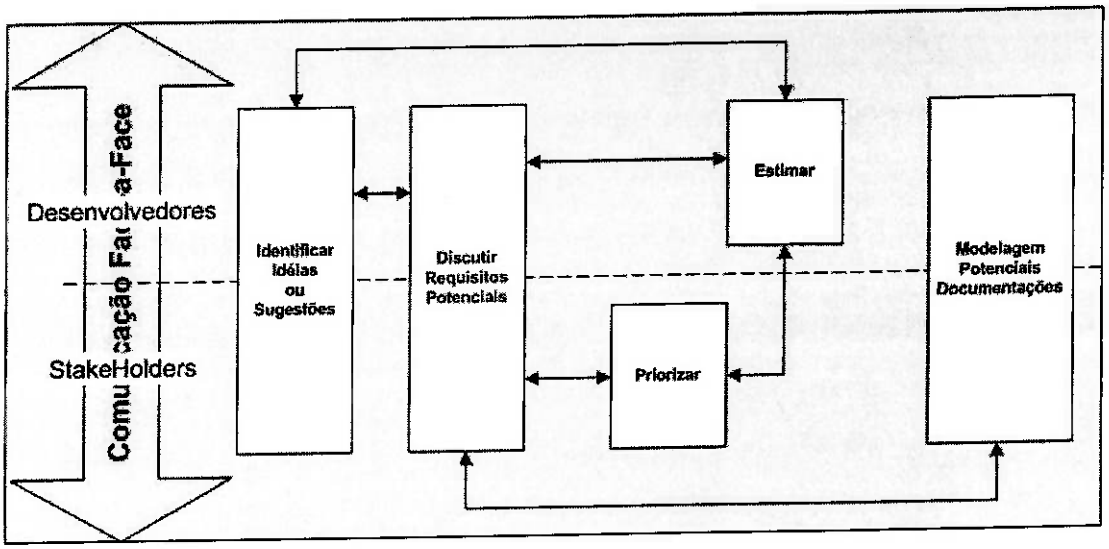

Figura 22 - Processo de Requisitos Ágeis com Comunicação Face-a-Face

A comunicação só é efetiva em um ambiente de desenvolvimento de software onde o espírito de colaboração exista. As revisões e as realimentações feitas pelos praticantes de Métodos Ágeis mostram que a interação direta e regular com cliente é um dos fatores chave para o sucesso do projeto. A importância de envolver clientes no processo de desenvolvimento, segundo Eberlein (2002), tem sido reconhecida há tempos. Esse envolvimento, no entanto, pode não ser concretizado devido a uma gerência inadequada do processo de desenvolvimento de requisitos de software, impossibilitando um clima de parceria e comprometimento. Clavadetscher (1988) afirma que, para conseguir o comprometimento do cliente com o projeto, este deveria participar diretamente do processo gerência dos requisitos para assegurar que suas necessidades estão sendo atendidas e obedecendo a suas prioridades, cabendo à equipe de desenvolvimento apenas ajudar o cliente a obter o entendimento completo do problema, pois somente o cliente é quem sabe o que sistema deve realmente fazer para o seu negócio.

O princípio do cliente *on-site* sugere que, pelo menos uma pessoa da equipe do cliente, participe da equipe de desenvolvedores, na maior parte do projeto, preferencialmente no mesmo ambiente. Sua tarefa principal é responder as perguntas e comunicar suas necessidades à equipe de desenvolvimento. Ele seleciona e ordena os requisitos a ser implementados e fornece a priorização dos mesmos. O desenvolvimento é, portanto, direcionado pelos interesses do negócio e isto ajuda a definir o que é de valor para o cliente (Goetz, 2002).

Os requisitos organizados desta forma proporcionam que a equipe de desenvolvimento crie objetos tangíveis, os sistemas funcionais, de forma incremental e iterativa nos menores intervalos de tempo possíveis, para verificação pelo cliente.

Este procedimento é importante, pois raramente o cliente conhece os requisitos do sistema no início do projeto, estando longe de possuir um modelo completo, conciso e não ambíguo que possa ser transferido ao desenvolvedor (Goetz, 2002). Observando e sentindo o sistema entregue, ele auxilia a explorar o universo de discurso e refinar os requisitos para o sistema de valor.

Essa iteração com um objeto tangível fornece uma realimentação constante para a equipe de desenvolvimento, gerando uma uniformização de conceitos e necessidades. A figura 22, que mostra um processo de captura de requisitos nos Métodos Ágeis, pode ser expandida para a figura 23, incluindo o mecanismo de *releases* de software funcional, atuando como instrumento de elicitação e verificação de requisitos.

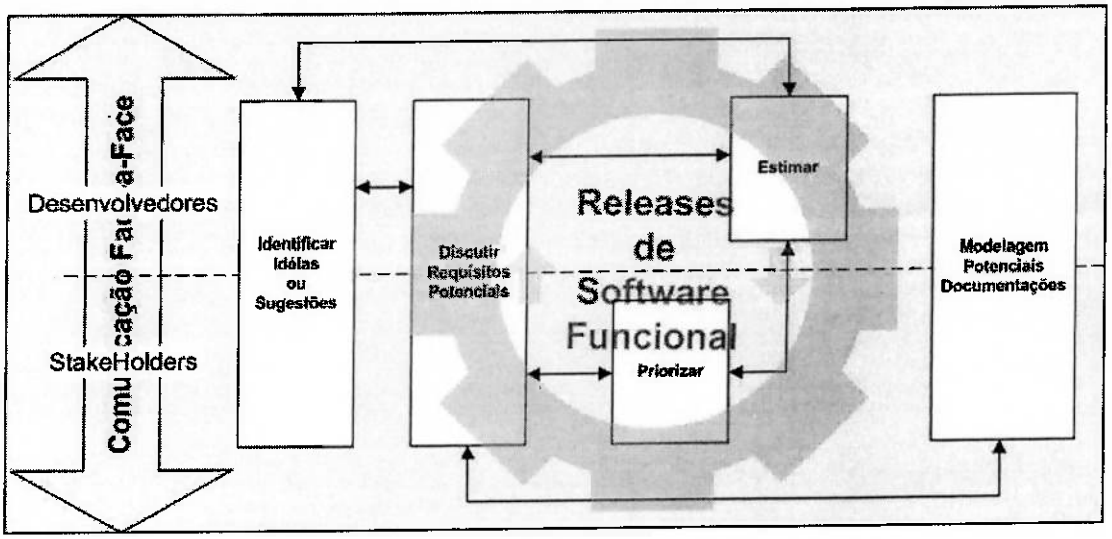

Figura 23 - Processo de Requisitos Ágeis com Entrega de Software Funcional

A especificação de requisitos como discutido na seção 2.1, pode ser considerada como uma documentação em um apropriado nível de detalhe, de modo a ser inteligível a todos os *stakeholders* envolvidos.

Nos Métodos Ágeis, a documentação formal abrangente não é uma prioridade conforme os valores definidos anteriormente na seção 3.2.1 item b, mas a utilização de uma documentação apropriada e de comum entendimento aos *stakeholders* envolvidos é relevante para o processo.

Os Métodos Ágeis se valem de cartões com algumas palavras ou casos de uso simplificados (as *user stories* em Extreme Programming, por exemplo) como documentos de requisitos de alto nível (Goetz, 2002). Estas descrições muito curtas e

abstratas servem principalmente como marcos ou notas promissórias para uma conversa futura, entre as equipes.
A documentação é complementada com os testes aceitação dos requisitos, escritos normalmente pelo próprio cliente, e serve como base para a construção do software funcional. Essa documentação de alto nível acaba atuando como um contrato de desenvolvimento entre cliente e desenvolvedor, válido apenas para o incremento vigente de desenvolvimento.

Como uma documentação concreta e verificável, o software funcional é considerado, por Goetz (2002), uma documentação eletrônica executável e atualizada *just in time*. Os requisitos estão dispostos no software e concretamente verificados; a cada iteração, a documentação é atualizada e compartilhada por todos os membros da equipe.

Esses aspectos viabilizam que uma boa quantidade de conhecimento seja transferida de cliente ao desenvolvedor, em um espaço de tempo muito curto, com um *overhead* burocrático muito pequeno, enfatizando a velocidade e a agilidade do método. A figura 23 do processo de captura de requisitos em Métodos Ágeis pode ser complementada com a figura 24, exibindo a existência da especificação de requisitos por todo o processo.

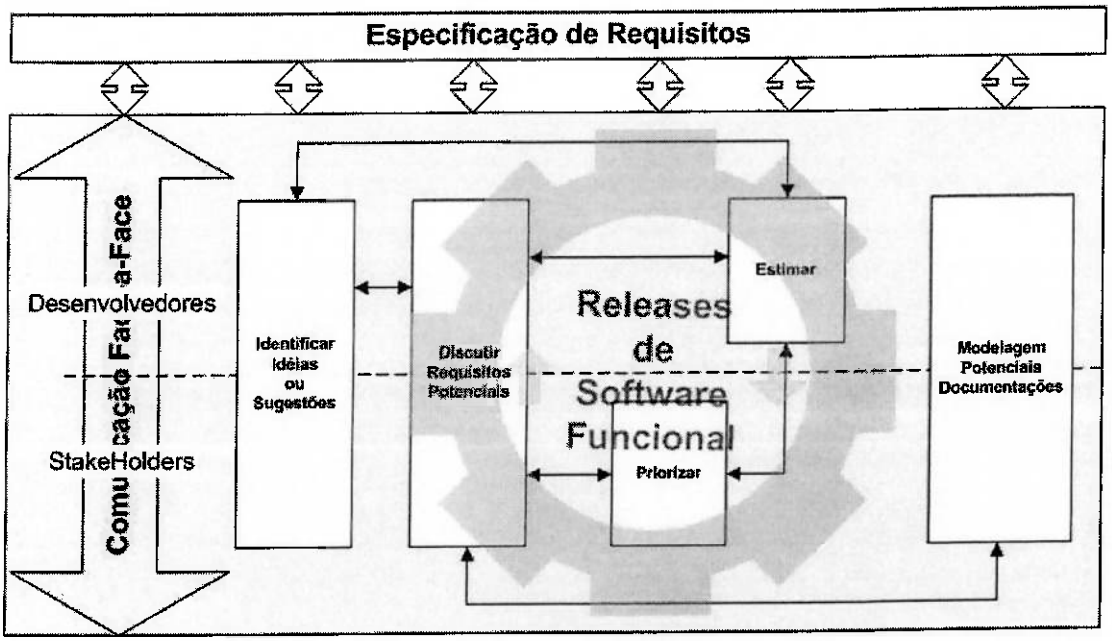

Figura 24 - Processo de Requisitos Ágeis com Especificação de Requisitos

O Método Ágil não especifica quando algum tipo de documentação formal é necessária ser enviada para fora do domínio das equipes de cliente e desenvolvedor. Alguns métodos colocam essa necessidade como um esforço que deve ser envolvido e caracterizado no processo. "*Se existe a necessidade do negócio por um documento, o cliente deve requerer o documento da mesma forma como ele requer uma função: através de um cartão. A equipe de desenvolvimento irá estimar o custo do documento e o cliente pode programá-la na iteração desejada*" (Jeffries, 2001).

A gerência de requisitos, um dos aspectos fundamentais da Engenharia de Requisitos, está presente nos Métodos Ágeis principalmente através dos princípios do cliente *on-site* e a integração contínua de software (Paulk, 2001). Utilizando as definições apontadas na tabela 1 da seção 2.2, podem-se comparar as práticas de gerencia de requisitos com alguns princípios dos Métodos Ágeis, conforme apresentado na tabela 4.

