UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Priscila Basto Fagundes

FRAMEWORK PARA COMPARAÇÃO E ANÁLISE DE MÉTODOS ÁGEIS.

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora:

Profª. Drª. Patrícia Vilain

Florianópolis, fevereiro de 2005.

FRAMEWORK PARA COMPARAÇÃO E ANÁLISE DE MÉTODOS ÁGEIS.

Priscila Basto Fagundes

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Dr. Raul Sidnei Wazlawick

Banca Examinadora

_____

Profª. Drª. Patrícia Vilain

_____

Prof. Dr. Vitório Bruno Mazzola

_____

Prof. Dr. Ricardo Pereira e Silva

_____

Prof. Dr. Daniel Schwabe

Ao meu marido Sandro e a nossa
semente que está para nascer.

AGRADECIMENTOS

Agradeço primeiramente ao meu marido Sandro, pelo seu amor, carinho, paciência e constante apoio nesta caminhada. Sem você tudo teria sido mais difícil...

Aos meus pais Sebastião e Wanda e irmãos Cristina e Carlos pelo estímulo e confiança.

À minha orientadora Profª. Patrícia Vilain, pela dedicação, oportunidades de aprendizado, apoio e também por ter acreditado em mim.

Aos amigos, Renata, Deco, Isa e Fê, pela amizade e momentos de descontração tão importantes.

À Universidade Federal de Santa Catarina.

À Equipe de Tecnologia da Informação do CIRAM/Epagri pela oportunidade.

À todos que direta ou indiretamente contribuíram para a realização desta pesquisa.

E finalmente aos Anjos, por me guiarem sempre pelo melhor caminho e me darem forças para alcançar meus objetivos.

"Qualquer caminho é apenas um caminho e não constitui insulto algum, para si mesmo ou para outros, abandoná-lo quando assim ordena o seu coração. Olhe cada caminho com cuidado e atenção. Tente-o tantas vezes quantas julgar necessárias... Então, faça a si mesmo e apenas a si mesmo uma pergunta: possui esse caminho um coração? Em caso afirmativo, o caminho é bom. Caso contrário, esse caminho não possui importância alguma."

Carlos Castañeda

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Cada vez mais os métodos ágeis têm despertado o interesse da comunidade de Engenharia de Software como uma alternativa para o desenvolvimento de sistemas de uma maneira mais rápida, eficiente e que atenda as reais necessidades dos clientes. Existe no mercado uma série de métodos disponíveis que utilizam a abordagem ágil e que, por seguirem os princípios ágeis, apresentam uma série de atividades semelhantes no seu processo de desenvolvimento.Este trabalho tem como objetivo principal propor um *framework* com as atividades dos métodos ágeis: *Extreme Programming* (XP), *Scrum*, *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD) e *Agile Modeling* (AM), de forma a permitir que os envolvidos no processo possam selecionar as atividades propostas de acordo com as suas reais necessidades. Para se alcançar este objetivo, foi realizada uma pesquisa sobre cada um destes métodos, bem como uma análise comparativa para identificar as atividades propostas por cada deles, de maneira a possibilitar a definição da estrutura do *framework*. Com o objetivo de verificar se o *framework* proposto apresenta coerência e diretrizes concretas para a sua utilização, ao final deste trabalho é apresentado um exemplo onde um sistema real é desenvolvido utilizando o mesmo.

Palavras – chave: Métodos Ágeis, *Framework*, *Extreme Programming*, *Scrum*, *Feature Driven Development*, *Adaptive Software Development*, *Agile Modeling*.

ABSTRACT

More and more often the Software Engineering community has paid attention to agile methods as a more efficient alternative for system development that supports customer real needs. There are several methods in the market based on the agile approach and, because these methods use the same agile principles, a lot of activities in their development processes resemble that approach. The main goal of this work is to propose a framework that includes activities of the following agile methods: Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Adaptive Software Development (ASD) e Agile Modeling (AM). The idea behind this framework is to allow stakeholders to select among the proposed activities according to the needs of their software development process. To accomplish this goal, a research was conducted to analyze the common activities of each of the selected agile methods in order to define the structure of the framework. Finally, as a way to verify that the proposed framework presents coherence and clear guidelines for its use, an example is presented where a real system is developed using such framework.

Keywords: Agile Methods, Framework, Extreme Programming, Scrum, Feature Driven Development, Adaptive Software Development, Agile Modeling.

1 INTRODUÇÃO

1.1 Contextualização

À medida que a sociedade torna-se cada vez mais dependente da indústria do *software*, ficam mais evidentes os problemas relacionados ao processo de desenvolvimento de sistemas: alto custo, alta complexidade, dificuldade de manutenção, e uma disparidade entre as necessidades dos usuários e o produto desenvolvido.

Acreditando que o processo utilizado é um dos motivos para a ocorrência destes problemas, um segmento crescente da Engenharia de Software vem defendendo a adoção de processos mais simplificados conhecidos como métodos[1] ágeis, diferentes dos processos tradicionais que são baseados na produção de uma grande quantidade de documentação e de modelos para guiar a programação e que demandam muito tempo para serem gerados (FOWLER, 2001).

Os métodos ágeis têm despertado um grande interesse entre a comunidade de desenvolvimento de *software*. Acredita-se que devido a esta demanda, uma considerável quantidade de métodos apresentando características ágeis têm surgido nos últimos anos, juntamente com livros técnicos abordando o assunto. Entretanto, por terem estes métodos, em sua maioria, uma publicação recente, é ainda incipiente a pesquisa e a comprovação acadêmica sobre eles. Sendo assim, surgiu a idéia da realização de um estudo mais profundo a respeito das práticas, processos e dos papéis da equipe de alguns métodos ágeis disponíveis no mercado. A partir deste estudo, percebeu-se que eles apresentam características bastante semelhantes possibilitando assim, a criação de um *framework*[2] contendo as atividades sugeridas por cada um dos métodos de forma a permitir a escolha de quais delas utilizar durante o processo de desenvolvimento de um sistema.

[1] De acordo com Rumbaugh (1995), "um método é um guia com o objetivo de auxiliar as pessoas a executarem alguma atividade". O termo "método" será utilizado ao longo deste trabalho sempre que o objetivo for o enfoque a um conjunto de procedimentos para o desenvolvimento de software.

[2] O conceito de *framework* adotado neste trabalho é o de um conjunto de atividades genéricas que pode ser adaptado a aplicações específicas, servindo como um molde para a construção de aplicações.

Nas seções seguintes serão apresentados com mais detalhes o objetivo geral e os objetivos específicos deste trabalho, quais as justificativas para o mesmo, alguns trabalhos considerados relevantes relacionados com o tema e a forma de estruturação do texto.

1.2 Objetivos do Trabalho

1.2.1 Objetivo Geral

O objetivo principal deste trabalho é propor um *framework* para os métodos ágeis: *Extreme Programming* (XP), *Scrum*, *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD) e *Agile Modeling* (AM), para ser utilizado durante o processo de desenvolvimento de sistemas, onde as atividades sugeridas por cada um dos métodos ágeis poderão ser selecionadas pela equipe de desenvolvimento de acordo a sua necessidade.

1.2.2 Objetivos Específicos

- Estudar as características gerais dos métodos ágeis;
- Estudar as práticas, os processos e os papéis da equipe dos métodos ágeis *Extreme Programming* (XP), *Scrum*, *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD) e *Agile Modeling* (AM);
- Realizar uma análise das atividades propostas por cada um dos métodos ágeis estudados em relação ao seu processo de desenvolvimento, identificando como são realizadas, em que fase do processo, e quais as práticas e papéis associados a elas.
- Propor um *framework* para os métodos ágeis XP, *Scrum*, FDD, ASD e AM, de acordo com a análise realizada;
- Apresentar um exemplo de utilização do *framework* proposto no desenvolvimento de um sistema real.

1.3 Justificativas do Trabalho

À medida que os métodos ágeis foram sendo estudados, notou-se que eles possuiam muitas semelhanças e também algumas diferenças em relação às atividades desenvolvidas durante o processo de desenvolvimento, à aplicação das práticas propostas durante a realização destas atividades e aos papéis sugeridos para fazerem parte da equipe. Por este motivo decidiu-se propor um *framework* que apresentasse as atividades sugeridas por cada um dos métodos estudados, de forma a serem utilizadas de acordo com as necessidades da equipe de desenvolvimento.

Além disso, para que um método de desenvolvimento de *software* seja utilizado por uma organização é de extrema importância que o mesmo seja amplamente conhecido pelos *stakeholders*[3] de forma a não comprometer a sua aplicação. Sendo assim, a equipe que optar pela utilização da abordagem ágil não necessitará conhecer todos os métodos ágeis existentes para, então, escolher qual deles se adapta melhor ao projeto que será desenvolvido, pois ela terá como opção um *framework* que reúne as características dos métodos ágeis XP, *Scrum*, FDD, ASD e AM.

1.4 Trabalhos Relacionados

Nesta seção serão apresentados trabalhos relacionados com o tema em questão. À medida que os mesmos foram sendo encontrados, constatou-se uma grande quantidade de trabalhos informais disponíveis principalmente na Internet. Já em relação a publicações existentes em livros e periódicos conceituados, não se pode afirmar o mesmo. Sendo assim, acredita-se ser relevante mencionar aqui apenas os trabalhos que apresentam segurança em relação a sua idoneidade.

Serão apresentados nesta seção os dois trabalhos encontrados sobre comparações entre os métodos ágeis que apresentaram maior credibilidade devido as suas referências, pela qualificação dos seus autores e pelo local onde foram publicados.

---

[3] Segundo Scott W. Ambler em http://www.agilemodeling.com/essays/activeStakeholderParticipation.htm, o termo "*stakeholders*" foi criado para designar todas as pessoas que, de alguma maneira, participam do processo de desenvolvimento direta ou indiretamente.

Já em relação a trabalhos relacionados a uma possível proposta de *framework* dos métodos ágeis, não se encontrou nada a respeito que tenha sido publicado em algum periódico de renome.

1.4.1 Novas Direções para os Métodos Ágeis: Uma Análise Comparativa (ABRAHAMSSON 2003).

Publicado pela IEEE *Computer Science*, este trabalho informa os resultados de um estudo comparativo entre os métodos ágeis ASD (*Adaptive Software Development)*, AM (*Agile Modeling*), *Crystal Family*, DSDM (*Dynamic Systems Development Method*), XP (*Extreme Programming*), FDD (*Feature Driven Development*), ISD (*Internet-Speed Development.*), PP (*Pragmatic Programming*) e *Scrum* em relação aos seguintes critérios:

- **Ciclo de vida de desenvolvimento de software:** critério utilizado para verificar se o método sugere um processo coerente para o desenvolvimento do sistema.
- **Administração de projeto:** critério utilizado para verificar se o método apóia a execução formal das atividades realizadas durante o todo o projeto.
- **Princípios abstratos *versus* direção concreta:** critério utilizado para verificar se as práticas e atividades provêem uma direção concreta em como uma tarefa específica deve ser executada.
- **Predefinição universal *versus* situação adequada:** critério adotado para averiguar até que ponto o método se adapta à situação real da organização que o utilizará.
- **Apoio empírico**: critério utilizado para verificar se os métodos foram fundamentados em evidências empíricas.

A seguir serão apresentados os resultados da comparação segundo Abrahamsson (2003).

Em relação ao ciclo de vida do desenvolvimento, a AM, o ISD, o *Scrum* e a PP não enfatizam o processo de desenvolvimento do *software*. Como a AM e o PP são complementares a outros métodos, a ausência de uma perspectiva de processo parece

aceitável. Nos outros métodos abordados (ASD, *Crystal Family*, DSDM, XP, FDD), o processo de desenvolvimento foi descrito e apresenta coerência.

Atualmente, a AM e a PP não focam a administração do projeto. Recentemente o XP foi complementado com algumas diretrizes referentes à administração do projeto, mas ainda não oferece uma visão inclusiva a respeito. Até a publicação do trabalho, a AM e a PP não abordavam o ponto de vista administrativo. O *Scrum* também não aborda esta atividade, mas a bibliografia recomenda que o XP seja utilizado como uma alternativa. Já o ASD, o FDD, o DSDM e o *Crystal Family* apresentam algumas atividades para a atividade de administração do projeto.

Em relação ao caso dos métodos nortearem de forma efetiva a execução de suas práticas e tarefas, cinco entre os nove métodos estudados não atendem esse critério, uma vez que não apresentam modos concretos em relação a como as suas práticas devem ser executadas. São eles: FDD, *Crystal Family*, ASD, DSDM e ISD. Já o XP, o *Scrum*, a PP e a AM enfocam esta questão. Já em relação ao poder de adaptação dos métodos, os métodos que mais apresentam esta característica são o *Crystal Family*, o FDD e o DSDM, por não imporem a utilização total de suas práticas. Por outro lado, os métodos ASD, AM, XP, ISD, PP e o *Scrum* sugerem que para que o projeto tenha sucesso, as práticas adotadas devem ser seguidas em sua totalidade.

A comparação em relação aos métodos serem baseados em experiência reais aponta que o ASD, o *Crystal Family*, o FDD, a AM e a PP não foram fundamentados em experimentos empíricos. Recentemente, o *Scrum* publicou um estudo[4] fundamentando a prática referente à adoção de reuniões diárias, portanto foi considerado que o método atende parcialmente o critério. O método ISD se baseia em nove estudos de caso, desta forma apresenta apoio empírico. O XP não é baseado em pesquisa sistemática, mas apresenta experiências reais que acabaram se tornando algumas das práticas sugeridas pelo método. Como o DSDM foi desenvolvido por um consórcio fechado, os documentos a respeito de experiências autênticas são restritos aos membros do mesmo. Sendo assim, foi entendido que o método não atende ao critério.

[4] Publicado em *"The Scrum software development process for small teams"*. (LINDA, 2000).

Além de apresentar uma discussão a respeito dos resultados, os autores acreditam que os mesmos auxiliarão os desenvolvedores na escolha do método que mais se adapte às suas necessidades e como conclusão destacam que a maioria deles não provê um apoio administrativo adequado durante o processo de desenvolvimento, não oferecem diretrizes concretas para a execução de suas práticas e não apresentam fundamentos baseados na experiência para justificarem o uso de suas práticas. Eles enfatizam que é mais importante a existência de métodos ágeis eficientes e de qualidade do que uma grande quantidade deles.

1.4.2 Selecionando um Processo Ágil: Comparando as Principais Alternativas (COHN, 2002).

O outro trabalho sobre comparações entre métodos ágeis que será abordado nesta seção foi escrito por Mike Cohn em outubro de 2002. Cohn é um dos membros fundadores da Aliança Ágil (maiores detalhes no capítulo 2) e elaborou um estudo cujo objetivo é verificar se os métodos ágeis FDD (*Feature Driven Development*), *Scrum*, XP (*Extreme Programming*), *Crystal Family* e DSDM (*Dynamic Systems Development Method*) atendem alguns critérios em relação ao Manifesto Ágil (também contemplados no capítulo 2).

O trabalho apresenta uma visão geral de cada um dos métodos e apresenta os seguintes resultados:

Critérios	FDD	*Scrum*	XP	*Crystal Family*	DSDM
Equipes auto-organizáveis e adaptáveis	Não	Sim	Sim	Sim	Parcialmente
Ausência de fases no processo de desenvolvimento	Não	Sim	Sim	Sim	Não
Planejamento mínimo	Não	Sim	Sim	Sim	Parcialmente
Escalabilidade	Sim	Sim	Sim	Sim	Sim
Refactoring	Não enfatizado	Sim	Sim	Sim	Sim
Iterativo e incremental	Sim	Sim	Sim	Sim	Sim
O progresso é medido através da entrega dos incrementos	Não	Sim	Sim	Sim	Sim
Projeto simples	Um pouco	Sim	Sim	Sim	Parcialmente
Envolvimento efetivo do cliente	Sim, mas não enfatizado	Sim	Sim	Sim	Sim
Relação adaptável com o cliente	Sim	Sim	Sim	Sim	Sim
Requisitos mutáveis	Não	Sim	Sim	Depende do tipo de	Sim

				projeto	
Inspeções freqüentes	Sim	Sim	Sim	Sim	Sim

Tabela 1. Comparação entre os métodos ágeis.
(Cohn, 2002)

Após os cinco métodos serem comparados, obteve-se como resultado que o XP, o *Scrum* e o *Crystal* eram os métodos que mais atendiam o Manifesto Ágil, enquanto o DSDM e o FDD atendiam parcialmente.

1.5 Estrutura do Trabalho

O presente trabalho está estruturado da seguinte forma. No capítulo 2 é apresentada uma visão geral da abordagem ágil, seu histórico, as diretrizes do Manifesto Ágil e os princípios ágeis, bem como as características dos métodos ágeis *Extreme Programming* (XP), *Scrum*, *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD) e *Agile Modeling* (AM).

No capítulo 3, é apresentada uma análise em relação às atividades propostas por cada um dos métodos ágeis, com o objetivo de identificar as fases do processo de desenvolvimento nas quais as mesmas são realizadas, quais as práticas e papéis incorporados a elas, bem como quais as atividades podem ser utilizadas sozinhas e quais dependem da realização de outras atividades para serem utilizadas sem comprometer o andamento do processo de desenvolvimento.

No capítulo 4 é apresentada a estrutura do *framework* para os métodos ágeis, com base na análise realizada no capítulo anterior, bem como as suas dependências e restrições.

No capítulo 5 é apresentado um exemplo aplicando o *framework* proposto, de forma a verificar se o mesmo apresenta diretrizes concretas e eficientes para ser utilizado.

Por fim, no capítulo 6 são apresentadas as considerações finais do presente trabalho, incluindo os objetivos alcançados, quais as contribuições acadêmicas do mesmo e algumas propostas de trabalhos futuros relacionados com o tema.

2 MÉTODOS ÁGEIS

O desafio mais comum dentro do desenvolvimento de *software*, é entregar um produto que atenda as reais necessidades do cliente, dentro do prazo e orçamento previstos. E como suporte a esta atividade, a Engenharia de Software oferece vários métodos que auxiliam os desenvolvedores durante o processo de desenvolvimento.

A "indústria do *software*" sempre contou com métodos cujo processo de desenvolvimento era baseado em um conjunto de atividades predefinidas, descritas como processos prescritivos (AMBLER 2004), onde muitas vezes, o trabalho começava com o levantamento completo de um conjunto de requisitos, seguido por um projeto de alto-nível, de uma implementação, de uma validação e, por fim de uma manutenção (SOMMERVILLE 2003). Entretanto, segundo Fowler (2001), estes métodos são considerados muito burocráticos e eficientes para projetos grandes e que não sofram muitas mudanças em seus requisitos.

A partir da década de 90, começaram a surgir novos métodos sugerindo uma abordagem de desenvolvimento ágil onde os processos adotados tentam se adaptar às mudanças, apoiando a equipe de desenvolvimento no seu trabalho (BECK, et. al., 2001). Estes novos métodos surgiram como uma reação aos métodos tradicionais[5] de desenvolvimento de *software*, ganhando com o passar dos anos um número cada vez maior de adeptos.

Com o intuito de definir um manifesto para encorajar melhores meios de desenvolver *software*, em fevereiro de 2001 um grupo inicial de 17 metodologistas, entre eles, Robert C. Martin, Jim Highsmith, Kent Beck, Mike Beedle, Alistair Cockburn, Martin Fowler, Steve Mellor, Ken Schwaber, Jeff Sutherland, e outros, formou a Aliança para o Desenvolvimento Ágil de *Software*, também conhecida como Aliança Ágil, que formulou um manifesto contendo um conjunto de princípios que definem critérios para os processos de desenvolvimento ágil de *software* (AMBLER 2004).

[5] Segundo Fowler (2003), os métodos que utilizam processos prescritivos são conhecidos como métodos tradicionais.

O Manifesto Ágil é baseado em quatro valores, definindo preferências e encorajando o enfoque em certas áreas. A seguir serão apresentados os valores do Manifesto Ágil de acordo com Beck et. al.(2001).

- Indivíduos e interações **valem mais** que processos e ferramentas.
- Um *software* funcionando **vale mais** que uma documentação extensa.
- A colaboração do cliente **vale mais** que a negociação de contrato.
- Responder as mudanças **vale mais** que seguir um plano.

Para auxiliar as pessoas a compreender o enfoque do desenvolvimento ágil, os membros da Aliança Ágil refinaram as filosofias contidas em seu manifesto em uma coleção de doze princípios (BECK, et. al. 2001), aos quais os métodos ágeis de desenvolvimento de *software* devem se adequar. Estes princípios são:

1. A prioridade é satisfazer ao cliente através de entregas de *software* de valor contínuas e freqüentes.
2. Receber bem as mudanças de requisitos, mesmo em uma fase avançada, dando aos clientes vantagens competitivas.
3. Entregar *softwares* em funcionamento com freqüência de algumas semanas ou meses, sempre na menor escala de tempo.
4. As equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto.
5. Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessários para a realização do trabalho.
6. A maneira mais eficiente da informação circular dentro da equipe é através de uma conversa face-a-face.
7. Ter o *software* funcionando é a melhor medida de progresso.
8. Processos ágeis promovem o desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante.

9. Atenção contínua a excelência técnica e a um bom projeto aumentam a agilidade.
10. Simplicidade é essencial.
11. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas.
12. Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz e então se ajustar e adaptar seu comportamento.

De acordo com Fowler (2001), os métodos ágeis podem ser considerados como um meio termo entre a ausência de processo e o processo exagerado. Como exemplo pode-se citar a geração de documentação. A documentação continua fundamental, mas a preocupação deve ser em não desperdiçar tempo com a criação de documentos que não serão úteis. O planejamento também deve ser utilizado, mas com a consciência de que os planos podem sofrer alterações (FOWLER, 2001).

Basicamente os métodos ágeis se diferem dos tradicionais em 2 aspectos:

a) são adaptativos ao invés de prescritivos;

b) são orientados às pessoas ao invés dos processos.

Vale salientar que os métodos ágeis não são contra os modelos de processo de desenvolvimento tradicionais. Eles propõem uma alternativa que busca a melhoria do processo, tornando-o mais ágil e menos burocrático.

Existem vários métodos classificados como ágeis a disposição das equipes de desenvolvimento. Tais métodos possuem em comum o fato de serem aplicados em projetos não muito complexos, utilizando ciclos iterativos curtos, planejamento guiado por funcionalidades, retroalimentação constante, tolerância a mudanças, proximidade da equipe, intimidade com o cliente, e um foco no ambiente geral de trabalho do time (HIGHSMITH 2001).

Entretanto, ao longo do estudo realizado neste trabalho, constatou-se que os métodos ágeis estudados, da mesma forma que apresentam características comuns, apresentam também algumas diferenças, principalmente no que diz respeito às práticas utilizadas durante o processo de desenvolvimento e os papéis da equipe. Sendo assim,

será a partir da análise destas semelhanças e diferenças que este trabalho apresentará uma proposta de unificação para os mesmos.

A escolha dos métodos ágeis para fazerem parte do *framework* não se deu de forma aleatória e sim de forma seqüencial à medida que cada um deles foi surgindo como oportunidade de estudo. Entretanto, o tempo disponível para o desenvolvimento do presente trabalho não possibilitou que fossem incorporados outros métodos ágeis além dos citados anteriormente, sendo assim optou-se por selecionar uma primeira fração dos métodos ágeis existentes para fazerem parte do *framework* que será apresentado neste trabalho. Convém salientar que existe a intenção de dar continuidade a esta pesquisa de forma que o *framework* contemple outros métodos considerados ágeis.

Nas próximas seções serão apresentadas as principais características dos métodos ágeis XP, *Scrum*, FDD, ASD e AM. Incluindo, uma visão geral, as suas práticas, seu processo de desenvolvimento, bem como os papéis de cada um dos envolvidos no projeto.

2.1 Extreme Programming (XP)

O Extreme Programming (Programação Extrema) é um método eficiente, flexível e de baixo risco para equipes pequenas e médias que desenvolvem *software* com requisitos dinâmicos ou em constante mudança (BECK, 2000). O criador do XP foi Kent Beck que aplicou pela primeira vez o método em 1996 em um projeto chamado *Chrysler Comprehensive Compensation* (C3) para a empresa *DaimlerChrysler*.

O XP segue um conjunto de valores, princípios e práticas básicas adotado durante o processo de desenvolvimento por uma equipe dividida em papéis específicos, visando alcançar eficiência e efetividade (BECK, 2000). Segundo Highsmith (2002a), a adoção do XP pelo mercado de desenvolvimento de *software*, se encontra em constante crescimento frente aos métodos tradicionais devido à redução de custos, rapidez no desenvolvimento e à adequação a mudança dos requisitos.

2.1.1 Os Valores do XP

Beck (2000) afirma que o sucesso de qualquer projeto depende da adoção de um estilo, que contempla um conjunto consistente de valores que servem para necessidades tanto humanas quanto comercias. A seguir serão apresentados os quatro valores adotados pela XP, de acordo com Beck (2000):

- **Comunicação**: É o primeiro valor do XP. O objetivo é uma comunicação rápida e ao mesmo tempo eficaz entre os *stakeholders*, por exemplo: dá-se preferência à sala de discussão (*chat*) a correio (*e-mail*), telefonemas a *chat*, conversas face-a-face a telefonemas, trabalhar na mesma sala a ter salas isoladas.
- **Simplicidade**: É o segundo valor do XP. O objetivo é simplificar continuamente o *software*. Simplicidade e comunicação estão diretamente relacionadas, pois quanto maior a comunicação, mais fácil a identificação do que realmente necessita ser feito e vice-versa, e como conseqüência tem-se um *software* mais simples.
- ***Feedback***: É o terceiro valor do XP. Todo problema é evidenciado o mais cedo possível, tanto pelo cliente como pela equipe de desenvolvimento, para que possa ser corrigido. O *feedback* concreto trabalha junto com a comunicação e a simplicidade.
- **Coragem**: Quando combinada com os três primeiros valores é extremamente valiosa. É preciso coragem para pedir ajuda quando necessário, reconstruir e simplificar o código que está funcionando, bem como para expor ao cliente que o prazo estimado para implementar determinada funcionalidade não é suficiente.

2.1.2 As Práticas do XP

A maioria das práticas sugeridas pelo XP causa polêmica à primeira vista e muitas não fazem sentido se aplicadas isoladamente. Segundo Beck (2000), nenhuma prática consegue se manter por si só. É a sinergia de seu conjunto que sustenta o processo de desenvolvimento do XP. A seguir serão apresentadas estas práticas de acordo com Beck (2000), Jeffries (2001), Astels (2002) e Fowler (2001):

- **Jogo do Planejamento -** Os participantes do jogo do planejamento são o cliente e os programadores. O cliente decide sobre: escopo, prioridade, composição e datas

das entregas. As responsabilidades dos programadores incluem: estimativas de tempo de desenvolvimento para cada funcionalidade, avaliação dos riscos técnicos e decisão sobre o processo de trabalho. O jogo do planejamento do XP divide-se em duas etapas chaves: Planejamento da Entrega e Planejamento da Iteração (JEFFRIES, 2001). O Planejamento da Entrega é uma atividade onde o cliente apresenta as funcionalidades desejadas aos programadores, e os programadores estimam a sua dificuldade. Com as estimativas das dificuldades disponíveis e com o conhecimento da importância das funcionalidades, o cliente apresenta um plano para o projeto contendo estas informações. Inicialmente, nem as prioridades nem as estimativas são verdadeiramente sólidas, até que a equipe comece realmente a executar o trabalho. O Planejamento da Iteração é onde a equipe de desenvolvimento (programadores) recebe orientação através das *user stories* (histórias de usuário) que apresentam informações a respeito das funcionalidades do sistema (BECK, 2000). A equipe de desenvolvimento constrói um *software* em iterações que duram de 1 a 4 semanas, entregando um *software* executável e útil no fim de cada iteração. Durante o Planejamento da Iteração, o cliente apresenta as *user stories* desejadas para a próxima iteração. Os programadores dividem as *user stories* em tarefas e estimam o tempo de desenvolvimento (em um nível mais detalhado que no Planejamento da Entrega).

- **Entregas Freqüentes -** Ao término de cada iteração é feita uma entrega, que deve ser tão pequena quanto possível, contendo as *user stories* mais importantes (BECK, 2000). A equipe libera um *software* rodando, com as *user stories* escolhidas pelo cliente para aquela iteração. O cliente poderá utilizar este *software* para avaliação ou para liberar aos usuários finais (JEFFRIES, 2001). O principal objetivo desta prática é que o progresso do *software* pode ser monitorado tanto pelo cliente quanto pelos programadores.

- **Metáfora -** O projeto de *software* no XP é guiado por uma metáfora simples (BECK, 2000). A metáfora ajuda a manter toda a equipe em sintonia com o projeto. Segundo Astels (2002), cada projeto do XP deve fazer uso de no mínimo uma metáfora para orientar a equipe e fornecer um contexto compartilhado. Tendo uma metáfora compartilhada, todos compartilham de uma visão geral do sistema e do problema que deve ser solucionado.

- **Projeto Simples -** O XP mantém o projeto o mais simples possível. Se o projeto for simples, ele permanece ágil e flexível. Para manter um projeto simples, é necessária uma revisão contínua (ASTELS, 2002). A equipe XP não desenvolve um grande projeto inicial, mas projeta continuamente durante todo o tempo (FOWLER, 2001).

- **Testes -** Os programadores escrevem testes de unidade e os clientes testes de aceitação freqüentemente com o objetivo de tornar o software cada vez mais confiável. No XP, os testes de unidade são escritos anteriormente ao processo de codificação e só após são executados. Os testes de unidade têm como função, obrigar os programadores a analisar melhor o que efetivamente será codificado. Para cada *user story* é escrito no mínimo um teste de aceitação. Os mesmos são escritos com base na seguinte questão: "O que precisaria ser conferido para se ter certeza de que uma determinada *user story* está OK?". Os clientes escrevem textualmente as respostas para este questionamento e executam os testes no incremento entregue. Os resultados destes testes devem ser publicados para toda a equipe.

- ***Refactoring* -** O XP usa um processo contínuo de melhoria de projeto chamado *refactoring*. *Refactoring* é a técnica empregada na reestruturação do código, cujo principal objetivo é fazer com que o mesmo fique mais reutilizável e compreensível, sem que haja mudança no seu comportamento (FOWLER, 2001). Deve-se fazer *refactoring* onde for possível e sempre que possível.

- **Programação em pares:** Segundo Jeffries (2001), "Todo *software* produzido em XP é construído por dois programadores, sentados lado a lado, na mesma máquina. Esta prática assegura que todo código produzido é revisado por, pelo menos, outro programador". O resultado é um aumento na velocidade do desenvolvimento, com testes mais eficazes e um código otimizado. É importante observar que existem dois papéis em cada par. O parceiro que está com o teclado e o *mouse* é responsável por pensar na melhor forma de implementar o método corrente. Enquanto o outro parceiro analisa estrategicamente se a abordagem utilizada irá funcionar, se existe algum teste que ainda não foi trabalhado e se existe alguma forma para simplificar o código (BECK, 2000).

- **Propriedade coletiva -** A qualquer momento, qualquer um que perceba uma oportunidade de agregar valor a alguma parte do código deve fazê-lo. Segundo Beck

(2000), em um projeto XP todos são responsáveis por todo o sistema. Isto não quer dizer que todos devam conhecer o código igualmente bem, mas todos devem saber o mínimo de cada parte. Um bom sistema de controle de versões é importante para garantir a aplicação desta prática.

- **Integração contínua -** As equipes XP mantêm o sistema integrado todo o tempo. O código é integrado e testado depois de algumas horas, no máximo depois de um dia de desenvolvimento. A forma mais simples para fazer isto é ter um servidor dedicado para a integração (BECK, 2000). Também nesta prática, um sistema de controle de versões é importante. Cada par de programadores é responsável por integrar seu próprio código. Isto pode ser feito quando os testes de unidade tenham sido executados 100% em cada funcionalidade planejada. A integração contínua evita ou descobre problemas de compatibilidade cedo.

