FACULDADE DE TECNOLOGIA DE SÃO PAULO

Renato Guimarães Carvalho

Uso de práticas ágeis em projetos de Data Mart

São Paulo

2011

FACULDADE DE TECNOLOGIA DE SÃO PAULO

Renato Guimarães Carvalho

Uso de práticas ágeis em projetos de Data Mart

Monografia submetida como exigência
parcial para a obtenção do Grau de
Tecnólogo em Processamento de Dados
Orientador: Prof. Valter Yogui

São Paulo

2011

LISTA DE ABREVIATURAS E SIGLAS

BI-**B**usiness **I**ntelligence

DDL- **D**ata **D**efinition **L**anguage

DM(s)-**D**ata **M**art(s)

DW(s)-**D**ata **W**arehouse(s)

DWBA-**D**ata **W**arehouse **B**us **A**rquithecture

EDW-**E**ntreprise **D**ata **W**arehouse

ETL –**E**xtract **T**ransform **L**oad

OLAP- **O**n-**l**ine **A**nalytical **P**rocessing

OLTP- **O**nline **T**ransaction **P**rocessing

SGDB(s)-**S**istemas **G**erenciadores de **B**anco de **D**ados

TDD-**T**est **D**riven **D**evelopment

RESUMO

O objetivo deste trabalho é demonstrar uma série de práticas pertencentes a metodologias ágeis de construção de software que podem ser aplicadas ao desenvolvimento de um Data Mart,e como elas podem ser utilizadas neste contexto afim de se obterem diversas vantagens,como maior flexibilidade e diminuição de risco no projeto.Esta nova abordagem busca também contornar alguns problemas inerentes a este tipo de projeto,os quais as metodologias tradicionais ,enfrentam.Dentre destes problemas,destacam-se a dificuldade de se determinar os requisitos de início no projeto e a alta mutabilidade destes ao longo do projeto.

O trabalho explana a motivação que justifica a utilização de ágil dentro do contexto de desenvolvimento de BI,ou seja,quais as características deste cenário de desenvolvimento que tornam a agilidade de um processo de software,desejada,e em que pontos positivos essa agilidade impacta o resultado final do projeto.

Estas práticas fornecem um framework através do qual o processo de construção pode ser otimizado,ou seja ,não substituem as principais técnicas de se desenvolver Data Mart,mas sim,representam uma nova maneira de implementar estas técnicas ao longo do projeto.

Palavras-Chave:Data Mart ,Metodologias Ágeis,Data Warehouse,BI

ABSTRACT

The main goal of this work is to show a series of agile software development practices that can be applied to Data Mart development and how they can be used in this context in order to gain several advantages,like more flexibility and lessen project´s risk.This new approach also seeks to avoid some problems inherent to this kind of project,wich many tradicional methodologies face.Among these problems,stands the difficulty in determine the requisites right from the project´s start,and the high degree of change of them all along the project.

This work explains the motivation that leads to the use of agile in the Business Intelligence´s application development cenario,that is,which of this cenario´s aspects makes agility needed in the software´s delopment process ,and,how positively agility impacts the result of the project.

These practices provide an framework ,trough wich the contruction´s process can be optimized,but instead of replace the main techniques used to develop Data Marts,they represent a new way to implement those techniques along the course of the project.

Key Words:Data Mart ,Agile Methodologies,Data Warehouse,BI

SUMÁRIO

1. DATA WAREHOUSE

Imon (1991, pg. 54) define um Data Warehouse(DW) como sendo:

"Uma coleção de dados orientados a assunto, detalhados, integrados, não-voláteis e que variam temporalmente tendo como objetivo dar suporte ao processo de tomada de decisões gerenciais da empresa."

Um DW então pode ser entendido como uma base de dados centralizada, separada do ambiente transacional dos sistemas de onde são extraídos os dados para análise. O DW possui uma estrutura otimizada para fornecer um ambiente analítico de alta performance para as aplicações de Business Intelligence(BI).

Imon (1991) ressalta três características fundamentais de um DW:

Dados Integrados: O DW é alimentado por diversos sistemas de origem (sistemas legados), que na maioria das vezes não foram projetados para serem integrados, logo, os dados entram de maneira inconsistente no DW e através de diversos processos de transformação, formatação, sumarização, dentre outros, esses dados devem apresentar uma única "aparência" física a nível corporativo, ou seja, eles devem apresentar um formato comum consistente.

Dados Não-voláteis: Os dados em sistemas operacionais são atualizados freqüentemente, normalmente registro a registro com uma freqüência regular. DWs exibem um padrão de comportamento diferente já que os dados são carregados(normalmente em massa),acessados,porém,não são atualizados da mesma maneira que no ambiente operacional.Quando dados são carregados no DW,é realizado um "snapshot",ou seja,uma reprodução estática do estado atualizado das bases de dados que é armazenada subseqüentemente às demais atualizações,o que dá origem a um histórico de dados.

Outra característica marcante é a presença de elementos que representam tempo e variação no tempo dentro dos dados do DW. Essa variação indica que cada conjunto de dados no DW são precisos durante um dado intervalo de tempo, de modo que os dados armazenados ganham marcações de tempo para indicar isso.

Dados orientados a assuntos: Em contraste aos sistemas operacionais que são organizados ao redor das atividades do negócio, o DW é composto de dados que representam áreas de atuação ou de interesse do negócio, por exemplo, na área de varejo, áreas de assunto fundamentais seriam: produtos, pedidos, vendedores etc..

1.1 Objetivos do DW

Prover acesso fácil e rápido as informações de modo que a mesma possa ser manipulada pelo usuário final de diversas formas. O tempo de espera deve ser curto e as ferramentas utilizadas para visualização de uma query, por exemplo, devem ser simples.

A informação provida pelo DW deve ser consistente, isso significa informação com credibilidade e de alta qualidade. Em outras palavras, antes dos dados serem liberados para uso, devem ser “limpos” e cuidadosamente manipulados a fim de apresentarem alto grau de qualidade.

O DW deve ser resistente a mudança, sendo construído para lidar com elas. Essas mudanças podem ser tanto nos requisitos dos usuários, como provenientes da natureza do negócio ou da tecnologia empregada. Qualquer uma dessas mudanças não poderá invalidar os dados atuais no DW.

Por conter informações de alto valor para os negócios, o DW deve prover segurança ao acesso das informações de modo que haja um controle rigoroso sobre este aspecto.

O DW deve servir como base para a tomada de decisões na organização, para que estas agreguem valor ao negócio e fomentem o uso e a necessidade que os usuários possuem de acessá-lo.

A comunidade de usuários deve apoiar e aceitar o DW para a implantação do mesmo ter sucesso. Como o uso o DW difere do uso de sistemas transacionais, por esse ser opcional, o DW irá fracassar se após alguns meses os usuários abandonarem o uso do mesmo. A simplicidade de acesso é o fator chave para reter o interesse dos usuários. Imon (1991)

2. DATA MARTS

2.1 Introdução

Os Data Marts atendem as necessidades de unidades específicas de negócio ao invés de servir interesses organizacionais da corporação inteira. Eles otimizam a entrega de informação de suporte à decisão e se focam na gerência sumarizada de dados exemplificativos ao invés do histórico de níveis atomizados. Eles podem ser apropriados e gerenciados por pessoal externo ao departamento de informática das corporações.

A crescente popularidade da implementação de Data Marts se baseia em alguns fatores:

- Os Data Marts têm diminuído drasticamente o custo de implementação e manutenção de sistemas de apoio à decisão, o que os coloca ao alcance de um número muito maior de organizações
- Protótipos podem ser construídos rapidamente, com alguns pilotos sendo construídos entre 30 e 120 dias e sistemas completos sendo construídos entre 3 e seis meses
- Os data marts têm o escopo mais limitado e são mais identificados com grupos de necessidades dos usuários, o que se traduz em esforços concentrados

Os departamentos autônomos e as pequenas unidades de negócio freqüentemente preferem construir o seu próprio sistema de apoio à decisão via Data Marts. Muitos departamentos de informática estão vendo a efetividade desta abordagem e estão agora construindo o DW por assunto ou um Data Mart por vez, gradualmente, ganhando experiência no desenvolvimento.

Em relação aos DWs, a utilização de Data Marts se traduz em tempo e custos de desenvolvimento muito menores e é adequada quando se deseja implantar uma solução de BI onde a organização não dispõe de tempo para aguardar a implementação de um DW corporativo.

2.2 Definição

O conceito de Data Mart evolui do conceito de DW. A diferença fundamental entre os dois conceitos, para Kimball (2002), é o escopo, de modo que Data Marts são altamente

concentrados no suporte a decisão de processos singulares na empresa ou de uma linha de negócio,ao invés de abordar áreas de assunto a nível corporativo como sugere Imon(1991).

2.3 Métodos de desenvolvimento

Existem na literatura modelos utilizados para se relacionar os Data Marts, o DW e o modo de construção de ambos, sendo as abordagens utilizadas por Imon (1991) e Kimball (2002) as principais referências no assunto.

Podemos encarar inicialmente o Data Mart como sendo um subconjunto do DW que surge após sua implementação e possui o objetivo de servir de fonte de acesso as informações para os usuários finais de departamentos ou segmentos específicos da organização, através de interfaces no Front End por aplicações de análise de dados. Os data marts nessa visão são completamente dependentes do DW,já que extraem seus dados da área de staging do mesmo e são otimizados para fornecer alta performance utilizando dados altamente agregados.

Esse tipo de implementação caracterizada como Top-Down é exposta por Imon (1991), e baseia-se inicialmente em um modelo corporativo dos dados, onde são delimitadas e modeladas diversas áreas de assunto dentro da empresa, para se construir primeiro o DW integrando os dados e posteriormente sendo desenvolvidos os Data Marts.

Outro modelo é exposto por Kimball (2002), onde identificamos uma abordagem Bottom-Up, ou seja, os DMs são vistos como sendo independentes em relação ao DW, de modo que eles são desenvolvidos previamente como se fossem pequenos DMs do ponto de vista estrutural e depois integrados.

Após o desenvolvimento iterativo e evolutivo dos Data Marts, há o crescimento e posterior integração destes, dando origem, no modelo de Kimball (2002), ao EDW. Vale ressaltar que esse tipo de construção,para Imon(1991) leva a redundâncias devido ao crescimento não planejado dos DMs.No trabalho de Kimball(2002) é descrito um modelo denominado como Data Warehouse Bus Architeture que tem como objetivo guiar o desenvolvimento dos DMs,eliminando esse tipo de problema e servindo como base para a integração e escalabilidade do EDW,ele será descrito a seguir.

Ainda há uma abordagem de desenvolvimento híbrida, onde os DMs não necessitam de esperar o DW ser completamente implementado para serem acessados, porém, seu

desenvolvimento segue um modelo de dados da organização que expressa à visão organizacional que existe dos diversos DMs a serem construídos. Este modelo serve como uma fundação para o desenvolvimento do DW garantindo que gaps de informação ou redundâncias sejam planejadas e catalogadas ao passo que o projeto avança. Vale ainda ressaltar que a independência dos Data Marts, nesse modelo, é tida como temporária, de modo que o DWs ira englobar os DMs assim que integrado, alimentando-os e determinado sua criação e evolução.

Este modo de desenvolvimento trata da necessidade que os usuários possuem por informação imediata ou em curto prazo, e que não pode esperar pela demorada implementação do DW e também do problema de crescimento não planejado da abordagem Bottom-Up

2.4 Data Marts e a Data Warehouse Bus Architecture

Segundo Kimball (2002), O DWBA é uma abstração ou modelo, que representa o conjunto de processos dos negócios conduzidos pela organização e é composto de definições padrões de tabelas fato e dimensões em conformidade. A idéia é de que Data Marts tratam de processos individuais do negócio e utilizarão as tabelas fatos e dimensões apresentadas no modelo, conforme o necessário.

Os processos apresentados neste modelo, são as atividades que geram valor para a organização, logo, para descrever as propriedades dos processos e poder mensurá-los são desenvolvidos os Data Marts, um de cada vez, sendo que a união desses irá resulta no DW.

Data Marts são compostos de tabelas fato e dimensões, sendo que algumas destas dimensões podem ser necessárias em mais de um Data Mart para representar um processo.

As tabelas que precisam ser compartilhadas precisam então, estar em conformidade entre os Data Marts. A tabela a seguir representa um exemplo dessa relação entre processos e

dimensões:

Processos do negócio	**Dimensões em conformidade**		
	Produto	Vendedor	Cliente
Vendas	X	X	X
Estoque	X		
Entregas	X		X

Para Kimball (2002), ao analisarmos o ambiente corporativo para construir um Data Mart devemos evitar a delimitação de barreiras entre os departamentos ou áreas da empresa sendo que os Data Marts devem ser organizados baseando-se primariamente em processos do negócio, como pedidos, compras etc. como é exibido na matriz de exemplo.

Kimball (2002) faz essa objeção, pois, segundo ele, diversos setores do negócio desejam freqüentemente analisar as mesmas métricas resultantes de um mesmo processo de negócio, logo, se os Data Marts forem criados tomando como bases departamentos que utilizam processos similares, haverá dados redundantes e esforço de ETL desnecessários. A abordagem de Kimball nesse aspecto, objetiva evitar a duplicação de mensurações feitas nestes processos.

