UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Visualizando Falhas reportadas nos SIGs/UFRN utilizando o iProject e o Suite Pentaho

Diego Jácome de Medeiros

Natal-RN

Dezembro de 2013

Diego Jácome de Medeiros

Visualizando Falhas reportadas nos SIGs/UFRN utilizando o iProject e o Suite Pentaho

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte.

Orientador

Roberta Coelho

Universidade Federal do Rio Grande do Norte – UFRN

Departamento de Informática e Matemática Aplicada – DIMAp

Natal-RN

Dezembro de 2013

Monografia de Graduação sob o título *Visualizando Falhas reportadas nos SIGs/UFRN utilizando o iProject e o Suite Pentaho* apresentada por Diego Jácome de Medeiros e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Dra. Roberta de Souza Coelho

Departamento de Informática e Matemática Aplicada - DIMAP

Universidade Federal do Rio Grande do Norte - UFRN

Dr. Eduardo Henrique da Silva Aranha

Departamento de Informática e Matemática Aplicada - DIMAP

Universidade Federal do Rio Grande do Norte - UFRN

Gestor Dalton Dantas de Oliveira

Departamento de Informática e Matemática Aplicada - DIMAP

Universidade Federal do Rio Grande do Norte - UFRN

Natal-RN, doze de dezembro de 2013.

Agradecimentos

Eu gostaria de agradecer a minha querida avó Socorro, que sempre foi minha figura materna, ao meu inocente pai Ivanaldo, a meu “timbuzástico” irmão João Vitor e minha amada companheira Dalva. Também gostaria de agradecer todos da minha família que de alguma forma contribuíram na minha criação.

Eu gostaria de agradecer aos meus professores, em especial a minha orientadora Roberta Coelho e a Luiz Amorin e André Maurício cujas as aulas de Teoria dos Grafos e Algoritmos 2 me trazem recordações de sutil deliciamento.

Gostaria de agradecer ao meus compadres de bar Igor Linnik, Breno Ramos, Rafael Figueiredo, Pablo de Melo, Suelton Miguel, Victor Pereira, Lucas Procópio e Tyago Tayrony (este último não frequenta bar, mas está no meu coração).

Também não posso esquecer dos meus amigos de D&D, pois a monografia pode atrasar, mas os dados precisam rolar.

É claro, também tenho que agradecer a meus amigos de trabalho, pois posso conversar com eles sobre computação sem ser chato.

Não menos importante, gostaria de agradecer aos personagens fictícios que ajudaram a moldar minha personalidade e caráter. Em especial ao Super-Homem por ter dado um toque de heroísmo aos meus valores morais. Ao Conan por ter me ensinado que é necessário lutar com unhas e dentes nesta vida. E a Frodo Bolseiro por ter me ensinado que por mais dura que seja a jornada sempre existe a esperança.

Visualizando Falhas reportadas nos SIGs/UFRN utilizando o iProject e o Suite Pentaho

RESUMO

Durante o ciclo de vida de uma aplicação, várias falhas são encontradas. Algumas falhas são encontradas durante atividades de testes, outras são identificadas pelos usuários finais da ferramenta (após a ferramenta entrar em produção). As falhas encontradas são uma preciosa fonte de informação para empresas de desenvolvimento de software, uma vez que através da análise das falhas encontradas podem ser realizadas melhorias no produto e no processo de desenvolvimento. Neste trabalho apresentamos duas ferramentas que foram construídas com o objetivo de auxiliar gerentes e engenheiros de software na análise de falhas (i) encontradas durante atividades de testes exploratórios (registradas pelas equipes de controle de qualidade) e (ii) reportadas pelos usuários após o sistema entrar em produção (registradas pela equipe de suporte ao usuário de uma corporação). A primeira ferramenta foi desenvolvida com tecnologia Java para Web e foi integrada ao iProject - um sistema de gestão de projetos da Superintendência de Informática da UFRN (SINFO). A segunda ferramenta foi desenvolvida utilizando a plataforma *Open Source Pentaho* de *Business Intelligence* e provê um cubo OLAP interativo para que os gestores possam navegar na informação. Neste trabalho, também foi realizada uma breve comparação do custo de desenvolvimento e das funcionalidades providas por cada ferramenta.

Palavras-chave: *Business Intelligence*, *Data Warehouse*, Falhas, ETL, Modelagem Dimensional, OLAP, Pentaho. Testes de Software, Testes Exploratórios.

Viewing Failures reported in SIGs/UFRN using iProject and Pentaho Suite

ABSTRACT

During the lifecycle of an application, several faults are found. Some faults are found during the testing activities, others are identified by end-users of the tool (when the tool goes to the production environment). The faults found are a valuable source of information for software development teams, once through the analysis of such faults the development team can improve the product and the process. In this work we present two tools that have been built with the objective of helping managers and software engineers in fault analysis (i) found during exploratory testing activities (recorded by quality control team) and (ii) reported by user after the system goes into production (recorded by the user support team of a corporation). The first tool was developed with Java technologies for web-development and has been integrated into iProject – a project management system of UFRN's Superintendency of Informatics (SINFO). The sesond tool was developed using the Pentaho Open Source Business Intelligence platform and provides an interactive OLAP cube so that managers can navigate the information. In this work, a brief comparison of the cost of development and the functionality provided by each tool was also performed.

Palavras-chave: *Business Intelligence*, *Data Warehouse*, Faults, ETL, Dimensional Modeling, OLAP, Pentaho. Software Testing, Exploratory testing.

Lista de Figuras

Sumário

1 Introdução

Este Capítulo é destinado a introduzir o trabalho. A Seção 1.1 discorre sobre as motivações que levaram o desenvolvimento do trabalho. A Seção 1.2 apresenta quais são os objetivos do trabalho. Por fim, a Seção 1.3 mostra como o documento está organizado.

1.1 Contexto e Motivação

A tomada de decisões é uma importante atividade na vida das empresas e na carreira de profissionais executivos. A tecnologia da informação oferece diversas ferramentas para facilitar o processo decisório, permitindo que os profissionais da área tenham acesso a dados e análises cada vez mais elaboradas.

No contexto da engenharia de software, a responsabilidade dos sistemas em grandes organizações crescem cada vez mais, consequentemente, a qualidade do software se torna um fator essencial. A presença de falhas no sistema não só gera insatisfação do usuário, como aumenta o custo do projeto devido a manutenção corretiva. É necessário que o engenheiro de software disponha de uma visão adequada sobre as falhas que ocorrem para que possa tomar as decisões mais eficientes para sanar o problema.

Para que estas decisões sejam tomadas, é importante que as falhas do sistema sejam catalogadas e organizadas de modo que os dados possam se tornar informações úteis. Tais falhas podem ser registradas durante a fase de testes, antes do software entrar em produção. Assim como, através de *feedbacks* recolhidos pelo suporte ao usuário, quando o software já está no ambiente de produção. Uma vez que os dados estejam sendo catalogados, é preciso que eles

sejam exibidos de forma que gerem conhecimento de negócio. Para isso é possível codificar manualmente relatórios ou utilizar ferramentas especializadas em análise negocial, como plataformas de *Business Intelligence*.

1.2 Objetivos

Este trabalho tem como objetivo propor e implementar ferramentas que possibilitem a análise de falhas ocorridas em softwares e registradas no processo de controle de qualidade ou suporte ao usuário. As ferramentas desenvolvidas possuem como meta principal gerar análises capazes de auxiliar o processo de tomada de decisões dos gestores do sistema SIG (Sistemas Integrados de Gestão), os sistemas corporativos da UFRN. Os objetivos específicos do projeto são:

1. Estudar técnicas, mecanismos e ferramentas existentes na área de Business Intelligence.
2. Propor uma ferramenta Java Web e uma ferramenta de Business Intelligence, que possam capturar os dados sobre as falhas ocorridas no sistema e exibi-las ao usuário.
3. Projetar e implementar tais ferramentas. A primeira utilizando a arquitetura padrão dos sistemas SIG. A segunda utilizando uma plataforma moderna de Business Intelligence.
4. Fazer uma análise comparativa de ambas as ferramentas, analisando os aspectos positivos e negativos da implementação e funcionalidades de cada uma.

1.3 Organização do Documento

Este documento está organizado em mais 6 Capítulos, além desse introdutório.

O Capítulo 2 apresenta a fundamentação teórica investigada para o desenvolvimento do trabalho, introduz os conceitos básicos de teste de software e apresenta a teoria por trás de aplicativos de *Business Intelligence*. O Capítulo 3 apresenta as principais ferramentas utilizadas no desenvolvimento da aplicação BI. O Capítulo 4 descreve sucintamente os sistemas institucionais da UFRN. O Capítulo 5 mostra detalhadamente a ferramenta Web para visualização de falhas, discorrendo sobre o contexto, a arquitetura e a implementação. O Capítulo 6 apresenta a ferramenta BI descrevendo o modelo dimensional desenvolvido, o processo de ETL e o cubo OLAP, mostra algumas métricas que foram obtidas através da ferramenta BI e faz uma breve discussão sobre as ferramentas desenvolvidas. Finalmente, o Capítulo 7 apresenta considerações finais, descrevendo as contribuições do trabalho, assim como possíveis trabalhos futuros.

2 Fundamentação Teórica

Neste Capítulo serão apresentados os fundamentos teóricos das áreas abordadas nesse trabalho. A Seção 2.1 apresenta os conceitos básicos relacionados a área de Teste de Software. A Seção 2.2 apresenta brevemente os principais conceitos de testes exploratórios. A Seção 2.3 descreve de modo geral a área de *Business Intelligence* (BI). A Seção 2.4 discorre sobre *Data Warehouses* e modelagem dimensional. A Seção 2.5 apresenta o processo de ETL (do inglês: *Extract, Transform and Load*) realizado na construção de um *Data Warehouse*. A Seção 2.6 discorre sobre os principais conceitos de ferramentas OLAP. Por fim, a Seção 2.7 faz as considerações finais sobre o Capítulo.

2.1 Conceitos Básicos

Durante o processo de desenvolvimento de software, mesmo se forem utilizados os melhores métodos, as melhores ferramentas e os melhores profissionais, o produto ainda está sujeito a falhas, visto que a inserção de um defeito no software é um ato humano.

Para promover a melhor compreensão e estudo dos conceitos relacionados a erros de software a IEEE faz vários esforços para padronizar a terminologia. O padrão IEEE número 610.12-1990 define os seguintes termos:

- *Defeito (fault)*: Passo, processo ou definição de dados incorretos, por exemplo: uma instrução ou comandos errados. Na linguagem comum, os termos “bug“ ou “erro“ são utilizados para expressar um defeito.

- *Engano (mistake)*: Ação humana que produz um resultado incorreto, como exemplo, a inserção de um trecho de código inconsistente. A manifestação de um “engano“ é um “defeito“.
- *Erro (Error)*: A diferença entre o valor ou condição obtida e o valor ou condição verdadeira, esperada ou teoricamente correta. Ou seja, qualquer estado intermediário incorreto ou resultado inesperado durante a execução do programa.
- *Falha (Failure)*: A incapacidade do software de realizar determinadas funções em relação à especificação. Ou seja, comportamento diferente do esperado pelo usuário.

Figura 1: Defeito x Erro x Falha Fonte: (Dias Arilo 2009)

A Figura 1 delimita as fronteiras dos conceitos. O *defeito* é uma inconsistência na aplicação propriamente dita e são inseridos inconscientemente por pessoas. Os *defeitos* podem gerar *erros*, ou seja, comportamentos diferentes do que foi especificado. Por fim, os *erros* geram *falhas* que são comportamentos inesperados que afetam o usuário final do sistema [Dias Neto 2009].

Segundo Dias Neto [Dias Neto 2009], no processo de desenvolvimento de software, o tamanho do projeto e o número de profissionais envolvidos na atividade são dois possíveis fatores que aumentam a complexidade da tarefa, consequentemente também

aumentam a probabilidade da ocorrência de defeitos. Por esta razão, o surgimento de falhas é inevitável.

Apesar de todos os esforços feitos para reduzir o número de erros, os defeitos surgem ao longo de todo ciclo de desenvolvimento do software gerando falhas para o usuário final. Abaixo segue um lista de exemplos da relação defeito-falha nos diversos ciclos de desenvolvimento.

1. Especificação errada ou incompleta pode gerar funcionalidades diferentes das esperadas pelo usuário, ou até mesmo ausentes.
2. Requisitos impossíveis de serem implementados devido a limitações de hardware e software podem impedir que determinados critérios de qualidade sejam satisfeitos.
3. Um projeto mal construído, como uma base de dados desorganizada, podem dificultar a evolução do software.
4. Codificação mal feita, podem adicionar defeitos no algoritmo capazes de ocasionar falhas de execução.

2.2 Testes Exploratórios

Nesta Seção serão apresentados os fundamentos de testes exploratórios. A Subseção 2.2.1 irá discorrer sobre os conceitos básicos de testes exploratórios. A Subseção 2.2.2 irá apresentar os testes exploratórios baseados em sessão que são utilizados pela equipe de controle de qualidade da SINFO/UFRN.

2.2.1 Conceitos Básicos

O teste exploratório é uma técnica de testes que enfatiza as habilidades do testador em tomar decisões sobre o que está sendo testado durante a execução do teste ao invés de seguir um roteiro planejado [Caetano, 2012]. Em uma definição mais básica, o teste exploratório é a criação e a execução ao mesmo tempo de um teste. O testador não possui informações detalhadas sobre o que vai testar e utiliza de seu conhecimento e experiência para a realização do teste, de tal maneira que a intuição, a criatividade e a habilidade do testador são indispensáveis para garantir a eficiência do teste [Caetano, 2006]. De acordo com o *Software Engineering Body of Knowledge* [SWebOK04], testes explanatórios é a técnica de teste mais utilizada.

As principais características dos testes exploratórios são [Bolton, 2008]:

- A criação, a execução, a interpretação e o aprendizado são feitos pelas mesmas pessoas.
- A criação, a execução, a interpretação e o aprendizado acontecem juntos, ao invés de serem executados em diferentes momentos no tempo.
- O testador faz suas próprias escolhas sobre o que será testado, quando testar e como testar, ao invés de seguir um roteiro.
- Tudo que foi aprendido durante o teste, serve de subsídio para realização de um novo teste.
- O testador foca em revelar novas informações sobre o produto, no lugar de confirmar as informações já conhecidas.
- O testador pode variar os seus testes, ao invés de repeti-los continuamente.

Segundo Caetano, [Caetano, 2006], é possível destacar as seguintes razões para a elaboração de testes exploratórios:

- Não existência de requisitos.
- Pouco tempo disponível.
- Não se conhece o aplicativo a ser testado.
- Ambientes poucos testados por testes convencionais.
- Tentativa de reproduzir defeito aleatório.
- Diagnóstico de comportamentos inesperados.
- Investigação de efeitos colaterais.
- Investigação de defeitos semelhantes.
- Medição de riscos.
- Determinação de defeitos críticos rapidamente.

2.2.2 Testes Exploratórios Baseados em Sessões

Testes exploratórios baseados em sessões é uma estratégia utilizada para tornar testes exploratórios mais efetivos e com objetivos mais claros [Caetano, 2006]. Testes exploratórios baseados em sessões permitem quantificar: (i) quanto trabalho foi feito; (ii) quais partes do software foram testadas; (iii) quanto tempo foi gasto durante o teste [Quinn, 2013].

Nesta abordagem, o teste exploratório é realizado em sessões de cerca de 60 minutos, podendo variar de 45 à 90 minutos. A idéia principal da sessão é garantir que durante esse período de tempo o testador permaneça totalmente focado no teste e não seja interrompido [Caetano, 2006].

Uma vez que o teste esteja completo, deve-se reportar os aspectos do trabalho para o gerente. Deve ser reportado as falhas

encontradas e um resumo do teste feito para que as informações sejam utilizadas para criar métricas da sessão [Quinn, 2013].

2.3 Business Intelligence

O termo *Business Intelligence* como é utilizado atualmente surgiu em 1989 criado pelo Gartner Group [Buchanan et al., n.d.]. O conceito descreve as habilidades de acesso, exploração, análise e manipulação de dados através de um vasto conjunto de aplicações, com o objetivo de transformar grandes quantidades de informações em conhecimento útil.

Numa pequena síntese, *Business Intelligence* possui como principais fundamentos o acesso a dados confiáveis, o aumento e a transparência e da compreensão do negócio e o suporte de tomada de decisões.

Um sistema de BI é enquadrado na parte de infra-estrutura de um sistema de informação. Essencialmente, o sistema BI não existe sozinho e precisa estar ligado a fonte de dados para coletar informações afim de gerar conhecimento. Tal conhecimento é produzido através de várias ferramentas e interfaces de visualização que agem sobre as informações coletadas [Sezões et al., 2006].

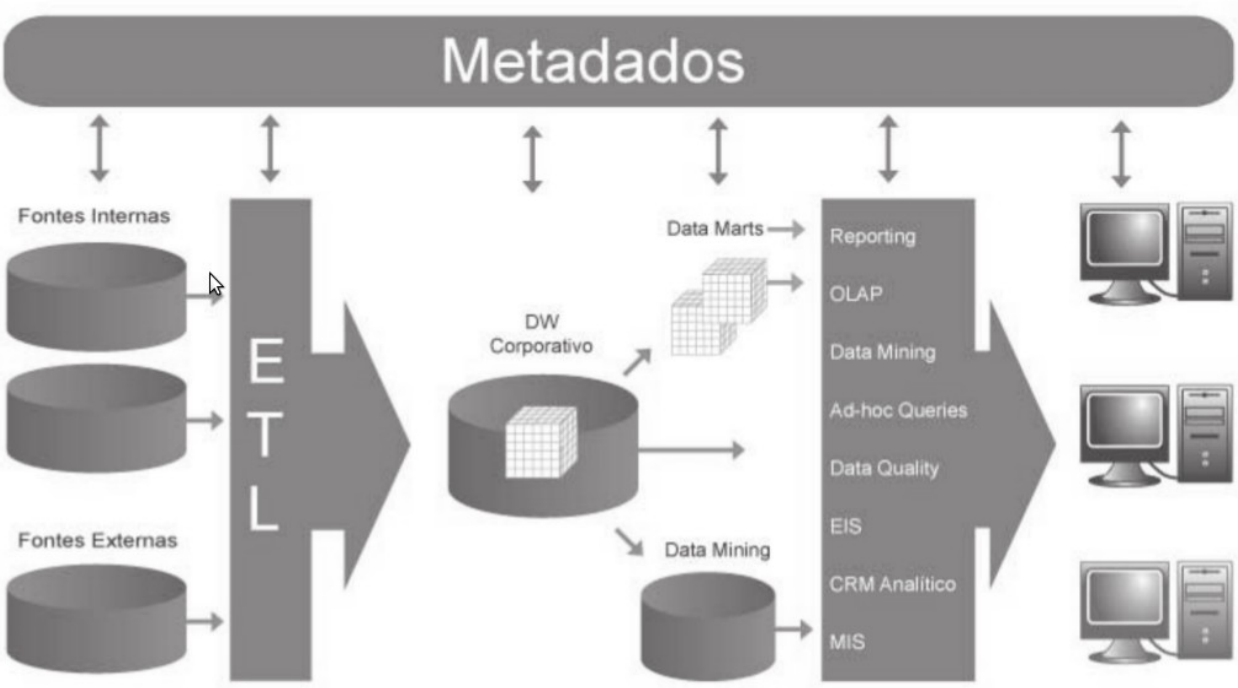


Figura 2: Arquitetura de um Sistema de BI Fonte: (Sezões, et al. 2006)

A arquitetura padrão de um sistema BI, ilustrada na Figura 2 é composta pelas seguintes partes:

- Módulo ETL (*Extract, Transfer and Load*): Componente responsável pela extração, pelo carregamento e pela transformação de dados. Se dedica a recolher informações nas mais diversas fontes (sistemas ERP, arquivos TXT ou ficheiros Excel);
- *Data Warehouse / Data Marts*: Locais ondem ficam armazenados os dados extraídos de diferentes sistemas. A vantagem desses repositórios é a possibilidade de armazenar as informações históricas e operacionais, facilitando operações de análises visando a tomada de decisões.
- *Front End*: É a interface final com o usuário. Se constitui desde relatórios padronizados, análise OLAP, *dashboards*, *data mining*, etc.

2.4 Banco de Dados, *Data Warehouse* e modelagem de dados

Nesta Seção serão apresentados os conceitos fundamentais de *Data Warehouses*, uma importante parte da arquitetura de sistemas de *Business Intelligence*. Na Subseção 2.4.1 será mostrado o contexto histórico que levou a criação dos *Data Warehouses*. Na Subseção 2.4.2 é definido o que *Data Warehouses*. Na Subseção 2.4.3 são mostradas as diferenças entre um banco de dados relacional e um *Data Warehouse*. A Subseção 2.4.4 discorre sobre a modelagem dimensional. Por fim, a Subseção 2.4.5 descreva o esquema estrela, uma forma popular de modelagem dimensional.

2.4.1 Contexto Histórico

Com o aumento dos sistemas de apoio tomadas de decisões na década de 80, foi necessário a criação de uma nova tecnologia para melhor administração de dados, visto que os sistemas transacionais desenvolvidos originalmente para satisfazer necessidades operacionais sofriam de dificuldade para gerenciar informações.

Segundo Inmon [Inmon 2005], a extração de dados de sistemas OLTP (*on-line transaction processing*) era dificultada por 3 fatores: a credibilidade dos dados, a baixa produtividade e a dificuldade de transformar dados em informações. Inmon aponta 5 causas para estes problemas:

1. Dados não baseados no tempo: As aplicações não haviam sido construídas com a intenção de manter dados históricos para análise negocial.

2. Diferença entre algoritmos: A aplicação de diferentes algoritmos trazia resultados diferentes capazes de surpreender os usuários, de modo que a credibilidade no sistema fosse afetada.
3. Diferentes níveis de extração de dados: Ocorre quando é feito uma extração de dados de uma outra extração feita anteriormente, fazendo com que os problemas acima sejam acentuados.
4. Dados externos: Quando dados externos são trazidos para aplicação, a origem dos dados não são capturadas, fazendo com que eles se tornem dados genéricos de fontes não identificadas.
5. Diferentes bases de dados: Com a ocorrência de diferentes bases de dados, se a análise for feitas nas bases separadas, corre o risco de resultados divergentes. Por outro lado, localizar e escrever programas para unificar os dados é custoso.

É neste contexto que surgem os *Date Warehouses*.

