FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JEFERSON DAVI PARCKERT

ASPECTOS DE AUTOMATIZAÇÃO DA DOCUMENTAÇÃO DA APLICAÇÃO DE *WORKFLOW* DE TESTE

Marília
2006

JEFERSON DAVI PARCKERT

ASPECTOS DE AUTOMATIZAÇÃO DA DOCUMENTAÇÃO DA APLICAÇÃO DE *WORKFLOW* DE TESTE

Monografia apresentada ao curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides Soares da Rocha, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para a obtenção do Titulo de Bacharel em Ciência da Computação (Área de concentração Engenharia de Software).

Orientador
Profa. Dra. Maria Istela Cagnin Machado

Co-Orientador
Prof. Dr. Edmundo Sérgio Spoto

Marília
2006

PARCKERT, Jeferson Davi

Aspectos de Automatização da Documentação da Aplicação de *Workflow* de Teste / Jeferson Davi Parckert; orientador: Maria Istela Cagnin Machado; co-orientador: Edmundo Sérgio Spoto. Marília, SP: [s.n] 2003.

78 f.

Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides de Soares da Rocha

1. Automatização 2. Documentação 3. Workflow

CDD:

JEFERSON DAVI PARCKERT

ASPECTOS DE AUTOMATIZAÇÃO DA DOCUMENTAÇÃO DA APLICAÇÃO DE *WORKFLOW* DE TESTE

Banca examinadora da monografia apresentada ao curso de Bacharelado em Ciência da Computação do UNIVEM, / F.E.E.S.R., para a obtenção do Titulo de Bacharel em Ciência da Computação. Área de Concentração: Engenharia de Software.

Resultado: _____

Orientador: Prof. Dr. _____

1º Examinador _____

2º Examinador _____

Marília, 30 de Novembro de 2006

Dedico este trabalho
ao meu pai Néri Nestor Parckert,
a minha mãe Elisete Teresinha Parckert,
e a minha Irmã Dirléia Claudia Parckert
pela dedicação, incentivo, paciência e sacrifícios pela minha formação.

À minha namorada Vanessa Cristina de Abreu Costa ,
Pelo apoio, paciência, compreensão, amor, amizade,
Companheirismo e muito incentivo.

Pois sem a presença e o amor de todos não teria alcançado meu objetivo.

AGRADECIMENTOS

Agradeço a Deus pela vida, saúde e principalmente por ter me dado forças para concluir mais esta etapa da minha vida.

Meu sincero obrigado a minha namorada Vanessa, pelo apoio, amor, compreensão, amizade e companheirismo durante toda esta etapa da minha vida, onde sem o apoio e compreensão da mesma tudo seria mais difícil .

A minha família em especial aos meus pais e irmã pelo apoio dado durante toda a minha vida, pois sem o apoio destes este caminho seria árduo e difícil .

Ao prof. Edmundo Sérgio Spoto pela Co-orientação deste trabalho, pelo apoio, confiança e pela colaboração. Meu sincero muito obrigado.

Aos meus amigos e colegas de faculdade que de alguma maneira contribuíram para o desenvolvimento do mesmo. Em especial meu muito obrigado à Renato e José Ricardo e Ronaldo.

Agradeço a Profa. Maria Istela Caginin pelo apoio durante o desenvolvimento deste trabalho e pela confiança depositada em min.

Ao Prof. Auri Marcelo Rizzo Vincenzi pelo material concedido para o desenvolvimento do mesmo .

A todos os professores da Instituição pelo conhecimento a min transmitido durante o decorrer do curso.

PARCKERT, Jeferson Davi. Aspectos de Automatização da Documentação da Aplicação de *Workflow* de Teste. 2006, 78 f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília , Fundação de Ensino Eurípides Soares da Rocha, Marília 2006.

RESUMO

O desenvolvimento de sistemas de software envolve uma série de atividades, uma delas é a atividade de teste de software esta por sua vez possui um elevado grau de dificuldade, mesmo sob estas perspectivas são poucas as ferramentas existentes que apóiem a documentação da atividade de teste. Este trabalho tem como principal propósito a elaboração de uma ferramenta que apóie a documentação de teste de software de forma sistemática levando-se em consideração a Norma IEEE 829, uma maneira de se sistematizar a atividade de documentação da atividade de teste de software é por intermédio de um *workflow*. *Workflow* por sua vez é a automação total ou parcial de um processo de negocio, desta maneira permitindo um melhor controle na execução das tarefas resultando numa melhor qualidade do produto final.

Palavras-chave: Teste de software, *workflow,* documentação, ferramenta.

PARCKERT, Jeferson Davi. Aspectos de Automatização da Documentação da Aplicação de *Workflow* de Teste. 2006, 78 f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília , Fundação de Ensino Eurípides Soares da Rocha, Marília 2006.

ABSTRACT

The systems software development involves a series of activities, one of them is the software test activity this for your time has a high difficulty degree, even under these perspectives there are little existent tools that support the documentation of the test activity. This work has as main purpose a tool elaboration in a systematic way that supports software test documentation being taken in consideration norm IEEE 829, a way to systematize the documentation activity of the software test activity is through a workflow. Workflow for your time is the total or partial automation of a business process, this way allowing a better control in the tasks execution resulting in a better final product quality.

Keywords: *software test, workflow, documentation, toll*.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BD – Banco de dados

C-USO – Uso Computacional

GFC – Grafo de Fluxo de Controle

IEEE – *Institute of Electrical and Electronic Engineers*

FADAT – Ferramenta de Apoio a Documentação de Teste de Software

ITEA - *Information for European Advancement*

JSP – Java Server Pages

OO – Orientado a Objetos

P-USO – Uso Predicativo

RUP – *Rational Unified Process*

SGBD – Sistema Gerenciador de Banco de Dados

WFMC - *Workflow Management Coalition*

WFMS – *Workflow Management System*

UML – *Unified Modeling Language*

UP – *Unified Process*

SUMÁRIO

INTRODUÇÃO

Atualmente, uma grande parte da população mundial depende de aplicações de software para realizar suas atividades diárias. Se alguns sistemas de uso global deixarem de funcionar, cerca de 40% da população sofrerá as conseqüências do problema, Rocha *et al*. (2001).

O desenvolvimento de sistemas de software envolve uma série de atividades de produção em que as oportunidades de inserção de falhas humanas são enormes, Pressmam (1995).

Uma das maneiras de garantir a qualidade de determinado software é a aplicação da técnica de teste de software, a qual engloba 40% do esforço total de uma instituição na confecção de determinado software, Pressmam (1995),

O teste de software em geral é a última revisão quanto da especificação, projeto e codificação. Se o teste for executado de maneira correta provavelmente o software terá uma boa qualidade, caso contrário o sistema terá uma maior probabilidade de ter uma má qualidade.

Para que o teste de software tenha eficácia, o mesmo deverá ser devidamente planejado, rigorosamente executado e apropriadamente avaliado para que se possa determinar o término da execução do teste de software.

De acordo com diversos autores (Pressman, 1995; Rocha *et al.*, 2001; Sommerville, 2003), aplicar a técnica de teste não é uma atividade trivial, deste modo é interessante ter uma ferramenta que gerencie bem como apóie a aplicação de teste de software em determinado sistema.

Uma das maneiras de sistematizar as atividades de teste é por meio de *workflows*. Segundo a *Workflow Management Coalition (WFMC)*, *workflow* é a automação total ou parcial

de um processo de negócio, durante a qual documentos e informação são trocados entres os participantes do processo.

Salienta-se que *workflow* é uma maneira de sistematizar qualquer atividade, deste modo possibilitará um maior controle na execução das tarefas, permitindo uma melhor qualidade no produto final.

Sob essas perspectivas, a união da técnica de teste de software com *workflow* torna uma atividade interessante, pois a junção dos mesmos traz uma melhor qualidade no produto de software final. Assim, *workflows* de teste modelam o processo de teste de software. Como exemplo de *workflow* de teste de software pode-se citar o *workflow* de teste do *RUP* (*Rational Unified Process*), Kruchten (2000).

Como pode se observar a atividade de teste é uma das atividades que colaboram para a garantia da qualidade do software. No entanto, não é freqüentemente praticada devido ao tempo e custo envolvidos. Nesse contexto, é importante que procedimentos e ferramentas estejam disponíveis para que os testadores possam realizar essa atividade com menos custo.

Sob estes aspectos, observou-se a necessidade de ferramentas que apóiam tanto a documentação quanto a utilização de *workflows* de teste, os quais têm como objetivo guiar as atividades de teste de maneira padronizada.

Este trabalho tem como principal objetivo apresentar a ferramenta desenvolvida, onde o principal objetivo da mesma e apoiar a documentação da aplicação de *workflow* de teste, mais especificamente no contexto de técnica de teste estrutural, possibilitando a sistematização da atividade de teste, no desenvolvimento de sistemas que utilizam banco de dados relacional e orientado a objetos (OO).

Essa ferramenta será acessada por meio da Internet, a fim de beneficiar um número maior de interessados. Para avaliar a ferramenta desenvolvida, foi realizado dois estudos de

caso um por Geraldi (2006) e o outro por Sarmento (2006), onde foi observado aspectos funcionais e de usabilidade.

Pode-se concluir que está ferramenta auxilia de forma efetiva e eficaz a documentação da atividade de teste de software, viabilizando e aprovando o desenvolvimento da mesma, onde esta poderá ser aperfeiçoada e ou melhorada em projetos futuros.

CAPITULO 1 – TESTE DE SOFTWARE

1.1 Considerações Iniciais

Neste capítulo será discutido a respeito de teste de software, pois é a sua aplicação que ajuda na determinação da qualidade que determinado software tem, sendo que quanto melhor elaborada e executada no sistema, maior será a probabilidade de qualidade do software. Deve-se destacar que será dada maior ênfase na técnica de teste de software estrutural por ser um dos temas relevantes para a confecção deste projeto, assim como o critério baseado em fluxo de dados.

O conteúdo deste capítulo está organizado da seguinte maneira. Na Seção 1.2 são contidas as definições de teste de software. Na Seção 1.3 é abordada a técnica de teste estrutural, enfocando na Seção 1.3.1 o critério baseado em fluxo de dados. Na Seção 1.4 é apresentada a Norma IEEE 829 que é uma norma para a documentação de teste de software, e por fim na Seção 1.5 são discutidas as considerações finais deste capítulo.

1.2 Conceitos de Teste de Software

Desenvolver uma aplicação de software com qualidade não é uma atividade trivial, isto é devido à possibilidade de erro do ser humano. Para que um software tenha boa qualidade o mesmo deve passar por várias etapas durante a sua confecção.

Segundo Crespo *et al.* (2000), teste é a verificação dinâmica do software, ou seja, é o processo de executar o software e comparar seus resultados com os requisitos especificados.

Uma das etapas mais importantes na confecção de um software é a aplicação de teste, a qual exige cerca de 30% a 40% do esforço da produção total de determinado software Pressman (1995).

De acordo com Crespo *et al.* (2000), os objetivos do teste é revelar a existência de erros no software, verificar se o software está funcionado de acordo com a especificação do sistema e fornecer uma avaliação da sua qualidade.

Segundo Pressman (1995), o intuito principal do teste de software é planejar testes que descubram de forma sistemática, diferentes classes de defeitos com tempo e esforços mínimos.

Para a aplicação da atividade de teste se faz necessário criar um plano de teste que contém quatro atividades: **planejamento**, **projeto de casos de teste**, **execução** e **avaliação dos resultados**, como apresentado na Figura 1.1. Estas atividades devem ser conduzidas ao longo de todo processo de confecção de software Rocha *et al.* (2001).

- **Planejamento:** o planejamento da atividade de teste deve fazer parte do planejamento global do sistema, culminando em um plano de teste que constitui um dos documentos cruciais no ciclo de vida de desenvolvimento de software. Nesse plano são estimados recursos e são definidos estratégias, métodos e técnicas de teste, caracterizando-se um critério de aceitação do software em desenvolvimento.
- **Projeto dos casos de teste:** um caso de teste é uma determinada entrada para o programa e sua correspondente saída esperada. Um bom caso de teste é aquele que tem alta probabilidade de revelar um defeito ainda não descoberto.
- **Execução do programa com os casos de teste:** nesta atividade executa-se o programa levando em conta a entrada dos casos de teste e verificando se a saída obtida é realmente a saída esperada.
- **Avaliação dos resultados:** nesta atividade verificam-se os resultados a fim de determinar se o software deverá ou não passar por outra fase de teste, com a finalidade de melhorar a sua qualidade.

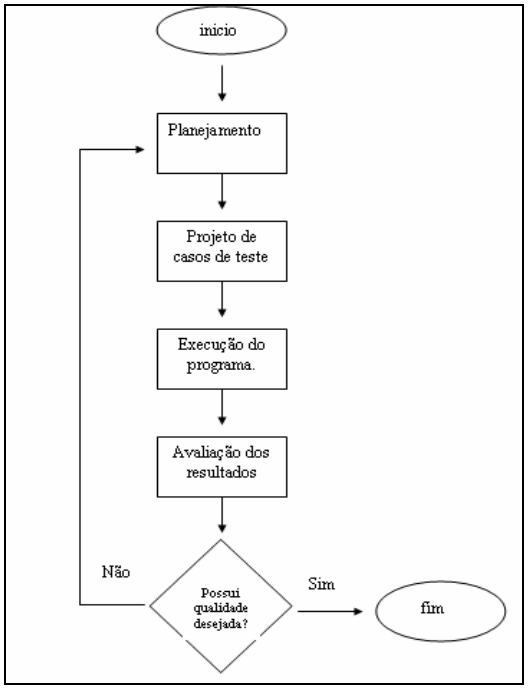

Figura 1.1 – Fluxograma da atividade de teste

A atividade de teste é dividida em três fases distintas: fase de teste de unidade, de integração e de sistema Rocha et al. (2001). Adicionalmente, tem-se a fase te teste de aceitação que é mencionada no RUP Kruchten, (2000). Estas fases estão descritas a seguir:

- **Fase de teste de unidade:** tem como principal objetivo identificar erros de lógica e de implementação, analisando-os por módulos separadamente. Uma unidade é um componente de software que não pode ser dividido.
- **Fase de teste de integração** caracteriza-se por identificar erros associados as interfaces, quando estas são integradas para formar o sistema .
- **Fase de teste de sistema:** tem como prioridade identificar erros relativos a funções e a características do sistema, comparando-os com a especificação do sistema.
- **Fase de teste de aceitação:** está diretamente ligada a aceitação e aprovação do software pelos usuários finais.

Com base nestas quatro fases pode-se minimizar a complexidade do software, abordando assim em cada fase um tipo de defeito e um determinado aspecto do software, de forma sistemática e eficaz.

Ainda com a utilização destas fases é possível desenvolver estratégias adequadas para a detecção e posterior correção de erros no sistema. Como resultado desta divisão em etapas, em geral, é possível ter um software com maior qualidade e confiabilidade.

Quanto às técnicas de teste de software, segundo Rocha *et al*. (2001), são classificadas de acordo com a origem das informações utilizadas e são denominadas como: técnica funcional, estrutural, baseada em erros e com base em máquinas de estado finito.

- **Técnica Funcional:** também chamada teste "caixa-preta", testa o sistema do ponto de vista do usuário, isto é, não considera a estrutura interna ou a forma de implementação do sistema. O teste funcional tem como principal característica avaliar o comportamento do sistema. Como principais critérios para esta técnica têm-se: particionamento de equivalência, análise do valor limite e grafo de causa-efeito.
- **Técnica estrutural:** também conhecida como teste "caixa-branca", procura exercitar todas as partes do código fonte de um sistema. Esta técnica será devidamente abordada na Seção 1.3. Como principais critérios para esta técnica têm-se: critério com base na complexidade, critério com base em fluxo de controle, critério com base em fluxo de dados. Este último será devidamente abordado na Seção 1.3.1.
- **Técnica baseada em erros:** tem por objetivo verificar se o software está livre de erros típicos e comuns cometidos pelo desenvolvedor durante o ciclo de desenvolvimento do software, especificadamente durante a fase de

implementação. Como principais critérios para esta técnica têm-se: semeadura de erros e análise de mutantes.

- **Com base em máquinas de estado finito:** tem como principal objetivo observar o estabelecimento de critérios de geração de seqüências de teste com base em máquinas de estados finitos, os quais têm sido aplicados no contexto de validação de software OO.

1.3 Técnica de Teste Estrutural

Segundo Spoto *et al*. (1995), a técnica de teste estrutural é baseada no conhecimento da estrutura interna do programa e tem como principal objetivo exercitar um conjunto de partes do software.