Tabela 4 - Engenharia de Requisitos x Métodos Ágeis

Gerência de Requisitos	
Engenharia de Requisitos	Métodos Ágeis

Definir a *baseline* dos requisitos.	Existe uma *baseline* e a cada ciclo de iteração o cliente define quais são os requisitos a serem implementados. A *baseline* pode ser alterada em função do melhor entendimento do software funcional.
Revisar as mudanças de requisitos propostos e avaliar o impacto de cada mudança proposta antes de aprova-los	Um dos princípios dos Métodos Ágeis é que toda a mudança é bem vinda, pelo aspecto iterativo e evolucionário. Os clientes refinam os requisitos a partir de um sistema funcional e os desenvolvedores estimam seu custo. Os clientes direcionam a aprovação.
Incorporar as mudanças de requisitos de forma controlada.	Os requisitos são incorporados na próxima iteração e verificados através do produto funcional entregue.
Manter os planos de projeto atualizados com os requisitos.	O planejamento nos Métodos Ágeis é apenas para a próxima iteração, pelo seu foco na adaptabilidade do processo e não da previsibilidade.
Negociar novos compromissos baseados na estimativa de impacto pelas mudanças dos requisitos.	O aspecto colaborativo das equipes e a presença do cliente na equipe de projeto promovem um comprometimento com o sucesso do projeto.
Manter rastreabilidade dos requisitos com suas correspondentes análises, modelos, códigos fonte e casos de teste.	O valor de entrega de software funcional, ao invés de uma documentação abrangente, minimiza a necessidade de níveis de rastreabilidade pela característica evolucionária e emergente dos requisitos.
Apontar o estado dos requisitos e das atividades das mudanças no andamento	O software funcional entregue é a medida básica do progresso do projeto;

do projeto.	os requisitos estão concretamente realizados ao final da iteração.

Os Métodos Ágeis podem controlar os requisitos emergentes de forma mais natural, dentro de seus processos, do que a maioria dos métodos tradicionais. Liberações freqüentes e rápidas de software funcional descobrem requisitos não claros, e riscos com novas tecnologias são reduzidos, habilitando a exploração de soluções alternativas com tecnologia.

Direcionando o processo para pessoas, estas passam a ser participantes cruciais e não apenas mais um dente da engrenagem do processo, com o aumento da responsabilidade e controle adquirido (Highsmith, 2001).

O processo de captura de requisitos é completado pela gerência de requisitos na figura 25, ressaltando sua importância em todo o processo como fator essencial para acomodação das mudanças freqüentes de requisitos.

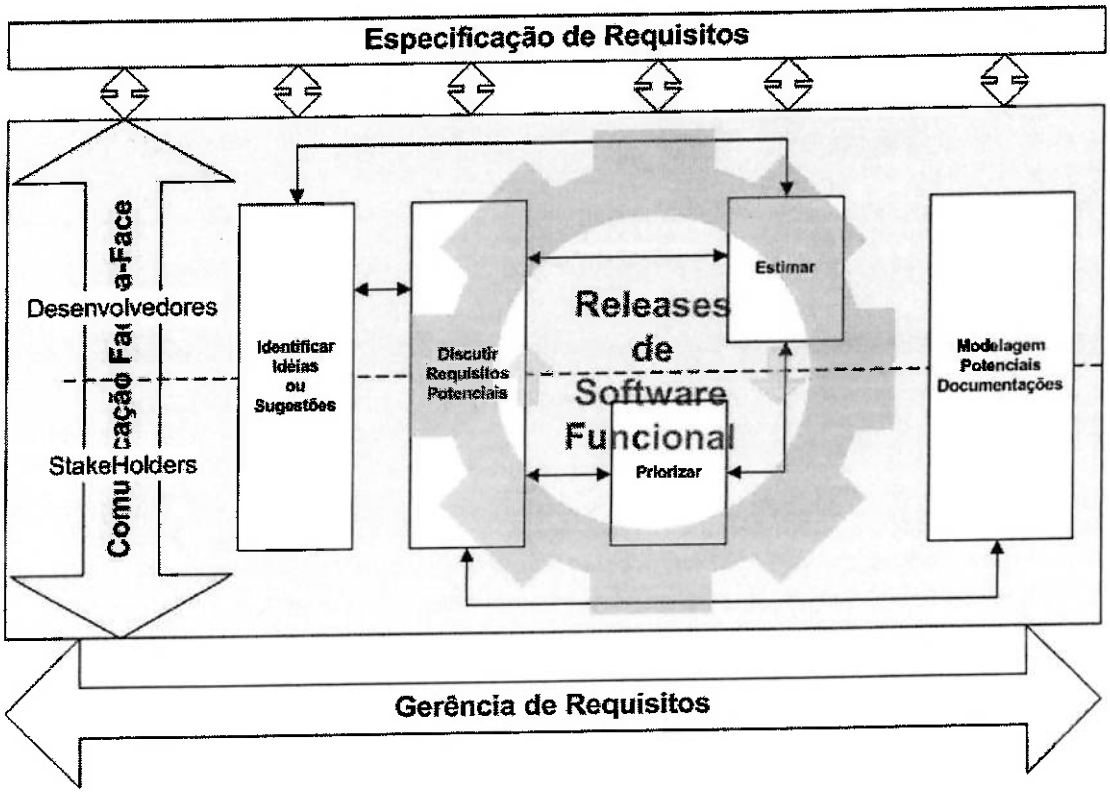

Figura 25 - Processo de Requisitos Ágeis com Gerência de Requisitos

4.3 Engenharia de Requisitos em Extreme Programming

Conforme definido na seção 3.3.1, o Extreme Programming faz o uso constante do princípio cliente *on-site*. O cliente deve fazer parte da equipe de projeto e a ele é atribuída a capacidade de resolver as dúvidas e tomar decisões relativas a prioridades de recursos e riscos.

Essas atribuições equivalem a dividir a responsabilidade do cliente em duas categorias (Duncan, 2001):

- Definição dos parâmetros do produto do ponto de vista funcional que são definidos através das *user stories* e testes de aceitação.

- Definição dos parâmetros do produto do ponto de vista executável que são definidos através do cronograma de entrega do incremento do produto e planejamento da iteração.

Os parâmetros do produto funcional são principalmente comunicados através de *user stories*. São similares aos casos de uso da UML, mas muito mais simples em seu escopo e devem ser escritas pelo cliente em cartões e não devem possuir mais que algumas sentenças. Para cada *user story*, o cliente deve escrever um teste de aceitação, normalmente no verso do cartão da *user story* (Astels, 2002).

Segundo Beck (1999), as *user stories* possuem dois componentes. O primeiro é um cartão escrito com poucas sentenças sobre a funcionalidade desejada e apontando, quando necessário, para um documento de apoio. O segundo, e não menos importante, é a série de conversas que deverá ocorrer entre o cliente e o desenvolvedor sobre a *user story*, que captura as documentações que a equipe ache relevante em referenciar no cartão.

Uma característica que diferencia do método tradicional é que os requisitos, em XP, não necessitam ser escritos para responder todas possíveis questões, uma vez que o cliente esteja sempre ali para responder as perguntas que surjam.

Esta técnica pode ficar rapidamente fora de controle para um desenvolvimento de grande porte, mas para pequenas e médias equipes podem oferecer uma substancial economia, agilidade e desburocratização (Duncan, 2001).

Os programadores analisam cada *user story* e estimam o esforço necessário para implementá-lo. O XP utiliza, como métrica, os *story points* ou semanas ideais de 40 horas para desenvolvimento (Astels, 2002). Se o programador estimar que a *user story*, de modo isolado, poderá levar que mais de três semanas ou três *story points* para implementá-lo, pede ao cliente que divida a *user story* em outras menores.

Uma vez que as *use stories* foram estimadas, o cliente seleciona quais irão ser implementadas para os *releases* a serem entregues focando, deste modo, os interesses principais do negócio.

Cada *user story* a ser implementada é dividida em tarefas. Um par de programadores irá trabalhar para solucionar uma tarefa por vez (Astels, 2002). O primeiro passo é solucionar a tarefa , após seu entendimento, escrevendo casos de testes unitários. Para Beck (1999) os casos de testes irão definir a quantidade de esforço de codificação para a tarefa. Uma vez que os testes obtiverem aceitação, a tarefa é considerada completa. Para qualquer nova integração de código, todos os testes deverão possuir nova aceitação.

Segundo Duncan (2001), os testes unitários para as tarefas, podem ser considerados como uma forma de requisitos, não simplesmente como uma forma de especificação executável, mas o reconhecimento e registro, por parte da equipe de desenvolvimento, de requisitos específicos para cada tarefa através de seus casos de testes, caracterizando-os como requisitos persistentes, em função do princípio de desenvolvimento baseados em testes do XP.

Os testes de aceitação, que correspondem aos testes tipo caixa-preta, de responsabilidade do cliente, devem assegurar que os cenários especificados sejam suficientemente completos para aquela iteração.

Esses testes de aceitação servem como um determinador de não ambigüidade e correção, quando os requisitos especificados pelo cliente encontram respaldo no código entregue (Duncan, 2001).

A especificação de requisitos, em Extreme Programming, não é identificada como um único documento, sendo uma coleção de *user stories*, testes de aceitação escritos pelo cliente e testes unitários de cada tarefa. O cliente, como parte da equipe de projeto, pode ser considerado como parte da especificação, desde que esteja disponível para responder as questões e esclarecer as ambigüidades (Duncan, 2001).

O XP , como um Método Ágil, possui a característica de um processo evolucionário com incrementos, os menores e mais rápidos possíveis, com entrega de software funcional. Esta característica permite que os clientes visualizem de forma concreta o progresso do projeto e realimentem de forma rápida com requisitos e mudanças necessárias para atingir suas expectativas.

Segundo Wiegers (1999), bons requisitos devem ser, conforme definido na seção 2.4: Completos, Corretos, Decomponíveis, Necessários, Priorizáveis, Sem ambigüidade e Verificáveis. Ainda, uma especificação deve ser: Completa, Consistente, Modificável, Rastreável.

Comparando as características definidas anteriormente com a análise de Duncan (2001), constata-se que apenas a rastreabilidade não é referenciada. A necessidade desta característica deve ser avaliada pelas equipes do cliente e do desenvolvimento e adaptada ao processo, se for o caso.

Prosseguindo com a análise, no método Extreme Programming, os requisitos devem ser, segundo Duncan (2001):

- *Correto, sem ambigüidade e necessários*: Com a presença do cliente *on-site*, ambigüidade e problemas de entendimento de requisitos são minimizados e de fácil solução. Os requisitos são considerados corretos se e somente se eles representam o sistema a ser construído. Uma vez que os próprios clientes escrevem as *user stories* sob o ponto de vista do interesse dos negócios, os requisitos resultantes devem ser corretos e necessários. Devido a tanta responsabilidade e liberdade, a seleção de um cliente representativo apropriado é crucial para o sucesso do projeto. Mesmo que o cliente não saiba exatamente o que deseja no início do projeto, a natureza evolucionária do desenvolvimento XP direciona o sistema para um melhor alinhamento com as necessidades do cliente.