- **Semana de 40-horas -** Não é recomendável fazer horas extras por períodos maiores que uma semana. Como também não é produtivo passar a noite acordado e trabalhar no dia seguinte. Um programador quando está cansado raciocina mais lentamente e fica mais distraído, a conseqüência disto pode ser o desenvolvimento de código com erros, erros que vão dar trabalho para serem corrigidos e exigir mais horas extras. De acordo com Beck (2000), a sobrecarga de trabalho é sintoma de sérios problemas no projeto.

- **Cliente presente -** Um "cliente real" deve estar sempre disponível, não somente para auxiliar, mas para fazer parte da equipe. Beck (2000) define por "cliente real" aquele cliente que realmente irá utilizar o sistema quando ele estiver em produção. Assim, o cliente poderá fornecer detalhes do sistema quando surgirem dúvidas. Se o cliente estiver sempre por perto, significa que toda e qualquer dúvida pode ser prontamente esclarecida permitindo um controle maior do que está realmente sendo implementado. Entretanto, algumas vezes, não é possível que o cliente esteja no mesmo local de trabalho que os programadores. Ainda assim, é possível trabalhar com XP, desde que se tenha um canal aberto de comunicação. Nestes casos, é indicado o uso de ambientes colaborativos através da Internet.

- **Padrões de codificação -** As equipes XP seguem um padrão de codificação comum a todos os membros. Desta forma, todo código no sistema pode ser visto como se

fosse escrito por um único programador. As particularidades do padrão não são importantes: o que é importante é que todo o código deve parecer familiar, em defesa da propriedade coletiva (JEFFRIES, 2001). A padronização do código mantém o mesmo consistente e fácil para toda a equipe entender e/ou fazer o *refactoring*. Não importa qual é o padrão, o importante é que exista um. O padrão deve ser adotado voluntariamente por toda a equipe (BECK, 2000).

2.1.3 As Fases do Processo do XP

Segundo Beck (2000), o projeto ideal em XP é aquele que inicia por uma curta fase de desenvolvimento, seguida de anos de produção e refinamentos simultâneos e finalmente encerra quando o projeto não faz mais sentido.

Durante todo o processo de desenvolvimento, poderão ser feitas reuniões diárias para que todos conheçam aquilo em que todos estão trabalhando. O método sugere que estas reuniões sejam realizadas em pé, garantindo mais agilidade BECK (2000).

Como pode ser observado na Figura 1, o ciclo de vida do XP é bastante curto e, à primeira vista, difere dos padrões dos modelos de processo convencionais.

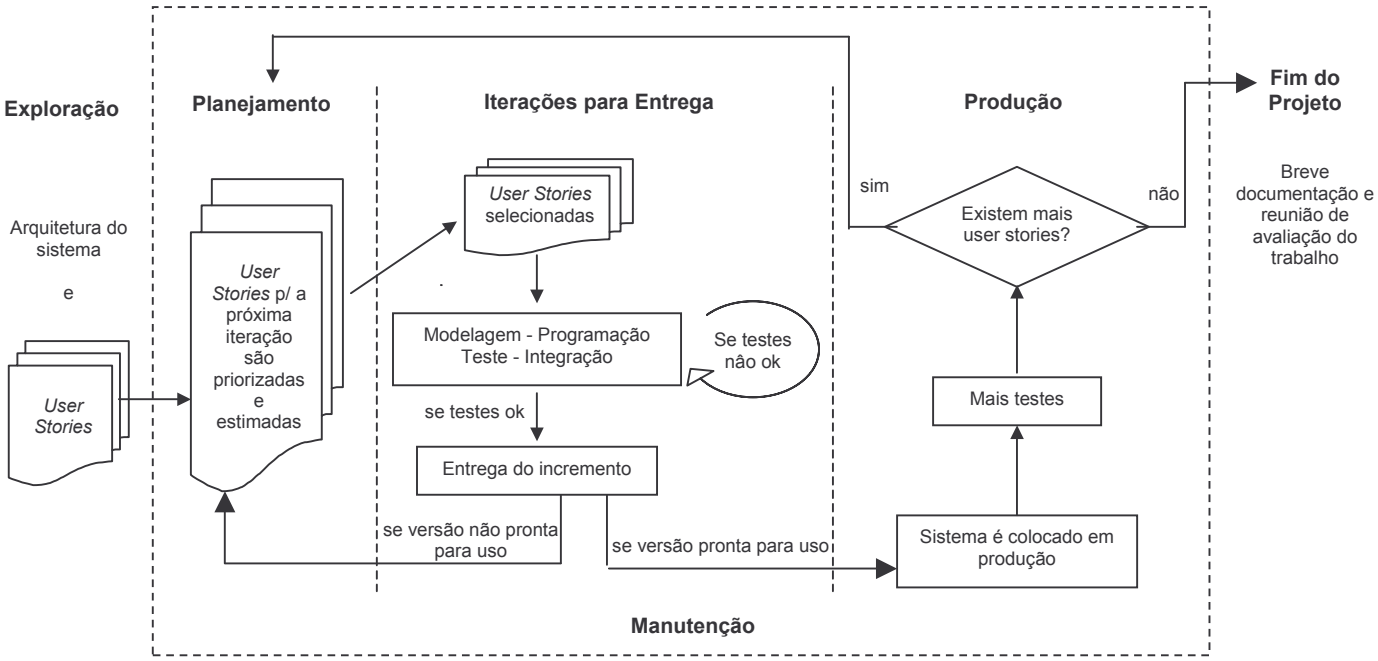


Figura 1. As Fases do Processo XP

2.1.3.1 Exploração

O objetivo da fase de Exploração é entender o real escopo do sistema. Este entendimento deve ser o suficiente para que ele possa ser estimado (BECK 2000).

A fase de Exploração começa com o cliente escrevendo as *user stories* (Figura 2). Wake (2002) considera que *user stories* menores tendem a ter um risco menor. Então, se uma *user story* for muito grande, o cliente deve dividir a *user story* em *user stories* menores. O segredo para o cliente escrever *user stories* úteis, segundo Beck (2000), é dar um *feedback* rápido logo nas primeiras *user stories*, para que ele aprenda a especificar o que os programadores precisam e não especificar o que eles não precisam.

Data: 22/12/2000 **Nova:** X **Dificuldade:** Média

Número da História: 005 - Criar Recibo **Prioridade do Usuário**: Alta **Técnico:** João Camargo

Referência Anterior: 001 – Efetuar Venda **Risco:** **Estimativa do Técnico:** 20 h

Descrição: Manter um recibo aberto com uma descrição breve de cada item escaneado e seu preço. Quando a venda for concluída, incluir o total na parte inferior do recibo.

Notas:

Acompanhamento da tarefa			
Data	Estado	A realizar	Comentário

Figura 2. Exemplo de *User Story*.
(adaptado de BECK, 2000).

Em paralelo às *user stories*, os programadores estão experimentando ativamente as diferentes tecnologias e as possíveis configurações, explorando as possibilidades para a arquitetura do sistema. Leva-se de 1 a 2 semanas testando a arquitetura, contudo esta deve ser testada de várias maneiras, ou seja, diferentes pares podem testar diferentes tecnologias e comparar, ou dois pares podem testar a mesma tecnologia e comparar as diferenças emergentes. Se este período não for suficiente para estes testes, então a mesma deve ser classificada como um risco para o projeto. Portanto, durante a fase de exploração pode-se convidar um especialista na tecnologia

escolhida, o qual não irá perder tempo com pequenos problemas, pois sabe por experiência como resolvê-los.

Beck (2000) ressalta que esta pequena exploração na arquitetura auxilia os programadores a terem uma idéia da implementação quando o usuário apresentar suas *user stories*, pois como será visto na próxima fase, os programadores precisam estimar cada tarefa de programação. Quando uma tarefa é feita, eles precisam repassar para o calendário o tempo gasto para executar aquela tarefa. A prática da estimativa cria confiança na equipe para quando eles realmente tiverem que estimar o tempo de implementação.

2.1.3.2 Planejamento

O objetivo da fase de Planejamento é definir a menor data e o maior conjunto de *user stories* que serão realizadas na primeira entrega. Esta definição é feita de acordo com estimativas entre cliente e programadores (BECK, 2000).

Assim que as *user stories* são coletadas, o Jogo de Planejamento é conduzido. O cliente decide quais *user stories* são vitais e que devem fazer parte da primeira entrega. Desta forma, pode-se elaborar uma lista priorizada das *user stories*. Se houver uma boa preparação durante a fase de Exploração, são necessários apenas alguns dias para a fase de Planejamento.

Wake (2002) apresenta alguns passos para auxiliar a fase de Planejamento durante o Jogo de Planejamento:

- o cliente classifica as *user stories* por valor: alto, médio ou baixo;
- os programadores qualificam as *user stories* por risco (opcional): alto, médio ou baixo;
- os programadores estimam o tempo para o desenvolvimento das *user stories*. O tempo é empiricamente determinado, ou seja, baseado na experiência dos programadores;
- clientes escolhem as *user stories* para a próxima entrega.

2.1.3.3 Iterações para Entrega

Segundo Beck (2000), os compromissos são divididos para serem executados em iterações que duram de 1 a 4 semanas. Para cada *user story* executada que faz parte da iteração é escrito um ou mais de teste(s) de aceitação pelo cliente. E antes da implementação os programadores escrevem os testes de unidade. Segundo Ambler (2004), é nesta fase que ocorre o maior trabalho de desenvolvimento, incluindo modelagem, programação, escrita e execução dos testes de unidade e aceitação e integração.

A primeira iteração mostra como a arquitetura do sistema irá se comportar. Então, as *user stories* escolhidas devem ser as que influenciarão diretamente na mesma. A pergunta chave para ser trabalhada nesta fase é: Qual a coisa de maior valor para a equipe para ser trabalhada nesta iteração? (BECK, 2000).

Uma importante característica do XP é que ele recomenda que sejam implementadas somente as funcionalidades que estejam marcadas para a iteração corrente (WELLS, 2001). Desta forma, o prazo final de cada iteração será seriamente respeitado, permitindo analisar qual a real velocidade do projeto durante a mesma. Então, o conjunto de *user stories* que farão parte da nova iteração estará melhor estimado. No final de cada iteração o cliente terá completado a execução de todos os testes de aceitação e os programadores executam os testes de unidade escritos antes da implementação. No final de cada iteração, o cliente receberá uma nova versão do sistema e o mesmo estará pronto para entrar em produção.

2.1.3.4 Produção

Esta fase começa no final da primeira iteração e segundo Ambler (2004), entrar em produção significa lançar o sistema no ambiente real de trabalho do cliente.

Devem-se implementar novos testes para provar se o sistema está estável o suficiente para entrar em produção. Pode ser necessário apenas ajustar o desempenho. E este é o melhor momento para realizar estes ajustes, pois se tem mais conhecimento do projeto, assim como existe a possibilidade de realizar estimativas reais de sobrecarga de produção do sistema.

Segundo Beck (2000), esta fase não significa que o sistema está concluído, o *software* continuará evoluindo, pois como o sistema está em produção, o cliente ainda pode descobrir novas necessidades ou mesmo alterá-las, neste caso o processo reinicia novamente na fase de Planejamento.

2.1.3.5 Manutenção

De acordo com Ambler (2004), a fase de Manutenção é o estado normal de um projeto XP, e compreende as fases de Planejamento, Iterações para a Entrega (a partir da segunda iteração) e Produção até a entrega final do sistema.

Como conseqüência, esta fase inclui atividades, como a operação e o suporte ao sistema através de um *help desk* por exemplo BECK (2000).

2.1.3.6 Fim do Projeto

Beck (2000) considera que "Morrer bem é tão importante quanto viver bem. Isto é uma verdade para o XP como para as pessoas". Quando não mais existir novas estórias, é o momento de finalizar o projeto. É o momento de escrever algumas páginas (de 5 a 10 páginas) sobre as funcionalidades do sistema, onde o resultado será um documento para auxiliar os desenvolvedores na realização de alguma alteração futura no sistema, se for o caso. Uma boa razão para finalizar o projeto é o cliente estar satisfeito com o sistema e não ter mais nada que consiga prever para o futuro.

Toda a equipe que trabalhou no sistema deve ser reunida para uma reavaliação. Aproveitando a oportunidade para analisar o que pode ter causado problemas no sistema e o que fez o projeto avançar.

2.1.4 A Equipe do XP

O XP sugere a adoção de diferentes papéis para a execução do projeto. A seguir, estes papéis são apresentados de acordo com Beck (2000).

O **Programador** é o coração do XP e é responsável por: Estimar prazos das *user stories*; Escrever e executar os testes de unidade; Implementar o sistema; Trabalhar em par; Fazer *refactoring* sempre que necessário; Solicitar ao cliente que esclareça ou divida uma estória quando necessário.

O **Cliente** é outro papel essencial do XP. Enquanto o programador sabe como programar, o cliente sabe o que deve ser programado. Suas responsabilidades são: Definir as funcionalidades do *software*; Escrever as *user stories*; Definir as prioridades para as *user stories*; Escrever e executar os testes de aceitação e esclarecer dúvidas sempre que solicitado.

O **Testador** no XP tem o papel realmente focado no cliente. Tem por responsabilidade: Auxiliar os clientes a escrever os testes aceitação; Executar os testes que forem solicitados e publicar os resultados para a equipe.

O **Rastreador** é a consciência da equipe, sua responsabilidade é: Coletar sinais vitais do projeto (métricas) 1 ou 2 vezes por semana; Manter todos informados do que está acontecendo e tomar atitudes sempre que as coisas parecerem ir mal.

O **Treinador** é responsável pelo processo de desenvolvimento como um todo. Notifica as pessoas quando elas estão se desviando do processo e conduz a equipe novamente para o mesmo. O treinador tem a função de: Garantir que o projeto permaneça extremo; Ajudar com o que for necessário; Manter a visão do projeto; Formular e comunicar uma tarefa na qual um programador deseja trabalhar.

O **Consultor** é um membro externo que possui o conhecimento técnico específico necessário e é responsável por auxiliar a equipe na resolução de problemas específicos.

O **Gerente** é a pessoa que precisa transmitir coragem, confiança e saber cobrar o que é de responsabilidade de cada um. O gerente tem várias responsabilidades: Gerenciar a equipe e seus problemas; Agendar as reuniões de planejamento; Garantir que as reuniões fluam como planejado; Escrever o que foi definido nas reuniões; Manter o rastreador informado dos acontecimentos das reuniões e buscar recursos.

Segundo Beck (2000), existem alguns inconvenientes que podem fazer o método fracassar, entre eles: grandes equipes, sistemas muito complexos, escolha de tecnologias que não suportam adequadamente modificações, rotinas e ambientes de trabalho impróprios e o mais importante, a cultura empresarial.

Beck (2000) sugere que o método seja adotado gradualmente para não causar grande impacto dentro da organização e que a equipe de desenvolvimento seja composta

por, no máximo, 10 pessoas. O ambiente físico também é importante, a equipe deve trabalhar em um mesmo lugar, possibilitando uma melhor comunicação.

Algumas empresas que utilizam o método XP com sucesso são: Ford, Symantec, BMW, Borland, entre outras, (ASTELS, 2002).

2.2 *SCRUM*

O *Scrum*[6] é um método ágil que possui como objetivo principal a entrega de um *software* de qualidade dentro de iterações, formadas por pequenos intervalos de tempo definidos chamados *Sprints*[7] que possuem aproximadamente um mês de duração (BEEDLE, 1998).

O método é baseado em princípios semelhantes aos do XP: requisitos pouco estáveis ou desconhecidos, e iterações curtas para promover visibilidade ao desenvolvimento, resultando em flexibilidade, receptividade e confiabilidade (SCHWABER e BEEDLE 2002).

2.2.1 As Práticas do *Scrum*

De acordo com Schwaber e Beedle (2002), o *Scrum* não requer ou fornece qualquer método específico para desenvolvimento de *software*, apenas estabelece um conjunto de regras e práticas gerenciais que devem ser adotadas para o sucesso de um projeto que utilize o método A seguir, serão apresentadas as práticas do *Scrum* de acordo com Schwaber e Beedle (2002).

- ***Product Backlog*:** define tudo o que é necessário no produto final baseado no conhecimento atual. É o ponto de partida do *Scrum* onde são definidas as funcionalidades, as prioridades, a tecnologia e as estratégias. A definição destes itens pode ser feita por qualquer pessoa ou setor envolvidos no projeto. Para algum item ser adicionado ao *Product Backlog* é necessário que a equipe esteja de acordo. Os itens do *Product Backlog* são documentados (Figura 3) com as seguintes

[6] A origem do termo *Scrum* é uma metáfora a uma tática utilizada do jogo de *Rugby* para recuperar a bola perdida.
[7] O termo *sprint* em inglês quer dizer corrida de curta distância.

informações: uma descrição sucinta, uma estimativa, que deverá sempre ser estipulada em horas, um responsável e uma prioridade (muito alta, alta e média).

Para facilitar a visualização é sugerido que os itens sejam separados por prioridade.

Prioridade	Item	Descrição	Tempo Estimado	Responsável
Muito Alta				
	1	Conexão com o Banco de Dados	40	Ana
	2	...	...	...
	3	Cadastro de Usuários	...	João
Alta				
	4			
	5	..		
	6			
Média				
	7	Impressão de Relatórios		
	8	...	...	...
	9			

Figura 3. Exemplo de Itens do *Product Backlog*

- ***Sprint*:** Considerada a parte mais importante do *Scrum*. É onde são executados os itens definidos no *Product Backlog*. Cada *Sprint* deve durar no máximo 30 dias. Não existem processos pré-definidos dentro de uma *Sprint,* mas sim reuniões periódicas (Reuniões Diárias do *Scrum*) que coordenam a realização das atividades existentes. É criado um *Sprint Backlog* com os itens selecionados para serem trabalhados na *Sprint* corrente. Ao final de cada *Sprint*, segundo Beedle (1998), é criada uma pequena versão do *software*, com o objetivo de mostrar ao cliente o que está sendo desenvolvido. Na *Sprint* acontece a integração da parte *software* que foi desenvolvida com as outras partes já implementadas e são feitos testes, garantindo um progresso real das atividades. Uma nova *Sprint* começa com um novo conjunto de itens no *Sprint Backlog,* sendo assim, dentro do desenvolvimento *Scrum*, podem acontecer N *Sprints*. Schwaber (1995) ressalta que o desenvolvimento dos itens dentro de uma *Sprint* não segue nenhum processo pré-definido, enquanto Abrahamsson (2002) sugere o uso das fases tradicionais de desenvolvimento de *software*: análise, projeto, implementação, testes e entrega. Outra sugestão seria a combinação de outros métodos ágeis que possuem processos definidos nesta fase.

- ***Sprint Backlog***: É o ponto de partida para cada *Sprint*. Consiste em um conjunto de funcionalidades selecionadas do *Product Backlog* para serem implementadas durante a *Sprint*. Os itens são selecionados pelo *Scrum Team* em conjunto com o *Scrum Master* e o Dono do Produto na Reunião de Planejamento da *Sprint*, de acordo com as prioridades dos itens. Quando todos os itens do *Sprint Backlog* estiverem prontos, um novo incremento do sistema é entregue.

- **Reunião de Planejamento da *Sprint:*** Cada *Sprint* inicia com uma reunião chamada de Reunião de Planejamento da *Sprint,* que tem por objetivo analisar os itens do *Product Backlog* a fim de priorizá-los para o desenvolvimento e, assim, definir o *Sprint Backlog*.

- **Reuniões Diárias do *Scrum:*** De acordo com Linda (2000), as Reuniões Diárias do *Scrum* são realizadas diariamente ou em dias alternados e possuem duração de 15 a 30 minutos, no máximo. Este tempo é suficiente para identificar os problemas, mas não para definir soluções. Discussões para resolução de obstáculos são feitas mais tarde, onde somente os envolvidos no problema participam. Segundo Schwaber e Beedle (2002), durante as Reuniões Diárias do *Scrum* são levantadas as três seguintes questões para cada membro da equipe:

    - O que foi finalizado desde a última reunião? (São registradas quais tarefas foram completadas e quais ainda estão pendentes).

    - Quais as dificuldades encontradas durante o trabalho? (São registradas todas as dificuldades encontradas para mais tarde encontrar uma maneira de resolvê-las).

    - Quais atividades pretende-se realizar até a próxima reunião? (São escolhidas as tarefas mais importantes. Devido ao curto espaço de tempo entre as reuniões, as tarefas são geralmente pequenas).

A reunião também possibilita que todas as pessoas fiquem informadas sobre o progresso e as dificuldades encontradas.

- **Revisão *da Sprint*:** No último dia de cada *Sprint*, o *Scrum Team* e o *Scrum Master* apresentam o incremento para o cliente, gerente e Dono do Produto numa reunião informal. Os participantes avaliam o incremento do produto e decidem sobre as

atividades seguintes. Na reunião poderão ser adicionados novos itens ao *Product Backlog*.

2.2.2 As Fases do Processo do *Scrum*

Como em todos processos de desenvolvimento de *software*, no *Scrum* há uma etapa onde são definidos os requisitos, uma onde acontece o desenvolvimento dos mesmos e por fim acontece a entrega final do sistema. A Figura 4 ilustra as fases do processo *Scrum*.

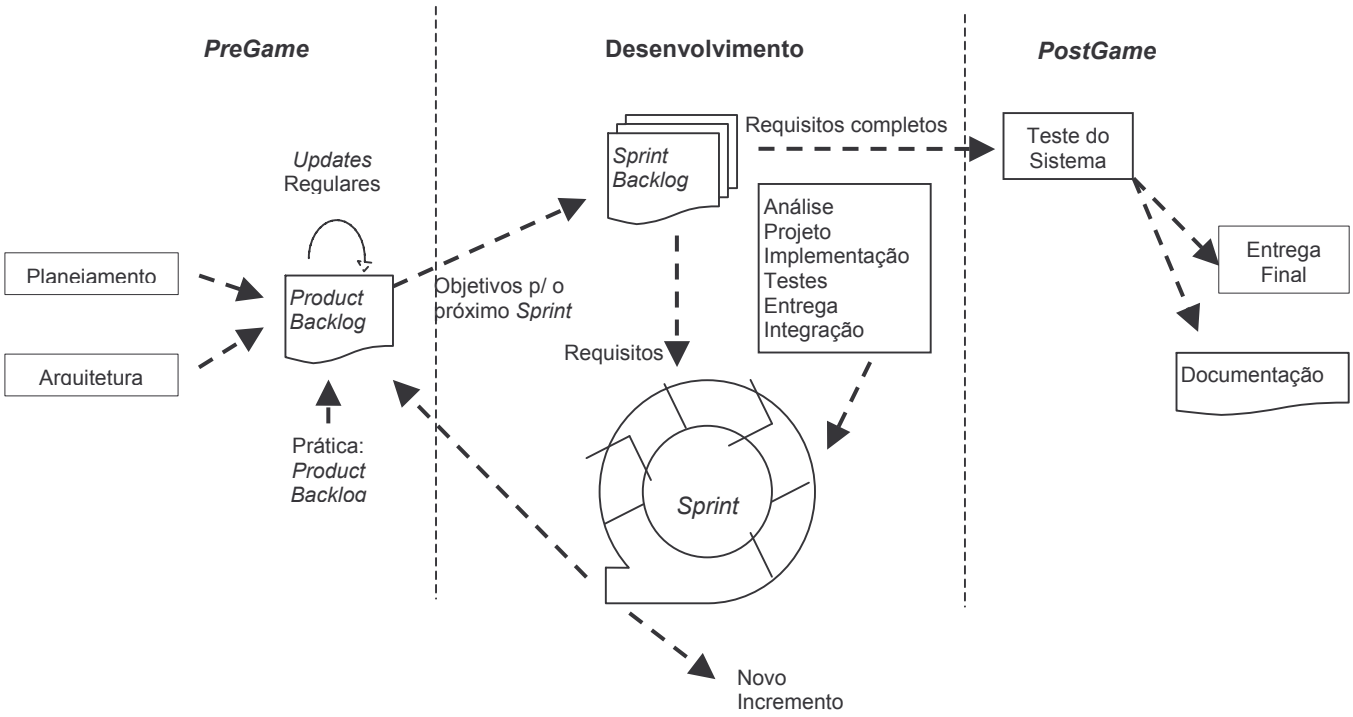


Figura 4. Fases do Processo *Scrum*

A seguir, as fases do *Scrum* são apresentadas de acordo com Schwaber e Beedle (2002).

2.2.2.1 *PreGame*

A fase de *PreGame* possui duas sub-fases: Planejamento e Arquitetura.

A sub-fase de Planejamento inclui a definição do sistema que está sendo desenvolvido. O *Product Backlog* é criado contendo todos os requisitos que são conhecidos até o momento. Os requisitos são priorizados e logo após é calculado o

esforço necessário para a sua implementação. O *Product Backlog* é constantemente atualizado com novos itens ou com itens mais detalhados, bem como com estimativas mais precisas e novas ordens de prioridade. O Planejamento inclui também a definição da equipe, ferramentas e outros recursos, avaliação dos riscos e treinamentos necessários. A cada iteração, o *Product Backlog* atualizado é revisado pela Equipe *Scrum*.

Na sub-fase de Arquitetura, é feito um projeto do sistema baseado nos itens atuais do *Product Backlog*. No caso de alguma alteração nos itens atuais, são identificadas as mudanças necessárias para implementar os itens do *Product Backlog* juntamente com os problemas que poderão surgir com estas alterações. É realizada uma reunião para revisar o projeto e reavaliar a proposta de implementação, logo após, são tomadas decisões com base nesta revisão.

2.2.2.2 Desenvolvimento

Também conhecida por G*amePhase,* nesta fase o sistema é desenvolvido através de várias *Sprints*, onde as funcionalidades são desenvolvidas ou modificadas em ciclos iterativos com a finalidade de produzir novos incrementos. Nesta etapa são *utilizadas as práticas da Sprint, Sprint Backlog,* Reunião de Planejamento da *Sprint,* Reuniões Diárias do *Scrum* e Revisão da *Sprint.*

2.2.2.3 *PostGame*

Na fase de *PostGame,* acontece a entrega final do *software*. Nela, os requisitos estão completos. Neste caso, não podem ser encontrados, tampouco adicionados novos itens. É hora de realizar mais testes de sistema e gerar uma breve documentação.

2.2.3 A Equipe do *Scrum*

De acordo com Schwaber e Beedle (2002), existem quatro papéis identificados no *Scrum*. Estes possuem tarefas e propósitos distintos durante o processo e suas práticas. A seguir, serão apresentadas as responsabilidades associadas a estes papéis.

O ***Scrum Master*** assegura que o projeto está sendo desenvolvido de acordo com as práticas e regras do *Scrum.* Tem como responsabilidades: Organizar as reuniões diárias; Representar o gerente do projeto e o técnico; Gerenciar todo processo do *Scrum*

na empresa; Assegurar que qualquer impedimento será resolvido e interagir com todos os envolvidos no projeto.

O **Dono do Produto** é a pessoa responsável pelo *Product Backlog*. Para desempenhar este papel é necessário que o Dono do Produto execute as seguintes funções: Tomar as decisões finais relacionadas ao *Product Backlog*; Liberar e escolher os itens que serão trabalhados no *Sprint Backlog* e calcular o esforço para o desenvolvimento de cada item.

O ***Scrum Team***, em conjunto com o *Scrum Master*, executa as ações necessárias para atingir os objetivos de cada *Sprint*. O *Scrum Team* deve ter no máximo sete pessoas e possui as seguintes responsabilidades: Determinar a criação do *Sprint Backlog*; Revisar os itens do *Product Backlog* e os impedimentos que por ventura venham a surgir; Trabalhar nos itens do *Sprint Backlog* e sugerir o que necessita ser acrescentado ou removido no projeto.

O **Cliente** participa ativamente na fase da elaboração dos itens que irão compor o *Product Backlog*.

Segundo Schwaber e Beedle (2002), não existem restrições quanto ao tamanho e complexidade de projetos para a utilização do *Scrum*. Entretanto, em projetos maiores podem ocorrer problemas em relação ao gerenciamento das mudanças dos requisitos, em relação as equipes de desenvolvimento e trocas de pessoal. Caso exista a necessidade de equipes com mais de 7 pessoas (quantidade máxima recomendada), é sugerido que as equipes sejam divididas em equipes menores.

Caso haja equipes múltiplas, o projeto não deve iniciar com todas as equipes ao mesmo tempo, e sim, com uma única equipe. O restante das equipes devem ser incorporadas ao projeto após a primeira *Sprint.*

Por se tratar de um método que não necessita de um alto investimento financeiro, o *Scrum* pode ser uma boa alternativa para as organizações que desejam acelerar o seu processo de desenvolvimento.

2.3 Feature Driven Developent (FDD)

O *Feature Driven Development* (Desenvolvimento Dirigido a Características) foi desenvolvido por Peter Coad e Jeff De Luca e foi utilizado pela primeira vez no desenvolvimento de um grande e complexo projeto de um sistema bancário em 1998. Como os outros métodos ágeis, o FDD é focado em pequenas iterações que geralmente duram 2 semanas, onde ao final ocorre a entrega de uma parte do sistema funcionando (HIGHSMITH 2002a). Segundo Palmer (2002), uma característica é uma funcionalidade definida pelo cliente.

A seguir serão apresentadas as práticas, as fases do processo de desenvolvimento do método, os papéis que compõe a sua equipe e por fim algumas considerações a respeito da sua aplicabilidade.

2.3.1 As Práticas do FDD

Segundo Abrahamsson (2002), o FDD consiste em um conjunto de práticas que devem ser usadas para garantir o sucesso da utilização do método em um projeto de *software*. A seguir serão apresentadas as práticas segundo Palmer (2003):

- **Modelagem dos Objetos de Domínio:** Consiste na construção de diagramas de classes UML[8] (*Unified Modeling Language*) que descrevem os objetos relevantes dentro do domínio do problema, bem como os relacionamentos entre eles. Para complementar os diagramas de classe UML, são desenvolvidos diagramas de seqüência UML que descrevem explicitamente como os objetos interagem para cumprir suas responsabilidades. Desenvolver um modelo dos objetos de domínio força a resolução de problemas prematuramente, mantendo a integridade conceitual do sistema e reduzindo a quantidade de tempo que uma equipe gasta para fazer o *refactoring* das suas classes, caso precise adicionar uma nova característica. Uma écnica sugerida pelos autores para modelar os objetos de domínio é a "*Object Modeling in Color with UML*"[9].

[8] Para maiores informações sobre a UML consultar BOOCH (1999).
[9] Técnica proposta por Peter Coad no livro *Java Modeling in Color with UML (Unified Modeling Language)* que consiste na utilização de cores, padrões e estratégias para a representação de diagramas de classe.

- **Desenvolvendo através de características:** É feita a identificação das características do sistema através de modelos de casos de uso, *user stories* (que podem ser incorporadas do XP) ou qualquer outra ferramenta que sirva para descrever os requisitos do sistema. Tradicionalmente, as características identificadas são divididas em grupos representados através de uma lista hierárquica. Após ser definida a lista de características, inicia-se o projeto e a construção de cada uma delas. Uma vez identificadas, as características serão utilizadas para guiar o desenvolvimento no FDD, o objetivo é mostrar o progresso através da implementação das mesmas.

A característica ou conjunto de características selecionadas para serem implementadas durante a iteração devem ser executadas dentro de 2 semanas. Este é o limite máximo.

Segundo Palmer (2003), toda característica poderia ser descrita utilizando o seguinte padrão: <ação> <artigo> <resultado> <preposição> <artigo> <objeto>. Por exemplo, "calcular o total da venda".