Como mencionado anteriormente, a técnica estrutural possui três critérios de teste: critério com base na complexidade, com base no fluxo de controle e com base no fluxo de dados; e estão comentados a seguir:

- **Critério com base na complexidade**: utiliza informações sobre a complexidade do programa para determinar os requisitos de teste, permitindo assim a medida quantitativa da complexidade lógica de um programa (Rocha *et al*., 2001).
- **Critério com base no fluxo de controle**: utiliza apenas as características de controle de execução do programa e tem como principal objetivo determinar quais estruturas (comandos, desvios) estão sendo utilizadas para que desta forma possa se definir quais estruturas poderão ser eliminadas. Este critério pode ser subdividido em outros sub-critérios (Rocha *et al*., 2001): Todos-Nós, Todos-Arcos, Todos-Caminhos, os quais de acordo com Spoto *et al*. (1995) são caracterizados da seguinte forma: **Todos-Nós** requer a execução de todos os nós

executáveis, por outro lado o critério **Todos-Arcos** requer a execução de todos os ramos executáveis e **Todos-Caminhos** requer a execução de todos os caminhos executáveis.

- **Critério com base no fluxo de dados**: este critério merece uma abordagem mais ampla e sistemática, pois é o assunto alvo, no qual o projeto tem sua base, portanto será devidamente abordado na Seção 1.3.1.

A maioria dos critérios da técnica estrutural utiliza uma representação do programa (código fonte), denominada Grafo de Fluxo de Controle (GFC), que é um grafo orientado, com um único nó de entrada e um único nó de saída, de modo que cada nó corresponde a uma seqüência de comandos que são sempre executados de forma única e cada arco representa uma transferência de controle entre os blocos.

No entanto, para que o GFC possa ser gerado, são necessários alguns conceitos que auxiliam na sua confecção: bloco, nó, ramo ou arco, nó de entrada, nó de saída, caminho simples e caminho completo (Spoto *et al*., 1995), descritos a seguir:

- **Bloco:** seqüência máxima de comandos, em que se o primeiro comando for executado os demais também deverão ser executados;
- **Nó:** é a representação gráfica de um bloco;
- **Ramo ou arco**: é a representação gráfica de uma possível transferência de controle entre determinados blocos;
- **Nó de entrada**: este é o nó que contém o primeiro comando do programa;
- **Nó de saída**: este é o nó que contém o último comando do programa;
- **Caminho Simples**: é aquele que todos os nós, exceto o primeiro e o último, são distintos;

- **Caminho Completo**: é aquele que o nó início é o nó de entrada e o nó final é o nó de saída.

Na Figura 1.2 apresentam-se algumas notações que são utilizadas para descrever o fluxo de controle lógico, que será utilizado para a confecção do GFC.

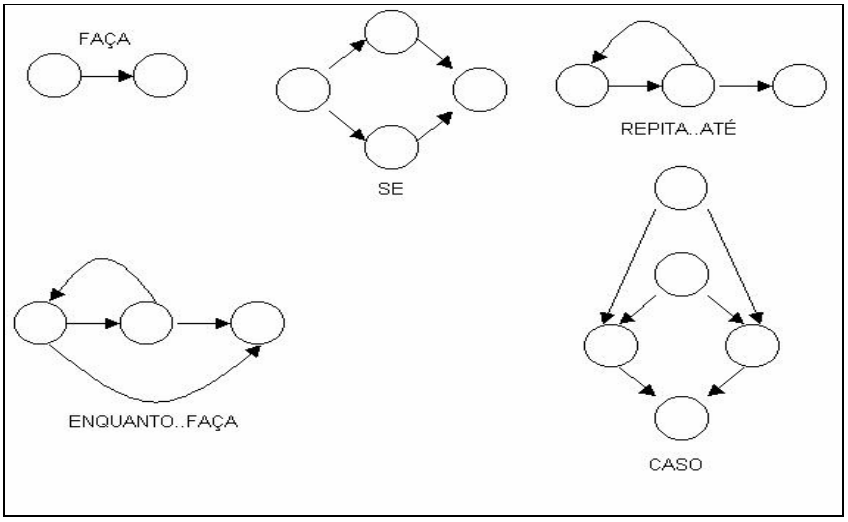

Figura – 1.2 Notação de grafo de fluxo (Pressman, 1995)

Para se criar o GFC deve-se primeiramente dividir o código fonte em blocos. Depois de definidos os blocos, devem-se estabelecer qual o nó de entrada e o nó de saída. Posteriormente, devem-se estabelecer as possíveis transferências de controle as quais representarão os ramos ou arcos.

Na Figura 1.3 (a) está representado um programa fonte o qual está dividido em blocos. Pode-se observar no código fonte que cada linha de código vem precedida de um número, representando o bloco que pertence. Enquanto que na Figura 1.3 (b) está representado o GFC, em que se pode observar que cada nó contém o número do respectivo bloco que representa. Com base no GFC são identificados os componentes (nós, arcos ou

caminhos) que devem ser executados para satisfazer determinado critério, caracterizando assim o teste estrutural.

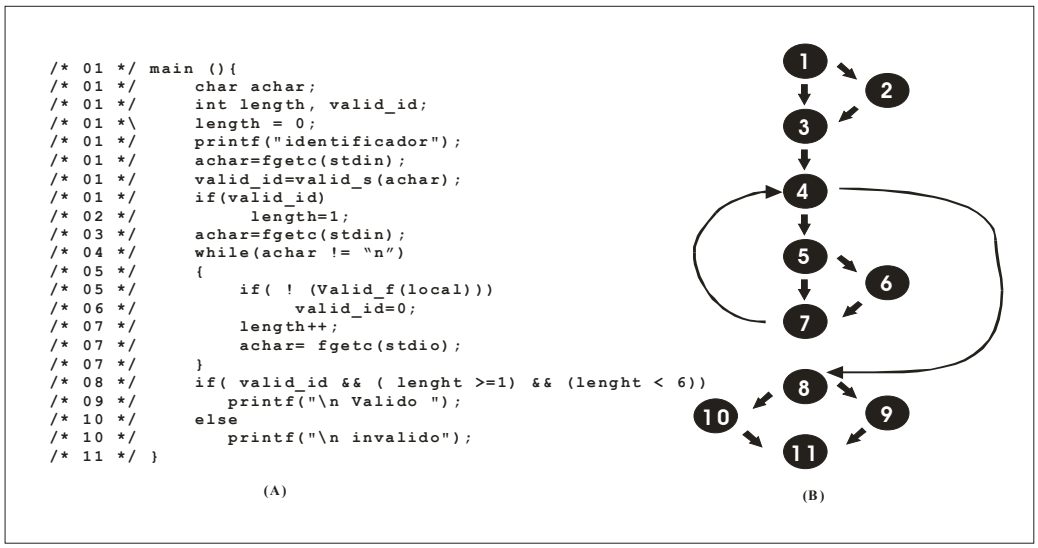

Figura 1.3 - Grafo de fluxo de controle (Maldonado *et al.*, 2001)

É importante salientar que o GFC será amplamente utilizado na Seção 1.3.1, em que é esboçado o critério baseado em fluxo de dados da técnica de teste estrutural.

1.3.1. Critério Baseado em Fluxo de Dados

O critério de teste baseado em fluxo de dados é um critério que faz parte da técnica de teste estrutural e tem como principal objetivo, segundo Pressman (1995), selecionar caminhos de teste de um programa de acordo com as localizações das definições e usos de variáveis no programa.

Segundo Maldonado *et al.* (2004), uma característica comum do critério baseado em fluxo de dados é que este requer que sejam testadas as iterações que envolvam as definições da variável e suas subseqüentes referências a essas definições.

O grafo def/uso do critério baseado em fluxo de dados é uma extensão do grafo de fluxo de controle apresentado na Seção 1.3. Neste grafo são adicionadas informações relacionadas a definição e uso das variáveis em determinado programa.

Como se pode observar na Figura 1.4, o grafo def/uso é o mesmo apresentado na Figura 1.3 (b) mas sofre uma adição de informações a respeito das variáveis. Estas informações estão relacionadas quanto a definição das variáveis bem como quanto ao seu respectivo uso.

A definição de uma variável refere-se a uma instrução em que o valor é atribuído a uma variável. Já o uso ocorre quando os valores das variáveis são utilizados. O uso destas variáveis é classificado, segundo Gonçalves (2003), da seguinte maneira:

- **C-Uso (uso computacional):** quando uma variável é usada na avaliação de uma expressão ou em um comando de saída, o uso está associado ao nó.
- **P-Uso (Uso predicativo):** quando a variável ocorre em um predicado, ou seja, quando os valores das variáveis são utilizados como condições de desvios (não somente em IF-THEN-ELSE mas em LOOPS também) (Lemos, 2005), então está diretamente relacionada ao fluxo de controle do programa, ou seja está associada ao arco.

Quando as variáveis são definidas em um determinado bloco, então no grafo de fluxo de controle estas são marcadas da seguinte maneira, *d={nome das variáveis}*. Por outro lado, quando as variáveis são utilizadas, estas podem ser classificadas como uso predicativo ou uso computacional.

Assim, quando as variáveis são utilizadas de forma predicativa estas devem ser adicionadas no grafo de fluxo de controle da seguinte maneira: *up={nome das variáveis}*. Se as variáveis são utilizadas de forma computacional então ocorre o mesmo que para o uso predicativo, mudando somente a nomenclatura *up={nome das variáveis}* para *uc={nome das variáveis},* como mostrado na Figura 1.4.

O grafo def/uso representado na Figura 1.4 mostra os pontos que exibem definição e uso de variáveis. Com base neste grafo são determinados quais caminhos devem ou não ser exercitados para atender os critérios baseados em fluxo de dados.

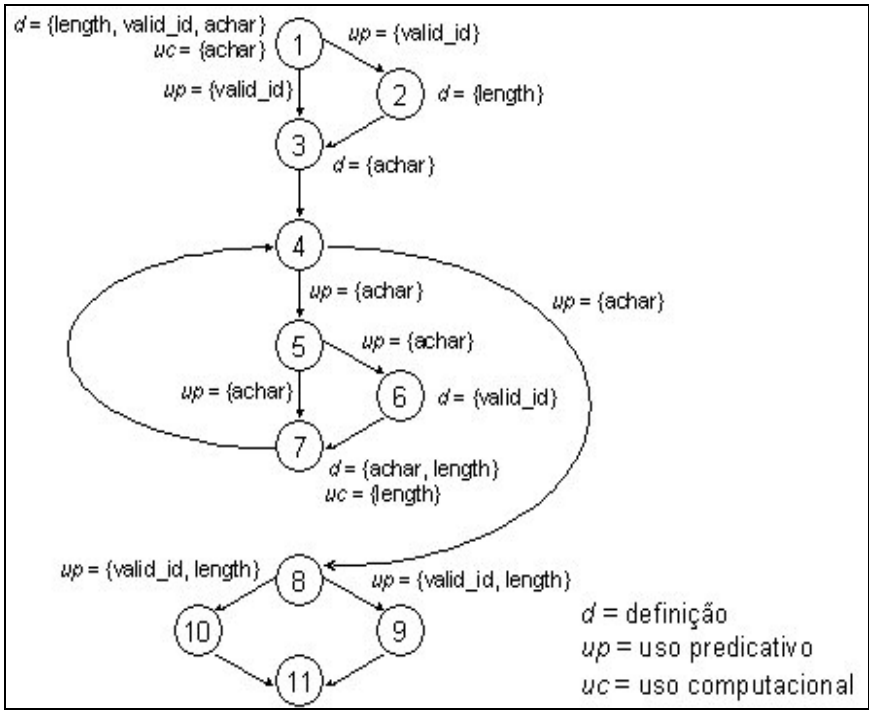

Figura 1.4 - Grafo Def/Uso (Maldonado *et al*., 2004)

O critério mais básico dos critérios baseados em fluxo de dados é o critério Todas-Definições (*all-defs*), onde desta família o critério mais utilizado é o critério Todos-Usos (*all uses*) (Maldonado *et al*., 2004) e estão descritos a seguir:

- **Todas-Definições:** exige que todas as definições de variáveis sejam exercitadas independente se por um *p-uso* ou *c-uso*.
- **Todos-Usos:** requer que todas as definições de variáveis e seus respectivos usos sejam exercitados pelos casos de teste por um caminho onde esta variável não sofre redefinição.

A maior parte dos critérios baseados em fluxo de dados não garante que o critério Todos-Ramos seja satisfeito, o qual está presente ma maioria dos programas, apenas exige que a variável seja executada de forma explícita.

1.4 Norma IEEE 829

Nesta seção é abordada a documentação para as atividades de teste de acordo com a Norma IEEE (*Institute of Electrical and Electronic Engineers*) 829, que é definida de forma independente de técnicas, métodos, estratégias, recursos e ferramentas de teste de software e descreve um conjunto de documentos (8 documentos) que podem ser produzidos durante as atividades de teste: plano de teste, especificação de projeto de teste, especificação de casos de teste, especificação de procedimento de teste, diário de teste, relatório de incidente de teste, relatório-resumo de teste e relatório de encaminhamento de item de teste os quais são definidos pela norma, que tem como finalidade cobrir as tarefas de planejamento, especificação e relatório de testes.

De acordo com a Norma IEEE 829, este conjunto de documentos pode ser utilizado para qualquer tipo de software, desde software cientifico até software de uso militar.

Para cobrir a tarefa de planejamento de teste é descrito apenas um documento, denominado plano de teste**,** segundo Crespo *et al.* (2000), este documento tem como finalidade apresentar o planejamento para a execução do teste incluindo a abrangência, recursos e cronograma das atividades de teste. Identifica os itens e as funcionalidades a serem testados, as tarefas de teste a serem realizadas e os seus respectivos riscos.

Para a tarefa de especificação são propostos três documentos: especificação de projeto de teste, especificação de casos de teste e especificação de procedimentos de teste, descritos a seguir (Crespo *et al.,* 2000):

- **Especificação de projeto de teste:** tem como prioridade refinar o documento criado na tarefa de planejamento de teste e identificar as funcionalidades e características a serem testadas. Este documento também apresenta os critérios de aprovação e, se existirem, identifica os casos e os procedimentos de teste;
- **Especificação de casos de teste:** tem com objetivo descrever determinado caso teste, incluindo os dados de entrada, os resultados esperados, as ações e condições gerais para a execução de teste. A separação deste documento do anterior deve-se ao fato de que deste modo permite a reutilização nos testes de outros projetos. Como exemplo, pode se citar a especificação de uma rotina de validação de CPF, que pode ser utilizada para todos os produtos de software que utilizam ou possuem esta funcionalidade;
- **Especificação de procedimento de teste:** especifica os passos para executar um conjunto de casos de teste. A separação deste documento em relação a especificação de caso de teste deve-se ao fato de que um procedimento de teste possui somente passos simples e não deve conter outros detalhes.

A tarefa de relatório de testes é coberta por quatro documentos: diário de teste, relatório de incidente de teste, relatório-resumo de teste e relatório de encaminhamento de item de teste, comentados a seguir Crespo *et al.* (2000):

- **Diário de teste:** tem como finalidade apresentar os registros cronológicos dos detalhes importantes relacionados com a execução dos testes;
- **Relatório de incidente de teste:** tem como objetivo documentar qualquer evento que ocorra durante a atividade de teste e que requer análise posterior;
- **Relatório-resumo de teste:** apresenta de forma simples e objetiva os resultados das atividades de teste associadas com uma ou mais especificações de projeto de teste e provê avaliações baseadas nestes resultados;

- **Relatório de encaminhamento de item de teste:** tem como principal objetivo identificar os itens que deverão ser encaminhados para teste, isto no caso de as equipes responsáveis pelo desenvolvimento e pelo teste serem distintas.

Deve-se salientar que mesmo que a norma possa ser aplicada para software de qualquer tamanho ou complexidade, projetos pequenos ou de pouca complexidade podem agrupar alguns documentos propostos, de modo a diminuir o gerenciamento e o custo de produção.

Na Figura 1.5 ilustra-se o relacionamento entre os documentos descritos anteriormente. Deve-se salientar que os itens que estão em cinza não foram descritos na norma.