- *Modificável*: O ciclo de vida do XP incentiva as mudanças na especificação dos requisitos em qualquer ponto do desenvolvimento do sistema. A especificação é traduzida em uma coleção de *user stories*, onde o cliente tem total poder sobre ela. Os requisitos resultantes, após a entrega de software funcional ou as mudanças no domínio de negócios, são traduzidos, através de novas *user stories,* e as alterações de prioridades são realizadas, conforme a necessidade. Em função do planejamento, testes e integração serem todos executados de forma incremental, o XP possui um alto grau no atributo de modificabilidade.

- *Verificável*: os testes de aceitação, escritos pelo cliente, e os testes unitários, escritos pelos programadores para solucionar as tarefas derivadas das *user stories* geradas pelo cliente, criam um conjunto de especificação/testes de requisitos que são verificáveis pelo princípio da presença *on-site* do cliente e seu comprometimento com o projeto. Pelo fato do cliente escrever os testes de aceitação baseados nas *user stories*, pode-se considerar uma especificação funcional armazenada em um formato não ambíguo e verificável.

- *Priorizável:* Ao escrever as *user stories*, o cliente atribui um nível de prioridade para cada uma. No plano de *release* e iteração, o cliente define quais *user stories* deseja que sejam implementadas em cada entrega. Assim, cada requisito é apontado pela sua importância relativa ao seu tempo. Além das prioridades, o custo estimado de cada *user story*, efetuado pela equipe de programadores, é agregado no fator de decisão das funções a serem entregues no ciclo a se iniciar.

- *Viável, decomponível:* O princípio de entrega de software funcional com valor para o negócio, de modo constante e em pequenas partes, garante a viabilidade dos requisitos, minimizando o risco de o cliente adquirir um produto inviável. Em XP, as partes com alto-risco, identificadas pelo cliente e pela equipe de desenvolvimento, são implementadas primeiro, de modo a componentes inviáveis serem identificados nos ciclos iniciais de forma rápida, para que o projeto possa ser abortado sem um custo expressivo. O XP faz uso da atividade de prototipação para especular sobre os componentes identificados como sendo de risco. Essa prototipação, chamada de *spike*, é executada no nível de uma exploração de código sobre a *user story* identificada.

- *Consistente:* A consistência dos requisitos é verificada a cada final do ciclo iterativo, através da realimentação obtida com a avaliação do software funcional de valor entregue para o cliente,com percepção no produto tangível sobre suas reais necessidades.

- *Completa, concisa*: O XP foca a programação como a atividade mais importante do método; assim, pouco esforço é gasto na criação de documentos formais, podendo considerar a especificação como bastante concisa. O custo dessa especificação concisa pode ser a falta de completude, que é tratado através do cliente *on-site*. Sua disponibilidade para a equipe de desenvolvimento pode ser considerada como a complementação da especificação.

O Extreme Programming, apesar de não referenciar explicitamente, possui atividades da Engenharia de Requisitos através de seu ciclo de vida, em estágios pequenos e informais, calcados em um processo disciplinado. O cliente une-se à equipe de desenvolvimento para escrever *user stories*, desenvolver testes de aceitação, fixar prioridades e responder as questões sobre os requisitos. As *user stories* são um simples escopo se comparados aos casos de uso e a sua finalidade é promover a conversação entre cliente e desenvolvedor para o refinamento dos requisitos. As *user stories* informais são transformadas em testes unitários e de aceitação que, unidos ao cliente e ao produto funcional entregue, caracterizam uma especificação.Nota-se um ganho nos atributos de não ambigüidade, requisitos corretos e verificáveis, pela forma como o cliente é incluído no processo, estando sempre presente para responder as perguntas e esclarecer dúvidas.

Goetz (2002) considera que os princípios utilizados em Extreme Programming proporcionam uma Engenharia de Requisitos eficiente para projetos com pequenas equipes, onde o tempo é um limitante extremo, através da minimização de especificações abrangentes baseado na comunicação verbal e software funcional disponibilizado rapidamente e de forma evolucionária.

Eberlein (2002) comenta que o XP é uma oportunidade para a Engenharia de Requisitos em termos de evolução, mas alerta sobre alguns pontos relevantes como: identificação da existência do real representante do negócio para compor a equipe do cliente, referência pouco clara sobre requisitos não funcionais, necessidade de uma melhor definição do processo de transformar *user stories* em tarefas para os programadores e a responsabilidade do cliente em escrever testes funcionais em termos de qualidade e efetividade.

4.4 Comparativo entre Rational Unified Process e Extreme Programming sob o ponto de vista de requisitos

Neste item será descrito o processo de desenvolvimento RUP (Rational Unified Process) de forma resumida e o fluxo de trabalho do processo referente a requisitos

de forma detalhada. É feito, então, um comparativo do RUP com o método de desenvolvimento ágil XP (Extreme Programming), do ponto de vista da Engenharia de Requisitos, por serem eles considerados os extremos em relação ao formalismo das suas atividades. Sendo o RUP um *framework*, onde seu processo pode ser particularizado para um determinado ambiente, torna-se interessante explorar quais equivalências possui com Extreme Programming, em termos de Engenharia de Requisitos, para projetos com pequenas equipes, requisitos instáveis e ciclos rápidos de desenvolvimento.

4.4.1 Rational Unified Process

RUP é um processo desenvolvido e mantido pela Rational Software Corporation. Trata-se de uma metodologia de desenvolvimento baseada em seis práticas utilizadas pela indústria de software: desenvolvimento iterativo, gerência de requisitos, arquitetura baseada em componentes, modelagem visual, verificação contínua da qualidade e mudanças controladas (Pollice, 2001).

Um projeto baseado no RUP percorre quatro fases: Concepção, Elaboração, Construção e Transição. Cada fase é composta por uma ou mais iterações, sendo que em cada iteração se utiliza o esforço de forma paralela de várias disciplinas, como: Requisitos, Análise e Projeto, Implementação, Testes e outras disciplinas de infraestrutura (Pollice, 2001).

As fases são descritas resumidamente a seguir:

- *Concepção.* A principal meta, na fase de concepção, é obter a colaboração de todos os *stakeholders* para os definir os objetivos do projeto. A fase de concepção é básica para os esforços de desenvolvimento, pois existe um risco significativo com relação a requisitos e negócio que devem ser explorados antes que o projeto seja executado.

  Os objetivos básicos da fase de concepção incluem (Booch, 1999):

- Estabelecer o escopo do projeto e condições limites, incluindo a criação da visão do projeto, critérios de aceitação, e a definição do que deve ser feito e o que não dever ser feito no produto.

- Capturar os casos de uso críticos do sistema, os cenários básicos de operação que irão direcionar uma grande parte da modelagem.

- Propor uma arquitetura candidata baseada em cenários primários de funcionalidade.

- Propor a estimativa inicial do custo total e dos prazos para todo o projeto.

- Estimar riscos potenciais.

- Preparar o ambiente de suporte para o projeto.

- *Elaboração.* A meta da fase de elaboração é definir uma *baseline* da arquitetura do sistema, proporcionando uma base estável para a maior parte do esforço de projeto e implementação referente à fase de construção. A arquitetura envolve a consideração dos requisitos mais significativos e a avaliação de seu risco. A estabilidade da arquitetura pode ser avaliada através de uma ou mais protótipos de arquitetura.

Os objetivos básicos da fase de elaboração são (Booch, 1999):

- Certificar que a arquitetura, os requisitos e o planejamento são suficientemente estáveis, e os riscos atenuados, de modo a prever e determinar o custo e prazo para o desenvolvimento completo.

- Capturar todos os riscos significativos de arquitetura do projeto.

- Estabelecer a *baseline* de arquitetura.

- Produzir um protótipo evolucionário de componentes com qualidade de produção, como também protótipos de caráter exploratórios para atenuar os riscos.

- Certificar que a arquitetura *baseline* fornecerá suporte para os requisitos do sistema a um custo e tempo razoáveis.

- Estabelecer o ambiente de suporte.

- *Construção.* A meta da fase de construção é identificar os requisitos restantes e complementar o sistema baseado na arquitetura definida. A fase de construção possui a mesma característica que um processo de manufatura, onde é enfatizada a gerência de recursos e o controle de operações para custos, prazos e qualidade. Neste sentido, a gerência proporciona uma transição de um desenvolvimento de uma propriedade intelectual durante a fase de concepção e elaboração, para o desenvolvimento de produtos funcionais durante a fase de construção e transição.

Os objetivos principais da fase de construção são (Booch, 1999):

- Minimizar o custo de desenvolvimento pela otimização de recursos e minimizando correções desnecessárias e retrabalho.

- Alcançar qualidade adequada.

- Alcançar versões funcionais do produto.

- Completar análise, projeto, desenvolvimento e testes de todos os requisitos funcionais.

- Gerar um produto desenvolvido de forma iterativa e incremental, apto para a transição ao ambiente de produção. Isto implica na descrição dos casos de uso restantes, adendos de modelagem, complementação da implementação e testes do software.

- Avaliar se o software, o ambiente e os usuários estão prontos para a aplicação ser instalada.

- Realizar o trabalho com um certo grau de paralelismo das atividades de desenvolvimento das equipes.

- *Transição.* O foco da fase de transição é transferir o software para os usuários finais. A fase de transição pode se estender por várias iterações e incluir o teste do produto para sua liberação, e pequenos ajustes baseados na realimentação dos usuários. Neste ponto do ciclo de vida, esta realimentação deve focar, principalmente, o ajuste fino do produto, da configuração, da instalação e das questões de usabilidade.

Os objetivos principais da fase de transição são (Booch, 1999):

- Realizar um beta teste para validar o sistema desenvolvido com relação às expectativas dos usuários.

- Realizar o treinamento dos usuários e dos operadores.

- Realizar atividades de refinamento tais como, correção de erros, ganhos de desempenho e usabilidade.

- Avaliar as *baselines* de desenvolvimento, em função da visão completa do projeto e critério de aceitação do produto.

- Preparar o usuário auto-suficiente em termos da utilização do produto.

- Obter a concordância dos *stakeholders* de que as *baselines* de implantação do produto estão completas.

- Obter a concordância dos *stakeholders* de que as *baselines* são consistentes com o critério de avaliação da visão do negócio.

A característica chave do RUP é a diminuição do risco. Através de três idéias principais: casos de uso, arquitetura e desenvolvimento iterativo e incremental, o RUP atua de forma a cobrir as necessidades de desenvolvedores para construir ferramentas para suporte de automação de processos, fluxos de trabalhos individuais, diferentes modelos, e integrar as atividades e seus modelos através do ciclo de vida do projeto (Booch, 1999).
A figura 26 exemplifica o processo sob uma perspectiva da iteratividade.

Figura 26 - Rational Unified Process – (Pollice, 2001)

4.4.2 Requisitos no Rational Unified Process

Booch (1999) define que o propósito essencial do fluxo de trabalho de requisitos no Rational Unified Process é direcionar o esforço de desenvolvimento para o sistema correto. Isto é alcançado pela descrição dos requisitos do sistema (condições e capacidades que o sistema deve possuir) e um contrato entre o cliente e os

desenvolvedores do sistema sobre o que deve e não deve ser feito sobre o sistema em questão. Definir um sistema significa traduzir e organizar o entendimento das necessidades dos *stakeholders* em uma descrição significativa do sistema a ser construído (Ericsson, 2002).

O maior desafio desta filosofia é que o cliente, considerado o especialista principal do negócio, deve ser capaz de ler, entender e absorver os resultados da captura dos requisitos. Para atingir este objetivo deve-se utilizar a linguagem do cliente para descrever estes resultados (Rational, 2001).

Uma vez capturados, os requisitos devem auxiliar o gerente de projeto a planejar as iterações e *releases* do produto para o cliente.