Todas as características devem ser agrupadas de acordo com a mesma área do negócio, onde o nome do grupo deve obedecer ao seguinte padrão: <ação – verbo no particípio> <artigo> <objeto>. Por exemplo, "Comprando um produto" ou "Realizando o pagamento".

- **Propriedade Individual da Classe:** Esta prática sugere que exista um responsável para cada classe ou conjunto de classes. Algumas vantagens da propriedade individual da classe são:
    - Um indivíduo é responsável pela integridade conceitual de uma classe. Quando novos métodos forem adicionados a esta classe o proprietário terá segurança de que o objetivo da classe será mantido e que as modificações serão feitas corretamente. O proprietário do código pode executar uma atualização mais rapidamente do que um outro desenvolvedor que não esteja familiarizado com esta parte do código.
    - O proprietário do código tende a fazer sempre melhor o que é de sua total responsabilidade.

Algumas desvantagens da propriedade individual da classe são:

- Pode acontecer de um desenvolvedor necessitar fazer modificações em classes que dependem de mudanças que estão sendo feitas nas classes que são de outro desenvolvedor. Isto poderia resultar em perda de tempo até que os dois desenvolvedores entrassem em sintonia.
- Caso o proprietário da classe saia do projeto por alguma razão, isso acarretará na necessidade de um entendimento do funcionamento desta por outro desenvolvedor, ocasionando em perda de tempo.

- **Equipes de Características:** A prática da propriedade individual da classe atribui classes a desenvolvedores específicos. Contudo, sabe-se que o desenvolvimento deve ser por característica, assim é necessário organizar equipes para as mesmas. São escolhidos desenvolvedores para serem líderes de equipes e é atribuído a eles um conjunto de características. Abaixo estão algumas considerações sobre as equipes de características:
  - Uma equipe de característica deve ter de 3 a 6 membros.
  - Por definição, uma equipe de características compreende todos os proprietários das classes relacionadas ao desenvolvimento de uma característica em particular.
  - Cada membro de uma equipe de característica executa uma característica sob a orientação de um desenvolvedor experiente.
  - Eventualmente, um proprietário da classe pode trabalhar para uma outra equipe da característica. Os desenvolvedores podem pertencer a 2 ou 3 equipes de características simultaneamente por um curto período de tempo.
  - Os programadores principais devem trabalhar em conjunto para resolver conflitos e para evitar sobrecarregar qualquer desenvolvedor em particular.
- **Inspeções:** O FDD confia em inspeções, feitas durante e ao final de cada iteração para assegurar a qualidade do projeto e do código. O objetivo preliminar das inspeções é a detecção de defeitos. Todos necessitam ver as inspeções como uma ferramenta para eliminação de erros e como uma grande oportunidade de aprendizado.

- **Construções Regulares:** Construções regulares possibilitam a detecção de erros de integração prematuramente. Uma construção regular assegura também que haja sempre um sistema atual e executável para ser apresentado ao cliente. Estas construções acontecem durante as iterações, onde acontece o desenvolvimento de um conjunto de características.
- **Administração de Configuração:** O FDD sugere a utilização de um sistema de controle de versões para datar e manter um histórico das alterações feitas em cada classe. Bem como, no que se refere aos requisitos, análise e o projeto de modo que facilite a visualização das modificações feitas. Todo artefato utilizado durante o desenvolvimento do sistema é candidato ao controle de versão. É sugerido solicitar uma autenticação dos clientes e gerentes de projeto para cada modificação, dando uma certa segurança ao contrato comercial entre o cliente e a empresa que está desenvolvendo o produto.
- **Relatório dos resultados:** O FDD sugere que todos os resultados ocorridos durante o projeto sejam disseminados para todos os membros da equipe e clientes. Podem ser disponibilizados através de páginas na internet, relatórios impressos etc.

2.3.2 As Fases do Processo do FDD

Segundo Coad (1999), o FDD possui definições claras em relação às etapas do processo (Figura 5), pois se acredita que elas contribuem para o sucesso do projeto, fazendo com que os envolvidos no mesmo o sigam, evitando que busquem alternativas próprias que poderiam dificultar o seu andamento. O FDD possui o foco no projeto e construção e é composto por cinco etapas: três delas (desenvolver um modelo, construir uma lista de características e planejar cada uma delas) são realizadas no início do desenvolvimento do sistema e as duas últimas (projeto e construção de cada característica) são completadas dentro de cada iteração.

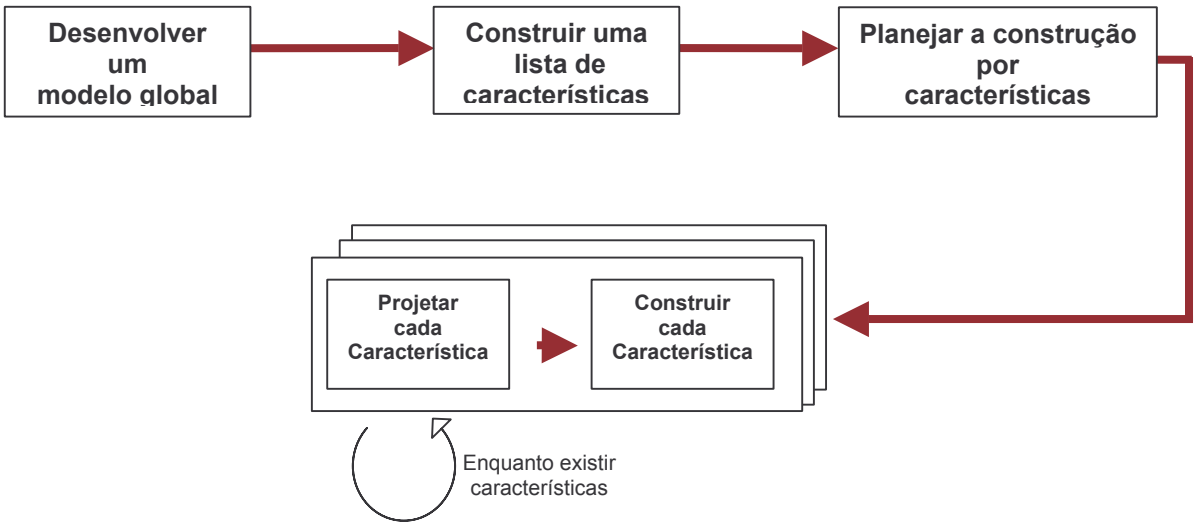


Figura 5. Fases do Processo FDD.

A seguir, serão descritas as etapas do FDD de acordo com Highsmith (2002a), De Luca (2002) e Abrahamsson (2002).

2.3.2.1 Desenvolver um modelo global.

É a primeira etapa do processo no FDD, onde são definidos o contexto e os requisitos do *software*. As principais pessoas envolvidas nesta etapa são o Especialista do Domínio e o Projetista. A tarefa requerida nesta etapa é a modelagem do domínio da aplicação, onde são construídos o diagrama de classe UML, o(s) diagrama(s) de seqüência UML e uma lista informal das características.

2.3.2.2 Construir uma lista de características.

O objetivo da segunda etapa do processo no FDD, conforme apresenta Highsmith (2002a), é construir uma lista completa de todas as características do produto a ser desenvolvido. Na lista, a equipe de desenvolvimento apresenta cada função esperada pelo cliente baseada na lista de características gerada na etapa anterior, podendo ser necessária a inclusão de novas classes no modelo de domínio, que deverá também ser refeito, apresentando agora os atributos e os métodos de cada classe. Cada característica da lista apresenta uma estimativa de tempo para o desenvolvimento e uma prioridade que determina a ordem/importância no desenvolvimento. A lista principal é dividida em listas menores que são agrupadas de acordo com as suas dependências, sendo que a execução de cada grupo de características não deverá exceder a 2 semanas.

2.3.2.3 Planejar a construção por características.

Nesta etapa, é construído um plano de execução dos conjuntos de características. As classes são distribuídas aos seus proprietários. A equipe de planejamento formada pelo Gerente do Projeto, Programador Chefe e Gerente de Desenvolvimento, é a responsável pela elaboração do plano de quais características serão desenvolvidas.

Conforme Highsmith (2002a), planejar no FDD implica também em verificar vários fatores que poderão auxiliar ou prejudicar no desenvolvimento de *software* como determinação dos riscos do projeto, complexidade, balanceamento dos trabalhos, etc.

2.3.2.4 Projetar cada característica.

Nesta etapa são executadas as seguintes tarefas pela equipe e o programador chefe:

- estudar a documentação existente;
- desenvolver o diagrama de seqüência para o conjunto de características;
- refinar o modelo do objeto;
- reescrever as classes e os métodos;
- inspecionar o projeto.

São resultados desse processo: o diagrama de seqüência detalhado, o diagrama de classes atualizado, além das características com toda documentação.

2.3.2.5 Construir cada característica.

É a última etapa de cada iteração do processo no FDD. A seguir, algumas tarefas requeridas nessa etapa que são executadas pela equipe e o programador chefe:

- implementação das classes e métodos;
- inspeção de código;
- teste de unidade;
- integração;
- testes de integração;

- entrega do incremento.

Todos os conjuntos de características devem passar pelas etapas de projeto e construção até que o sistema esteja concluído.

2.3.3 A Equipe do FDD

Segundo Palmer (2002), o FDD classifica os papéis da equipe em três categorias: Chave, Suporte e Adicional.

Os seis papéis chaves são:

O **Gerente de Projeto** é o responsável financeiro e administrativo do projeto. Uma de suas responsabilidades é gerenciar a viabilidade do projeto oferecendo todas as condições necessárias à equipe para o desenvolvimento do trabalho. No FDD, a "última palavra" é dada por ele.

O **Arquiteto Principal** é quem elabora o projeto geral do *software* a ser desenvolvido, tomando decisões finais em relação ao projeto técnico. Essa função poderá ser dividida entre o Projetista de Domínio e o Projetista Técnico.

O **Gerente de Desenvolvimento** é responsável por gerenciar as atividades diárias do projeto resolvendo problemas que poderão ocorrer com a equipe. Pode combinar as atividades desenvolvidas pelo Arquiteto Principal e Gerente de Projeto.

O **Programador Chefe** geralmente é o programador com maior experiência dentro da equipe, participando na análise dos requisitos e no projeto do *software*. É considerado como um dos papéis mais importantes no projeto FDD, atuando principalmente nas duas últimas etapas do processo.

O **Proprietário de Classe** trabalha subordinado ao Programador Chefe, tendo como tarefas projetar, codificar, testar e documentar. É o responsável pelo desenvolvimento das classes atribuídas a ele.

O **Especialista no Domínio** pode ser um usuário, analista de negócios, cliente ou qualquer pessoa que conheça bem o domínio do problema. Sua tarefa é informar as funcionalidades que deverão ser atendidas pelo *software* e entender como os requisitos estão sendo desenvolvidos.

Os seis papéis de suporte são:

O **Gerente de domínio** conduz os peritos de domínio a resolver a diferenças de opinião relativa aos requisitos do sistema.

O **Gerente de Versão** é responsável por controlar o progresso no desenvolvimento através de constantes revisões em conjunto com o Programador Chefe. Ele informa suas atividades ao Gerente de Projeto.

O **Especialista na Linguagem** é o membro da equipe responsável por possuir um conhecimento completo de uma linguagem de programação específica ou tecnologia. Este papel é particularmente importante quando a equipe do projeto resolve utilizar uma nova tecnologia.

O **Coordenador de Configuração** é a pessoa responsável pelas tarefas de administração do sistema de controle de versão e a publicação da documentação, durante a atividade de construção.

O ***Toolsmith***[10] é responsável por construir ferramentas de suporte para o desenvolvimento, teste e conversão de dados no projeto. Também pode trabalhar com modelagem e manutenção de bancos de dados e *websites* para propósitos específicos do projeto.

O **Administrador de Sistema** possui a tarefa de configurar, administrar e diagnosticar os servidores, estações de trabalho e desenvolvimento e testar os ambientes usados pela equipe do projeto.

Por fim, os três papéis adicionais são:

Os **Testadores** verificam se o sistema que está sendo produzido satisfará os requisitos do cliente. Pode ser uma equipe independente ou parte da equipe de projeto.

Os **Desenvolvedores** são responsáveis por converter dados existentes ao formato requerido pelo novo sistema e participar no desenvolvimento de novos lançamentos. Pode ser uma equipe independente ou parte da equipe de projeto.

O **Escritor técnico** é responsável pela documentação de usuário.

[10] Não foi encontrada uma tradução para o termo *Toolsmith.*

De acordo com Palmer (2002), o FDD é indicado em projetos que envolvam entre 10 e 30 pessoas na equipe de desenvolvimento do projeto e é satisfatório tanto para projetos novos, quanto para projetos que já se encontram em desenvolvimento, ou projetos que tem como objetivo a entrega de uma segunda versão de um sistema já existente. Os autores também sugerem que as organizações devem adotar o método gradualmente, em pequenas partes até chegar em sua totalidade.

Para Highsmith (2002a), este método, juntamente com o XP, é considerado "um dos melhores métodos ágeis existentes na atualidade no mercado".

2.4 Adaptive Software Development (ASD)

O *Adaptive Software Development* (Desenvolvimento Adaptável de *Software*) foi desenvolvimento por James A. Highsmith III. Muitos dos princípios do ASD tiveram origem numa pesquisa feita anteriormente pelo autor sobre métodos de desenvolvimento iterativo e incremental. O antecessor mais conhecido do ASD é o "*RADical Sofware Development*[11]", desenvolvido por Highsmith e S. Bayer (ABRAHAMSSON, 2002).

Conforme Highsmith (2002a), o ASD encoraja o Desenvolvimento Incremental e iterativo. Fundamentalmente, seu objetivo é prover um processo para auxiliar no desenvolvimento ágil de *software*. Este é um processo dedicado à aprendizagem contínua, dirigido a mudanças, reavaliações e grande colaboração entre desenvolvedores e clientes.

2.4.1 As Práticas do ASD

De acordo com Highsmith (2002a), o ASD não possui um conjunto de práticas definidas e sim um conjunto de propriedades que caracterizam o processo de desenvolvimento adaptativo. A seguir são apresentadas as propriedades, conforme Abrahamsson (2002):

[11] Para mais informações a respeito do RAD, acessar: http://www.adaptivesd.com/articles/RAD_article.htm

- **Dirigido a missões:** As atividades para cada ciclo de desenvolvimento são justificadas através de uma missão, onde cada iteração possui a sua, e que pode mudar ao longo do projeto.
- **Baseado em características:** As atividades de desenvolvimento devem ser orientadas à características, ou seja, a construção de uma pequena parte do *software* de cada vez.
- **Iterativo:** O processo de desenvolvimento é dividido em pequenos ciclos (iterações) com o objetivo final de entregar um grupo de requisitos implementados que satisfaça a missão definida para cada iteração.
- **Prazos pré-fixados:** A ambigüidade em projetos de *software* complexos pode ser reduzida fixando-se prazos finais tangíveis em uma base regular, forçando os participantes do projeto a tomarem decisões que podem representar algum risco no começo do projeto.
- **Tolerância a mudanças:** As mudanças são freqüentes. Assim, é sempre melhor estar pronto a adaptá-las do que controlá-las, fazendo uma constante avaliação de quais componentes podem mudar.
- **Orientado a riscos:** Características de alto risco são desenvolvidas primeiro.

2.4.2 As Fases do Processo do ASD

O processo do ASD é focado em resultados, não em tarefas, e os resultados são as características desenvolvidas durante as iterações (HIGHSMITH 2002a). O ASD possui ciclos de três fases: Especulação, Colaboração e Aprendizado (Figura 6).

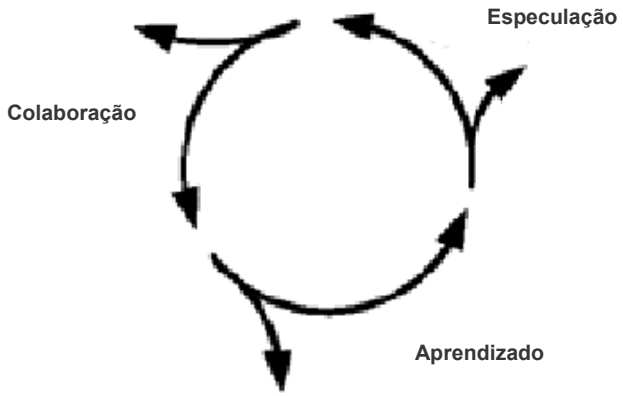


Figura 6. O Processo ASD
(adaptado de Abrahamsson, 2002).

A seguir são descritas as fases do processo ASD (Figura 7) de acordo com Highsmith (2002a) e Abrahamsson (2002).

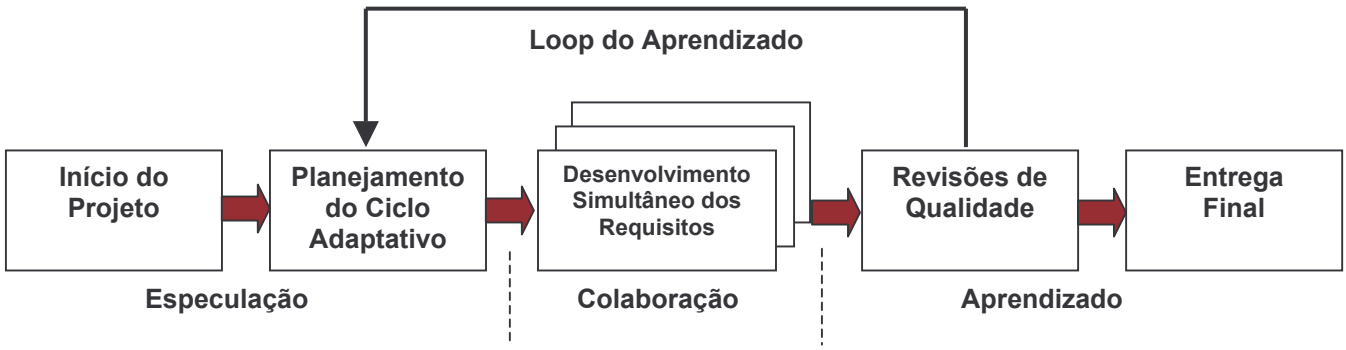


Figura 7. As Fases do Processo ASD.
(adaptado de Highsmith, 2002a).

2.4.2.1 Especulação

A fase de Especulação consiste na definição dos prazos, objetivos e de um plano baseado em características, pois todo processo é focado em partes do sistema. Esta fase possui duas sub-fases chamadas Inicio do Projeto e Planejamento do Ciclo Adaptável.

Na sub-fase Inicio do Projeto, é estabelecido o objetivo do projeto e é feito um esboço das características, dando uma idéia inicial do tamanho do sistema. A identificação dos riscos também é contemplada nesta sub-fase. Para projetos pequenos ela não deve durar mais que 5 dias, mas este prazo aumenta para 2 ou 3 semanas para projetos maiores. Os prazos e custos são estabelecidos com base na complexidade das características. Vale salientar que os prazos e custos são vagos, uma vez que as mesmas podem sofrer alterações.

Na sub-fase Planejamento do Ciclo Adaptável, são definidas a quantidade de iterações bem como o tempo necessário para cada uma delas. Sugere-se que cada ciclo dure de 4 a 8 semanas. O tamanho do projeto e o grau de incerteza em relação a possíveis alterações são dois fatores importantes na determinação do tempo de cada iteração.

Após estabelecido o número de iterações e o tempo para cada uma delas, a equipe elabora uma missão para cada iteração. As características a serem desenvolvidas em cada iteração são decididas em conjunto pelos desenvolvedores e clientes, bem como a ordem em que serão implementadas. Estas definições são feitas em sessões de JAD[12] (*Joint Application Development).* Uma sessão de JAD é essencialmente um *workshop* onde os desenvolvedores e os representantes do cliente se encontram para discutir sobre as características do sistema. É utilizada uma planilha eletrônica para documentar o planejamento das iterações.

2.4.2.2 Colaboração

A fase de Colaboração é composta por várias iterações, onde acontece o desenvolvimento das características escolhidas para fazerem parte da mesma. Estas características podem ser desenvolvidas simultaneamente.

É importante salientar que o enfoque do ASD é no resultado final e suas qualidades, ou seja, na qualidade do sistema desenvolvido em cada iteração. O método não sugere tarefas ou processos a serem utilizados para produzirem estes resultados.

2.4.2.3 Aprendizado

[12] Wood, J. and D. Silver, Joint Application Development, 2nd ed., New York : Wiley, 1995.

A fase de Aprendizado se divide em duas sub-fases, Revisões de Qualidade e Entrega Final.

As Revisões de Qualidade acontecem no final de cada iteração, onde é apresentado um *software* contemplando os requisitos escolhidos para serem desenvolvidos durante a mesma.

Estas revisões possuem o objetivo de verificar:

- A qualidade funcional do sistema através da definição de grupos de clientes para testar a aplicação.
- A qualidade técnica, onde um par de programadores é responsável por revisar e avaliar o código do sistema.
- O desempenho da equipe, através de revisões das tarefas realizadas durante a iteração.
- O progresso geral do projeto, onde são verificados se os objetivos da iteração foram alcançados.

A sub-fase de Entrega Final consiste na entrega final do sistema para o cliente.

2.4.3 Os Papéis da Equipe

Não foram encontradas descrições detalhadas a respeito de como deve ser formada a equipe no método ASD. Entretanto, Abrahamsson (2002) sugere a necessidade de um Patrocinador (*Executive Sponsor*), nomeado como a pessoa com responsabilidade global em relação ao produto que está sendo desenvolvido.

Como o método incorpora sessões de JAD no seu processo de desenvolvimento, pode-se citar os além dos membros que participam destas sessões, o Patrocinador; O Facilitador que possui a responsabilidade de liderar e planejar as sessões; O Escriba responsável por fazer as anotações necessárias; E os Desenvolvedores como possíveis membros da equipe do ASD, (ABRAHAMSSON 2002). Apesar de não apresentar papéis bem definidos, Highsmith (2002a) enfatiza a importância da existência da colaboração entre os envolvidos no projeto.

Segundo Abrahamsson (2002), o ASD não possui limitações para sua aplicação, ou seja, pode ser utilizado tanto em sistemas simples quanto complexos e também não estabelece número máximo de pessoas na sua equipe.

Uma característica interessante do ASD é que ele não obriga que os membros da equipes estejam num mesmo local como outros métodos ágeis. Porém, o desenvolvimento distribuído está relacionado principalmente com as habilidades da equipe, que devem ser as mais especializadas possíveis. Para isso, o ASD oferece algumas técnicas[13] para aumentar a colaboração entre a mesma sugerindo estratégias para compartilhar a informação e utilizar ferramentas de comunicação para apoiar o desenvolvimento distribuído.

2.5 Agile Modeling (AM)

A *Agile Modeling* (Modelagem Ágil) foi introduzida por Scott W. Ambler em 2002 e tem como objetivos principais a geração de modelos e documentação eficazes que dêem suporte ao desenvolvimento de sistemas, sem que o processo perca a agilidade.

Como a maioria dos métodos ágeis, a AM adota um conjunto de práticas guiado por princípios e valores. Não é um processo prescritivo, ou seja, não define procedimentos detalhados sobre como criar um determinado tipo de modelo, invés disso, fornece conselhos sobre como utilizar a tarefa de modelagem de forma eficiente e ágil (AMBLER 2004).

Uma questão importante sobre a AM é que ela não é um método completo. Embora mostre a importância das atividades de programação, testes, gerência, suporte ao sistema etc, ela não contempla as mesmas no seu escopo. Como o seu foco é na modelagem e documentação, Ambler (2004) afirma que a AM deve ser utilizada em conjunto com outros métodos de desenvolvimento.

---

[13] De acordo com Abrahamsson (2002), estas técnicas são sugeridas no livro *"Agile Software Development: A Collaborative Approach to Managing Complex Systems"* publicado em 2000 por Highsmith.

2.5.1 Os Valores da AM

De acordo com Ambler (2004), a AM adota uma perspectiva similar ao XP em relação aos valores utilizados como base para o método. Além da Comunicação, Simplicidade, *Feedback* e Coragem, a AM sentiu a necessidade de incorporar mais um valor: a Humildade. Como os quatro primeiros valores já foram descritos anteriormente no Capítulo 3 sobre o XP, a seguir será apresentado apenas o valor que ainda não foi abordado.

A **Humildade** deve fazer parte da maneira como os envolvidos no projeto se relacionam. Os "modeladores ágeis" devem respeitar as pessoas com as quais trabalham, percebendo que elas talvez possuam outras prioridades e experiências e, portanto, terão pontos de vista diferentes, bem como saber admitir que não sabem tudo e haverá um momento em que precisarão pedir ajuda para realizar bem o seu trabalho.

2.5.2 As Práticas da AM

O ponto forte da AM são as suas práticas. Elas estão divididas em práticas básicas que devem ser adotadas em sua totalidade para garantir um bom resultado e em práticas suplementares que podem ser utilizadas de acordo com a necessidade.

As práticas básicas e suplementares da AM estão organizadas em categorias. A seguir elas serão apresentadas segundo Ambler (2004).

- **Práticas Básicas:**
  - **Práticas para modelagem iterativa e incremental:**
    - *Aplicar o(s) artefato(s) correto(s):* Existe uma grande quantidade de artefatos e diagramas para representar graficamente cada modelo e a escolha pelo correto é extremamente importante. Por isso, a melhor maneira de não errar é saber o máximo possível sobre o artefato escolhido, bem como sobre outras alternativas existentes.
    - *Criar diversos modelos em paralelo:* Pode ser que seja necessária a geração de vários modelos ao mesmo tempo. Por exemplo, quando se está desenvolvendo em Java, pode ser que seja necessária a criação de um diagrama de classes UML para definição da estrutura do sistema, um

diagrama de estados UML para explorar o funcionamento interno de uma classe, um outro diagrama de seqüência UML para mostrar a troca de mensagens entre dois objetos, enfim, o trabalho poderá ser mais produtivo utilizando diversos modelos simultaneamente do que focando um único por vez.

- *Iterar em outro artefato:* Dificuldades em trabalhar com um determinado artefato é um sinal que se deve repetir o mesmo processo com um outro artefato. Por exemplo, se através de um caso de uso essencial não se consegue descrever a lógica do negócio, pode ser que seja mais indicado utilizar um fluxograma ou um modelo CRC.
- *Modelagem incremental:* Ao utilizar uma estratégia de desenvolvimento iterativa (premissa ágil), também se utiliza uma estratégia de modelagem incremental. Assim, modela-se um pouco, codifica-se um pouco, testa-se um pouco e entrega-se uma parte do sistema.

- **Práticas para um trabalho de equipe eficaz:**
  - *Modelagem em conjunto:* Não há problema algum em desenhar um esboço simples sozinho, mas depois de concluído é importante discutir o resultado com outras pessoas. Esta prática ajuda a melhorar a comunicação, a criar um vocabulário comum entre a equipe e aumenta a chance da realização de um bom trabalho.
  - *Participação ativa do cliente:* Esta prática está diretamente relacionada à prática anterior e descreve a necessidade de se ter o cliente ou qualquer pessoa com habilidade para fornecer informações relativas aos requisitos do sistema em construção, presente no momento que for preciso.
  - *Posse coletiva:* Todos podem trabalhar em qualquer modelo ou artefato no projeto, caso necessitem. Esta prática permite o aumento do conhecimento intelectual, pois não existe apenas um especialista e sim vários, além de promover o entendimento do sistema entre os membros da equipe.
  - *Visualização dos modelos:* Propõe a criação de um local físico (uma parede) ou virtual (página *web*), onde todos os modelos gerados são expostos para a

equipe e o cliente. Desta forma o cliente acompanha o progresso do trabalho e a equipe tem os modelos a sua disposição para eventuais consultas.

- **Práticas que permitem a simplicidade:**
  - *Criação de conteúdo simples:* O conteúdo real dos modelos deve ser o mais simples possível, desde que satisfaça as necessidades dos clientes e da equipe. Um modelo é simples quando comunica tudo o que deve ser comunicado, não possui informação duplicada e tem a menor quantidade de elementos possível.
  - *Apresentação de modo simples:* Semelhante à prática anterior, sendo que esta enfatiza a forma como é apresentado o modelo. Algumas técnicas para simplificar os diagramas incluem evitar: cruzamento de linhas; linhas curvas; linhas diagonais; itens que significam a mesma coisa com tamanhos diferentes, itens demais e detalhes desnecessários.
  - *Utilização de ferramentas simples:* Como a proposta é agilidade e simplicidade, é importante que os modeladores utilizem uma ferramenta que supra as necessidades do projeto, não importando se é uma ferramenta CASE (*Computer Aided Software Engineering*) de última geração ou um simples desenho feito a lápis em uma folha de papel.
- **Práticas para validar o trabalho:**
  - *Considerar a testabilidade:* As atividades de testes no código não fazem parte da AM, pois o método é focado apenas na modelagem do sistema. Portanto, os "modeladores ágeis" devem ter em mente a seguinte questão: "Como isto pode ser testado?".
  - *Comprovar com código:* A melhor maneira de validar um modelo é escrever o código correspondente a ele. A ordem das atividades é modelar, codificar e testar.

- **Práticas Suplementares:**
  - **Práticas para melhorar a produtividade:**

- *Aplicar as convenções de modelagem*: A idéia é que a equipe adote um conjunto comum de convenções de modelagem. O padrão mais comum é a UML, mas nem tudo é possível ser representado pela UML. Neste caso, os padrões devem incluir descrições de notação para qualquer modelo não-UML. Vale salientar que a compreensibilidade é mais importante que seguir convenções e padrões.
- *Utilizar padrões de projeto*[14] *com moderação:* Se existir a suspeita que um padrão de projeto seja aplicável, modele-o de forma a implementar a mínima funcionalidade necessária no momento, mas que se torne fácil de refazer mais tarde, caso necessário. Esta prática está diretamente relacionada com a simplicidade.
- *Reuso dos recursos já existentes:* A reutilização de modelos agiliza a tarefa de modelagem. Esta prática reforça a importância da existência dos modelos, bem como da documentação.

- **Práticas para a documentação ágil:**
  - *Descartar modelos temporários:* Devem ser guardados apenas os modelos que forem realmente úteis. Modelos temporários geralmente estão desatualizados e guardando-os, além de contribuir para a desorganização, corre-se o risco de alguém desavisado tomar uma decisão baseada em um modelo incorreto.
  - *Formalização dos modelos de contrato:* Os modelos de contrato servem para formalizar o que é solicitado pelo cliente e o que é de responsabilidade da equipe desenvolvedora.
  - *Atualizações somente quando necessário:* As atualizações nos modelos devem acontecer somente quando necessário, do contrário é perda de tempo. Esta prática está diretamente relacionada a "*Descartar modelos temporários*", pois quanto menos modelos e documentação permanentes forem mantidos, mais fácil será atualizá-los quando preciso.

[14] Recomenda-se a leitura do livro "Padrões de Projeto – Soluções Reutilizáveis de Software Orientado a Objetos". Gamma. E; Hekm, R; Johnson, R; Vlissides, J. Ed. Bookman. 2000, para mais informações sobre padrões de projeto.

- **Práticas relacionadas com a motivação:**
  - *Modelar para entender:* A modelagem serve para auxiliar na compreensão do problema a ser solucionado. Sempre buscando a melhor e mais simples solução que satisfaça os requisitos do sistema.
  - *Modelar para comunicar:* A modelagem serve também para comunicar aos outros, integrantes ou não da equipe, como o sistema foi desenvolvido, bem como seus propósitos.