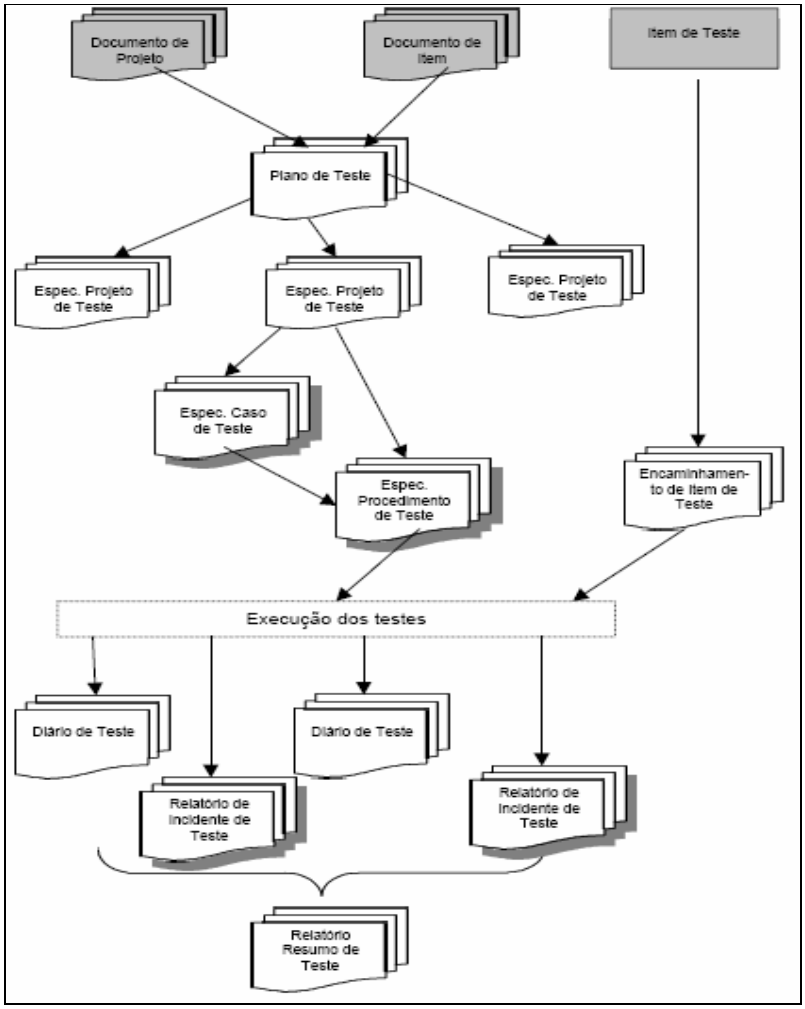

Figura 1.5 – Relacionamento entre os documentos de teste (Crespo *et al.*, 2000)

1.5 Considerações Finais

Neste capítulo apresentou-se o conteúdo de teste de software, com suas definições, usos, características e sua importância na confecção de software.

Deve-se observar, que a técnica de teste estrutural e o critério baseado em fluxo de dados têm uma abrangência maior se comparados as outras técnicas e critérios. Isto ocorreu devido a importância destes para este trabalho, pois a ferramenta que será desenvolvida abordará o cadastro e o uso de *workflow* de teste neste contexto.

CAPÍTULO 2 - *WORKFLOW*

2.1. Considerações Iniciais

Uma das maneiras de sistematizar as atividades de teste é por meio de *workflows*. Para que se tenha um melhor entendimento do assunto, neste capítulo apresentam-se as definições, conceitos e um modelo existente de *workflow*.

2.2. Conceitos de *Workflow*

A engenharia de software vem centralizando seus objetivos na melhoria da qualidade de software, bem como na redução dos esforços, custos e na reutilização de especificações e processos. A reutilização nesse contexto é uma técnica de construção de software a partir de processos e modelos previamente definidos (Silva, 2001).

Workflow pode ser definido como um conjunto coordenado de atividades interligadas e com um objetivo em comum. Na prática, descreve uma sucessão de etapas necessárias para processar documentos e formulários, enquanto estes tramitam em uma organização. Essas atividades podem ser executadas por um ou mais sistemas de computador, por um ou mais agentes humanos, ou então por uma combinação destes, Matos (2004).

A ordem de execução e as condições pelas qual cada tarefa é iniciada também estão definidas no *workflow*, sendo que o mesmo é capaz ainda de representar a sincronização das tarefas e o fluxo de informações (Moro, 1998).

A utilização de um *workflow* proporciona vários benefícios, entre eles podemos destacar a simplificação dos formulários previstos, o acesso remoto, o arquivamento e a recuperação de informações simplificadas, a habilidade de rapidamente trilhar as informações

submetidas, a possibilidade de saber os responsáveis de cada tarefa do processo e o aumento no tempo de linhas de informação, Moro (1998).

Segundo Matos (2004), na execução do *workflow* pode haver três tipos de agentes (atores): executor, responsável e executor da atividade; descritos a seguir:

- **Executor:** ator que inicia o *workflow*.
- **Responsável:** ator que tem responsabilidade sobre o *workflow*.
- **Executor da atividade:** ator responsável pela execução de uma determinada atividade.

Para se representar um processo é necessária a adoção de um modelo de *workflow,* de forma que todos os elementos de tal processo sejam descritos, Silva (2001).

Em função do tipo de atividade que um *workflow* modele, ele pode ser classificado da seguinte forma, Matos (2004):

- ***Workflow ad hoc*:** as tarefas são simples e não automatizadas, geralmente são criadas para controlar o fluxo de um único documento em uma única ocasião;
- ***Workflow* administrativo:** as tarefas são pouco estruturadas e repetitivas, a coordenação das tarefas pode ser fracamente automatizada, geralmente através de correio eletrônico;
- ***Workflow* de produção:** as tarefas são mais estruturadas, requisitando coordenação mais complexa e automatizada com pouca interferência humana.

Os modelos de *workflow* podem ser divididos em dois grupos, Matos (2004):

- **Modelos baseados em comunicação**: o trabalho é considerado um conjunto de interações humanas bem definidas, representando compromissos realizados entre as pessoas envolvidas; e

- **Modelos baseados em atividades**: o trabalho é composto por uma seqüência de atividades, sendo que cada atividade recebe entradas e produzem suas respectivas saídas.

Um dos maiores problemas da modelagem de sistemas de *workflow* é que praticamente cada sistema de gerenciamento de *workflow* utiliza sua própria técnica de modelagem, ou seja, não há um modelo aceito de forma unânime para a modelagem de *workflow*.

Com o intuito de resolver este impasse, a WfMC propôs um modelo de referência com o intuito de estabelecer padrões na área de *workflow*. Neste modelo são definidos cinco componentes, que comunicam-se entre si por meio de interfaces, como apresentado na Figura 2.1:

- Componente de definição de processo: especifica um padrão para as interfaces entre as ferramentas e os serviços de execução de *workflow*;
- Componente *workflow* da aplicação cliente: tem como objetivo definir padrões de execução do *workflow* para que as tarefas que os *workflows* oferecem ao usuário sejam mantidas;
- Componentes aplicações invocadas: tem como finalidade criar um padrão de interface, com o objetivo de permitir que o *workflow* em execução utilize várias aplicações. Como exemplo pode-se citar correio eletrônico e serviço de fax;
- Componentes outros serviços de *workflow*: tem como característica definir vários modelos que devem interagir com produtos de fabricantes seguindo o padrão de cada fabricante.
- Componentes ferramentas de monitoramento e de administração: tem como tarefa gerar um padrão de funções para monitoramento e controle do *workflow*.

Segundo Matos (2004), o modelo de referência permite que o serviço seja construído de forma distinta desde que sejam oferecidas interfaces que provocam a transparência entre os métodos do *workflow* com os métodos padronizados pela WfMC.

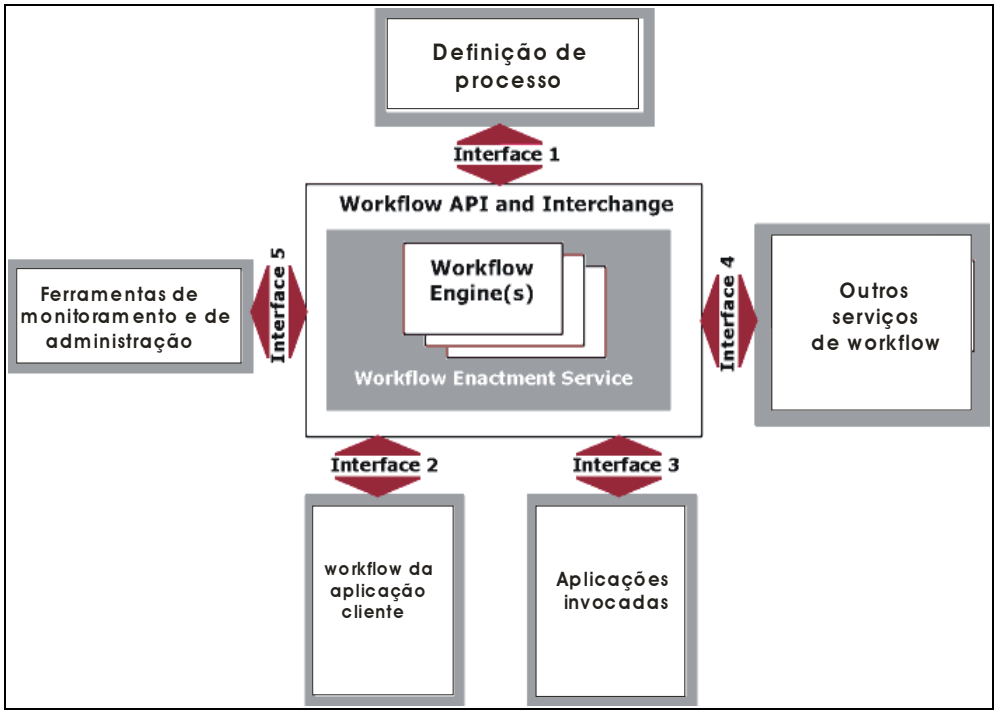

Figura 2.1 – Modelo de referência WfMC (adaptado de WMC, 1995)

Em síntese, um sistema de *workflow* deve representar o fluxo de atividades dos processos de forma a facilitar a alocação e o controle de execução das atividades, oferecendo recursos para a alocação de atividades aos atores, baseando-se no comportamento dinâmico dos processos, Matos (2004).

2.3. *Workflow* de Teste de Software

Workflow de teste pode ser definido como um conjunto de processos coordenados com o intuito de apoiar e controlar a aplicação de teste de software de maneira eficiente e sistemática (Matos, 2004).

O teste de software tem sido um ponto decisivo no nível de qualidade do software. Sob esta perspectiva é conveniente a criação de um *workflow* de teste não só para apoiar a atividade de teste, mas também para controlar e validá-la.

Devido a importância de *workflow* de teste para este trabalho, na Seção 2.3.1 é abordado o *workflow* de teste do RUP, descrevendo suas definições, aplicações e suas respectivas tarefas.

2.3.1. *Workflow* de Teste do RUP

Rational Unified Process (RUP), Kruchten (2000) é um processo completo de projeto de software criado pela *Rational Software Corporation* com o intuito de viabilizar que grandes projetos de software sejam bem sucedidos. Os modelos de desenvolvimento de software descritos pelo RUP são técnicas já testadas por várias instituições desenvolvedoras de software.

Deve-se salientar que o RUP é um processo robusto e devido a este fato é mais aplicável a grandes equipes de desenvolvimento onde a dificuldade de gerenciamento é maior.

Segundo Matos (2004), o RUP é um processo analítico, incremental, baseado em ciclos, sendo que após cada ciclo deverá haver uma versão de software. Para que uma versão do software seja produzida, o processo deverá passar por um ciclo, onde cada ciclo deve passar por quatro fases:

- Concepção: entendimento da necessidade e visão do projeto;

- Elaboração: especificação e abordagem dos pontos de maior risco;
- Construção: está diretamente ligada ao desenvolvimento do sistema;
- Transição: ajustes, implantação e transferência de propriedade do sistema.

Como o principal objetivo do RUP é promover o teste do produto de forma gerenciada ou controlada, o *workflow* de teste do RUP utiliza basicamente duas técnicas (Matos, 2004):

- **Plano de testes:** caracteriza-se pelo planejamento de todo o processo de teste de software, incluindo análise de riscos e definição de responsabilidades;
- **Casos de teste:** constituído pelo conjunto de casos de teste desenvolvido para determinada atividade de teste, contendo resultados esperados e condições de execução.

No RUP a atividade de teste pode ser vista em três visões ou dimensões diferentes, Matos (2004):

- **Qualidade:** devem ser assegurados os aspectos relacionados a qualidade, confiabilidade e desempenho do sistema;
- **Estágio de teste:** está relacionado com os diferentes estágios de teste de software, os quais são: teste de unidade, teste de integração, teste de sistema e teste de aceitação; descritos na Seção 1.2.
- **Tipo de teste:** está diretamente ligado aos tipos de teste, sendo que os mais comuns são, Matos (2004):
  - ✓ Teste de *benchmark*: compara os objetivos do teste com padrões conhecidos;
  - ✓ Teste de integridade: verifica a confiabilidade, robustez e tolerância a falhas;
  - ✓ Teste de desempenho: testa a desempenho do teste em diferentes configurações;
  - ✓ Teste de *stress*: aplica o teste de desempenho em condições anormais ou extremas.

Na Figura 2.2 é mostrado o *workflow* de teste do RUP a partir de um diagrama de atividade UML (*Unified Modeling Language*), Matos (2004), contendo a seqüência de tarefas do *workflow* de teste do RUP. Nesta representação, pode-se perceber as tarefas pelas quais determinado teste deve passar. Salienta-se que esta não é a representação ideal, pois não apresenta os atores nem a forma de interação entre as tarefas.

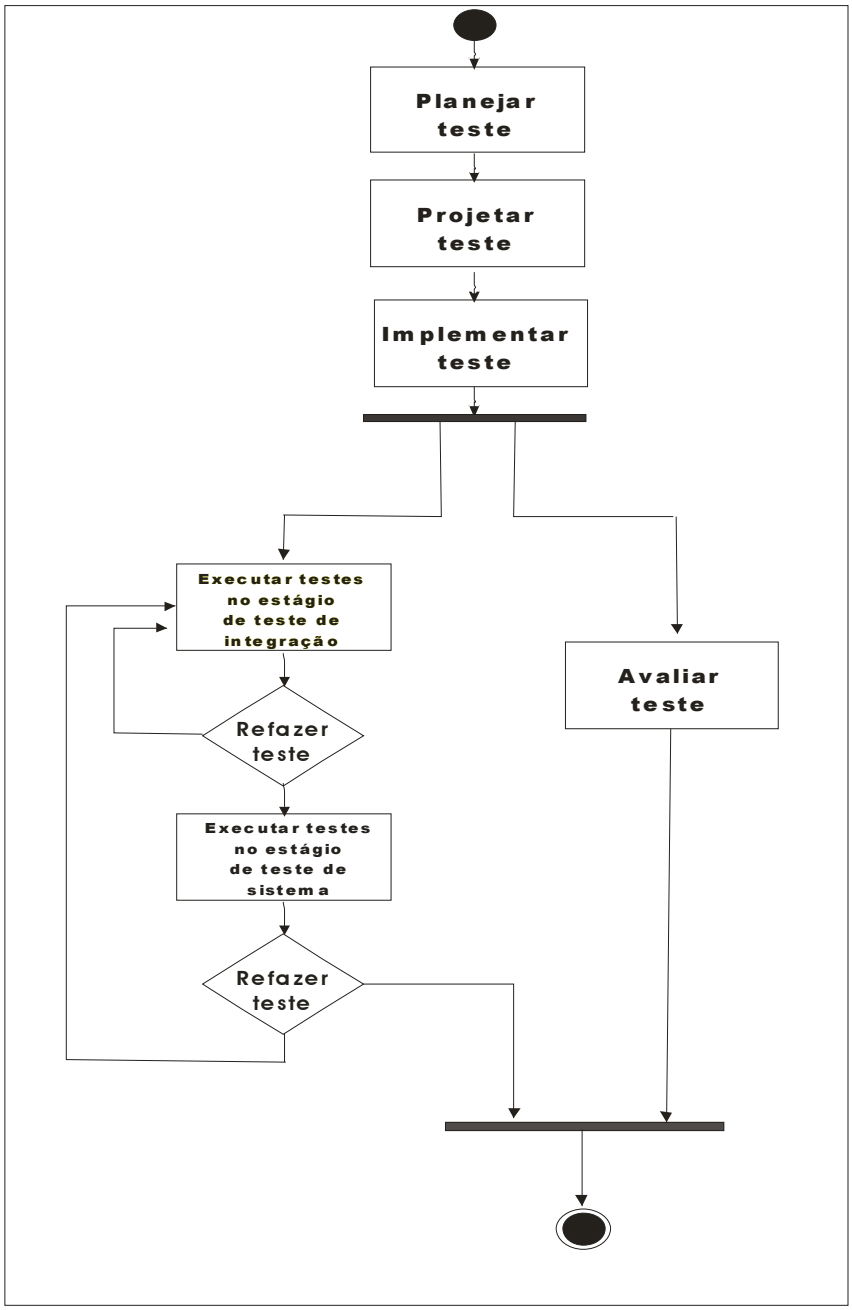

Figura 2.2 – *Workflow* de teste do RUP (adaptado de Kruchten, 2000)

A Figura 2.2 é constituída por seis tarefas, descritas a seguir Kruchten (2000):

- **Planejar os testes:** seu propósito é descrever a implementação e execução do teste. No plano de teste deve conter os requisitos para o teste e suas respectivas estratégias. Um simples plano de teste deve conter a descrição dos tipos de teste, sua implementação e execução;
- **Projetar os testes:** tem como objetivo identificar, descrever e gerar o modelo de teste e os relatórios;
- **Implementar os testes:** tem como principal característica criar procedimentos para implementar o teste de software, seguindo o que foi definido na tarefa anterior (projetar os testes).
- **Executar testes na fase de teste de integração:** seu principal objetivo é executar o teste de integração.
- **Executar testes na fase de teste de sistema:** tem como prioridade efetuar teste completo do sistema.
- **Avaliar os testes:** tem como propósito avaliar o sistema quanto à confiabilidade e qualidade, determinando a qualidade do software.