2.4.2 Definição de *Date Warehouse*

A definição mais comum da literatura define *Data Warehouse* como uma coleção de dados que são [Mundin, 2009]:

- Integrados: O *Data Warehouse* não necessita acender a várias fontes de dados. Os dados deverão ser carregados no *Data Warehouse* e convertidos para uma única codificação.
- Organizados por assunto: Dados que são utilizados em aplicações operacionais possuem detalhes que podem ser irrelevantes para a análise de negócio. Os dados dos *Data*

Warehouse são compartimentados nos principais assuntos e negócios dos utilizadores. [Inmon, 2005]

- Variam no tempo: *Data Warehouses* possuem o histórico da informação, eles devem manter algum elemento de registro temporal que permita a análise da evolução histórica.
- Não são voláteis: Um vez que os dados sejam adicionados no *Data Warehouse* eles não são alterados ou removidos, de forma a fornecer registros históricos corretos e credíveis.
- Acessíveis: A função primária do *Data Warehouse* é fornecer acesso a registros de forma rápido e fácil [Sezões et al., 2006].

Os *Data Warehouses*, agrupam uma vasta gama de informações, que podem ser divididas em conjuntos menores de dados, agrupados de forma lógica. Essas unidades menores são chamadas de *Data Marts* [Sezões et al., 2006].

2.4.3 Diferença entre *Data Warehouses* e Banco de Dados Transacionais

Segundo Comem, [Comem, 2001], *Date Warehouses* diferem dos bancos de dados transacionais em diversos níveis.

Os dados do bancos de dados transacionais são fisicamente diferentes dos dados utilizados nos *Data Warehouses*. Enquanto os dados do banco de dados são utilizados para operações do dia à dia, os dados do *Data Warehouse* são utilizados para atender necessidades informacionais. A modelagem dos dados utilizados nos bancos de dados transacionais também diferem da modelagem adotada por *Data Warehouses*. Bancos de dados transacionais utilizam a modelagem entidade/relacionamento, os *Data Warehouses* utilizam modelagem dimensional.

Além disso, o processamento que é realizado em cima dos dados também diferem. As operações realizadas nos bancos de dados são procedimentos repetitivos que permitem atualizar e remover dados. As operações realizadas nos *Data Warehouses* costumam ser algoritmos heurísticos, programas não repetitivos e procedimentos. Os dados dos *Data Warehouses* podem ser recalculados, no entanto, não podem ser atualizados diretamente.

Por fim, as comunidades de usuários que serão atendidas por banco de dados transacionais é diferente da atendida por *Data Warehouse*, visto que a utilização do primeiro é feita pela comunidade operacional, enquanto a do segundo é feita pela comunidade gerencial.

A Figura 3 ilustra as diferenças entre um banco de dados transacional (Sistema Operacional) e um *Data Warehouse* (Sistema Informacional).

Característica	Sistema Operacional	Sistema Informacional
Tipo de dados	Detalhados	Detalhados e sumariados
Organização dos dados	Por aplicação	Por assunto
Estabilidade dos dados	Dinâmico	Estático
Qualidade dos dados	Na entrada	No processo (ETL[7])
Estrutura dos dados	Otimizados para transações	Otimizados para pesquisas complexas
Dados por transação	Poucos (dezenas)	Muitos (milhares)
Freqüência de acesso	Alta	Média para baixa
Volume de dados	Megabytes – Gigabytes	Gigabytes – Terabytes
Tipo de Informação	Atual e volátil	Histórica e não volátil
Operação	Atualização	Leitura e análise
Processamento	Dirigido à transação (OLTP[8])	Dirigido à análise (OLAP[9])
Uso	Operacional, repetitivo e estruturado	Informativo, analítico e não estruturado
Comunidade atendida	Funcional, com necessidades cotidianas. Decisões do dia-a-dia	Gerencial, com necessidades gerenciais. Decisões estratégicas. longo-prazo
Redundância	Não ocorre (normalizado)	Ocorre (desnormalizado)
Objetivo	Manutenção do negócio	Análise do negócio
Interação	Pré-definida	Pré-definida e ad hoc
Histórico	Baixo (até 3 meses)	Alto (até 10 anos)
Tempo de resposta	Até 2-3 segundos	De segundos a minutos
Atualização	Atualizado em tempo real	Atualizado periodicamente (*Batch*)
Disponibilidade	Alta	Atenuada

Figura 3: Diferença entre um sistema operacional e um sistema informacional Fonte: (Come, 2001)

2.4.4 Modelagem Dimensional

De acordo com Kimball, [Kimball, 2002], a modelagem dimensional é uma técnica utilizada para tornar o banco de dados mais simples e entendíveis e possui como objetivo dá suporte ao usuário final.

A principal diferença entre o modelo entidade/relacionamento é o grau de normalização. Modelos normalizados são úteis para operações transacionais de inserção e atualização. No entanto, possuem baixa performance na consulta de grande volumes de dados. O modelo dimensional compensa o desempenho com a redundância planejada dos dados [Ramos, 2012].

Kimball, [Kimball, 2002], metaforiza o modelo dimensional como uma abstração de um cubo, com as arestas rotuladas de tempo, loja e produto. Desta maneira se torna simples fatiar o cubo e se aprofundar em suas dimensões para extrair mais detalhes do processamento interno.

A Figura 4 ilustra a abstração de um cubo.

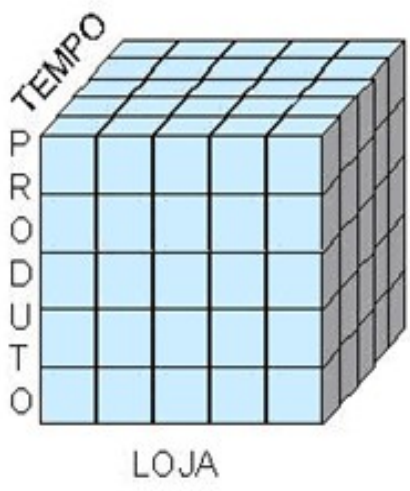


Figura 4: Abstração do Modelo Dimensional Fonte: (Moreira, 2006)

2.4.5 Esquema Estrela

O esquema estrela é talvez uma das maneiras mais populares de se construir uma estrutura de dados para *Data Warehouses* [Sezões et al., 2006]. Seu conceito supõe a criação de tabelas dimensionais que ficam ligadas entre si através de uma tabela fato. As tabelas dimensionais contêm as definições das características dos eventos, enquanto as tabelas fatos, armazenam os fatos decorridos e as chaves estrangeiras para as características que se encontram das tabelas dimensionais [Sezões et al., 2006].

Segundo Kimball, [Kimball, 2002], a tabela fato é a principal tabela do modelo dimensional, onde as medições numéricas do negócio estão armazenadas. Um registro da tabela fato possui um conjunto de colunas que correspondem as chaves primárias de cada uma das tabelas dimensionais no esquema estrela. Além disso, as colunas fatos contém também, colunas que descrevem volume, frequência, valor, ou qualquer outras medidas numéricas que possam ser agregadas (através de somas, contagens e médias) num *query* de SQL [Sezões et al., 2006].

As tabelas dimensionais são compostas por atributos que contém a descrição do negócio [Ramos, 2012], por exemplo, o nome do produto, o preço do produto, etc. Em geral, a quantidade de registros nas tabelas dimensionais é esmagadoramente menor do que a quantidade de registros nas tabelas fatos. A Figura 5 ilustra um esquema estrela onde a tabela fato é *sales_fact_1998* e as demais tabelas são tabelas dimensionais.

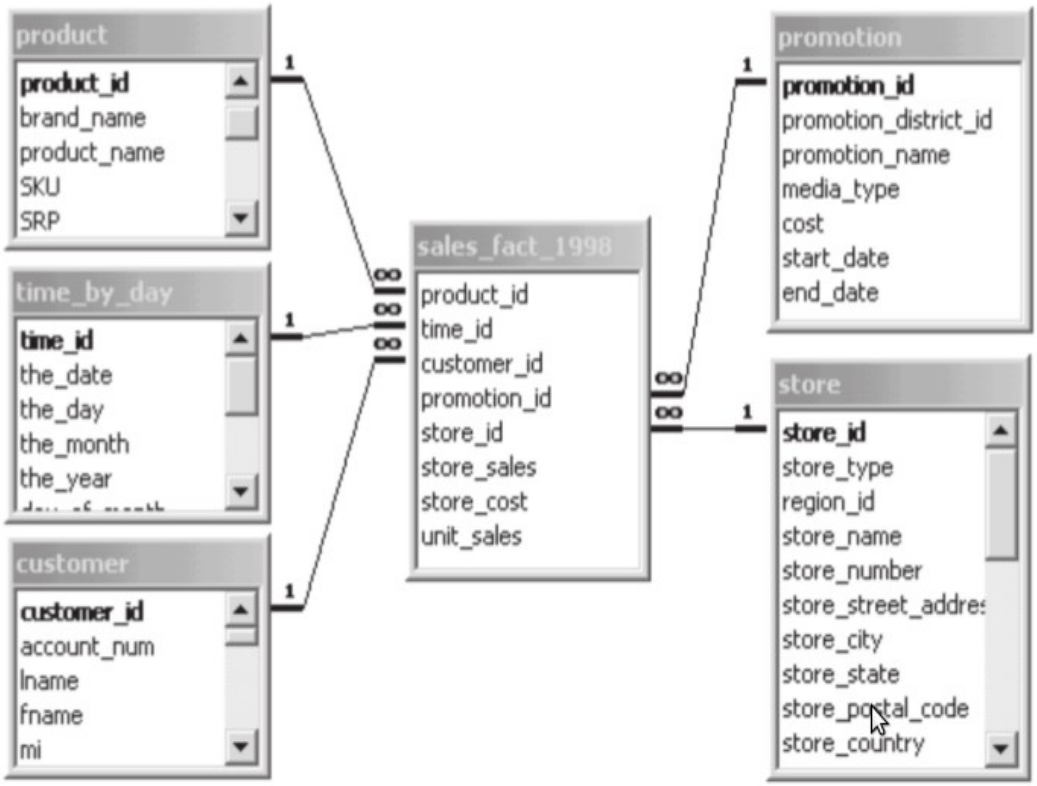


Figura 5: Esquema Estrela Fonte: (Sezões, et al. 2006)

A simplicidade do modelo, favorece o desempenho, visto que os otimizadores dos bancos de dados podem processar esquemas simples mais eficientemente com poucos *joins* e podem realizar fortes previsões sobre tabelas com índices apropriados [Kimball, 2002].

2.5 Módulo ETL

Nesta Seção serão apresentados os principais conceitos de um sistema ETL. Na Subseção 2.5.1 será definido o que é um sistema ETL. Na Subseção 2.5.2 serão mostrados os principais componentes de um sistema ETL.

2.5.1 Definição de Sistemas ETL

Para Kimball, [Kimball, 2004], os sistemas ETL (do inglês: *extract – transform – load*) são o alicerce de *Data Warehouses*. Um sistema ETL deve: (i) extrair dados de diferentes fontes; (ii) assegurar a qualidade e a consistência dos dados, de modo que dados extraídos de fontes separadas possam ser utilizados juntos; (iii) e por fim, entregar os dados no formato pronto para apresentação, para que desenvolvedores possam construir aplicações e o usuário final possa tomar decisões. Embora o sistema ETL não seja visível para o usuário final, eles consomem facilmente 70% dos recursos da implementação e manutenção de um *Data Warehouse* [Kimball, 2004].

2.5.2 Componentes de Sistemas ETL

Segundo Kimball, [Kimball, 2004] e [Kimball, 2008], Os sistemas ETL são formados por 4 principais componentes: a extração dos dados, a limpeza e consolidação dos dados (transformação), a entrega dos dados e o gerenciamentos dos dados.

A etapa inicial do processo de ETL é entender as diferentes fontes de dados e transferir os dados para o ambiente do *Data Warehouse*, onde o módulo ETL pode operar independente de sistemas operacionais. Em alguns casos, os dados capturados nesta etapa são descartados após o processo de limpeza. Em outros, eles são mantidos como *backups* de longo-prazo. A segunda etapa são os processos de limpeza e consolidação de dados, também chamados de transformação. O processo de limpeza é custoso e envolve muitos passos para garantir a qualidade dos dados. Durante este processo é feito a validação de valores, a remoção de dados duplicados e até a verificação de regras de negócio complexas. O processo de limpeza pode exigir a intervenção humana para julgar os dados. Ao final da

limpeza os dados são salvos semi-permanentemente, visto que o processo é difícil e irreversível.

A consolidação de dados é mais simples do que o processo de limpeza. Durante a consolidação, os dados de diferentes fontes são unidos no *Data Warehouses*. As operações mais comuns na fase de consolidação são a mudança de rótulos dos dados para que eles possam ser unidos numa base comum, além de operações matemáticas com o objetivo de racionalizar diferentes proporções.

A etapa de entrega consiste em preparar os dados para a entrega. Durante esta etapa, os dados são fisicamente estruturados em um conjunto de esquemas, normalmente seguindo o modelo dimensional.

A etapa de gerenciamento é composta por serviços auxiliares como gerenciamentos de *jobs*, planos de *backup*, verificações de segurança, etc.

2.6 Análise OLAP

Nesta Seção serão mostrados os principais conceitos por trás das aplicações OLAP. A Subseção 2.6.1 discorre sobre a definição e a utilidade das aplicações OLAP. A Subseção 2.6.2 mostra as principais diferenças entre sistemas OLTP e sistemas OLAP. A Subseção 2.6.3 descreve o cubo OLAP e suas principais operações. A Subseção 2.6.4 mostra as diferentes arquiteturas de aplicações OLAP.

2.6.1 Definição de OLAP

Segundo Sezões, [Sezões, et al. 2006], OLAP (do inglês: *online analytical processing*) é um conceito que se refere a aplicações capazes de efetuar de forma rápida e partilhada, a análise de informação multidimensional, originária de diversas fontes de dados.

Aplicações OLAP são complementares a *Data Warehouses*. Enquanto os *Data Warehouses* armazenam e gerenciam os dados, as aplicações OLAP transformam esses dados em informação estratégica permitindo a analistas, gestores e executivos o acesso rápido, consistente e interativo de uma grande variedade de visões [OLAP Council, 1997].

Em adição, a utilidade de aplicações OLAP são justificadas visto que se a principal fonte de dados presentes numa corporação são relatórios gerados pelo sistema, toda vez que uma análise necessite ser feita novos relatórios terão que ser produzidos. Estes relatórios precisam de tempo para serem produzidos pela equipe de TI. Além disso, eles apresentam os seguintes pontos negativos: são estáticos e o acumulo de diferentes relatórios geram problemas de manutenção e credibilidade [Teixeira et al., 2007].

2.6.2 Diferenças entre OLTP e OLAP

Sistemas de banco de dados relacionais, também conhecido como sistemas transacionais ou ainda sistemas OLTP (do inglês: *online transactional processing*) possuem o desempenho e os requisitos funcionais diferentes dos sistemas OLAP.

Primeiramente, os sistemas OLTP são orientados para a comunidade de usuários operacionais e utilizados para transações e processamento de *querys*. Enquanto os sistemas OLAP são orientados para a comunidade de usuário gerenciais e utilizados para análise de dados [Reddy et al., 2010].

Os sistemas OLTP gerenciam os dados atuais com alto nível de detalhamento. Os sistemas OLAP gerenciam uma grande quantidade de dados históricos e oferecem facilidades para agregação e sumarização. Além disso, as informações são armazenadas em diferentes níveis de granularidade, tornando mais fácil a utilização destas informações para o apoio à tomada de decisões [Reddy et al., 2010].

Sistemas OLTP adotam uma modelagem entidade/relacionamento, enquanto os sistemas OLAP adotam a utilização do esquema estrela ou do esquema floco de neve e uma base de dados orientada a assuntos [Reddy et al., 2010].

Em relação a visão, sistemas OLTP focam principalmente na exibição dos dados atuais, sem se preocupar com os dados históricos. Os sistemas OLAP são capazes de expandir diversas versões de um esquema, mostrando o processo evolutivo de uma organização [Reddy et al., 2010].

Pro fim, os sistemas OLTP são caracterizados principalmente por processar um grande volume de transações atômicas, tendo que implementar mecanismo de concorrência, controle e recuperação de dados. Enquanto, a maioria das operações dos sistemas OLAP são *read-only* [Reddy et al., 2010].

2.6.3 Cubo OLAP

No OLAP, as informações são abstraídas em cubos multidimensionais, criados a partir de um esquema estrela, os cubos contém valores quantitativos e medidas, permitindo a visualização de diversos ângulos. Essas medidas são organizadas em categorias descritivas denominadas dimensões que formam assim a estrutura do cubo.

As aplicações OLAP devem efetuar diversas operações sobre o cubo, as mais comuns são [Teixeira et al., 2007]:

- *Drill Across*: Ocorre quando um usuário pula um nível intermediário dentro de uma mesma dimensão. Por exemplo: Em uma dimensão de tempo composta por ano, semestre, trimestre e mês. A operação *Drill Across* ocorre quando o usuário passa de ano direto para semestre ou mês .
- *Drill Down*: Ocorre quando o usuário aumenta o nível de detalhamento da informação, diminuindo a granularidade.
- *Drill Up*: Contrário do *Drill Down*, o usuário aumenta a granularidade.
- *Drill Throught*: Ocorre quando o usuário passa de uma informação contida numa dimensão para outra. Por exemplo, inicia na dimensão tempo e no próximo passo, analisa a informação por região.
- *Slice and Dice:* A operação de *slice* seleciona uma determinada dimensão no cubo, enquanto a operação de *dice* define um subcubo selecionando uma ou mais dimensões.
- *Pivot*: Realiza a rotação dos dados, afim de exibir uma apresentação alternativa.

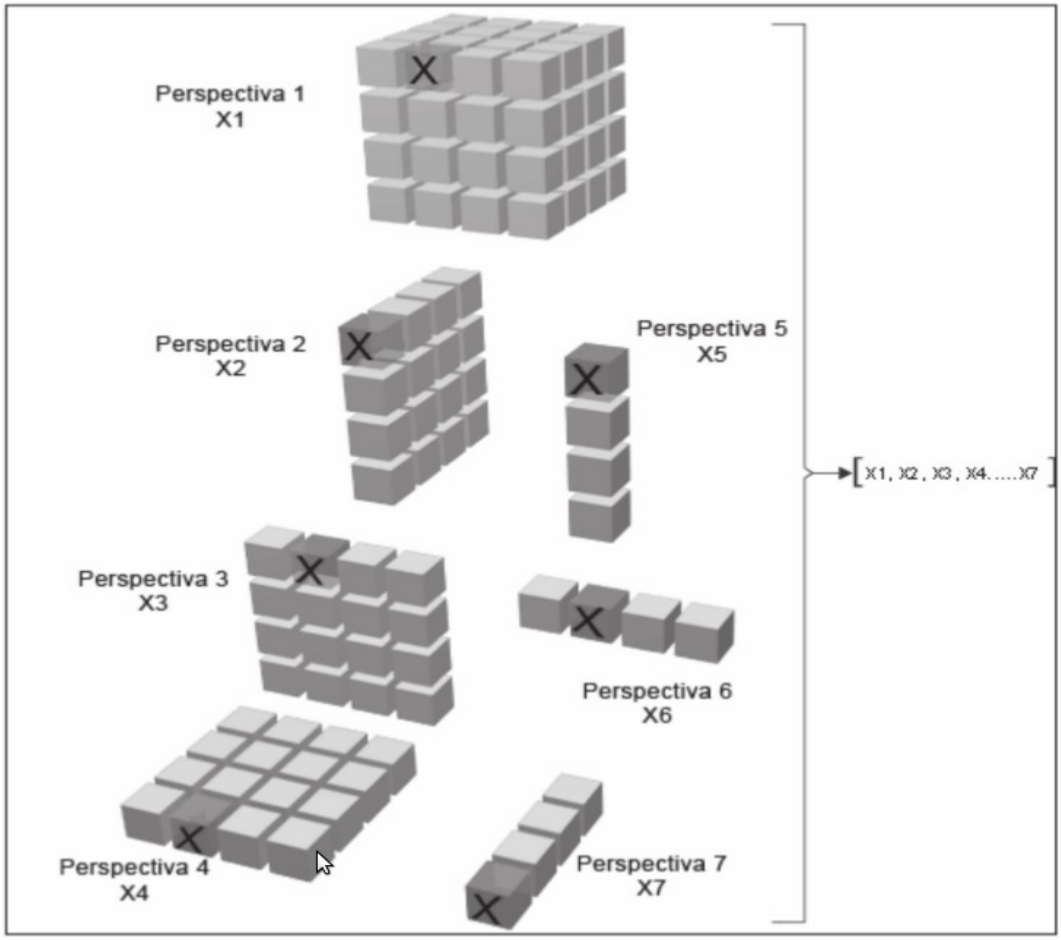


Figura 6: Cubo OLAP Fonte: (Sezões, et al. 2006)

A Figura 6 ilustra algumas operações que podem ser realizadas num cubo OLAP. (i) A perspectiva X1 exibe o cubo completo; (ii) a perspectiva X2 exibe uma fatia do cubo após uma operação de *slice*; (iii) a perspectiva X3 exibe uma nova fatia do cubo após uma operação de *slice* e uma operação de *drill down,* para aumentar o nível de granularidade; (iv) a perspectiva X4 mostra uma visão alternativa dos dados devido a uma operação de *pivot*; (v) a perspectiva X5 os dados após novas operações de *slice* e *pivot*, onde o cubo foi cortado e rotacionado; (vi) e (vii) mostram operações de *drill up* e *drill down* em diferentes dimensões.

2.6.4 Arquitetura OLAP

As aplicações OLAP se distinguem entre si através de sua arquitetura. Os principais tipos de arquitetura OLAPs são: ROLAP (do inglês: *Relational Online Analytical Processing*), MOLAP (do inglês: *Multidimensional Online Analytical Processing*) e o HOLAP (do inglês: *Hybrid Online Analytical Processing*).

O ROLAP é caracterizado por manter os dados nas tabelas relacionais originais, enquanto gera outras tabelas para armazenar valores agregados. Estes dados agregados são somas com um baixo nível de detalhes derivadas dos dados. Esta funcionalidade de agregação permite aumentar significativamente o desempenho, no entanto, o ROLAP ainda é a solução mais lenta comparada com as outras opções [Sezões, et al. 2006].