A Figura 8, mostra a relação entre as práticas descritas anteriormente, organizando-as em sete categorias. As categorias "Validação", "Iterativa e Incremental", "Trabalho em Equipe" e "Simplicidade" consolidam as práticas básicas. As práticas suplementares estão consolidadas nas categorias "Documentação", "Motivação" e "Produtividade". Segundo Ambler (2004), as práticas da AM complementam-se, apóiam-se e habilitam umas as outras.

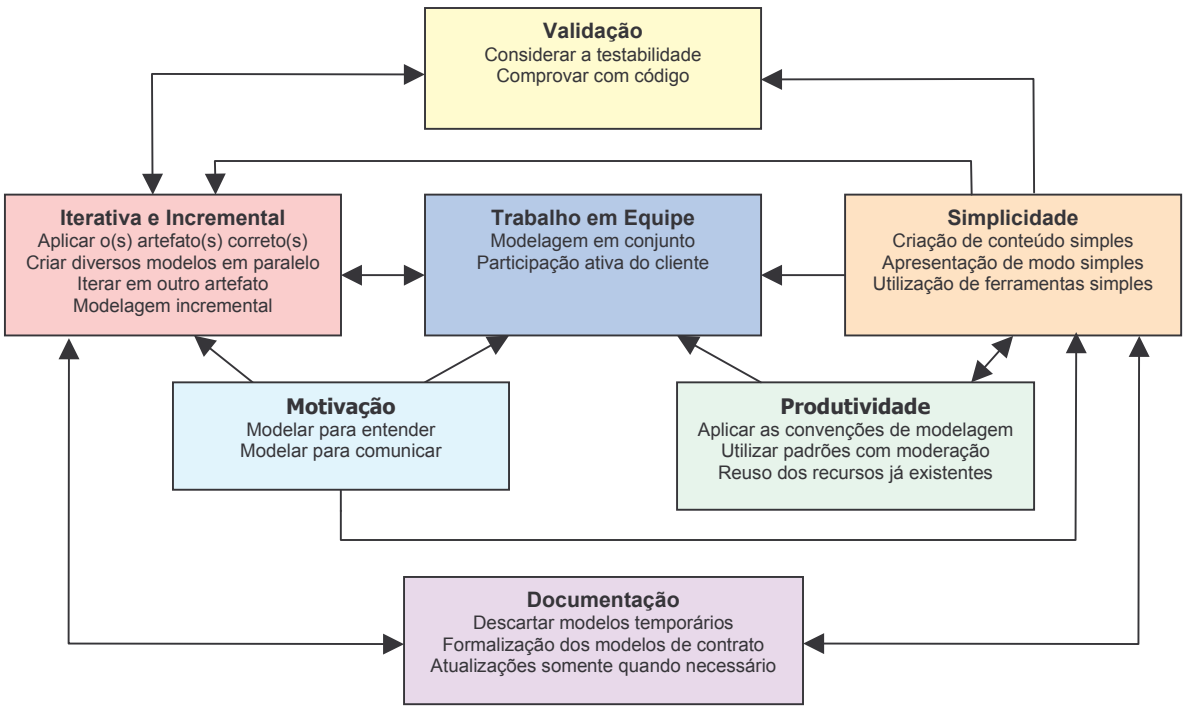


Figura 8. A Relação entre as Práticas AM.
(adaptado de Ambler, 2004).

2.5.3 Sessões e Documentação Ágeis

Conforme descrito anteriormente, a AM não é um método completo de desenvolvimento, e sim um método que complementa outros processos de desenvolvimento, conforme observado na Figura 9, AMBLER (2004). Sendo assim, a AM não possui um ciclo de vida de desenvolvimento próprio. Outros métodos a incorporam em seu ciclo de vida.

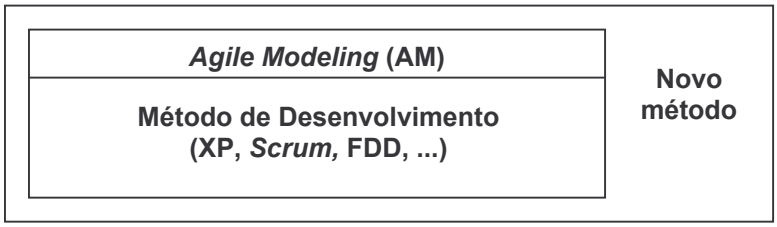


Figura 9. A AM complementando outros métodos.
(adaptado de Ambler, 2004).

Por este motivo nesta sessão não serão descritas "As Fases do Processo AM", como nos outros capítulos sobre os métodos ágeis. Neste caso, é mostrado como deve ser conduzida uma sessão de modelagem para a geração de modelos e como estes modelos se transformam em documentação, conforme Ambler (2004).

2.5.3.1 Sessões de AM

Uma sessão de modelagem é uma atividade onde várias pessoas têm como objetivo principal o desenvolvimento de um ou mais modelos, fornecendo uma oportunidade delas colaborarem entre si e através da comunicação de suas necessidades chegarem a um melhor entendimento e trabalharem na busca de uma solução. São nas sessões de modelagem ágil que serão aplicadas as práticas da AM.

A duração das sessões pode variar de alguns minutos a no máximo 3 dias, sendo que a maioria leva entre 10 e 30 minutos. A justificativa para a grande variação de tempo é que no início do projeto talvez exista a necessidade de longas sessões, uma vez que durante esta fase, a preocupação é em obter uma visão global do projeto, enquanto que depois, a modelagem acontecerá em iterações focadas em partes específicas do sistema.

Após cada sessão de modelagem, devem ser feitas validações em cima do(s) modelo(s) criado(s). Estas validações podem acontecer através de revisões com outra pessoa da equipe ou através da prática "Comprovar com código". Quanto mais rápido for o retorno, mais chance de saber se o que está sendo modelado reflete ou não os requisitos e a arquitetura do *software*.

Existem algumas sugestões que podem ser adotadas para "forçar" uma sessão de modelagem curta. Em primeiro lugar, as sessões devem ser realizadas em pé, ao redor de um quadro ou mesa, pois a maioria das pessoas só quer ficar de pé por curtos períodos. Em segundo, fazer das sessões curtas um hábito, assim quando as sessões durarem mais tempo, as pessoas ficarão inquietas. Em terceiro, manter o foco das sessões em um único tópico, ou seja, nos requisitos daquele momento. Em quarto lugar, as sessões devem ser finalizadas assim que o objetivo for alcançado.

As sessões de modelagem devem focar a criação de modelos relacionados às fases mais importantes do desenvolvimento tradicional, como requisitos, análise, arquitetura e projeto.

Em cada uma dessas sessões de modelagem a equipe trabalha em diversos modelos de uma só vez (prática "Criar diversos modelos em paralelo"), os que forem mais apropriados a cada fase. Em uma sessão de modelagem de requisitos, por exemplo, podem ser criados diagramas de casos de uso UML ou *user stories.* Em uma sessão de modelagem de projeto, poder ser gerados diagramas de classe UML ou diagramas de estado UML.

O importante é sempre levar em consideração que a ferramenta escolhida para gerar os modelos em cada uma das fases, deve suprir as reais necessidades do projeto (prática "Aplicar o(s) artefato(s) correto(s)").

2.5.3.2 Documentação Ágil

Alguns modelos se tornarão documentos, ou parte destes, ou serão descartados após cumprir suas finalidades. Alguns modelos serão usados para guiar a implementação ou apenas para guiar o desenvolvimento de outros.

Sob o ponto de vista da AM, um documento é qualquer artefato externo ao código-fonte, cujo objetivo é transmitir informações de forma persistente. Já o conceito

de modelo, é uma abstração que descreve um ou mais aspectos de um problema ou uma solução possível para este. O código-fonte é uma seqüência de instruções, incluindo os comentários que as descrevem, para um sistema computacional. E o termo documentação inclui tanto os documentos quanto os comentários do código-fonte.

Um fator importante a ser levado em consideração são os tipos de documentos a serem criados durante o desenvolvimento de um sistema. A seguir alguns dos documentos mais comuns que poderão ser criados para serem entregues como parte da documentação do sistema:

Documento (s)	Público	Descrição
Modelos de contrato	Outras equipes	Descreve a interface para um sistema ou uma parte do mesmo
Decisões de projeto	Desenvolvedores Desenvolvedores de manutenção Gerentes de projeto	Um resumo de decisões críticas relacionadas ao projeto e à arquitetura que a equipe tomou durante o desenvolvimento
Visão geral executiva	Gerência sênior Gerência de usuários Gerência de projeto	Uma definição da visão de sistema e um resumo da estimativa atual de custos, benefícios previstos, riscos, estimativas de pessoal e marcos no cronograma.
Documentação de operações	Pessoal de operações	Inclui uma indicação das dependências com as quais o sistema está envolvido; a natureza de sua interação com outros sistemas, banco de dados e arquivos; referências e procedimentos de *backup*; um resumo dos requisitos de disponibilidade/confiabilidade para o sistema e diretrizes de resolução de problemas.
Visão geral do projeto	Desenvolvedores Gerentes Desenvolvedores de manutenção Pessoal de operação	Um resumo de informações cruciais, como a visão do sistema, principais contatos dos usuários, tecnologias e ferramentas usadas para construir o sistema e processos cruciais de operação.
Documentos de requisitos	Desenvolvedores Desevolvedores de manutenção Usuários Gerentes de usuários	Define o que o sistema irá fazer, resumindo ou compondo os artefatos de requisitos, como definições de regras de negócio, casos de uso, *user stories*, ou protótipos de interface essencial, por exemplo.
Documentos de suporte	Pessoal de suporte	Inclui materiais de treinamento específicos para o pessoal de suporte; toda a documentação de usuário para usar como referência na resolução de problemas; um guia para a solução de problemas; procedimentos para lidar com problemas difíceis; e uma lista de pontos de contato dentro da equipe de manutenção.

Documentação do sistema	Desenvolvedores de manutenção Desenvolvedores	Fornece uma visão geral do sistema e ajuda as pessoas a compreendê-lo. Algumas informações incluídas no documento são, uma visão geral da arquitetura técnica, da arquitetura do negócio e dos requisitos de alto nível do sistema. A arquitetura detalhada e os modelos de projeto, ou referências a eles, também podem estar incluídos.
Documentação de usuário	Usuários Gerentes de usuários	Inclui, um manual de referência, um guia de uso, um guia de suporte ou até mesmo materiais para treinamento.

Tabela 2. Documentos possíveis criados pela equipe de desenvolvimento. (adaptado de Ambler 2004)

A equipe deve se focar na criação de documentação que forneça o máximo valor para os clientes e deve ser criada somente quando for a melhor opção, o objetivo é uma documentação simples que cumpra a função desejada. Abaixo, serão listados alguns aspectos importantes relativos à documentação ágil.

- O ponto chave é a comunicação efetiva, não a documentação.
- A documentação deve ser pequena e econômica.
- A documentação só deverá ser criada se é realmente necessário.
- A documentação deve ser boa apenas o suficiente.
- A documentação faz parte do sistema tanto quanto o código-fonte.
- O objetivo principal da equipe é desenvolver *software*.
- O benefício de ter documentação deve ser maior do que o custo de criá-la e mantê-la.
- Cada sistema tem suas necessidades de documentação próprias e únicas. Não existe uma solução única.
- A pergunta é: Por que a documentação é necessária e por que se acredita que precisa dela? E não: Quem quer documentação?
- O investimento na documentação do sistema é uma decisão de negócio, não técnica.
- Atualizar a documentação apenas quando for necessário.

2.5.4 A Equipe da AM

A AM sugere três papéis compondo a equipe de modelagem: o Facilitador, o Escrivão e o Observador. Muitas sessões de modelagem funcionam perfeitamente sem ninguém nestes papéis, principalmente as sessões pequenas onde os desenvolvedores estão modelando para compreender uma parte do sistema. Entretanto, as sessões maiores de modelagem, especialmente as no início do projeto, geralmente terão pessoas nestes papéis (AMBLER 2004). A seguir serão apresentadas as responsabilidades atribuídas a cada um deles.

O **Facilitador** é o responsável pelo planejamento, execução e gerência das sessões de modelagem.

O **Escrivão** é o responsável pelo registro das informações durante uma sessão de modelagem.

O **Observador** não participa das sessões efetivamente, ele está lá para assistir o que acontece de modo que possa aprender. Ao envolvê-lo na sessão de modelagem, não só o número de pessoas agregando valor à sessão cresce, como também melhora o aprendizado da mesma.

Não existem comprovações a respeito do uso da AM no desenvolvimento de sistemas muito complexos, principalmente em sistemas críticos, como por exemplo, um sistema de controle de tráfego aéreo, pois os mesmos necessitam de uma documentação mais detalhada. Em relação ao tamanho da equipe, a AM é indicada para equipes médias e pequenas e que estejam localizadas na mesma área, pois o contrário pode afetar de forma negativa a comunicação entre os membros da mesma (AMBLER 2004).

Ambler (2004) sugere que a AM seja utilizada em conjunto com outros métodos ágeis. A adoção do método em projetos que utilizam processos prescritivos não é impossível, mas é provável que o sucesso do projeto seja comprometido. Portanto, em relação a sua aplicabilidade é recomendado que a AM seja utilizada de acordo com o sugerido pelo método que está sendo complementado por ela.

Como pôde ser observado, a AM não trata de nenhuma técnica de modelagem. Existem centenas de técnicas que podem ser aplicadas no projeto. A escolha de quais

adotar é da equipe de desenvolvimento. O importante é a aplicação das práticas da AM para a criação dos modelos e documentação, e não a técnica utilizada AMBLER (2004).

3 COMPARAÇÃO E ANÁLISE DOS MÉTODOS ÁGEIS

Neste capítulo será apresentada uma análise em relação as atividades propostas por cada um dos métodos ágeis durante o seu processo de desenvolvimento. A partir da mesma será possível, identificar as semelhanças existentes entre os métodos, e também, a identificação das práticas e papéis necessários para que estas atividades sejam realizadas de modo a serem incorporadas nas atividades que farão parte do *framework*.

3.1 Identificação e análise das atividades

Para possibilitar a identificação das atividades realizadas durante o processo de desenvolvimento dos métodos ágeis, foi necessário a definição de alguns critérios. Os critérios adotados para servirem de base para a identificação e análise são as atividades sugeridas pelo Desenvolvimento Incremental, uma vez que esta abordagem serviu de "inspiração" para o desenvolvimento ágil.

Segundo Larman (2003), os princípios do Desenvolvimento Incremental vem sendo utilizados implicitamente desde os anos 50 em projetos distintos utilizando diferentes abordagens de desenvolvimento, mas foi a partir da década de 90 que ele passou a chamar a atenção, principalmente com o surgimento da abordagem ágil.

O Desenvolvimento Incremental foi sugerido com o propósito de diminuir o "retrabalho" no processo de desenvolvimento e de proporcionar aos clientes oportunidades de adiar decisões sobre os requisitos, até que eles tenham alguma experiência com o sistema (SOMMERVILLE 2003). No Desenvolvimento Incremental (Figura 10) os clientes identificam, em um esboço, os requisitos do sistema e selecionam quais são os mais e quais são os menos importantes. Em seguida é definida uma série de estágios (iterações) de entrega, onde em cada um é fornecido um subconjunto de funcionalidades executáveis. As funcionalidades a serem desenvolvidas em cada iteração dependem da prioridade de cada uma delas (SOMMERVILLE 2003).

Após a identificação dos incrementos, as funcionalidades a serem entregues na primeira iteração são detalhadas e desenvolvidas. Paralelamente a este

desenvolvimento, outras funcionalidades podem ser analisadas para fazerem parte dos outros incrementos. Uma vez que cada incremento é concluído e entregue, os clientes podem colocá-lo em operação. O fato dos clientes poderem experimentar o sistema gradualmente facilita o esclarecimento das funcionalidades para os incrementos subseqüentes e à medida que novos incrementos são concluídos, eles são integrados às iterações existentes, de modo que o sistema melhora a cada novo incremento entregue (SOMMERVILLE 2003).

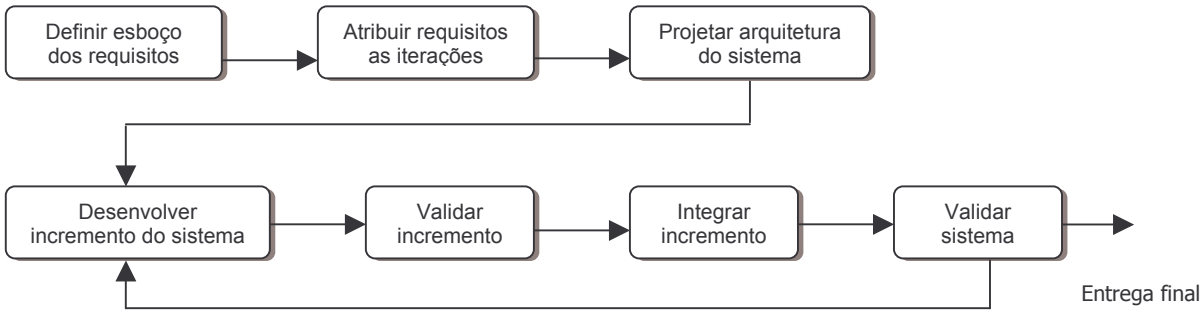


Figura 10. Desenvolvimento Incremental (adaptado de Sommerville, 2003).

Sommerville (2003) afirma que os métodos ágeis são fundamentados no Desenvolvimento Incremental. Desta forma, acredita-se ser possível fazer a identificação e análise das atividades propostas pelos métodos ágeis utilizando como base as atividades do Desenvolvimento Incremental.

Ë importante salientar que, de acordo com Sommerville (2003), o processo utilizado para o desenvolvimento de cada incremento deve ser o mais apropriado, ou seja, não importa qual o processo será utilizado desde que o conjunto de atividades e resultados associados levem a produção de um produto de *software*.

Sendo assim, será apresentada a seguir, um estudo contendo as atividades realizadas pelos métodos ágeis XP, Scrum, FDD e ASD, juntamente com as práticas e papéis adotados pelas mesmas de acordo com as atividades propostas pelo Desenvolvimento Incremental.

Por não apresentar um processo próprio de desenvolvimento, da AM serão apresentados apenas os artefatos que podem ser gerados para cada uma das atividades de acordo com a Tabela 2 do Capítulo 2. Segundo a AM, as suas práticas e papéis devem ser adotados durante a geração destes artefatos, sendo assim, eles não serão listados no estudo apresentado, uma vez que estão relacionados com as atividades propostas pelo Desenvolvimento Incremental de uma forma geral. Vale salientar que toda a documentação gerada deve se manter sempre atualizada e o seu principal objetivo é satisfazer as necessidades das pessoas envolvidas no projeto. Para saber mais sobre as práticas e papéis da AM, sugere-se a leitura da sessão 2.5 deste trabalho.

Para cada uma das atividades propostas pelo Desenvolvimento Incremental será apresentada uma tabela contendo um resumo da atividade proposta e as práticas e os papéis associados à atividade para cada um dos métodos ágeis estudados. Logo após será apresentado como e quando estas atividades são realizadas durante o processo de desenvolvimento e como as práticas e papéis são adotados durante a realização da mesma.

Atividade - Definição do Esboço dos Requisitos			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	Clientes escrevem as *user stories*.	• Jogo do Planejamento • Metáfora • Cliente Presente	• Cliente • Gerente • Programadores
Scrum	Definição do *Product Backlog*	• *Product Backlog*	• *Scrum Master* • Dono do Produto • Cliente
FDD	Geração de artefatos para a documentação dos requisitos (casos de uso, *user stories,* Diag. de Classe e de Seqüência da UML)	• Desenvolvendo através de características	• Gerente de Projeto • Arquiteto Principal • Especialista no Domínio • Gerente de Domínio
ASD	Requisitos definidos durante as sessões JAD	-	• Patrocinador • Facilitador • Escriba
AM	Geração de um(a): Documento de Requisitos; Visão Geral Executiva; Visão Geral do Projeto	-	-

Tabela 3. Atividade - Definição do Esboço dos Requisitos.

No **XP**, a definição preliminar dos requisitos é realizada na fase de Exploração, onde são adotadas as práticas: Jogo do Planejamento que contempla, entre outras atividades, a escrita das *user stories* pelos clientes; Metáfora que auxilia os clientes na definição das *user stories*; e Cliente Presente uma vez que os responsáveis por escreverem um esboço dos requisitos são os próprios clientes. Os papéis propostos pelo XP que possuem relação com a atividade de levantamento inicial dos requisitos são o Cliente que é responsável pela escrita das *user stories* e o Gerente que é a pessoa responsável pela condução do Jogo do Planejamento, e também possui a tarefa de transmitir coragem, confiança e saber cobrar o que é de responsabilidade de cada um.

No ***Scrum***, a definição do esboço dos requisitos acontece durante a subfase de Planejamento durante a fase de *PreGame*, onde os requisitos conhecidos até o momento são listados dando origem ao *Product Backlog*. O documento contendo os requisitos que fazem parte do *Product Backlog*, agrupados de acordo com a sua prioridade, apresenta as seguintes informações: descrição do requisito, tempo estimado para o desenvolvimento e responsável pelo desenvolvimento. No *Scrum*, os papéis propostos para fazerem parte da atividade em questão são: O *Scrum Master* que assegura que o projeto está sendo desenvolvido de acordo com as práticas e regras do *Scrum*; o Dono do Produto que é a pessoa responsável pelo *Product Backlog,* ou seja, tem a responsabilidade de tomar qualquer decisão relacionada ao mesmo;. e o Cliente que deve participar ativamente na fase da elaboração dos itens que irão compor o *Product Backlog*.

No **FDD**, os requisitos já conhecidos são listados, definidos e documentados através de casos de uso ou *users stories* durante a fase Desenvolver um Modelo Global que consiste na primeira etapa do processo no FDD. A prática Desenvolvendo Através de Características sugere que além dos artefatos gerados para a documentação dos requisitos sejam construídos um diagrama de classe UML e diagrama(s) de seqüência UML com o objetivo de auxiliar na compreensão do projeto. O FDD sugere que os requisitos sejam representados através de uma lista hierárquica. Os papéis sugeridos pelo FDD relacionados à definição preliminar dos requisitos são o Gerente de Projeto que possui entre as suas responsabilidades a de gerenciar a viabilidade do projeto oferecendo todas as condições necessárias à equipe para o desenvolvimento do trabalho; o Arquiteto Principal é quem elabora o Modelo Global do *software* a ser desenvolvido;

o Especialista no Domínio que pode ser um usuário, analista de negócios, cliente ou qualquer pessoa que conheça bem o domínio do problema, tendo como tarefa a de informar as funcionalidades que deverão ser atendidas pelo *software*; e o Gerente de Domínio responsável por conduzir os Especialistas de Domínio a resolverem as diferenças de opinião relativas aos requisitos do sistema.

O **ASD** não apresenta nenhuma prática em relação à definição preliminar dos requisitos, mas durante a subfase Início do Projeto são realizadas sessões JAD com a presença dos representantes dos clientes, objetivando identificar os requisitos conhecidos até o momento. Entretanto o método não sugere nenhum documento para formalizar e organizar os requisitos definidos durante as sessões JAD. Como o método realiza sessões de JAD para elucidação dos requisitos, pode-se citar, além dos membros que participam destas sessões, o Patrocinador como a pessoa com responsabilidade global em relação ao produto que está sendo desenvolvido, o Facilitador que possui a responsabilidade de liderar e planejar as sessões e o Escriba responsável por fazer as anotações necessárias.

A **AM** sugere a criação do Documento de Requisitos para apresentar informações sobre os mesmos. Entre os artefatos propostos para fazerem parte do Documento de Requisitos estão: modelos contendo as regras de negócio, casos de uso, *user stories*, ou protótipos de interface essencial. Outro documento que poderá ser gerado ao final desta atividade é um documento chamado Visão Geral Executiva contendo uma definição da visão de sistema e um resumo da estimativa atual de custos, benefícios previstos, riscos e estimativas de pessoal. A AM sugere também a criação de um documento apresentando uma Visão Geral do Projeto apresentando um resumo de informações como a visão do sistema – caso não tenha sido gerado o documento Visão Geral Executiva -, principais contatos dos usuários, tecnologias e possíveis ferramentas usadas na construção do sistema.

Atividade – Atribuição dos Requisitos as Iterações			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	Equipe técnica e clientes definem as *user stories* que serão desenvolvidas na próxima iteração. As iterações duram de 1 a 4 semanas.	• Jogo do Planejamento • Cliente Presente	• Cliente • Gerente

Scrum	Definição do *Sprint Backlog. As Sprints* (iterações) duram no máximo 30 dias.	• *Product Backlog* • *Sprint Backlog* • Reunião de Planejamento da *Sprint*	• *Scrum Master* • Dono do Produto • *Scrum Team*
FDD	As características são agrupadas, priorizadas e distribuídas aos responsáveis pelo seu desenvolvimento. As iterações duram no máximo 2 semanas.	• Desenvolvendo através de características • Propriedade Individual da Classe	• Programador Chefe • Especialista no Domínio • Proprietário de Classe
ASD	Definição do número de iterações e quais os requisitos que serão desenvolvidos em cada uma delas. As iterações duram de 4 a 8 semanas.	• Prazos pré-fixados • Orientado a riscos	-
AM	Refinamento do documento: Visão Geral Executiva	-	-

Tabela 4. Atividade - Atribuição dos Requisitos as Iterações.

A atribuição dos requisitos às iterações no **XP** é feita na fase de Planejamento, durante a adoção da prática Jogo do Planejamento, onde acontece a definição das *user stories* que serão implementadas durante cada iteração. Esta atividade acontece toda a vez que uma nova iteração é iniciada, ou seja, esta atividade não é realizada uma única vez, e sim quantas vezes forem o número de iterações. A prática Cliente Presente é adotada, uma vez que o cliente participa da escolha das *user stories* que serão implementadas durante a iteração. O tempo de duração das iterações no XP devem durar de 1 a 4 semanas. Os papéis associados a esta atividade são o Cliente que define as prioridades para as *user stories*, influenciando diretamente na alocação dos requisitos as iterações, e o Gerente que participa e auxilia os clientes na definição do planejamento do desenvolvimento.

No ***Scrum*** os requisitos que serão desenvolvidos em cada *Sprint* são definidos a partir da lista de requisitos contida no *Product Backlog.* Durante a subfase de Planejamento na fase de *PreGame* acontece a escolha dos requisitos que serão desenvolvidos durante a *Sprint.* Estes requisitos são selecionados durante a Reunião de Planejamento da *Sprint* de acordo com a sua prioridade, dando origem ao *Sprint Backlog* que consiste em uma lista contendo os requisitos que serão desenvolvidos durante a *Sprint* (iteração). O tempo de duração das iterações no *Scrum* deve durar no máximo 30 dias. Os papéis relacionados a esta atividade são: O *Scrum Master,* uma vez que o mesmo deve assegurar que o projeto está sendo desenvolvido de acordo com as práticas e regras do *Scrum;* O Dono do Produto que toma as decisões finais relacionadas ao *Product Backlog*; libera e escolhe os itens que serão trabalhados no *Sprint Backlog*;

E o *Scrum Team*, que determina a criação do *Sprint Backlog,* calcula o esforço para o desenvolvimento de cada item, revisa os itens do *Product Backlog* e sugere o que necessita ser acrescentado ou removido.

No **FDD**, esta atividade é contemplada durante as fases "Construir uma Lista de Características", onde as características (requisitos) são agrupadas e priorizadas de acordo com a sua importância e dependência dando origem aos conjuntos de características que serão desenvolvidos durante as várias iterações. E a fase "Planejar a Construção por Características", onde é feito um planejamento do desenvolvimento destes conjuntos, bem como as características são distribuídas aos seus proprietários que serão os responsáveis pelas implementações. As práticas relacionadas a esta atividade são: Desenvolvendo Através de Características que consiste no agrupamento e planejamento do desenvolvimento das mesmas, e; Propriedade Individual da Classe que sugere que exista um responsável para cada característica ou conjunto de características. O tempo de duração das iterações no FDD tem uma duração máxima de 2 semanas. Os papéis sugeridos pelo FDD para a realização desta atividade são: O Programador Chefe que participa da análise dos requisitos, priorizando-os e estimando-os, juntamente com o Especialista no Domínio. E o Proprietário de Classe, que trabalha subordinado ao Programador Chefe, recebendo as características que ficarão sob a sua responsabilidade.

Durante a subfase de Planejamento do Ciclo Adaptável na fase de Especulação do **ASD**, são definidas a quantidade de iterações e quais os requisitos que serão implementados durante cada uma delas. A prática referente à Prazos pré-fixados é adotada para garantir que o tempo definido para cada iteração deve ser respeitado e cumprido. De acordo com a prática Orientado a Riscos, as características que representam algum risco para o bom andamento do projeto devem ser prioritárias em relação as outras. O tempo de duração das iterações devem durar de 4 a 8 semanas. O ASD não sugere explicitamente nenhum papel associado a esta atividade.

A **AM** por não apresentar um processo de desenvolvimento para produção do software, nada propõe em relação a atividade de alocação dos requisitos as iterações. Porém, pode-se acrescentar informações relacionadas as definições ocorridas durante esta atividade no documento sobre a Visão Geral Executiva gerado na atividade anterior.

Atividade – Projeto da Arquitetura do Sistema			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	Atividade realizada paralelamente a escrita das *user stories*.	• Projeto Simples	• Consultor
Scrum	Desenvolvimento de um projeto geral do sistema baseado nos itens atuais do *Product Backlog*	-	• *Scrum Master* • *Scrum Team*
FDD	Construção de Diag. de Classe Diag. de Seqüência da UML.	• Modelagem dos Objetos de Domínio	• Arquiteto Principal
ASD	-	-	-
AM	Geração de uma Documentação do Sistema contendo uma visão geral da arquitetura técnica e da arquitetura do negócio do sistema	-	-

Tabela 5. Atividade - Projeto da Arquitetura do Sistema.

O **XP** propõe que paralelamente à escrita das *user stories*, seja realizado o projeto da arquitetura do sistema. Apesar de não apresentar sugestões em relação a como este projeto deve ser feito, nem quais os artefatos devem ser gerados, o XP adota a prática Projeto Simples, que sugere que o projeto deverá ser o mais simples possível e refinado continuamente durante todo o processo de desenvolvimento. Como papel para auxiliar na execução desta atividade, o XP, sugere o Consultor, que é um membro externo à equipe e que possui o conhecimento técnico necessário e tem o objetivo de auxiliar a equipe na resolução de problemas específicos em relação a eventuais problemas que venham a surgir.

Na subfase de Arquitetura no *PreGame* do ***Scrum*** é feito um projeto geral do sistema baseado nos itens atuais do *Product Backlog*. Apesar de sugerir que seja realizado um projeto da arquitetura do sistema, o *Scrum* não sugere nenhuma técnica ou prática associada a esta atividade. Mesmo não apresentando diretrizes para a execução desta atividade, o *Scrum* coloca que o *Scrum Team*, em conjunto com o *Scrum Master*, são responsáveis por executar as ações necessárias para atingir os objetivos de cada *Sprint*, e o projeto da arquitetura do sistema exerce função essencial para que estes objetivos sejam atingidos.

O **FDD** sugere através da prática Modelagem dos Objetos de Domínio que seja construído um diagrama de classes da UML para representar a arquitetura do sistema

durante a fase Desenvolver um Modelo Global. Estes diagramas descrevem os objetos relevantes do sistema, bem como os seus relacionamentos. Para complementar o diagrama de classes também poderão ser gerados diagramas de seqüência da UML. A pessoa sugerida pelo FDD para realizar esta atividade é o Arquiteto Principal. É ele quem elabora o projeto geral do *software* a ser desenvolvido, tomando decisões finais em relação ao projeto técnico.

O método **ASD** não sugere nenhuma atividade, prática ou papel relacionado a atividade de projeto de arquitetura durante seu processo de desenvolvimento.