Deve-se salientar que as tarefas, **executar testes na fase de teste de sistema**, **executar testes na fase de teste de integração** e **avaliar os testes** ocorrerão de forma concorrente ou paralela.

2.4. Considerações Finais

Após a apresentação destes modelos pode-se concluir que *workflows* é uma opção para sistematizar qualquer operação, a fim de obter um maior controle e uma melhor qualidade do sistema a ser desenvolvido.

Uma característica que deve ser ressaltada a respeito de *workflows* é que os mesmos possibilitam que atividades sejam transferidas de uma pessoa para outra sem que haja perda significativa de tempo e de custo no desenvolvimento de software.

CAPITULO 3 – FADAT FERRAMENTA DE APOIO A DOCUMENTAÇÃO DA APLICAÇÃO DE TESTE DE SOFTWARE

3.1. Considerações iniciais

A atividade de teste de software é uma das atividades que colaboram para garantia da qualidade do software, deve-se salientar que a mesma não é uma atividade trivial, isto devido a vários agravantes, dentre os quais serão apresentados apenas três que são os principais fatores que colaboraram para o desenvolvimento deste trabalho:

- **Tempo:** O tempo torna-se um agravante, pois, segundo (Pressmam,1995), a aplicação de teste de software consome cerca de 30% a 40% do esforço de produção total de determinado software;
- **Custo:** O custo para a realização da atividade de teste de software é alto;
- **Ferramentas:** Após uma breve pesquisa constatou-se a escassez de *workflows* que auxilia tanto na elaboração da documentação quanto na aplicação de teste de software.

Para que a atividade de teste de software seja realizada num menor intervalo de tempo, com maior qualidade e menor custo, observou-se a necessidade de que procedimentos e ferramentas estejam disponíveis para os testadores.

Workflow por sua vez é uma maneira de se sistematizar atividades, ou seja uma ou várias atividades organizadas que tem por objetivo a realização de uma atividade.

Sob estas perspectivas surgiu o interesse em desenvolver uma ferramenta para documentar a aplicação de *workflows* de teste, a qual tem como principal objetivo facilitar a atividade de documentação de teste de software para os testadores.

3.2. Modelo Conceitual

O Diagrama de Classe apresentado na Figura 3.1 contem as classes do modelo da ferramenta FADAT (Ferramenta de Apoio a Documentação da Aplicação de Teste de Software), deve-se salientar que todas as classes foram mapeadas para o SGBD (Sistema Gerenciador de Banco de Dados) *FireBird*SQL (2005), com exceção das classes *Proprietário, Simples e Administrador*.

A classe *usuário* é responsável por armazenar todos os dados pertinentes ao usuário, deve se salientar que em virtude dos usuários exercerem diferentes papeis na ferramenta, mas terem os mesmos atributos o controle referente a qual classe de usuário um determinado usuário pertence se fará pelo atributo *permissão_user* da classe *usuário.*

A classe *workflow* por sua vez controlará todos os dados relevantes a um projeto de *workflow*, podendo ser criado um novo projeto de *workflow*, atualizado, consultado e removido, este *workflow* estará associado a um usuário que será o usuário proprietário do *workflow.*

Cada *workflow* poderá ser associada a uma ou varias atividades, onde a classe *atividade* será responsável por controlar os dados referentes a cada atividade, deve-se salientar que as atividades serão padronizadas para todos os projetos de *workflow.*

O usuário proprietário ficara incumbido de definir qual o executor para cada uma das atividades que estão representadas na Figura 3.2. Deve se salientar que cada atividade só poderá ser executada pelo executor cadastrado para cada atividade.

Para cada atividade deverá haver um executor, para controlar quem é o executor para determinada atividade criou-se a classe *executor*, que será responsável por controlar os dados pertencentes a cada executor, deve-se salientar que o executor deve estar devidamente cadastrado na ferramenta.

Uma atividade é auxiliada por uma ferramenta de apoio classe *ferramenta_apoio,* onde a mesma controlará os dados referentes a uma ferramenta de apoio a qual poderá auxiliar na determinada atividade de teste. Associada a classe *ferramenta_apoio* esta a classe *etapa_teste,* onde esta classe terá o objetivo de controlar quais as etapas de teste que está ferramenta cobre, como exemplo pode-se citar etapa de integridade, sistema e unidade.

Uma atividade atende a uma técnica de teste, classe *técnica_teste*, está classe será responsável por armazenar qual a técnica de teste que determinada atividade atende pode se citar a técnica estrutural, técnica funcional e técnica baseada em erros. A classe *técnica_teste* possui um critério de teste, classe *critério_teste,* que será responsável por controlar quais os critérios de teste que determinada técnica possui.

Durante a execução das atividades serão produzidos artefatos, estes poderão ser artefatos de entrada ou de saída, onde artefatos de saída correspondem a artefatos gerados pela iteração do usuário na determinada atividade, por outro lado artefatos de entrada são documentos gerados por atividades anteriores.

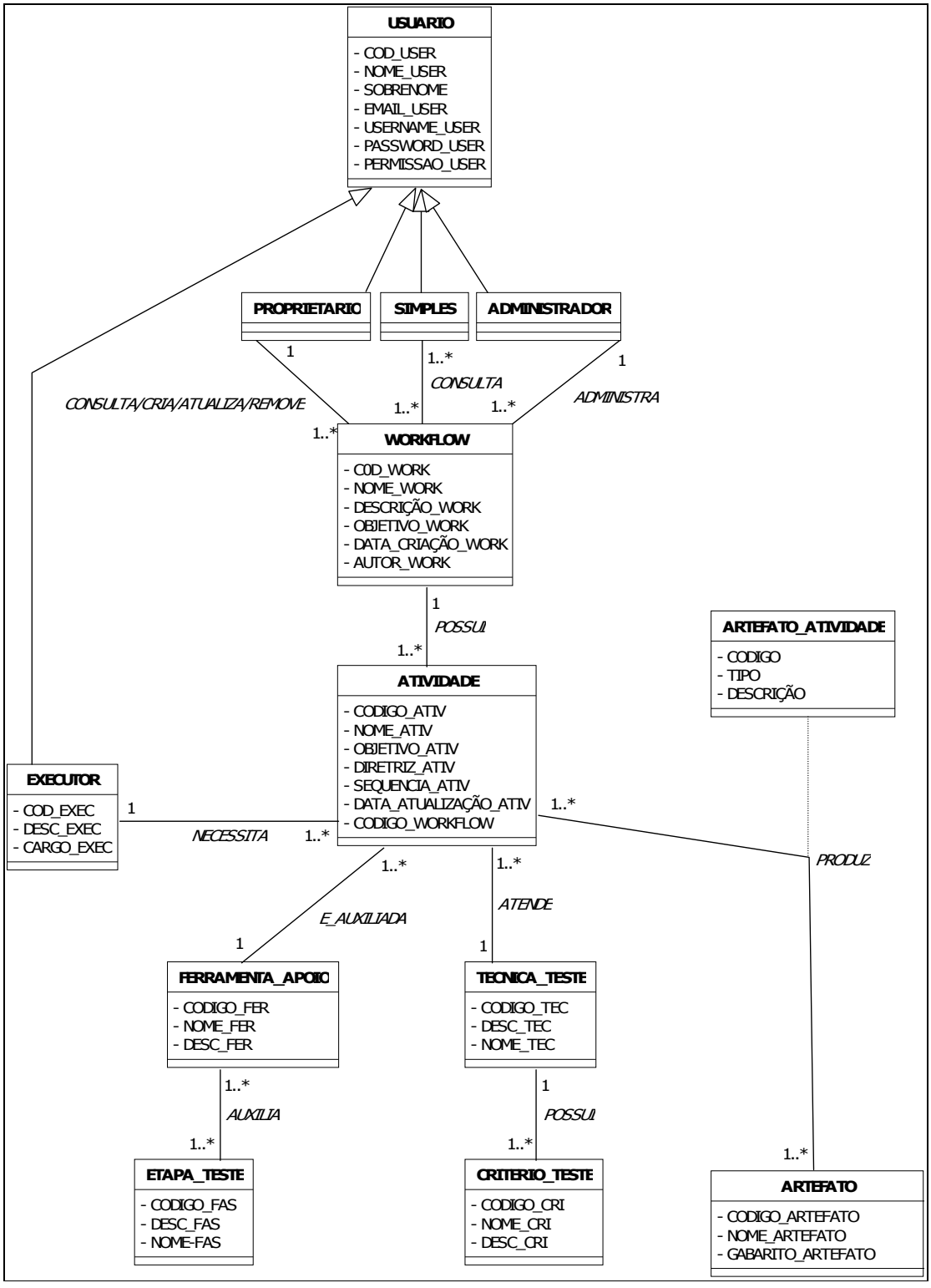

Figura 3.1 – Diagrama de Classe da FADAT – Ferramenta de Apoio a Documentação da Aplicação de Teste de Software

Na Figura 3.2 está representado o *workflow* criado e adotado para controlar as atividades da ferramenta supracitada, este *workflow* foi adaptado a partir do workflow de testes do *Rational Unified Process* (RUP), Kruchten (2000).

Como se pode observar na Figura 3.2, o *workflow* adotado para a ferramenta é composto de cinco atividades as quais são: *planejar teste, projetar teste, implementar teste, executar teste(s)* e *avaliar teste,* deve-se salientar que as atividades de *executar teste(s)* e *avaliar teste* podem ocorrer de forma paralela.

Outro ponto que deve ser enfatizado é de que ligado a determinada atividade de teste(s) poderá haver mais de um documento relacionado para a mesma atividade.

Caso a atividade *executar teste* tenha que ser refeita, deve se salientar que a versão criada anteriormente não deve ser removida, mas sim deve ser criada uma nova versão para está atividade.

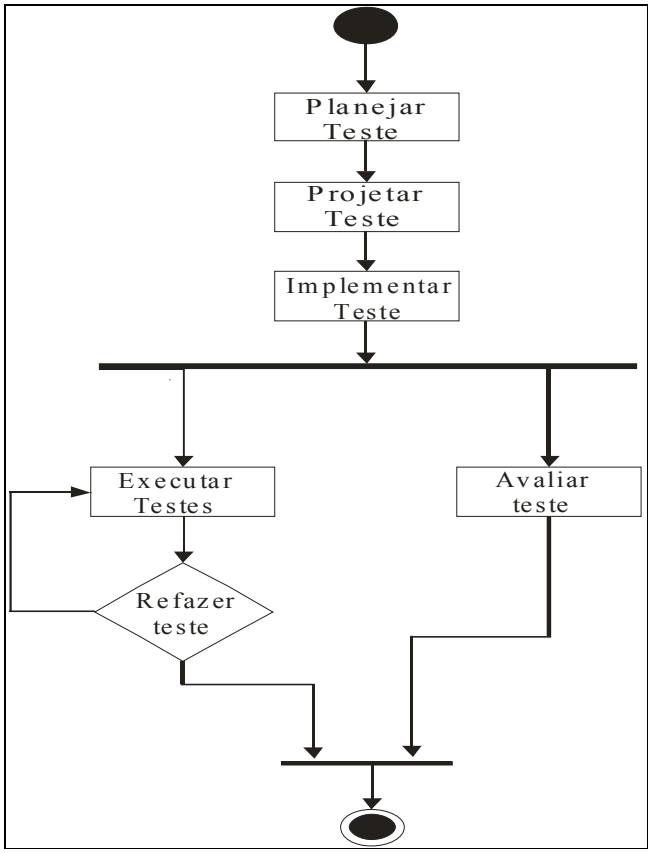

Figura 3.2 – *Workflow* de teste – adaptado de Kruchten (2000)

Após breve analise do modelo conceitual da ferramenta FADAT Figura 3.1 e breve estudo da documentação proposta pela Norma IEEE 829 descrita no Item 1.4 do Capitulo 1, identificou-se as seguintes funcionalidades para a implementação desta ferramenta:

1. **Login na ferramenta:** esta funcionalidade tem por objetivo liberar o acesso a visualização e cadastros diversos, na ferramenta, isto devera ocorrer após informar-se um nome de usuário e uma respectiva senha;
2. **Gerenciamento de usuários:** tem como principal objetivo controlar o gerenciamento de usuário na ferramenta as funções diretamente ligadas a esta funcionalidade são: o cadastro consulta, alteração e remoção de usuários da ferramenta;
3. **Gerenciamento de *workflow*:** esta funcionalidade tem como principais funções o cadastro, consulta, alteração e remoção de projetos de *workflows* na ferramenta e seu principal objetivo é controlar os dados referentes a *workflows* ;
4. **Gerenciamento de atividades:** tem como principal objetivo gerenciar as atividades referentes a um determinado *workflow,* as funções referentes a esta funcionalidade pode-se citar a cadastro, consulta de atividades;
5. **Gerenciamento de executores:** tem como principal objetivo determinar um executor para cada atividade referentes a um determinado projeto de *workflow*. Esta funcionalidade deve permitir a inclusão, consulta e alteração de executores, deve-se salientar que o executor deve estar devidamente cadastrado na ferramenta;
6. **Gerenciamento de Ferramentas de Apoio:** esta tem como principal objetivo o controle de ferramentas de apoio às quais podem ser utilizadas para uma determinada atividade de teste de software, esta funcionalidade é complementada pela funcionalidade *gerenciamento de fase de teste*;

7. **Gerenciamento de etapas de teste:** o principal objetivo desta funcionalidade é complementar a funcionalidade de *gerenciamento de ferramentas de apoio,* esta funcionalidade ira gerenciar quais as etapas de teste que uma ferramenta de apoio cobre;
8. **Gerenciamento de técnicas de teste:** o principal intuito desta funcionalidade e gerenciar quais as técnicas de teste que uma determinada atividade ira atender, esta funcionalidade é complementada pela funcionalidade *gerenciamento de critérios de teste*;
9. **Gerenciamento de critérios de teste:** esta funcionalidade complementa a funcionalidade *gerenciamento de técnicas de teste*, esta funcionalidade tem como principal objetivo definir quais os critérios de teste que determinada técnica possui;
10. **Gerenciamento de artefatos:** tem como principal objetivo definir os artefatos que serão gerados e utilizados durante a execução das atividades. Deve permitir o *download* do arteg=fato gerado pela ferramenta;
11. **Consultas Diversas:** está funcionalidade deve permitir que o usuário realize as consultas referentes as funcionalidades descritas anteriormente.

Como se pode observar na Figura 3.1., foram criados quatro tipos de usuário, *usuário simples, usuário proprietário do workflow e usuário administrador do sistema*, e *usuário executor de aatividade,* cada usuário terá permissão e ou acesso liberado a determinadas funcionalidades, as quais estão descritas anteriormente e representadas respectivamente nas Figuras 3.3, 3.4 , 3.5 e 3.6.

O usuário simples possui a visão mais limitada da ferramenta, este terá somente acesso as seguintes funcionalidades: efetuar o *login* na ferramenta, realizar consultas diversas e

gerenciamento de usuário. Vale lembrar que estas funcionalidades foram descritas anteriormente. Na Figura 3.3 são representadas as funcionalidades que o usuário simples tem acesso na ferramenta, ou seja, o caso de uso do ator usuário simples.