O MOLAP armazena as informações numa estrutura de dados multidimensionais. Ele mapeia as visões multidimensionais em cubos de dados formados por uma estrutura de *arrays*. A vantagem de utilizar esta tecnologia é que ela permite rápida indexação para dados sumarizados pré-computados. O MOLAP é extremamente rápido, no entanto, apresenta desvantagens em relação a escalabilidade e o tempo dedicado a sua criação [Han, 2000].

Por fim, o HOLAP é o meio-termo entra as duas tecnologias anteriores, deixando os dados originais numa tabela relacional e armazenando as agregações numa estrutura multidimensional [Sezões, et al. 2006].

2.7 Considerações Finais

Neste Capítulo foram apresentados os principais conceitos relacionados as áreas de Teste de Software e Testes Exploratórios. Em adição, foi dada atenção especial a área de *Business Intelligence,* onde foi apresentada a teoria por trás do processo de criação de uma aplicação BI. Os conceitos apresentados neste Capítulo serão de suma importância para o bom entendimento deste documento a partir do próximo Capítulo.

3 Ferramentas de Business Intelligence

Este Capítulo irá apresentar as ferramentas utilizadas no desenvolvimento desse trabalho. A Seção 3.1 apresenta o SQL Power Architect [http://www.sqlpower.ca/page/architect], uma ferramenta para modelagem de dados criada para apoiar projetistas de *Data Warehouses*. A Seção 3.2 mostra de modo geral a plataforma de *Business Intelligence* chamada Pentaho também utilizada nesse trabalho. A Subseção 3.2.1 apresenta a ferramenta Schema Workbench para construção de cubos OLAP. A Subseção 3.2.2 apresenta a ferramenta Pentaho Data Integration para processos ETL. A Subseção 3.2,2 discorre sobre o sistema OLAP Mondrian, também conhecido como Pentaho *Analyzer*. Por fim, a Seção 3.3 faz as considerações finais sobre o Capítulo.

3.1 SQL *Power Architect*

SQL *Power Architect* é uma ferramenta de código aberto para modelação de dados criada para desenvolvedores de Data Warehouses. Ela permite que os usuários façam *data-profiling*, engenharia reversa no banco de dados, além de gerar automaticamente metadados para processos ETL [SQL Power n.d.].

As principais funcionalidades oferecidas pelo SQL Power Architect utilizadas neste trabalho foram: o acesso a banco de dados via JDBC, a comparação de modelos e estruturas de dados para identificar discrepâncias, a capacidade *drag* and *drop* para criação de tabelas e colunas e a funcionalidade de modelagem de esquemas OLAP, como cubos, métricas, dimensões, níveis e hierarquias.

3.2 Plataforma Pentaho

Pentaho é uma suíte *Open Source* desenvolvida em Java que fornece soluções de *Business Intelligence* através de componentes capazes de realizar o acesso, integração, visualização, exploração e análise preditiva de dados [Pentaho Corporation n.d.].

As principais funcionalidades do Pentaho são a integração de dados, a geração de relatórios e de *dashboards* e a análise de dados através de exploração de cubos OLAP e de técnicas de mineração.

Os componentes por trás dessas funcionalidades são:

- Pentaho Data Integration (Kettle): É uma ferramenta flexível e robusta para integração de dados. O componente é responsável pelo suporte de técnicas de ETL possuindo uma biblioteca de transformação com mais de 100 objetos de mapeamento, além de também oferecer suporte de armazenamento de dados para dimensões [Vieira, 2013]. O PDI também oferece um oferece um ambiente gráfico *drag-and-drop* para minimizar a criação de código para o preparo dos dados [Pentaho Corporation n.d.].
- Pentaho *Analyzer* (Mondrian): É um motor OLAP baseado na arquitetura HOLAP capaz de lê dados de instruções SQL ou outras fontes e agregá-los em memória cache aumentando a velocidade de consultas complexas [Sidemar, 2010]. A ferramenta permite ao usuário construir cubos e navegar pela informação através de uma interface *drag-and-drop*, realizando operações de *drill up, drill down, drill across, drill throught e slice and dice*. Além disso o componente também permite a geração de relatórios e gráficos [Pentaho Corporation n.d.].
- Pentaho *Reporting*: Ferramenta principal para geração de relatórios. Dá suporte aos formatos HTML, PDF, CSV e excel. Possui interface *drag-and-drop* para construção do relatório,

armazena dados em cache para agilizar as consulta e oferece opções de *templates* e de conFiguração como tamanho do relatório, renomeação das colunas, fonte da letra, etc [Pentaho Corporation n.d.].

- Pentaho *Dashboard*: Ferramente que permite a criação de painéis de controle capaz de reunir na mesma tela tabelas de dados (através da integração com o Modrian), relatórios (através da integração com o Pentaho *Reporting*), gráficos, arquivos e páginas Web, oferecendo a opção de parametrizar estas informações. Também oferece interface *drag-and-drop* para criação dos *dashboards* [Pentaho Corporation n.d.].
- Pentaho *Data Mining* (Weka): Componente mais antigo da suíte, oferece um ambiente gráfico para *Data Mining* e consiste num conjunto de algoritmos de aprendizado de máquina que buscam padrões nos dados com o objetivo de predizer o futuro e apoiar a tomada de decisão [Pentaho Corporation n.d.].

A Figura 7 ilustra os principais módulos da plataforma Pentaho.

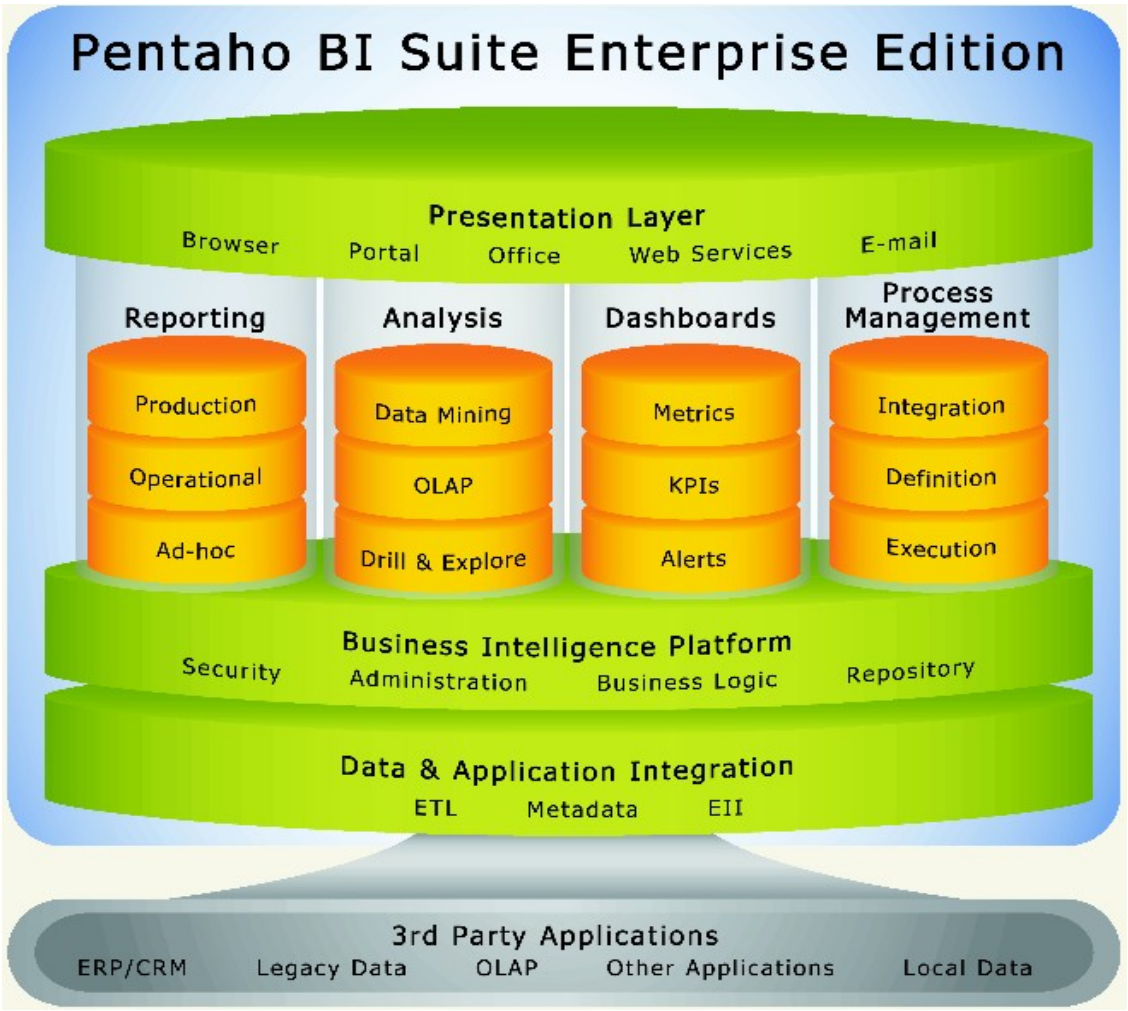


Figura 7*:* Diagrama da Plataforma Pentaho Fonte: (www.ambientelivre.com.br – Tutoriais)

3.2.1 Pentaho *Schema Workbench*

O *Schema Workbench* é a principal ferramenta para desenhar, editar e publicar esquemas OLAP no Pentaho. Tais esquemas são modelos de metadados estruturados no formato XML que são utilizados pelo *engine* do Modrian. Esses modelos podem ser considerados cubos OLAP e são mapeados em tabelas fato e tabelas dimensão encontradas no banco de dados. A criação de um modelo

de metadados dispensa a construção e a manutenção de um cubo físico [Pentaho Corporation n.d.].

A ferramenta provê uma interface gráfica que permite desenhar cubos e dimensões facilmente e associá-los as suas respectivas tabelas do banco de dados. Os cubos criados são descritos num arquivo XML e armazenado em disco, para posteriormente serem exportados para o Pentaho. [Ferreira, et al., n.d.].

3.2.2 Pentaho *Data Integration*

O Pentaho *Data Integration* (PDI), também conhecido como Kettle, é um ambiente gráfico com suporte *drag and drop* para operações de ETL que utiliza uma abordagem dirigida a metadados. O PDI é uma ferramenta bastante flexível pode ser utilizada na: população de *Data Warehouses*, migração de dados entre diferentes bases e aplicações, extração de grande quantidade de dados, limpeza e integralização de dados, rápida prototipação de esquemas ROLAP, etc [Pentaho Corporation n.d.].

O PDI foi construído em linguagem java e consiste em 4 aplicações distintas [Bouman, 2006]:

- *Spoon*: O *spoon* é uma ferramenta gráfica orientada ao usuário final. O spoon é responsável por modelar o fluxo de dados, partindo do *input*, passando pelas transformações, até o *output*.
- *Pan*: Pan é uma ferramenta de linha de comando que executa as transformações modeladas pelo *spoon*.
- *Chef*: *Chef* é uma ferramenta gráfica orientada ao usuário final utilizada para modelar *jobs*. *Jobs* consistem num conjunto de transformações ou *steps* de *jobs*.

- *Kitchen*: *Kitchen* é uma ferramenta de linha de comando utilizada para executar os *jobs* criados pelo *chef*.

Para a boa utilização da ferramenta é necessário o conhecimento de dois conceitos referentes ao fluxo de dados [Ferreira, et al., n.d.]:

- *Step*: O step é a unidade mínima do processo de transformação. Ele representa uma ação realizada por um dado, como um *input*, um *output* ou uma transformação.
- *Hop*: *Hop* é a representação gráfica da concatenação entre dois *steps*, com a origem e o destino. Um *hop* só possui uma origem e um destino, porém mais de *hop* podem sair ou chegar no mesmo step.

3.2.3 Mondrian

O Mondrian, também conhecido como Pentaho *Analyzer*, é uma ferramenta de análise interativa que permite usuário de negócio não técnicos criarem relatórios ou gráficos atrativos e interativos de maneira fácil e rápida. O Mondrian é um servidor OLAP de código aberto escrito em Java, capaz de executar querys construídas em linguagem MDX. Ele lê dados de um banco de dados relacional, agrega o resultado em memória cache e apresenta ao usuário de forma multidimensional [Pentaho Corporation n.d.].

O sistema OLAP do Mondrian é formado por 4 camadas. A primeira camada é a camada de apresentação que determina o que o usuário pode ver ou interagir. A segunda camada é a camada dimensional, ela analisa, valida e executa *querys* MDX. Por eficiência, a camada dimensional envia requisições para as células da memória, onde os dados estão agregados. A terceira camada, é a camada

estrela responsável por gerenciar a agregação do cache. Uma agregação é um conjunto de valores mensuráveis (células) armazenados em memória. A camada estrela envia requisições para as células. Se as células não estiverem em cache ou não poderem ser derivadas de outras agregações, o gerenciador de agregações manda uma requisição para a camada de armazenamento. A camada de armazenamento é um SGDB relacional responsável por trazer dados para as agregações [Pentaho Corporation n.d.].

A estratégia do Mondrian consiste em armazenar dados fatos no banco de dados relacionais e armazenar agregações de dados em memória cache para serem consultados por grupos de *querys*. Desta forma o Mondrian pode ser definido como tendo uma arquitetura HOLAP. O armazenamento multidimensional dos dados pode reduzir as entradas e saídas e tornar as operações mais rápidas em algumas circunstâncias, no entanto, a complexidade do estágio de modelagem dimensional pode não compensar esses benefícios [Pentaho Corporation n.d.].

3.3 Considerações Finais

Neste Capítulo foram apresentados as principais ferramentas utilizada na implementação da parte prática deste trabalho. As ferramentas apresentadas neste Capítulo serão importantes para o bom entendimento dos próximos Capítulos, onde serão apresentadas o desenvolvimento da ferramenta de visualização de falhas integrada ao Pentaho.

4 Sistemas Institucionais da UFRN

Neste Capítulo serão apresentados os sistemas institucionais da UFRN. A Seção 4.1 apresenta a visão geral dos sistemas. A Seção 4.2 apresenta a arquitetura dos sistemas, onde estará inserida a ferramenta Web de visualização de falhas. Por fim, a Seção 4.3 apresenta as considerações finais do Capítulo.

4.1 Visão Geral

Os sistemas institucionais da UFRN (também conhecidos por Sistemas SIG) foram desenvolvidos com intenção de serem utilizados no auxílio da execução das atividades acadêmicas e administrativas [SINFO/UFRN, 2012a].

Inicialmente a instituição utilizava vários sistemas acadêmicos e administrativos. Mas pelo fato destes sistemas não serem integrados, surgiam diversas dificuldades, entre elas [SINFO/UFRN, 2012a]:

• O usuário necessitava criar mais de um login e senha para se autenticar nos diversos sistemas;

• A manutenção e desenvolvimento de alguns desses sistemas era feita através da contratação de software de terceiros;

• Equipes distintas de desenvolvedores de software envolvidas na informatização da instituição;

• A idéia de acrescentar novas funcionalidades a alguns sistemas dependia de informações alimentadas em outros sistemas e os mesmos não eram integrados.

Devido a necessidade de integração surgiu a ideia de criar os sistemas institucionais onde todas as atividades acadêmicas e administrativas fossem centralizadas.

Os sistemas institucionais é composto por vários sistemas, mas existem três principais [SINFO/UFRN, 2012a]:

• SIGAA (Sistema Integrado de Gestão de Atividades Acadêmicas): utilizado para a execução das atividades da área fim da UFRN, a área acadêmica. Com operações relacionadas ao ensino infantil, médio, técnico, graduação, pós-graduação, pesquisa, extensão, monitoria, etc.

• SIPAC (Sistema Integrado de Patrimônio, Administração e Contratos): utilizado para a execução das atividades administrativas, como, solicitações de material, diárias, passagens, material informacional, suprimento de fundos, controle e execução orçamentária, controle de almoxarifado e patrimônio, controle das compras e licitações, tramitação de processos e documentos protocolados, execução de obras, contratos, convênios, liquidação de despesas, etc.

• SIGRH (Sistema Integrado de Gestão e Recursos Humanos): utilizado para auxiliar a gestão e gerenciar as informações dos recursos humanos. Controle de dados pessoais e funcionais dos servidores, das férias e frequências, realização de avaliações funcionais, controle de informações sobre capacitação, concursos, aposentadoria, etc.

Cada sistema é composto por Subsistemas (também chamados de módulos). O SIGAA, por exemplo, possui os subsistemas de Graduação, Stricto-sensu, Extensão, Pesquisa etc.

Além destes, são desenvolvidos outros sistemas, como: SIGAdmin (para administração dos demais sistemas), SIGED (para armazenamento de documentos), SIGPP (para gestão de

planejamento e projetos), iProject (para gerência de desenvolvimento dos sistemas), dentre outros. Abaixo segue a Figura 8 do SIGAA e seus módulos. (Os módulos em cinza estão desabilitados devido a restrições de permissão).

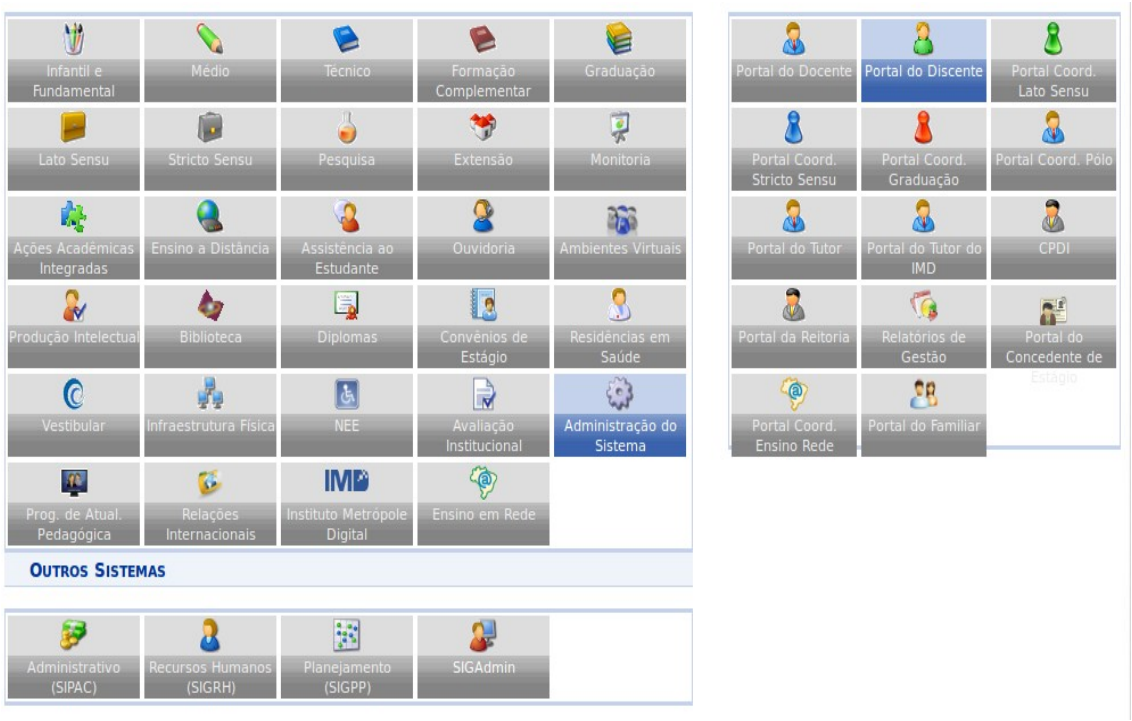


Figura 8: Módulos do Sistema Sigaa

4.2 Arquitetura

Os sistemas SIG foram desenvolvidos utilizando a tecnologia Java Enterprise Edition (JavaEE). Outras tecnologias utilizadas foram: Hibernate 3.2, Java-Server Faces, RichFaces, Struts, EJB, Spring, Javascript, JQuery e o servidor de aplicação JBoss. O Eclipse foi adotado como IDE para desenvolvimento.

A nível de código, a arquitetura é composta por cinco projetos que fornecem toda infraestrutura para implementação dos demais sistemas (SIGAA, SIPAC, SIGRH, etc).

Estes projetos da arquitetura estão dispostos da seguinte forma [Dias, 2012]:

• Arquitetura - *Framework*: Contém as classes do núcleo do *framework* utilizado no desenvolvimento dos projetos, como as classes envolvidas no desenvolvimento das camadas (*controllers*, DAOs genéricos, etc.) e algumas classes de domínio mais importantes e estruturantes, como UsuarioGeral, UnidadeGeral e PessoaGeral. Possui ainda as classes responsáveis pela auditoria, classes envolvidas com a segurança dos sistemas e classes utilitárias e auxiliares.

• EntidadesComuns: Esse projeto contém as classes de domínio que são comuns a todos os sistemas, como Unidade, Responsavel, PerfilPessoa, etc. Além disso, contém outras classes que são utilizadas em todos os sistemas mas que não fazem parte do framework, como alguns DAOs, servlets para consultas, classes para implementação de componentes do tipo Autocomplete, etc.

• ServicosIntegrados: No projeto ServicosIntegrados ficam as interfaces dos serviços remotos dos sistemas, bem como os *Data Transfer Objects* (DTOs) utilizados para troca de informações nesses serviços. Possui ainda um contexto /servicos com a definição de diversos serviços disponibilizados pela arquitetura, como serviço de identificação biométrica, serviço de conversão de documentos para PDF, serviço de OCR (do inglês: *Optical Character Recognition*) e alguns outros serviços utilizados por aplicações *desktop*, como serviços para autenticação de usuários e para a criação de *tokens* de autenticação.

• *SharedResources*: Não possui classes, contém recursos estáticos que precisam ser disponibilizados para uso dos demais projetos. Entre esses recursos, podemos citar imagens, arquivos de

estilo (CSS) e arquivos Javascript. Define um contexto chamado /shared, através do qual esses recursos podem ser acessados.

• LIBS: O objetivo deste projeto é agrupar todas as bibliotecas utilizadas pelos sistemas. Todos os arquivos JAR utilizados pelos sistemas ficam nesse projeto, que não contém nenhuma classe.