As atividades do fluxo de trabalho de requisitos do Rational Unified Process e os artefatos resultantes assumem diferentes formas durante as diferentes fases (Concepção, Elaboração, Construção e Transição) e suas iterações:

- Na fase de concepção, os analistas identificam a maioria dos casos de uso, delimitando o sistema e o escopo do projeto, e detalhando os mais críticos.

- Durante a fase de elaboração, os analistas capturam a maioria dos requisitos remanescentes, de modo a dimensionar o esforço que será requerido para o projeto. A meta é capturar cerca de 80% dos requisitos e ter descoberto a maioria dos casos de uso ao final da fase de elaboração.

- Os requisitos restantes serão capturados durante a fase de construção.

- Na fase de transição não existe nenhuma captura significativa de requisitos, a menos que exista mudança de requisitos.

A descrição do fluxo trabalho de requisitos do Rational, Unified Process, segundo a Rational (2001), está apresentada na figura 27.

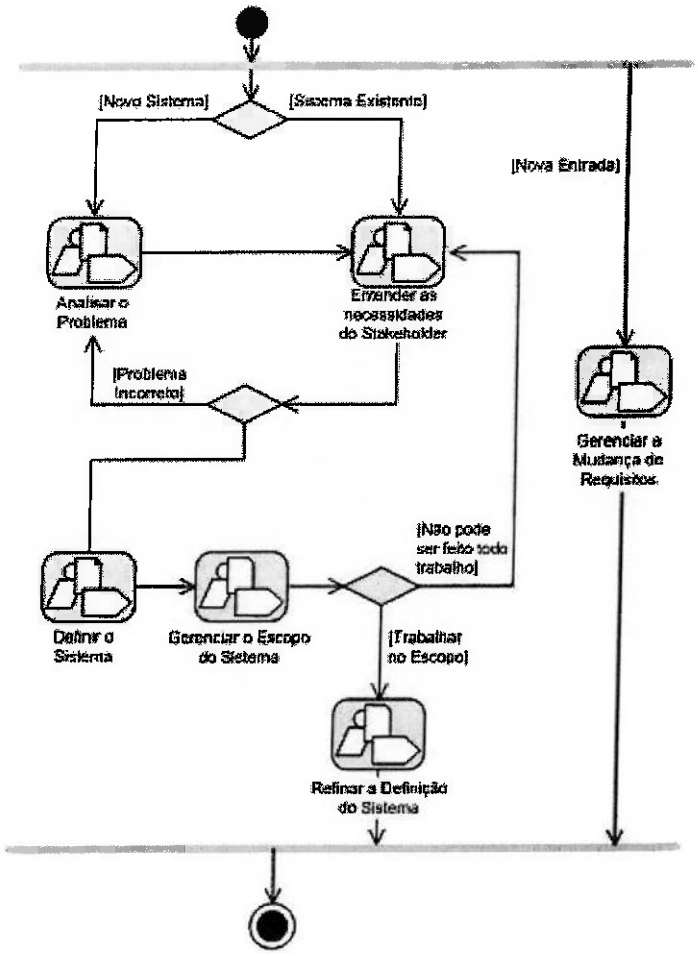

Figura 27 - Fluxo de Trabalho de Requisitos – (Rational, 2001)

As atividades do fluxo, mostradas na figura 27 podem ser detalhadas em relação aos seus objetivos (Rational, 2001):

- *Análise do Problema:* Seus objetivos principais são: obter um acordo sobre o problema ser resolvido, identificar os *stakeholders*, definir as interfaces do sistema e identificar condições e limites impostos pelo sistema.

- *Entendimento das necessidades dos stakeholders:* Seu propósito é coletar e elicitar as reais necessidades dos *stakeholders* do projeto, de modo inteligível.

- *Definição do sistema:* Seus objetivos são: alinhar a equipe de projeto no entendimento do sistema, executar uma análise de alto-nível sobre os resultados das necessidades coletadas dos *stakeholders*.

- *Gerência do escopo do sistema*: Seus objetivos são: priorizar e refinar a seleção de funções e requisitos que deverão ser incluídos na iteração corrente, definir a coleção de casos de uso que representam, de forma significativa, as funções principais do sistema, e definir quais atributos de requisitos e de rastreabilidade serão mantidos no projeto.

- *Refinamento da definição do sistema:* Os propósitos desta atividade são: definir em detalhe o fluxo de eventos de casos de uso, detalhar as especificações suplementares, desenvolver uma Especificação de Requisitos de Software detalhada, se necessário, e modelar e prototipar a interface do usuário.

- *Gerência da mudança de requisitos:* Seus objetivos são: avaliar formalmente os pedidos de mudanças de requisitos, determinar seus impactos na coleção existente de requisitos, estruturar o modelo de casos de uso, configurar os atributos apropriados dos requisitos e da rastreabilidade e verificar formalmente os resultados dos requisitos se estão em conformidade com a visão do usuário sobre o sistema.

Do ponto de vista da geração dos artefatos, as seguintes atividades são executadas no fluxo de trabalho de requisitos no RUP:

- *Desenvolvimento do Plano de Gerência de Requisitos.* Desenvolver um plano de documentação dos requisitos, seus atributos para rastreabilidade e gerência dos requisitos.

- *Desenvolvimento da Visão do Projeto.* Obter um acordo sobre quais problemas o sistema precisa resolver, identificar os *stakeholders* para o sistema, definir os limites do sistema, descrever as funções básicas do sistema.

- *Elicitação das necessidades dos stakeholders.* Entender quem são os *stakeholders* para o projeto, capturar os requisitos de forma completa, priorizar as requisições dos *stakeholders*.

- *Captura do vocabulário comum.* Definir um vocabulário comum que pode ser usado em todas as descrições textuais do sistema, especialmente nas descrições dos casos de uso.

- *Identificação dos Atores e dos Casos de Uso.* Delinear as funcionalidades do sistema, definir quem e o que irá interagir com o sistema, dividir o modelo em pacotes com atores e casos de uso, criar diagramas do modelo de casos de uso, realizar uma inspeção do modelo de casos de uso.

- *Gerência das dependências.* Utilizar os atributos e a rastreabilidade dos requisitos de projeto para auxiliar na gerência do escopo do projeto e gerência da mudança dos requisitos.

- *Estruturação do Modelo de Casos de Uso.* Extrair o comportamento dos casos de uso que necessitam ser considerados como casos de uso abstratos (comportamento comum, opcional, excepcional), encontrar novos atores abstratos que definem regras que são compartilhados por vários atores.

- *Priorização dos casos de uso.* Definir parâmetros para a seleção do conjunto de cenário e casos de uso que serão analisados na iteração corrente, definir a coleção de cenários e casos de uso que possuam alguma representação significativa com funcionalidade principal, definir o conjunto de casos de uso e cenários que possuem uma cobertura substancial sobre a arquitetura ou que realcem ou ilustrem um ponto específico ou delicado na arquitetura.

- *Detalhamento dos casos de uso.* Descrever o fluxo de eventos dos casos de uso em detalhes, de modo que o cliente e os usuários possam entendê-los.

- *Detalhamento dos Requisitos de Software.* Coletar, detalhar e organizar um conjunto de artefatos que descrevem completamente os requisitos de software do sistema ou do subsistema.

- *Modelagem da Interface do Usuário.* Construir um modelo de interface de usuário que forneça suporte para a melhoria da usabilidade.

- *Prototipação da Interface do Usuário.* Criar um protótipo de interface do usuário.

- *Revisão dos Requisitos.* Verificar formalmente se os resultados do fluxo de trabalho de requisitos estão em conformidade com a visão do cliente sobre o sistema.

Os seguintes artefatos podem ser gerados pelo fluxo de trabalho de requisitos:

- *Glossário.* Define os termos mais importantes para o projeto.

- *Requisitos de Stakeholder.* Contém qualquer tipo de requisitos que um *stakeholder* pode ter sobre o sistema a ser desenvolvido.

- *Plano de Gerência de Requisitos.* Descreve a documentação dos requisitos, tipos de requisitos e seus respectivos atributos.

- *Casos de Uso.* Define uma seqüência de ações que um sistema executa e os resultados de valor para um determinado ator.

- *Especificação de Requisitos de Software.* Contém os requisitos de software para o sistema.

- *Visão.* Define o ponto de vista do *stakeholder* sobre o produto a ser desenvolvido especificando, nos termos dos *stakeholders*, necessidades chaves e funções.

- *Modelo de Casos de Uso.* Modela as funções pretendidas para o sistema e seu ambiente, e serve como um contrato entre o cliente e o desenvolvedor.

- *Especificação Suplementar.* Captura os requisitos de sistema que não podem ser capturados nos casos de uso e seu modelo.

- *Atributos dos Requisitos.* Contém um repositório de requisitos do projeto, atributos e dependências para a gerência dos requisitos.

- *Storyboard.* É a descrição lógica e conceitual de como o caso de uso se comportará através da interface do usuário, incluindo a interação entre ator e sistema.

- *Protótipo da Interface do Usuário.* É o protótipo funcional da interface do usuário.

4.4.3 Requisitos em RUP versus Requisitos em XP

Nesta seção são discutidas, de forma comparativa, as atividades do fluxo de trabalho de requisitos do RUP, expostas em 4.4.2, em relação ao método Extreme Programming. As atividades são aquelas apresentadas na figura 27 e são as seguintes:

- Análise do problema

- Entendimento das Necessidades dos *Stakeholders*

- Definição do Sistema

- Gerência do Escopo do Sistema

- Refinamento da Definição do Sistema

- Gerência das Mudanças de Requisitos.

4.4.3.1 Análise do Problema

RUP

A atividade primária é desenvolver a Visão do Projeto. A visão expressa os requisitos iniciais com as características chaves que o sistema deve possuir para resolver os problemas críticos e encontrar as reais necessidades dos *stakeholders* (Ericsson, 2002), que são traduzidas em uma coleção de funções de alto nível de abstração que o sistema deve possuir. Estas funções devem ser classificadas, através de um peso relativo e da origem da necessidade, possibilitando que as dependências possam ser gerenciadas.

XP

A Visão do Projeto, em XP, corresponde à conceituação do sistema. Ela uniformiza as expectativas das equipes do cliente e de desenvolvimento, através de uma compreensão do sistema que deve ser criado. Essa compreensão é obtida por meio de reuniões entre os clientes e os desenvolvedores e registrada através de uma declaração sobre a finalidade da criação ou ampliação do sistema. Os clientes são autores e responsáveis por essa declaração, cabendo aos desenvolvedores avaliar os riscos, os impactos iniciais e a tecnologia para atender a solicitação. Caso os clientes não consigam conceituar o sistema, utiliza-se a Metáfora que representa um conceito comum às duas equipes (cliente e desenvolvedores), capaz descrever e conceituar o sistema, servindo como ponto de partida para a compreensão inicial do sistema.

RUP

A análise do problema também identifica os atores principais, identificando algumas maneiras que eles irão interagir com o sistema, onde as descrições devem estar mais para o processo do negócio do que para o comportamento do sistema. Os *stakeholders* são consultados sob vários pontos de vista para ajudar a refinar a descrição do problema e participam da negociação da priorização das principais funções, objetivando um entendimento geral dos recursos e esforços necessários para o desenvolvimento.