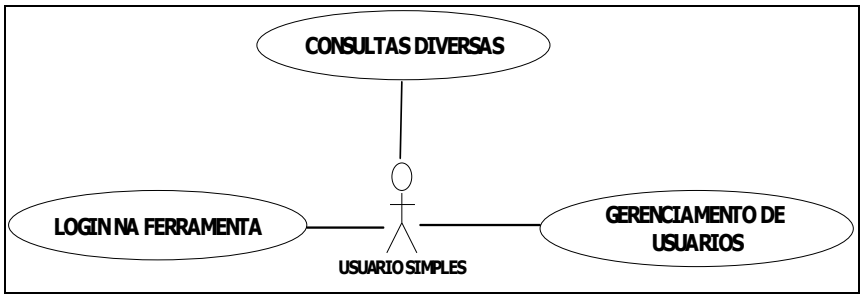

Figura 3.3 – Diagrama de Caso de Uso Usuário Simples

O usuário executor por sua vez terá acesso semelhante ao do usuário simples, sua diferença em relação ao anterior se deve que o mesmo poderá cadastrar determinada atividade referente a um *workflow*. Na Figura 3.4 é apresentado o diagrama de caso de uso do usuário executor.

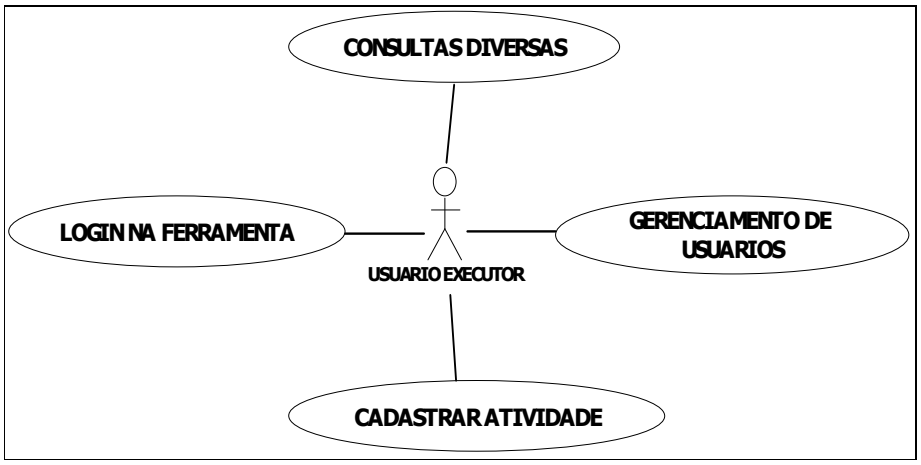

Figura 3.4 – Diagrama de Caso de Uso Usuário Executor

O usuário proprietário por sua vez tem acesso a uma quantidade intermediaria de funcionalidades, entre as principais funcionalidades deste usuário, pode-se citar a funcionalidade de gerenciamento de executores. Na Figura 3.5 é representado o diagrama de caso de uso do ator usuário proprietário do projeto de *workflow,* ou seja, as funcionalidades que este tem permissão para utilizar na ferramenta.

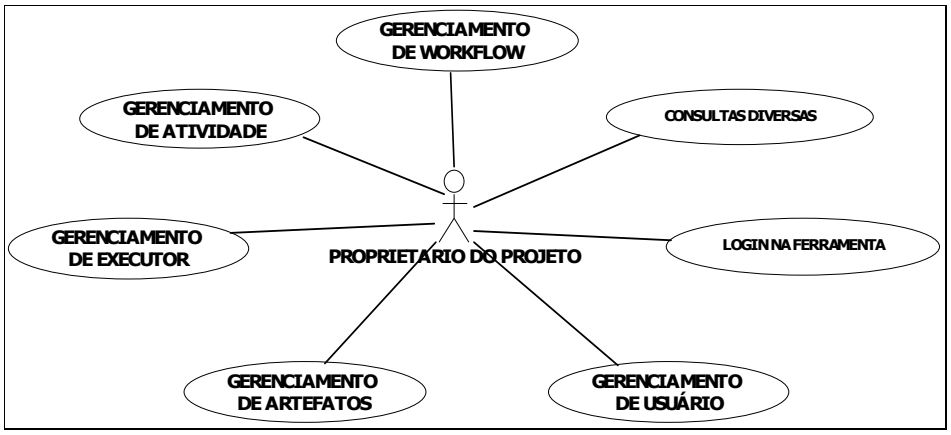

Figura 3.5 – Diagrama de Caso de Uso do Usuário Proprietário do Projeto

Na Figura 3.6 é representado o diagrama de caso de uso do usuário administrador do sistema, este possui acesso e visão global sobre todas as funcionalidades existentes na ferramenta, suas funcionalidades exclusivas em relação aos demais usuários refere-se ao gerenciamento de ferramenta de apoio, etapas de teste, técnicas de teste e critérios de teste. O usuário administrador do sistema tem acesso a todos os projetos de *workflow* mesmo que os mesmos não tenham sido cadastrados por ele.

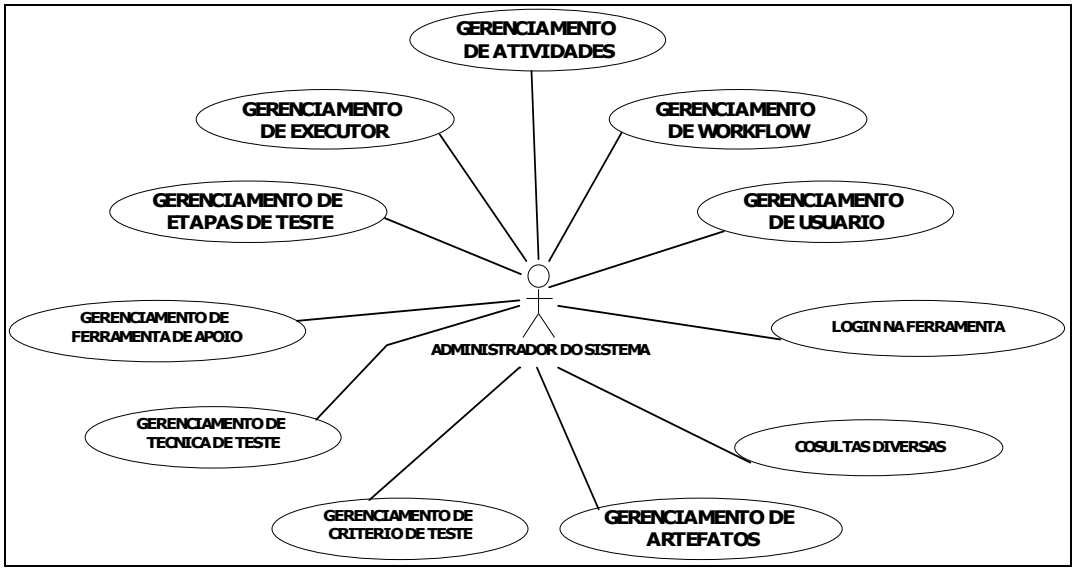

Figura 3.6 – Diagrama de Caso de Uso do Usuário Administrador do Sistema

3.3. Arquitetura

Para o apoio a documentação de *workflow* de teste de software, criou-se uma solução arquitetural visando à construção de uma infra-estrutura computacional, com o intuito de facilitar a para os testadores a atividade de documentação da aplicação de teste de software.

Nesta solução arquitetural representada pela Figura 3.7 pode-se observar a existência de quatro módulos, os quais podem comunicar-se entre si, o produto final desta solução é a produção de artefatos os quais consistem na documentação da atividade de teste de software, tendo como base a Norma IEEE 829.

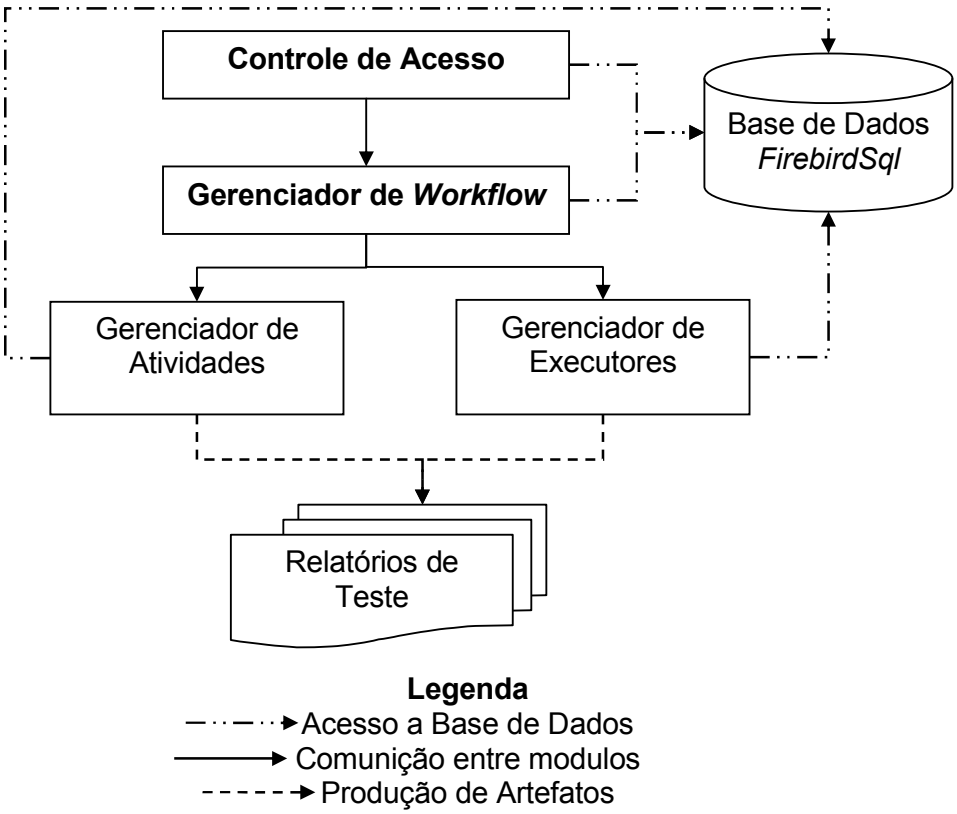

Figura 3.7 – Solução Arquitetural da FADAT

Para esta solução arquitetural, como pode ser observado foram criados quatro módulos que são responsáveis pelo processo que resultara na produção de artefatos os quais consistem nos relatórios de teste segundo a Norma IEEE 829. Os quatro módulos criados foram: *modulo de Controle de acesso, gerenciador de workflow, gerenciador de executores e gerenciador de atividades.*

- **Controle de acesso:** este módulo é responsável por verificar e liberar o acesso a as funcionalidades da ferramenta ao usuário, isto ocorrera após o usuário informar seu nome de usuário e sua respectiva senha, após esta verificação será liberado o acesso para as funcionalidades;
- **Gerenciador de *workflow*:** este terá como principal objetivo gerenciar os workflows cadastrados na ferramenta, como funções podem-se citar a consulta,

inserção, alteração e remoção dos *workflows,* a ultima só poderá ser realizada se o *workflow* não tiver nenhuma dependência;

- **Gerenciador de Executores:** este módulo terá como finalidade principal gerenciar quais serão os executores responsáveis por cada atividade referente a um determinado *workflow*, esta função só poderá ser executada pelo autor do *workflow,* as funções relacionadas a este modulo pode-se citar a inclusão alteração e consulta de usuário por atividade, deve-se salientar que poderá ser cadastrado apenas um executor por atividade;
- **Gerenciador de Atividades:** este será responsável por gerenciar as atividades que deverão ser realizadas para que se produza um relatório de teste de software, estas atividades estão diretamente ligadas ao *workflow* adotado na Figura 3.2.

3.4. Implementação

Após a análise da documentação da norma IEEE 829 e do modelo conceitual iniciou-se a implementação da ferramenta. Para a implementação foi utilizada a linguagem de programação *Web* JSP (*Java Server Pages*) utilizando o servidor de aplicações Apache Toncat com modulo de JSP e foi utilizado o SGBD (Sistema Gerenciador de Banco de Dados) *Firebird*SQL (2005).

Na Figura 3.8 é representado o esquema da arquitetura utilizada para implementar a ferramenta FADAT, como se pode observar carrega-se a ferramenta no *browser* do cliente, esta faz requisições para o servidor *web,* que por sua vez transmite a requisição ao banco de dados este responde a requisição e por sua vez o servidor envia a resposta para o *browser* do cliente após o processamento.

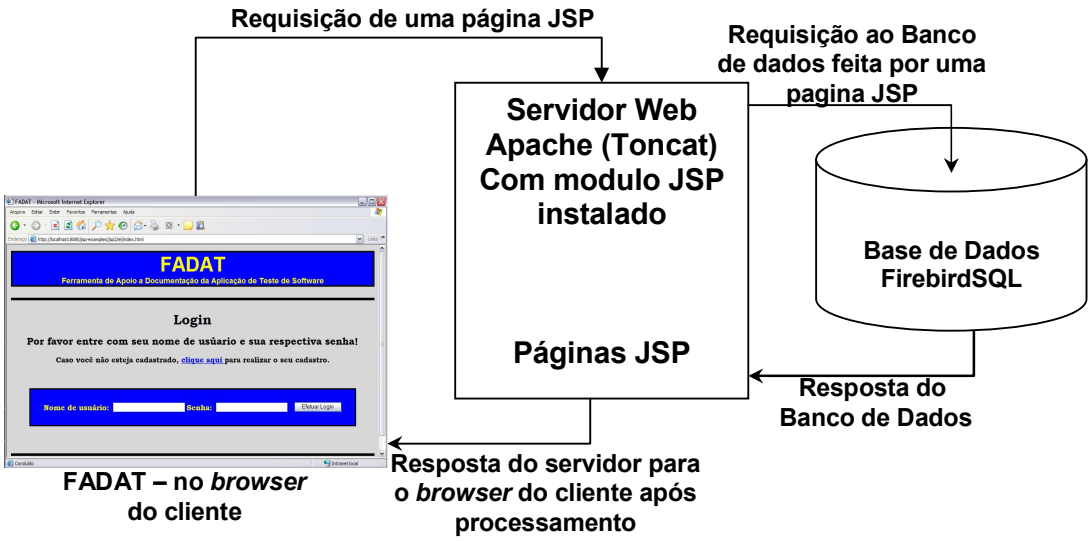

Figura 3.8 – Esquema da arquitetura da FADAT

Inicialmente implementou-se o cadastro de usuários, onde permitiu-se que o usuário entrasse com seus dados entre estes, o nome de usuário e sua respectiva senha, após o cadastro foi implementado a autenticação do usuário na ferramenta, pois uma vez cadastrado na ferramenta o usuário poderá realizar a autenticação no momento em que desejar, para efetuar a mesma o usuário deverá informar o nome de usuário e sua respectiva senha.

Na Figura 3.9 é apresentada a tela inicial da ferramenta FADAT que corresponde a tela de autenticação, onde o usuário deverá informar seu nome de usuário e sua respectiva senha, caso o usuário não seja cadastrado este terá a opção de realizar o cadastro na ferramenta.

Na Figura 3.10 é apresentado o trecho de código que realiza o login na ferramenta seu funcionamento tem como base comparar todos os nomes de usuário e suas respectivas senhas armazenadas na base de dados com o nome de usuário e a senha informada pelo usuário.

Figura 3.9 – Tela de Autenticação da Ferramenta

```
1: <%//inicio do codigo java
2:     String usuario=request.getParameter("usuario");
3:     String password=request.getParameter("passqord");
4:     try{
5:         Statement stmt  = conectar.conectarBD();
6:             try{
7:                String Sql ="SELECT NOME_USUARIO,SENHA_USER FROM USUARIO";
8:                 ResultSet r= executeQuery(Sql);
9:                 String user;
10:                String senha;
11:                boolean teste=false;
12:                while(r.next()){
13:                    user=r.getString(1);
14:                    if(user.equals(usuario.toUpperCase())){
15:                        senha=r.getString(2);
16:                        if(senha.equals(password.toUpperCase())){
17:                            teste=true;
18:                        }//fim if senha
19:                    } //fim if nome usuario
20:                }//fim while
21:                r.close();
22:                conectar.desconectarBD();
23:                if(teste){
24:                    out.println("Login realizado com Sucesso");
25:                }else{
26:                    out.println("Erro no Login: usuario ou senha errado");
27:                }
28:        }catch(Exception eS){out.println("Erro!!"+eS);    } //fin catch
29:    }catch (Exception eS){out.println("Erro!!"+eS);    }//fim catch
30: %>
31:
```

Figura 3.10 – Trecho de código que controla o login

Após a implementação do cadastro e da autenticação iniciou-se a implementação do *workflow,* para este foram implementadas as funções de cadastrar, alterar, consultar e remover *workflow* da ferramenta. Deve-se lembrar que esta última função poderá ocorrer se e somente se o *workflow* não possuir nenhuma dependência em relação à classe *atividade*.