A organização dos projetos podem ser visualizados na Figura 9 abaixo:

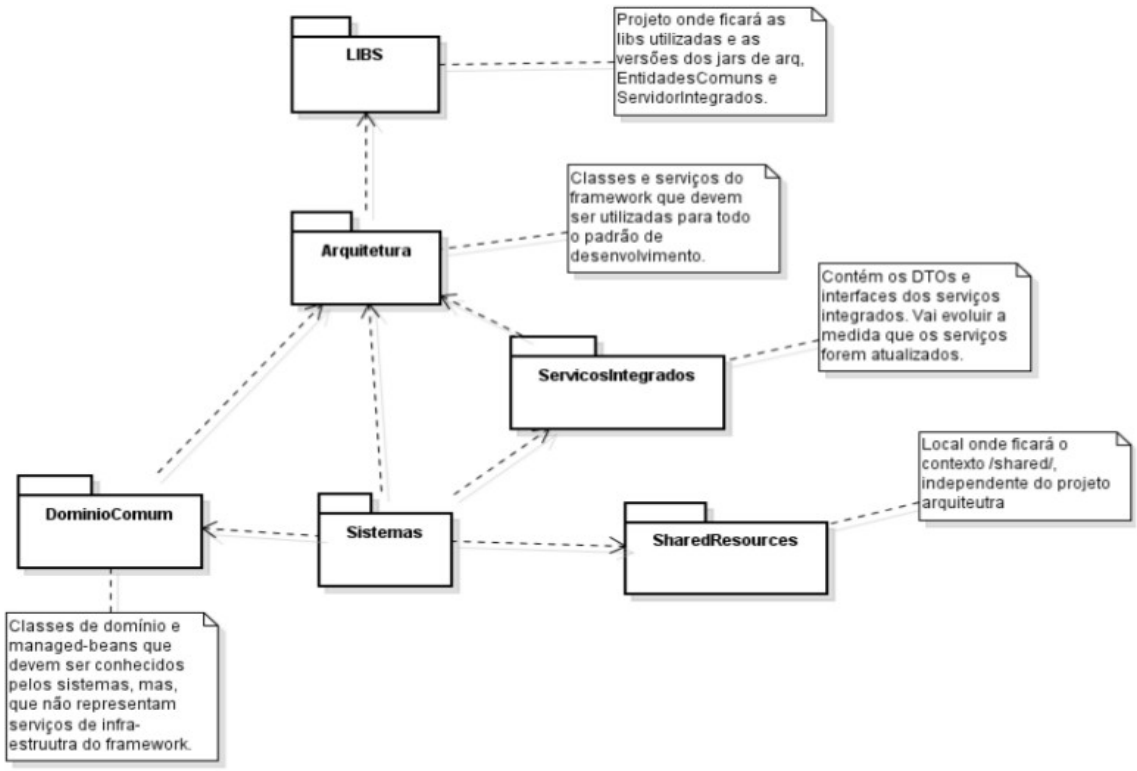


Figura 9: Pacotes da Arquitetura Fonte: (Dias, 2012)

4.3 Considerações Finais

Neste Capítulo foi apresentado uma visão geral dos sistemas SIG, assim como foi mostrado o *framework* da arquitetura do projeto. Tal conhecimento é importante para o bom entendimento do Capítulo

que descreve a ferramenta de visualização de falhas integrada no iProject.

5 Visualizador de Falhas SINFO/UFRN – Versão Integrada no iProject

Este Capítulo irá apresentar a ferramenta Web para visualização de falhas detectadas durante a fase de testes e reportadas pela equipe de Suporte da SINFO/UFRN. A Seção 5.1 apresenta o contexto em que a ferramenta está inserida. A Seção 5.2 descreve os requisitos funcionais da ferramenta. A Seção 5.3 apresenta a arquitetura da ferramenta. A Seção 5.4 discorre sobre os detalhes de cada componente da ferramenta. A Seção 5.5 mostra uma visão geral da ferramenta. Por fim, a Seção 5.6 apresenta as considerações finais do Capítulo.

5.1 Contexto

Neste trabalho foi desenvolvido um sistema para visualizar falhas reportadas pelas equipes de Controle de Qualidade e Suporte da SINFO/UFRN. Este sistema foi adicionado ao iProject, o sistema de gestão de projetos da SINFO (superintendência de informática da UFRN), como mostrado ao longo do Capítulo. O principal objetivo do iProject é a gerência do processo de trabalho através do fluxo de tarefas. As tarefas são utilizadas para organizar o trabalho das equipes de requisitos, suporte ao usuário, desenvolvimento e controle de qualidade interagem entre si através de logs, que são registrados nas tarefas. Existem dois principais tipos de fluxos: O fluxo quando uma tarefa reflete um novo caso de uso ou um aprimoramento e o fluxo quando a tarefa reflete uma manutenção ou correção de falha no sistema em produção. A ferramenta desenvolvida nesse trabalho atua nesses dois fluxos que serão detalhados a seguir:

Quando o fluxo representa um novo caso de uso ou um aprimoramento, a equipe de requisitos faz o levantamento dos requisitos (Passo 1), modela o caso de uso (Passo 2) e repassa para os gestores homologarem (Passo 3). Os gestores enviam a tarefa para a equipe de desenvolvimento implementar a nova funcionalidade (Passo 4), ele repassa a tarefa para equipe de controle de qualidade realizar os testes (Passo 5). Sempre que um testador encontra falhas, ele registra um *log* identificando o tipo de falha com uma *hashtag* específica e retorna a tarefa para o desenvolvedor (Passo 6). Após a tarefa ter sido validada pelo controle de qualidade (Passo 7), o desenvolvedor solicita validação técnica (Passo 8). Após a tarefa ter sido validada pelos gestores, o desenvolvedor solicita atualização (Passo 9 e 10), para depois validar novamente a tarefa no ambiente de homologação (Passo 11), posteriormente a tarefa é validada pelo integrador (Passo 12). Por fim, a tarefa é integrada no ambiente de produção (Passo 13).

A Figura 10 ilustra o fluxo de uma tarefa que representa um novo caso de uso.

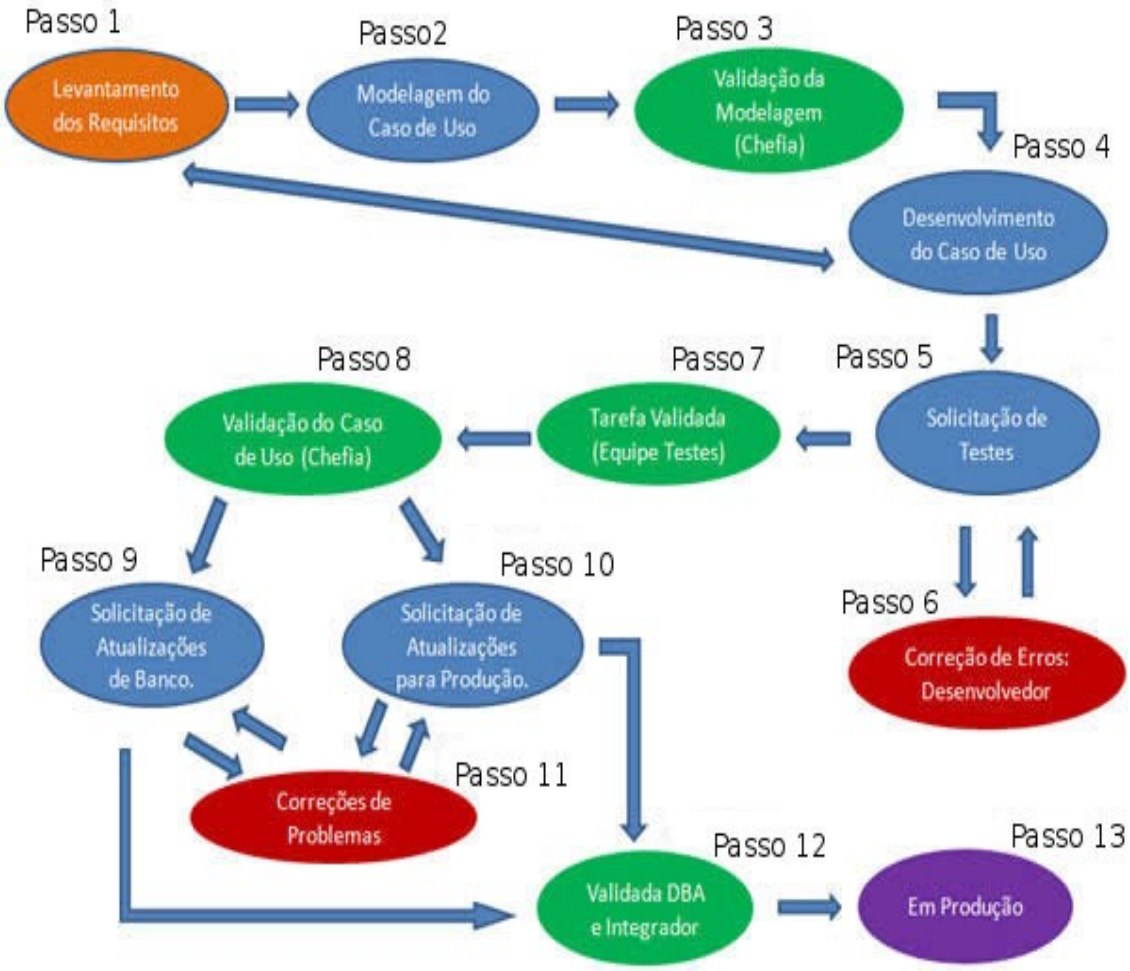


Figura 10: Fluxo de Tarefas de Aprimoramento dos Sistemas SIG Fonte: (Wiki dos Sistemas SIG)

Quando a tarefa corresponde a uma manutenção ou correção de falha no sistema em produção o fluxo é basicamente o mesmo, com a única diferença que nestes casos a tarefa é criada pela a equipe de suporte ao usuário.

A Figura 11 ilustra o fluxo de uma tarefa que corresponde a manutenção ou correção de uma falha no sistema.

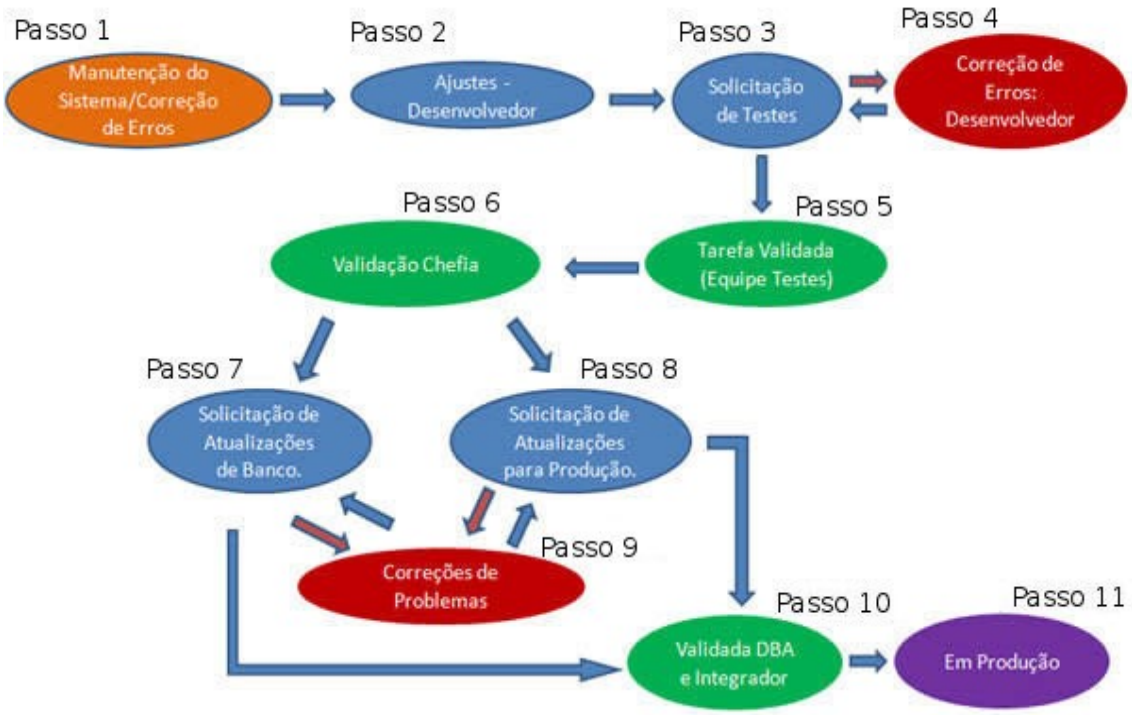


Figura 11: Fluxo de Tarefas de Erros dos Sistemas SIG Fonte: (Wiki dos Sistemas SIG)

Tanto os analistas de suporte, tanto os testadores, quando criam uma tarefa ou registram um *log* para reportar uma ou mais falhas, ele registra na descrição da tarefa o tipo de falha através de uma *hashtag* específica. Foram definidas juntamente com os gerentes de Suporte e Controle de Qualidade 5 *hashtags* para identificar os tipos de falhas:

- *#BD*: Indica que a falha foi ocasionada por problemas no banco de dados, por exemplo, uma tabela não populada no ambiente de testes.
- *#CI*: Indica que a falha foi um comportamento inesperado, um defeito inserido no código, como um *Nullpointer*.

- *#Navegação*: Indica que a falha consiste no redirecionamento incorreto entre duas páginas. Um exemplo seria o botão voltar retornando para uma página indevida.
- *#Negócio*: Indica falha na implementação das regras de negócio, como exemplo, um cálculo que está retornando um valor diferente do esperado.
- *#Visual*: Indica falhas no estilo e na formatação da página, como uma tabela mal formada ou um estilo fora do padrão do sistema.

A Figura 12 ilustra um exemplo de um *log* onde são registradas *hashtags.*

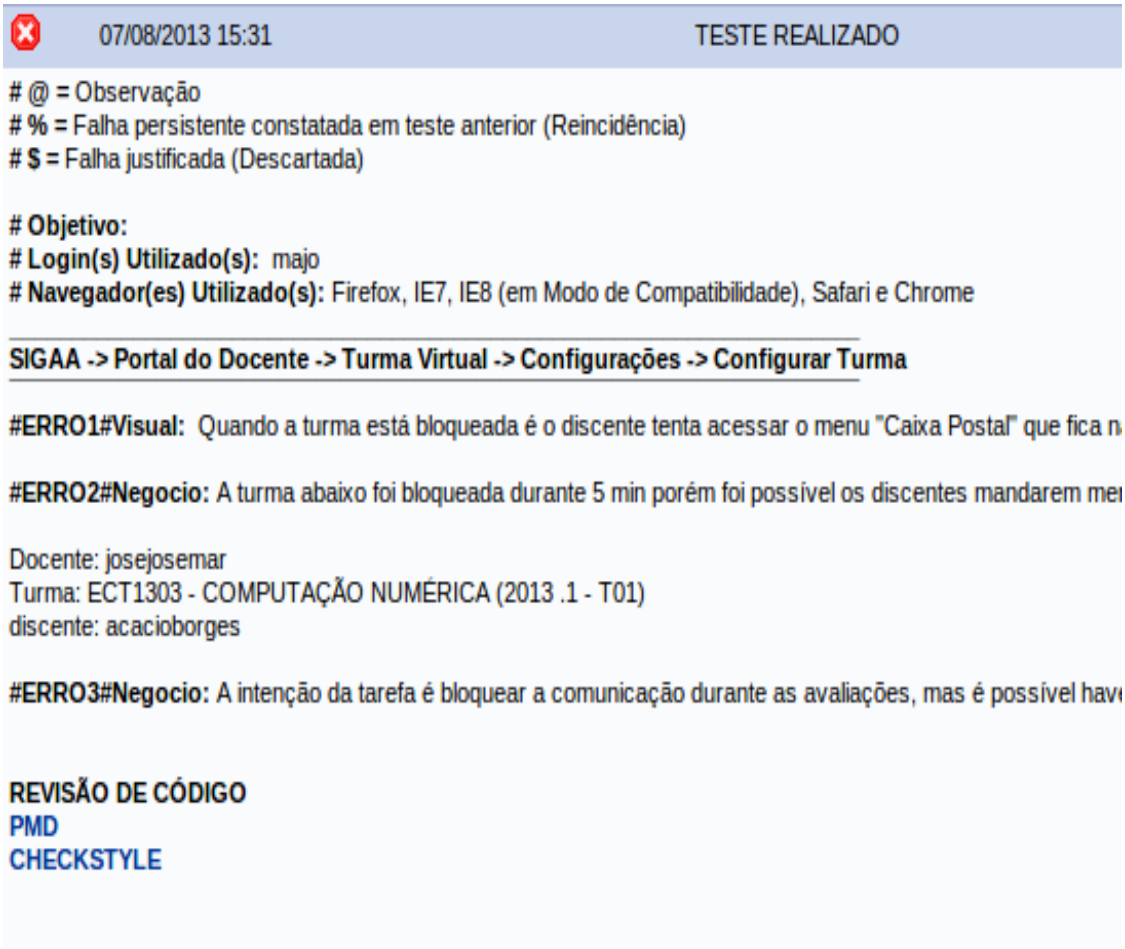


Figura 12: Exemplo de log de teste Fonte: (iproject)

O visualizador Web de falhas tem a função de varrer os *logs* de testes e descrições de tarefas criadas pelo suporte em busca das *hashtags* de erro para poder gerar um relatório de falhas do sistema, juntamente com gráficos para ajudar a visualização.

5.2 Requisitos Funcionais

A ferramenta de implementada tem como objetivo exibir diferentes visões e dados sobre as falhas que ocorrem no sistema e foram registradas pela equipe de controle de qualidade e suporte ao usuário. O fluxo do caso de uso é dividido em várias telas, onde cada uma oferece diferentes informações e níveis de detalhamento. Nesta Seção será mostrado os principais requisitos funcionais da ferramenta de visualização de interesses:

- **RF01** – **Exibir quantidade de falhas por tipo de erro e sistema:** É exibido um relatório com a quantidade de falhas e seus respectivos tipos por sistema. As informações são consultadas por período e opcionalmente por sistema, colaborador e tipo de erro.

- **RF02 – Exibir gráfico de barras da quantidade de falhas do sistema:** Utiliza as informações listadas na RF01 e apresenta os dados em formato de um gráfico de barras.

- **RF03 – Exibir gráfico de pizza da quantidade de falhas do sistema:** Utiliza as informações listadas na RF01 e apresenta os dados em formato de um gráfico de pizza.

- **RF04 – Exibir listagem de tarefas com falha no sistema:** Ao clicar na quantidade de falhas de algum sistema listado na RF01, será exibida a listagem com todas as tarefas que possuem falhas do sistema.

• **RF05 – Exibir detalhes das tarefas:** Ao clicar no número ou no ícone “Detalhes da Tarefa” exibidos na listagem da RF04 será apresentada todos os detalhes das tarefas, inclusive, todos os *logs* registrados nela.

• **RF06 – Gerar CSV de fatos:** É gerado um arquivo CSV com dados a serem exportados para a ferramenta BI. É possível selecionar a geração de dados para a tabela fato, neste caso, cada linha do CSV terá as seguintes informações: O tipo de erro, o sistema, o número da tarefa, o responsável, a data e a quantidade de falhas. Ou a geração de dados para a tabela dimensional tarefa. Neste caso, cada linhas do CSV terá as seguintes informações, o número da tarefa e seu título.

5.3 Arquitetura

A ferramenta foi construída dentro do pacote iProject e utiliza a arquitetura compartilhada dos demais pacotes institucionais da UFRN. Por esta razão foi possível utilizar toda a infraestrutura compartilhada, sendo necessário criar apenas os artefatos referentes ao caso de uso.

Abaixo segue a Figura 13 que ilustra a arquitetura da ferramenta integrada ao iProject.

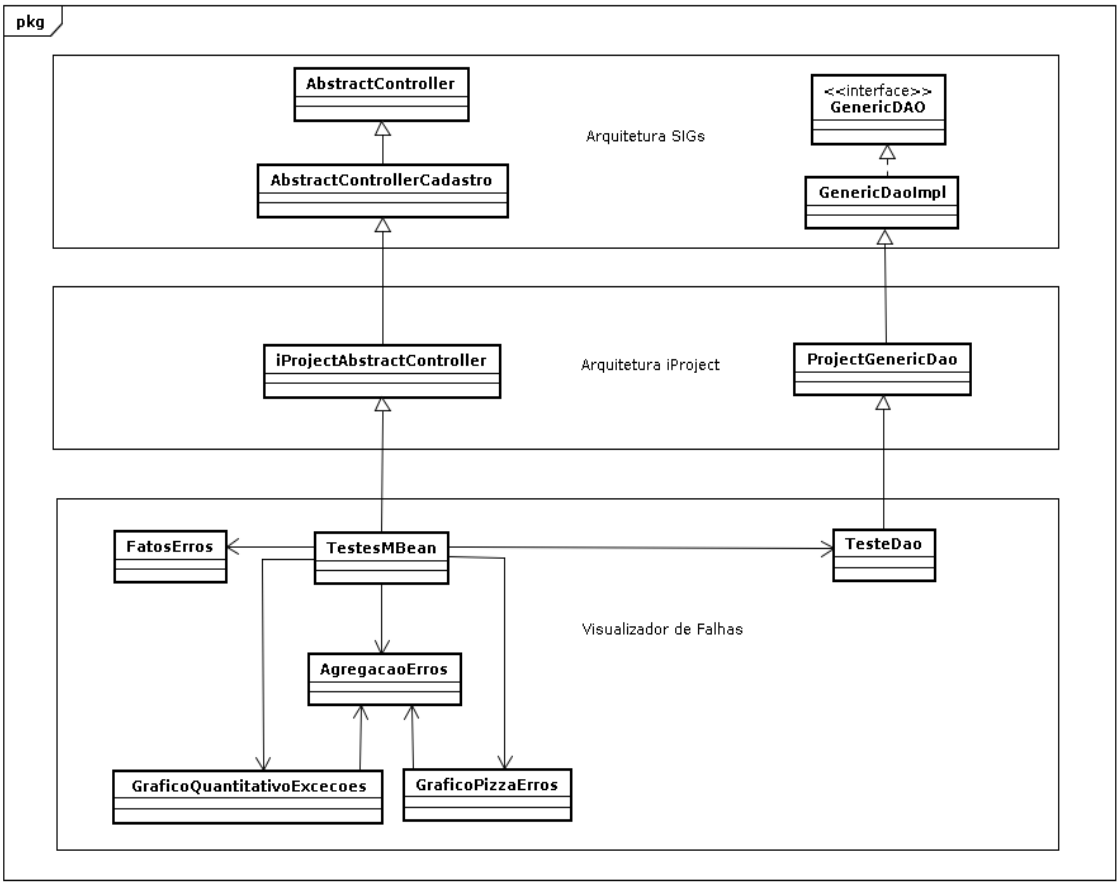


Figura 13: Arquitetura geral da ferramenta

No primeiro nível, se encontra a arquitetura dos sistemas SIG que é utilizada como base para os demais sistemas da instituição. As principais classes da arquitetura utilizada para compor o visualizador de falhas foram: o AbstractController, o controlador genérico do padrão MVC; o AbstractControllerCadastro, classe abstrata que estende o AbstractController e possui métodos e variáveis comuns para auxiliar a construção de *controllers* de cadastro; o GenericDAO é uma interface que contém as assinaturas dos métodos de persistência que devem ser implementados por todos os DAOs dos projetos da instituição; o GenericDaoImpl é a implementação dos métodos comuns do GenericDAO, como o *save, update e delete*.