XP

Não existe e identificação formal dos atores. Os casos de uso que são origem dessas informações no RUP correspondem às *user stories* em XP. A diferença entre eles é que o caso de uso é uma coleção completa de ações iniciadas por um ator, alguém ou algo externo ao sistema, proporcionando um valor visível e um *user story* é apenas um escopo da funcionalidade, escrita em poucas palavras, necessária para solucionar um problema e utilizado com um ponto de partida para conversas futuras entre cliente e desenvolvedor. Um caso de uso pode conter vários *user stories* (Pollice, 2001). No XP, o fato do cliente fazer parte da equipe de desenvolvimento e ser responsável pelas *user stories* dispensa o levantamento formal dos atores do sistema.

RUP

Devido ao envolvimento de várias pessoas de diferentes capacitações e habilidades, torna-se importante a definição de uma terminologia que será utilizada pelo projeto. Inicialmente os termos serão definidos no glossário que deverá ser mantido através do ciclo de vida do projeto.

XP

Não possui um glossário de termos, que é substituído pelas *user stories* e pela verbalização entre os membros das equipes do cliente e de desenvolvimento (Smith, 2001). A comunicação face-a-face é outra prática fundamental em XP, que pode justificar a inexistência de um glossário como um artefato específico pois, pela presença constante do cliente na equipe de projeto e pela sua colaboração contínua, a transferência de conhecimento acontece através desse canal. O código acaba

refletindo o glossário do projeto, pois enquanto se desenvolve o sistema, aprende-se e refina-se continuamente o vocabulário que é usado para definir suas classes (Astels, 2001).

RUP

O desenvolvimento de um plano de gerência de requisitos é iniciado e deverá fornecer informações sobre os artefatos de requisitos que deverão ser desenvolvidos, os tipos de requisitos que deverão ser gerenciados para o projeto, os atributos dos requisitos que deverão ser coletados e a rastreabilidade de requisitos que será utilizada.

Embora a prioridade, o esforço, os recursos, as estimativas de mudanças e o controle de dependências sejam enfatizadas nas fases iniciais, é importante ressaltar que devem continuar por todo o ciclo de desenvolvimento do projeto (Ericsson, 2002).

XP

Os requisitos não são coletados formalmente, eles emergem das iterações e da colaboração entre as equipes. Não existe um plano de gerência de requisitos em XP, mas o existe o plano de uma iteração. O planejamento de uma iteração envolve a priorização das *user stories* definidas pelo cliente e a verificação das dependências entre as *user stories* sob o ponto de vista de requisitos. Cada *user story* é estimada pela equipe de desenvolvedores, em termos de esforço, e um custo é estimado e fixo para a iteração.Um dos princípios básicos dos Mmétodos Áágeis e, conseqüentemente, do XP é que todas as mudanças são bem vindas. A mudança é incentivada no processo do XP, pois com a ela existe o aprendizado e o refinamento dos requisitos. O produto de valor entregue para o cliente responde como artefato de requisito, sob o ponto de vista da aceitação e certificação dos requisitos, através da sua realimentação constante. Astels (2001) afirma: "*A iteração é um reconhecimento de que fazemos as coisas erradas antes de fazê-las certas...Criamos algo rapidamente para podermos criar algo melhor da próxima vez*".

4.4.3.2 Entendimento das Necessidades dos *Stakeholders*

RUP

O propósito principal desta atividade é coletar e elicitar informação dos *stakeholders* envolvidos no projeto, de modo a entender quais são os seus reais requisitos. Young (2001) define os reais requisitos como conjunto de todos os requisitos que refletem as expectativas e as necessidades dos clientes, agregando valor ao seu trabalho e ao seu negócio. Esta atividade é executada, principalmente durante as fases de concepção e elaboração.

Os requisitos possuem muitas fontes em projeto de software. Eles se originam de qualquer um que tenha interesse no projeto. Cliente, parceiros, usuários finais, especialistas do domínio, gerência, membros da equipe de projeto, política da empresa e agências reguladoras são algumas fontes de requisitos. Isto torna importante saber como os requisitos devem ser capturados, como ter acesso a eles e como eleger e elicitar a informação para o projeto de software (Ericsson, 2002).

O analista de sistema, em colaboração com os principais *stakeholders*, identificam os *stakeholders* adicionais, capturam as necessidades e determinam quais os requisitos e as funções chaves por meio de entrevistas, *workshops*, *storyboards*, casos de uso do processo do negócio e outras técnicas. Um ou mais analistas de sistemas atuam como facilitadores para estas sessões. Os analistas de sistemas armazenam, categoriza e priorizam os principais requisitos dos *stakeholders*.

XP

A principal forma de entendimento das necessidades dos *stakeholders* é a entrega de software funcional. " *Após um sistema assumir a sua forma concreta, nós podemos falar sobre o que ele tem de certo ou não. Isso acontece porque existe uma estrutura de referência*" (Astels, 2002). As *user stories* e a comunicação face-a-face procuram criar um canal de comunicação entre o modelo mental do cliente e o modelo mental do desenvolvedor, e o software funcional transforma este entendimento em algo tangível. As realimentações dos clientes, sobre a entrega, representam a real necessidade sobre um modelo concreto de referência e levam a um melhor entendimento dos requisitos. "*...quando o sistema é entregue, a equipe do cliente pode querer fazer alterações. Entretanto, essa é a natureza do software, ser fácil e facilitar as alterações...Embora as user stories possam ser escritas a qualquer*

momento, após a atividade inicial, elas são escritas com mais freqüência na entrega. Isso é um indicador de sucesso" (Astels, 2002).

RUP

Baseado em um melhor entendimento das necessidades, o analista de sistemas refina a documento de Visão do Projeto, procurando desenvolver uma declaração do objetivo do produto. Em duas ou três sentenças, esta declaração deve estabelecer o real valor do projeto. Esta declaração deve conter os usuários alvos, os problemas a serem resolvidos, os benefícios a serem atingidos e o ganho de competitividade. Todos os membros da equipe devem entender este tema de projeto. O glossário de termos é atualizado para facilitar um entendimento comum dos termos.

XP

A conceituação não é uma atividade exclusiva e deve ser refinada em qualquer momento durante o projeto, o planejamento, o desenvolvimento ou a entrega, objetivando uma melhor compreensão dos requisitos para o sistema e da abordagem técnica que será usada para realizar os requisitos (Astels, 2002). Em XP o glossário, apesar de não existir formalmente, é refinado continuamente, pois é um processo constante de aprendizado de termos e significados que são compartilhados pelos membros das equipes. Do ponto de vista da equipe de desenvolvimento é uma imersão no mundo do cliente.

4.4.3.3 Definição do Sistema

RUP

Definir um sistema significa traduzir e organizar o entendimento das necessidades dos *stakeholders* em uma descrição significativa do sistema a ser construído (Ericsson, 2002).

O objetivo desta atividade é uniformizar o entendimento da equipe sobre o sistema a ser construído, executar uma análise de alto nível sobre os resultados coletados das necessidades dos *stakeholders*, refinar a Visão de Projeto para incluir funções no sistema, refinar o modelo de casos de uso, e formalizar os resultados em modelos e documentos (Rational, 2001).

O glossário está atualizado para refletir o entendimento corrente sobre os termos usados para descrever as características e os requisitos capturados no modelo de casos de uso e nas especificações complementares.
O analista de sistema utiliza a coleção de funções definidas no refinamento da Visão do Projeto, para derivar e para descrever os casos de uso que representam o comportamento esperado do sistema, segundo a visão dos usuários. O caso de uso serve como um contrato entre o cliente, os usuários e os desenvolvedores em como as funções selecionadas irão trabalhar no sistema. Isto ajuda a fixar expectativas realistas e metas para os desenvolvedores e ajuda os clientes e os usuários a validar que o sistema está indo de encontro as suas expectativas.
A definição do sistema irá focar o detalhamento na identificação dos atores e casos de uso e expandir os requisitos não funcionais globais. É executado, tipicamente, nas iterações iniciais das fases de concepção e elaboração, embora possam ser revisados conforme a necessidade, como forma de responder a mudanças de requisitos sob controle da gerência do escopo.

XP
A uniformização do entendimento do sistema a ser construído é fundamentada nos valores de colaboração e comunicação face-a-face.
As *user stories*, escritas pelo cliente, diferentemente dos casos de uso, são um compromisso para uma conversa futura entre cliente e desenvolvedor, e funcionam como um balizador para o planejamento das iterações e estimativas de esforços (Beck, 1999).
As seqüências de eventos esperadas pelo cliente estarão no próprio produto funcional, proporcionando de forma tangível a certificação dos requisitos pretendidos nas *user stories*. Através da realimentação do cliente *on-site*, os requisitos vão emergindo e as mudanças necessárias vão sendo absorvidas pelo processo.
Esta atividade percorre todo o ciclo de vida do software em função de seu processo característico: iterativo e incremental.

4.4.3.4 Gerência do Escopo do Sistema

RUP

A gerência do escopo é uma atividade de controle da coleção de requisitos do sistema, de forma a se adequar aos recursos disponíveis para o projeto (tempo, pessoas e dinheiro), devendo ser realizado de forma contínua, com desenvolvimento iterativo e incremental, com o desmembramento do escopo em porções menores e melhor gerenciáveis (Ericsson, 2002).

Utilizar atributos de requisitos como prioridade, esforço e risco, como base para negociação da inclusão de requisitos, é particularmente uma técnica usual para a gerência de escopo.

Seus objetivos são: priorizar e refinar a seleção de funções e requisitos que são incluídos na iteração corrente, definir a coleção de casos de uso que representam algum significado para uma funcionalidade central e definir quais atributos de requisitos e rastreabilidade devem ser mantidos.

Após identificar os requisitos no nível de funcionalidade, descrever a maioria dos atores, casos de uso, e outros requisitos descritos nas especificações suplementares, o analista de sistemas deve atribuir valores, de forma mais precisa possível, para os atributos dos requisitos como prioridade, esforço, custo e risco. Isto permite um melhor entendimento de como determinar o escopo inicial do sistema a ser entregue e também pode possibilitar o arquiteto de sistemas identificar casos de uso arquiteturais do sistema.

XP

A gerência de escopo em XP se dá através das estimativas de esforço baseadas nas *user stories*, seu respectivo custo (o XP não menciona nenhum método específico de quantificação da estimativa de esforço e a estimativa de custos iniciais é feita apenas por vivência e experiência dos programadores). "*Muitos desenvolvedores novos em XP podem achar difícil estimar alguma coisa. E eles estão certos, mas nós melhoramos com a experiência. Esse aspecto da XP talvez seja o mais assustador para aqueles que nunca o experimentaram antes*" (Beck, 1999). O desmembramento do escopo em porções menores é feito através do plano de *releases*, baseado em pilhas de *user stories*. Pilhas de *user stories* representa um conjunto de cartões, onde as *user stories* estão escritas, priorizadas e agrupados por uma determinada afinidade, por exemplo, um módulo ou parte específica do sistema. A criação de um plano de

release permite que as equipes obtenham uma estimativa inicial do custo geral do projeto, fornecendo dados para questões de custo e benefício e viabilidade do projeto.

A meta do plano de *release* é ajudar o cliente a identificar as funções que o sistema deve possuir, dar aos programadores a chance de explorar a tecnologia e fazer estimativas, e fornecer uma idéia de tempo para todo projeto (Wake, 2001).

O controle do risco é favorecido pelo plano de *releases* que devem ser o mais curtos possíveis e por *spikes* de planejamento. *Spike* de planejamento é uma exploração, em código, de uma determinada *user story*. O objetivo de um *spike* não é produzir código e sim uma estimativa (Beck, 1999).