Após implementar a classe *workflow* iniciou-se a implementação das classes que irão auxiliar a classe *atividade* a gerar os artefatos de saída. Inicialmente implementou-se a classe *etapa de teste (classe etapa_teste)* onde a mesma será utilizada pela classe ferramenta de apoio (*classe ferramenta_apoio*), após a etapa anterior implementou-se a classe critério de teste (*classe critério_teste*) a qual será utilizada pela classe técnica de teste (*classe técnica_teste)*.

Logo após as etapas descritas anteriormente implementou-se o gerenciador de executores, a qual tem a finalidade de controlar os respectivos executores para cada atividade, deve-se salientar que está já faz parte da classe atividade.

E após as etapas supracitadas iniciou-se a implementação da classe atividades (*classe atividade*), esta foi implementada juntamente com as classes artefato e Artefato atividade, respectivamente (*classe artefato, classe artefato_atividade*), deve-se salientar que, pelo motivo de que uma determinada atividade possa depender de um artefato de entrada, onde este é um artefato de saída para outra atividade, implementou-se de forma simultânea estas classes.

3.5. Exemplos de Uso

Para a ferramenta apresentada, realizou-se dois exemplos de uso, um realizado por ,Geraldi (2006) e por, Sarmento (2006), onde o primeiro realizou o cadastro de um *workflow* teste ABDR e o segundo um cadastro de *workflow* de teste OO.

Neste tópico será demonstrado o passo a passo que Geraldi (2006) realizou na ferramenta para obter o documento do relatório de plano de teste, deve-se salientar que todos os relatórios gerados seguem o modelo da Norma IEEE 829.

O primeiro passo realizado por Geraldi (2006) para cadastrar seu *workflow,* foi realizar o cadastro de usuário na ferramenta, os campos: *nome, sobrenome, e-mail, nome de usuário e senha* são solicitados para realizar o cadastro. Na Figura 3.11 é representada a tela de cadastro gerada pela ferramenta em questão.

Figura 3.11 – Cadastro de Usuário

Após o cadastro é gerada uma tela informando para o usuário se o cadastro foi ou não realizado com sucesso. Na Figura 3.12 esta demonstrada a tela de cadastro de usuário realizado com sucesso para o cadastro do usuário "*Renato Geraldi*".

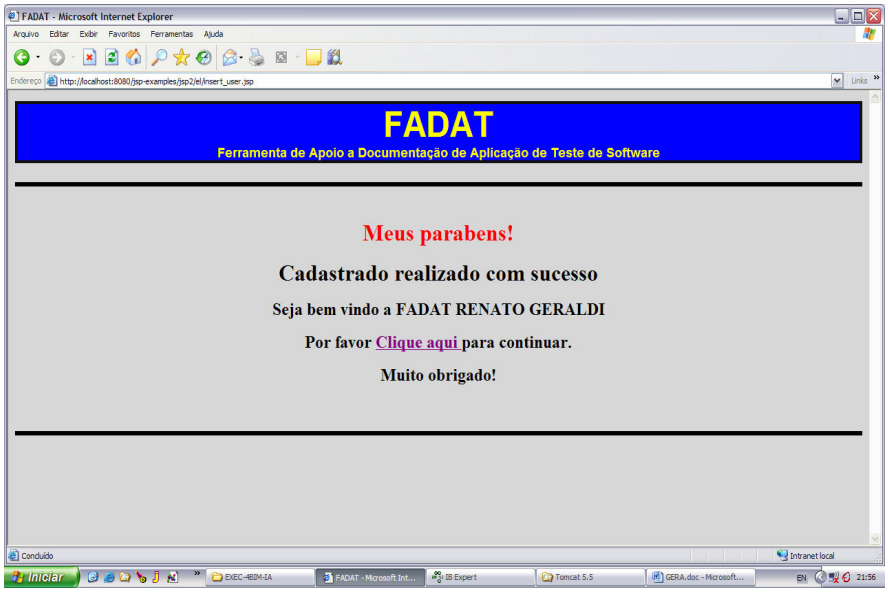

Figura 3.12 – Cadastro de Usuário Realizado com Sucesso

Deve-se salientar que no caso do usuário já estar cadastrado na ferramenta o mesmo deverá apenas realizar o login. Na Figura 3.13 é representada a tela de login gerada pela ferramenta, onde o usuário deverá informar seu nome de usuário e sua respectiva senha.

Figura 3.13 – Login na ferramenta

Após o cadastro de usuário e ou *login* será gerada uma tela ao usuário informando se o *login* e ou *cadastro* foi ou não realizado com sucesso, na Figura 3.14 esta representado a tela de cadastro realizado com sucesso, caso o usuário realize o *login* na ferramenta será mostrado uma mensagem de login realizado com sucesso como demonstrado na Figura 3.15.

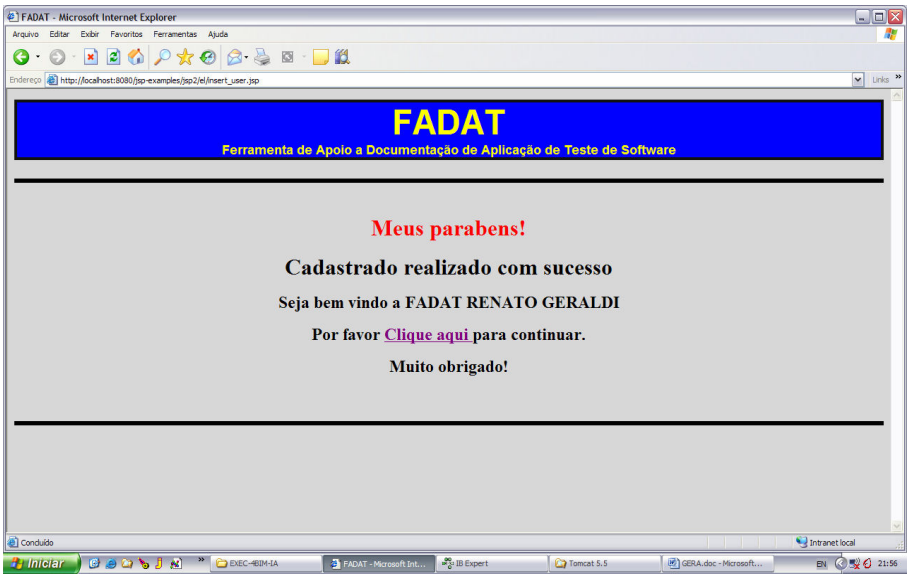

Figura 3.14 – Cadastro de Usuário Realizado com Sucesso

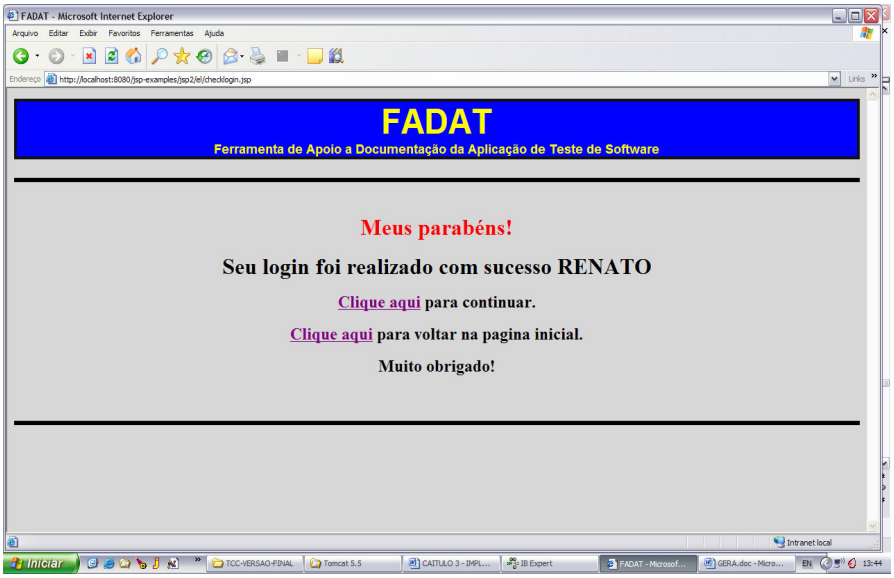

Figura 3.15 – Login Realizado com Sucesso

Após o cadastro e ou *login* o usuário terá acesso à página principal da ferramenta, a qual contêm todas as funcionalidades referentes a *usuário, workflow* e *atividades*. Na Figura 3.16 está representada a página principal da ferramenta.

Figura 3.16 – Página Principal da Ferramenta

Após o usuário estar devidamente cadastrado na ferramenta e possuir acesso à página principal da ferramenta, o mesmo irá iniciar o próximo passo para gerar os documentos de teste de software, o passo mencionado será o passo de cadastrar um *workflow* na ferramenta, para que posteriormente possam ser geradas as atividades para o mesmo o que resultará na criação dos relatórios de teste.

Na Figura 3.17 é representado o cadastro de *workflow* realizado por, Geraldi (2006), onde o mesmo informou o nome do *workflow*, objetivo e sua descrição os campos nome e sobrenome apenas auxiliarão na validação da atividade.

Como pode ser observada na Figura 3.1 a classe *workflow* contem o campo data de criação e na Figura 3.17 não possui nenhum campo para a entrada da data, isto ocorre pelo

motivo de que a data e gerada automaticamente pela ferramenta na hora do cadastro do *workflow*.

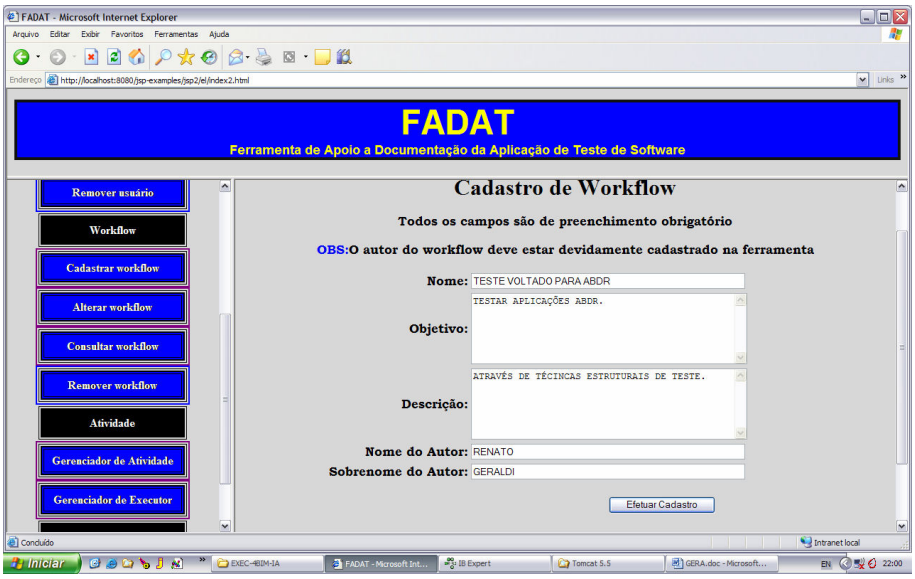

Figura 3.17 – Cadastro de *Workflow*

Após o cadastro do *workflow* o usuário será informado se o cadastro foi realizado com sucesso ou se ocorreu algum erro no decorrer do mesmo. Na Figura 3.18 representa-se a mensagem de sucesso ou erro demonstrada para o usuário após o cadastro de *workflow*, no caso como pode ser observado mostrou-se a mensagem de sucesso.

Figura 3.18 – Mensagem de Sucesso ou Erro para o cadastro de *Workflow*

Após o termino dos passos anteriores o usuário já estará habilitado a criar os documentos de teste pela ferramenta, mas para que isto seja realizado o mesmo deverá passar por mais algumas atividades.

A primeira atividade que o usuário deverá realizar será o gerenciamento de executores, onde o autor do *workflow* ira determinar qual o executor para cada atividade.

Para a ferramenta foram criadas oito atividades, sendo que cada uma destas atividades irá gerar um relatório, este por sua vez segue a norma IEEE 829, ou seja, cada documento da norma representará uma atividade.

Na Figura 3.19 é apresentado o cadastro de um *executor* para a atividade *Plano de Teste,* para efetuar o cadastro de executor para a atividade de Plano de Teste o usuário terá que informar o nome do *workflow,* o nome do usuário que irá realizar a atividade, qual o cargo deste usuário, a descrição do cargo e a senha do autor do *workflow,* deve-se salientar que esta ação foi gerada por Geraldi (2006).

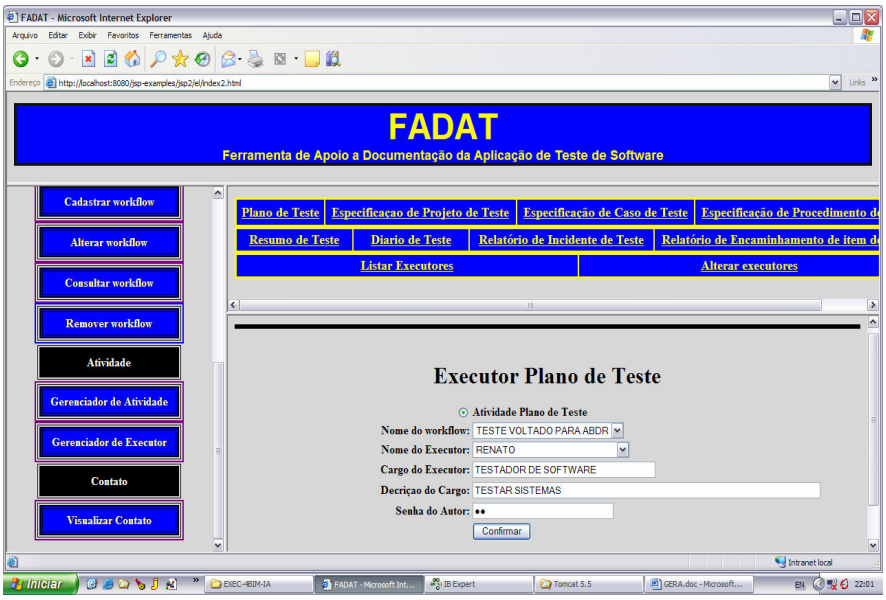

Figura 3.19 – Cadastro de Executor para a Atividade de Plano de Teste

Deve se salientar que esta ação deve ser repetida para todas as atividades, caso esta ação não seja realizada ocorrerá um erro na hora de cadastrar a atividade. Para o gerenciamento de executores podem-se utilizar ainda as funções de listar executores de um determinado *workflow* e alterar determinado executor de um determinado *workflow*.

Após o cadastro do executor para cada atividade, o usuário será informado se o cadastro foi realizado com sucesso ou se durante a realização do mesmo ocorreu algum erro. Na Figura 3.20 é representada a mensagem de sucesso para o cadastro de executor para a atividade de plano de teste.

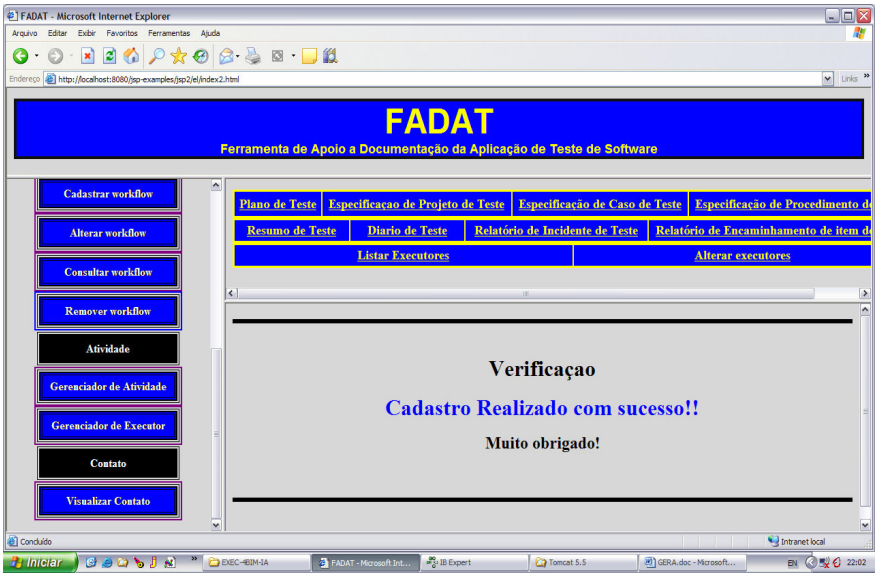

Figura 3.20 – Mensagem de sucesso para cadastro do executor

Após determinar qual o executor para cada atividade o usuário poderá dar início ao cadastro das atividades, este cadastro irá auxiliar o executor na confecção dos artefatos, estes artefatos serão gerados de acordo com a Norma IEEE 829.