Na visão de Kimball (2002) os Data Marts não representa somente dados sumarizados, de modo que dados altamente granulares (com muitos detalhes) devem permear o Data Mart. As sumarizações servem ao propósito de otimização de performance, mas não como substitutos aos dados detalhados que permitem ao Data Mart responder as requisições analíticas inesperadas dos usuários, o que caracteriza uma flexibilidade que um Data Mart composto somente de dados agregados não apresentaria.

2.5 Estrutura do Data Warehouse/Data Marts

Kimball (2002) descreve uma arquitetura de DW/DM composta por diversas camadas: Sistemas operacionais (transacionais), área de “Staging” de dados, área de apresentação de dados, e ferramentas de acessos aos dados, segue uma descrição resumida de cada uma delas:

Sistemas operacionais: É todo o ambiente externo ao DW, que servem como fonte de dados para extração e posterior carga no DW. Esse ambiente é composto dos sistemas "legados" ou transacionais, que armazenam os dados na forma mais primitiva (detalhada, precisa e não redundante).

Estes sistemas, na maioria das vezes não foram projetados para ter seus dados integrados e cabe aos processos de ETL tratar os dados e integrá-los, esse procedimento é feito em uma camada da arquitetura do DW denominada de Data Staging Area (área de estágio de dados).

Data Staging Area: Corresponde a área onde todos os dados provenientes dos sistemas operacionais são tratados para serem enviados a área de apresentação de dados.

Kimball (2004), afirma que a área de staging e a área de apresentação de dados são: física, lógica e administrativamente separadas a fim de evitar que o time de ETL tenha que prover segurança de acesso em nível de linha de código, por exemplo.

Na área de Staging são realizadas quatro principais atividades que compõe o processo de ETL (Extração, transformação e carregamento de dados), segundo Kimball (2004):

(1) Extração: Extrair dados significa que a aplicação devera ler e entender os dados para então colocá-los na área de estágio para maiores modificações como corrigir conflitos domínio, lidar com elementos ausentes, corrigir nomenclatura de elementos ou colocar os dados em um formato especifico.

Os dados não-tratados são transferidos para armazenamento com um mínimo de reestruturação, porém sem que transformações significativas tenham ocorrido ainda. Dados dos sistemas legados (como por exemplo, XML) são transcritos para arquivos ou bases de dados relacionais nessa etapa. Esse baixo nível de transformações garante que o processo de extração seja rápido e simples e permite maior flexibilidade para reiniciar o processo caso haja alguma interrupção.

(2) Limpeza: Muitas vezes os dados dos sistemas de origem apresentam uma qualidade inferior ou não aceitável ao requerido pelo DW. Essa etapa é composta pelas atividades que garantem a qualidade dos dados extraídos, dentre elas:

- Checar a consistência de valores
- Remover Duplicações
- Garantir conformidade com as regras de negócio

(3) Colocar dados em conformidade: Conformidade dos dados é uma característica requerida quando diversas fontes de dados são "misturadas" em um DW. Queries não podem acessar essas fontes separadas simultaneamente ao menos que dados textuais possuam um nome comum ou dados referentes a mensurações numéricas tenham sido racionalizados matematicamente para que as diferenças existentes ao se integrar os dados não sejam discrepantes a ponto de invalidar a informação apresentada.

Conformar os dados requer um padrão a nível corporativo a ser estabelecido e seguido a fim normalizá-los.

(4) Entregar dados para o Front Room: O objetivo principal da área de staging, também citada por Kimball (2004) como "Back Room", é deixar os dados preparados para a execução de Queries. Segundo Kimball (2004), o produto final entregue pelos processos são os dados estruturados fisicamente, de maneira "simétrica", em esquemas dimensionais denominados esquemas estrela.

Para Kimball (2004), estes esquemas dimensionais são requeridos por diversas ferramentas de análise de dados e são fundamentais para a construção de cubos OLAP, reduzindo o tempo de resposta das Queries e simplificando o desenvolvimento da aplicação. O conjunto destes esquemas compõe a área de apresentação de dados.

Há uma discussão, quanto ao modo como os dados são armazenados na área de staging, que diz respeito à necessidade de usarem estruturas normalizadas dentro da área de staging do DW, seja ela proveniente da aplicação ou resultante de transformações.

Kimball (2004) ressalta que o objetivo final da área de estágio não é produzir estruturas normalizadas, apesar de as mesmas poderem ser utilizadas, com o evidente prejuízo de serem necessários maiores recursos, pois, como o seu método de desenvolvimento utiliza modelagem dimensional, ou seja, estruturas dimensionais são utilizadas na área de apresentação de dados, há um re-trabalho no processo de transformação de dados para normalizar estruturas e para montar a estrutura dimensional na área de apresentação. Ele ainda aponta que existem soluções alternativas de staging que despendem menos custos e possuem maior eficiência do que normalizar estruturas na área de estágio se estas não existirem.

> Há ainda a ressalva de que estruturas normalizadas devem ser mantidas longes de Queries (devido a baixa performance resultante) e que a área de apresentação é estritamente dimensional.Kimball(2002)

Área de apresentação de dados: É o DW/DM na visão do usuário, ou seja, tudo que os usuários enxergam e conseguem acessar através das ferramentas de manipulações de dados.

Kimball (2002) reforça a idéia de que os dados devem ser armazenados, acessados e apresentados em esquemas dimensionais. Este tipo de abordagem, segundo ele, permite criar modelos de negócio simples e intuitivos.

Ambos os tipos de modelagem de dados, o dimensional e a 3FN podem ser representados em modelos ER, pois ambos são formados por tabelas relacionadas por Joins. O que os diferencia, no entanto, é o nível de normalização. Bancos de dados normalizados atendem muito bem as necessidades de sistemas transacionais, pois permitem que, por exemplo, uma atualização toque a base de dados em um único local. No entanto este tipo de modelagem já não é tão adequado ao DW, pois além de o esquema ter uma complexidade alta, SGDBs relacionais não conseguem executar Queries eficientemente contra estruturas normalizadas, o que resulta em queda de performance.

Fatores fazem com que o uso de estruturas normalizadas contrariem o próprio sentido de existência do DW: alto desempenho de acesso a dados de forma simples e intuitiva.

Os objetivos da modelagem dimensional são então: facilidade no entendimento do modelo, resistência a mudanças e performance alta nas Queries.

Vista a importância do papel que modelos dimensionais exercem no método de desenvolvimento de data marts de Kimball (2002) segue uma descrição desse tipo de modelagem.

2.6 Modelagem dimensional

Ao visualizarmos a exibição de dados na área de apresentação, estes estarão estruturados de uma forma dimensional. Para isso será utilizada a técnica de modelagem dimensional.

A modelagem dimensional é utilizada quando se há necessidade em colocar o entendimento do modelo e a performance como prioridades do design de um esquema de banco de dados e é caracterizada também por ser um modelo que acomoda mudanças com facilidade. Kimbal (2002)

Esse tipo de modelagem se adéqua ao cenário de desenvolvimento de DW, pois, como os requisitos não são claros, há necessidade de se entender como o usuário final enxerga o negócio, para que a solução possa efetivamente auxiliar na tomada de decisão.

O modelo dimensional permite representar a visão do negócio que o usuário possui através da identificação de elementos familiares a eles, que se traduzem em métricas (ou mensurações) de processos organizacionais. Estas métricas são utilizadas pelos próprios usuários no dia-a-dia para indicar a performance dos processos que existem dentro da organização. Ponniah (2007)

Ao analisarmos desempenho, a modelagem dimensional fornece um modelo com poucas tabelas (muito menos que no modelo ER), sendo assim, querys são executadas de maneira mais eficiente por requerem uma quantidade de Joins reduzida.

Ao relacionarmos a modelagem dimensional com o nível de detalhamento das informações apresentadas pelo Data Mart vale lembrar que o objetivo não é fornecer somente um conjunto de informações pré-sumarizadas, pois isso iria remeter a uma baixa adaptabilidade a mudanças de requisitos (usuários confrontarem o DM com querys inesperadas), mas sim informações com alto detalhamento complementadas por agregações que são realizadas ao se identificar quais informações o usuário necessita ver agrupadas de forma freqüente. Kimball (2002)

Os principais componentes deste tipo de modelo são fatos, dimensões e atributos destas dimensões.

A modelagem dimensional procura cercar os fatos com o maior número de informações contextuais (dimensões).

2.6.1 Fatos

As tabelas de representação de fatos, nada mais são do que mensurações de algum processo do negócio. Estes dados dão suporte a operações matemáticas utilizadas para analisar o processo em si.

Uma tabela de fatos é a intersecção de diversas métricas utilizadas para avaliar um determinado processo (ex: quantidade vendida, produto vendido, loja em que foi vendido) que resulta em uma lista de dimensões descritivas do processo em si.

Tabelas fatos são ricas em linhas, porém possuem poucas colunas, e vale observar, segundo kimball (2002), que operações de DW freqüentemente retornam uma quantidade muito grande de linhas fato, logo agregar tais linhas se torna uma prática importante ao se deparar com tal quantidade de registros.

Outra característica importante do modelo dimensional, que leva a necessidade de se agregar tais dados, diz respeito às tabelas fatos ocuparem aproximadamente 90% do tamanho de armazenamento, sendo que, valores numéricos e aditivos são os mais úteis neste tipo de abordagem por serem facilmente manipulados. kimball(2002)

Podemos caracterizar quando a possibilidade de agregação, os dados das tabelas fatos como: Aditivos, semi-aditivos e não-aditivos

2.6.2 Dimensões

As dimensões são descrições detalhadas de cada elemento na tabela de fatos (cada mensuração do processo) sendo tabelas ricas em atributos (colunas) e ligadas à tabela de fato por chave estrangeira.

Segundo kimball (2002) o poder de um DW esta diretamente relacionado à profundidade das tabelas de dimensões.

Ao se identificar fatos e dimensões é necessário uni-los, o que resulta nos chamados esquemas estrela.

2.6.3 Esquemas Estrela

Ao definirmos quais serão as tabelas fato e as respectivas dimensões é necessário uni-las. A estrutura que possibilita este tipo de ligação são os esquemas em estrela.

Existe uma relação, para Imon (1991), que diz respeito ao volume de dados presentes em tabelas fato ser muito maior, ou condensar muito mais dados do que as tabelas dimensão, o que leva a uma heterogeneidade implícita no modelo dimensional e por conseqüência isso é caracterizado como sendo uma "desvantagem" ao se modelar dessa forma, tratando-se de data warehouses.

Imon (1991) ressalta que essa forma de modelo é propícia quando se desenvolvem os Data Marts, pois estes são focados em requisitos de negócio de um departamento ou processo em particular, logo, assim que estes requisitos são determinados, uma estrutura em estrela é construída, e esta é otimizada para atender esses requisitos específicos. Porém,quando se trata de DWs,o modelo e a implementação não tem como foco serem otimizadas para um determinado subconjunto de requisitos dentro da organização, sendo que a natureza destes é corporativa, portanto, um modelo dimensional, na visão de Imon, não é adequado, pois privilegia a eficiência de uma área de atuação em particular da organização somente.

Visto isso, os esquemas estrela podem ser enxergados como estruturas centralizadas por uma tabela fato que se liga a diversas tabelas de dimensões através de chaves estrangeiras. Para montar o esquema é necessário identificar os relacionamentos entre as tabelas dimensão e a fato. Kimball (2002) recomenda o uso de chaves sintéticas (Surrogates Keys) a fim de substituir as chaves "naturais" ou provenientes dos sistemas de origem. Essa prática consiste em colocar seqüência numéricas (inteiros) como identificadores chave nas tabelas, para obter uma maior performance ao se responder querys, tornar o modelo resistente a mudanças inerentes a chaves advindas dos dados dos sistemas transacionais e possibilitar uma integração de dados mesmo que estes possuam, originalmente, chaves pouco consistentes.

3. METODOLOGIAS ÁGEIS

Highsmith (2009) atesta que nos encontramos em um cenário de grande turbulência dentro do mundo corporativo, o que leva as organizações a criarem a necessidade de se adaptar a mudanças impostas pelo mercado ou agentes externos de forma cada vez mais rápida e dinâmica, do mesmo modo que elas precisam introduzir mudanças neste ambiente a fim de desestabilizar concorrentes e obter vantagens competitivas no mercado.

O cenário a que Highsmith (2009) se refere influi no modo que as empresas se comportam no que se refere a processos e estratégias de negócio. Essas empresas buscam o dinamismo e a velocidade de resposta aos estímulos do mercado,o que leva conseqüentemente a necessidade de soluções informatizadas que possam ser aplicadas em meio a esse ambiente altamente "caótico" Highsmith(2002) .No entanto,apesar de existir uma demanda crescente de tais soluções,as dificuldades de implantação de tais projetos é cada vez maior.Estudos realizados pelo Standish Group(1995),sobre diversos projetos de software mostrou que a realidade dos projetos de software é caracterizada por uma alta taxa de projetos cancelados(31%) e custos de desenvolvimento que extrapolam o que foi projetado(53% dos projetos),devido a diversos fatores dentre os quais sempre estiveram presentes:má especificação dos requisitos dos usuários e pouco envolvimento com o cliente.