A definição de um esboço ou um esqueleto da arquitetura do sistema não é uma prioridade no XP. A arquitetura emerge conforme as iterações ocorrem no processo de desenvolvimento. Pelo fato da adoção contínua de mudanças que podem ter um impacto global no projeto, inclusive na arquitetura, sua definição nas fases inicias ou um direcionamento pela mesma pode criar resistência no princípio da aceitação da mudança em XP.

"XP diz - adote a mudança, enquanto métodos direcionados pela arquitetura dizem - Algumas coisas são difíceis de mudar, então planeje o esqueleto primeiro" (Wake, 2001).

A figura 28 exibe as atividades de requisitos em XP.

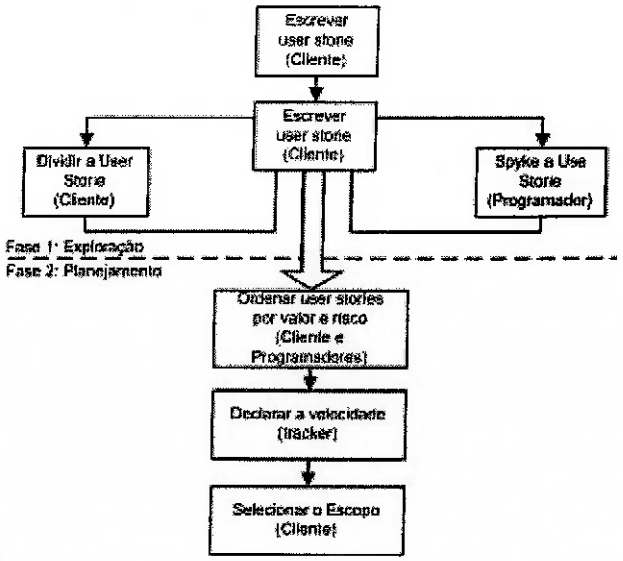

Figura 28 - Atividades Requisitos XP: Exploração e Planejamento (Wake, 2001)

RUP

O plano de iteração, também conhecido como plano de desenvolvimento, é desenvolvido em conjunto pelo gerente de projeto e pela equipe de desenvolvimento. O plano de iteração define o número e a freqüência de iterações planejadas para entrega. Os elementos de alto risco dentro do escopo devem ser planejados para as iterações iniciais.

XP

O plano de iteração é criado a partir do plano de *release*. O *release* é dividido em diversas iterações com tempo determinado, onde as *user stories* são programadas como parte dessas iterações para estender o *release* produzido pela iteração anterior, levando em conta as dependências das *user stories* e as prioridades definidas pela equipe do cliente. A duração das iterações deve ser tão curta quanto possível, levando em conta fatores como a cultura da organização, o ambiente de desenvolvimento e as equipes de trabalho, proporcionando uma redução de riscos com relação aos requisitos. O XP ressalta a importância da primeira iteração, pois ela define o foco principal do sistema e o seu sucesso cria o engajamento das equipes no sucesso do projeto, iniciando o movimento iterativo e colaborador do método. "*O período de tempo que leva à entrega desse primeiro release é considerado como o*

mais perigoso. A entrega desse sistema central é um marco importante. Ele prova para a equipe do cliente que XP funciona. Ele prova para a equipe de desenvolvimento que eles podem fazer com que o projeto aconteça..." (Beck, 1999).

A figura 29 exibe a continuação do processo de requisitos em XP.

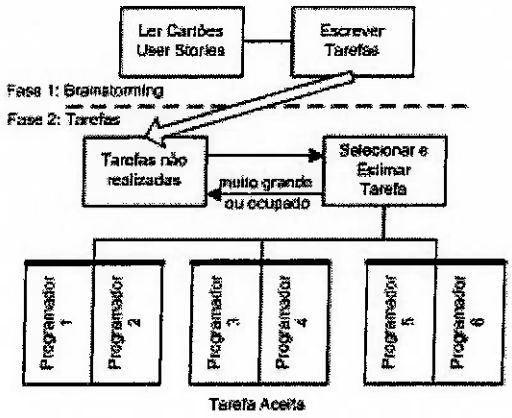

Figura 29 - Atividades de Requisitos: Brainstorming *user story* e definição tarefas XP (Wake, 2001)

RUP

O analista de sistemas deve determinar os valores de prioridade, esforço, custo, risco, etc., em colaboração com os *stakeholders* apropriados, para alimentar o repositório dos atributos de requisitos. Isto será utilizado pelo gerente de projeto no planejamento das iterações e proporcionará, ao arquiteto de software, identificar os casos de uso mais significativos para a arquitetura.

XP

A equipe de desenvolvimento determina os valores de custo e esforço necessários para cada *user story* escrita pelo cliente. O cliente define a prioridade dos requisitos, através das *user stories* e planos de *release* e, com ajuda da equipe de desenvolvimento, riscos e custos envolvidos para cada iteração. Com a entrega freqüente do produto de software de forma tangível, o cliente gerencia os requisitos e direciona o plano das iterações conforme suas necessidades e perspectivas. Em XP não existe uma preocupação com definição de uma arquitetura para o sistema nas fases iniciais, apenas considera como um fenômeno emergente à medida que as iterações acontecem.

4.4.3.5 Refinamento da Definição do Sistema

RUP

O propósito desta atividade é refinar os requisitos, descrevendo o fluxo de casos de uso em detalhes, expandir as especificações suplementares, desenvolver uma Especificação de Requisitos de Software e modelar e prototipar a interface do usuário.

O refinamento da definição do sistema se inicia com o detalhamento dos casos de uso, com uma descrição breve dos atores, e o entendimento do escopo do projeto, refletindo a repriorização das funcionalidades no escopo que se acredita ser viável de ser atingido em termos de orçamento e datas.

O resultado desta atividade é um entendimento mais profundo das funções do sistema, expresso em casos de uso detalhados, especificação suplementar revisada e detalhada, além dos elementos de interface de usuário. Uma Especificação de Requisitos de Software pode ser desenvolvida, se necessário, como complementação aos casos de uso detalhados e Especificações Suplementares.

XP

Não possui um documento formal de especificação de requisitos; a sua necessidade deve partir do cliente que deve descrevê-lo como uma *user story* que fará parte do projeto. O refinamento dos requisitos é resultado da entrega rápida e freqüente de software funcional para a equipe do cliente que, a partir de uma referência tangível, escreve novas *user stories* para as próximas iterações. A prototipação e a modelagem da interface do usuário não são atividades específicas em XP, mas pode ser realizada como um resultado que emerge do processo iterativo e incremental. As prototipações são referidas em XP como ferramentas de avaliação de riscos, para estimar a velocidade da equipe de desenvolvimento, para explorar alternativas de arquiteturas e para refinar uma determinada funcionalidade (Booch, 1998).

RUP

Torna-se importante trabalhar em conjunto com os usuários e potenciais usuários do sistema quando da modelagem e prototipação da interface de usuário. Isto pode ser

usado para direcionar a usabilidade do sistema, para ajudar a emergir requisitos não descobertos e para refinar a definição de requisitos.
Os resultados destes refinamentos da definição do sistema podem ser submetidos à atividade de gerência de escopo.Uma vez que se sabe mais sobre o sistema, as prioridades tendem a mudar.
O refinamento da definição do sistema deve considerar duas questões chaves: o desenvolvimento de descrições mais detalhadas da definição do sistema e a verificação se o sistema está de acordo com as expectativas e necessidades que os *stakeholders* tenham descrito.

XP

A entrega constante de software funcional e o cliente como membro da equipe de projeto favorecem o refinamento dos requisitos do sistema. A idéia de exploração e aprendizado com os requisitos caminham juntas em um projeto de XP. Conforme as iterações acontecem, novos requisitos emergem e mudanças são manifestadas. Essas alterações são incorporadas nas iterações futuras nos planos de *release* e iteração. "*A equipe de desenvolvimento precisa entregar constantemente e de forma iterativa versões do sistema para o cliente. O plano de release é utilizado para descobrir pequenas unidades de funções que façam um bom sentido para o negócio e possam ser integradas ao ambiente dos usuários nas fases iniciais do projeto.*" (Beck, 1999).
A realimentação constante do cliente, através dos testes e dos *releases*, fornece a verificação das expectativas do cliente e do software entregue, que representam dois dos valore do XP – Comunicação e Realimentação. "*Com o cliente posicionado desta maneira, a necessidade de muitos documentos intermediários pode ser reduzida...*" (Smith, 2001).

4.4.3.6 Gerência das Mudanças de Requisitos

RUP

A mudança de alguns requisitos é desejável, e pode significar que a equipe está engajada com os *stakeholders*. O inimigo não é a mudança, mas sim a não gerência. A capacidade de acomodar as mudanças de requisitos é uma medida da sensibilidade

e flexibilidade operacional que a equipe do projeto e os *stakeholders* possuem e é um dos atributos que contribuem para o sucesso do projeto (Ericsson, 2002).

O propósito desta atividade é avaliar formalmente a mudança de requisitos solicitada e determinar o seu impacto nos requisitos existentes, além de estruturar os casos de uso, configurar apropriados atributos de requisitos e verificar formalmente se o resultado da disciplina de requisitos está em conformidade com o ponto de vista do cliente sobre o sistema.

As mudanças nos requisitos afetam naturalmente os modelos produzidos na disciplina de Análise e Projeto, os modelos de teste criado na disciplina de Teste e o material de suporte ao usuário da disciplina de implementação. A relação de rastreabilidade identificada na gerência de dependência identifica as relações entre os requisitos e os outros artefatos. Estes relacionamentos são chaves para o impacto da mudança de requisitos.

Outro conceito importante é a trajetória histórica dos requisitos. Pela captura natural e racional das mudanças de requisitos, os revisores recebem a informação necessária para responder à mudança requisitada.

XP

As mudanças são bem vindas em Extreme Programming, pois elas caracterizam que o processo de aprendizado da equipe e a emergência dos requisitos estão acontecendo. A mudança de requisitos não é encarada como um erro, mas como um avanço do produto em direção da expectativa do cliente.

A mudança é pedida pelo cliente, através de uma nova *user story* que, por sua vez, é estimada em termos de esforço pela equipe de desenvolvimento e priorizada pelo cliente para o planejamento das próximas iterações (Astels, 2002).

O princípio do cliente *on-site* proporciona o seu engajamento na colaboração com o projeto. Mudanças que acarretam grandes esforços que possam inviabilizar a iteração são analisadas em conjunto e negociadas entre as equipes.

4.5 Aderência do Extreme Programming ao Capability Maturity Model (CMM)

O SW-CMM é um modelo de avaliação de maturidade de desenvolvimento de software amplamente aceito na comunidade profissional. Nesta seção é feita uma

apresentação resumida do modelo SW-CMM como um todo, com um detalhamento da KPA (*Key Area Process*) da Gerência de Requisitos, para então discutir a aderência do método Extreme Programming a este modelo.

4.5.1 O modelo SW-CMM

Desenvolvido pelo Instituto de Engenharia de Software da Universidade de Carnegie Mellon, como um modelo para a avaliação e melhoria da maturidade organizacional para construção de software, o SW-CMM tem sido amplamente adotado pela comunidade de software (Paulk, 2001).

No CMM, foram estabelecidos os níveis de maturidade que a organização pode alcançar para desenvolver software, que são (Maldonado, 2001):

- *Nível 1 Inicial :* O desempenho no desenvolvimento de software de qualidade depende diretamente da competência das pessoas.

- *Nível 2 Repetível*: Os métodos de gerência de software são documentados e acompanhados. Políticas organizacionais orientam os projetos estabelecendo processos de gerência.