Deve-se salientar que a atividade só poderá ser realizada pelo executor determinado no cadastro de executores, onde para cadastrar uma determinada atividade o executor deverá proceder da seguinte forma:

1. **Passo**: Primeiramente ir para a página de controle de atividades (ver Figura 3.21);

Figura 3.21 – Página Gerenciador de Atividades

2. **Passo**: escolher qual atividade o mesmo deseja cadastrar. No caso apresentado, será a atividade do plano de teste, onde o mesmo deverá escolher o *workflow*, selecionar seu nome e informar a senha de usuário, como demonstrado na Figura 3.22.

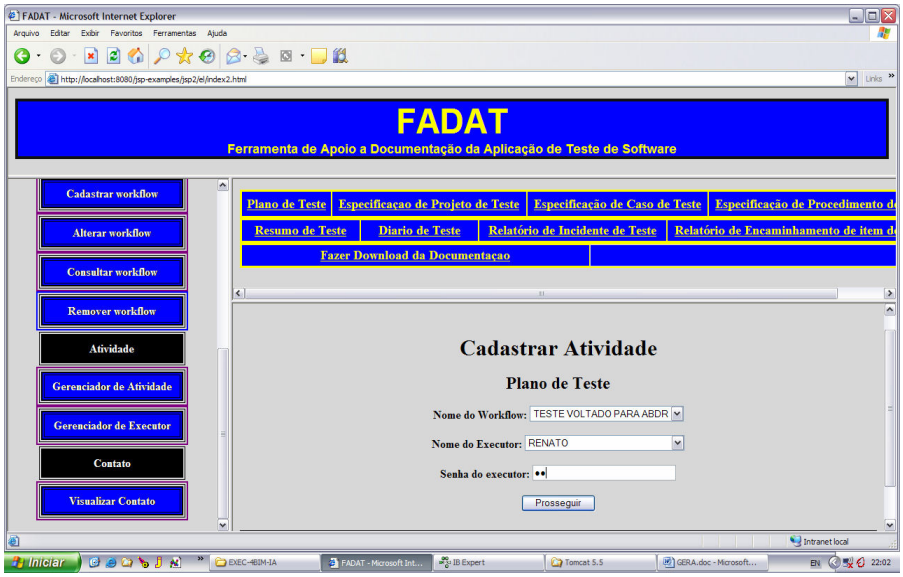

Figura 3.22 – Atividade Plano de Teste

3. **Passo**: neste passo o usuário deverá escolher qual a ferramenta de apoio, a etapa de teste que a mesma cobre, em seguida a técnica de teste utilizada e seu

respectivo critério e por fim o objetivo do relatório, como demonstrado na Figura 3.23.

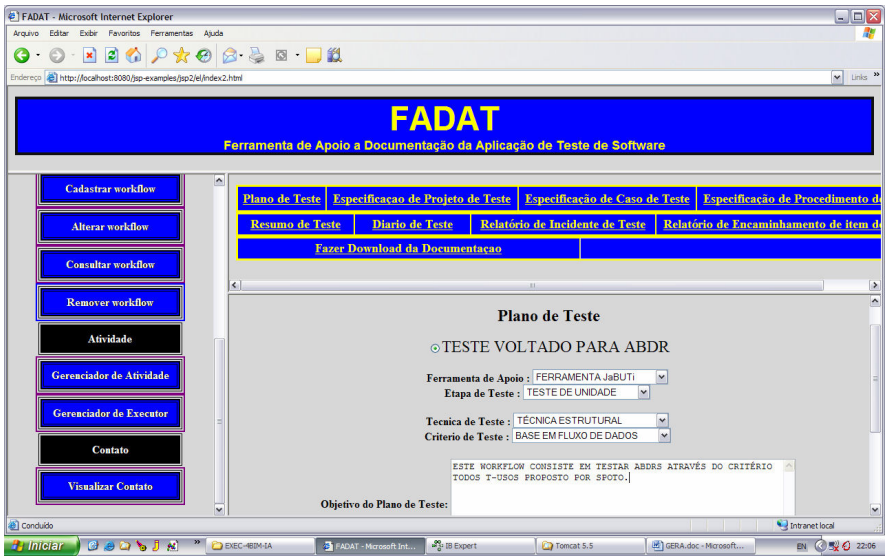

Figura 3.23 – Escolher Ferramentas e Tecnicas

4. **Passo**: Preencher os campos de forma adequada até que apareça a tela de cadastro de atividade realizada com sucesso representada pela Figura 3.24. Deve-se salientar que se no decorrer do preenchimento o usuário não preencher algum campo, este aparecerá no artefato com a seguinte mensagem "*DADOS NÃO INFORMADO PELO EXECUTOR*".

Figura 3.24 – Atividade Realizada com Sucesso

Após os quatro passos citados anteriormente a atividade estará completa, não podendo ser mais alterada, esta estará somente disponível para *download,* deve-se salientar que o mesmo só poderá ser realizado pelo autor do *workflow*. No apêndice B demonstra-se um exemplo da documentação gerada pela ferramenta.

Na Figura 3.25 é representada uma tela que corresponde ao cadastro da atividade *Diário de teste* onde o usuário deverá informar os seguintes campos: Objetivo do relatório identificador do diário de teste, descrição do teste e registros de atividades e eventos.

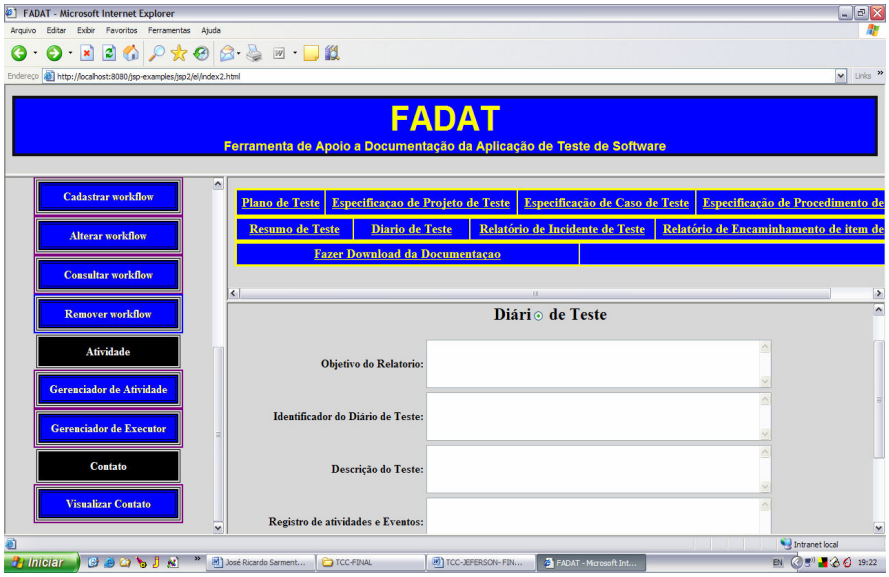

Figura 3.25 – Cadastro de Diário de Teste

Na Figura 3.26 é representado o código fonte responsável em pegar os dados informados pelos clientes na Figura 3.25 e gerar a documentação da atividade de diário de teste, as quais ficarão armazenadas na ferramenta e disponíveis para *download*. Deve-se salientar que caso o usuário não informe algum campo este será marcado como "não informado pelo executor da atividade". No Apêndice B apresenta-se um exemplo da documentação gerada pela ferramenta para a atividade: diário de teste.

```
<%//inicio do codigo java
    String cod_wor=request.getParameter("wor");
    String ident=request.getParameter("ident");
    String desc=request.getParameter("desc");
    String reg=request.getParameter("reg");
    String obj=request.getParameter("obj");
    try{
        Statement stmt  = conectar.conectarBD();
            try{
                String sql="SELECT NOME_USER,SOBRENOME_USER,CARGO FROM ";
                sql=sql+"USUARIO, EXECUTOR WHERE COD_EXEC=CODIGO_USER";
                sql=sql+"AND COD_ATIV=5 AND COD_WORK="+wor;
                ResultSet r= executeQuery(sql);
                String nome;
                String cargo;
                while(r.next()){
                    nome=r.getString(1)+" "+r.getString(2);
                    cargo=r.getString(3);
                }//fim while
                r.close();

                String nome_arq=wor+"5.doc";
                File newfile = new File(nome_arq);
                RandomAccessFile arq = new RandomAccessFile(newfile,"rw");
                arq.seek(newfile.length());
                arq.writeBytes("RELATÓRIO DIARIO DE TESTE \r\n\r\n");
                arq.writeBytes("EXECUTOR DA ATIVIDADE: "+nome+" \r\n");
                arq.writeBytes("CARGO DO EXECUTOR: "+cargo+" \r\n");
                arq.writeBytes("DATA DE INICIO: "+data_atual+"\r\n\r\n");
                arq.writeBytes("OBJETIVO \r\n\r\n");
                arq.writeBytes("\t"+obj+"\r\n\r\n");

                //escreve nos arquivos
                raf.writeBytes("IDENTIFICADOR DO DIARIO  DE TESTE \r\n\r\n");
                if(ident==""){
                  arq.writeBytes("\tDADOS NÃO INFORMADO PELO EXECUTOR \r\n");
                }else{
                    arq.writeBytes("\t"+ident+"\r\n\r\n");
                }
                arq.writeBytes("DESCRIÇÃO DO TESTE\r\n\r\n");
                if(desc==""){
                    arq.writeBytes("\tDADOS NÃO INFORMADO PELO EXECUTOR\r\n");
                }else{
                    arq.writeBytes("\t"+desc+"\r\n\r\n");
                }
                arq.writeBytes("REGISTROS DE ATIVIDADES E EVENTOS\r\n\r\n");
                if(reg==""){
                    arq.writeBytes("\tDADOS NÃO INFORMADO PELO EXECUTOR\r\n");
                }else{
                    arq.writeBytes("\t"+reg+"\r\n\r\n");
                }
                arq.close();
                conectar.desconectarBD();
        }catch(Exception eS){out.println("Erro!!"+eS);         } //fin catch
    }catch (Exception eS){out.println("Erro!!"+eS);    }//fim catch
%>

```

Figura 3.26 – Código fonte para gerar o documento Diário de Teste

Para realizar o *download* da documentação o mesmo deverá ir à página de *download* da documentação onde o mesmo deverá informar o nome do *workflow*, o nome do autor e a senha do autor, como demonstrado na Figura 3.27, após esta ação será listado todos os documentos completos para o determinado *workflow* como representado na Figura 3.28.

Deve-se salientar que os documentos gerados pela ferramenta possuem como forma de nomeação a concatenação do código do workflow mais o código da atividade. Por exemplo, realizar a atividade plano de teste que possui código igual a 1 para um *workflow* que possui código igual a 15, neste caso o arquivo será nomeado como “151.doc”.

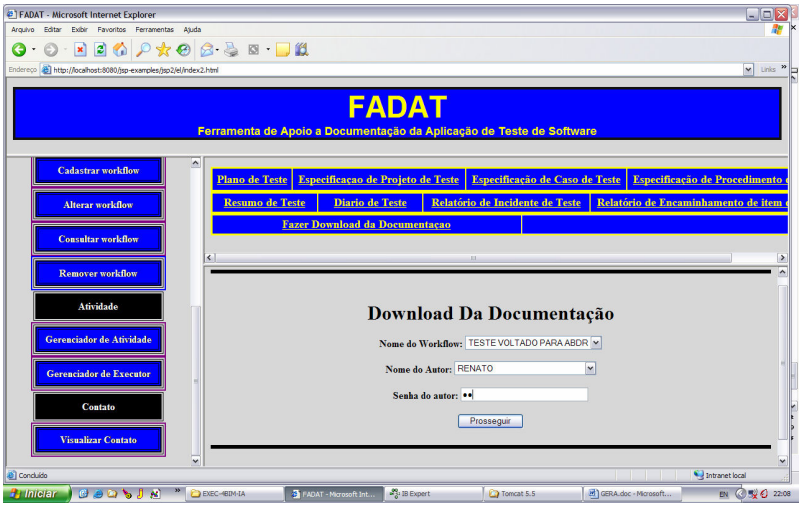

Figura 3.27 – Página *download* da documentação

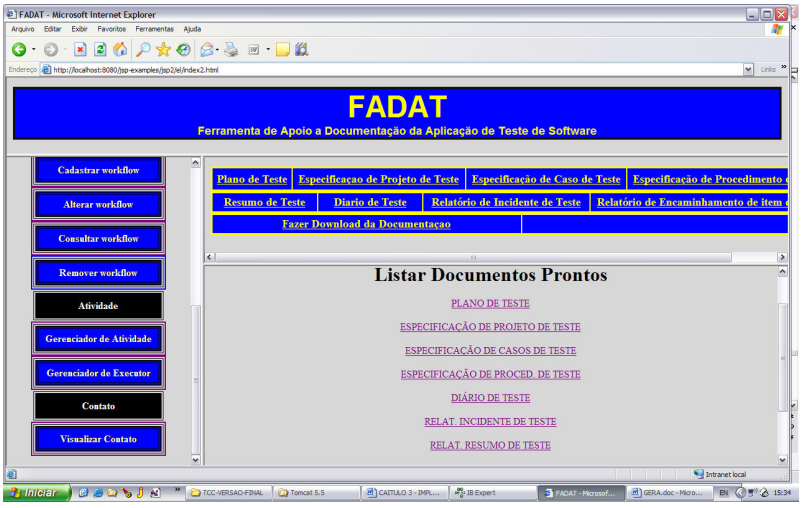

Figura 3.28 – Documentos disponíveis para *download*

3.6. Considerações Finais

Após o término da implementação da ferramenta, pode-se concluir que a mesma corresponde às perspectivas iniciais, onde a principal perspectiva em relação a sua funcionalidade era gerar documentos da aplicação de teste de software de forma sistemática e eficaz, facilitando desta maneira a documentação da mesma.

Outro ponto que observado após o término da implementação é que esta ferramenta poderá ser aprimorada tornado-a mais eficiente e útil para o desenvolvimento de software. Os pontos que podem ser melhorados estão descritos no tópico: Trabalhos Futuros; apresentado no capítulo final.

No próximo capitulo são apresentadas as conclusões referentes ao desenvolvimento da ferramenta supracitada.

CONCLUSÕES

Considerações Iniciais

Este capítulo tem como principal objetivo apresentar as conclusões observadas durante o desenvolvimento da ferramenta citada no Capitulo 3. Primeiramente são apresentados os resultados obtidos e suas respectivas contribuições para a sociedade demonstrando assim a viabilidade do trabalho apresentado, bem como as dificuldades e limitações encontradas.

Outro ponto importante a ser abordado neste capítulo é a perspectiva de trabalhos futuros onde são apresentadas algumas perspectivas de continuação para este trabalho.

Resultados Obtidos e Contribuições

Nesta seção são relatados os resultados obtidos durante o desenvolvimento deste trabalho, os quais viabilizam a continuidade e ou aprimoramento da ferramenta ou mesmo aprovam o desenvolvimento da ferramenta apresentada no Capítulo 3.

Como principal resultado obtido pode-se citar a implementação de uma ferramenta *web,* que auxilia na confecção de artefatos (“documentos”) da aplicação de teste de software de forma sistemática e eficiente e principalmente seguindo uma especificação, no caso a Norma IEEE 829, facilitando a documentação da atividade de teste de software para os executores da mesma.

Outro fator de suma importância que pode ser considerado uma contribuição é o fato de que os artefatos gerados após a utilização da ferramenta ficarão armazenados na ferramenta e disponíveis para *download*, isto pode ser considerado importante pois em caso de perda da documentação gerada o usuário poderá acessar a ferramenta e fazer o *download* da documentação perdida.

Dificuldades e limitações

Durante o desenvolvimento do projeto não foram encontradas pequenas dificuldades, as quais foram resolvidas após um breve estudo e ou pesquisa e ou consultas com professores que trabalham na área.

Entre estas pequenas dificuldades encontradas pode-se citar as peculiaridades do servidor Apache Tomcat, onde após uma boa pesquisa foi resolvido, outra dificuldade encontrada foi de realizar a conexão com o SGBD, isto porque a conexão com o SGBD em JSP possui algumas particularidades, que também foram resolvidas após um sucessivo estudo e apoio de professores e profissionais da área.