Em meio a tal contexto, fica claro que é cada vez mais freqüente a existência de situações onde diversos elementos tornam as necessidades do negócio inerentes a sistemas de informação altamente mutáveis e muitas vezes difíceis de determinar a priori. Ao analisarmos a área de desenvolvimento de software usando uma abordagem metodológica,diversas formas de conduzir projetos já foram utilizadas, afim de se entregar produtos de forma mais rápida,com maior qualidade e mais baratos Salo(2006).

Essa busca por melhores meios de se projetar aplicações foi sempre motivada por um cenário onde o desenvolvimento de software era tido como uma atividade que poderia ser abordada sistematicamente de forma similar a engenharia, através de processos definidos e repetíveis, o que não é mais aceitável visto o padrão de comportamento do mercado e a taxa de fracassos em implantações de projetos de software. Highsmith (2002)

Houve a necessidade, então, de se abordar a construção de software de um modo que permitiria aos envolvidos nos esforços de projeto lidar com este cenário, a fim de se criar e evoluir produtos de software com qualidade e que agregassem valor ao cliente.

Dentro deste ambiente surgiram as metodologias ágeis, que encaram o desenvolvimento de software como sendo uma atividade não caracterizada por ser um processo definido, ou seja, onde os resultados não podem ser mensurados de forma sistemática e que visam permitir ao projeto se adequar a ambientes de desenvolvimento, especialmente onde requisitos são voláteis ou onde esta se adotando uma nova tecnologia. Highsmith (2002)

3.1 **Definição de ágil**

Diversos autores definem o termo agilidade, para Highsmith (2009,p.45) agilidade é:

"A habilidade de se criar e responder a mudanças a fim de se lucrar um cenário de negócios turbulento e de obter o equilíbrio entre flexibilidade e estabilidade"

Cockburn(2000,p.149) afirma:

"Ser ágil implica em ser eficiente e dirigível. Um processo ágil é tanto leve como suficiente,a leveza é um meio de se manter dirigível,a suficiência é um meio de se manter no jogo".

Agilidade, em desenvolvimento de software pode ser considerada, tendo como base as duas definições acima, como sendo a habilidade de reação e adaptação em relação ao nível de mudanças no ambiente somada a capacidade de se entregar em curtas janelas de tempo produtos que agreguem valor aos usuários finais. Um processo ágil está sempre preparado para atingir um determinado grau de adaptabilidade. Vale ressaltar que, ao falarmos em ágil não tratamos somente da ênfase na velocidade de entrega, mas principalmente na flexibilidade que é combinada junto a ela,ao utilizarmos este tipo de abordagem.

Outro ponto importante ao se definir ágil é a mudança do foco do projeto, que passa da conformidade com requisitos detalhados para ser a entrega de valor contínua ao usuário. Segundo Highsmith (2009), em projetos ágeis, o escopo do projeto não é fixo e pode ser constantemente ajustado se esta mudança justificar o valor que irá ser agregado. Fowler (2005) afirma que a própria natureza da construção de software, é caracterizada por ser uma atividade intensiva em design, o que torna a estimativa de custos e prazos uma tarefa árdua. Ainda ressalta que o desenvolvimento de software é um processo fundamentalmente dependente de indivíduos, uma variável dificílima de quantificar e que não é levada em conta em abordagens baseadas puramente em princípios de engenharia.

3.2 Processos Preditivos e Adaptativos

Processos, no âmbito de desenvolvimento de software são definidos por Humphrey (1995) como sendo a seqüência de passos necessária para se desenvolver e manter software e que tem como objetivo proporcionar um framework técnico e gerencial que permita o uso de métodos, ferramentas e pessoas na atividade de construção.

Abrahamsson. P(2002,pg.10) mostra uma comparação,exibida na tabela a seguir, entre o enfoque metodológico tradicional que ao longo do tempo obteve "prestígio" por ser considerado a forma mais adequada de se desenvolver software, e o enfoque que segundo os autores foi tido como "marginalizado" ao se tratar de desenvolvimento de software,em especial os processos utilizados:

O Desenvolvimento de sistemas é:	
Abordagem Privilegiada (Tradicional)	**Novas Abordagens**
Um processo controlável e gerenciável	Processo com nível de aleatoriedade dirigido por acidentes e oportunidades
Processo linear e seqüencial	Processos simultâneos, que se intercalam e onde existem Gaps
Um processo universal e replicável	Processo repleto de particularidades e características únicas
Um processo racional, determinado e orientado a objetivos	Negociável, maleável e baseado em compromissos

Ao analisar esse comparativo percebemos que tradicionalmente existe uma concentração em processos que devem seguir conformidade a planos. Cockburn(2000) afirma que essa abordagem parte do pressuposto de que a atividade de construção de software se encontra em um nível "Racional" ao caracterizarmos a metodologia de desenvolvimento,isto é,baseadas em métodos e técnicas como se é utilizado nas disciplinas de engenharia e análise de sistemas.Porém o desenvolvimento de software ,para ele,se encontra em um estágio onde uma abordagem metodológica "Heurística" é a mais adequada,ou seja,baseada no aprendizado

e lições contínuas que devem levar,conforme o conhecimento cresce,a se estabelecerem soluções padronizadas para problemas determinados.Highsmith(2002) também comenta o privilégio que é dado ao aspecto do controle em processos tracionais ao invés do enfoque na execução.Para ele, muitas vezes,quando se assume que planos são estritamente corretos,a atividade de controle passa a servir ao propósito corretivo de erros e discrepâncias afim de justificar por que tais situações foram encontradas,ao invés de servir como instrumento de aprendizado e evolução do processo.

Segundo Abrahamsson(2002),tradicionalmente,a condição fundamental de aplicabilidade de uma metodologia linear é a determinação completa dos requisitos antes das atividades de design e implementação se iniciarem,o que não é passível de se fazer em todos os tipos de projetos.Martin Fowler caracteriza esse tipo de processo como sendo preditivo.

Para Highsmith (2009), essa concentração na especificação detalhada de requisitos define, em processos tradicionais, o escopo do projeto, o qual permanece fixo ao longo do desenvolvimento, o que por conseqüência leva a dificuldade de adaptação do processo quando outras duas variáveis são adicionadas como fixas: prazos e custos.

Uma abordagem ágil utiliza processos flexíveis e adaptativos, como definidos por Martin Fowler (2005), que permitam aos desenvolvedores modificarem as especificações ao longo do projeto já que requisitos são mutáveis.

O processo iterativo é direcionado a responder a mudanças no ambiente e deve permitir a entrega de valor ao cliente rapidamente de forma incremental. Segundo Highsmith(2009),as práticas utilizadas para se alcançar tal objetivo são:

- Desenvolvimento iterativo e incremental
- Iterações dirigidas à entrega de novas funcionalidades (features)
- Utilização de limites de tempo para cada iteração (Timeboxing)

3.3 Desenvolvimento iterativo e incremental

Para se adotar um processo adaptável, as metodologias ágeis se utilizam da prática de iterações dentro do processo de desenvolvimento.

O desenvolvimento iterativo e incremental não é uma prática recente no universo da engenharia de software. Dentre diversos autores,um exemplo de precursor a este tipo de

desenvolvimento é Boehm,que em 1980 sugeriu o método Espiral que se baseava em iterações.

De acordo com Larman (2004) uma iteração pode ser considerada como sendo um mini-projeto, no qual as atividades de análise de requisitos, design, implementação e teste são conduzidas a fim de se produzir um subconjunto da versão final do sistema. Ao fim de cada interação é entregue uma versão parcial do sistema,testada ,integrada e estável.A parte incremental do processo,envolve a ação de adicionar funcionalidades a um sistema a cada entrega feita nas diversas iterações.

Larman (2004) aponta que, ao se utilizar iterações, o modo de se determinar requisitos de software também muda. Ao invés de serem especificados rigidamente no início do projeto eles são descoberto e refinados ao longo das interações, sendo que as iterações iniciais usualmente são responsáveis por revelar os requisitos de grande valor para o negócio, os quais incidem em um alto grau de impacto na arquitetura do sistema.

Entrega de funcionalidades (Features): A abordagem de desenvolvimento por Features ou funcionalidades, iniciou-se a ser utilizada na metodologia denominada FDD (Feature Driven Development) idealizada por Jeff De Luca em 1997.

Features são funcionalidades que o software apresenta e que possuem valor sob a ótica do usuário, que as enxerga pela perspectiva do negócio. Dentro das diversas iterações realizadas dentro do projeto,são implementadas novas funcionalidades de forma incremental afim de se entregar de forma freqüente partes ou subconjuntos do sistema. As features são basicamente decomposições ao nível mais simples possível, dos requisitos expressos pelos usuários.

As Features devem ser simples, pois sua implementação deve respeitar o tempo limite da iteração,caso contrário,deve-se aplicar maior decomposição ao se analisar o problema em questão.

O uso de features tem por objetivo permitir demonstrar aos usuários o progresso obtido ao longo do desenvolvimento, ao mesmo tempo em que se obtém o feedback por parte deles em relação às novas funcionalidades implementadas. Dessa forma,o usuário consegue perceber melhor suas necessidades ao utilizar aquela determinada parte do sistema o que muitas vezes leva a uma evolução do entendimento e por conseqüência,dos requisitos do projeto.

Essa evolução leva conseqüentemente a adaptação do processo de desenvolvimento com base na re-alimentação e contrasta com a definição rígida dos requisitos logo no início do projeto encontrada nas metodologias tradicionais.

A utilização desse modo de desenvolvimento permite, para Highsmith (2009) estabelecer uma “interface” entre desenvolvedores e usuários, onde os usuários priorizam as funcionalidades que são mais importantes sobre a óptica do negócio e os desenvolvedores determinam: quais tarefas deverão ser realizadas a fim de entregá-las e quais recursos serão consumidos para satisfazer restrições de prazo e custo.

Segundo Beck (1990), uma abordagem iterativa e baseada na entrega de funcionalidades em curtos períodos de tempo reduz o risco em projetos de software, de modo que funcionalidades de alto valor para o negócio são implementadas de forma prioritária e submetidas à avaliação do usuário-final o quanto antes.

4. ÁGIL EM PROJETOS DE DM/BI

4.1 Por que usar ágil nesse tipo de projeto?

Projetos de DM/BI apresentam diversas características que os tornam particularmente complexos:

- É necessário uma equipe com um leque de habilidades amplo (modelam de dados, desenvolvimento de ETL, limpeza de dados, design OLAP, dentre outras)
- As organizações tendem a almejar o desenvolvimento de DW a nível organizacional ou Data Marts com uma amplitude de funções elevada, o que aumenta a complexidade e ofusca a determinação correta do escopo.
- Usuários possuem expectativas irreais em relação a este tipo de tecnologia
- Ao passo que os usuários ganham melhor entendimento do DW/DM, seus "desejos" ou necessidades mudam
- Muitas vezes o DW/DM pode expor problema na qualidade dos dados o que leva a desconfiança por parte dos usuários finais.
- Coiller (2011) ainda ressalta que a taxa de projetos de DW/DM que falham gira em torno de 50%%, se caracterizarmos por falhos projetos que:
- Extrapolam custos e prazos
- Apresentam funcionalidades esperadas não foram implementadas
- Deixam usuários insatisfeitos
- Possuem performance, disponibilidade ou escalabilidade inaceitáveis
- Exibem dados sem valor(são pobres aos olhos do usuário)

Nota-se que a implementação do DM por si só, não garante que o projeto seja "bem sucedido", já que para qualificar um projeto desta maneira, existem diversos elementos de avaliação focados principalmente nos usuários, e que devem ser atendidos.

Para Coiller (2011), abordar um projeto deste tipo de maneira tradicional leva a um aumento no risco do projeto por diversas razões, identificadas a partir das características citadas acima:

- Instabilidade e mutabilidade nos requisitos
- Desconexão entre usuários e desenvolvedores
- Incapacidade dos processos de desenvolvimento de responder a mudanças
- Devido aos fatores acima o produto final apresenta pouco ou nenhum valor ao usuário

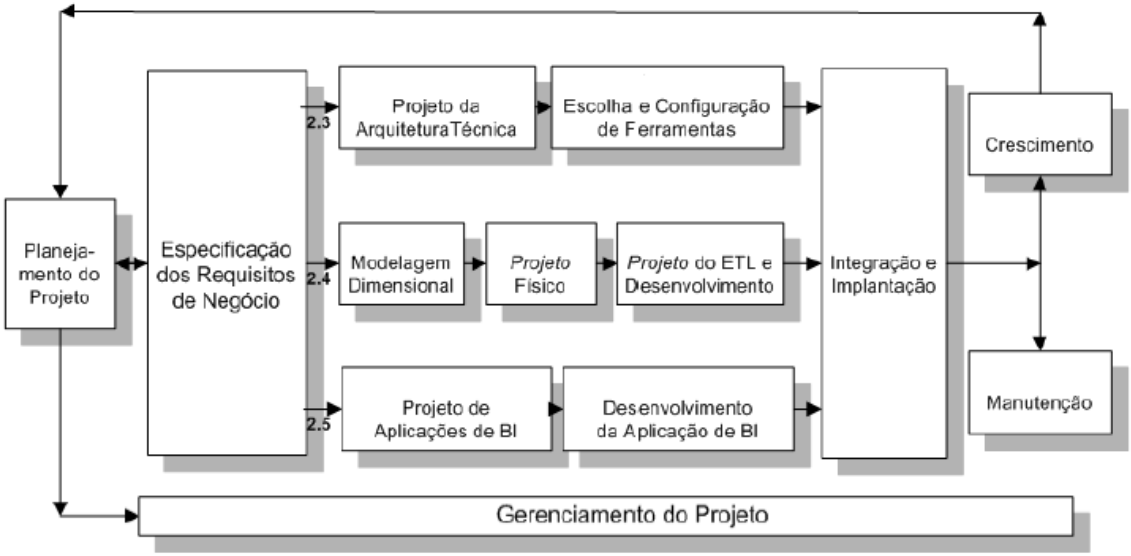
Para ele, um estilo ágil de desenvolvimento busca atuar em cima destes pontos críticos de modo que o objetivo de uma metodologia ágil dentro do âmbito de projetos de DW/BI se traduz em:

- Entregar continuamente software de qualidade que agregue valor ao cliente
- Reduzir o grau de risco em projetos deste tipo

Para realizar isso, o conjunto de práticas, valores e princípios ágeis utilizados na construção de software devem se adaptar ao ambiente de DM/BI norteando e permeando todo o ciclo de desenvolvimento.