A **AM** sugere a geração de um documento chamado de Documentação do Sistema, que contempla, entre outras informações, uma visão geral da arquitetura técnica e da arquitetura do negócio do sistema. A arquitetura detalhada, também podem ser incluída neste documento.

Atividade – Desenvolver Incremento do Sistema			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	Implementação das *user stories* que fazem parte da iteração corrente. Características importantes: programação m dupla, *refactoring* constante e propriedade coletiva do código.	• Padrões de codificação • Programação em pares • Propriedade coletiva • *Refactoring* • Cliente presente • Entregas Freqüentes	• Programador • Cliente • Rastreador • Treinador • Consultor
Scrum	Implementação dos requisitos contemplados no *Sprint Backlog* para a *Sprint* corrente. Não apresenta processo de desenvolvimento pré-definido.	• *Sprint* • Reuniões Diárias do *Scrum*	• *Scrum Master* • *Scrum Team*
FDD	Análise da documentação existente, geração de Diag. de Seqüência da UML, refinamento do modelo gerado nas atividades anteriores e implementação das características que serão desenvolvidas durante a iteração corrente. Características importantes: Equipes responsáveis por grupos de características.	• Equipes de Características • Administração de Configuração • Construções Regulares	• Programador Chefe • Proprietário de Classe • Gerente de Versão • Especialista na Linguagem • *Toolsmith* • Administrador de Sistema
ASD	Implementação dos requisitos que fazem parte da iteração corrente. Não apresenta diretrizes concretas para esta atividade.	• Tolerância a Mudanças	• Desenvolvedores
AM	Geração de uma Documentação de Operações, contemplando as dependências em que o sistema está envolvido e diretrizes para a resolução de problemas. E um documento contendo as	-	-

Tabela 6. Atividade - Desenvolver Incremento do Sistema.

No **XP** a atividade de desenvolvimento dos requisitos durante as iterações é realizada durante a fase de Iterações para a Entrega, onde sugere que para cada conjunto de *user stories* selecionadas para fazerem parte da iteração sejam realizadas as atividades de modelagem e programação. Para dar suporte a realização destas atividades o XP propõe a adoção das práticas: Padrões de Codificação, que consiste na definição de padrões para a estruturação e geração do código com o objetivo de manter o mesmo consistente e fácil para possibilitar a compreensão por toda a equipe. O método coloca que não importa o padrão adotado e sim a existência de um; A prática de Programação em Pares, sugere que um dos programadores deve ser responsável por pensar na melhor forma de implementação e o outro analisa estrategicamente se a abordagem utilizada irá funcionar, se existe algum teste que ainda não foi realizado e se existe alguma forma para simplificar o código; A prática de Propriedade Coletiva coloca que em um projeto XP todos são responsáveis por todo o sistema e que qualquer um que perceba uma oportunidade de agregar valor a alguma parte do código deve fazê-lo, é proposto também a utilização de um sistema de controle de versões; A realização de *Refactoring* consiste na reestruturação do código sempre que possível, com o objetivo é fazer com que o mesmo fique mais reutilizável e compreensível; A prática Cliente Presente coloca que o mesmo deve estar sempre presente e disponível para fornecer detalhes do sistema quando surgirem dúvidas. Caso a presença do cliente não seja possível em tempo integral deve ser estabelecido um canal aberto de comunicação entre o mesmo e a equipe de desenvolvimento. E por fim, a prática Entregas Freqüentes coloca que ao término de cada iteração deverá ser feita a entrega das *user stories* desenvolvidas durante a iteração. Os papéis relacionados a atividade de desenvolvimento no XP são: O Programador que é responsável por implementar o sistema, trabalhar em par; realizar o *refactoring* sempre que necessário e interagir com o cliente para o esclarecimento de dúvidas em relação as *user stories*. O Cliente que deve estar sempre disponível para a elucidação de questões que por ventura surgirem. O Rastreador, responsável por medir o progresso do desenvolvimento 1 ou 2 vezes por semana e manter todos informados do que está acontecendo. O Treinador que é responsável pelo processo de desenvolvimento

como um todo, notifica as pessoas quando elas estão se desviando do processo e conduz a equipe novamente para o mesmo. E o Consultor, membro externo que possui o conhecimento técnico necessário e é responsável por auxiliar a equipe na resolução de problemas específicos.

Esta atividade acontece na fase de Desenvolvimento do ***Scrum***, durante a *Sprint*, onde são implementados os requisitos contemplados no *Sprint Backlog*. Schwaber (1995) ressalta que o desenvolvimento dos itens dentro de uma *Sprint* não segue nenhum processo pré-definido, ou seja, o desenvolvimento dos requisitos pode ser realizado de acordo com qualquer processo existente para este fim. Segundo Abrahamsson (2002), uma sugestão seria a combinação de outros métodos ágeis que possuem processos definidos nesta fase. Durante cada uma das *Sprints* acontecem as Reuniões Diárias do *Scrum* que devem durar de 15 a 30 minutos e têm como objetivo que todos os envolvidos se mantenham informados sobre o progresso e as dificuldades encontradas. Apesar de não seguir processos pré-definidos, o *Scrum* sugere dois papéis relacionados as práticas adotadas durante a realização desta atividades: O *Scrum Master* assegura que o projeto está sendo desenvolvido de acordo com as práticas e regras do *Scrum* e organiza as reuniões diárias. E o *Scrum Team*, que em conjunto com o *Scrum Master*, executa as ações necessárias para atingir os objetivos de cada *Sprint* e desenvolve os itens do *Sprint Backlog*.

No **FDD** o desenvolvimento dos conjuntos de características durante as iterações acontece durante as fases Projetar e Construir cada Característica, onde acontecem as atividades de análise da documentação existente, geração de diagramas de seqüência para o conjunto de características, refinamento do modelo gerado durante as atividades de definição dos requisitos e projeto da arquitetura do sistema e por fim, da implementação das classes e métodos correspondentes às características que serão desenvolvidas durante cada um das iterações. A prática Equipes de Características sugere que as atividades citadas anteriormente fiquem sob a responsabilidade das equipes formadas pelos desenvolvedores. As equipes devem ter de 3 a 6 membros e cada membro da equipe analisa, projeta e implementa uma característica ou um conjunto de características sob a orientação de um desenvolvedor experiente. A prática Administração de Configuração sugere a utilização de um sistema de controle de versões para datar e manter um histórico das alterações feitas. Todo artefato utilizado

durante o desenvolvimento é candidato ao controle de versão. É sugerido também solicitar uma autenticação dos clientes e gerentes de projeto para cada modificação, dando uma certa segurança ao contrato comercial. A prática Construções Regulares coloca que as características desenvolvidas durante a iteração devem ser apresentadas e/ou entregues ao cliente. Os papéis sugeridos pelo FDD para a realização desta atividade são: O Programador Chefe que é geralmente o programador com maior experiência dentro da equipe. O Proprietário de Classe, papel subordinado ao Programador Chefe, é o responsável pelo desenvolvimento das classes atribuídas a ele. O Gerente de Versão é responsável por controlar o progresso no desenvolvimento através de constantes revisões em conjunto com o Programador Chefe. O Especialista na Linguagem é o membro da equipe responsável por possuir um conhecimento completo de uma linguagem de programação específica ou tecnologia. Este papel é particularmente importante quando a equipe do projeto resolve utilizar uma nova tecnologia. O *Toolsmith* é responsável por construir ferramentas de suporte para o desenvolvimento. E por fim, o Administrador de Sistema possui a tarefa de configurar, administrar e diagnosticar os servidores, estações de trabalho e desenvolvimento e testar os ambientes usados pela equipe do projeto.

Durante a fase de Colaboração do **ASD**, os requisitos que fazem parte de cada uma das iterações são desenvolvidos. É importante salientar que o método coloca que o desenvolvimento das diferentes iterações pode acontecer simultaneamente e que o enfoque do ASD é no resultado final e suas qualidades, ou seja, na qualidade do sistema desenvolvido em cada iteração. O método não sugere tarefas ou processos a serem utilizados para produzirem estes resultados. A prática relacionada ao desenvolvimento é basicamente a Tolerância a Mudanças que coloca que as mesmas são freqüentes e é sempre melhor estar pronto a adaptá-las do que controlá-las, fazendo uma constante avaliação de quais componentes podem mudar. O ASD não oferece muitos detalhes em relação a atividade de desenvolvimento, o método coloca apenas que a equipe pode ser formada por Desenvolvedores, que seriam os responsáveis diretos pela implementação dos requisitos durante as iterações.

A **AM** sugere alguns documentos a serem gerados durante e ao final desta atividade: A Documentação de operações inclui uma indicação das dependências com as quais o sistema está envolvido; a natureza de sua interação com outros sistemas,

banco de dados e arquivos e diretrizes de resolução de problemas. Um documento contendo as Decisões de Projeto contemplando as decisões críticas relacionadas ao projeto e à arquitetura que a equipe tomou durante o desenvolvimento

Atividade – Validar Incremento			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	Os programadores executam os testes de unidade e os clientes executam os testes de aceitação. Consideração importante: ambos os testes são escritos antes da atividade de implementação.	• Testes • Cliente presente	• Programador • Cliente • Testador
Scrum	As validações são realizadas ao final de cada *Sprint*. O *Scrum* não adota nenhum processo de validação pré-definindo.	• *Sprint*	• *Scrum Team*
FDD	Os testes e inspeções são executados pelos próprios programadores após a implementação.	• Inspeções	• Testadores • Proprietário de Classe
ASD	Grupos de clientes revisam e testam o incremento. Um par de programadores é responsável por revisar e avaliar o código.	• Tolerância a Mudanças	-
AM	-	-	-

Tabela 7. Atividade - Validar Incremento.

No **XP** a validação dos requisitos desenvolvidos durante a iteração ocorre na fase de Iterações para a Entrega, através da execução dos testes de aceitação e dos testes de unidade. A prática Testes é adotada e sugere que os programadores escrevam os testes de unidade e os clientes os testes de aceitação. Ambos são escritos anteriormente ao processo de codificação e só após são executados. Os clientes escrevem textualmente os testes de aceitação e executam os mesmos no incremento resultante da iteração, recomenda-se que, para cada *user story* exista no mínimo um teste de aceitação. A prática Cliente Presente coloca que o mesmo deve estar sempre presente e disponível para escrever e realizar os testes de aceitação. Além dos Programadores e Clientes, um outro papel sugerido pelo XP para realizar esta atividade é o Testador, que possui a tarefa de auxiliar os clientes a escrever os testes aceitação, executar os testes que forem solicitados e publicar os resultados para a equipe.

O ***Scrum*** sugere que a atividade de validação dos itens do *Sprint Backlog* implementados, seja realizada no final da *Sprint*, durante a fase de Desenvolvimento.

Igualmente a atividade de desenvolvimento, o *Scrum* não adota nenhum processo pré-definido para a atividade de validação dos requisitos implementados. Sendo assim, pode-se utilizar o processo de validação de outros métodos disponíveis. O papel relacionado a esta atividade é basicamente do *Scrum Team*, por ser responsável por executar as ações necessárias para que o objetivo de cada *Sprint* seja atingido, incluindo os testes e validações.

No **FDD** os testes acontecem ao final de cada iteração na fase "Construir cada Característica", onde o conjunto de características implementado durante a mesma é testado através de testes de unidade pelos proprietários de classe – os programadores -. O FDD adota a prática de Inspeções que sugere que o código gerado deve ser inspecionado para assegurar a qualidade do projeto e do código. O objetivo preliminar das inspeções é a detecção de defeitos e verificar se o código foi escrito de maneira compreensível. Além dos Proprietários de Classe, outro papel sugerido pelo FDD para esta atividade são os Testadores, que verificam se o sistema que está sendo produzido satisfará os requisitos do cliente, esta, pode ser uma equipe independente ou parte da equipe de projeto.

O **ASD** não adota nenhuma prática relacionada a validação dos requisitos, mas coloca que a validação dos requisitos desenvolvidos durante a fase de Colaboração acontece na subfase Revisões de Qualidade na fase de Aprendizado, onde a qualidade funcional do sistema gerado na iteração é verificada através da definição de grupos de clientes para revisar e testar a aplicação. Também a qualidade técnica é verificada, onde um par de programadores é responsável por revisar e avaliar o código do sistema. Vale salientar que apesar de atribuir responsabilidades de testes, revisões e avaliações o ASD não sugere explicitamente nenhum papel para a execução das mesmas.

A **AM** não sugere a geração de documentos relacionados a atividade de testes e validações

Atividade – Integrar Incremento			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	A integração acontece paralelamente ao desenvolvimento das *user stories*.	• Integração contínua	• Programador

Scrum	Atividade realizada ao final de cada *Sprint*. Não é proposta nenhuma diretriz para a realização desta atividade.	-	-
FDD	Atividade realizada após os testes no incremento.	-	-
ASD	-	-	-
AM	-	-	-

Tabela 8. Atividade – Integrar Iincremento.

No **XP**, a atividade de integração acontece ao longo da fase de Iterações para a Entrega através da prática Integração Contínua, que sugere que as equipes devem manter o sistema integrado todo o tempo, ou seja, à medida que o código vai sendo gerado ele vai sendo integrado. O papel relacionado a esta atividade é do Programador, onde cada par de programadores é responsável por integrar seu próprio código. Isto deve ser feito após os testes de unidade terem sido executados com 100% de sucesso em cada requisito implementado. A integração contínua evita ou descobre problemas de compatibilidade cedo.

A integração do resultado das várias *Sprint*s acontece ao final de cada uma delas na fase de Desenvolvimento. O ***Scrum*** não sugere nenhuma prática em relação a atividade de integração, nem tampouco sugere como esta atividade deve ser realizada, ou os papéis envolvidos.

No **FDD** a integração do conjunto de características implementado durante a iteração corrente com os outros, acontece ao final de cada iteração durante a fase "Construir cada Característica", após os testes. O método não apresenta maiores detalhes em relação a como esta atividade deve ser conduzida e também não relaciona nenhuma prática ou papel a ela.

O **ASD** não sugere nenhuma atividade específica de integração dos sistemas desenvolvidos em cada iteração durante o seu processo de desenvolvimento.

A **AM** não sugere a geração de documentos relacionados a atividade de integração dos incrementos resultantes de cada iteração.

Atividade – Validar Sistema			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	O sistema disponibilizado ao cliente para que o mesmo realize validações no sistema como um todo.	-	-
Scrum	O cliente valida o sistema integrado em uma reunião no último dia da *Sprint.*	• *Sprint* • Revisão da *Sprint*	• *Scrum Master* • Cliente
FDD	Esta atividade ocorre através das inspeções e dos testes de integração realizados pelos Testadores e Proprietários de Classe	-	• Testadores • Proprietário de Classe
ASD	-	-	-
AM	-	-	-

Tabela 9. Atividade - Validar Sistema.

O **XP** sugere que as integrações devem acontecer paralelamente às implementações das *user stories* e conseqüentemente aos testes, porém, o mesmo não sugere práticas, papéis, nem atividades específicas de validação após os incrementos integrados. Entretanto ele recomenda que durante as fases de Produção o resultado da iteração pode ser posto em operação dentro da organização, onde será avaliado e poderá sofrer alteração de acordo com as considerações levantadas pelos clientes.

O ***Scrum*** valida a integração dos incrementos, ao final de cada *Sprint* durante a fase de Desenvolvimento através da prática Revisão da *Sprint*. No último dia de cada *Sprint*, é apresentado o incremento para o Cliente durante uma reunião que é conduzida pelo *Scrum Master*. Os participantes avaliam o incremento e decidem sobre as atividades seguintes. Após a entrega do incremento ao cliente na fase de *PostGame*, o *Scrum* sugere que seja realizado algum tipo de acompanhamento em relação a utilização do mesmo, onde mais são executados mais testes com o objetivo de garantir a qualidade do mesmo. Conforme colocado anteriormente, os papeis que participam da revisão da *Sprint* são o Cliente e o *Scrum Master.*

Após a integração dos conjuntos de características, o **FDD** valida todos os incrementos gerados na fase "Construir cada Característica", através das inspeções e dos testes de integração realizados pelos Testadores e Proprietários de Classe. O método não sugere nenhuma prática em relação a validação dos incrementos integrados.

O **ASD** não sugere nenhuma atividade específica de validação do sistema como um todo durante o seu processo de desenvolvimento.

A **AM** não sugere a geração de documentos relacionados a atividade de validação dos sistema.

Atividade – Entrega Final			
Método	**Especificação da Atividade**	**Práticas**	**Papéis**
XP	O cliente deve estar satisfeito com o sistema e não ter mais nada a acrescentar em relação a funcionalidades	-	-
Scrum	Não devem existir mais itens no *Product Backlog* a serem desenvolvidos	-	-
FDD	O sistema é entregue após todos os conjuntos de características passarem pelas etapas de projeto e construção.	-	-
ASD	Atividade realizada após não existirem mais requisitos a serem desenvolvidos	-	-
AM	Refinamento da Documentação de Operações e geração de Documentos de Suporte e da Documentação de Usuário.	-	-

Tabela 10. Atividade - Entrega Final.

O **XP** não apresenta práticas ou papéis em relação a esta atividade, mas coloca que a entrega final do sistema acontece durante a fase de Fim do Projeto, e para que isso ocorra, o cliente deve estar satisfeito com o sistema e não ter mais nada a acrescentar em relação a funcionalidades. O XP sugere que seja oferecido um suporte ao cliente com o objetivo de solucionar eventuais dúvidas e problemas decorrentes do uso do *software* entregue.

No ***Scrum***, o sistema é entregue durante a fase de *PostGame*. Nesta fase, não existem mais itens no *Product Backlog* a serem desenvolvidos. O *Scrum* não sugere nenhuma prática ou papel em relação a entrega final do sistema.

O **FDD** não possui uma fase específica e nem apresenta práticas ou papel em relação à entrega final do sistema, portanto o método sugere que todos os conjuntos de características devem passar pelas etapas de projeto e construção até que o sistema esteja pronto para ser entregue.

A atividade de entrega do sistema final no **ASD** é feita durante a subfase de Entrega Final na fase de Aprendizado. Esta atividade não apresenta práticas ou papéis relacionados, e só é realizada após não existirem mais requisitos a serem desenvolvidos.

A **AM** sugere que a Documentação de Operações gerada na atividade de Desenvolvimento seja refinada com informações a respeito de referências e procedimentos de *backup*. Outro documento proposto é o Documento de Suporte que inclui materiais de treinamento específicos para o pessoal de suporte, toda a documentação de usuário para usar como referência na resolução de problemas, um guia para a solução de problemas, procedimentos para lidar com problemas difíceis e uma lista de pontos de contato dentro da equipe de manutenção. E por fim a geração da Documentação de Usuário que contempla, um manual de referência, um guia de uso, um guia de suporte e até mesmo materiais para treinamento.

Como pôde ser observada, a análise realizada possibilitou a identificação das práticas e dos papéis relacionados a cada uma das atividades propostas pelos métodos ágeis, além de permitir a identificação dos seus pontos comuns. No próximo capítulo, as atividades identificadas serão agrupadas de acordo com as suas semelhanças e/ou dependências de modo a dar origem a um *framework* que possa ser utilizado de acordo com as necessidades da equipe técnica envolvida no projeto.

4 *FRAMEWORK* PARA OS MÉTODOS ÁGEIS.

O objetivo principal da proposta de um *framework* para os métodos ágeis é prover uma estrutura que, apresente de forma organizada e consistente, as atividades sugeridas pelos métodos ágeis *Extreme Programming* (XP), *Scrum*, *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD) e *Agile Modeling* (AM), no decorrer dos seus processos de desenvolvimento.

E da mesma forma possibilitar que estas atividades sejam executadas conforme as reais necessidades dos *stakeholders,* fazendo com que os mesmos não necessitem se adaptar as atividades propostas e sim que as atividades sejam adaptadas ao processo de desenvolvimento com o qual a equipe de desenvolvimento está acostumada.

4.1 Estrutura do *Framework*

Para que o *framework* proposto não apresente atividades redundantes, as atividades propostas foram agrupadas de acordo com as semelhanças – quando existirem - identificadas no capítulo anterior. Quando não for possível o agrupamento, serão apresentadas as diferentes atividades para que as mesmas sejam realizadas de acordo com a necessidade. É importante salientar que as práticas e os papéis deverão ser utilizados juntamente com a atividade utilizada, de maneira a não comprometer o sucesso da sua aplicação.

De forma a possibilitar uma melhor compreensão do *framework*, o mesmo será apresentado da seguinte forma:

- nome e descrição da atividade;
- quais as práticas e papéis incorporados à mesma e;
- sua(s) origem(s).

É importante salientar que a equipe interessada em utilizar o *framework* proposto tenha total conhecimento de todas as atividades sugeridas, para somente depois "escolher" quais delas farão parte do seu processo de desenvolvimento. Desta forma a equipe não correrá o risco de utilizar uma atividade em um primeiro momento do

processo e em um segundo momento descobrir que a atividade realizada poderia ser substituída por outra com melhores resultados. A conseqüência deste erro pode comprometer o sucesso da utilização do *framework* e do projeto como um todo.

As atividades apresentadas a seguir obedecem à estrutura sugerida pelo Desenvolvimento Incremental, uma vez que o mesmo serviu como base para a análise realizada no capítulo anterior, e servirá também como base para as atividades sugeridas pelo *framework*. Sendo assim, sugere-se que o processo de desenvolvimento do Desenvolvimento Incremental (Figura 10) seja seguido.

4.1.1 Atividades de Definição dos Requisitos

- **User Stories –** Os clientes escrevem as *user stories* que apresentam informações a respeito das funcionalidades do sistema bem como a sua prioridade de desenvolvimento. Segundo Beck (2000), para cada funcionalidade deverá ser escrita uma *user story*. Para que esta atividade seja realizada com sucesso é necessária a adoção da prática Cliente Presente e os papéis da equipe relacionados à mesma são o Cliente e a Equipe de Desenvolvimento, que tem a tarefa de auxiliar os clientes na escrita das *user stories*. Origem da atividade: XP.
- **Lista de Requisitos –** Geração de um documento contendo uma descrição sucinta dos requisitos que farão parte do sistema – estes poderão ser detalhados utilizando qualquer outro artefato que a equipe julgar conveniente - , agrupados de acordo com a sua prioridade, apresentando as seguintes informações: descrição do requisito, tempo estimado para o desenvolvimento e responsável pelo desenvolvimento. Um exemplo deste documento pode ser observado na Figura 12. Este documento é gerado com a participação do Cliente e da Equipe de Desenvolvimento, portanto é importante a adoção da prática Cliente Presente. Os papéis envolvidos são o Cliente e a Equipe de Desenvolvimento. Origem da atividade: *Scrum* e FDD.

Lista de Requisitos

Prioridade	Item	Descrição	Tempo Estimado	Responsável
Muito Alta				
	1	Conexão com o Banco de Dados	40	Ana
	2	...	...	...
	3	Cadastro de Usuários	...	João
Alta				
	4			
	5	..		
	6			
Média				
	7	Impressão de Relatórios		
	8	...	...	...
	9			

Figura 11. Lista de Requisitos

- **Documentação Detalhada dos Requisitos –** As *user stories* podem servir para a documentação dos requisitos do sistema, mas caso a equipe de desenvolvimento não opte pela geração das mesmas, ou mesmo decida que exista a necessidade de outros artefatos para este fim, poderão ser desenvolvidos Casos de Uso e/ou Diagramas de Classe da UML, ou qualquer outro artefato utilizado para documentar os requisitos. Os papéis envolvidos são basicamente a Equipe de Desenvolvimento, que deve possuir conhecimento sobre o artefato gerado, e o Cliente ou um Especialista no Domínio, que auxiliará a equipe na compreensão do problema. Origem da atividade: FDD.

- **Organização da Documentação -** Para que a documentação gerada se mantenha organizada, podem ser criados: um Documento de Requisitos contemplando os artefatos gerados, um documento chamado Visão Geral Executiva contendo uma definição da visão de sistema e um resumo da estimativa atual de custos, benefícios previstos, riscos e estimativas de pessoal, e um documento contendo uma Visão Geral do Projeto apresentando um resumo de informações como a visão do sistema – caso não tenha sido gerado o documento Visão Geral Executiva -, principais contatos dos usuários, tecnologias e possíveis ferramentas usadas na construção do sistema. Não existe um responsável direto pela realização desta atividade, sendo assim ela

poderá ser executada por qualquer membro da Equipe de Desenvolvimento. Origem da atividade: AM.

4.1.2 Atividades de Atribuição dos Requisitos às Iterações

- **Planejamento das Iterações –** Assim que os requisitos forem sendo definidos pode-se começar a atividade de planejamento do desenvolvimento que consiste na distribuição dos requisitos às iterações de acordo com as prioridades dos requisitos, suas dependência e seus riscos. Sendo assim, os requisitos já devem estar agrupados e priorizados. Sugere-se que no início de cada iteração seja feita uma reunião – ou mais, se a equipe julgar necessário - com o objetivo de realizar este planejamento, neste caso a prática a ser adotada é das Reuniões de Planejamento da Iteração. O cliente poderá participar do planejamento do desenvolvimento, e neste caso é essencial a adoção da prática Cliente Presente. Os papéis associados a esta atividade são a Equipe de Desenvolvimento e o Cliente, caso o mesmo participe do planejamento. Origem da atividade: XP, *Scrum*, FDD e ASD.

- **Duração das Iterações –** Um dos princípios da abordagem ágil é: "A prioridade é satisfazer ao cliente através de entregas de *software* de valor contínuas e freqüentes". Para que isso seja possível, as iterações não devem durar muito tempo e, sempre no seu final, deverá ser disponibilizada uma parte do sistema ao cliente que poderá ou não ser colocada em operação. O tempo de duração das iterações proposto pelos métodos ágeis que deram origem a este *framework* varia de 1 a 8 semanas. Sendo assim, a equipe de desenvolvimento deverá escolher o tempo de duração das iterações de acordo com a suas necessidades. Vale salientar que, para não comprometer a agilidade das entregas, o tempo não deverá ultrapassar o limite máximo que é de 8 semanas. As práticas relacionadas a esta atividade são basicamente as Entregas Freqüentes e Contínuas e Prazos Pré-Fixados e os envolvido diretos são os membros da Equipe de Desenvolvimento e o Gerente do Projeto. Origem da atividade: XP, *Scrum,* FDD e ASD.

- **Distribuição dos Requisitos aos Responsáveis** – Esta atividade consiste na distribuição dos requisitos aos responsáveis pelo seu desenvolvimento, considerando que a total responsabilidade do desenvolvimento de um requisito ou conjunto de requisitos é de um programador ou uma equipe de programadores. Sugere-se que no caso das equipes de programadores, as mesmas tenham no máximo, 6 membros e um responsável geral. A prática adotada durante esta atividade é a Propriedade Individual da Classe e os papéis são os Programadores e os Responsáveis pelas Equipes de Programadores. Origem da atividade: FDD.

- **Atualização da Documentação –** Pode-se acrescentar as definições ocorridas durante esta atividade nos documentos gerados na atividade de Definição dos Requisitos, contemplando as informações a respeito da iteração a qual eles pertencem, bem como atualizar o documento Visão Geral Executiva com informações a respeito dos custos estimados e de pessoal, caso necessário. *Origem da atividade:* AM.

4.1.3 Atividades de Projeto da Arquitetura do Sistema

- **Projeto Geral do Sistema –** Esta atividade consiste na realização de um projeto geral do sistema baseado nos requisitos conhecidos até o momento. Pode ser realizado paralelamente à atividade de Definição dos Requisitos e deve ser refinado continuamente durante todo o processo de desenvolvimento. Sugere-se a adoção da prática Projeto Simples que coloca que um projeto simples facilita a compreensão, bem como a sua manutenção. Como possíveis artefatos para representar o projeto da arquitetura do sistema propõem-se: Diagramas de Classe e Diagramas de Seqüência da UML, ou qualquer outro artefato com este propósito, ficando a critério da equipe de desenvolvimento escolher o que melhor lhe convir. O importante é não desperdiçar tempo no aprendizado de novos artefatos, se os que já são de conhecimento da equipe suprem tais necessidade. Os papéis relacionados a esta atividade são a Equipe de Desenvolvimento e o Gerente do Projeto. Origem da atividade: XP, *Scrum*, FDD e AM.

- **Projeto Detalhado do Sistema –** Caso a equipe julgue necessário, pode-se detalhar o projeto do sistema considerando cada um dos conjuntos de requisitos que serão desenvolvidos durante as iterações. Sendo assim, sugere-se que no início de cada iteração seja realizado este projeto. Os artefatos gerados podem ser os mesmos utilizados para a realização do Projeto Geral do Sistema. Também durante esta atividade a prática do Projeto Simples deve ser adotada. A decisão da realização desta atividade ou não deve vir da Equipe de Desenvolvimento – que será a responsável pela sua execução – e do Gerente do Projeto. Origem da atividade: FDD e AM.
- **Documentação do(s) Projeto(s) Gerado(s) –** Para formalizar e refinar os projetos gerados, sugere-se a geração de um documento chamado de Documentação do Sistema, que contempla, além dos artefatos gerados nas outras atividades, uma visão geral do projeto da arquitetura técnica e da arquitetura do negócio do sistema. O projeto da arquitetura detalhada também pode ser incluído neste documento. Origem da atividade: AM.

4.1.4 Atividades de Desenvolvimento do Incremento do Sistema

- **Implementação dos Requisitos Durante Cada Iteração –** Esta atividade consiste basicamente na geração do código referente aos requisitos que fazem parte da iteração corrente. Para uma melhor organização, sugere-se a adoção da prática Padrões de Codificação, que consiste na definição de padrões para a estruturação e geração do código com o objetivo de manter o mesmo consistente para facilitar a compreensão e a manutenção, bem como a utilização de um sistema de controle de versões. Os papéis envolvidos nesta atividade são os programadores. Origem da atividade: XP, *Scrum*, FDD e ASD.
- **Escrita dos Testes de Unidade e Aceitação –** Esta atividade coloca que os testes de unidade e os testes de aceitação poderão ser escritos anteriormente ou paralelamente à atividade de implementação, para somente depois da mesma serem executados. A escrita dos testes de unidade antes da implementação tem como objetivo, obrigar os programadores a analisar melhor o que efetivamente

será codificado. Os testes de aceitação poderão ser escritos pelos clientes ou pelos próprios programadores e a pergunta chave é: "O que precisaria ser conferido para se ter certeza de que um determinado requisito está OK?". Caso o cliente efetue a escrita dos testes de aceitação, a prática Cliente Presente deverá ser adotada. Os papéis relacionados a esta atividade são os Programadores e os Clientes, se for o caso. Origem da atividade: XP.

- **Desenvolvimento Coletivo de Código –** Neste caso não existem responsáveis diretos pelo desenvolvimento de um requisito ou conjuntos deles. Sendo assim, é importante que toda a equipe de desenvolvimento tenha pleno conhecimento e compreensão do que será desenvolvido. A prática relacionada a esta atividade é a Propriedade Coletiva que coloca que todos são responsáveis por todo o sistema e que qualquer um que perceba uma oportunidade de agregar valor a alguma parte do código deve fazê-lo. Os papéis envolvidos são basicamente os programadores. *Origem da atividade:* XP.

- **Desenvolvimento em Duplas –** Esta atividade sugere que o código seja gerado por duplas de programadores, onde um dos programadores deve ser responsável por pensar na melhor forma de implementar o método corrente e o outro analisa estrategicamente se a abordagem utilizada irá funcionar, se existe algum teste que ainda não foi trabalhado e/ou se existe alguma forma de deixar o código mais simples. A prática associada a esta atividade é a Programação em Pares. Segundo Beck (2000), o código gerado por duplas de programadores está menos suscetível a erros, portanto o Gerente do Projeto deverá avaliar se esta atividade é viável, devido principalmente ao aumento dos custos. Origem da atividade: XP.