No nível intermediário se encontra a arquitetura do iproject. As principais classes desta arquitetura utilizadas na construção da

ferramenta foram o iProjectAbstractController e o ProjectGenericDao que estendem respectivamente as classes da arquitetura AbstractControllerCadastro e GenericDaoImpl e fornecem um conjunto de métodos auxiliares para construção de controladores e DAOs referentes apenas ao projeto iproject.

Por fim, na última camada se encontram as classes da ferramenta desenvolvida que serão detalhadas na próxima Seção.

Para a construção da ferramenta foi utilizada a IDE eclipse com o *plugin* SVN para a conexão com o repositório que contém os códigos fontes do sistema. A ferramenta foi implementada utilizando a linguagem Java com o *framework* JavaServer Faces (JSF). Para construção dos gráficos foi utilizada a API JavaFreeChart mais a biblioteca Cewolf 1.0. O servidor que executa a aplicação é o JBoss.

5.4 Projeto Detalhado

Esta Seção possui o objetivo de apresentar com mais detalhes os componentes do visualizador de falhas de falhas, descritos no diagrama da Seção anterior.

5.4.1 TestesMBean

TestesMBean é um controlador responsável por fazer a intermediação entre a camada de visualização e a camada de modelo da aplicação. A classe possui a responsabilidade de chamar a camada de persistência (DAOs), enviar os dados para a camada lógica processar o resultado e por fim repassar os dados para serem exibidos pela camada de apresentação (JSPs).

TestesMBean estende do controlador padrão do iProject, o iProjectAbstractControllerMBean e possui relação de dependências com as classes: TesteDao, FatosErros, AgregacaoErros, GraficoQuantitativoExcecoes e GraficoPizzaErros.

Abaixo, a Figura 14 exibe os diagramas de classe de TestesMBean.

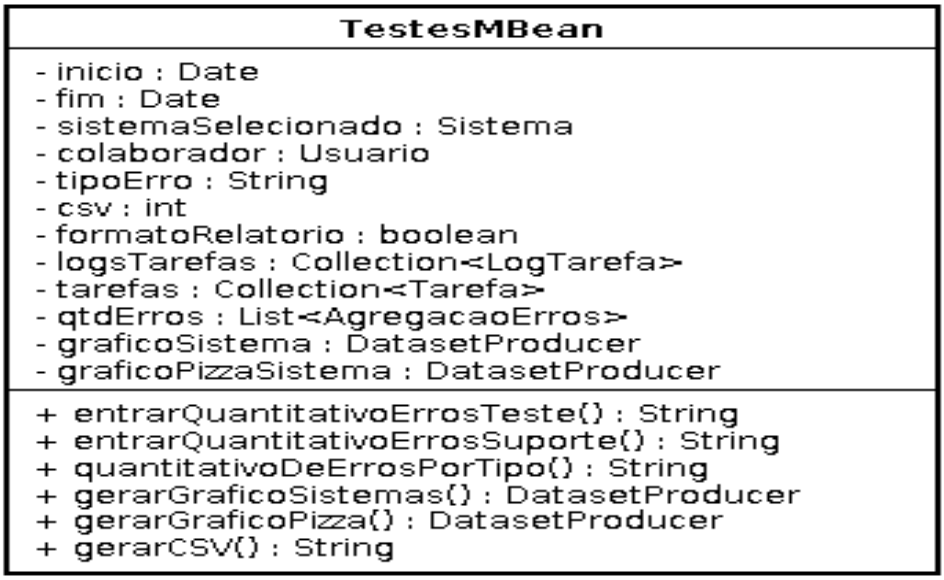


Figura 14: Classe TestesMBean

Abaixo segue a descrição dos principais atributos e métodos da classe TestesMBean. Os demais atributos e métodos que foram omitidos possuem apenas o objetivo de auxiliar tais funções.

- Atributos:
  - inicio: Data início do formulário. Esse atributo é utilizado para filtrar as informações. Apenas erros que ocorreram nesta data ou posterior a ela serão retornados no formulário.
  - Fim: Data fim do formulário. Esse atributo é utilizado para filtrar as informações. Apenas erros que ocorreram nesta data ou anterior a ela serão retornados para o formulário.

- sistemaSelecionado: Sistema selecionado para filtrar as informações. Apenas erros do sistema selecionado serão retornados no formulário, se nenhum sistema for selecionado, serão retornados os erros de todos os sistemas.
- colaborador: Usuário da equipe de controle de qualidade ou suporte que registrou o erro. Apenas erros do colaborador selecionado serão retornados no formulário, se nenhum colaborador for selecionado, serão retornados os erros de toda equipe.
- tipoErro: Tipo de erro utilizado para filtrar informações. Apenas erros do tipo selecionado serão retornados no formulário, se nenhum tipo de erro for selecionado, serão retornados todos os tipos de erros.
- csv: CSV contendo os fatos sobre as informações selecionadas para filtragem. Utilizado para auxiliar o processo de ETL da ferramenta BI.
- formatoRelatório: Indica se o formulário deve ser exibido no formato de relatório para impressão.
- logsTarefas: Armazena os *logs* das tarefas que possuem registro de erros e que estão de acordo com as opções de filtros selecionados.
- tarefas: Armazena as tarefas que possuem registros de erros e que estão de acordo com as opções de filtros selecionados.
- qtdErros: Armazena o resultado a ser exibido após o processamentos dos *logs* das tarefas e das tarefas pela camada lógica.
- graficoSistema: Armazena o gráfico do resultado em formato de barras.

  - graficoPizzaSistema: Armazena o gráfico do resultado em formato de pizza.

- Métodos:
  - entrarQuantitativoErrosTeste: Responsável por iniciar os objetos referentes ao formulário de falhas encontradas pelo controle de qualidade.
  - entrarQuantitativoErrosSuporte: Responsável por iniciar os objetos referentes ao formulário de falhas encontradas pelo suporte ao usuário.
  - quantitativoDeErrosPorTipo: Realiza a consulta dos *logs* das tarefas e das tarefas com falhas de acordo com os filtros selecionados, através da classe TesteDao. Após isso, invoca a camada de negócio para tratar os dados consultados. Nesta fase é chamado o método agregar da classe AgregacaoErros para popular o atributo qtdErros. Após isso, são invocados os métodos gerarGraficoSistemas, gerarGraficoPizza e opcionalmente o método gerarCSV, caso assim tenha sido determinado pelos filtros. Por fim, retorna os resultados obtidos para a JSP.
  - gerarGraficoSistemas: Gera o gráfico de barras a ser exibido instanciando a classe GraficoQuantitativoExcecoes.
  - gerarGraficoPizza: Gera o gráfico de pizza a ser exibido instanciando a classe GraficoPizzaErros.
  - gerarCSV: Gera o CSV de fatos utilizado no processo ETL da ferramenta BI. Durante este procedimento são chamados um dos dois métodos: gerarFatosTestes e gerarFatosSuporte da classe FatosErros.

5.4.2 TesteDao

TesteDao é uma classe da camada de persistência da aplicação. A classe herda do DAO genérico do projeto iproject, ProjectGenericDao, que por sua vez é uma extensão da classe da arquitetura GenericDaoImpl, que implementa a interface mãe dos demais DAOs dos sistemas: GenericDAO.

A classe TesteDao é utilizada por vários outros casos de uso no sistema. Todos os métodos que não possuem relação com a ferramenta de visualização de falhas foram omitidos. Abaixo a Figura 15 exibe o diagrama de classe de TesteDao.

Figura 15: Classe TesteDao

Abaixo segue a descrição dos métodos da classe TesteDao utilizados utilizados na ferramenta de visualização de falhas:

- Métodos:
  - logsTarefasComErrosDeTeste: Responsável por consultar o banco de dados Comum e montar uma lista de objetos LogTarefa. A consulta é determinada pelas opções de filtragem que são passadas como parâmetros para o método. A consulta do método irá buscar informações das seguintes tabelas: iproject.log_tarefa, iproject.tarefa, iproject.sistema, iproject.subsistema, iproject.equipe, iproject.membro_equipe, comum.usuario e comum.pessoa.

– entrarQuantitativoErrosSuporte: Responsável por consultar o banco de dados Comum e montar uma lista de objetos Tarefa. A consulta é determinada pelas opções de filtragem que são passadas como parâmetros para o método. A consulta do método irá buscar informações das seguintes tabelas: iproject.tarefa, comum.usuario, comum.pessoa. iproject.equipe. iproject.membro_equipe, iproject.sistema e iproject.subsistema,

5.4.3 AgregacaoErros

Esta classe de domínio armazena as informações sobre as falhas do sistema. O principal objetivo da classe é contabilizar a quantidade de cada tipo de erro ocorrido num determinado sistema. São os dados desta classe que serão apresentados na camada de visualização. Abaixo a Figura 16 exibe o diagrama de classe de AgregacaoErros.

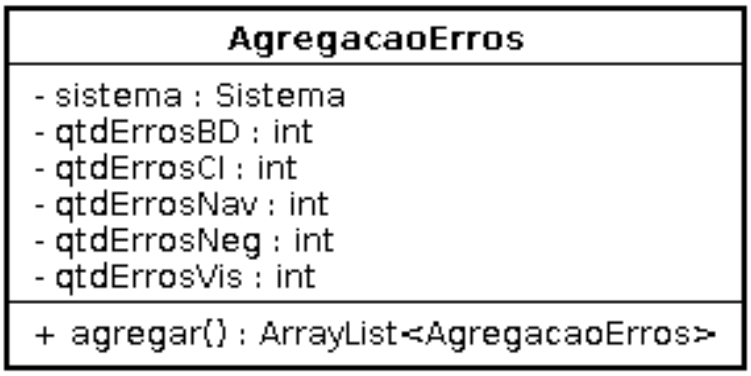


Figura 16: Classe AgregacaoErros

Abaixo segue a descrição dos principais atributos e métodos da classe:

- Atributos:

  - sistema: Sistema referente as informações sobre a quantidade de erros.
  - qtdErrosBD: Quantidade de erros de banco de dados.
  - qtdErrosCI: Quantidade de erros de comportamento inesperado.
  - qtdErrosNav: Quantidade de erros de banco de navegação.
  - qtdErrosNeg: Quantidade de erros de negócio.
  - qtdErrosVis: Quantidade de erros de visualização.
- Métodos:
  - agregar: Recebe como parâmetro uma coleção de LogTarefa, uma coleção de Tarefa e uma coleção de Sistema. O método é responsável por varrer as coleções em busca de hashtags de erros e gerar uma lista de AgregacaoErros que será utilizada na camada de apresentação.

5.4.4 FatosErros

Classe de domínio armazena as informações sobre os fatos que serão utilizados no *Data Warehouse* da ferramenta BI. A classe está inserida no contexto do processo ETL visto que ela auxilia a extração e a limpeza dos dados, garantindo que as informações buscadas sejam bem formadas.

A classe FatosErros possui como objetivo recuperar os dados para popular a tabela fato e a tabela dimensional tarefa da aplicação BI. Por esta razão, a classe é constituída de forma a responder a seguinte pergunta: “Quantos erros do tipo <TipoErro>, foram encontrados no subsistema <sistema>, no dia <data>, registrados pelo responsável <responsavel>, na tarefa <tarefa>”.

Abaixo a Figura 17 exibe o diagrama de classe de FatosErros.

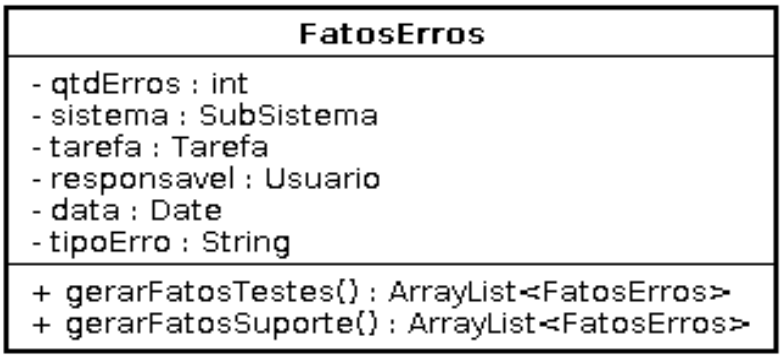


Figura 17: Classe FatosErros

Abaixo segue a descrição dos principais atributos e métodos da classe:

- Atributos:
  - qtdErros: Quantidade de erros num determinado fato.
  - sistema: Sistema no qual ocorreu os erros.
  - tarefa: Tarefa na qual ocorreu os erros.
  - responsavel: Usuário responsável por ter registrado o erro na tarefa.
  - data: Data em que o erro foi registrado.
  - tipoErro: Tipo do erro registrado.
- Métodos:

- gerarFatosTestes: Recebe como parâmetro uma coleção de LogTarefa e uma coleção de Sistema. O método é responsável por varrer as coleções em busca de *hashtags* de erros e gerar uma lista de FatosErros que será utilizada para gerar um CSV afim de auxiliar o processo de ETL para ferramente BI.
- gerarFatosSuporte: Recebe como parâmetro uma coleção de Tarefa e uma coleção de Sistema. O método possui a mesma função que o gerarFatosSuporte, com exceção que varre a coleção de tarefas.

5.4.5 GraficoQuantitativoExcecoes

Classe genérica utilizada para gerar o gráfico em barras disponível na biblioteca Cewolf. A classe implementa a interface DatasetProducer. Abaixo segue a Figura 18 que ilustra o diagrama de classe do GraficoQuantitativoExcecoes.

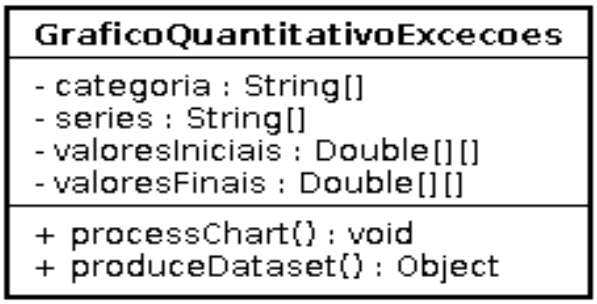


Figura 18: Classe GraficoQuantitativoExcecoes

Abaixo segue a descrição dos principais atributos e métodos da classe:

- Atributos:

  - categoria: *Array* de categorias (elementos do eixo Y) do gráfico, no caso do visualizador de erros, as categorias são os nomes dos sistemas.
  - series: *Array* de categorias (elementos do eixo X) do gráfico, no caso do visualizador de erros, as categorias são os tipos de erros.
  - valoresIniciais: Para cada categoria, tem-se o valor inicial de cada série.
  - valoresFinais: Para cada categoria, tem-se o valor final de cada série.
- Métodos:
  - processChart: Processa as conFigurações do gráfico em barras.
  - produceDataset: Produz o conjunto de dados (*Dataset*) que será mapeado no gráfico.

5.4.6 GraficoPizzaErros

Classe genérica utilizada para gerar o gráfico em pizza disponível na biblioteca Cewolf. A classe implementa a interface DatasetProducer. Abaixo, segue a Figura 19 que exibe o diagrama de classe de GraficoPizzaErros.

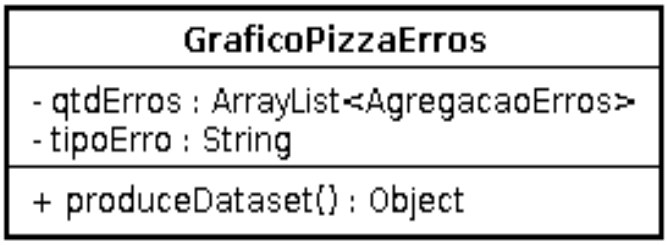


Figura 19 **Classe GraficoPizzaErros**

Abaixo segue a descrição dos principais atributos e métodos da classe:

- Atributos:
  - qtdErros: Lista de AgregacaoErros de onde serão extraídos a quantidade de cada tipo de erro por sistema.
  - tipoErro: Variável que indica se o formulário foi filtrado por tipo de erro. Neste caso, as fatias da pizza passam a ser os sistemas.
- Métodos:
  - produceDataset: Produz o conjunto de dados (*Dataset*) que será mapeado no gráfico.

5.5 Visão Geral

O usuário poderá acessar as operações através dos seguintes caminhos: *iProject → Testes → Gerência de Falhas → Relatório Quantitativo de Erros de Testes* e *iProject → Suporte → Relatórios → Quantitativo de Erros de Suporte* (o usuário precisa está logado no sistema). A tela inicial irá exibir as opções de filtro para o relatório.

Abaixo segue a Figura 20 que exibe o filtro da consulta:

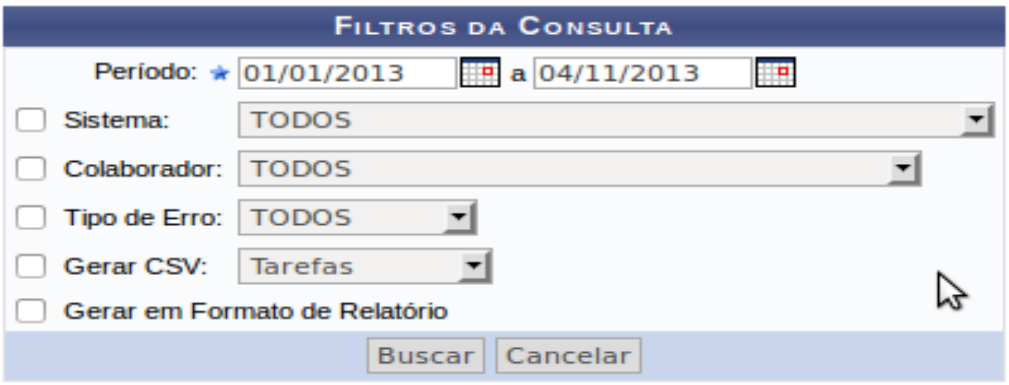
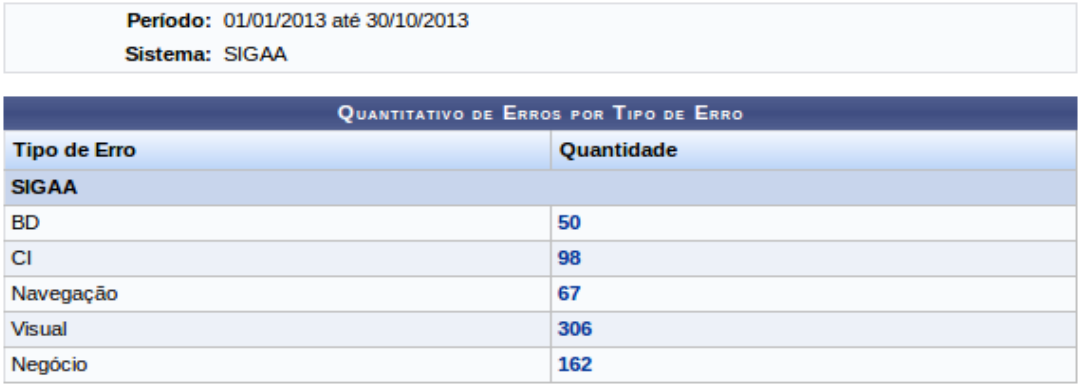


Figura 20: Filtro do Relatório Quantitativo de Erros de Testes Fonte: (iproject)

Ao consultar, serão exibidos o relatório quantitativo de erros e os gráficos referentes ao relatório. No relatório serão listados os tipos de erros ocorridos e a quantidade de ocorrência dos erros, tais informações são agrupadas por sistema. A Figura 21 ilustra o relatório.



Período: 01/01/2013 até 30/10/2013
Sistema: SIGAA

Quantitativo de Erros por Tipo de Erro

Tipo de Erro	Quantidade
SIGAA	
BD	50
CI	98
Navegação	67
Visual	306
Negócio	162

Figura 21: Relatório Quantitativo de Erros de Testes Fonte: (iproject)

Logo abaixo do relatório serão exibidos os gráficos de barra e de pizza em relação a informação consultado. A Figura 22 exibe os gráficos gerados.

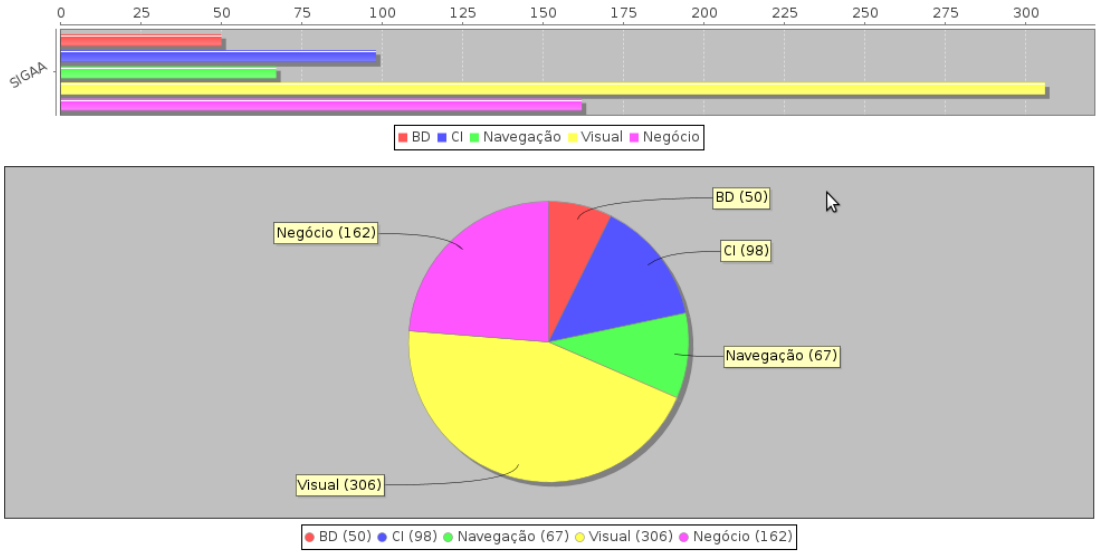


Figura 22: Gráficos do relatório quantitativo de erros de testes Fonte: (iproject)

Ao consultar com a opção “Gerar CSV”, será gerado um arquivo CSV com os dados da consulta. Tal arquivo será utilizado na exportação de dados para ferramenta BI.