Para Coiller (2011), a adoção de tais elementos caracterizados como ágeis, não muda radicalmente o modo fundamental de se construir sistemas de DM/BI ou invalida as melhores práticas que foram estabelecidas por muitos autores como Imon (1991) ou Kimball (2002), mas implica, na verdade, em um modo ou estilo alternativo de desenvolvimento que se propõe a suprir as diversas carências das abordagens tradicionais que foram apresentadas.

Kimball (2002) descreve um fluxo representativo das etapas de desenvolvimento de um DM:



Tendo como base essa seqüência de atividades, Collier (2011) propõe a adoção de uma série de praticas ágeis a fim de tornar o processo de desenvolvimento "ágil". As práticas se concentram em agir sobre as atividades principais no desenvolvimento de tais sistemas que vem a ser:

- Especificação dos requisitos do Negócio

- Criação dos modelos dimensionais
- Projeto de ETL
- Implementação física do modelo (carregar dados para as tabelas multidimensionais)
- Implantação

Para Collier (2011), essas atividades, dentro de um contexto ágil são repetidas a cada iteração, e quando aliadas as praticas ágeis, permitem que o desenvolvimento seja abordado de uma maneira evolutiva.

Diversos autores, dentre eles Highsmith (2009) e Collier (2011), afirmam que uma aplicação de DM/BI não deve ser desenvolvida em um "BIG BANG", termo usado por eles para se referir ao modo de desenvolvimento cascata ou outras metodologias pré-descritivas (como cunhado por Fowler (2005), onde, de uma especificação/determinação minuciosa de requisitos, se da origem logo no início do projeto, a construção de um design detalhado de todo o sistema a qual a implementação deverá ter conformidade.

Uma abordagem evolutiva objetiva construir o sistema em ciclos de iterações limitadas por um período de tempo (na maioria das vezes de 2 a 3 semanas), onde ao final de cada iteração são resultadas novas funcionalidades (features), que são implementadas tendo como base as "Histórias de usuários" e são do ponto de vista técnico e do usuário respectivamente testadas e aceitadas (testes de aceitação) estando elas prontas para integrar o resto do sistema.

Qualquer evolução requer adaptação, e esse elemento é alcançado através da contínua maturação de funcionalidades inicialmente simples que, de forma incremental e ao longo das iterações adquirem um nível de complexidade mais elevado.

É facilmente observado que cada etapa subseqüente no desenvolvimento evolutivo depende intrinsecamente das fases anteriores, ou seja, a implementação se baseia completamente no que já foi desenvolvido. Dai surge à necessidade de práticas que dêem suporte a cada fase do projeto para que o desenvolvimento evolutivo seja viável.

4.2 **Administração do "Débito técnico"**

O termo débito técnico é utilizado por Collier (2011) para descrever aspectos técnicos negativos quando examinamos o código de uma aplicação, que por existirem acabam por

dificultar qualquer tipo de alteração, ou adição de nova funcionalidade. Esses aspectos se referem a escolhas de design pobres feitas devido a falta de embasamento,erros de desenvolvedores e muitas vezes quando questões de performance são privilegiadas levando a criação de designs sub-otimizados,dentre outras razões.

Em sistemas de DW/BI a preocupação em administrar esse débito também é essencial em um desenvolvimento iterativo, já que à medida que o sistema cresce e aumenta sua complexidade através da adição de novas fontes de dados, maior volume de dados e mais capacidades, o custo de Mudanças se eleva.Highsmith (2009).

Quando uma equipe não controla seu débito técnico, o sistema eventualmente atinge um ponto de estagnação onde defeitos são prevalecentes e se torna muito custoso e arriscado adicionar uma funcionalidade nova. Times ágeis identificam,monitoram,priorizam e corrigem o grau de débito técnico ao longo do desenvolvimento,sendo que o tempo de implementação de novas funcionalidades deve ser balanceado com o tempo gasto em redução desse débito.

Quanto ao tempo gasto na redução, Collier (2011) aponta que times podem dedicar iterações somente a este propósito ou alocar uma porcentagem de tempo em cada iteração.

4.3 **Desenvolvimento evolutivo de banco de dados do DM**

Como já foi visto, métodos ágeis abordam o desenvolvimento de software de uma maneira diferente em diversos âmbitos. A maneira como o design do sistema em questão é desenvolvido em um contexto ágil, em que pontos isso influencia a construção e através de que práticas ocorre à implementação, constituem as características fundamentais desse processo de desenvolvimento de software.

As seções a seguir buscam dar ênfase em mostrar como alguns desses métodos de desenvolvimento de software podem ser aplicados a elaboração do design dos bancos de dados que irão compor o Data Mart. Essa busca em adequar os métodos ágeis a esse contexto é de grande importância dentro de um cenário de desenvolvimento de um Data Mart (ou mesmo de outro tipo de outra aplicação de BI), pois esse tipo específico de projeto possui atividades intensamente focadas em: manipulação de dados e especificação e implementação de design de bases de dados otimizadas para consulta.

Ao tratarmos, então, dessa adaptação de práticas ágeis ao contexto de bases de dados, nos deparamos com o conceito de desenvolvimento evolutivo de bases de dados, que pode ser aplicado através da adoção de diversas práticas, como refatoração e testes automatizados.

Para Martin Fowler (2003), a atividade de design não é tratada, dentro de um framework ágil, como sendo uma etapa do desenvolvimento anterior a construção. Ela é tida como sendo um processo contínuo, o qual é inter-relacionado a construção, testes e entrega do software.

A evolução de bancos de dados então, partindo da concepção de design de Martin Fowler, consiste no processo de construção do esquema de bases de dados que ocorre ao longo das iterações de um projeto ágil, onde mudanças no esquema de BD são feitas gradativamente, de uma maneira controlada, através da utilização de uma série de técnicas críticas que dão suporte a este modo de desenvolvimento:

- Uso de histórias de usuários na especificação de requisitos
- Modelagem de dados evolutiva
- Configuração de ambientes de desenvolvimento (Sandboxes)
- Administração dos artefatos de Banco de Dados
- Refatoração de banco de dados
- Testes

4.3.1 Especificação de requisitos

Collier (2011) mostra que diversos projetos de BI norteiam o desenvolvimento focando nos dados que a aplicação irá apresentar e como uma abordagem desse tipo pode ser nociva ao projeto, se forem avaliadas as variáveis citadas anteriormente que indicam se o produto agregou ou não valor ao negócio do usuário.

Ele defende que a atenção no desenvolvimento deve se concentrar nos requisitos do negócio, ou seja, no problema inerente ao negócio sobre qual a solução de BI irá atuar e auxiliar no processo de tomada de decisão.

Os dados irão apenas dar suporte ao funcionamento do sistema, ou seja, o desenvolvimento de um sistema de BI não deve ser dirigido para alcançar a representação desses dados por si só, mas sim, utilizá-los para atender as necessidades do negocio, ou seja, satisfazer os requisitos através de funcionalidades que agreguem valor de forma contínua.

Ao se tratar de funcionalidades, métodos ágeis de se desenvolver software têm a característica de entregar funcionalidades a cada iteração. Para atingir tal objetivo, essas metodologias buscam se apoiar em práticas de especificações de requisitos que dêem suporte a um desenvolvimento evolutivo (iterativo e incremental), de modo que um projeto

considerado ágil não possui uma longa fase de detalhamento de requisitos inicial, já que estes evoluem e se modificam ao longo do projeto.

Tendo isso em mente, diversas metodologias ágeis (XP, SCRUM) adotam o uso de histórias de usuários, que constitui uma técnica que permite que as funcionalidades que os usuários necessitam no sistema sejam capturadas e organizadas em sua essência sem que haja a necessidade de se conduzir um processo de levantamento de requisitos intenso. Ao utilizá-las, os detalhes da implementação surgem gradativamente no ciclo de vida, através da interação entre desenvolvedores e usuários, e não são determinados todos a priori.

Para Leffingwell (2011), histórias de usuários são ferramentas que tem o propósito de definir o comportamento do sistema de uma maneira entendível simultaneamente pelos desenvolvedores e usuários. (Nota-se aqui, como práticas ágeis buscam solucionar o problema da desconexão entre o time desenvolvimento e os clientes já citado). Ao se examinar o âmbito de análise do sistema em questão, o uso de histórias de usuários foca em determinar o que agrega valor para o usuário ao invés de realizar uma decomposição funcional como é feito em processos tradicionais.

Highsmith (2009,p.67) define uma história de usuário como sendo:

"*Uma sentença expressa pelo usuário que pode ser relacionada a uma necessidade ou objetivo do negócio"*

Uma história de usuário pode ser expressa pelo seguinte formato:

Como **<Papel Usuário>** eu gostaria de poder **<Realizar determinada ação>** para que eu possa**<sentença que expresse o objetivo>**

Um exemplo seria:

"Como analista financeiro eu gostaria de poder ver o lucro por cliente e por transação ao longo do tempo para que eu pudesse identificar tendências positivas ou negativas."

Uma história de usuário representa uma unidade funcional, sendo que, o progresso do desenvolvimento é demonstrado ao usuário através da entrega de código testado e integrado que programa a história. Uma história deve ter as seguintes características:

- Representar valor de negócio para os clientes
- Quando implementada pode ser demonstrada para os clientes através de uma funcionalidade, a fim de se obter feedback

- Pode ser implementada em apenas uma iteração, de modo que ela seja completa em sua arquitetura e com qualidade de em nível de produção.

Há uma distinção entre histórias de usuários e os requisitos em si citada tanto Leffingwell (2011) como por Coiller (2011). Eles afirmam que historias de usuários não são os requisitos propriamente ditos (em outras palavras, o que o sistema fará), mas sim elas são uma representação (esboço) destes requisitos. Elas são negociáveis e indicam para onde o desenvolvimento deve se nortear.

Por serem negociáveis e entendíveis pelos usuários e desenvolvedores, as histórias de usuários permitem que haja uma colaboração efetiva entre ambas as partes a fim de se compreender melhor o que realmente precisa ser feito, ou seja, os requisitos.

Após se criarem as histórias de usuários, estas são agrupadas e priorizadas, e dentro da iteração elas serão programas pelos desenvolvedores.

Anterior a fase de implementação, uma fase de modelagem de dados irá ser realizada, porem de maneira diferente das abordagens tradicionais. Essa modelagem será um artefato produzido a partir da interação entre o time e os clientes no processo de criação das histórias, e assim como a especificação de requisitos, não visa ser uma etapa intensa e longa que atrase o desenvolvimento, pois o foco não é modelar o sistema de uma só vez no chamado BDUF (Big Design Up Front-grande design inicial). A utilização da modelagem ágil,citada por Ambler(2002) se encaixa nesse processo, na criação dos modelos dimensionais e é descrita a seguir.

4.3.2 Modelagem Ágil e a criação dos modelos dimensionais

Para Ambler (2002), modelagem ágil é uma metodologia baseada em práticas que focam na produção eficiente de modelos e documentação de sistemas de software. Essa metodologia é guiada por valores e princípios ágeis,logo não se caracteriza por ser um processo pré-descritivo.Ambler(2002) destaca que AM(Agile Modeling ou modelagem ágil)

não definem procedimentos detalhados para criação de modelos,mas sim,fornece meios para se modelar de forma eficiente.

A modelagem ágil tem como principais objetivos:

- Definir um meio de colocar em prática valores, princípios e técnicas ágeis a fim de se obter uma modelagem enxuta (sem trabalho desnecessário) e eficiente
- Adequar à construção de modelos em metodologias ágeis que se baseiam em iterações

AM se baseia nos seguintes princípios:

Entrega de software funcionando é o principal objetivo: Para Ambler (2002), modelos não são produtos a serem entregues, são artefatos que suportam o desenvolvimento, o que pode ser estendido a qualquer documentação, logo, não servem com indicadores de progresso em um projeto. Entrega de Software funcionado e que possui valor para o cliente é o objetivo,e qualquer atividade que não contribua diretamente com este objetivo deve ser questionada e evitada,caso não possa ser justificada de maneira adequada.