- **Realização de *Refactoring* –** Esta atividade sugere a reconstrução do código sempre que necessário. O objetivo é fazer com que o mesmo fique mais reutilizável e compreensível. Deve-se ficar atento ao desperdício de tempo, caso o *refactoring* não seja realmente necessário. Os envolvidos nesta atividade são os Programadores e a prática adotada é o *Refactoring*. Origem da atividade: XP.

- **Realização de Reuniões Diárias –** Esta atividade sugere a realização de reuniões diárias com a finalidade de que todos os envolvidos se mantenham informados sobre o progresso e as dificuldades encontradas. Os questionamentos que a equipe deve se fazer são: O que foi finalizado desde a última reunião?; Quais as dificuldades encontradas durante o trabalho?; Quais atividades pretende-se realizar até a próxima reunião?. Estas reuniões devem durar de 15 a 30 minutos e serem realizadas com os participantes em pé, justamente para que os mesmos não se sintam à vontade para ultrapassar o tempo estipulado. Os participantes das reuniões diárias são os Programadores e o Gerente do Projeto e a prática adotada é a chamada Reuniões Diárias. Origem da atividade: XP e *Scrum.*

- **Desenvolvimento Simultâneo –** Caso a equipe esteja apta e tenha condições de executar esta atividade sem comprometer o andamento do processo de desenvolvimento, o desenvolvimento dos conjuntos de requisitos que fazem parte de diferentes iterações poderá acontecer simultaneamente. Para isto acontecer deverá ser levado em consideração o número de programadores disponíveis. Esta decisão deverá ser tomada pela Equipe de Desenvolvimento em conjunto com o Gerente do Projeto. Origem da atividade: ASD.

- **Integração Paralela ao Desenvolvimento –** A atividade de integração do código gerado com os outros já implementados poderá ser realizada após o desenvolvimento do incremento, após a validação do mesmo – como será abordado a seguir – ou paralelamente à geração do código. Caso a equipe opte pela última alternativa, a integração deverá ocorrer diariamente. Segundo Beck (2000), a integração diária evita ou descobre problemas de compatibilidade mais cedo. A prática relacionada a esta atividade é a Integração Contínua e os responsáveis pela mesma são os Programadores. Origem da atividade: XP.

- **Documentação do Desenvolvimento** - Alguns documentos podem ser gerados durante e ao final desta atividade, entre eles: A Documentação de Operações, que inclui uma indicação das dependências entre os subsistemas. E um documento contendo as Decisões de Projeto, contemplando as decisões críticas relacionadas ao projeto e à arquitetura, que a equipe tomou durante o

desenvolvimento. Outra sugestão seria a inserção de comentários no código, esta atividade auxilia de forma significativa a compreensão do mesmo. Origem da atividade: AM.

4.1.5 Atividades de Validação do Incremento

- **Integração do Incremento Antes da Validação –** Caso a atividade de validação dependa diretamente da integração do incremento com outro(s), a integração deverá ser realizada antes da atividade de validação. Os responsáveis por esta atividade são os Programadores. Origem da atividade: XP.

- **Execução dos Testes de Unidade e Testes de Aceitação –** Após a implementação são executados os testes de unidade e os testes de aceitação. Se os mesmos já tiverem sido escritos anteriormente à codificação, a tarefa é somente a de executá-los, caso contrário, eles deverão ser escritos (item 4.1.4.2) para posteriormente serem executados no incremento resultante. Propõe-se que os programadores fiquem com a responsabilidade de executar os testes de unidade. Em relação aos testes de aceitação, os mesmos poderão ser executados pelos clientes ou por uma pessoa que não tenha envolvimento com a programação, a fim de evitar sugestionamentos. Após a execução dos testes, os problemas encontrados devem ser solucionados e novos testes realizados a fim de confirmar se o incremento realmente apresenta qualidade e se encontra de acordo com as necessidades dos clientes. Caso aconteçam alterações ou inclusões significativas nos requisitos durante os testes de aceitação, poderá existir a necessidade de algumas atividades serem refeitas. As práticas adotadas durante esta atividade são Cliente Presentes, caso o mesmo seja responsável pela execução dos testes de aceitação e a prática dos Testes. Os envolvidos são os Clientes, os Testadores, caso o cliente não execute os testes de aceitação, e os Programadores. Origem da atividade: XP, FDD e ASD.

- **Inspeção de Código –** Esta atividade tem como objetivo principal, a detecção de defeitos e verificar se o código foi escrito de maneira compreensível. Para a

realização desta atividade sugere-se que os programadores inspecionem os códigos uns dos outros, ou seja, o programador nunca inspecionará o seu próprio código, apenas o do seu colega. É importante salientar que as inspeções não devem ser realizadas com o objetivo de prejudicar o programador que gerou determinado código e sim como uma grande oportunidade de aprendizado e troca de informações. A prática relacionada com esta atividade é a Inspeção e os envolvidos com a mesma são basicamente os Programadores. *Origem da atividade:* FDD e ASD.

4.1.6 Atividade de Integração do Incremento

- **Integração do Incremento Resultante –** Esta atividade só deverá ser realizada caso a equipe de desenvolvimento não tenha optado por realizar a atividade de integração paralela ao desenvolvimento ou não tenha sido necessário integrar o incremento para que seja possível a realização das validações. Neste caso, esta atividade deverá ocorrer no final da iteração e depois de todas as validações e inspeções – se for o caso - terem sido executadas com sucesso. Os envolvidos na integração do incremento resultando são basicamente os Programadores. Origem da atividade: *Scrum* e FDD.

4.1.7 Atividades de Validação do Sistema

- **Reunião de Revisão da Iteração –** Esta atividade sugere que seja realizada uma reunião ao final da iteração, onde é feita uma avaliação da iteração pela equipe de desenvolvimento, são identificados os problemas e quais as soluções encontradas para os mesmos. Em um primeiro momento os participantes da reunião de revisão são o Gerente do Projeto e a Equipe de Desenvolvimento. Em um segundo momento são convocados os Clientes para que seja apresentado o incremento e sejam realizados testes de integração. A prática relacionada a esta atividade é a Reunião de Revisão da Iteração. Origem da atividade: *Scrum* e FDD.

- **Colocar Sistema em Operação –** Esta atividade sugere que o sistema seja validado como um todo dentro do ambiente em que será utilizado pelo cliente. Ela é recomendada caso o cliente solicite ou caso a equipe queira verificar como o sistema se comportará no real ambiente em que será utilizado. É importante que exista um acompanhamento pela equipe de desenvolvimento junto ao cliente e que o mesmo tenha conhecimento que o sistema ainda se encontra na fase de validações, ao contrário corre-se o risco do cliente acreditar que o sistema já está concluído, quando na verdade ele ainda necessita de testes como um todo. Os principais envolvidos nesta atividade são os Clientes, o Gerente do Projeto e os Suportes ao Cliente que podem ser os próprios Programadores. *Origem da atividade*: XP e *Scrum.*

4.1.8 Atividades de Entrega Final

- **Entrega do Sistema ao Cliente** – Esta atividade só deverá ser executada após não existirem mais requisitos a serem desenvolvidos e o cliente estar totalmente satisfeito com o sistema gerado. Pode ser realizada uma reunião para oficializar a entrega final com todos os envolvido no projeto. Sugere-se que seja oferecido um suporte ao cliente com o objetivo de solucionar eventuais dúvidas e problemas decorrentes do uso do *software* entregue. Os participantes desta atividade são todos os envolvidos no projeto. Origem da atividade: XP, *Scrum,* FDD e ASD.
- **Geração de uma Breve Documentação –** Caso a equipe de desenvolvimento tenha optado por não gerar uma documentação detalhada ao longo do processo de desenvolvimento, sugere-se que após a entrega final do sistema ao cliente seja gerado um mínimo de documentação sobre o mesmo. A idéia é escrever algumas páginas (de 5 a 10 páginas) sobre as funcionalidades do sistema, onde o resultado será um documento para auxiliar os desenvolvedores na realização de alguma alteração futura no sistema, se for o caso. Os envolvidos nesta atividade são os Programadores e o Gerente do Projeto. Origem da atividade: XP e *Scrum.*

- **Refinamento da Documentação Gerada –** Sugere-se que sejam acrescentadas na Documentação de Operações informações a respeito de procedimentos de *backup.* Outros documentos que poderão ser criados são: os Documentos de Suporte que inclui materiais de treinamento específicos para o pessoal de suporte e uma lista de pontos de contato dentro da equipe de manutenção. Também pode ser incluída a Documentação de Usuário, que contempla um manual de referência, um guia de uso, um guia de suporte e até mesmo materiais para treinamento. Os participantes desta atividade são os Programadores e o Gerente do Projeto. Origem da atividade: AM.

Como pôde ser observado, as atividades que fazem parte do *framework* são baseadas nas atividades propostas por cada um dos métodos ágeis selecionados para fazerem parte do mesmo e apresentam diretrizes para a sua utilização, podendo ser utilizadas de acordo com as necessidade da equipe de desenvolvimento. Vale ressaltar que a ordem de execução das atividades devem obedecer a ordem sugerida pelo Desenvolvimento Incremental e que, dependendo da necessidade, as atividades podem ser realizadas mais de uma vez.

Entretanto, algumas atividades são dependentes umas das outras, ou seja, só podem ser realizadas após ou em conjunto com outras atividades. Outra questão a ser levada em consideração é que existem atividades que só podem ser realizadas se outras não forem e vise-versa. Na próxima sessão serão apresentadas as atividades dependentes, excludentes – que não poderão ser realizadas se outras forem - e também opcionais que fazem parte do *framework* proposto, de maneira a auxiliar a equipe de desenvolvimento na sua utilização.

4.2 Identificação das Atividades Opcionais, Dependentes e Excludentes.

De maneira a possibilitar uma melhor compreensão das atividades opcionais, das atividades que dependem de alguma forma da realização de outras atividades e das atividades que não devem ser realizadas se outras forem, ou vice-versa, será apresentada

uma Matriz de Rastreabilidade (Figura 12) contendo todas as atividades propostas pelo *framework*.

As atividades que apresentam um * ao lado do nome – na horizontal - são opcionais durante o processo de desenvolvimento, ou seja, podem ou não ser realizadas. Contudo, vale salientar que a equipe de desenvolvimento deve ficar atenta para as dependências entre as atividades, pois uma determinada atividade pode ser opcional, mas dependendo da escolha das atividades a serem realizadas ela poderá se tornar uma atividade obrigatória.

Para identificar as dependências entre as atividades será utilizada a letra "D", que indica que a atividade apresentada na horizontal depende da realização da atividade apresentada na vertical. Se uma atividade depender de outra, e/ou mais atividades, a letra "D" juntamente com um * fará esta representação.

Já a letra "E" indica que determinada atividade é excludente, ou seja, que a atividade apresentada na horizontal exclui a possibilidade de realização da atividade apresentada na vertical. Neste caso a equipe de desenvolvimento deverá optar por uma das duas atividades.

		Definição dos Requisitos				Atribuição dos Requisitos				Projeto do Sistema				Desenvolver Incremento do Sistema								Validar Incremento			Int.	Validar Sistema		Entrega Final		
		User Stories	Lista de Requisitos	Documentação Detalhada	Organização da Documentação	Planejamento das Iterações	Duração das iterações	Distribuição dos Requisitos	Atualização da Documentação	Projeto Geral	Projeto Detalhado	Documentação do Projeto	Implementação	Escrita dos testes antes da implementação	Desenv. Coletivo	Desenv. em Duplas	*Refactoring*	Reuniões Diárias	Desenv. Simultâneo	Integração Paralela	Documentação do Desenvolvimento	Integração Antes da Validação	Execução dos Testes	Inspeção de Código	Integração Depois da Validação	Reunião de Revisão	Colocar Sistema em Operação	Entrega do Sistema	Breve Documentação	Refinamento da Documentação
Definição dos Requisitos	User Stories *																													
	Lista de Requisitos *																													
	Documentação Detalhada *				D*	D*				D*	D*		D*	D*																
	Organização da Documentação				D*	D*				D*	D*		D*	D*																
Definição dos Requisitos	Planejamento das Iterações						D			D	D		D																	
	Duração das iterações								D																					
	Distribuição dos Requisitos *								D						E	D														
Atribuição dos Requisitos	Atualização da Documentação								D																					
	Projeto Geral									D*	D	D*	D	D																
	Projeto Detalhado											D*																		
Projeto do Sistema	Documentação do Projeto																													
	Implementação																D													
	Escrita dos testes antes da implementação *																													
	Desenv. Coletivo *															E														
Desenvolver Incremento do Sistema	Desenv. em Duplas *															D														
	Refactoring *																D													
	Reuniões Diárias																													
	Desenv. Simultâneo																													
	Integração Paralela																													
	Documentação do Desenvolvimento																													
Validar Incremento	Integração Antes da Validação																													
	Execução dos Testes																													
	Inspeção de Código																													
Int.	Integração Depois da Validação																													
Validar Sistema	Reunião de Revisão																													
	Colocar Sistema em Operação																													
Entrega Final	Entrega do Sistema																													
	Breve Documentação																													
	Refinamento da Documentação																													

		Definição dos Requisitos				Atribuição dos Requisitos				Projeto do Sistema			Desenvolver Incremento do Sistema									Validar Incremento			Int.	Validar Sistema		Entrega Final		
		User Stories	Lista de Requisitos	Documentação Detalhada	Organização da Documentação	Planejamento das Iterações	Duração das iterações	Distribuição dos Requisitos	Atualização da Documentação	Projeto Geral	Projeto Detalhado	Documentação do Projeto	Implementação	Escrita dos testes antes da implementação	Desenv. Coletivo	Desenv. em Duplas	*Refactoring*	Reuniões Diárias	Desenv. Simultâneo	Integração Paralela	Documentação do Desenvolvimento	Integração Antes da Validação	Execução dos Testes	Inspeção de Código	Integração Depois da Validação	Reunião de Revisão	Colocar Sistema em Operação	Entrega do Sistema	Breve Documentação	Refinamento da Documentação
Desenvolver Incremento do Sistema	Reuniões Diárias *																													
	Desenv. Simultâneo *					**D**																								
	Integração Paralela *									**D**			**D**									**E**			**E**					
	Documentação do Desenvolvimento *												**D**																	**D**
Validar Incremento	Integração Antes da Validação *																			**E**					**E**					
	Execução dos Testes	**D***	**D***	**D***									**D**	**D**						**D***		**D***								
	Inspeção de Código *												**D**																	
Int.	Integração Depois da Validação *												**D**							**E**		**E**	**D**							
Validar Sistema	Reunião de Revisão												**D**							**D***		**D***	**D**		**D***					
	Colocar Sistema em Operação *												**D**							**D***		**D***	**D**		**D***					
Entrega Final	Entrega do Sistema																									**D***	**D***			
	Breve Documentação *			**E**					**E**			**C**									**E**									**E**
	Refinamento da Documentação *			**D/E**					**D/E**			**D/E**									**D/E**								**E**	

Figura 12. Matriz de Ratreabilidade das atividades opcionais, dependentes e excludentes do *framework*.

4.2.1 Considerações sobre a Matriz de Rastreabilidade.

- Durante a atividade de Definição dos Requisitos, é obrigatória a realização de no mínimo uma das seguintes atividades: ***User Stories,*** **Lista de Requisitos** e/ou **Documentação Detalhada dos Requisitos.**
- A atividade de **Organização da Documentação** é dependente da atividade de **User Stories** e/ou **Lista de Requisitos** e/ou **Documentação Detalhada dos Requisitos**.
- Para que a atividade de **Planejamento das Iterações** seja realizada é obrigatória a realização de no mínimo uma das seguintes atividades: ***User Stories,*** **Lista de Requisitos** e/ou **Documentação Detalhada dos Requisitos,** pois durante estas atividades os requisitos são agrupados e priorizados e estas definições são fundamentais para que o planejamento das iterações seja realizado.
- A atividade de **Duração das Iterações** é dependente da realização da atividade de **Planejamento das Iterações**, pois sem a realização da mesma, não se sabe quais os requisitos alocados às iterações e, conseqüentemente, o tempo estimado para o seu desenvolvimento. Caso o tempo estimado para o desenvolvimento de um conjunto de requisitos ultrapassar o tempo máximo sugerido, ou seja, 8 semanas, é recomendada a decomposição deste conjunto em subconjuntos, sendo assim, pode ser que seja necessária a realização da atividade de **Planejamento das Iterações** novamente.
- A atividade de **Distribuição dos Requisitos aos Responsáveis** é opcional, ou seja, pode ou não ser realizada.
- A atividade de **Atualização da Documentação** depende da atividade de **Planejamento das Iterações** e **Duração das Iterações,** pois sem as definições feitas durante a mesma não tem o que ser feito durante a atividade de Atualização da Documentação**.** Caso a atividade de **Distribuição dos Requisitos aos Responsáveis** tenha sido realizada sugere-se que sejam incluídas informações a respeito nos documentos gerados.
- A atividade de **Projeto Geral do Sistema** depende da realização da atividade de **User Stories** e/ou **Lista de Requisitos** e/ou **Documentação Detalhada dos**

Requisitos e da realização da atividade de **Planejamento das Iterações,** pois é a partir das informações sobre os requisitos e das dependências existentes entre eles que a atividade de Projeto poderá ser realizada.

- A atividade de **Projeto Detalhado do Sistema** é opcional e depende das atividades de **User Stories** e/ou **Lista de Requisitos** e/ou **Documentação Detalhada dos Requisitos,** bem como do **Planejamento das Iterações,** pois o projeto detalhado é feito no conjunto de requisitos que fazem parte da iteração e do **Projeto Geral do Sistema,** pois as mesmas fornecerão subsídios para a geração de um projeto mais detalhado do sistema.
- A atividade de **Documentação do(s) Projeto(s) Gerado(s)** é opcional e depende das atividades de **Projeto Geral do Sistema** e/ou **Projeto Detalhado do Sistema.**
- A atividade de **Implementação dos Requisitos Durante Cada Iteração** é obrigatória e depende da realização da atividade de **User Stories** e/ou **Lista de Requisitos** e/ou **Documentação Detalhada dos Requisitos,** para guiar os programadores durante a implementação**.** Também depende da atividade de **Planejamento das Iterações** para ter conhecimento dos prazos e dependências entre os requisitos, e da atividade de **Projeto Geral do Sistema** para auxiliar os programadores na compreensão do sistema como um todo.
- A atividade de **Escrita dos Testes de Unidade e Aceitação** antes da implementação é opcional e depende da realização da atividade de **User Stories** e/ou **Lista de Requisitos** e/ou **Documentação Detalhada dos Requisitos,** bem como da atividade de **Projeto Geral do Sistema** para auxiliar os programadores e clientes na definição e escrita dos testes de unidade e de aceitação, respectivamente.
- A atividade de **Desenvolvimento Coletivo de Código** é opcional e exclui a possibilidade de realização da atividade de **Distribuição dos Requisitos aos Responsáveis.** Uma vez que sugere que não existem responsáveis diretos pelo desenvolvimento de um requisito ou conjuntos deles. Sendo assim, a equipe de desenvolvimento deve optar pela utilização de uma ou de outra atividade.

- A atividade de **Desenvolvimento em Duplas** é opcional e depende da atividade de **Distribuição dos Requisitos aos Responsáveis,** uma vez que a dupla de programadores é responsável pela implementação dos requisitos atribuídos a ela ou a equipe a que a dupla pertence. Esta atividade também pode ser realizada em conjunto com a atividade de **Desenvolvimento Coletivo de Código.** Neste caso as duplas de programadores poderão participar do desenvolvimento não só dos requisitos que estão sob a sua responsabilidade, mas de todos os requisitos que estão sendo implementados durante a iteração.
- A atividade de ***Refactoring*** é opcional e poderá ser realizada toda vez que a equipe de desenvolvimento julgar necessário. É dependente da atividade de **Implementação** uma vez que o ***Refactoring*** consiste na reconstrução do código gerado.
- A atividade de **Realização de Reuniões Diárias** é opcional e fica sob a responsabilidade da equipe julgar a necessidade ou não dos encontros diários para verificar o andamento do processo de desenvolvimento.
- A atividade de **Desenvolvimento Simultâneo** é opcional e depende das definições feitas na atividade de **Planejamento das Iterações**, pois é com base nestas definições que a equipe decidirá quais os requisitos pertencentes às diferentes iterações poderão ser desenvolvidos simultaneamente.
- A atividade de **Integração Paralela ao Desenvolvimento** é opcional e depende da atividade de **Projeto Geral do Sistema,** uma vez que o projeto pode auxiliar a equipe a constatar as dependências e necessidades de integração antes da validação e depende da atividade de **Implementação dos Requisitos Durante Cada Iteração,** pois a integração só poderá ser realizada após a geração de pelo menos uma parte do código. Caso esta atividade for realizada as atividades de **Integração Antes da Validação** e **Integração Depois da Validação** não deverão ser realizadas.
- A atividade de **Documentação do Desenvolvimento** é opcional e depende basicamente da atividade de **Implementação dos Requisitos Durante Cada Iteração,** pois é onde os acontecimentos significativos ocorridos durante esta atividade são documentados.

- A atividade de **Integração do Incremento Antes da Validação** é opcional e exclui a possibilidade de realização das atividades de **Integração Paralela ao Desenvolvimento** e **Integração Depois da Validação**.
- A atividade de **Execução dos Testes de Unidade e Testes de Aceitação** dependem das atividades **User Stories** e/ou **Lista de Requisitos** e/ou **Documentação Detalhada dos Requisitos,** uma vez que, é com base nos requisitos dos clientes que serão executados os testes de aceitação, verificando se o sistema está sendo desenvolvido de acordo com o solicitado. Ela também depende da atividade de **Implementação dos Requisitos Durante Cada Iteração,** pois os testes são executados no sistema resultante da iteração, da atividade de **Escrita dos Testes de Unidade e Aceitação,** uma vez que os testes são executados de acordo com o planejado durante a escrita dos mesmos, e das atividades de **Integração do Incremento Antes da Validação** e/ou **Integração Paralela ao Desenvolvimento,** caso a integração do incremento resultante seja necessária para execução dos testes de unidade e/ou aceitação.
- A atividade de **Inspeção de Código** é opcional e dependente da atividade de **Implementação dos Requisitos Durante Cada Iteração,** uma vez que as inspeções são realizadas no código gerado.
- A atividade de **Integração Depois da Validação** é opcional e exclui as atividades de **Integração Paralela ao Desenvolvimento** e **Integração do Incremento Antes da Validação,** pois não requer que o incremento resultante da iteração tenha sido integrado com o(s) outro(s). Ela também é dependente da atividade de **Implementação dos Requisitos Durante Cada Iteração** e da atividade de **Execução dos Testes de Unidade e Testes de Aceitação,** pois a integração do incremento resultante com os outros – sem a dependência de integração antes da validação – só deverá ser realizada após todos os testes terem sido executados com sucesso.
- A atividade de **Reunião de Revisão da Iteração** é dependente das atividades de **Implementação dos Requisitos Durante Cada Iteração** e **Execução dos Testes de Unidade e Testes de Aceitação** e das atividades de **Integração Paralela ao Desenvolvimento** e/ou **Integração do Incremento Antes da**

Validação e/ou **Integração do Incremento Resultante,** uma vez que é a atividade de fechamento da iteração e possibilita a execução de testes de integração.

- A atividade de **Colocar Sistema em Operação** é opcional e dependente das atividades de **Implementação dos Requisitos Durante Cada Iteração** e **Execução dos Testes de Unidade e Testes de Aceitação** e das atividades de **Integração Paralela ao Desenvolvimento** e/ou **Integração do Incremento Antes da Validação** e/ou **Integração do Incremento Resultante,** pois o sistema só poderá ser colocado à disposição do cliente após a realização de todas as atividades citadas anteriormente.
- A atividade de **Entrega do Sistema ao Cliente** é dependente da atividade de **Reunião de Revisão da Iteração** e/ou da atividade de **Colocar Sistema em Operação,** pois o sistema só deverá ser entregue se estiver de acordo com as expectativas do cliente e não apresentar "*bugs*". Lembrando sempre que o sistema só é considerado finalizado após todos os requisitos terem sido desenvolvidos.
- A atividade de **Geração de uma Breve Documentação** é opcional e exclui as atividades de **Documentação Detalhada dos Requisitos, Atualização da Documentação, Documentação do(s) Projeto(s) Gerado(s)** e **Documentação do Desenvolvimento,** uma vez que só necessitará ser gerada se as atividades citadas não tiverem sido realizadas.
- A atividade de **Refinamento da Documentação Gerada** é opcional, exclui à **Geração de uma Breve Documentação,** e dependente das atividades de **Documentação Detalhada dos Requisitos, Atualização da Documentação, Documentação do(s) Projeto(s) Gerado(s)** e **Documentação do Desenvolvimento,** pois consiste no refinamento dos documentos gerados durantes as mesmas.

Sugere-se que a equipe de desenvolvimento interessada em utilizar o *framework* compare e analise em detalhes todas as possibilidades de atividades contidas no

framework, bem como as suas dependências e restrições para que, só após, decida quais delas utilizar.

No próximo capítulo será apresentado um exemplo de utilização do *framework* proposto.

5. EXEMPLO DE UTILIZAÇÃO DO FRAMEWORK

A fim de verificar a eficiência do *framework* proposto, foi desenvolvido um sistema real utilizando o mesmo. O sistema desenvolvido pertence a um projeto de pesquisa executado pela equipe de Tecnologia da Informação do Centro de Informações de Recursos Ambientais e de Hidrometeorologia de Santa Catarina – CIRAM/Epagri e possui entre os seus objetivos o desenvolvimento de um *site* para dar suporte à troca de informações entre as empresas participantes do projeto, bem como para disponibilizar informações a respeito de descargas atmosféricas a fim de monitorar a intensidade e a localização da ocorrência de raios nos estados do Rio Grande do Sul, Santa Catarina e Mato Grosso do Sul.

Existe uma série de funcionalidades a serem desenvolvidas para que o sistema completo seja colocado em funcionamento, algumas delas já são conhecidas, porém outras ainda não são, pois é necessário que os sensores que captarão as informações a respeito das descargas atmosféricas já estejam instalados e funcionando perfeitamente para que se tenha total conhecimento das informações que serão disponibilizadas. A previsão para que os sensores comecem a coletar os dados que serão transformados em informações pelo sistema a ser desenvolvido é fevereiro de 2005. Sendo assim, farão parte do estudo de caso somente as funcionalidades conhecidas até dezembro de 2004. As funcionalidades já definidas correspondem às ferramentas de acesso ao sistema, a um fórum de discussão e a um mural de avisos.

É importante salientar que a equipe técnica responsável pelo desenvolvimento do projeto não utilizava nenhum método formal de desenvolvimento de software. Entretanto, em conversa com os membros da mesma, obteve-se a informação de que na maioria dos sistemas, já desenvolvidos por ela, eram realizadas as seguintes atividades: entrevista com o cliente, reunião da equipe para discussão da solução, programação da solução, apresentação para o cliente, correção de erros, aplicação de melhorias, implantação e manutenção. Apesar de adotadas, as mesmas não seguiam nenhum processo prescritivo para serem realizadas.

Devido a não utilização de um método pré-definido para o desenvolvimento de seus projetos, a equipe técnica colocou-se a disposição para utilizar o *framework* em

questão, uma vez que já há algum tempo sentia a necessidade de desenvolver seus projetos utilizando um método que desse suporte ao desenvolvimento, sem que o mesmo causasse muito impacto na sua adoção, gerasse uma quantidade mínima de documentação, fosse iterativo, incremental e oferece uma certa flexibilidade na sua utilização.

Durante os primeiros contatos com a equipe técnica, constatou-se que todos os membros da mesma possuíam conhecimentos superficiais a respeito da abordagem ágil, com destaque para o método ágil XP. Sendo assim, a primeira providência a ser tomada foi o nivelamento do conhecimento sobre métodos ágeis através de uma apresentação e um estudo detalhado sobre todas as atividades propostas pelo *framework* por todos os membros da equipe. Convém salientar que este nivelamento e estudo foram realizados antes do início do projeto, não comprometendo o tempo estimado para a execução do mesmo.

Outra observação a ser feita é que durante todo o desenvolvimento do projeto a pessoa responsável pela proposição do *framework* realizou o acompanhamento do mesmo de maneira:

- a verificar se o *framework* proposto estava sendo aplicado de forma correta;
- a auxiliar a equipe técnica em qualquer dúvida ou problema encontrado;
- a analisar as considerações - positivas e negativas - feitas por todos os envolvidos no projeto em relação ao *framework*.

Para possibilitar uma visão geral das atividades realizadas durante o processo de desenvolvimento utilizado no exemplo, a tabela a seguir mostra todas atividades propostas pelo *framework* e as atividades realizadas – destacadas em negrito - durante o desenvolvimento do sistema que serviu para exemplificar a utilização do mesmo.

Atividades propostas pelo *Framework*		
User Stories	**Documentação do Projeto**	**Integração Antes da Validação**
Lista de Requisitos	**Implementação**	**Execução dos Testes**
Documentação Detalhada	**Escrita dos testes**	**Inspeção de Código**
Organização da Documentação	Desenvolvimento Coletivo	Integração Depois da Validação
Planejamento das Iterações	Desenvolvimento em Duplas	**Reunião de Revisão**
Duração das iterações	*Refactoring*	Colocar Sistema em Operação
Distribuição dos Requisitos	**Reuniões Diárias**	**Entrega do Sistema**
Atualização da Documentação	Desenvolvimento Simultâneo	Breve Documentação
Projeto Geral	Integração Paralela	**Refinamento da Documentação**
Projeto Detalhado	Documentação do Desenvolvimento	

Tabela 11. Atividades realizadas durante o exemplo de utilização do *framework.*

A seguir será apresentado o exemplo de utilização do framework.

5.1 Atividades de Definição dos Requisitos

Lista de Requisitos

Neste primeiro momento foram identificados como usuários que auxiliarão a equipe de desenvolvimento no processo de desenvolvimento, os representantes de cada uma das empresas que fazem parte do projeto. Foi realizada uma reunião com estes usuários a fim de apresentar uma visão geral do projeto, e começar o processo de levantamento de requisitos do sistema através de entrevistas com os mesmos.

Nestas entrevistas foi possível gerar uma lista preliminar dos requisitos que farão parte do sistema conforme apresentado a seguir na Figura 13. As informações a respeito do tempo estimado para o desenvolvimento e os responsáveis pelo mesmo, não foram inseridas neste documento por ainda não serem conhecidas.

Lista de Requisitos

Prioridade	Item	Descrição
Muito Alta		
	1	Cadastrar Usuário
	2	Consultar Usuário
	3	Autorizar Cadastro
	4	Efetuar *Login*
	5	Montar menu do usuário
	6	Excluir usuário
Alta		
	7	Visualizar Avisos
	8	Incluir Avisos
	9	Excluir Avisos
Média		
	10	Incluir tema no fórum
	11	Visualizar temas no fórum
	12	Postar mensagem p/ um tema no fórum
	13	Excluir tema no fórum

Figura 13. Exemplo de Lista de Requisitos

Documentação Detalhada dos Requisitos

Uma vez que alguns membros da equipe de desenvolvimento já tinham experiência na geração de Casos de Uso para formalizar os requisitos do sistema, e tinham a necessidade e preocupação com tal formalização optou-se por realizar também esta atividade para documentar os mesmos. Com base no detalhamento dos requisitos foi possível estimar o tempo de desenvolvimento dos mesmos. Esta informação foi incluída nos Casos de Uso.

Organização da Documentação

Após a geração dos Casos de Uso e de algumas entrevistas com os usuários a fim de verificar se os Casos de Uso estavam de acordo com o solicitado por eles, foi criado o Documento de Requisitos (Anexo 1), contemplando os artefatos gerados, bem como o documento contendo a Visão Geral do Projeto, conforme pode ser observado na Figura 14. Estes documentos foram gerados, de modo a suprir a necessidade de formalização destas informações.