- *Nível 3 Definido:* Existe um processo de software definido na empresa, com a preocupação de ser um processo padronizado que é especializado para cada projeto de software.

- *Nível 4 Gerenciado:* O processo é medido e gerenciado quantitativamente, com a possibilidade de previsão de desempenho dentro dos limites quantificados.

- *Nível 5 Otimizado:* É focado na melhoria contínua do processo, onde a mudança de tecnologia e as mudanças no próprio processo são gerenciadas de forma a não causar impacto na qualidade do produto final.

Para cada nível do CMM, com a exceção do nível 1, existem conjuntos de metas e objetivos que devem ser satisfeitos para que a organização possa atingir um determinado nível. Esses objetivos e metas podem ser atingidos através da implantação de KPAs (*Key Process Areas*) ou áreas chave de processo (Crissis, 1993).

A seguinte tabela sumariza as KPAs por nível de maturidade do CMM (Paulk, 2001).

Tabela 5 - SW-CMM - níveis de maturidade

Nível	Foco	Key Process Áreas (KPAs)
5: Otimizado	Melhoria contínua do Processo.	Prevenção de Defeito. Gerenciamento da mudança tecnológica. Gerenciamento da mudança no processo.
4: Gerenciado	Qualidades dos Processos e Produtos.	Gerenciamento quantitativo do processo. Gerenciamento da qualidade de software.
3: Definido	Processos de Engenharia e Suporte Organizacional.	Foco no processo da organização. Definição do processo da organização. Programa de Treinamento. Gerenciamento integrado de software. Engenharia de produto de software. Coordenação intergrupos. Revisões.
2: Repetível	Gerenciamento de Projeto.	Gerencia de Requisitos

		Planejamento do Projeto de Software. Acompanhamento de Projeto de Software. Gerenciamento de subcontrato de software. Garantia na qualidade de software. Gerenciamento da configuração de software.
1: Inicial	Sem processo.	

4.5.2 XP e SW-CMM nível dois

A avaliação do XP é feita em relação à KPA – Gerência de Requisitos, pertencente ao nível 2 do modelo SW-CMM.

O propósito da Gerência de Requisitos é estabelecer e manter um acordo comum entre o cliente e a equipe de desenvolvimento, em relação às necessidades do cliente que direcionarão o projeto de software.

Esse acordo é referido como os requisitos do sistema alocado para o software, cobrindo os requisitos técnicos e não técnicos do projeto de software, e forma a base para estimativa, planejamento e desempenho, para as atividades no ciclo de vida do projeto de software (Bush, 1993).

A tabela 6 exibe as práticas necessárias para satisfazer a KPA Gerência de Requisitos no Modelo SW-CMM (Bush, 1993).

Tabela 6 - KPA Gerência de Requisitos Nível Dois

Metas	1) Os requisitos do sistema são controlados para estabelecer uma

	baseline para o gerenciamento e a engenharia de software. 2) Planos de Software, Produtos e Atividades são consistidos com os requisitos de sistema.
Compromisso	Deve cumprir o compromisso de uma política organizacional documentada para a gerência dos requisitos de software.
Capacidades	1) Para cada projeto, pela análise dos requisitos, as responsabilidades são atribuídas, alocando-as para hardware, software e outros componentes de sistema. 2) Os requisitos são documentados. 3) Recursos adequados são providenciados para a gerência dos requisitos. 4) Membros da equipe de Engenharia de Software e outras equipes relacionadas com software são treinadas para suas atividades de gerência de requisitos.
Atividades	1) A equipe de engenharia de software revisa os requisitos antes que eles sejam incorporados ao projeto de software. 2) A equipe de engenharia de software utiliza os requisitos como base para o plano de software, produtos funcionais e atividades. 3) Mudanças de requisitos são revisadas e incorporadas ao projeto de software.
Métrica	1) As métricas são criadas e utilizadas para determinar o estado das atividades, de modo a gerenciar os requisitos alocados.
Verificações	1) As atividades de gerência de requisitos são revisadas com o gerente sênior periodicamente. 2) As atividades de gerência de requisitos são revisadas com o gerente de projeto periodicamente e na ocorrência de eventos. 3) A equipe de qualidade de software revisa e/ou audita as atividades e produtos funcionais para gerência de requisitos e informação dos resultados.

Segundo Paulk (2001), o XP satisfaz o enfoque da KPA de nível 2, Gerência de Requisitos do modelo SW-CMM, considerando as condicionantes de um método voltado para equipes pequenas.

Na análise da aderência do XP à KPA de Gerência de Requisitos no Modelo SW-CMM, Ford (2002) considera como pontos fortes:

- O escopo das próximas iterações é determinado rapidamente, combinando as prioridades do negócio e as estimativas técnicas. O cliente decide o escopo, prioridade e data, através do ponto de vista do negócio, enquanto a equipe técnica estima e verifica o progresso.
- Ao final de cada iteração, as equipes técnica e do cliente reavaliam os resultados da iteração e promovem as correções dos requisitos para a próxima iteração.
- As iterações curtas facilitam a realimentação constante do cliente, promovendo a consistência das atividades, produtos e planos de software com os requisitos do sistema.
- Os recursos são controlados pela equipe do cliente através do papel do patrocinador que garante que o projeto possua as pessoas, os equipamentos e os financiamentos para atingir seus objetivos.
- Os requisitos emergem conforme as iterações do processo são realizadas. Requisitos são descobertos através das iterações curtas e revisados pela entrega freqüente de software funcional. Os requisitos emergentes são transformados em *user stories*, estimados pela equipe de desenvolvimento e priorizados pela equipe do cliente.

A necessidade de uma documentação formal escrita em SW-CMM é importante e está presente na Capacidade número dois – Os requisitos são documentados. As *user stories* isoladas não podem ser consideradas como uma documentação, pois como Beck (1999) define, elas constituem um compromisso para uma conversação entre cliente e desenvolvedor, possuindo mais valor como ferramenta de gerência,

estimativas e gerador de tarefas para os programadores. O XP de forma nativa não é capaz de satisfazer essa habilidade necessária à KPA; a documentação formal deverá ser implementada como uma *user story* e fará parte do ciclo de desenvolvimento iterativo. No entanto, Paulk (2001) considera que, pelas condicionantes do tamanho da equipe e cliente como parte integrante da equipe de desenvolvimento, a documentação de requisitos, sob o ponto de vista de resultado, pode ser alcançado pela combinação das *user stories*, cliente *on-site* e entrega contínua de software podendo satisfazer, por inferência, esse ponto da KPA.

Ford (2002) aponta, como fraqueza, o fato de XP não apresentar um treinamento formal de gerência de requisitos, assumindo que as equipes do cliente e de desenvolvimento possuem a experiência necessária para cumprir esta capacidade. No entanto, Jeffries (2001a) afirma que o treinamento está baseado nas práticas do XP, pois um dos princípios do XP é ensinar aprendendo (Beck, 1999). "*Se você não incentivar as pessoas a aprenderem, elas resistirão ao aprendizado quando ele for realmente necessário*".

Outra fraqueza apontada por Ford (2002) é a falta de métricas e medidas sobre o estado dos requisitos para a gerência de requisitos. No entanto, segundo Astels (2002), o estado dos requisitos é controlado pela entrega freqüente de software funcional ao cliente; com o software operacional em mãos, o cliente pode verificar, de forma tangível, quais requisitos foram cumpridos.

De forma geral, Paulk (2001) afirma que *"Xp possui boas práticas de engenharia que podem trabalhar bem com CMM e outros métodos estruturados. O importante é considerar cuidadosamente as práticas de XP e implementá-las no ambiente certo."*

5 CONCLUSÃO

O capítulo final discorre sobre as principais conclusões e as contribuições relativas aos capítulos apresentados e as sugestões para complementar a pesquisa e impulsionar novos trabalhos.

5.1 Conclusões e Contribuições

A principal contribuição, com relação aos tópicos apresentados, é ter explicitado, sob o ponto de vista de conceito, as atividades de Engenharia de Requisitos em dois tipos de processos de filosofias diferentes: um processo direcionado pelo planejamento que busca a previsibilidade dos requisitos (RUP) e um processo direcionado pela adaptabilidade que busca se acomodar à emergência dos requisitos em um projeto de software (XP).

Uma conclusão é que considerar ou desconsiderar o ponto de vista de um ou outro processo é altamente arriscado, se não se levar em conta uma variável de vital importância: o ambiente e o domínio onde a disciplina de Engenharia de Requisitos estará imersa. Aplicar um determinado processo, por melhor que ele seja estruturado, em um ambiente desfavorável é o primeiro passo para o fracasso do projeto.

O reconhecimento das características dos dois pontos de vista e a aplicação do processo correto nos projetos devem ser enfatizados na aplicação da Engenharia de Requisitos, ressaltando a sua capacidade de se moldar de forma efetiva e eficiente aos domínios do negócio.

Pode-se obter vantagem destas abordagens, se souber dosar e integrar, de forma harmoniosa, os dois pontos de vista buscando uma sinergia, complementando as falhas de um com as qualidades do outro e vice-versa, e obtendo requisitos de qualidade para o projeto.

Sob essa perspectiva, o Rational Unified Process possui mais profundidade para acomodar um processo mais formal de Engenharia de Requisitos e maior capacidade de talhar os seus processos, conforme o ambiente de negócios. Possui mecanismos consistentes para conseguir a mobilidade entre o processo direcionado pela previsibilidade e adaptabilidade dos requisitos.

Em termos de aplicabilidade do processo e obtenção de resultados tangíveis, os Métodos Ágeis por meio de seus valores e princípios praticados somam contribuições importantes à Engenharia de Requisitos. Estes métodos não devem ser considerados como inovações ou descoberta de alguma espécie de "bala de prata", mas apenas como um retorno saudável a alguns princípios básicos do desenvolvimento de software: o cliente como a chave para o sucesso do projeto, o processo evolucionário para diminuição de riscos e conceitos da disciplina de Engenharia de Requisitos, utilizados por todo o ciclo de vida do projeto. Princípios esses, que são adotados de forma vigorosa pelos Métodos Ágeis, a fim de proporcionar rápida resposta a um meio onde a velocidade e instabilidade de requisitos são característicos.

Outra conclusão importante é maneira pela qual os Métodos Ágeis encaram o elemento humano em seu processo. O tratamento dos requisitos, nos métodos ágeis, busca uma engenharia social entre seus participantes e interessados, ressaltando a importância dos requisitos, através da integração, comunicação e colaboração intensiva entre os envolvidos, engajando-os no alcance de seus objetivos e valorizando-os em um processo de Engenharia de Requisitos.

Considerar o elemento humano como apenas um recurso em um projeto de desenvolvimento de software, desconsiderando-o ou relegando, a um segundo plano, a sua individualidade, criatividade e inteligência, deve ser considerado como um risco a ser avaliado no projeto. A tentativa de utilizar um processo baseado em algumas heranças de disciplinas mais antigas da Engenharia que desconsideram o fator humano, em um ambiente onde a interação e a colaboração das pessoas, como

fontes geradoras e consumidoras de requisitos, é um dos fatores preponderantes para alcançar um produto correto e de qualidade, merece uma reavaliação.

Finalmente, a Engenharia de Requisitos não deve ser apenas encarada, de forma minimalista, como uma disciplina burocratizante e rígida, como é vista por alguns agilistas, pois informalmente seus valores e princípios estão presentes em todo o ciclo de vida dos Métodos Ágeis. Deve ser encarada como a arte de aplicar os conhecimentos científicos e empíricos na criação de processos efetivos e consistentes ao ambiente em que está inserido.