Leveza no desenvolvimento: Esse princípio se traduz em criar apenas modelos e documentação suficientes para que o desenvolvimento possa ocorrer. Qualquer documentação que é mantida dentro do projeto deve sofre manutenção com o tempo, pois ocorrem mudanças no ambiente, o qual estes artefatos representam. Quanto maior for o número de modelos, ou mais complexo for seu nível de detalhamento, o tempo gasto para atualizá-los e mantê-los corretos e precisos aumenta proporcionalmente.

Aqui, Ambler (2002) ressalta um equilíbrio que deve ser mantido ao se balancear: a perda de agilidade ao manter um modelo com a vantagem de dispor da informação representada por ele em um maneira abstrata para todo o time.

Modelagem Incremental: Para que o processo de desenvolvimento seja adaptável a mudanças, é necessária uma abordagem incremental. Sendo assim, grandes mudanças podem ser traduzidas em uma série de pequenas modificações em um sistema. Ao relacionarmos esses fatos com modelagem, Ambler (2002) ressalta que é primordial que ao modelar, o desenvolvedor não tente representar toda o problema de uma só vez, ou se ater a todos os detalhes inicialmente, criando um modelo que englobe todos esses aspectos. Ao invés disso,ele sugere que o desenvolvimento de um modelo pequeno e detalhado que possam evoluir com o tempo.

Simplicidade: Esse princípio empresta a idéia de simplicidade exibida por Beck (1999) onde ele afirma que, na maioria das vezes, a solução mais simples funciona da melhor maneira, e por ela ser simples, é fácil de ser implementada. A implicação dessa afirmação ao se tratar de modelagem diz respeito à preocupação que deve existir em não sobrecarregar o sistema com modelos que possuem sua complexidade elevada em determinado momento, pois buscam representar funcionalidades não necessárias naquela dada etapa do projeto.

Esse princípio se traduz na pratica de modelar da maneira mais simples os requisitos existentes e refatorar o sistema no futuro para acomodar os requisitos que evoluíram.

Modelar com propósito: Esse ponto se refere ao porquê de se criar uma documentação. Para Ambler (2002) essa motivação pode ser advinda de fatores como: modelar para entender ou modelar para comunicar algo.

Qualquer outra razão para se criar modelos como: solicitação não justificável de terceiros, por se caracterizar uma atividade obrigatória a ser cumprida em processo pré-descritivo ou modelar ao invés usar comunicação direta,quando se tem a opção, são inválidas como justificação para construção do artefato.

Para Ambler (2002), ao criarmos um modelo, inicialmente devem ser identificados: o propósito e a audiência para qual ele será exposto. Apos isso, é construído um modelo que seja suficientemente preciso e detalhado. Assim que ele foi criado o trabalho de modelagem termina.

Isso implica em não adicionar detalhes não necessários e delimitar um tempo adequado a fase de modelagem, ou seja, após cumprir seu propósito a atividade de modelagem não continua indefinidamente, pois a construção de software é a prioridade.

Uma aspecto fundamental das metodologias ágeis é a utilização de desenvolvimento iterativo, a fim de se entregar software rapidamente. Da mesma forma que se utiliza uma abordagem iterativa na construção do software, é utilizada em AM um abordagem incremental as fases de modelagem. AM foca em reduzir os tempo de sessões de modelagem,já que o escopo do modelo é sempre reduzido,afim de servir como base para se desenvolver uma pequena porção do código apenas.

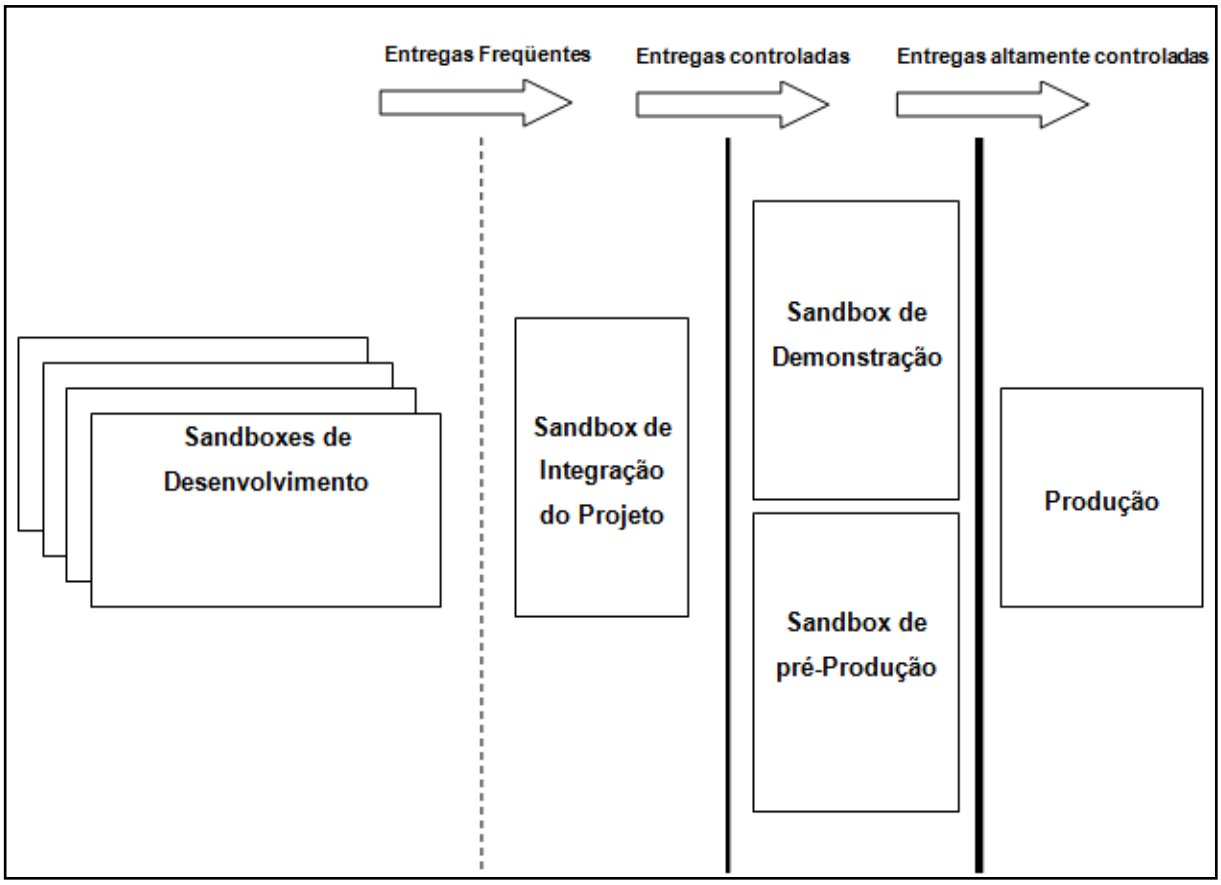
Isso se justifica segundo Ambler (2002), pelo fato de que, ao passo em que se avança no projeto sem o feedback concreto que a entrega de software proporciona, a chance de se construir um modelo que representa algo que não é necessário aumenta, o que se traduz em

desperdícios de esforços. Em ambientes de BI,isso se identifica claramente,pois,como ressaltado por Imon(1991),os requisitos podem se modificar ao passo em que os usuários ganham melhor entendimento do sistema e interagem com ele.

4.3.3 Configuração do ambiente de desenvolvimento

Sistemas de BI são tradicionalmente desenvolvidos, utilizando um ambiente de desenvolvimento compartilhado em ambientes de: desenvolvimento, pré-produção (testes) e produção. Para Collier (2011) e Fowler (2006), em muitos projetos isso impede que desenvolvedores experimentem com novas idéias, e até mesmo ocasiona conflitos entre modificações no código de dois ou mais desenvolvedores. Eles ainda ressaltam que a transição do sistema entre as fases (ex: de teste para produção) é um processo complexo e que acaba por envolver configurações manuais, ajustes etc. o que ocasiona em um grande dispêndio de tempo.

A fim de dar suporte a entregas freqüentes de software e ao modelo evolutivo de desenvolvimento, uma infra-estrutura é sugerida por Ambler (2006) onde são utilizadas Sandboxes de desenvolvimento. O esquema é descrito na figura a seguir:



Para Martin Fowler (2006) essa infra-estrutura permite:

- A obtenção de um ambiente de desenvolvimento separado para cada desenvolvedor
- Dar suporte a execução de toda a suíte (coleção) de testes
- Entrega rápida de funcionalidades nos ambientes de produção
- Possibilidade de se retornar a uma versão estável caso algum problema grave ocorra

Collier (2011) define uma Sandbox como sendo uma réplica do ambiente de produção onde se espera que o sistema seja implantado.

Uma Sandbox pode ser constituída por um Server dedicado, uma partição ou mesmo um simples diretório dedicado. O ambiente completo inclui diversas Sandboxes, onde cada desenvolvedor possui uma Sandbox individual, existindo uma Sandbox de integração do trabalho de todos, uma para demonstração para usuários e outra para pré-produção.

Essa estrutura é utilizada para possibilitar a implementação das refatorações de bancos de dados, que serão responsáveis por todas as modificações feitas no esquema e nos dados dos mesmos, bem como, fornecer um ambiente seguro para a execução de testes dessas bases de dados. Para que isso aconteça cada Sandbox irá conter uma cópia do BD contendo dados de

teste, sendo que estes dados podem ser amostras de dados de produção caso seja possível extraí-los. Mudanças no esquema das Bases de dados devem ser capazes de migrar de maneira correta os dados para o esquema mais novo.

4.3.3.1 Administração dos artefatos de Banco de Dados

Em projetos de software, uma gama de arquivos são utilizados, logo, administrar todos eles é uma tarefa trabalhosa. Diversas ferramentas foram desenvolvidas com o propósito de administrar os componentes que envolvem a construção do software, elas são denominadas de Ferramentas de Administração de Código Fonte (Source Code Management tools), embora sejam referenciadas usualmente como sistemas de controle de versão ou repositórios, sendo exemplos dessas ferramentas o Subversion e o CVS.

Um dos principais objetivos do repositório para Ken Coiller é permitir que a versão corrente do sistema de BI seja colocada ou retirada de qualquer uma das Sandboxes de uma maneira fácil e rápida.

O repositório deve conter todos os scripts ETL, Store Procedures, Scripts DDL, definições de cubos OLAP, ou seja, basicamente todos os elementos necessários para se construir o sistema.

Nessa infra-estrutura contendo diversas Sandboxes e um sistema de controle de versão, os desenvolvedores podem copiar a versão mais recente do BD compartilhado para sua Sandbox, realizar quaisquer alterações sem que elas causem impacto direto no ambiente compartilhado e então tentar integrar as mudanças ao repositório. Fowler (2003) ressalta que essa prática correspondem à integração contínua de software encontrada em projetos XP,e que bancos de dados dentro deste processo são administrados de maneira similar ao código fonte.

4.3.4 Refatoração de Bancos de dados

Fowler (1999,p.85) descreve a técnica de refatoração como sendo:

"*Uma maneira disciplinada de reestruturar o código em pequenos passos de cada vez"*

A prática de refatoração suporta a abordagem evolutiva de desenvolvimento, evoluindo o código fonte de uma aplicação através da implementação de pequenas mudanças nele, feitas no decorrer das iterações do projeto.

Para Collier (2011), refatorações servem a dois propósitos básicos: Evoluir de modo seguro, design e modelos e eliminar o "Débito Técnico", sem danificar features (funcionalidades) e componentes que estavam funcionando previamente. Isso demonstra para ele uma preocupação predominante no desenvolvimento iterativo,de que o trabalho feito na iteração atual não tenha efeitos adversos no que já foi construído.

Uma aspecto que se destaca na refatoração para Ambler (2006) diz respeito ao fato do código manter o comportamento semântico após as modificações, de modo que, não são adicionadas ou removidas funcionalidades, objetivando apenas a melhoria do design do código já existente.

Tendo em vista uma aplicação de DM/BI, a prática de refatoração é utilizada também em outro contexto, já que na implementação dessas soluções é despendida uma grande parcela de tempo na estruturação das bases de dados que dão suporte aos aplicativos de BI.

Collier (2011) ressalta que as práticas de refatoração de banco de dados estabelecidas por Ambler (2003) são aplicadas diretamente ao conceito de Data Mart evolutivo. Visto isso,a prática de refatoração é aplicada as bases de dados a serem estruturadas,para permitir o desenvolvimento adaptativo e incremental dessas soluções. De forma similar, a definição de refatoração de código descrita por Fowler (2003), Ambler (2006,p130) define uma refatoração de banco de dados como sendo:

"*Uma simples mudança no esquema de banco de dados que ao melhorar o design existente, deixa inalterados as semânticas comportamentais e informacionais do mesmo."*

Tendo em vista essa definição fica evidente que a refatoração de banco de dados é um processo mais complexo se comparado a uma refatoração de código-fonte, já que a semântica dos dados deve-se manter inalterada. Isso se torna um fator crítico quando somada a possibilidade de uma base de dados poder possuir um elevado grau de acoplamento com outras aplicações que a utilizam, sendo que qualquer mudança nessa base refletiria no funcionamento das aplicações. Para Ambler (2006), uma refatoração de banco de dados deve preservar o funcionamento de qualquer aplicação que utilizem a base, sendo muitas vezes

necessário um período de transição onde os esquemas atualizados e antigos de banco de dados funcionem em paralelo, onde a mudança de acesso é feita gradativamente.