Visão Geral do Projeto

O presente projeto contempla o processo de desenvolvimento de um sistema de informação a ser utilizado pelas empresas da área de energia elétrica para o monitoramento e troca de informações a respeito da ocorrência de descargas atmosféricas nos estados Rio Grande do Sul, Santa Catarina e Mato Grosso do Sul.

O projeto será executado pela equipe de Tecnologia da Informação do Centro de Informações de Recursos Ambientais e de Hidrometeorologia de Santa Catarina – CIRAM/EPAGRI e disponibilizado via *Web* através de um *site* armazenado dentro do próprio Centro.

Os principais objetivos do sistema em desenvolvimento são possibilitar a troca de informações a respeito do projeto entre os usuários autorizados, através de um fórum de discussão e um mural de avisos, bem como monitorar a intensidade e a localização da ocorrência de raios de modo a minimizar substancialmente os prejuízos causados por eventos meteorológicos críticos relacionados a descargas atmosféricas e precipitações intensas.

Equipe de Desenvolvimento[15]

Papel	Nome	Contato
Gerente do Projeto	Gerente	gerente@projeto...
Responsável pelos Requisitos e Testador	Requisitos	requisitos@projeto...
Programador(es)	Programador	programador@projeto...
Usuário 1	Usuário 1	usuario1@projeto...
Usuário 2	Usuário 2	usuario2@projeto...
Usuário 3	Usuário 3	usuario3@projeto...
Usuário 4	Usuário 4	usuario4@projeto...
Usuário 5	Usuário 5	usuario5@projeto...

Tecnologia (hardware e software)

Servidor	Software(s)
Web	• Sistema Operacional *Linux Red Hat* 7.3; Apache 1.3; PHP4; *Smart*CVS.
Banco de Dados	• Sistema Operacional *Linux Red Hat* 7.3; MySQL 4.0.13

Figura 14. Visão Geral Executiva

5.2 Atividades de Atribuição dos Requisitos as Iterações

Planejamento das Iterações.

Esta atividade começou a ser desenvolvida paralelamente a Definição dos Requisitos. E foi concluída com a definição da quantidade de iterações e quais os requisitos que seriam alocados a elas. Como a equipe de desenvolvimento já havia feito

[15] De modo a preservar a real identidade dos envolvidos no projeto, optou-se por utilizar nomes e *e-mails* fictícios toda a vez que for necessária alguma referências aos mesmos.

o agrupamento e priorização dos mesmos, os requisitos foram alocados as iterações de acordo com esta definição, ou seja, os requisitos com prioridade “Muito Alta” serão desenvolvidos na 1ª iteração e os requisitos com prioridade “Alta” e Média serão desenvolvidos na 2ª iteração. Para auxiliar na organização do desenvolvimento, a equipe optou por estipular datas de início e fim das iterações, baseadas nas estimativas de tempo de desenvolvimento de cada um dos requisitos. Informações estas que serão documentadas na atividade de Atualização da Documentação.

Duração das Iterações.

Em virtude do tempo estimado, da prioridade e da baixa complexidade dos conjuntos de requisitos, a equipe técnica definiu que os mesmos seriam divididos em duas iterações com duração máxima de duas semanas cada.

Distribuição dos Requisitos aos Responsáveis

Vale salientar que a equipe de desenvolvimento é composta por apenas um programador. Sendo assim, durante o Planejamento das Iterações, o mesmo tomou conhecimento dos requisitos que ficarão sob a sua responsabilidade.

Atualização da Documentação

Durante esta atividade foram incluídas informações a respeito das definições em relação as iterações e seu tempo de duração no documento contento a lista dos requisitos. Conforme pode ser observado na Figura 15.

Lista de Requisitos

Prioridade	Item	Descrição	Iteração	Data de início e fim
Muito Alta				
	1	Cadastrar Usuário	1ª	De: 03/12/2004 Á: 13/01/2005
	2	Consultar Usuário		
	3	Autorizar Cadastro		
	4	Efetuar *Login*		
	5	Montar menu do usuário		
	6	Excluir usuário		
Alta				
	7	Visualizar Avisos	2ª	De: 14/01/2005 Á: 24/01/2005
	8	Incluir Avisos		
	9	Excluir Avisos		
Média				
	10	Incluir tema no fórum		
	11	Visualizar temas no fórum		
	12	Postar mensagem p/ um tema no fórum		
	13	Excluir tema no fórum		

Figura 15. Lista de Requisitos Atualizada

5.3 Atividades de Projeto da Arquitetura do Sistema

Projeto Geral do Sistema

Foi realizada uma reunião entre os membros da equipe de desenvolvimento, para a definição dos artefatos necessários para auxiliar na implementação dos requisitos do sistema, bem como da definição dos padrões de codificação e nomenclatura que serão utilizados durante o projeto e a geração do código. Tendo em vista a baixa complexidade dos mesmos, foi decido nesta reunião que a equipe de desenvolvimento deverá gerar um modelo de dados baseado nos Casos de Uso, para representar as entidades, atributos e relacionamentos, com o objetivo de auxiliá-los na criação do banco de dados que armazenará os dados relativos aos requisitos que serão implementados.

Documentação do(s) Projeto(s) Gerado(s)

Sendo assim, a equipe técnica gerou o modelo de dados na ferramenta *Enterprise Architect 4.0* – que permite a exportação do modelo gerado para o MySQL - e o mesmo foi anexado a Documentação do Sistema, juntamente com os padrões de codificação e nomenclatura. Os – Padrões de codificação e nomenclatura poderão ser visualizados no Anexo 2, e os Modelos de Dados gerados poderão ser visualizado no Anexo 3.

5.4 Atividades de Desenvolvimento do Incremento do Sistema

Implementação dos requisitos durante cada iteração

A implementação do conjunto de requisitos pertencente as iterações aconteceu dentro do prazo estabelecido, onde em um primeiro momento aconteceu a definição dos exportação dos modelos de dados criado na ferramenta *Enterprise Architect 4.0* para o banco de dados MySQL e geradas as telas no editor HTML *Macromedia Dreamweaver 3*. Após as telas serem geradas, iniciou-se o processo de escrita do código - também realizado no editor HTML *Macromedia Dreamweaver 3*.

Escrita dos Testes de Unidade e Aceitação

É importante salientar que como foram utilizados casos de uso para o detalhamento dos requisitos, os mesmos serão utilizados para a realização dos testes de aceitação, sendo assim, a equipe de desenvolvimento não encontrou necessidade dos mesmos serem escritos. Em relação aos testes de unidade, a equipe decidiu que os mesmos seriam escritos à medida que o código ia sendo gerado.

Realização de Reuniões Diárias

Foram realizadas reuniões diárias, com a presença da equipe de desenvolvimento, durante a implementação, todas com o objetivo de verificar o andamento da implementação, quais os problemas encontrados e qual a solução para os

mesmos. Como se trata do desenvolvimento de requisitos relativamente simples, a quantidade de problemas encontrados durante esta atividade foi mínima, sendo que os mesmos foram solucionados sem comprometer o planejamento das iterações.

5.5 Atividades de Validação do Incremento

Inspeção de Código

Após a implementação, o Responsável pelos Requisitos realizou a inspeção do código e solicitou ao programador que adicionasse comentários em algumas partes do código que geraram questionamentos.

Integração do Incremento Antes da Validação

A equipe optou pela realização desta atividade durante a segunda iteração, pois houve a necessidade de integrar os requisitos desenvolvidos durante a segunda iteração com os desenvolvidos na primeiro para que as validações pudessem ser executadas.

Execução dos Testes de Unidade e Testes de Aceitação

Após o desenvolvimento e a inspeção de código da primeira iteração e do desenvolvimento, da inspeção e da integração da segunda iteração, foram executados os testes de unidade e aceitação. Sendo assim, os usuários foram convocados para fazerem os testes de aceitação de acordo com os casos de uso nos incrementos resultantes.

Não foi encontrado nenhum erro de sistema durante os testes de aceitação realizados, nem tampouco foram alterados e/ou sugeridos novos requisitos. As únicas considerações feitas foram a respeito da apresentação das telas, principalmente em relação aos objetos que compõem as mesmas. Estas considerações foram imediatamente incorporadas às telas dos incrementos.

5.6 Atividade de Validação do Sistema

Reunião de Revisão da Iteração

Após a atividade de desenvolvimento e validação ter ocorrido com sucesso, foi realizada uma reunião com a equipe de desenvolvimento e o propositor do *framework*, a fim de avaliar o andamento do processo de desenvolvimento. Em relação a entrega dos requisitos desenvolvidos, o Gerente do Projeto sugeriu a não disponibilização dos incrementos desenvolvidos para os usuários das empresas que fazem parte do projeto, mesmo estando ele pronto para entrar em operação, pois o projeto ainda possui funcionalidades a serem desenvolvidas. O Testador – mesmo Responsável pelos Requisitos – realizou mais alguns testes de sistema e nenhum problema foi constatado.

5.7 Atividades de Entrega Final

Entrega do Sistema ao Cliente

Conforme colocado no início do Capítulo 5, ainda restam funcionalidades a serem definidas para que o projeto seja considerado concluído. As funcionalidades desenvolvidas até agora servem exclusivamente para dar suporte a troca de informações a respeito das descargas atmosféricas, informações estas que ainda não se tem conhecimento. Sendo assim, o Gerente do Projeto, convocou outra reunião e colocou esta questão aos usuários das empresas, que concordaram que não fazia sentido disponibilizar o sistema desenvolvido até agora para operação, uma vez que o mesmo só será utilizado quando o restante das funcionalidades forem desenvolvidas.

Sendo assim, a atividade de entrega dos conjuntos de requisitos só será realizada em um segundo momento, após os sensores serem instalados e as informações a respeito das descargas atmosféricas serem disponibilizadas no *site*.

Refinamento da Documentação Gerada

Como este exemplo de utilização do *framework* tem o objetivo de verificar se o mesmo tem condições de ser utilizado no desenvolvimento de um projeto de software,

mesmo com o projeto considerado não concluído, a equipe técnica optou por realizar a atividade de reorganização da documentação de forma a deixar a Documentação do Sistema pronta para receber os novos documentos que serão gerados futuramente no desenvolvimento das funcionalidades que ainda precisam ser desenvolvidas. Esta atividade foi realizada rapidamente, uma vez que a quantidade e complexidade dos documentos gerados eram mínimas.

Algumas telas pertencentes aos conjuntos de requisitos desenvolvidos neste projeto podem ser visualizadas no Anexo 4.

5.8 Conclusões do Estudo de Caso

Acredita-se ser importante salientar que, após todos os conjuntos de requisitos previstos serem desenvolvidos, o propositor do *framework* realizou uma reunião com a equipe de desenvolvimento a fim de verificar se a utilização do mesmo aconteceu de acordo com as expectativas da equipe. Nesta reunião, todos os envolvidos se manifestaram de forma positiva em relação aos resultados, colocando que a facilidade de aplicação do *framework* no projeto em questão possibilitou que o mesmo fosse utilizado sem causar impactos significativos na rotina de trabalho da equipe de TI do CIRAM/EPAGRI.

Desta forma, acredita-se que o *framework* ajudou positivamente a equipe no desenvolvimento do projeto, da mesma forma que agregou conhecimento aos seus membros, uma vez que muitas atividades propostas por ele eram desconhecidas.

No próximo capítulo serão apresentadas as considerações finais deste trabalho, mostrando de forma sucinta o que foi realizado ao longo do mesmo, possibilitando a identificação dos objetivos alcançados, quais as contribuições acadêmicas deste estudo e, por fim, algumas propostas de trabalhos futuros relacionados com o tema em questão.

6. CONSIDERAÇÕES FINAIS

6.1 Objetivos Alcançados

Neste trabalho, primeiramente foi apresentado um estudo contendo as características gerais dos métodos ágeis XP, *Scrum*, FDD, ASD e AM, mostrando em detalhes as suas práticas, suas atividades e os papéis da equipe pertencente a cada um dos métodos, onde foram identificadas suas semelhanças e diferenças, de maneira que a idéia preliminar de propor um *framework* com as atividades dos métodos citados tornou-se viável. De forma geral, suas abordagens baseiam-se fortemente na decomposição de problemas complexos em problemas de granularidade menor, onde o processo de desenvolvimento ocorre em ciclos curtos e ao final de cada iteração acontece a entrega de uma pequena parte do sistema funcionando.

Este estudo possibilitou a realização de uma comparação e análise que permitiu de forma mais clara a identificação dos aspectos comuns entre estes métodos, bem como das suas diferenças, fornecendo subsídios para a proposta do *framework*. Sendo assim, com base nesta análise foi proposto o *framework* para comparação e análise de métodos ágeis, que apresenta um conjunto de atividades baseadas nas atividades propostas por cada um dos métodos ágeis que fazem parte do mesmo e que podem ser selecionadas de acordo com a necessidade dos *stakeholders*.

E, por fim, o *framework* proposto foi aplicado no desenvolvimento de um sistema real para verificar a sua eficiência e coerência. A partir desta aplicação, pode-se concluir que o *framework* proposto, através das atividades instanciadas, possibilitaram à equipe de desenvolvimento desenvolver um sistema que atendesse as necessidades do cliente e, principalmente, dentro do prazo pré-estabelecido para a entrega.

Outros resultados significativos foram identificados através da reunião realizada para verificar se as expectativas da equipe técnica foram alcançadas com a utilização do *framework* proposto Nesta reunião foram levantadas várias questões positivas em relação ao mesmo, como por exemplo, a sua facilidade de aplicação, a importância da documentação e formalização dos requisitos para uma compreensão preliminar a respeito do sistema, das validações realizadas e dos artefatos e modelos gerados auxiliando a implementação.

Entretanto, acredita-se ser de extrema importância que o *framework* seja utilizado em outros projetos com diferentes complexidades, tendo em vista que o projeto apresentado neste estudo de caso é considerado de tamanho pequeno e apresenta um baixo grau de dificuldade. Sendo assim, pode-se afirmar que a complexidade do projeto pode comprometer o sucesso do mesmo, e somente aplicando o *framework* em outros projetos com estas características, é que de fato se poderá afirmar que o mesmo atende completamente os objetivos propostos pela abordagem ágil.

6.2 Contribuições

Conforme apresentado na introdução, a quantidade de pesquisas e publicações acadêmicas referentes aos métodos ágeis ainda é mínima. Sendo assim, espera-se, com este trabalho, contribuir para o aumento do conhecimento das pessoas interessadas na abordagem ágil, bem como em despertar o interesse em relação à mesma, reconhecendo a sua importância no contexto atual, já que, cada vez mais, existe a necessidade de alternativas que dêem suporte ao desenvolvimento de sistemas de qualidade que satisfaçam de uma maneira geral os seus clientes.

Em relação aos trabalhos que também apresentam comparações entre métodos ágeis, pode-se dizer que as comparações realizadas no presente trabalho apresentaram objetivos diferentes dos mostrados nos mesmos, ou seja, o de realizar uma comparação para propor um *framework* para comparação e análise dos métodos ágeis XP, *Scrum*, FDD, ASD e AM. Conforme pôde ser observado, a comparação realizada em Abrahamsson (2003), além de não propor um *framework*, apresenta outras diretrizes para comparar os métodos ágeis e utiliza os métodos ágeis *Crystal Family*, DSDM, ISD e PP, métodos estes que não fazem parte do escopo das comparações realizadas aqui. Já a comparação apresentada em Cohn (2002), também apresenta outros critérios de comparação. Cohn (2002) apresenta uma comparação em relação aos princípios ágeis, o que não é o caso deste trabalho, e esta comparação é feita entre os métodos ágeis XP, *Scrum,* FDD, *Crystal Family* e DSDM, e neste trabalho a comparação é realizada entre os métodos ágeis XP, *Scrum*, FDD, ASD e AM.

Sendo assim, acredita-se que este trabalho possui objetivos diferentes dos apresentados nos trabalhos relacionados com o tema, entretanto os mesmos

possibilitaram um embasamento teórico sobre os métodos ágeis e auxiliaram, de certa forma, na escolha das diretrizes para a realização das comparações apresentadas.

6.3 Trabalhos Futuros

Como sugestões de trabalhos futuros, objetivando dar continuidade à proposta apresentada neste trabalho, pode-se relacionar o seguinte:

- Incorporação dos outros métodos ágeis disponíveis no mercado ao *framework* proposto de forma a possibilitar que o mesmo contemple, senão todos, a maioria dos métodos ágeis, agregando mais valor a ele;

- Aplicação do *framework* em outros projetos, com tamanhos e complexidades diferentes deste que foi utilizado como exemplo, de maneira a verificar se o *framework* proposto pode se utilizado em diferentes contextos com sucesso.

7 BIBLIOGRAFIA

ABRAHAMSSON, Pekka; SALO, Outi. *Agile Software Development Methods– Review and Analysis*. Espoo 2002, VTT Publications 478 107.

ABRAHAMSSON, Pekka; WARSTA, Juhani; SIPONEN, Mikko; RONKAINEN, Jussi. *New Directions on Agile Methods: A Comparative Analysis*. IEEE *Computer Science*, Maio de 2003.

ADM (Advanced Development Methods, Inc)**.** *The Philosophy of Scrum*. Disponível em < http://www.controlchaos.com/old-site/philo.htm>. Ano: 2004 Acesso em abr. 2004.

AMBLER, Scott W**.** Modelagem ágil: práticas eficazes para a Programação Extrema e o Processo Unificado. Trad. Acauan Fernandes. Porto Alegre: Bookman, 2004

ASTELS, David; MILLER, Granville; NOVAK, Miroslav. *Extreme Programming Explained*: Guia Prático. Rio de Janeiro: Ed. Campus, 2002.

BECK, Kent. *Embracing change with Extreme Programming*. IEEE *Computer Science*, Outubro de 1999.

BECK, Kent. *Extreme Programming Explained*: Embrace change. Reading, Massachusetts: Ed. Addison-Wesley, 2000.

BECK, K.; COCKBURN, A.; JEFFRIES, R.; HIGHSMITH, J., *Agile Manifesto.* Ano 2001. Disponível em <http://www.agilemanifesto.org>, Ano: 2001. Acesso em fev. 2004.

BEEDLE, Mike; DEVOS, Martine; SHARON, Yonat; SCHWABER, Ken; SUTHERLAND, Jeff. *SCRUM: An extension pattern language for hyperprodutive software development*. Pattern Languages of Programs'98 Conference, 1998.

BOOCH , G.; Rumbaugh, J.; Jacobson, I. *The Unified Modeling Language User Guide.* Ed. Addison-Wesley, 1999

BOEHM, B., *Get Ready for Agile Methods, with Care*. IEEE Computer *Science*, janeiro de 2002.

BOEHM, B. DEMARCO**,** T. *The Agile Methods Fray*. IEEE Computer *Science*, junho de 2002.

COAD, Peter. *Java Modeling in Color with UML*. Prentice Hall, 1999.

COCKBURN, Alistair. *Agile Software Development.* Addison-Wesley. 2001.

COCKBURN, Alistair; HIGHSMITH, J *Agile Software Development: The people factor.* IEEE *Computer Science*, Novembro de 2001.

COCKBURN, Alistair. Escrevendo Casos de Uso Eficazes. Bookman. 2005.

COHEN, David; LINDVALL, Mikael; COSTA, Patricia. *A State of the Art Report: Agile Software Development.* Fraunhofer Center for Experimental *Software* Engineering Maryland and The University of Maryland, 2003.

COHN, Mike. *Selecting an Agile Process: Comparing the Leading Alternatives.* Ano: 2002. Disponível em: http://www.mountaingoatsoftware.com/pres/Selecting 021015.pdf. Acesso em maio de 2004.

De LUCA, Jeff. *Feature-Driven Development (FDD) Overview Presentation.* Ano: 2002. Disponível em: <http://www.nebulon.com/articles/fdd/download/ fddoverview.pdf >. Acesso em mai 2004.

FOWLER, Martin. *The New Methodology*. Ano: 2001. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em abr. 2004.

HIGHSMITH, Jim; BAYER, San. *RADical Software Development. American Programmer Magazine*, 1994

HIGHSMITH, Jim; COCKBURN, Alistair**.** *Agile Software Development: The Business of Innovation.* IEEE *Computer Science*, Setembro, 2001.

HIGHSMITH, Jim. *Agile Software Development Ecosystems*. Addison Wesley, 2002a.

HIGHSMITH, Jim. *What is Agile Software Development? CrossTalk Magazine*, Outubro de 2002b.

HIGHSMITH, Jim. *Life-The Artificial and the Real*. *Software Development*. Dezembro de 2002c.

IEEE. *IEEE Recommended Practice for Software Requirements Specification*". In Thayer, R.H e Dorfman, M. *Software Requirements Engineering* IEEE *Computer Society Press*, Los Alamitos, USA, 1998.

JEFFRIES, Ron. XP Magazine Contents: *What is Extreme Programming?* 2001. Disponível em: <http://www.xprogramming.com/xpmag/index.htm>. Acesso em abr. 2004.

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

LARMAN, Craig. *Iterative and Incremental Development: A Brief History*. IEEE *Computer Science*, Junho, 2003.

LINDA Rising, NORMAN Janoff, *The SCRUM Software Development Process for Small Teams.* IEEE *Software*, Agosto de 2000.

MARTIN, Robert C. *Agile Processes - (Cap.1).* Ano 2001. Disponível em: http://www.objectmentor.com/resources/articles/agileProcess.pdf. Acesso em junho de 2004.

PALMER, Stephen; FELSING, John *A Practical Guide to Feature-Driven Development. Prentice Hall,* 2002.

PALMER, Stephen. *Feature Driven Development - Integrating Best Practices*. Ano:2003 Disponível em <http://www.step10.com/process/IntegratingBestPractices.html> Acesso em mai de 2004.

PRESSMAN, Roger S. Engenharia de *Software.* São Paulo: Makron Books, 1995.

RUMBAUGH, James. *What Is a Method?* Journal of Object Oriented Programming. Outubro de 1995.

SUTHERLAND, Jeff. *SCRUM Software Development Process*. Ano: 2000. Disponível em: <http://jeffsutherland.com/*Scrum*/index.html>. Acesso em abr. 2004.

SCHWABER, Ken. *Scrum Development Process*. OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag.

SCHWABER, Ken; BEEDLE, Mike. *Agile Software Development with SCRUM.* Prentice Hall, 2002.

SCHWABER, Ken. *Agile Process and Self-Organization*. Ano: 2002. Disponível em http://www.agilealliance.org/articles/articles/Agile_Processes_and_Self_Organization.pdf. Acesso em jul. 2004

SOMMERVILLE, Ian, Engenharia de *Software*, São Paulo: Addison-Wesley, 2003.

WAKE, William C. *Extreme Programming Explored.* Reading, Massachusetts: Ed. Addison-Wesley, 2002.

WELLS, Don. *Extreme Programming: A gentle introduction.* Ano: 2001. Disponível em: <http://www.extremeprogramming.org>. Acesso em abr. 2004.

ZANATTA, Alexandre. *xScrum*: uma proposta de extensão do *Scrum* para adequação ao CMMI. UFSC. Trabalho individual. Florianópolis: 2004.

ANEXO 1 – Documento de Requisitos

Ator	Descrição
Usuário	Este ator é qualquer usuário que navegue pelo *site*. Não possui obrigatoriedade de cadastro.
Administrador	Este ator é um tipo de usuário que possui cadastro e permissão de administrador. Possui acesso a todas as informações do *site*.
Usuário Cadastrado	Este ator é um tipo de usuário que possui cadastro e pertence a uma determinada empresa. Ele só possui acesso às sessões e informações da empresa a qual pertence.

Caso de Uso	**Efetuar *Login* [CDU001]**
Descrição	O seguinte caso de uso mostra o processo de *login* no sistema através da validação dos dados inseridos.
Prioridade	Muito Alta
Estimativa	8 h
Atore(s)	Usuário Cadastrado e Administrador
Pré-condição	Usuário conectado ao *site* do projeto
Cenário de sucesso principal	1. Usuário digita o e-mail* e senha*; 2. Usuário submete as informações; 3. Sistema identifica a empresa o qual o usuário pertence através do e-mail digitado; 4. Incluir CDU002.
Extensões	2a. Usuário não possui cadastro; 2a1.Sistema mostra mensagem informando que o usuário não possui cadastro e abre a página para que o mesmo seja realizado; 2a2. Incluir CDU010. 2b. Usuário digita e-mail e senha incorretos; 2b1. Sistema mostra uma mensagem informando que os dados digitados estão incorretos. 2c. Cadastro ainda não autorizado; 2c1. Sistema mostra uma mensagem informando que o cadastro ainda não foi autorizado. 2d. Algum campo não foi preenchido; 2d1. Sistema mostra uma mensagem informando que todos os campos deverão ser preenchidos.
Observações	1. O usuário poderá limpar os dados digitados e digitar as informações quantas vezes desejar. 2. *. Campos de preenchimento obrigatório.

Caso de Uso	**Montar menu do usuário [CDU002]**
Descrição	Este caso de uso descreve o processo de montagem do menu do *site*. A disponibilidade das seções depende do tipo do usuário e o grupo a que pertence.
Prioridade	Muito Alta
Estimativa	8 h
Atore(s)	-
Pré-condição	Usuário Cadastrado ou Administrador logado no sistema
Cenário de sucesso principal	1. Sistema verifica o tipo do usuário logado; 2. Sistema mostra o *site* de acordo com o tipo de usuário logado.

Extensões	1a. Usuário tipo: Usuário Cadastrado; 1a1.Sistema verifica a empresa que o Usuário Cadastrado faz parte; 1a2. Sistema monta o *site* com as informações da empresa e monta o menu com as sessões disponíveis às empresas que fazem parte do consórcio. 1b. Usuário tipo: Administrador; 1b1. Sistema monta o *site* monta o menu com as sessões disponíveis aos administradores.

Caso de Uso	**Cadastrar Usuário [CDU010]**
Descrição	O seguinte caso de uso mostra o processo de cadastro de um usuário no *site*.
Prioridade	Muito Alta
Estimativa	4 h
Atore(s)	Usuário
Pré-condição	Usuário conectado ao *site* do projeto
Cenário de sucesso principal	1. Usuário informa os seguintes dados: nome*, e-*mail*, *site*, senha*, data de nascimento*, sexo*, área de interesse no *site**, outra área de interesse, empresa**, profissão*, outra profissão, área de atuação pessoal*, endereço comercial, cidade, estado*, outro estado, país*, outro país, cep, fone, fax, comentários. 2. Usuário submete os dados; 3. Sistema efetua cadastro do usuário inserido o dado data do cadastro automaticamente; 4. Sistema envia um e-mail p/ o(s) administrador(es) informando sobre o novo cadastro; 5. Sistema mostra uma mensagem informando que o cadastro foi efetuado com sucesso e que ele deve aguardar um e-mail informando que seu cadastro foi autorizado.
Extensões	4a. Dados submetidos estão incompletos; 4a1. Sistema mostra uma mensagem com os dados que faltam. 4b. Já existe usuário cadastrado com o e-mail digitado; 4b1. Sistema mostra uma mensagem, solicitando um novo e-mail.
Observações	1. O usuário poderá limpar os dados digitados e digitar as informações quantas vezes desejar. 2. *. Este campo é de preenchimento obrigatório. 3. **. Campo empresa - sistema busca as empresas que fazem parte do consórcio. 4. A quantidade de caracteres máximos de cada um dos campos deverá respeitar o estipulado no Modelo de Dados anexado à Documentação do Sistema.

Caso de Uso	**Autorizar Cadastro [CDU011]**
Descrição	O seguinte caso de uso mostra o processo de autorização do cadastro de um determinado usuário.
Prioridade	Muito Alta
Estimativa	4 h
Atore(s)	Administrador
Pré-condição	Administrador logado no sistema.
Cenário de sucesso principal	1. Sistema mostra todos os usuários cadastrados; 2. Administrador seleciona o usuário que deseja autorizar; 3. Administrar efetua autorização do usuário; 4. Sistema altera o status do usuário e envia um e-mail para o mesmo avisando sobre a autorização do seu cadastro.

Extensões	6a. Erro no envio do e-mail; 6a1. Sistema mostra uma mensagem informando sobre o erro no envio do e-mail.

Caso de Uso	**Excluir usuário [CDU012]**
Descrição	O seguinte caso de uso mostra o processo de exclusão de um usuário.
Prioridade	Muito Alta
Estimativa	4 h
Atore(s)	Administrador
Pré-condição	Administrador logado no sistema.
Cenário de sucesso principal	1. Sistema mostra todos os usuários cadastrados; 2. Administrador seleciona o usuário que deseja excluir; 3. Sistema exclui usuário.
Extensões	3a. Usuário possui algum tema do fórum ou aviso relacionado 3a1. Sistema mostra uma mensagem, avisando que o usuário não poderá ser excluído.

Caso de Uso	**Consultar Usuário [CDU013]**
Descrição	O seguinte caso de uso mostra o processo de acesso aos dados de um determinado usuário
Prioridade	Muito Alta
Estimativa	4 h
Atore(s)	Administrador
Pré-condição	Administrador logado no sistema.
Cenário de sucesso principal	1. Sistema mostra todos os usuários cadastrados; 2. Administrador seleciona o usuário desejado; 3. Sistema mostra todos dados do usuário selecionado na tela.

Caso de Uso	**Visualizar Avisos [CDU020]**
Descrição	O seguinte caso de uso mostra o processo de visualização da lista de avisos disponíveis ao ator empresas.
Prioridade	Alta
Estimativa	4 h
Ator(es)	Usuário Cadastrado e Administrador
Pré-condição	Administrador ou Usuário Cadastrado logado no sistema.
Cenário de sucesso principal	1. Entrar no visualizador de avisos; 2. Sistema mostra todos os avisos cadastrados disponíveis ao ator logado.

Extensões	1a. Usuário logado: Administrador; 1a1. Sistema busca as empresas cadastradas; 1a2. Administrador seleciona opção desejada; 1a3. Sistema busca todos os avisos cadastrados de acordo com a opção selecionada e mostra a lista na tela, com as seguintes informações: data da inclusão, usuário que incluiu, empresa a qual pertence o usuário que incluiu o aviso e título do aviso. 1b. Usuário logado: Usuário Cadastrado; 1b1. Sistema busca todos os avisos disponíveis ao usuário logado, com as seguintes informações: data da inclusão, usuário que incluiu, empresa a qual pertence o usuário que incluiu o aviso e título do aviso.
Observações	Caso o usuário logado deseje visualizar a descrição do aviso, ele deverá clicar no título do mesmo.

Caso de Uso	**Incluir Avisos [CDU021]**
Descrição	O seguinte caso de uso mostra o processo de incluir avisos.
Prioridade	Alta
Estimativa	8 h
Atore(s)	Administrador e Usuário Cadastrado.
Pré-condição	Administrador ou Usuário Cadastrado logado no sistema.
Cenário de sucesso principal	1. Ator digita as seguintes informações: título, descrição, data em que aviso deverá sair do ar e seleciona quais as empresas que terão acesso ao aviso; 2. Ator submete as informações; 3. Sistema inclui aviso no sistema associado ao usuário logado que incluiu o aviso; 4. Sistema envia mensagem para o e-mail de todos as pessoas que terão acesso ao aviso, com uma mensagem que existe um novo aviso no *site* (com link para o *site*).
Extensões	4a. Problemas na inserção do aviso; 4a1. Sistema mostra uma mensagem ao usuário logado informando que a inserção do aviso não foi possível. 4b. Algum campo não foi preenchido; 4b1. Sistema mostra uma mensagem ao usuário logado informando que todos os campos deverão ser preenchidos.
Observações	1. O usuário poderá limpar os dados digitados e digitar as informações quantas vezes desejar. 2. *. Campos de preenchimento obrigatório.