5.2 Continuidade da Pesquisa

Em relação à Engenharia de Requisitos em métodos ágeis, existe a necessidade de uma discussão mais detalhada sobre alguns pontos, em função de controvérsias e falta de melhor esclarecimento. Os principais pontos considerados são:

- Seleção do cliente nos métodos ágeis: Quando os representantes são vários e dispersos geograficamente, existe a dificuldade da acomodação de vários pontos de vista sobre o mesmo requisito. Para esses pontos os métodos ágeis são relativamente vagos e imprecisos quanto ao seu processo.

- Requisitos não funcionais: não foi identificada nenhuma referência direta à captação e à gerência de requisitos não funcionais em Métodos Ágeis, sendo tratados da mesma forma que os requisitos funcionais.

- Contrato de desenvolvimento: É importante discutir sobre as características de um contrato para desenvolvimento ágil, uma vez que pouca documentação é gerada e os objetivos do sistema, seus requisitos e seus incrementos vão sendo definidos à medida que o entendimento sobre o sistema vai aumentando.

- Escalabilidade: Os Métodos Ágeis e os seus processos de requisitos são condicionados para pequenas equipes, normalmente entre 10 e 12 pessoas.

> Existem referências de suas utilização para equipes de até 200 pessoas, mas pouco se comenta em relação ao detalhamento de como essa escalabilidade foi conseguida, mantendo as características de sua agilidade, principalmente em termos de comunicação e colaboração (Eberlein, 2002).

Esses pontos não devem ser considerados como uma crítica aos Métodos Ágeis, mas deve-se investir um esforço de pesquisa no entendimento de sua reação a esses eventos, seu comportamento e visualização de seus limites e condicionantes.

REFERÊNCIAS BIBLIOGRÁFICAS

(Ambler,02) Ambler, S.W., Agile Requirements Modeling, **The Oficial Agile Modeling Site.**
Disponível em www.agilemodeling.com>. Acesso em 25/05/02

(Astels,02) Astels,D.; Miller, G.; Novak, M., **A Practical Guide to eXtreme Programming**, 1.ed., Prentice Hall, 2002

(Beck el al.,01) Beck, K.;Beedle, M.;Cockburn, A.;Cunningham, W.;Fowler, M.;Grenning, J.;Highsmith, J.; Hunt, A.;Jeffries, R.;Kern, J.; Marick, B.; Martin, R.C.; Mellor, S.; Schwaber, K.;Sutherland, J.; Thomas, D.; Van Bennekum, A., **Manifesto for Agile Software Development.**
Disponível em <www.agilealliance.org>. 08/06/02.

(Beck,99) Beck, K., **eXtreme Programming Explained: Embrace Change**, 1.ed, Addison-Wesley, 1999

(Beedle, 01) Beedle, M.; Schuwaber, K., Agile Software Development with Scrum. **Jeff Sutherland´s Object Technology Web Site.**
Disponível em <www.jeffsutherland.com/scrum/index.html>.
Acesso em 13/10/02

(Boehm,00) Boehm, B.W., Spiral Development: Experience, Principles and Refinements, **Spiral Development Workshop 2000, Software Engineering Institute - Carnegie Mellon University** - CMU/SEI-2000-SR-008, 2000

(Boehm,02) Boehm, B.W.; DeMarco,T., The Agile Methods Fray, **IEEE Computer Vol 35, nº 6** - June 2002

(Boehm,81) Boehm, B.W., **Software Engineering Economics**, Prentice Hall, 1981

(Booch,98) Booch, G.; Newkirk, J.; Martin, R.C., **Object Oriented Analysis and Design with Aplications**, 2.ed.,Addison-Wesley,1998

(Booch,99) Booch, G., Jacobson, I., Rumbaugh, J., **The unified software development process**, Addison-Wesley, 1999.

(Bush,93) Bush, M.; Crissis, M.B.; Garcia, S.M., Paulk, M.C., Webber, C.V., **Key Practices of the Capability Maturity Modelo for Software V.1.1** - CMU/SEI-93-TR-025 -1993.
Disponível em <www.sei.cmu.edu>. Acesso em 17/11/02

(Charette,02) Charette, R., The Decision is in: Agile Versus Heavy Methodologies, **Cutter Consortium: Sample Issues, vol.2, nº 19**, 2002.
Disponível em <www.cutter.com/freestuff/epmu0119.html>. Acesso em 15/09/02

(Clavadetscher,98) Clavadetscher, C., User Involvement: Key to Success, **IEEE Software, Vol. 15, nº 2**, March/April 1998

(Coad,99) Coad, P.; De Luca, J.; Lefebvre, E., Java Modeling in Color with UML, **Agile Alliance**. Disponível <www.agilealliance.org/articles>. Acesso em 06/10/02

(Cockburn, 01a) Cockburn, A., **Agile Software Development**, 1.ed, Addison-Wesley, 2001

(Cockburn,01) Cockburn, A.; Jim Highsmith, **Agile Software Development: The business of innovation**, IEEE Computer Magazine Vol.34, nº 9 - September 2001

(Crissis,93) Crissis, M.B.;Curtis, B.; Paulk, M.C.; Weber C.V, **Capability Maturity Model for Software V.1.1** - CMU/SEI-93-TR-024 – 1993.
Disponível em <www.sei.cmu.edu>. Acesso em 17/11/02

(Dorfman,97) Dorfman, M.;Thayer, R.H., **Software Requirements Engineering**, IEEE Computer Society Press, 1997

(DSDM,01) DSDM Consortium, Introducing DSDM into na Organization, **DSDM Consortium**, 2001.
Disponível em <www.dsdm.org>. Acesso em 03/10/02

(Duncan,01) Duncan, R., The Quality of Requirements in Extreme Programming, **Institute for Signal and Information Processing - Mississipe State University**.
Disponível em <www.isip.msstate.edu/publications/journals/crosstalk/2001/extremeprogramming/crosstalk_final.pdf>. Acesso em 14/09/02

(Easton,02) Easton, Z.; Stapleton, J.; Tuffs, J.; West, D., Inter-operability of DSDM with the Rational Unified Process, **DSDM Consortium**, 2002.
Disponível em <www.dsdm.org>. Acesso em 05/10/02

(Eberlein,02) Eberlein, A.; Leite, J.C.S.P., Agile Requirements Definition: A View from Requirements Engineering, **International Workshop on Time Constrained Requirements Engineering 2002**.
Disponível em <http://www-di.inf.puc-rio.br/~julio/tcre-site/p4.pdf>. Acesso em 19/10/02

(Ericsson,02) Ericsson, M.; Oberg, R.; Probasco, L., Applying Requirements Management with Use Cases, **Rational Software Corporation** White Paper TP505 2002.
Disponível em <www.rational.com>. Acesso em 28/07/02

(Ford,02) Ford, S.; Man, J.; Zhou, Q., **Is XP at CMM Maturity Level Two?, University of Calgary**, Updated April 2002.Disponível em <www.enel.ucalgary.ca/~zhouq/XPCMM623.html. Acesso em 16/11/02

(Fowler, 02) Fowler, M., **The New Methodology** - update June 2002.
Disponíve em <www.martinfowler.com/articles/newMethodology.html>. Acesso em 21/07/02

(Fowler,02a) Highsmith, J., Does Agility Work?, **Software Development Magazine** - June 2002. Disponível em <www.sdmagazine.com/print/documentID=25469>. Acesso em 03/08/02

(Goetz,02) Goetz, R., How Agile Process Can Help in Time-Constrained Requirements Engineering, **International Workshop on Time Constrained Requirements Engineering 2002.** Disponível em <http://www-di.inf.puc-rio.br/~julio/tcre-site/p8.pdf>. Acesso em 14/09/02

(Highsmith,00) Highsmith, J.; Orr, K.T., **Adaptive Software Development : A collaborative aproach to managing complex systems**, Dorset House, 2000

(Highsmith,01) Fowler, M.; Highsmith, J., **The Agile Manifesto, Software Development Magazine** - August 2001. Disponível em <www.sdmagazine.com/print/documentID=11649>. Acesso em 21/07/02

(Holanda,99) Holanda, A.B., **Novo Aurélio Século XXI: O Dicionário da Lingua Portuguesa**, 3.ed., Nova Fronteira, 1999

(Janoff,00) Janoff, N.S.; Rising, L., The Scrum software development process for small teams, **IEEE Software, Vol. 17, n° 4** - Jul/Ago 2000

(Jeffries,01) Jeffries, R., Essential XP: Documentation, **Extreme Programming Resource**, update November 2001. Disponível em <www.xprogramming.com/xpmag/expDocumentationInXP.htm>. Acesso em 29/09/02

(Jeffries,01a) Jeffries, R., What is Extreme Programming?, **Extreme Programming Resource**, update August 2001. Disponível em <www.xprogramming.com/xpmag/expDocumentationInXP.htm>. Acesso em 29/09/02

(Jones,98) Jones, C., **Estimating Software Costs**, New York,McGraw Hill, 1998

(Kotonya,97) Kotonya, G.;Sommerville, I., **Requirements Engineering**, Jonh Wiley & Sons Ltd, 1997

(Maldonado,01) Maldonado, J.C.; Rocha, A.R.C.,; Weber, K.C., **Qualidade de Software**, São Paulo, Prentice Hall, 2001.

(Paulk,01) Paulk, M.C., Extreme Programming from a CMM Perspective, **IEEE Software, Vol 18, nº 6**, November/December 2001

(Pollice,01) Pollice, G., Using the Rational Unified Process for Small Projects: Expanding Upon eXtreme Programming, **Rational Software Corporation** White Paper TP183 2001. Disponível em <www.rational.com>. Acesso em 13/10//02

(Pressman,01) Pressman, R.S., **Software Engineering: a practitioner's approach**, 5th.Ed., McGraw-Hill, 2001

(Rational,01) Rational Rup Model 2001A.04.00.1B, **Rational Software Corporation**, 2001

(Rising,02) Rising, L., **Agile Meetings**, The Software Testing and Quality Engineering Magazine May/June 2002. Disponível em <members.cox.net/rising1/articles/>. Acesso em 15/09/02

(Schwaber,01) Schwaber, K., Get Ready for Scrum. **Scrum Development Process Web Site**.Disponível em <www.controlchaos.com>. Acesso em 03/11/02

(Schwaber,02) Schwaber, K., The Impact of Agile Process on Requirements Engineering, **Agile Alliance** - September 2002. Disponível em <www.agilealliance.org/articles>. Acesso em 5/10/2002

(Smith,01) Smith, J., A Comparision of RUP and XP, **Rational Software Corporation** White Paper TP167 2001. Disponível em <www.rational.com>. Acesso em 01/06/02

(Tomayko,02) Tomayko, J.E., Engineering of Unstable Requirements Using Agile Methods, **International Workshop on Time Constrained Requirements Engineering 2002,** Disponível em <http://www-di.inf.puc-rio.br/~julio/tcre-site/p1.pdf>. Acesso em 22/09/02

(Wake,01) Wake, W.C., **Extreme Programmin Explored**, 1.ed., Addison-Wesley,2001

(Wells,02) Wells, D., **Extreme Programming: A gentle introduction**, update August 2002. Disponível em <www.extremeprogramming.org>. Acesso em 27/10/02

(Wiegers,99) Karl E. Wiegers, **Software Requirements**, Microsoft Press, 1999

(Young,01) Young, R.R., **Effective requirements practices**, Addison-Wesley, 2001