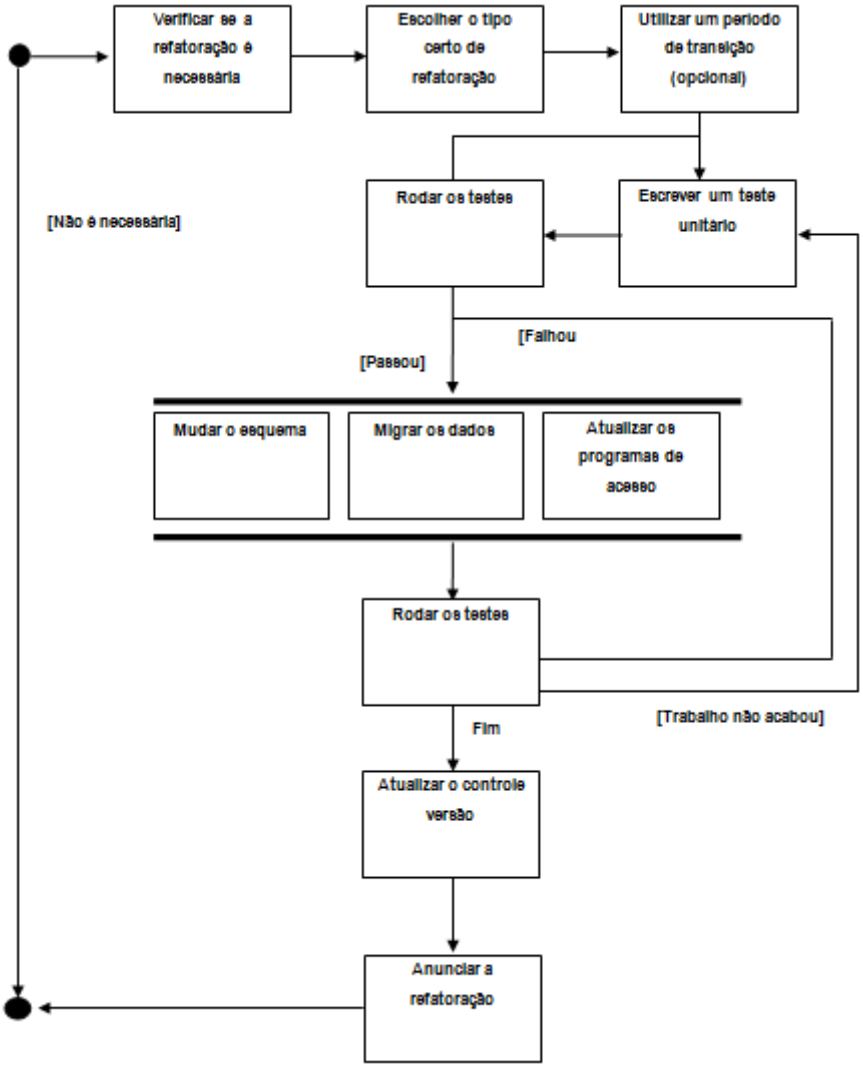
4.3.4.1 Outros motivos que levam a refatoração

Collier (2011) mostra que refatorações não são feitas somente em DMs em desenvolvimento, mas também em DMs em produção. Ambler(2006) elícita algumas característica no ambiente de banco de dados que indicam a possível necessidade de refatorações:

- Resistência a mudanças nas estruturas de BD por parte da equipe: Pode significar um nível de Débito Técnico não desejado e que deve ser eliminado.
- Colunas com vários propósitos diferentes: Colunas que apresentam semânticas diferentes dependendo do contexto de utilização
- Tabelas com vários propósitos: Tabelas utilizadas para armazenar tipos diferentes de entidades podem significar falhas no modelo de dados (clientes e fornecedores na mesma tabela, por exemplo)
- Data redundante: Duplicidade de dados gera possíveis inconsistências
- Tabelas com número excessivo de linhas: Acarretam problemas de desempenho
- Tabelas com número excessivo de colunas: Falta de coesão, a tabela não tem um propósito definido e armazena dados de diversas entidades.

4.3.4.2 Principais etapas da refatoração

Ambler (2006) sugere uma série de etapas que compõe o processo de refatoração, essas etapas são descritas na figura a seguir:



A Sandbox do desenvolvedor é o ambiente inicial onde são desenvolvidos e testados o código e as mudanças feitas ao esquema do BD. Para Ambler(2003) a necessidade de uma refatoração é tipicamente identificada por um desenvolvedor que está implementando uma funcionalidade ou consertando um defeito.

1. Inicialmente é preciso verificar se a refatoração é apropriada e quais as justificativas para fazê-la. Ambler (2006) exemplifica que, muitas vezes, podem existem solicitações da criação de uma estrutura no banco de dados por parte dos desenvolvedores, sendo que tal estrutura já existe, porém o desenvolvedor pode não está a par disso por não ter uma familiaridade com o esquema do BD assim como um DBA possuiria.

Esse exemplo justifica a intensa colaboração que deve haver entre DBAs e desenvolvedores no projeto ágil, onde uma comunicação entre estes membros da equipe deve existir a fim de se determinar:

- A motivação para realizar a mudança (por exemplo: são mudanças necessárias para implementar uma funcionalidade? Se afirmativo, elas refletem quais requisitos do negócio?)
- Se o tipo de refatoração é o mais adequado
- Quais impactos serão causados (Por exemplo: Determinado código de acesso ao BD deverá ser modificado também)
- Se os impactos que ela irá causar compensam os esforços despendidos nela

2. Determinada a necessidade, deve ser definido o tipo de refatoração apropriado. Cabe aos DBAs e desenvolvedores determinarem qual a melhor maneira de se implementar uma estrutura ou lógica dentro do BD. Ambler (2006) lista as possíveis refatorações de BD em diversas categorias:

Categoria	Descrição	Exemplo
Estrutural	Mudança na definição de uma tabela ou de uma View	Mover uma coluna de uma tabela para outra, separar dados de uma coluna em várias outras colunas
Qualidade dos Dados	Mudança que aumenta o nível de qualidade das informações na base de dados	Modificar uma coluna para não permitir a entrada de valores NULL
Integridade Referencial	Mudança para garantir que integridade referencial exista ou garantir que a exclusão de linhas não necessárias seja feita corretamente	Adicionar um Trigger para acionar deleção em cascata, código que era implementado antes fora da base de dados
Arquitetural	Mudança que melhora a maneira de como aplicações externas a base de dados interagem com a mesma	Tornar uma operação realizada por uma aplicação disponível para várias linguagens acessarem através da criação de uma Store procedure
Método	A mudança em um método (Store procedures/ function, Triggers) a fim de melhorar a qualidade dele	Reno mear uma Store Procedure para torná-la mais fácil de identificar e entender
Transformação não considerada como refatoração	Uma mudança que altera a semântica no esquema da base de dados	Adicionar uma coluna nova a uma tabela já existente

Ambler (2006) ressalta que, quando se está implementando uma refatoração estrutural de BD,ou uma destas subcategorias, há necessidade de se determinar se os dados na base estão “limpos”, ou seja, com qualidade suficiente para sofrerem este processo. Dependendo dessa qualidade pode haver a necessidade de se realizar uma refatoração para aumentar a qualidade destes dados,antes de proceder para uma refatoração estrutural.

3. Se a refatoração for feita em um DM em produção, um período de transição deve ser estabelecido. Durante esse período o esquema antigo e novo funcionam em paralelo,sendo que uma aplicação de BI utiliza apenas um dos esquemas de BD,porem nunca os dois.Os dados em ambos os esquemas devem ser sincronizados para garantir que as aplicações continuem funcionando,independente do esquema.Ao mesmo passo as aplicações de BI são modificadas para acessarem o novo esquema,culminando com a desativação do esquema antigo e sendo iniciada uma fase de testes.
4. Para Collier (2011) o único modo de se mudar um modelo de dados é através da execução rigorosa de testes automatizados durante todo o processo. Todos as formas possíveis de acesso que as aplicações de BI fazem na base de dados devem ser testados e a suíte de testes deve ser mantida atualizada constantemente.
5. Após determinar o tipo de refatoração e construir os testes necessários, a mudança é implementada através de scripts e testes de regressão são construídos. Ao mesmo passo é necessário um acompanhamento das mudanças no sistemas de controle de versão.
6. As aplicações de BI que acessam o esquema devem então serem modificadas sendo que Ambler (2006) sugere que sejam seguidas as práticas de refatoração de código descritas por Fowler (1999).
7. Testes de regressão devem ser executados

Por fim, a refatoração deve ser submetida ao controle de versão e anunciada a equipe.

4.3.5 Integração de testes ao desenvolvimento

Diferente do processo de integração das modificações de código fonte ao repositório, as mudanças em bancos de dados (refatorações) requerem alguns cuidados especiais, e devem, para Collier (2011) serem todas automatizadas.

Para que a evolução dos esquemas de bancos de dados utilizados pelo Data Mart ocorra através das refatorações, é necessária para Ambler (2003) a existência de uma suíte de testes, que atuem tanto no nível de aplicação, como de bases de dados para garantir que estas mudanças, que serão feitas freqüentemente a cada iteração, não entrem em conflito uma com a outra, ou causem efeitos inesperados em funcionalidades já existentes no sistema.

Collier (2011) mostra que projetos de DM/DW que seguem abordagens tradicionais postergam os testes para o fim do ciclo de desenvolvimento. Para adotar ágil nesses projetos, ele advoga que os testes devem ser integrados aos processos de desenvolvimento.

A integração dos testes permite que defeitos não se acumulem e reduz o nível de débito técnico tornando o Feedback referente à qualidade do produto freqüente e imediato.

Unir os testes ao processo de desenvolvimento torna possível a abordagem evolutiva (incremental e iterativa) de um projeto, fornecendo meios confiáveis e eficientes para suportá-la.

O projeto de um Data Mart, por ser uma aplicação de BI, se caracteriza por possuir atividades intensivamente relacionadas a tratamentos de dados, num grau muito maior do que aplicações de software convencionais, logo, as principais praticas ágeis que se referem a testes (como a metodologia TDD) que foram criadas inicialmente para serem utilizadas em aplicações OO, devem ser submetidas a uma mudança para este contexto.

As seções a seguir explanam estas práticas de testes e como elas se encaixam com o processo de refatoração de BD, para que o Data Mart seja desenvolvido de uma maneira evolutiva.

4.3.5.1 Tipos de testes utilizados

Para Marick (1994), existem quarto dimensões ou perspectivas ao tratar testes de software:

- **Aceitação do negócio:** Dimensão focada no usuário final e que tem como objetivo avaliar a capacidade do sistema de entregar as funcionalidades de valor que o usuário esperava, ou seja, se o sistema faz o que o usuário esperava e precisava;
- **Validação do Produto:** Validação que diz se o sistema desempenha sua funções corretamente, constituindo uma analise do produto entregue
- **Aceitação técnica:** Avalia se o produto atende requisitos a nível técnico determinados pelos desenvolvedores
- **Validação do sistema:** Serve como meio de avaliar o sistema em nível de código e assegurar aos desenvolvedores que o software se comporte da maneira desejada

Ao analisarmos metodologias ágeis de desenvolvimento, existe uma prática, criada por Kent Beck e explanada em Beck (1999), que aborda os aspectos de aceitação técnica e validação do sistema, denominada Testes unitário. Estes tipos de testes tem como foco testar a menor unidade do código possível dentro do código, no caso de programação estruturada funções ou procedimentos ou no caso de linguagens orientadas a objeto interfaces ou classes.

Para Collier (2011), dentro de um ambiente de DW/DM, os testes unitários são integrados ao desenvolvimento a fim de se permitir uma abordagem evolutiva de desenvolvimento, sendo que, a ênfase do processo se dá no teste de estruturas de bases de dados, scripts SQL, Store Procedures, PL/SQL e componentes de ETL.

Outros tipos de testes da aplicação de BI citados por Collier (2011) se caracterizam por avaliar como o sistema se comporta em situações de stress, seja quando é dada uma sobrecarga nos recursos do sistemas devido a intenso uso a fim de se determinar o modo como ele aborta a execução, ou mesmo testes de carga de alto volumes de dados a fim de se avaliar a desempenho da aplicação (este aspecto é crucial em aplicações de BI de modo que as atualizações são todas feitas baseando-se em carga de dados, logo, testar esse processo pode indicar um gargalo que prejudique todo o funcionamento do sistema).

4.3.5.2 Automatização de testes

Segundo Collier (2011) dentro de um ambiente iterativo e adaptativo, não é prática a utilização de testes manuais, de modo que, para que haja a integração dos testes ao desenvolvimento é necessário que os procedimentos de testes sejam automatizados. Essa necessidade se justifica pelo fato da execução de testes manuais demorarem períodos demasiadamente longos para serem realizados e por não caracterizarem uma prática que possa ser executada de forma suficientemente repetível e confiável aponto de permitir a execução de testes de maneira eficiente e sustentável ao longo das iterações.