Caso de Uso	**Excluir Avisos [CDU023]**
Descrição	O seguinte caso de uso mostra o processo de exclusão de um determinado aviso.
Prioridade	Alta
Estimativa	4 h
Atore(s)	Administrador e Usuário Cadastrado.
Pré-condição	Administrador ou Usuário Cadastrado logado no sistema.
Cenário de sucesso principal	1. Ator seleciona o aviso que deseja excluir e efetua exclusão; 2. Sistema solicita confirmação de exclusão; 3. Sistema exclui o aviso selecionado mostrando uma confirmação para a exclusão solicitada.

Extensões	1a. Usuário logado: Administrador; 1a1. Sistema busca as empresas cadastradas; 1a2. Administrador seleciona a empresa desejada; 1a3. Sistema busca todos os avisos cadastrados de acordo com a opção selecionada e mostra a lista na tela ordenada pela data da inclusão com as seguintes informações: data da inclusão, usuário que incluiu, empresa que pertence o usuário que incluiu o aviso, título do aviso e uma caixa de opções (para o ator selecionar qual o aviso deseja excluir). 1b. Usuário logado: Usuário Cadastrado; 1b1. Sistema busca todos os avisos incluídos pelo usuário logado, com as seguintes informações e mostra a lista na tela ordenada pela data da inclusão: data da inclusão, usuário que incluiu, empresa que pertence o usuário que incluiu o aviso e título do aviso; 3a. Caso o ator não confirme a exclusão; 3a1. Sistema retorna para a página anterior.

Caso de Uso	**Incluir tema no fórum [CDU024]**
Descrição	O seguinte caso de uso mostra o processo de inclusão de temas no fórum.
Prioridade	Média
Estimativa	8 h
Atore(s)	Administrador e Usuário Cadastrado.
Pré-condição	Administrador ou Usuário Cadastrado logado no sistema.
Cenário de sucesso principal	1. Ator digita as seguintes informações: título, descrição e seleciona quais as empresas que terão acesso ao tema em questão; 2. Sistema inclui tema no fórum no sistema associado ao usuário logado que o incluiu e data de inclusão; 3. Sistema envia mensagem para o e-mail de todos as pessoas que terão acesso ao tema, com uma mensagem que existe um novo tema para discussão no *site* (com link para o *site*).
Extensões	4a. Problemas na inserção do tema no fórum; 4a1. Sistema mostra uma mensagem ao usuário logado informando que a inserção do tema não foi possível. 4b. Algum campo não foi preenchido; 4b1. Sistema mostra uma mensagem ao usuário logado informando que todos os campos deverão ser preenchidos.
Observações	1. O usuário poderá limpar os dados digitados e digitar as informações quantas vezes desejar. 2. Preenchimento obrigatório de todos os campos.

Caso de Uso	**Visualizar temas no fórum [CDU025]**
Descrição	O seguinte caso de uso mostra o processo de visualização dos temas postados no fórum.
Prioridade	Média
Estimativa	6 h
Atore(s)	Administrador e Usuário Cadastrado.
Pré-condição	Administrador ou Usuário Cadastrado logado no sistema.
Cenário de sucesso principal	1. Sistema busca todos os temas cadastrados que podem ser visualizados pelo usuário logado – de acordo com a empresa dele - e mostra a lista na tela, com as seguintes informações: data da postagem, usuário que postou o tema, o título, a quantidade de mensagens postadas para aquele tema e a empresa do usuário que inclui o tema.

Extensões	3a. Caso o ator deseje ler as mensagens postadas para um determinado tema da lista: 3a1. Ator seleciona o tema; 3a2. Sistema busca todas as mensagens postadas para o tema e mostra na tela com as seguintes informações: data da postagem, usuário que postou, o título e a descrição da mensagem.

Caso de Uso	**Postar mensagem p/ um tema no fórum [CDU026]**
Descrição	O seguinte caso de uso mostra o processo de postagem de uma mensagem para um determinado tema no fórum.
Prioridade	Média
Estimativa	8 h
Atore(s)	Administrador e Usuário Cadastrado.
Pré-condição	Administrador ou Usuário Cadastrado logado no sistema.
Cenário de sucesso principal	1. Incluir [CDU025] passo 3; 2. Ator seleciona o tema que deseja postar mensagem; 3. Ator digita a mensagem que deseja postar 4. Ator submete a mensagem; 5. Sistema inclui a mensagem associada ao tema, com as seguintes informações: usuário que o incluiu (usuário logado) e data de postagem. 6. Sistema envia mensagem para o e-mail de todos as pessoas que possuem acesso ao tema em questão, avisando que existe uma nova mensagem para tema X no *site* (com um link para o *site*).
Extensões	5a. Problemas na postagem da mensagem; 5a1. Sistema mostra uma mensagem ao usuário logado informando que a inserção da mensagem não foi possível. 5b. Algum campo não foi preenchido; 5b1. Sistema mostra uma mensagem ao usuário logado informando que todos os campos deverão ser preenchidos.
Observações	1. O usuário poderá limpar os dados digitados através de um botão "limpar" e digitar a mensagem quantas vezes desejar. 2. Preenchimento obrigatório de todos os campos.

Caso de Uso	**Excluir tema no fórum [CDU027]**
Descrição	O seguinte caso de uso mostra o processo de exclusão de um tema no fórum.
Prioridade	Média
Estimativa	4 h
Atore(s)	Administrador e Usuário Cadastrado.
Pré-condição	Administrador ou Usuário Cadastrado logado no sistema.
Cenário de sucesso principal	1. Ator seleciona o tema que deseja excluir e efetua exclusão; 2. Sistema exclui o tema selecionado mostrando uma confirmação para a exclusão solicitada.

Extensões	2a. Usuário logado: Administrador; 2a1. Sistema busca as empresas cadastradas; 2a2. Administrador seleciona opção; 2a3. Sistema busca todos os temas cadastrados de acordo com a opção selecionada e mostra a lista na tela ordenada pela data da inclusão com as seguintes informações: data da inclusão, usuário que incluiu o tema, empresa que pertence o usuário que inclui o tema, título e uma caixa de opções (para o ator selecionar qual o tema deseja excluir). 2b. Usuário logado: Usuário Cadastrado; 2b1. Sistema busca todos os temas incluídos pelo usuário logado que não possuem mensagens associadas e eles e mostra a lista na tela ordenada pela data da inclusão com as seguintes informações: data da inclusão, usuário que incluiu o tema e título. 3a. Caso acontecer algum problema com a exclusão do tema; 3a1. O sistema deverá mostrar uma mensagem ao usuário.

ANEXO 2 – Documentação do Sistema - Padrões de codificação e nomenclatura.

Padrões de codificação:

- Todos os arquivos .php deverão apresentar um cabeçalho contendo a descrição do mesmo, o programador responsável e a data da última atualização;
- A indentação do código deve ter 4 espaços SEM TAB;
- As estruturas de controle devem ter um espaço entre o nome da estrutura de controle e a abertura dos parênteses, para diferenciá-los de chamada de funções;
- Os nomes das funções, classes etc, devem ser escritos em letras minúsculas e no caso de nomes compostos, utilizar o caractere "_" entre as palavras;
- Funções devem ser chamadas SEM espaços entre o nome da função, os parênteses de abertura e o primeiro argumento. Os demais argumentos devem conter um espaço entre ele e a vírgula anterior;
- Utilizar espaços para alinhar verticalmente o sinal que atribui valor às variáveis;
- Comentar o código sempre que possível, utilizando o padrão /**comentário*/.

Padrões de nomenclatura:

- Os nomes dos arquivos .php deverão apresentar uma relação direta com o seu conteúdo; serem escritos em letras minúsculas, sem acento ou "ç" e quando compostos, utilizar o caractere "_" entre as palavras;
- Os nomes das tabelas do banco de dados deverão ser escritos sempre em letras maiúsculas; sem acento ou "ç"; sempre no singular; quando compostos utilizar o caractere "_" entre as palavras e apresentar relação direta com o objeto em questão. Exemplo: FUNCIONARIO;

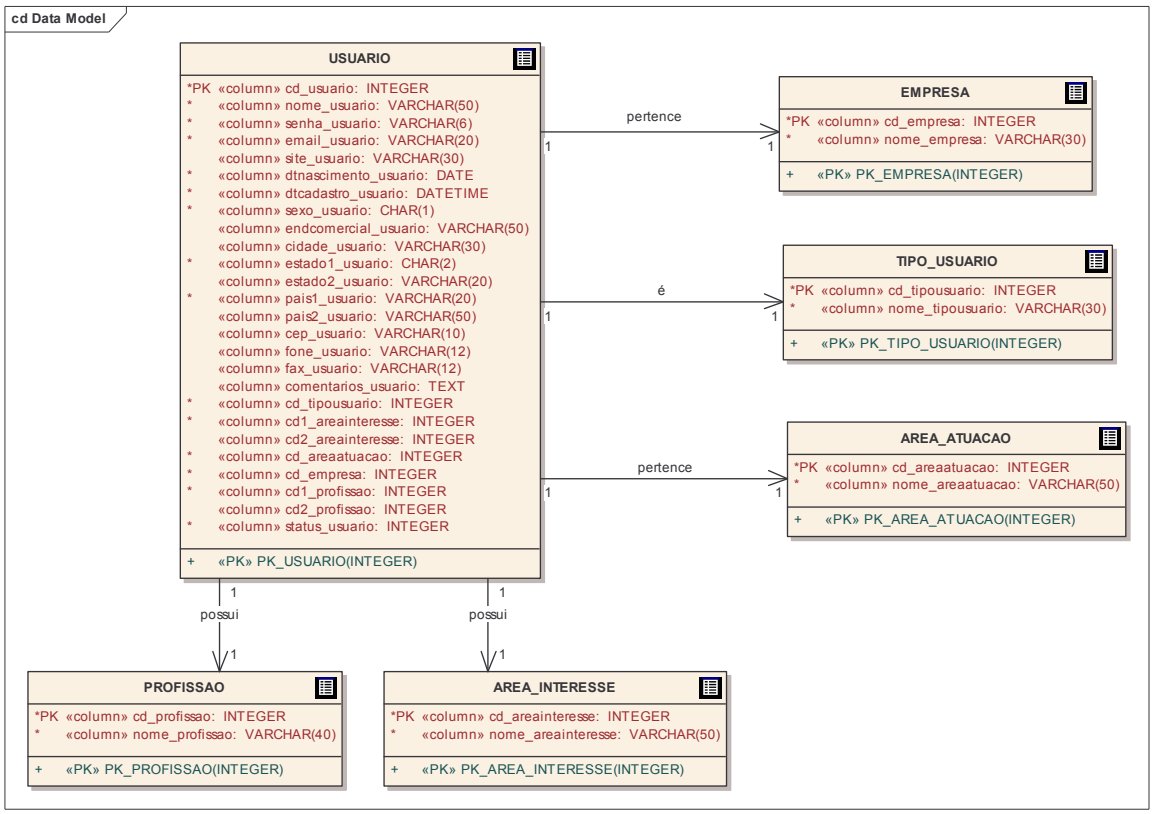
- No caso de tabelas dependentes de outras para a sua existência, referenciar a tabela pai no nome da entidade dependente. Exemplo: Tabela_Dependente: DEPENDENTE; Tabela_Pai: FUNCIONARIO; Tabela_Resultante: DEPENDENTE_FUNCIONARIO;

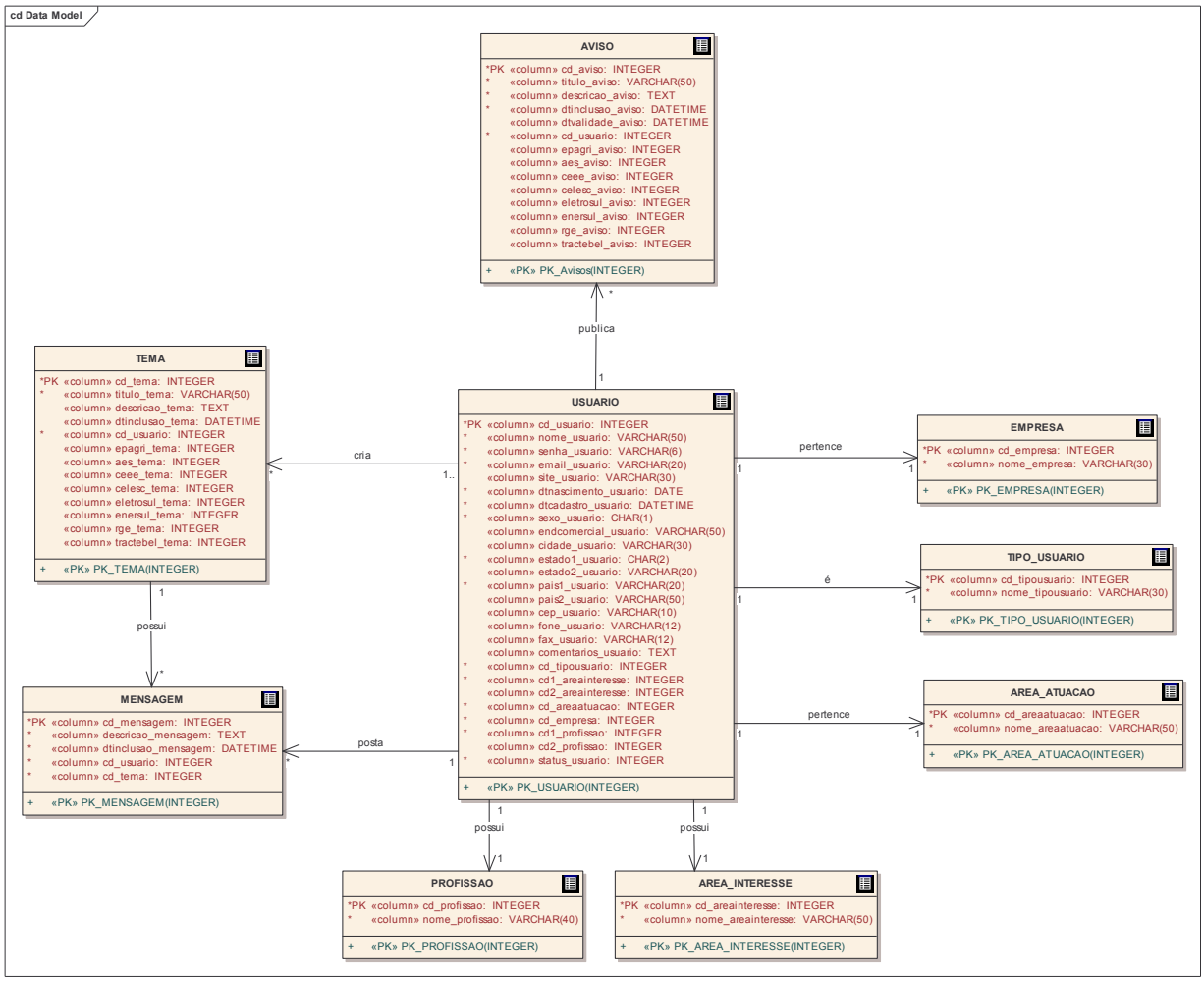
- No caso de tabelas surgidas de relacionamentos “muitos para muitos”, o objeto principal deve ser o verbo que define o relacionamento na sua forma substantiva, seguida dos nomes das tabelas que participam do relacionamento. Exemplo: Verbo: ALUGAR; Tabelas do Relacionamento: CLIENTE e FILME; Tabela_Resultante: ALUGUEL_CLIENTE_FILME.

- Os nomes das tabelas do banco de dados deverão ser escritos sempre em letras maiúsculas; sem acento ou “ç”; sempre no singular; quando compostos utilizar o caractere “_” entre as palavras e apresentar relação direta com o objeto em questão. Exemplo: FUNCIONARIO;

- As chaves estrangeiras devem conter os mesmos nomes utilizados na tabela de origem das mesmas.

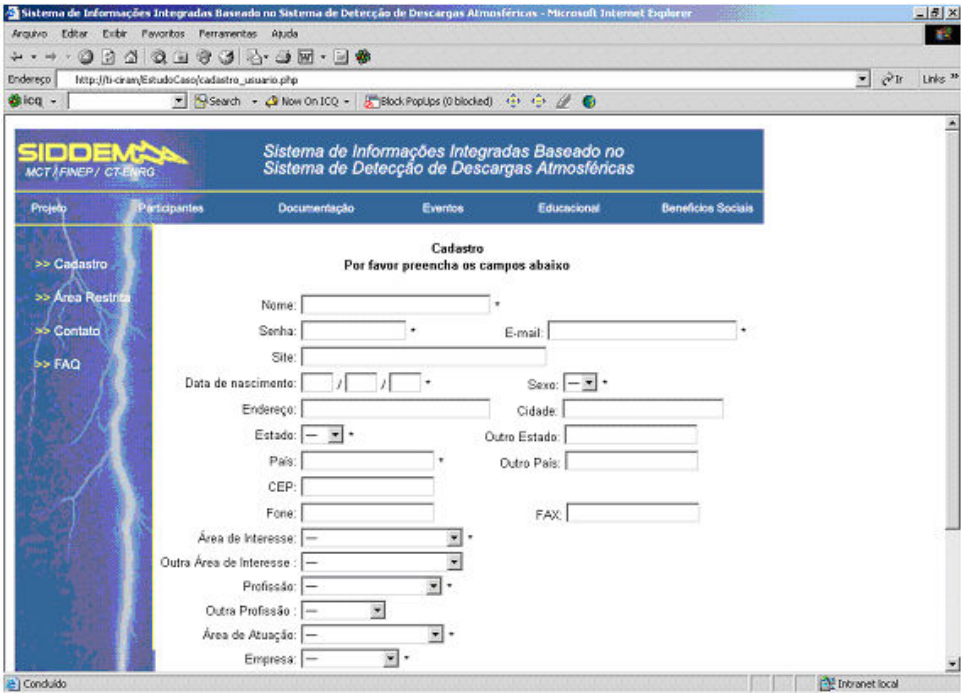
ANEXO 3 – Documentação do Sistema - Modelos de Dados



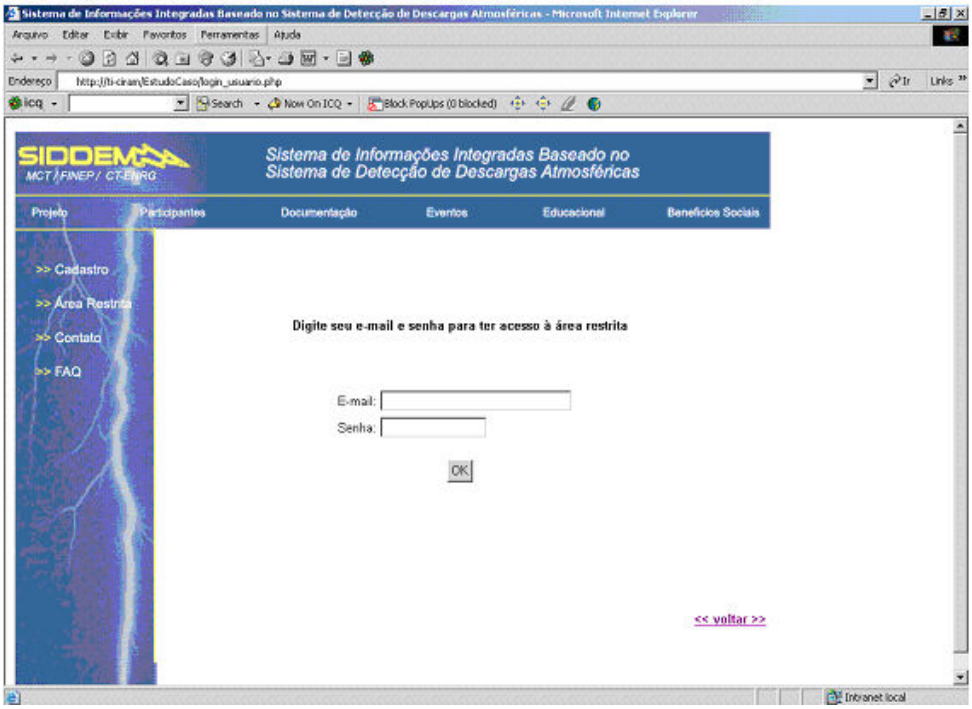

cd Data Model
AVISO
*PK «column» cd_aviso: INTEGER
* «column» titulo_aviso: VARCHAR(50)
* «column» descricao_aviso: TEXT
* «column» dtinclusao_aviso: DATETIME
«column» dtvalidade_aviso: DATETIME
* «column» cd_usuario: INTEGER
«column» epagri_aviso: INTEGER
«column» aes_aviso: INTEGER
«column» ceee_aviso: INTEGER
«column» celesc_aviso: INTEGER
«column» eletrosul_aviso: INTEGER
«column» enersul_aviso: INTEGER
«column» rge_aviso: INTEGER
«column» tractebel_aviso: INTEGER
+ «PK» PK_Avisos(INTEGER)
publica
TEMA
*PK «column» cd_tema: INTEGER
* «column» titulo_tema: VARCHAR(50)
«column» descricao_tema: TEXT
«column» dtinclusao_tema: DATETIME
* «column» cd_usuario: INTEGER
«column» epagri_tema: INTEGER
«column» aes_tema: INTEGER
«column» ceee_tema: INTEGER
«column» celesc_tema: INTEGER
«column» eletrosul_tema: INTEGER
«column» enersul_tema: INTEGER
«column» rge_tema: INTEGER
«column» tractebel_tema: INTEGER
+ «PK» PK_TEMA(INTEGER)
cria
USUARIO
*PK «column» cd_usuario: INTEGER
* «column» nome_usuario: VARCHAR(50)
* «column» senha_usuario: VARCHAR(6)
* «column» email_usuario: VARCHAR(20)
«column» site_usuario: VARCHAR(30)
* «column» dtnascimento_usuario: DATE
* «column» dtcadastro_usuario: DATETIME
* «column» sexo_usuario: CHAR(1)
«column» endcomercial_usuario: VARCHAR(50)
«column» cidade_usuario: VARCHAR(30)
* «column» estado1_usuario: CHAR(2)
«column» estado2_usuario: VARCHAR(20)
* «column» pais1_usuario: VARCHAR(20)
«column» pais2_usuario: VARCHAR(50)
«column» cep_usuario: VARCHAR(10)
«column» fone_usuario: VARCHAR(12)
«column» fax_usuario: VARCHAR(12)
«column» comentarios_usuario: TEXT
* «column» cd_tipousuario: INTEGER
* «column» cd1_areainteresse: INTEGER
«column» cd2_areainteresse: INTEGER
* «column» cd_areaatuacao: INTEGER
* «column» cd_empresa: INTEGER
* «column» cd1_profissao: INTEGER
«column» cd2_profissao: INTEGER
* «column» status_usuario: INTEGER
+ «PK» PK_USUARIO(INTEGER)
pertence
EMPRESA
*PK «column» cd_empresa: INTEGER
* «column» nome_empresa: VARCHAR(30)
+ «PK» PK_EMPRESA(INTEGER)
é
TIPO_USUARIO
*PK «column» cd_tipousuario: INTEGER
* «column» nome_tipousuario: VARCHAR(30)
+ «PK» PK_TIPO_USUARIO(INTEGER)
pertence
AREA_ATUACAO
*PK «column» cd_areaatuacao: INTEGER
* «column» nome_areaatuacao: VARCHAR(50)
+ «PK» PK_AREA_ATUACAO(INTEGER)
possui
MENSAGEM
*PK «column» cd_mensagem: INTEGER
* «column» descricao_mensagem: TEXT
* «column» dtinclusao_mensagem: DATETIME
* «column» cd_usuario: INTEGER
* «column» cd_tema: INTEGER
+ «PK» PK_MENSAGEM(INTEGER)
posta
possui
PROFISSAO
*PK «column» cd_profissao: INTEGER
* «column» nome_profissao: VARCHAR(40)
+ «PK» PK_PROFISSAO(INTEGER)
possui
AREA_INTERESSE
*PK «column» cd_areainteresse: INTEGER
* «column» nome_areainteresse: VARCHAR(50)
+ «PK» PK_AREA_INTERESSE(INTEGER)

ANEXO 4 – Algumas telas do sistema desenvolvido no estudo de caso.

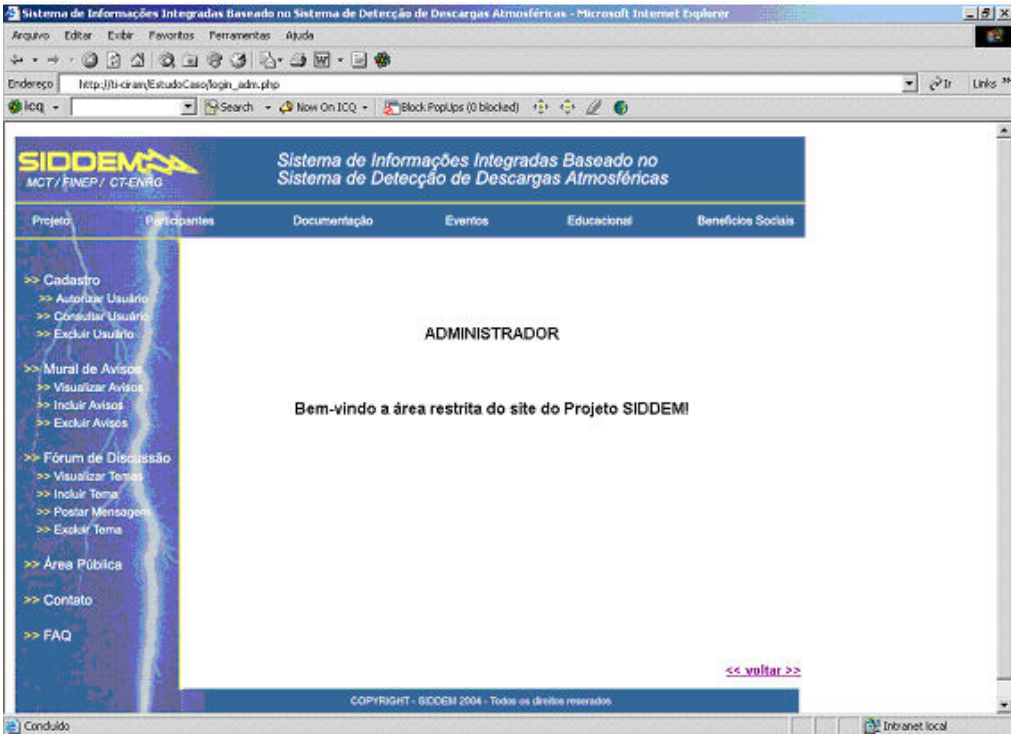
1 – Formulário para cadastro de usuários.



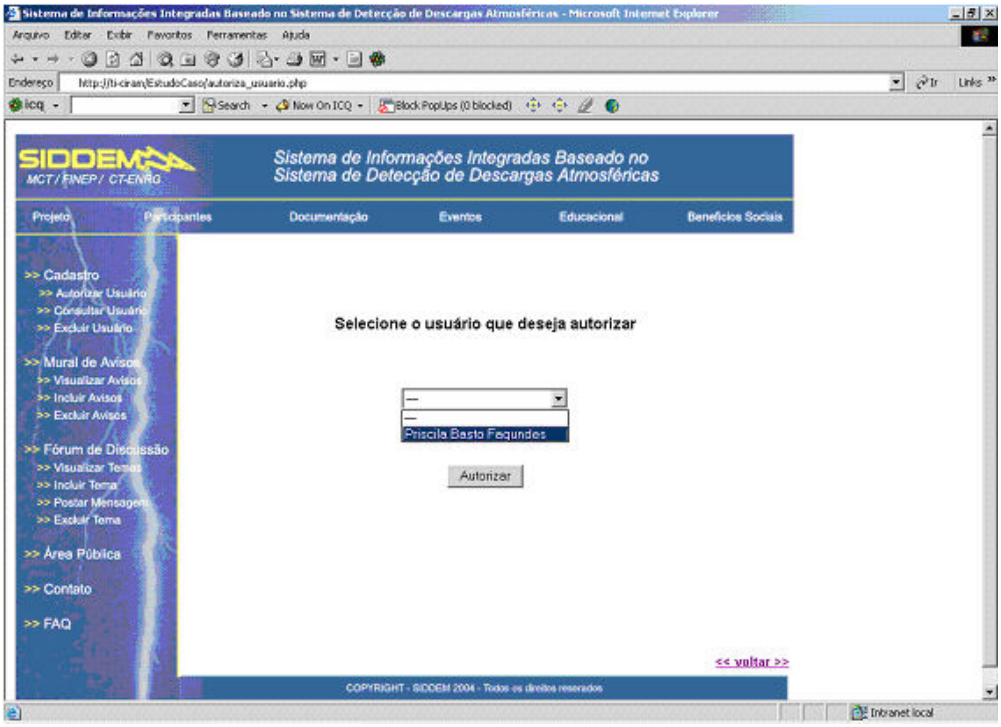
2 – Entrada dos dados para acesso à área restrita do *site*.



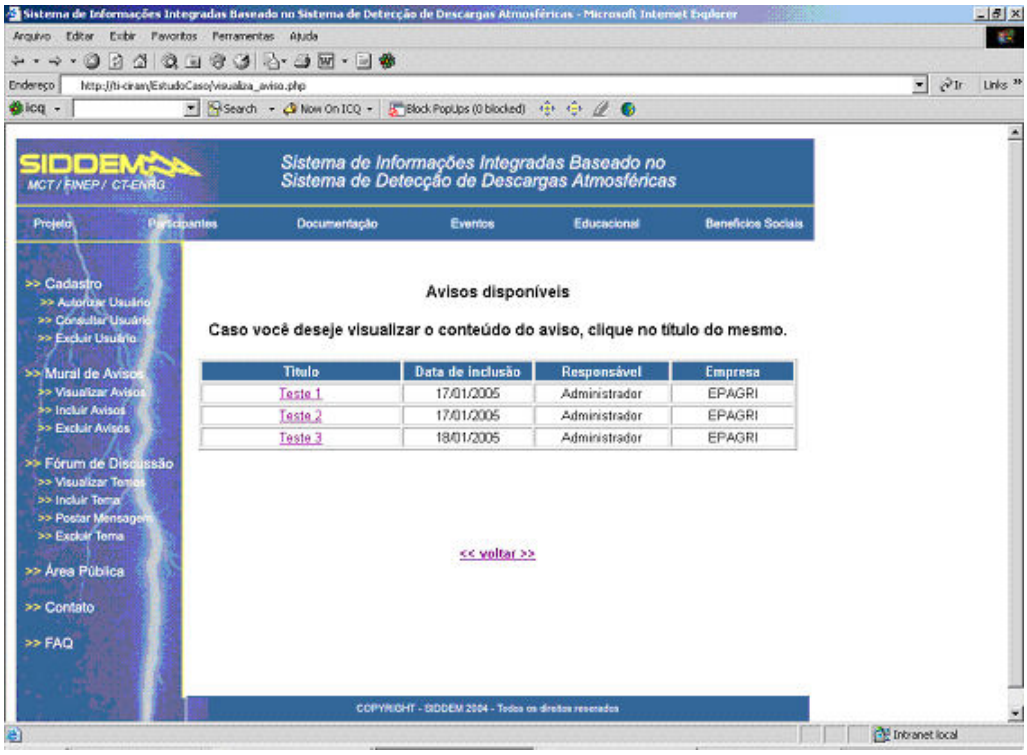
3 – Usuário Administrador logado no sistema



4 – Processo de autorização de um usuário para acesso à área restrita.



5 – Visualização dos avisos disponíveis no Mural de Avisos



6 – Formulário para inclusão de tema no Fórum de Discussão