Ao clicar em algum valor numérico da coluna “Quantidade” do relatório, o fluxo será redirecionado para uma tela onde serão listadas todas as tarefas utilizadas na contabilização dessa quantidade. A Figura 23 ilustra a listagem das tarefas.

Figura 23: Listagem de Tarefas Fonte: (iproject)

Além disso, é possível visualizar os detalhes e os logs da tarefa clicando no número da tarefa ou no ícone “Detalhes da Tarefa”. A Figura 24 exibe os detalhes de uma tarefa.

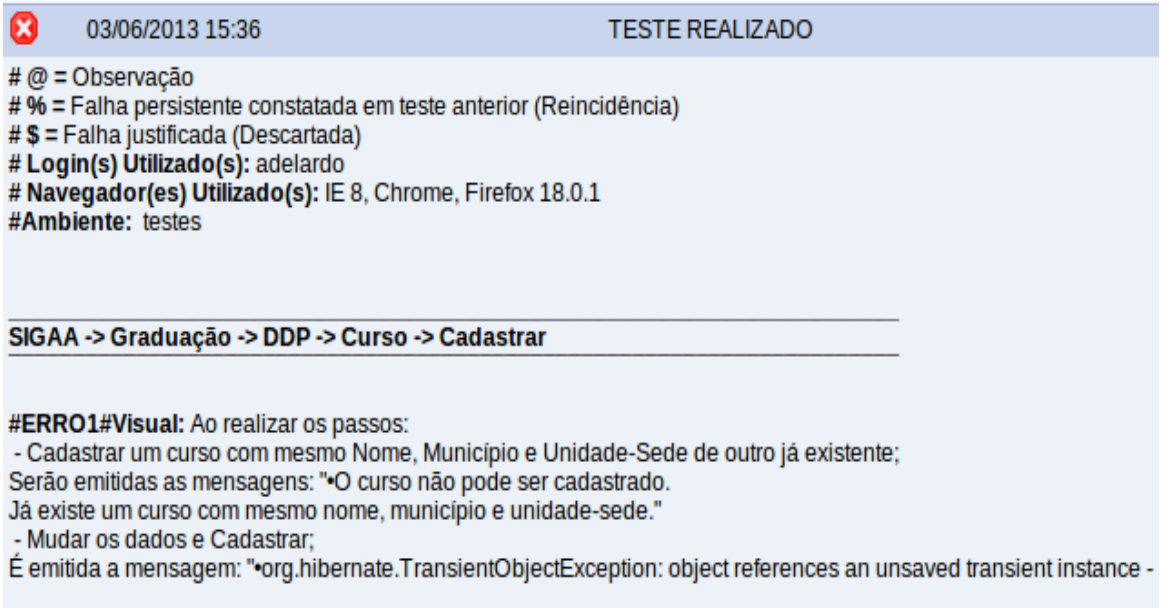


Figura 24: Logs ao visualizar detalhes de uma tarefa Fonte: (iproject)

5.6 Considerações Finais

Neste Capítulo foi apresentado em detalhes a ferramenta implementada e inserida no iProject. A ferramenta possui como objetivo ajudar na análise das quantidades de falhas e tipos de erros encontrados nos sistemas através dos testes exploratórios ou identificados pelos usuários finais no ambiente de produção.

6 Visualizador de Falhas SINFO/UFRN – Versão integrada ao Pentaho

Este Capítulo irá apresentar a ferramenta BI para visualização de falhas que foi desenvolvida no contexto desse trabalho. A Seção 6.1 faz uma breve apresentação do contexto em que a ferramenta está inserida. A Seção 6.2 exibe de modo geral as principais operações da ferramenta. A Seção 6.3 discorre sobre o modelo dimensional. A Seção 6.4 apresenta a apresenta detalhadamente as tabelas do modelo. A Seção 6.5 mostra como foi feito o processo de ETL. A Seção 6.6 expõe resumidamente o cubo erros. A Seção 6.7 exibe algumas métricas que podem ser obtidas através da ferramenta. A Seção 6.8 faz uma discussão final sobre a implementação da ferramenta BI.

6.1 Contexto

O visualizador BI de falhas foi desenvolvido como resultado de uma investigação cujo objetivo foi investigar como um sistema de *Business Intelligence* poderia ser utilizado para apoiar a atividade de análise de falhas reportadas pela equipe de Suporte e Controle de Qualidade da SINFO, descritas no Capítulo anterior. A ferramenta desenvolvida nesta etapa do trabalho está inserida na plataforma Pentaho e se constitui de um cubo OLAP no Mondrian capaz de contabilizar a quantidade de falhas por tipo, sistema, responsável, tarefa e data.

Além da ferramenta ter sido criada para motivar a utilização de plataformas de BI, ela também oferece diversas vantagens já mencionadas nos Capítulos anteriores, como: a capacidade do

usuário final interagir com a informação e a eliminação da necessidade de criação de novos formulários Webs por parte da equipe de desenvolvimento.

6.2 Visão Geral

Como apresentado nos Capítulos 2 e 3 a plataforma Pentaho permite o desenvolvimento de um Cubo OLAP integrado com um *Data Warehouse* para realização de consultas sobre informações do cubo, visualização, exploração e análise preditiva de dados. Por esta razão, a ferramenta desenvolvida implementa um Cubo OLAP com as seguintes dimensões: (i) Tipo de Erro; (ii) Sistema; (iii) Responsável; (iv) Tarefa e (v) Data e com a métrica (vi) Quantidade de Erros. Uma vez que o cubo tenha sido criado, a plataforma se encarrega da geração de gráficos e análises deixando para o usuário apenas o trabalho percorrer o cubo e realizar as operações já mencionadas nos Capítulos anteriores, como cortar, girar, expandir, etc. O usuário poderá navegar em diferentes níveis de hierarquia, como por exemplo, o ano, o mês ou o dia específico em que a falha ocorreu.

Pelo o fato desta versão da ferramenta ter sido construída com intenção realizar uma demostracao inicial da utilidade da ferramenta Pentaho neste contexto, a mesma ainda não foi incluida no no ambiente de producao da SINFO. Porém este trabalho foi adicionado no *Wiki* da SINFO e pode ser utilizado como tutorial sobre o Pentaho que será útil para futuras iniciativas que queiram continuar este trabalho.

Para acessar a ferramenta é necessário além de instalar a plataforma Pentaho e integrá-la com a base de dados, deve-se exportar o arquivo XML que representa o cubo utilizando a ferramenta Schema Workflow.

É importante ressaltar que para a utilização da plataforma Pentaho é necessário apenas instalá-la, visto que a mesma oferece um banco de dados e cubos pré-definidos para demonstração.

Para acessar as operações o usuário deverá se logar no Pentaho como administrador, clicar em *New Analysis* e escolher o *schema* Erros, como ilustrado na Figura 25.

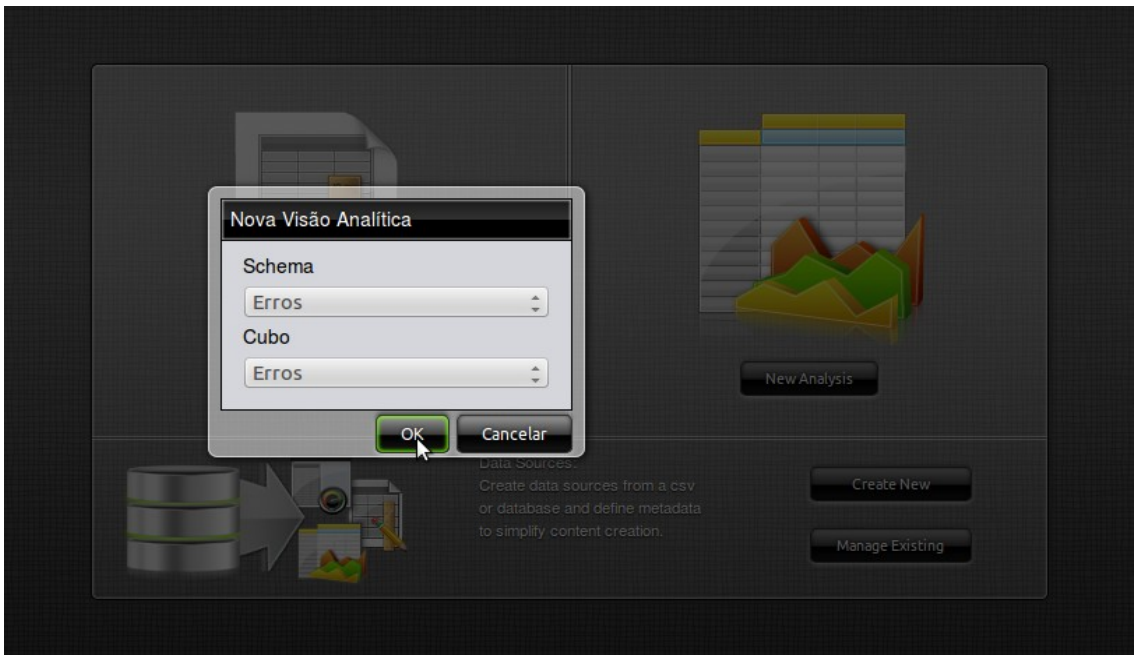


Figura 25: Acesso ao cubo de erros

Após criar uma nova análise, o cubo de erros será exibido, juntamente com um série de opções do Mondrian. O cubo de erros conterá as dimensões: Tipo de Erro, Sistema, Responsável, Tarefa e Tempo. O cubo também conterá a métrica Quantidade. A Figura 26 ilustra o cubo de erros.

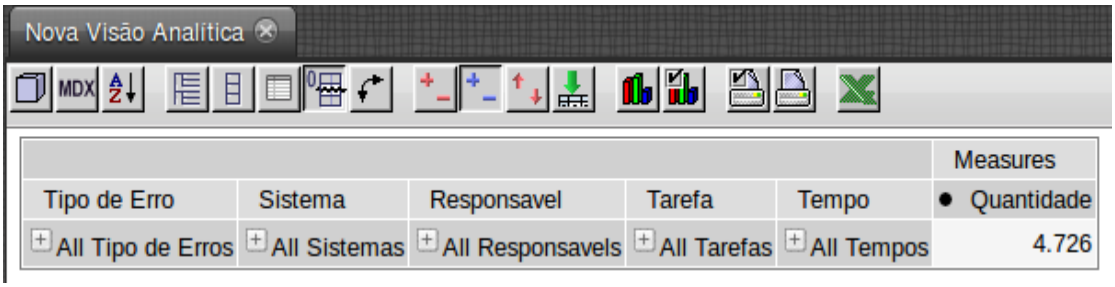


Figura 26: Cubo de erros

O usuário poderá interagir com o cubo realizando operações de *Drill up* e *Drill Down* para obter diferentes níveis de detalhamento da informação. A Figura 27 ilustra o *Drill Down* na dimensão tipo Erro.

					Measures
Tipo de Erro	Sistema	Responsavel	Tarefa	Tempo	Quantidade
All Tipo de Erros	All Sistemas	All Responsavels	All Tarefas	All Tempos	4.726
#BD	All Sistemas	All Responsavels	All Tarefas	All Tempos	338
#CI	All Sistemas	All Responsavels	All Tarefas	All Tempos	635
#Navegacao	All Sistemas	All Responsavels	All Tarefas	All Tempos	465
#Negocio	All Sistemas	All Responsavels	All Tarefas	All Tempos	1.124
#Visual	All Sistemas	All Responsavels	All Tarefas	All Tempos	2.164

Figura 27: *Drill down* em tipos de erros

O Mondrian também oferece a funcionalidade chamada OLAP *Navigator* que permite ao usuário personalizar a visão do cubo cortando-o de diversas maneiras diferentes, seja linhas ou colunas. O OLAP *Navigator* é apresentado na Figura 28.

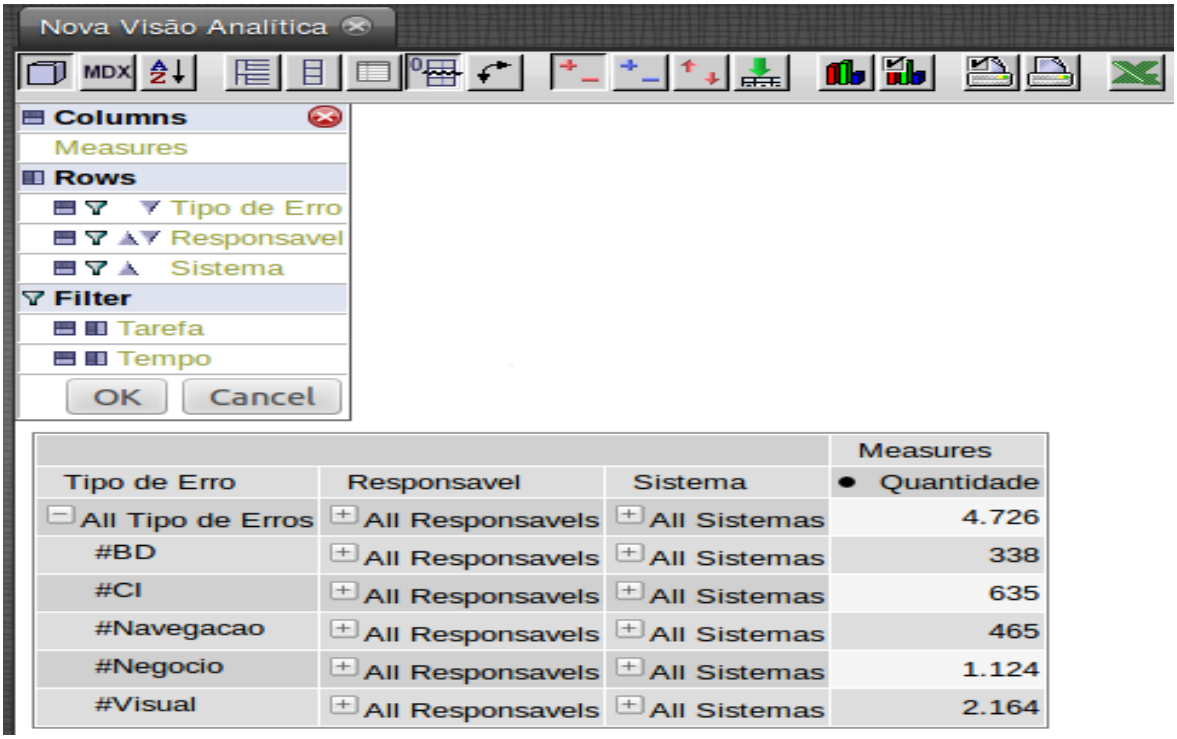


Figura 28: OLAP *Navigator*

O Mondrian também permite que o usuário troque as linhas por colunas no cubo através da operação *Swap Axes*. O resultado da operação é exibido na Figura 29.

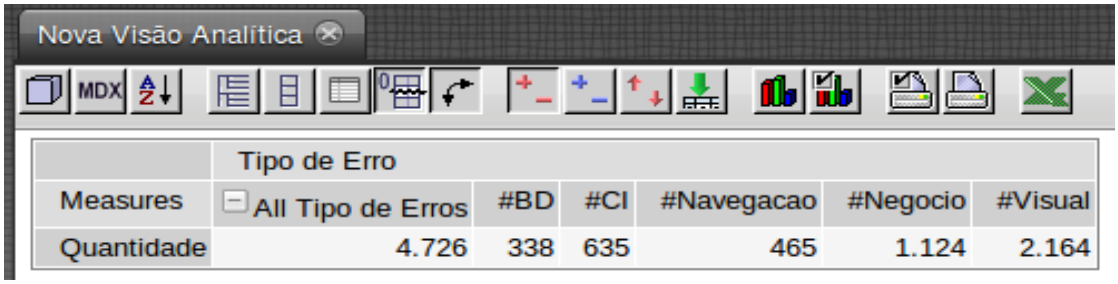


Figura 29: Operação de *Swap Axes*

Outra funcionalidade útil e importante do Mondrian é a capacidade de gerar e conFigurar facilmente diversos tipos de gráficos para auxiliar a visualização do cubo. A Figura 30 exibe a tela de conFiguração das propriedades dos gráficos.

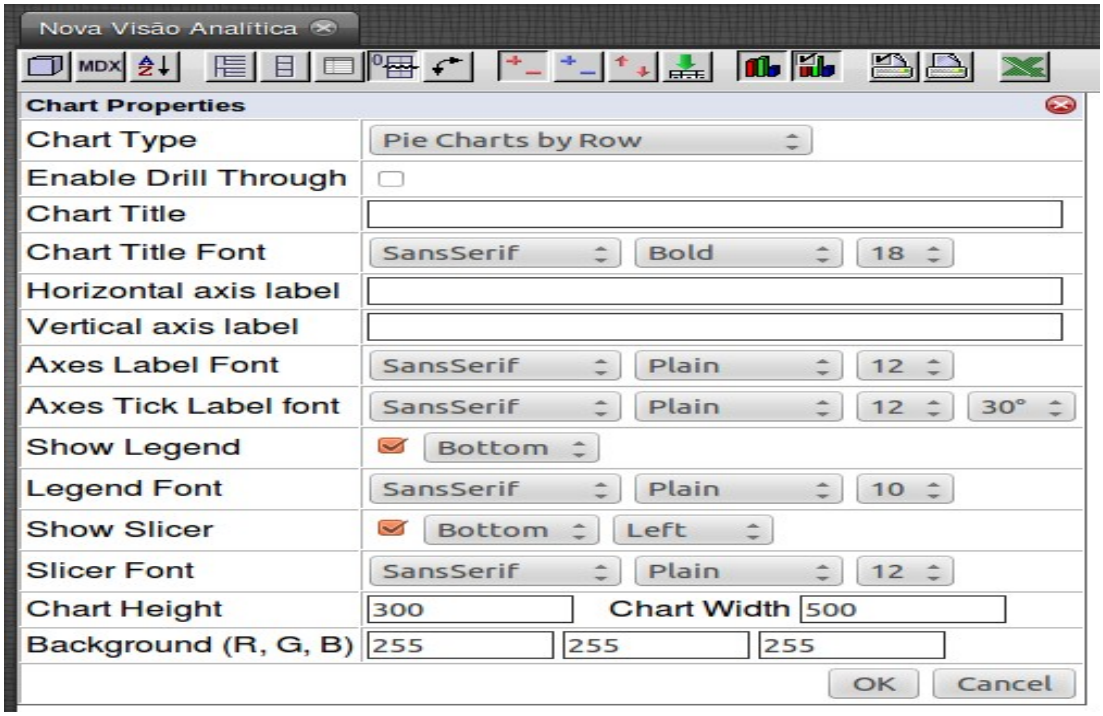


Figura 30: Propriedades dos Gráficos

Com o Mondrian foi possível criar os gráficos de barra e de pizza semelhantes aos da ferramenta Web, mas com a vantagem dos gráficos se adequarem dinamicamente as diferentes visões do cubo. O gráfico de pizza é ilustrado na Figura 31.

			Measures
Tipo de Erro	Responsavel	Sistema	Quantidade
All Tipo de Erros	All Responsavels	All Sistemas	4.726
#BD	All Responsavels	All Sistemas	338
#CI	All Responsavels	All Sistemas	635
#Navegacao	All Responsavels	All Sistemas	465
#Negocio	All Responsavels	All Sistemas	1.124
#Visual	All Responsavels	All Sistemas	2.164

Slicer:

Slicer:

All Tipo de Erros.All Responsavels.All Sistemas. #BD.All Responsavels.All Sistemas.
#CI.All Responsavels.All Sistemas. #Navegacao.All Responsavels.All Sistemas.
#Negocio.All Responsavels.All Sistemas. #Visual.All Responsavels.All Sistemas.

Figura 31: Gráfico de Pizza do Mondrian

6.3 Modelo Dimensional

O modelo dimensional foi feito com o auxílio da ferramenta SQL *Power Architect*. Após o modelo ter sido desenhado, ele foi inserido no banco de dados através da opção de engrenharia reversa da ferramenta SQL *Power Architect*. Na Figura 32 segue a visão geral do modelo.

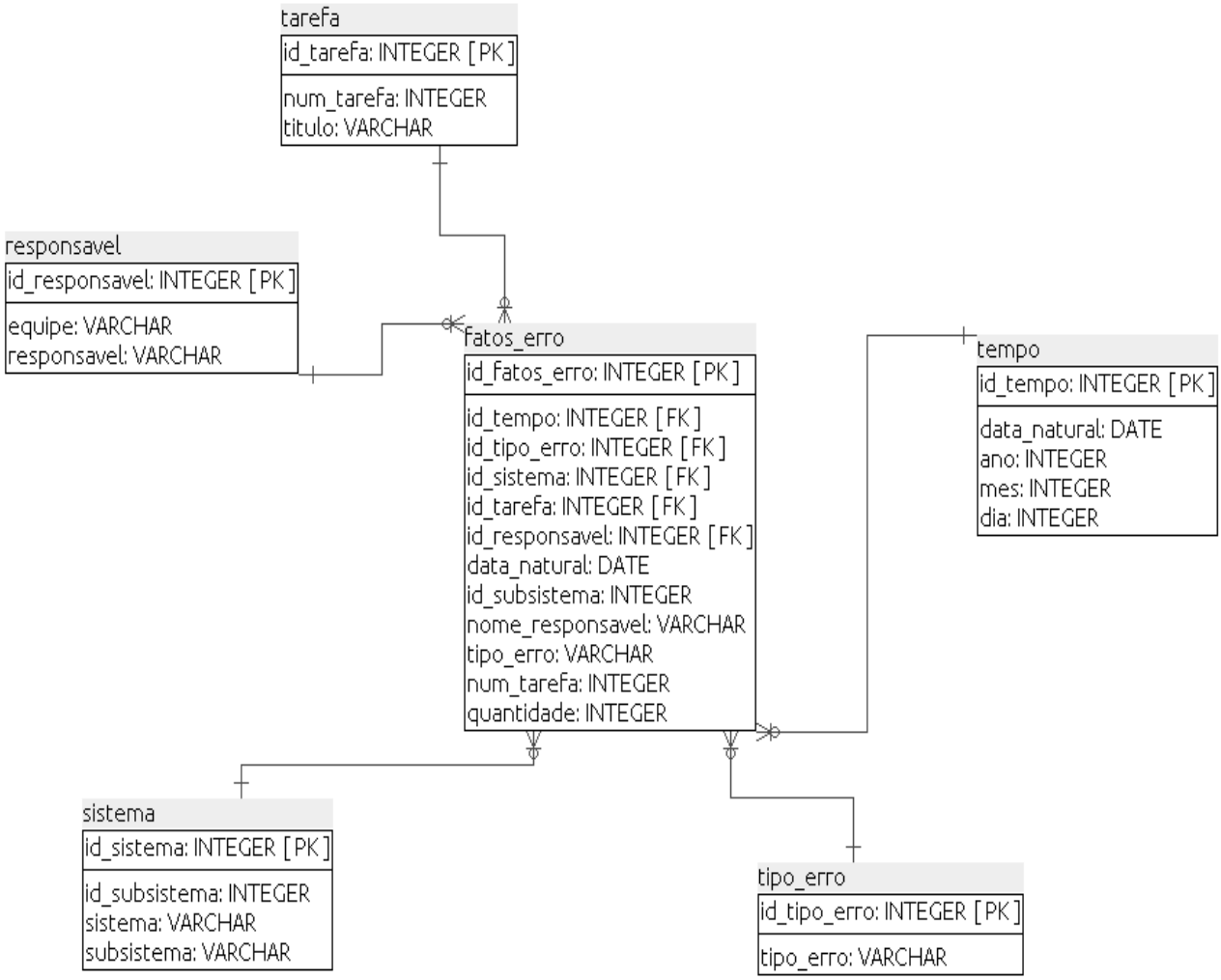


Figura 32: Modelo Dimensional

O coração do modelo é a tabela fato, fatos_erro. A tabela fato_erro contém as chaves estrangeiras para as tabelas dimensionais, as chaves naturais utilizadas no processo de ETL e a métrica Quantidade. Orbitando a tabela fatos_erro estão as demais tabelas dimensionais que serão descritas detalhadamente na Seção a seguir.