4.3.5.3 TDD(Test Driven Development)

Beck (1999,p.56) esboça sucintamente a estratégia de testes usada na metodologia ágil conhecida como XP com a seguinte frase:

"*Nós escrevemos testes antes de codificarmos o programa, a cada momento. Nós também devemos preservar estes testes, a fim de executá-los em conjunto freqüentemente.*"

Esse trecho reflete características fundamentais de um método de se desenvolver software, denominado TDD (Teste Driven Development) criado por Beck(1999) que enfatiza a integração de todo o processo de testes ao desenvolvimento, tornando estas atividades intrinsecamente dependentes, de modo que, em um cenário onde é adotada essa forma de

desenvolvimento não é escrito código de implementação da aplicação, sem que antes haja um teste escrito para avaliar a funcionalidade a ser desenvolvida.

A prática de TDD divide a construção de código nas seguintes etapas:

1 **Construção de um teste:** Nessa etapa é codificado um teste, com um escopo reduzido, como Beck(1999) afirma: ”Apenas o suficiente para o código fonte a ser testado falhar”, nota que nenhum código referente à funcionalidade deve ser escrito antes desta etapa.
2 **Rodar o teste:** Nessa parte, são executados os testes, a fim de presenciar o código fonte falhar.
3 **Adicionar uma pequena modificação ao código:** Após a falha do código, é realizada uma pequena modificação nele, apenas suficiente para permitir que o código passe no teste que falhou.
4 **Verificar que o código passou o teste:** O código satisfez as condições dos testes e foi executado com sucesso.
5 **Refatorar:** Nessa etapa, aparentemente opcional, se encontra a chave para o desenvolvimento evolutivo. É na refatoração que o programador busca fazer com que o programa passe no teste especificado, da melhor maneira possível. Buscar isso se traduz em satisfazer requisitos técnicos como eficiência,clareza,baixo acoplamento do código fonte,dentre outros.
6 **Voltar a realizar o primeiro passo:** O ciclo se repete, assim que é feita outra modificação no código.

Beck (1999) ainda coloca duas regras fundamentais ao se adotar TDD:

- Um código referente a uma nova funcionalidade só é criado quando um teste falhou
- É mandatório eliminar qualquer duplicidade encontrada

Na visão de Beck (1999), isso gera diversas mudanças de comportamento no ambiente:

- O código é desenvolvido de uma maneira orgânica e prove Feedback para tomada de decisões.
- Os programadores escrevem seus próprios testes
- O ambiente deve prover resposta rápida a pequenas mudanças

- O design deve ser composto por componentes com baixo acoplamento e coesos (caracterizando uma estrutura normalizada) a fim de facilitar os testes.

O uso de TDD, para Ambler (2003) fornece diversas vantagens, pois eleva o nível de construção de testes unitários realizados ao longo do projeto. Na visão de Ambler (2003), essa prática permite que o software seja desenvolvido de uma maneira muito mais produtiva.

A diferença fundamental de um modo tradicional de se codificar uma aplicação e TDD é a maneira com que o código é desenvolvido. Funcionalidades são implementadas aos poucos, ou segundo Beck (1999), em pequenos passos, de modo que, sempre que uma funcionalidade for adicionada, a probabilidade de se identificar e corrigir um erro na mesma é muito maior se o código pelo qual ela é composta for reduzido.

Para Ambler (2006) a prática de testar se caracteriza mais como sendo um ato de design do que de verificação, o que acaba por tornar o nível de design da aplicação detalhado. Essa faceta do TDD é explicada por Beck (1999), ele demonstra que ao se construir um teste, o programador é forçado (ou estimulado), a analisar diversos requisitos, tanto técnicos (no caso de um teste unitário) como referentes a requisitos dos usuários (em testes de funcionalidades) a fim de elaborar um teste que confronte o programa contra estes requisitos.

Ao verificar que o programa não passou em um teste, são necessários ajustes no código, feitos através das refatorações, que constituem o processo de design em si, havendo uma evolução da estrutura (em conseqüência, do design) conforme novos testes são adicionados. Nota-se aqui, a influência das práticas de desenvolvimento incremental e evolutivo.

4.3.5.4 Adoção de TDD ao desenvolvimento ágil de Data Mart

O TDD é uma prática de testes/desenvolvimento que foi inicialmente criada para ser utilizada na criação de aplicações usando orientação a objetos, porém, Collier (2011), mostra que dentro de um contexto de desenvolvimento de aplicações voltadas para BI é possível, também, aplicar essa metodologia ao desenvolvimento e explorar os benefícios que a automatização de testes fornece de uma forma sistemática e confiável, levando-se em consideração algumas peculiaridades deste tipo de projeto, que não são presentes na construção tradicional de software:

Volumes de dados: Sistemas de BI lidam com um volume muito maior de dados do que sistemas transacionais, logo, por mais simples que pareça de se criar um teste, ao se

analisar um ambiente onde irão ocorrer manipulações de dados, possivelmente de diversas fontes, a complexidade desse mesmo teste pode crescer.

Testar código orientado a objeto: Em sistemas de BI, não é comum se utilizar código orientado a objeto. No lugar, se encontram scripts ETL, Store Procedures e outra infinidade de códigos procedurais. Esse fato se torna um problema quando consideramos que ferramentas de teste de software são otimizadas para testar código orientado a objeto onde há uma definição clara das unidades do programa (Units),sendo elas coesas e com baixo acoplamento,o que contrasta com o ambiente de BI onde é difícil se definir estritamente uma unidade de teste.

Natureza diversificada do código: O código utilizado em sistemas de BI inclui uma variedade de linguagens como PL/SQLT-SQL/MDX, XMLA dentre outros, o que torna difícil escolher ferramentas de testes adequadas que englobe toda essa gama de linguagens.

Apesar de existirem estes obstáculos, Collier (2011) afirma que as práticas de testes criadas por Beck (1999), dentre outros autores que abordam metodologias ágeis, são perfeitamente aplicáveis ao contexto de Bi, desde que os testes sejam adaptados para atuarem sobre as bases de dados de forma automatizada.

Ambler (2003), afirma que a prática do TDD é utilizável no contexto de bases de dados permitindo a obtenção dos benefícios oferecidos por esse tipo de abordagem de desenvolvimento, sendo somada a eles, a garantia da qualidade dos dados que é um requisito fundamental na construção de uma aplicação de Bi para Collier (2011), o que por si só, constitui uma forte justificativa para se utilizar esta prática em um projeto de BI ágil.

4.3.5.5 Teste de bases de dados

Ao se testar bases de dados existem variáveis que devem ser levadas considerações adicionais ao processo de testes, que podem tornar o processo de testes complexo:

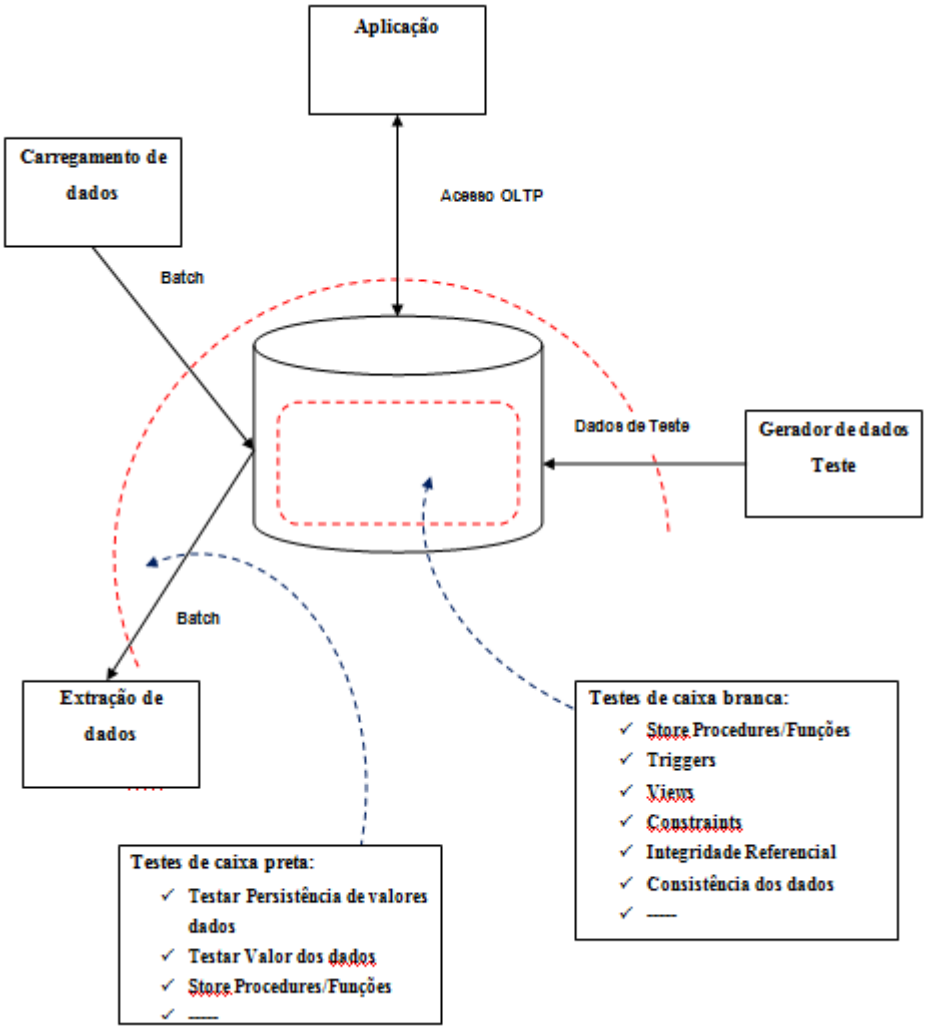
- Os esquemas de banco de dados e tabelas
- A necessidade de se inserir as linhas necessárias nas tabelas para executar um teste (amostra de dados de teste)
- É mandatório verificar o estado do banco de dados após a execução do teste
- Na etapa final deve haver uma limpeza do banco de dados. Isso é necessário pois a execução seqüencial de testes pode fazer com que, a um teste falhar, esse influencie o

funcionamento do teste seguinte, devido um estado inconsistente dos dados que ele produziu(Ambler(2003) caracteriza isso como acoplamento entre testes)

Existem diversas categorias de testes de bases de dados segundo Collier (2011):

- **Testes de funcionalidades do BD:** Focam em testar o comportamento do BD no lado cliente que acessa os dados. Testes de Store Procedures, Triggers e outros objetos programáveis do BD se enquadram nessa categoria. Nessa categoria de testes, há também uma fase de inspeção de dados a fim de confrontar o resultado que determinada operação (ex: Store Procedures) com os resultados esperados (ex: verificar a exatidão de um cálculo).
- **Teste de Esquemas:** Consiste em testar o esquema de BD e garantir que ele corresponda ao esperado. Um exemplo seria testar uma View para verificar se ela retorna determinado número de colunas com os tipos de dados apropriados
- **Segurança:** Dentro do âmbito de segurança, testes devem garantir que as permissões e restrições de usuários sejam respeitadas.

A figura evidencia a distinção que ocorre ao nos referirmos ao escopo do teste, ou seja, o que este teste irá validar dentro do ambiente. Essa distinção tem como objetivo determinar qual será a natureza dos resultados obtidos ao se executar determinado teste.



Devido a essa variedade de elementos que influem no processo de testes de BDs, ao planejar a execução de testes de base de dados, e determinar qual o escopo do teste, devem-se levar em consideração duas principais categorias de testes:

- Teste de interface
- Testes estruturais (teste de esquemas do BD)

A primeira categoria se refere a testar os principais pontos de acesso ao BD, focando na verificação de valores retornados/transmitidos entre elementos externos e o BD, ou seja, o aspecto funcional do BD. Para Factor (2011) esse teste é caracterizado como um teste de caixa preta, ou seja, o foco do teste é verificar o comportamento de I/O de um componente, sem levar em consideração qualquer estrutura interna do mesmo.

Em relação à segunda categoria, Factor (2011) os caracteriza como testes de caixa branca. Nesse tipo de testes, a estrutura interna do BD é levada em consideração ao se executar o teste, como integridade referencial. Após se executar uma refatoração, Factor(2011) ressalta que há grande chance de testes de caixa branca falharem,e esses testes devem ser atualizados para refletir a refatoração feita.

4.3.5.6 Etapas de teste de base de dados

Para Ambler (2003) existem três etapas principais dentro do processo de teste de BD:

(1) Colocar o BD em um estado conhecido antes de rodar qualquer teste: Ambler (2003) mostra duas maneiras de se realizar essa tarefa dentro de uma Sandbox:

- Reconstruir a base de dados, incluindo a re-criação do esquema e o re-carregamento dos dados de teste
- Reinicializar valores dos dados existentes no BD: Isso pode ser feito apagando todos os dados e inserindo os novos, ou realizando Updates. Ambler (2003) aponta que a primeira opção apresenta menores riscos, e em casos de grandes volumes de dados (o que é uma característica de sistemas de BI) pode ser até mais eficiente.

Dentro desta etapa ainda é necessária a criação de dados de teste a serem utilizados. Ambler(2006) mostra que esses dados podem ser criados das seguinte formas:

- **Possuir dados de origem externos:** Há possibilidade de se manter uma definição externa de dados, em arquivos, XML ou conjunto secundário de tabelas, de onde eles serão carregados para a realização de cada teste.
- **Manter Scripts de criação de dados:** São desenvolvidos e mantidos Scripts, que podem usar linguagem de manipulação de dados (DML) ou mesmo código de aplicação, e que são responsáveis pelas deleções, inserções ou Updates necessários para criar os dados de teste
- **Encapsular os dados no teste:** Ao rodar um determinado teste individual, o mesmo coloca o BD em um estado conhecido e requerido para sua execução

As duas últimas maneiras de se gerarem os dados de teste ainda possibilitam que os scripts e testes sejam armazenados dentro do repositório de código.

(2) Rodas os testes de BD utilizando uma ferramenta de testes de regressão: Para Ambler (2006), testes de regressão são testes que tem como objetivo, detectar erros (ou

regressões), que foram introduzidos no sistema após a implementação de alguma modificação, afetando funcionalidades já existentes.

Esses testes são realizados, através de ferramentas que possibilitem tanto automatizar esses testes, como reverter o BD para um estado conhecido. Exemplos de ferramentas desse tipo são:DBunit,DBFit,XTUnit.

Essa etapa se subdivide em três atividades:

Carregar uma amostra de dados fixa: Os dados a serem testados devem possuir o menor volume possível, porém ao mesmo passo devem conter um exemplo representativo dos dados encontrados no ambiente de produção, ou seja, as diversas variações referentes a valores devem ser simuladas no ambiente de teste a fim de encontrar áreas de insuficiência.

O ambiente de testes deve replicar o ambiente de produção, porém com os dados de teste em lugar dos dados do ambiente de produção. Carregar esses dados deve ser uma etapa precursora ao teste do processo,e deve poder ser feita de uma maneira rápida.Uma ferramenta de geração dedados como Data Factory,pode ser utilizada para isso.

Executar o processo a ser testado: Aqui entram os testes unitários, sendo que o processo sob teste deve ser o menor e mais coeso o possível, em outras palavras, deve realizar uma tarefa especifica bem definida, evitando a criação de scripts SQL grandes ou processos de ETL que realizam uma grande variedade de tarefas devido ao fato destas estarem relacionadas ou acopladas. O desenvolvimento ágil foca em criar componentes pequenos e concisos, sendo que, desta maneira, os testes podem ser simplificados.

Verificar os dados resultantes: Deve-se estabelecer uma relação entre os dados resultantes do processo testado e o conjunto de dados esperados (o que é, ou não, correto, de estar incluído no conjunto de dados resultante).

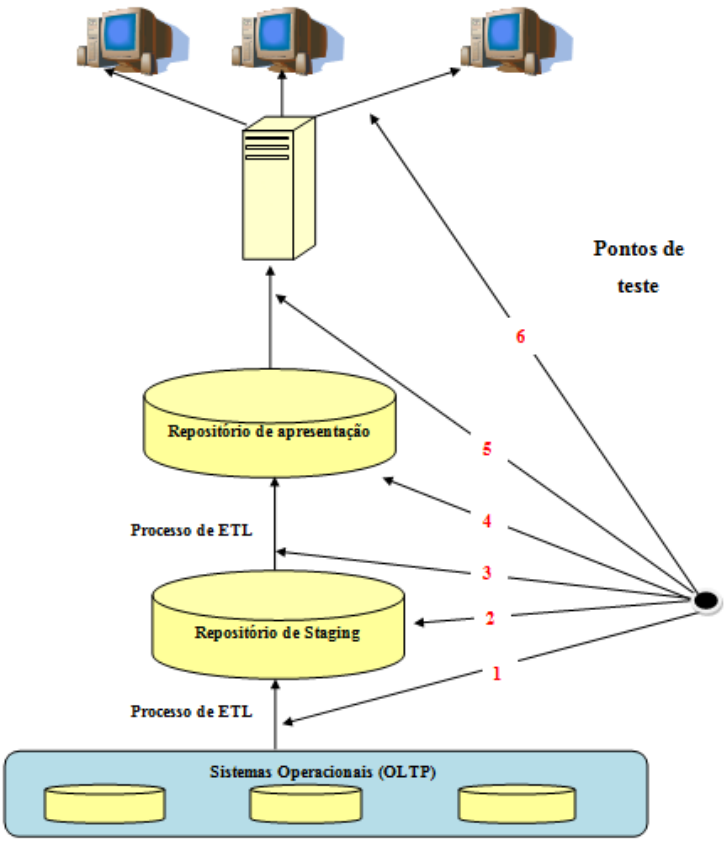
Este procedimento de comparação e avaliação deve ser realizado, também, sempre que um elemento for adicionado aos dados de entrada, sendo mandatório que todos os resultados após a execução do processo estejam em conformidade com os valores esperados (esse procedimento é permitido pela automatização de testes).

(3) Retornar o ambiente para o estado anterior ao teste

A ultima etapa corresponde a retornar o BD para o estado conhecido antes da execução a fim de se evitar o acoplamento entre testes.

4.3.5.7 Pontos de testes na arquitetura de um DW

Collier (2011), mostra que existem diversos pontos dentro da arquitetura do DM onde é crucial a execução de testes:



1. Códigos de atualização/preparação de dados, compostos pelos scripts de ETL que extraem dados dos sistemas operacionais (ou legados) devem ser validados bem como qualquer código utilizado para corrigir, formatar, limpar, ou seja, que envolva uma transformação de dados para a carga na área de staging.
2. Códigos de preparação, limpeza e correção de dados para armazenamento na área de staging
3. Códigos que transformam os dados da área de staging para os esquemas multidimensionais (esquemas estrelas, por exemplo).
4. Qualquer código ou script utilizado para manipular os esquemas multidimensionais a fim de se obter dados derivados ou transformados.

5. Camada de acesso aos dados: Inclui as aplicações do lado servidor que provém aos usuários acesso ao DM. Isso pode incluir especificações de cubos OLAP por exemplo.
6. Camada da aplicação de BI, que envolve as aplicações que apresentam os dados aos usuários e permitem que eles executem querys no DM.

A fim de se integrar testes ao desenvolvimento de uma aplicação de BI (no caso um Data Mart), e levando em consideração essa arquitetura, esses pontos, para Ken Collier (2011), refletem as áreas onde há manipulação pesada de dados dentro do sistema, daí a necessidade de execuções de testes nesses pontos.

CONCLUSÕES

Ao tratarmos de desenvolvimento de Data Mart,a alta mutabilidade dos requisitos dos usuários e a dificuldade que se tem de determinar requisitos no início,ou mesmo ao longo do projeto,acabam por caracterizar um cenário muito propício a utilização de práticas ágeis devido a adaptabilidade necessária ao se abordar projetos desta natureza.

Para que a adoção de metodologias ágeis em um projeto de Data Mart ocorra, de maneira que o processo de desenvolvimento se mantenha sustentável ao longo das etapas do projeto e sejam usufruídas as vantagens deste tipo de abordagem, é necessário que exista um ambiente que dê suporte ao uso de processos ágeis.

Esse ambiente é criado inicialmente pela modificação da cultura da própria organização ou setor de TI no que se refere a desenvolvimento de software, a fim de assimilar idéias como refatorações e testes de bases de dados, por exemplo, para que esta cultura não se torne um obstáculo, no momento de implantação de um processo extremamente iterativo, incremental e dinâmico. Caso isso não ocorra, técnicas e práticas fundamentais para se obter agilidade dificilmente seriam colocadas em prática pela equipe, o que invalidaria a abordagem.

Especificando-nos na série de práticas que sustentam a agilidade, temos o desenvolvimento evolutivo das bases de dados utilizadas pelo Data Mart como sendo o pilar principal que permite a adoção do desenvolvimento iterativo e que fornece um grau maior de flexibilidade no projeto.

A evolução das bases de dados irá se fundamentar nos testes e refatorações destas bases, feitas ao longo do projeto, tanto a caráter de correções, como para possibilitar a implementação de novas funcionalidades. Aqui há um ponto de grande importância que se refere à necessidade de se utilizar ferramentas que automatizem o processo de testes e refatorações em banco de dados. Essas ferramentas,bem como a idéia de se evoluir um esquema de bases de dados,da mesma forma que se evolui uma aplicação iterativamente, estão emergindo e se maturando no mercado ainda,o que caracteriza um obstáculo ao tentarmos adotar estas práticas ao ciclo de desenvolvimento se analisarmos o número de desenvolvedores aptos a usarem tais ferramentas e a dificuldade de penetrar tais conceitos e práticas dentro de uma organização a qual estas práticas são novas(este problema ainda se refere a cultura da organização ,no que se refere a desenvolvimento,citada anteriormente).

O modelo evolutivo de banco de dados assegura a construção de um produto que atenda as necessidades e agregue valor aos clientes, já que há a integração no ciclo de desenvolvimento, das atividades de teste e entrega contínua de funcionalidades, para se obter o feedback do cliente ao longo de todo o projeto.

Essa integração de atividades (modelagem, codificação, testes), através das práticas de TDD e refatoração, por exemplo,implicam em uma melhoria na qualidade do software,tanto do ponto de vista técnico como do usuário,de modo que os testes e validações não são postergados para o fim do projeto.

Há uma ressalva, no entanto, no que diz respeito à colaboração do cliente, e presença do mesmo continuamente dentro de todo o desenvolvimento, que as metodologias ágeis enfatizam. Essa realidade não existe em diversos projetos,o que torna algumas práticas ágeis,como o trabalho contínuo com o cliente no mesmo local de trabalho difíceis de serem realizadas.Isso não invalida a possibilidade de se desenvolver testes de aceitação com o cliente,e de demonstrar funcionalidades parcialmente ao longo do projeto para a validação.

O ambiente ágil conta com a integração de diversas etapas do ciclo de desenvolvimento a cada iteração. A característica da equipe também é influenciada, de modo que este ambiente requer freqüentemente que desenvolvedores adquiram habilidades genéricas ,ou diversificadas.Isso é visível ao se adotar TDD,onde os programadores devem ser capazes de codificar rotinas de testes(função que poderia seria incumbida a testers somente) ,ou quando é necessário analisar modificações em bases de dados e validá-las com um DBA para aplicar uma refatoração,por exemplo.

Por fim,o objetivo da agilidade em um projeto de BI, não é apenas o encurtamento do ciclo de desenvolvimento, mas sim, a diminuição de risco do projeto através da adaptabilidade adquirida no processo. Somadas, as práticas usadas são capazes de fornecer um framework para a otimização dos processos de construção do Data Mart.

REFERÊNCIAS

Humphrey, W. S. 1995. **A Discipline for Software Engineering**. Addison Wesley Longman, Inc. 242 p.

Larman, C. 2004. **Agile and Iterative Development: A Manager's Guide**.Pearson Education, Inc. Boston. 342 p.

Fowler, M. 1999. **Refactoring: Improving the Design of Existing Code**. Menlo Park, CA: Addison-Wesley Longman.

Fowler M. **Evolutionary Database Design**.2003.Disponível em :< http://martinfowler.com/articles/evodb.html>. Acessado em: 25 de agosto de 2011

Fowler M. **Continuous Integration**.2006.Disponível em:<http://www.martinfowler.com/articles/continuousIntegration.html>.Acesso em 30 de agosto de 2011.

Abrahamsson .P;Salo O.**Agile Software Development methods Review and Analysis**.2002.

Collier,K. **Agile analytics : A Value Driven Approach to Business Intelligence and Data Warehousing**.Estados Unidos:Prenctice Hall,2011.384 p.

Kimball,R.**The Data Warehouse Toolkit:The Complete Guide to Dimensional Modeling**.Estados Unidos: John Wiley and Sons Inc,2002.464 p.

Kimball,R.**The Data Warehouse ETL Tollkit**. Estados Unidos: John Wiley and Sons Inc,2004.528 p.

Highsmith,J.**Agile Project Management: Creating Innovative Products, Second Edition**.Addison-Wesley Professional.2009.392 p.

W.H,Inmon.**Building the Data Warehouse**. Estados Unidos: John Wiley and Sons Inc,1991.

Ambler,S ; Sadalage P. Refactoring **Databases :Evolutionary Database Design**. Addison Wesley Professional.2006.384 p.

Ambler,S. Agile **Modeling:Effective Practices for extreme programming and the Unified Process**. Estados Unidos: John Wiley and Sons Inc,2002.402p.

Leffingwell D.**Agile Software Requeriments**.Estados Unidos:Pearson Education Inc 2011.560p.

Ambler,S.**Agile Database Technique - Effective Strategies for the Agile Software Developer**. John Wiley and Sons Inc.2003.373 p.

Factor P.**The Red Gate Guide to SQL Server Team-based Development**.2011.

Beck,K.**Extreme Programming Explained:Embrace Change**.1999.190 p.

Ponniah.P.**Data Modeling Fundamentals**. John Wiley and Sons Inc.2007.460 p.

Carvalho.T.C.**Aplicação de práticas ágeis na construção de Data Warehouse evolutivo**.2009.116 p.Dissertação (Mestrado em ciências da computação)-Universidade de São Paulo.

Highsmith,J;Cockburn,A.**What Is Agile Software.Development?.**2002.

Palmer,Stephen R.**A practical Guide to feature-driven Development**.304 p.

Standish Group.**Chaos Report**,1995.

Salo O.**Enabling Software Process Improvement in Agile Software Development Teams and Organisations**.2006.

Fowler M. **The New Methodology**.2005.Disponível em:<(http://martinfowler.com/articles/newMethodology.html)>.Acesso em 25 agosto de 2011

Larman C.**Agile and iterative development: a manager's guide**.2004.342p.

Cockburn.A.**Agile Software Development.**2000.