6.4 Tabelas Detalhadas

Nesta Seção serão apresentadas detalhadamente cada tabela que compõe o modelo dimensional. A Subseção 6.4.1 expõe a tabela fatos_erro. A Subseção 6.4.2 mostra a tabela tarefa. A Subseção 6.4.3 mostra a tabela responsavel. A Subseção 6.4.4 apresenta a tabela tempo. A Subseção 6.4.5 trata da tabela sistema. Para concluir, a Subseção 6.4.6 expõe a tabela tipo_erro.

6.4.1 Tabela fatos_erro

Como já foi dito anteriormente, a tabela fatos_erro é a tabela fato do modelo dimensional. Ele está ligada por chaves estrangeiras às demais tabelas dimensionais e contém a métrica Quantidade. A tabelas fatos_erro também contém as chaves naturais que serão utilizadas para compor o relacionamento com as tabelas dimensionais durante o processo de ETL. A utilidade das chaves naturais será explicada na Seção 6.5 que discorre sobre o processo ETL. Abaixo, segue a Figura 33 com a ilustração da tabela fatos_erro.

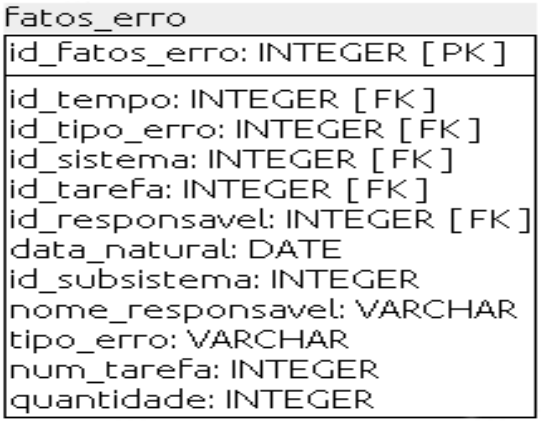


Figura 33: Tabela fatos_erro

Abaixo segue a descrição das colunas da tabela.

- id_fatos_erro: Chave primária.

- id_tempo: Chave estrangeira utilizada para compor o relacionamento com a tabela tempo.
- id_tipo_erro: Chave estrangeira utilizada para compor o relacionamento com a tabela tipo_erro.
- id_sistema: Chave estrangeira utilizada para compor o relacionamento com a tabela sistema.
- id_tarefa: Chave estrangeira utilizada para compor o relacionamento com a tabela tarefa.
- id_responsavel: Chave estrangeira utilizada para compor o relacionamento com a tabela responsavel.
- data_natural: Chave natural que representa a data em que o fato ocorreu.
- id_subsistema: Chave natural que representa o subsistema que o fato ocorreu.
- nome_responsavel: Chave natural que representa o nome do responsável que registrou fato.
- tipo_erro: Chave natural que contém a *hashtag* do tipo do erro do fato.
- num_tarefa: Chave natural que contém o número da tarefa do fato.
- Quantidade: Métrica do fato. Basicamente é a resposta da pergunta: “Qual a quantidade de erros do tipo <TipoErro>, foram encontrados no subsistema <sistema>, no dia <data>, registrados pelo responsável <responsavel>, na tarefa <tarefa>?”.

6.4.2 Tabela tarefa

Tabela dimensional que representa a tarefa em que a falha foi encontrada. A Figura 34 ilustra a tabela dimensional tarefa.

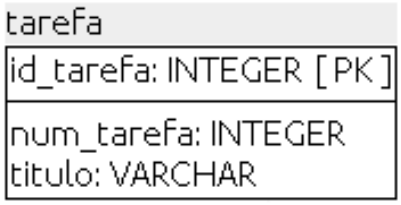


Figura 34: Tabela tarefa

Abaixo segue a descrição das colunas da tabela.

- id_tarefa: Chave primária.
- num_tarefa: Chave natural que guarda o número da tarefa do iproject.
- titulo: Descritor da tarefa que irá aparecer no cubo OLAP, o título é composto pelo número da tarefa mais seu título.

6.4.3 Tabela responsavel

Tabela dimensional que representa o colaborador responsável por ter registrado a falha. A Figura 35 ilustra a tabela dimensional responsavel.

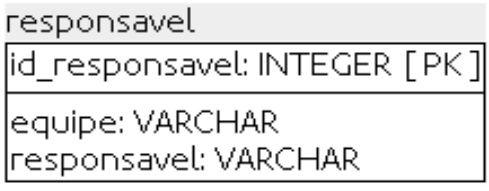


Figura 35: Tabela responsavel

Abaixo segue a descrição das colunas da tabela.

- id_responsavel: Chave primária.
- equipe: Nome da equipe do responsável: Controle de Qualidade ou Suporte ao Usuário. Será visualizado no cubo OLAP como o nível mais alto da hierarquia.
- responsavel: Nome completo do colaborador que registrou a falha na tarefa. O nome do responsável, além de aparecer na visualização do cubo OLAP também irá servir como chave natural.

6.4.4 Tabela tempo

Tabela dimensional que representa a data em que a falha foi registrada. A Figura 36 ilustra a tabela dimensional tempo.

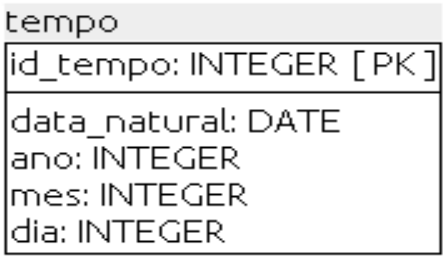


Figura 36: Tabela tempo

Abaixo segue a descrição das colunas da tabela.

- id_tempo: Chave primária.
- data_natural: Chave natural que guarda a data completa em que ocorreu a falha.
- ano: Ano em que ocorreu a falha. Será visualizado no cubo OLAP como o nível mais alto da hierarquia tempo.

- mes: Mês em que ocorreu a falha. Será visualizado no cubo OLAP como o nível intermediário da hierarquia.
- dia: Dia em que ocorreu a falha. Será visualizado no cubo OLAP como o nível mais baixo da hierarquia tempo.

6.4.5 Tabela sistema

Tabela dimensional que representa o sistema e o subsistema da tarefa em que a falha foi registrada. A Figura 37 ilustra a tabela dimensional sistema.

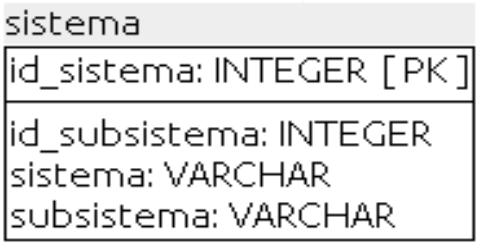


Figura 37: Tabela sistema

Abaixo segue a descrição das colunas da tabela.

- id_sistema: Chave primária.
- id_subsistema: Chave natural que identifica o subsistema em que ocorreu a falha.
- sistema: Nome do sistema em que ocorreu a falha que será visualizado no cubo OLAP. Nível mais alto da hierarquia.
- subsistema: Nome do subsistema em que ocorreu a falha. Será visualizado no cubo OLAP e é o nível mais baixo da hierarquia sistema.

6.4.6 Tabela tipo_erro

Tabela dimensional que representa o tipo da falha registrada. A Figura 38 ilustra a tabela dimensional tipo_erro.

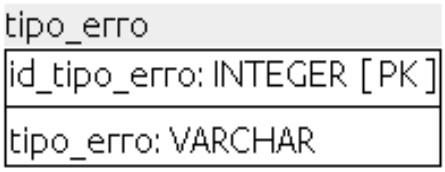


Figura 38: Tabela tipo_erro

Abaixo segue a descrição das colunas da tabela.

- id_tipo_erro: Chave primária.
- tipo_erro: Tipo de erro da falha, além de ser visualizada no cubo OLAP, também serve de chave natural.
- sistema: Nome do sistema em que ocorreu a falha que será visualizado no cubo OLAP. Nível mais alto da hierarquia.
- subsistema: Nome do subsistema em que ocorreu a falha. Será visualizado no cubo OLAP e é o nível mais baixo da hierarquia sistema.

6.5 Processo de ETL

O processo de ETL foi desenvolvido em 3 etapas. A primeira etapa se constituiu na extração e limpeza dos dados utilizando a ferramenta Web descrita nos Capítulos anteriores. A segunda etapa foi o carregamento do banco de dados com as informações extraídas, nesta etapa foi utilizada a ferramenta PDI. Por fim, foi feito um último processo de transformação utilizando *scripts* SQL.

Durante a primeira etapa, foi utilizado a o visualizador Web de falhas do iProject para gerar arquivos CSV contendo os dados das tabelas fatos_erro e tarefa. Como nesta etapa as tabelas dimensionais ainda não estão populadas no *Data Warehouse* e a informação para as chaves estrangeiras ainda não existe, a ferramenta Web gera o CSV apenas com os dados das chaves naturais. Além disso, o visualizador Web não só extrai os dados, mas também fez a limpeza dos mesmos, garantindo sua consistência.

A estratégia de utilizar a ferramenta Web no processo de ETL foi escolhida visto que as *hashtags* que identificam o tipo de erro são registradas em campos textuais e as bibliotecas Java oferecem um melhor suporte para o tratamento de Strings. Além disso, foi possível aproveitar a consulta da ferramenta Web para buscar as informações necessárias diminuindo o esforço. A extração das demais tabelas dimensionais foi feita através de *scripts* SQL simples.

A segunda etapa, foi utilizado o PDI para popular o *Data Warehouse*. Tal etapa se constituiu apenas de criar transformações no PDI que recebiam como entrada os CSVs gerados pela ferramenta Web e planilhas xls geradas a partir dos *scripts* SQL e tinham como *output* as tabelas do *Data Warehouse* que deveriam ser populadas. A Figura 39 ilustras as transformações feitas no PDI.

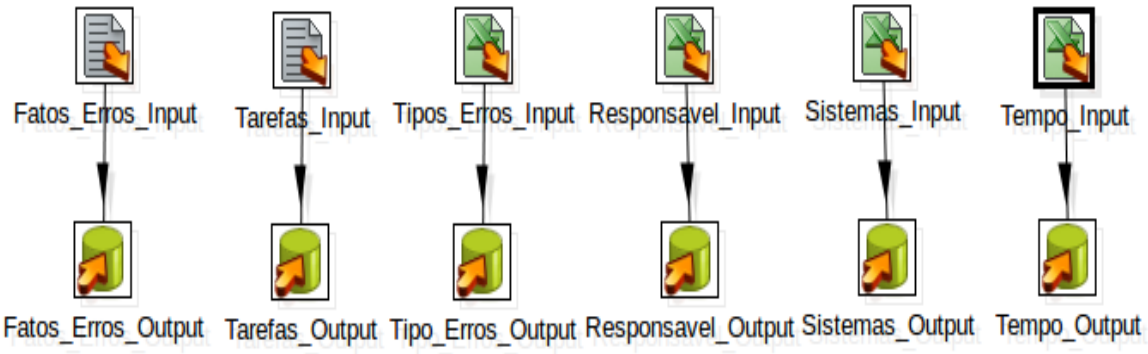


Figura 39: Transformações do PDI

A última etapa se constitui de criar as relações entre a tabela fato e as tabelas dimensionais. Para isso são utilizados *scripts* SQL que atualizam as chaves estrangeiras da tabela fato com as chaves primárias das tabelas dimensionais através de um *update* que faz uma comparação entre as chaves naturais das tabelas. É necessário um *script* para cada tabela dimensional, no entanto, tais *scripts* são bastantes simples e não demandam tempo.

6.6 Cubo Erros

O cubo erros foi criado com a ferramenta Schema Workflow. Para criá-lo bastou adicionar a tabela fato, as dimensões e as métricas abaixo da tabela fato e as hierarquias e os níveis abaixo das dimensões.

Com o esboço do cubo pronto foi necessário mapear cada elementos com suas respectivas tabelas e colunas do *Data Warehouse* e exportar o *schema* salvo para o Pentaho. Ao final do processo o cubo acabou composto por 5 dimensões referentes as tabelas dimensionais. Isto faz com que o cubo possa ser denominado hipercubo. Abaixo segue a Figura 40 que ilustra o mapeamento do cubo na ferramenta Schema Workflow.

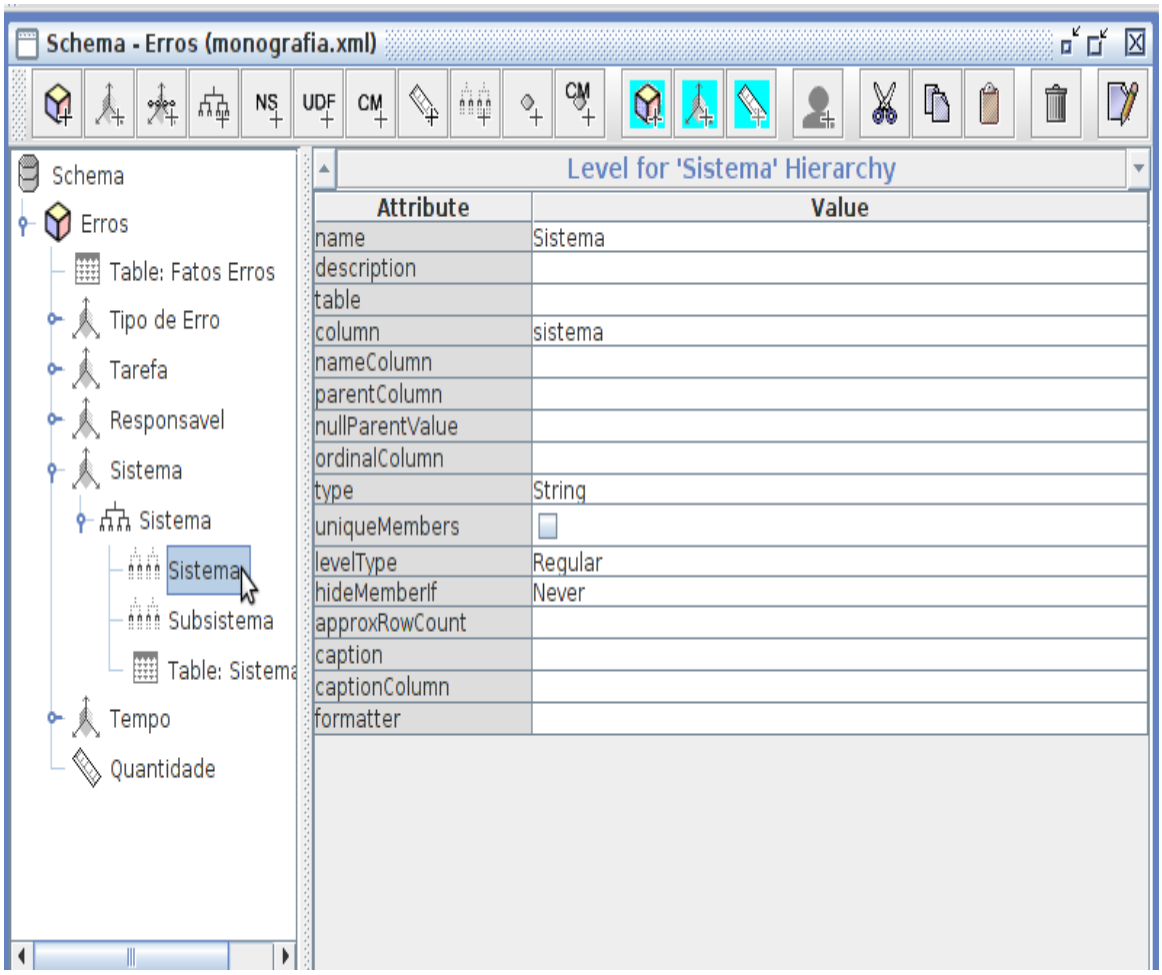


Figura 40: Mapeamento do cubo

Abaixo segue a Figura 41 que exibe a hierarquia de sistemas do cubo:

		Measures
Tipo de Erro	Sistema	Quantidade
All Tipo de Erros	All Sistemas	4.726
	COMUM	24
	IPROJECT	50
	PORTAIS	1
	REDE SOCIAL	287
	SIGAA	2.478
	SIGADMIN	22
	SIGEVENTO	3
	SIGPP	6
	SIGRH	647
	SIPAC	1.046
	SUCUPIRA	162

Figura 41: Hierarquia do cubo no pentaho

6.7 Métricas

Este Seção tem como objetivo apresentar algumas métricas que foram possíveis de obter com o desenvolvimento da ferramenta BI. As métricas neste Capítulo consideram todos as falhas registradas pelas equipes de controle de qualidade e suporte ao usuário, embora seja possível filtrá-las por um determinado intervalo de tempo. Subseção 6.7.1 irá apresentar a quantidade de falhas por sistema. A Subseção 6.7.2 irá apresentar a quantidade em que aparece cada tipo de erro nos sistemas. A Subseção 6.7.3 irá mostrar a quantidade de falhas nos principais módulos do SIPAC. A Subseção 6.7.4 irá mostrar a quantidade de falhas nos principais módulos do SIGAA. A Subseção 7.5 irá a mostrar a quantidade de falhas na turma virtual.

6.7.1 Quantidade de Falhas por Sistema

Abaixo segue a Figura 42 exibindo a quantidade de falhas encontradas nos sistemas:

		Measures
Tipo de Erro	Sistema	Quantidade
All Tipo de Erros	All Sistemas	4.726
	SIGAA	2.478
	SIGADMIN	22
	SIGRH	647
	SIPAC	1.046
	SUCUPIRA	162

Slicer:

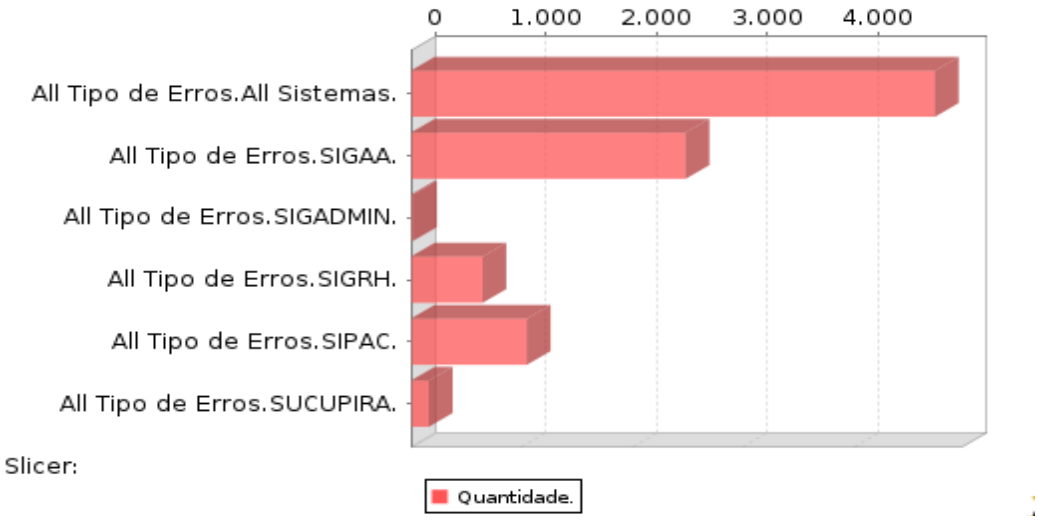


Figura 42: Quantidade de Falhas por Sistemas

6.7.2 Quantidade de Tipos de Erro

Abaixo segue a Figura 43 exibindo a quantidade de tipos de erro:

		Measures
Tipo de Erro	Sistema	Quantidade
All Tipo de Erros	All Sistemas	4.726
#BD	All Sistemas	338
#CI	All Sistemas	635
#Navegacao	All Sistemas	465
#Negocio	All Sistemas	1.124
#Visual	All Sistemas	2.164

Slicer:

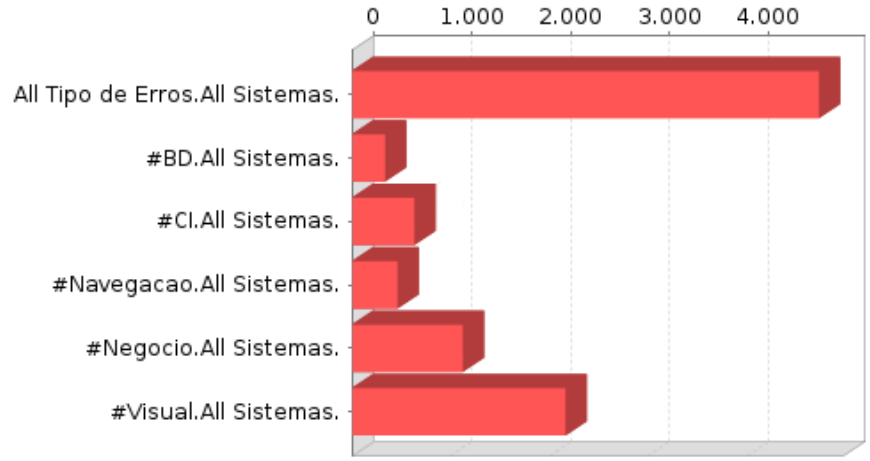


Figura 43: Quantidade de Tipos de Erros

6.7.3 Quantidade de falhas nos principais módulos do SIPAC

Abaixo segue a Figura 44 exibindo a quantidade de falhas nos principais módulos do SIPAC. O nível de informação encontrado nesse relatório não é obtido através da ferramenta Web:

		Measures
Tipo de Erro	Sistema	Quantidade
All Tipo de Erros	BIBLIOTECA	77
	COMPRAS	82
	CONTRATOS	41
	INFRA-ESTRUTURA	38
	PATRIMONIO IMOVEL	123
	PORTAL ADMINISTRATIVO	31
	PROJETOS/CONVENIOS	48
	PROTOCOLO	36
	REQUISICOES	63
	TRANSPORTES	225

Slicer:

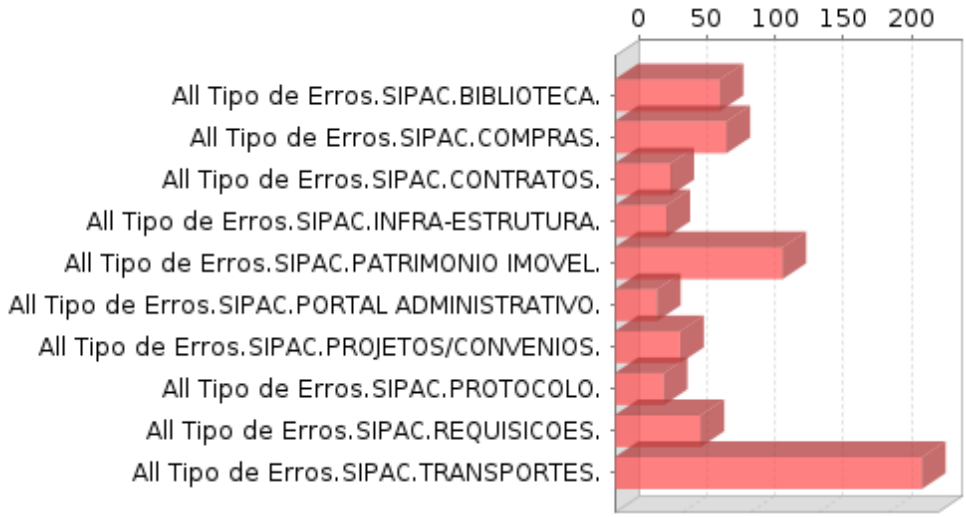


Figura 44: Quantidade de falhas nos principais sistemas do SIPAC

6.7.4 Quantidade de falhas nos principais módulos do SIGAA

Abaixo segue a Figura 45 exibindo a quantidade de falhas nos principais módulos do SIGAA. O nível de informação encontrado nesse relatório não é obtido através da ferramenta Web:

		Measures
Tipo de Erro	Sistema	Quantidade
All Tipo de Erros	BIBLIOTECA	144
	ENSINO EM REDE	208
	EXTENSAO	254
	GRADUACAO	321
	INFANTIL	29
	LATO SENSU	83
	MEDIO	79
	METROPOLE DIGITAL	206
	MONITORIA	113
	PORTAL DO DISCENTE	35
	PORTAL DO DOCENTE	31
	STRICTO SENSU	159
	TECNICO	43
	TURMA VIRTUAL	86

Slicer:

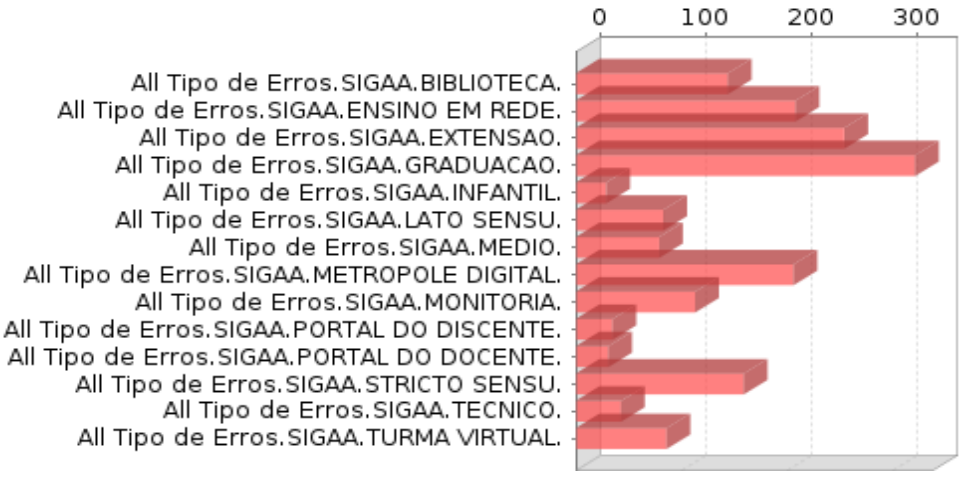


Figura 45: Quantidade de falhas nos principais sistemas do SIGAA

6.7.5 Quantidade de falhas da Turma Virtual

Abaixo segue a Figura 46 exibindo a quantidade de falhas na turma virtual, um dos módulos mais acessados e conhecidos do SIGAA. O nível de informação encontrado nesse relatório não é obtido através da ferramenta Web:

		Measures
Tipo de Erro	Sistema	Quantidade
All Tipo de Erros	TURMA VIRTUAL	86
#BD	TURMA VIRTUAL	6
#CI	TURMA VIRTUAL	15
#Navegacao	TURMA VIRTUAL	5
#Negocio	TURMA VIRTUAL	34
#Visual	TURMA VIRTUAL	26

Slicer:

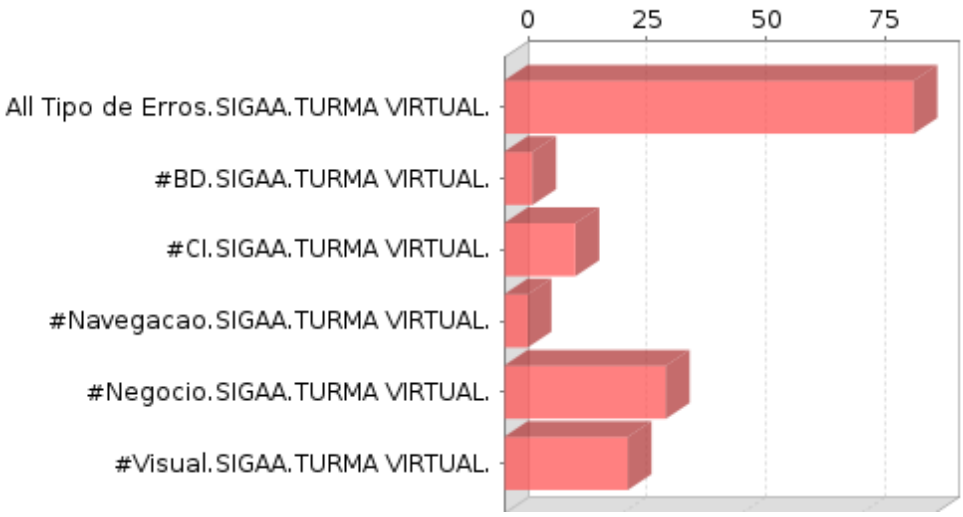


Figura 46: Quantidade de falhas na Turma Virtual

6.8 Discussões

Com o objetivo de apoiar iniciativas futuras na utilização de BI em atividades da SINFO, como também outros projetos que não tenham experiência em BI, falarei um pouco sobre o desenvolvimento das duas versões desta ferramenta. Procurarei responder a seguinte questão: é vantajoso abandonar um módulo web composto por dezenas de relatórios para a criação de uma aplicação BI?

Nesta Seção irei discutir um pouco sobre o esforço realizado na criação das duas versões do Visualizador de Falhas e nas funcionalidades que cada versão oferece.

6.8.1 Comparação de Esforço

O desenvolvimento da versão Web foi realizado ao longo de 1 mês. Para esta versão foi necessário implementar (i) 2 JSPs, (ii) 2 classes de domínio, (ii) 1 Controlador, (iv) novas operações numa classe DAO.

Já o desenvolvimento da versão BI da ferramenta (integrada ao Pentaho), se for desconsiderado o tempo investido no aprendizado teórico, levou 1 semana para ser concluído. Podemos perceber que foi demandado menos esforço no seu desenvolvimento do que no desenvolvimento da ferramenta Web. Para esta versão foi necessário implementar: (i) 6 tabelas, (ii) 1 arquivo de script SQL para auxiliar a etapa de transformação, (iii) 1 arquivo de script SQL gerado com o auxílio da ferramenta SQL *Power Architect,* (iv) 1 arquivo de script SQL gerado com o auxílio da ferramenta Pentaho *Data Integration*, (v) 1 arquivo XML gerado pela ferramenta Schema Workbench. Ou seja, nesta versão não foi necessário desenvolver a interface para integração dos dados, já que o Pentaho oferece o suporte necessário para que o usuário possa criar sua própria visão da informação.

No entanto, deve-se considerar que a ferramenta BI utilizou da ferramenta Web no auxílio do processo de ETL. Também deve-se ponderar que em um ambiente de produção seria necessário personalizar a plataforma Pentaho para que ela se adeque ao padrão de layout da corporação – como a ferramenta é *Open Source* esta personalização é possível. Por último também é preciso considerar que em um contexto mais complexo talvez a modelagem dimensional necessite de mais esforço, neste caso, deve-se fazer um exame mais detalhado das regras negociais da corporação antes do início da implementação.

6.8.2 Funcionalidades fornecidas por cada versão

No levantamento de funcionalidades, a ferramenta BI supera largamente a ferramenta Web, pois a primeira oferece várias opções para a visualização da informação. Por exemplo, a Figura 47 ilustra uma análise que só pode ser feita utilizando a ferramenta BI.

		Measures
Tipo de Erro	Sistema	Quantidade
All Tipo de Erros	TURMA VIRTUAL	86
#BD	TURMA VIRTUAL	6
#CI	TURMA VIRTUAL	15
#Navegacao	TURMA VIRTUAL	5
#Negocio	TURMA VIRTUAL	34
#Visual	TURMA VIRTUAL	26

Slicer:

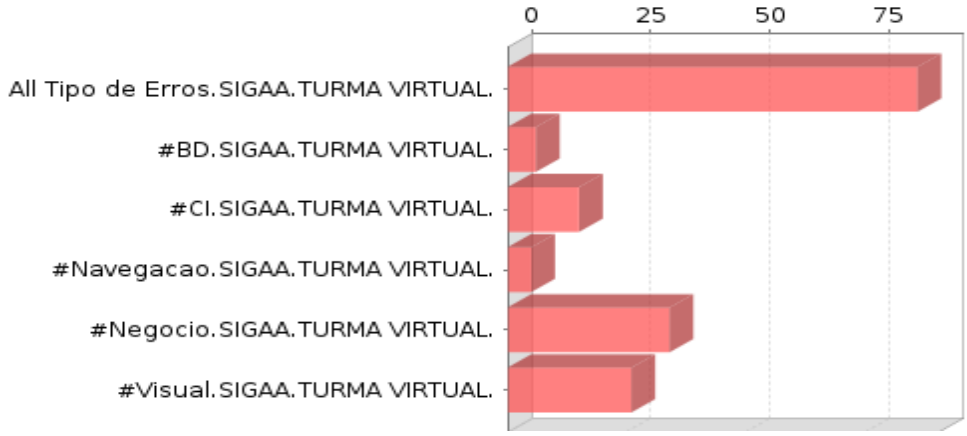


Figura 47: Análise única da ferramenta BI

Percebemos também que o Pentaho oferece interatividade com o usuário e diversos número de visões e gráficos, como por exemplo, gráficos em linha, em barra ou em pizza de informações escolhidas pelo próprio usuário. No entanto, a ferramenta Web possui a vantagem de está inserida no sistema transacional da corporação, ou seja, ela acessa a mesma base de dados onde as falhas são registradas pela equipe de controle de qualidade e suporte fazendo com que o usuário possa acessar todos os detalhes das falhas assim que as mesmas são reportadas por estas equipes. Além disso, caso o

usuário deseje navegar para uma operação diferente, como uma operação de persistência, ele não precisará se *deslogar* do sistema. Já a interface integrada ao Pentaho necessita que a base de dados seja migrada e atualizada para o *Data Warehouse* e o usuário terá o desconforto de acessar outro sistema para pode realizar suas análises.

Além disso, é importante salientar que caso a ferramenta BI esteja inserida em um contexto com outros tipo de usuário, como por exemplo, usuários com perfil administrativo, é necessário que o usuário final esteja disposto a interagir com a plataforma ou o esforço de implementação terá sido em vão.

7 Considerações Finais

Neste Capítulo são apresentadas as conclusões deste trabalho. A Seção 7.1 irá apresentar as contribuições do trabalho. A Seção 7.2 irá discorrer sobre os possíveis trabalhos futuros.

7.1 Contribuição do Trabalho

As contribuições geradas pelo trabalho foram as seguintes:

- **Um padrão para identificação de falhas reportadas nos SIGS/UFRN:** Este padrão foi definido juntamente com os coordenadores de Controle de Qualidade, Suporte e com a orientadora deste trabalho e permite identificar e listar as falhas que são registradas pelas equipes de controle de qualidade e suporte ao usuário. As informações são rastreadas e exibidas pelas versões da ferramenta integrada ao iProject e integradas ao Pentaho;
- **Ferramentas para Visualizar Falhas integrada ao iProject:** Projeto e implementação de uma ferramenta para visualização e análise de falhas registradas num sistema corporativo. As ferramentas foram desenvolvidas utilizando-se tecnologia Java Enterprise Edition (JavaEE), além de: Hibernate 3.2, Java-Server Faces, RichFaces, Struts, EJB, Spring e o servidor de aplicação JBoss;
- **Ferramenta para Visualizar Falhas integrada ao Pentaho:** Foi ilustrada e descrita a implementação de uma ferramenta BI, desde a modelagem dimensional, o processo ETL e a criação do cubo OLAP. Além disso, foi mostrado a utilização de uma ferramenta BI para propósitos de engenharia de software;
- **Comparação inicial:** Realização de uma breve análise comparativa entre as ferramentas tendo como objetivo principal

justificar a utilização de ambas as ferramentas no contexto dos sistemas da UFRN.

7.2 Trabalhos Futuros

Está Seção apresenta alguns trabalhos futuros para continuidade do projeto.

- **Aprimoramento em diversos níveis da ferramenta BI:** A ferramenta BI pode ser estendida com a adição de novos dados. Como por exemplo, a build da tarefa que ocorreu falha ou os registros de e-mail de erro recebidos pelos sistema. É possível ser feita uma revisão do processo ETL, visto que a literatura é vasta em relação ao assunto. Também pode ser feito um estudo comparativo da aplicação de diversos plugins do pentaho na ferramenta.

- **Implementação e estudo de caso de uma aplicação BI em um módulo dos sistemas SIG:** Existem módulos dos sistemas SIGs que são compostos basicamente por operações de geração de relatórios, como por exemplo, o portal da reitoria. Tais módulos poderiam ser refeitos numa plataforma BI, além disso, poderia ser feito o estudo de caso da construção do módulo.

- **Impulsionar outras iniciativas para a qualidade dos SIG/UFRN:** As ferramentas desenvolvidas permitem identificar e analisar as falhas que ocorrem nos sistemas. Com tal informação é possível desenvolver outras iniciativas com objetivo de diminuir essas falhas encontradas. Por exemplo: se percebemos que a maioria das falhas encontradas em um dado período estão relacionadas a problemas na implantação de regras de negócio, um treinamento pode ser ministrado abordando as peculiaridades das regras de negócio de um dado módulo.

- **Implantar e avaliar a utilidade das ferramentas:** Implantar ambas as ferramentas no ambiente de produção e avaliar sua utilidade através de questionários elaborados para serem respondidos pelos gerentes das equipes.

- **Adicionar gráficos treemap na ferramenta integrada ao Pentaho:** Investigar a API Google Charts para aprimorar o módulo da ferramenta integrada ao iProject com o intuito de apresentar as informações em um gráfico treemap.

Referências Bibliográficas

ANDRADE, Luís André. “Definição de uma Arquitetura de Data Warehousing para Gestão de Ciência e Tecnólogia no Brasil.“ (Mestrado em Ciência da Computação) – Universidade Estadual de Maringá – 2004.

AUGUSTO, Denis; HENRIQUE Fábio; ANTONIO Luis; DEMARCO, Augusto. “Business Intelligence“. 2005.

BARBOSA, E.; MALDONADO, J.C.; VINCENZI, A.M.R.; DELAMARO, M.E; SOUZA, S.R.S. e JINO, M. “Introdução ao Teste de Software”. XIV Simpósio Brasileiro de Engenharia de Software 2000.

BARROS, Fábio. “Guia para implementação prática de um Data Warehouse.“ São Paulo: Revista Computer World, 2002, n.367.

BOLTON, Michael. “Evolving Understanding about Exploratory Testing”. 2008.

Disponível em:

<http://www.developsense.com/blog/2008/09/evolving-understanding-about/>

Acesso em: 28 nov. 2013.

BOUMAN, Roland. “Pentaho Data Integration: Kettle turns Data into Business”.

Disponível em:

<http://rpbouman.blogspot.com.br/2006/06/pentaho-data-integration-kettle-turns.html>

Acesso em: 28 out. 2013.

BUCHANAN, Leigh; OCONNEL Andrew. “A Brief History of Decision Making.“

CAETANO, Cristiano. “Testes Exploratórios de A a Z”. 2006

Disponível em:

<http://www.linhadecodigo.com.br/artigo/1102/testes-exploratorios-de-a-a-z.aspx>

Acesso em: 28 nov. 2013.

CAETANO, Cristiano. “Uma breve introdução a testes exploratórios”. 2012. Disponível em:

<http://www.qualister.com.br/introducao-sobre-testes-exploratorios-sbtm>

Acesso em: 28 nov. 2013.

COME, Gilberto. “Contribuição ao estudo da implementação de data warehousing: Um caso no setor de telecomunicações.“ (Mestrado em Administração de Empresas) - Universidade de São Paulo – 2001.

DIAS, Arilo. “Introdução a Teste de Software“. Engenharia de Software Magazine”. Ed. 01, 2007.

DIAS, Leandro. “Gerenciador de Falhas: Uma ferramenta para análise de fluxos de exceções nos sistemas SIG“. (Monografia de Ciência da Computação) - Universidade Federal do Rio Grande do Norte – 2012.

FERREIRA, Manuele; RAMOS, Gustavo; BERNARDO, Luis; SILVA, Robson; MAIOR, Bruno. “Plataforma Pentaho de Business Intelligence: Manual de Utilização”. Universidade Federal da Bahia – n.d.

Getting Started with Pentaho 3.7.0. Pentaho Corporation

HAN, Jiawei; KAMBER, Micheline. “Data Mining: Concepts and Techniques” 3a edição. Morgan Kaufmann. 2000.

IEEE Standard 610-1990: IEEE Standard Glossary of Software Engineering Terminology, IEEE Press.

INMON, W. H. “Building the Date Warehouse“. 3a edição. New York, NY, EUA. John Wiley and Sons, 2005.

KIMBALL, Ralph; CASERTA Joe. “The Data Warehouse ETL Toolkit“. New York, NY, EUA: John Wiley and Sons, 2004.

KIMBALL, Ralph; ROSS Margy; THORNTHWAITE, Warren; MUNDY, Joy; BECKER, Bob "The Data Warehouse Lifecycle Toolkit". 2a edição. New York, NY, EUA: John Wiley and Sons, 2008.

KIMBALL, Ralph; ROSS Margy. "The Data Warehouse Toolkit". 2a edição. New York, NY, EUA: John Wiley and Sons, 2002.

Documentação do Modrian. Disponível em:

<http://Mondrian.pentaho.com/documentation/>

Acesso em: 28 out. 2013

MOREIRA, Eduardo. "Modelo Dimensional para Data Warehouse."

Disponível em:

<http://imaster.com.br/artigo/3836>

Acesso em: 26 out. 2013.

MUNDIN, Édson. "Implementação de Data Warehouse para Pequenas Empresas: Estudo de Caso para o Setor de Distribuição de Medicamentos." (Especialização em Banco de Dados Oracle e DB2) – Centro Universitário de Maringá – 2009.

OLAP Council White Paper. Disponível em:
<http://olapcouncil.org/research/whtpaply.htm>

Acesso em: 27 out. 2013

Pentaho Infocenter. Disponível em:
<http://infocenter.pentaho.com/help/index.jsp>

Acesso em: 04 ago. 2013

QUINN, Brendan. “Use Session Based Testing to Structure Exploratory Testing”. 2013. Disponível em:

<http://www.techwell.com/2013/02/use-session-based-testing-structure-exploratory-testing>

Acesso em: 03 dez. 2013

REDDY, Satyanarayana; SRINIVASUO, Rallanbandi; CHANDER, Poorna; REDDY, Srikanth. “Data Warehousing, Data Mining, OLAP and OLTP technologies are Essentials Elements to Support Decission-Making Process in Industries“. Internacional Journal on Computer Science and Engineering , 2010

RIBEIRO, Viviane. “O que é ETL?”. Disponível em:

<http://vivieaneribeiro1.wordpress.com/2011/06/28/o-que-e-etl-2/>

Acesso em: 27 out. 2013.

SEZÕES, Carlos; OLIVEIRA, José; BAPTISTA, Miguel. “Business Intelligence“. Porta, Princípia , 2006

SIDEMAR, João. “BI Open Source: Conhecendo o Pentaho”. Disponível em:

<http://imasters.com.br/artigo/16080/gerencia-de-ti/bi-open-source-conhecendo-pentaho/>

Acesso em: 04 ago. 2013

TEIXEIRA, Erika; LOURDES, Mônica; MOREIRA, Teresinha; “OLAP: Características, Arquitetura e Ferramentas”. Instituto Vianna Júnior - 2007

THOMSEN, Erik. “OLAP Solutions” 2a edição. New York, NY, EUA: John Wiley and Sons, 2002.

TINKHAM, Andy; KANER, Cem; “Exploring Exploratory Testing” 2003

VIEIRA, Marcio. “Uma Introdução ao Pentaho Business Intelligence e Business Analytics Open Source“. Disponível em:

<http://softwarelivre.org/flisol2013-curitiba/uma-introducao-ao-pentaho-business-intelligence-e-business-analytics-open-source.pdf>

Acesso em: 04 ago. 2013