Trabalhos Futuros

Várias perspectivas futuras de trabalho foram observadas e ou identificadas para a continuidade deste trabalho, as quais têm o intuito de aprimorar a ferramenta já existente com o intuito de facilitar a atividade de teste de software, a qual não é uma atividade trivial como descrito anteriormente neste trabalho:

1) Implementar um controle de versão para as atividades, pois o projeto atual não permite que um documento seja alterado após o cadastro. Deve-se salientar que esta funcionalidade deveria ter sido, mas devido à falta de tempo não foi possível implementa-la;
2) Uma nova característica que pode ser adicionada à ferramenta já existente é a documentação automática da aplicação de teste de software. Isto pode ser feito através do estudo de ferramentas que apóiam a aplicação do teste de software, como por exemplo, a JaBUTi e a Pooktool. Isto pode ocorrer da seguinte maneira:

importar os resultados gerados pelas ferramentas e a partir destes resultados gerar os relatórios com base na norma IEEE 829;

3) Outra funcionalidade que pode ser adicionada a ferramenta implementada é o apoio à aplicação de teste de software através de manuais, fluxogramas explicativos de acompanhamento de desenvolvimento;
4) Outra possibilidade é a junção dos Itens 2 e 3 descritos anteriormente.

Caso surja interesse de continuar o trabalho deve-se salientar que o interessado deve ler com atenção o Apêndice A deste trabalho para que o mesmo não passe pelas mesmas dificuldades e limitações apresentadas neste trabalho.

Considerações Finais

Após o termino da implementação deste projeto, pode-se concluir que o mesmo foi de suma relevância, pois se apresentou uma ferramenta eficiente e sistemática para a documentação da atividade de teste de software.

Acredita-se que está ferramenta será de extrema importância para o desenvolvimento de software, pois a mesma controla quais os executores responsáveis por cada atividade da aplicação de teste de software.

Sob estes aspectos pode-se concluir que a ferramenta apresentada será de extrema utilidade para a sociedade mais especificadamente para os testadores de software, pois a mesma dinamiza a documentação bem como a aplicação da atividade de teste de software.

REFERÊNCIAS

ATLEE, Joanne. Workflow Management Systems - A Standard Architecture for Automating Workflows. Tutorial, dezembro, 1997.

BARBOSA, E.F; Maldonado, J.C; Vincenzi, A.M.R.; Delamaro, M.E; Souza, S. R. S; Jino, M. "Introdução ao Teste de Software". ICMC-USP-São Carlos, 2000.

BATISTA, Denerval M. DBValTool: Uma Ferramenta para Apoiar o Teste e a Validação do Projeto do Banco de Dados Relacional – Dissertação de mestrado – UFPR – Curitiba – PR – 2003.

BONFIM JUNIOR, F. T. "JSP a Tecnologia Java na Internet", Editora Érica, São Paulo 2002.

CRESPO, A. N.; MARTINEZ, M. R. M.; JINO M.; ARGOLLO JUNIOR, M. T. "Documentação de Teste Referência: Norma std 829 - 1998". Relatório Técnico RT-IEEE 829 (versão preliminar), Centro de Pesquisas Renato Archer (CENPRA), Campinas-SP, 2000.

FIREBIRDSQL FireBird, on-line, disponível em http://www.firebirdsql.org – Ultimo acesso em 15 de Agosto de 2006.

GERALDI, R., "Criação De Um *Workflow* De Teste Voltado Para Banco De Dados", Trabalho de Conclusão de Curso de Graduação, UNIVEM, Marília, SP, Novembro, 2006.

GONÇALVES, K. V. Teste de software em Aplicações de Banco de Dados Relacional. Universidade de Campinas, Campinas-SP, 2003, Disponível em: http://www.ic.unicamp.br/mp/Material/Trabalhos /TF_Klausner.pdf. Acesso em Março de 2006.

JAVA Java, on-line, disponível em *http://java.sum.com* – Ultimo acesso em 15 de Maio de 2006.

KRUCHTEN, P. "*The Rational Unified Process an Introduction*", 2ª edição, editora Addison – Wesley, 2000.

LEMOS, O. A. L. "Teste de programas orientados a aspectos uma abordagem estrutural para AspectJ", dissertação apresentada a ICMC – USP, São Carlos 2005.

MALDONADO, J. C. Barbosa E. F. Vincenzi A. M. R. Delamaro, M. E. Souza, S. R. S. Jino, M. "Introdução ao teste de software" (versão 2004 - 01), notas didáticas do ICMC – USP, São Carlos 2004.

MALDONADO, J.C.; Martiniano, L. A. F.; Dória, E.S.; Fabbri, S.C.C.P.F.; Mendonça, M. "Readers Project: Replication of Experiments -A Case Study Using Requeriments Documents". Evaluation Workshop - International Cooperation, CNPq, Rio de Janeiro, 2001.

MATOS, E. S. "*Workflow* para testes de software". Dissertação de Mestrado. Universidade Federal de Campina Grande, Campina Grande-PB, 2004.

MORO, M. M. "*Workflow* na WEB". Trabalho final entregue para disciplina de Sistema de Banco de dados Distribuídos. Disponível em: http://www.inf.ufrgs.br/~mirella/*workflow*/home.html, Porto Alegre-RS, 1998.

PRESSMAN, R. S. "Engenharia de Software", Editora Makron Books, São Paulo 1995.

PRESSMAN, R. S., "*Software Engineering: A practitioner's approach"*. Quinta edição. McGraw-Hill, 2000.

PRESSMAN, R. S., "*Software engineering: A practitioner's approach"*. Sexta edição. McGraw-Hill, 2005.

RAMALHO, J. A. "HTML Avançado", Editora Makron *Books,* São Paulo 1997

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. Qualidade de Software Teoria e Pratica. Editora Prentice Hall, São Paulo, 2001.

SARMENTO, J. R. , "Criação de Um *Workflow* de Teste para o Desenvolvimento de Aplicação Orientado a Objetos", Trabalho de Conclusão de Curso de Graduação, UNIVEM, Marília, SP, Novembro, 2006.

SILVA, A. V. "Modelagen de Processos para Implementação de Wo*rkflow*: Uma avaliação Critica", Tese de pos-graduação. Universidade Federal do Rio de Janeiro, Março de 2001.

SOMMERVILLE, I. Engenharia de Software. Editora Pearson, São Paulo 2003.

SPOTO, E. S.; Peres, L. M. Bueno, P. M. “Um Estudo de Critérios de Teste de Software Baseados em Fluxo de dados”. Monografia de Conclusão de Curso. Universidade Estadual de Campinas, Junho de 1995.

SPOTO, “Edmundo S., Teste Estrutural de Programas de Aplicação de Banco de Dados Relacional”. Campinas: UNICAMP-FEE-DCA, Tese de Doutorado, 2000.

TOMCAT, Fundação Apache projeto Jakarta , on-line, disponível em *http://jakarta.apache.org*. – Ultimo acesso em 15 de Maio de 2006.

WMC, *Workflow Management* Coalition. “The *Workflow* Reference Model”. Technical Report TC-1003. Disponível por http://wfmc.org.

APÊNDICE A – Manual de Instalação da FADAT

Primeiramente deve-se instalar a maquina virtual Java (para esta ferramenta foi utilizada JDK -1_5_0-windows-i586), a mesma encontra-se no CD de instalação deste trabalho. Deve-se lembrar que está pode ser obtida de forma gratuita no endereço *http://java.sum.com*.

Em seguida deve-se obter e instalar o servidor de páginas JSP e *Servlets*, a qual é desenvolvida pela Fundação Apache no projeto Jakarta, seu código é aberto e o programa é gratuito. O mesmo pode ser obtido no CD de Instalação deste trabalho ou pelo endereço *http://jakarta.apache.org*.

Deve-se prestar atenção na hora de instalar o servidor de páginas JSP, pois o mesmo utiliza uma porta para a comunicação, geralmente é utilizada a porta *default* que é a porta *8080,* mas caso você tenha instalado em seu computador o Banco de Dados *Oracle,* utilize outra porta como por exemplo a porta *8888,* pois o Banco de Dados *Oracle* utiliza a porta *8080* gerando assim conflito de portas e causando o ma funcionamento do servidor de páginas JSP.

Após os procedimentos supracitados devera ser criadas duas variáveis de ambiente, CATALINA_HOME e JAVA_HOME, onde CATALINA_HOME é o diretório base do Toncat e o JAVA_HOME é o diretório base da plataforma Java.

Por exemplo se o diretório base da plataforma Java for *C:\Arquivos de programas\Java\jdk1.5.0* e o diretório base do *Toncat* é *C:\Arquivos de programas\Apache Software Foundation\Tomcat 5.5,* estas variáveis devem ser:

- JAVA_HOME = *C:\Arquivos de programas\Java\jdk1.5.0;*
- CATALINA_HOME = C:\Arquivos de programas\Apache Software Foundation\Tomcat 5.5.

Caso queira testar se está funcionado vá ao diretório base do *Toncat* na pasta *bin* e de um duplo clique no ícone *toncat5w.exe* este ira startar o servidor *Toncat, após* startar o servidor abra o *browser* e digite o seguinte endereço *http://localhost:numero_da_porta_escolhida* *no* caso da ferramenta implementada foi utilizada a porta *8888* portando o endereço a ser digitado seria o seguinte *http://localhost:8888*.

Após instalar a plataforma Java e o servidor Toncat você deverá instalar o Banco de Dados, para a ferramenta foi utilizado o Banco de Dados *FireBird* Versão 1.5.3, o mesmo pode ser obtido através do CD do trabalho ou pelo endereço http://www.firebirdsql.org.

Para instalá-lo proceda da seguinte maneira:

I. Instale de forma normal escolhendo a melhor configuração para a sua utilização;
II. Após a instalação deve-se liberar uma porta no *firewall,* como demonstrado na Figura A.1;
III. Deve-se copiar o arquivo *fbclient.dll* que esta na pasta *bin* do FireBird para a pasta *system32* que esta dentro da pasta *WINDOWS;*
IV. Após as etapas supracitadas deve-se copiar os arquivos, *jaas.jar, log4j-core.jar, mini-j2ee.jar, jaybird-full-2.0.1.jar, mini-concurrent.jar* e *firebirdsql-full.jar,* que estão na pasta jar do CD deste Trabalho e coloca-los nas seguintes pastas; ../common/lib do TomCat, ../WEB-INF/lib da sua aplicação Web e na *C:\Arquivos de programas\Java\jdk1.5.0\jre\jdbc.* Obs: não coloque outros arquivos jars (do FireBird) porque o TomCat se perde e não acha o drive correto
V. Logo após deve-se configurar as variáveis de ambiente no windows XP (painel de controle – sistema - avançado - variáveis de ambiente): onde será adicionado os seguintes valores:

VI. **CLASSPATH com o valor:** *C:\Arquivos de programas\Java\jdk1.5.0_01\ jre\bin; C:\Arquivos de programas\Java\jdk1.5.0_01\ jre\jdbc\jaybird-full-2.0.1.jar; C:\Arquivos de programas\Apache Software Foundation\Tomcat 5.5\bin\servlet-api.jar; C:\Arquivos de programas\Apache Software Foundation\Tomcat 5.5\ bin\ jaybird-full-2.0.1.jar;*

VII. **PATH com o valor:** *C:\WINDOWS\system32;C:\WINDOWS;C:\Arquivos de programas\Java\jdk1.5.0_01\bin;C:\Arquivos de programas\Java\jdk1.5.0_01\ jre\bin;C:\Arquivos deprogramas\Java\jdk1.5.0_01\jre\lib;C:\Arquivos de programas\ Java\ jdk1.5.0_01\jre\jdbc\jaybird-full-2.0.1.jar*

VIII. **JAVA_HOME com o valor:** *C:\Arquivos de programas\Java\jdk1.5.0_01*

IX. **CATALINA_HOME com o valor:** C:\Arquivos de programas\Apache Software Foundation\Tomcat 5.5

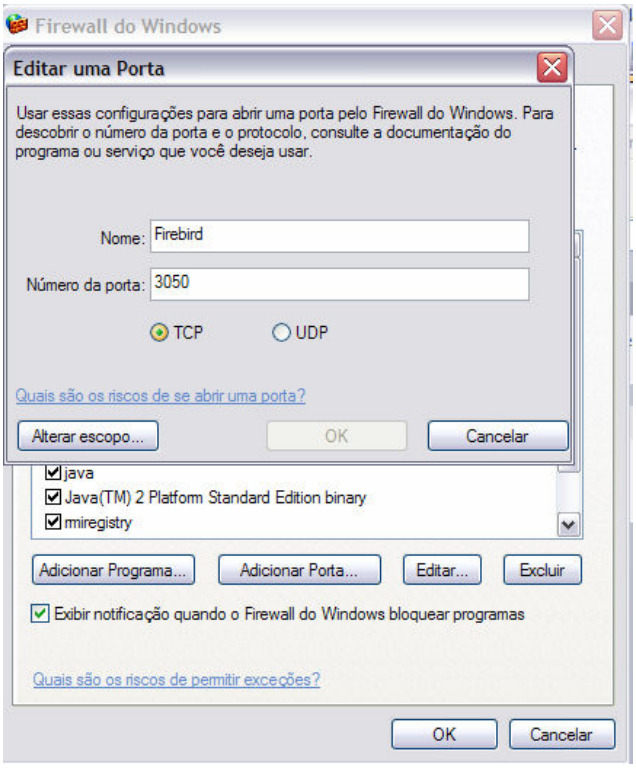

Figura A.1 – Liberar porta no Firewall

Após as etapas descritas anteriormente copie todos os arquivos da pasta ..\codigo_fonte e cole na pasta ...Tomcat 5.5\webapps\jsp-examples\jsp2\el.

Após instale o programa IBExpert que esta na pasta ...\IBExpert do CD deste trabalho, após instalar registre a base de dados que esta na pasta ...\webapps\jsp-examples\jsp2\el como demonstrado na Figura A.2. Para registrar execute o programa IBExpert e após clique em D*atabase* e em seguida em *Registry Database,* então ira aparecer a tela como representado na Figura A.2.

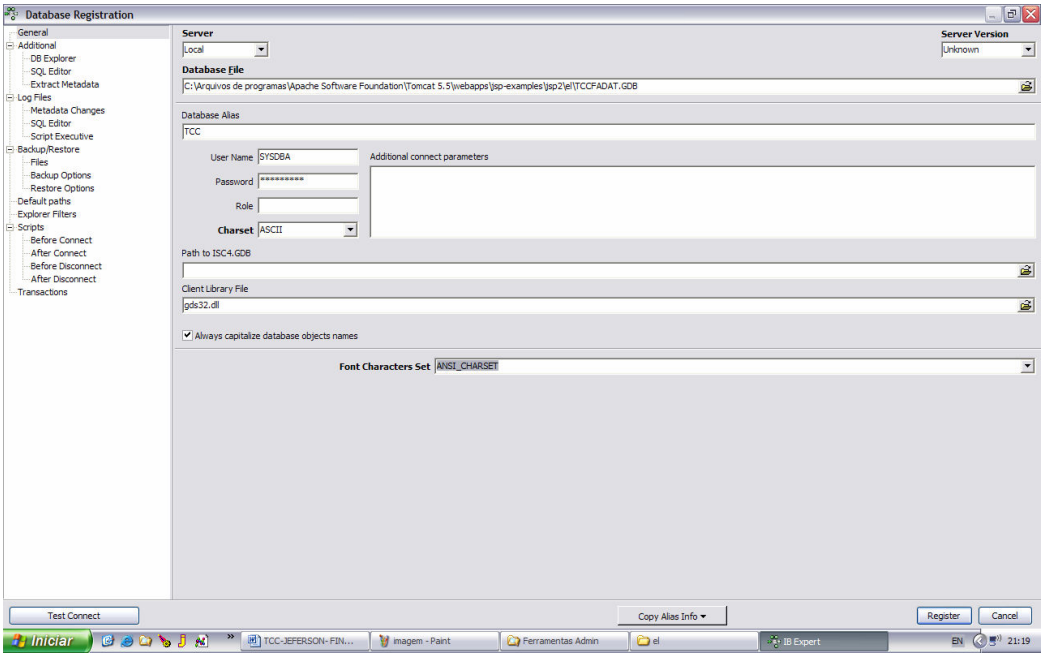

Figura A.2 – Registrar Base de dados

Deve-se lembrar que o *username* e o *password do Firebird* são respectivamente "SYSDBA" e "masterkey", sendo o primeiro obrigatoriamente tudo em maiúsculo e o segundo todo em minúsculo.

Após o termino da instalação já pode se executar a ferramenta implementada neste trabalho, para executar proceda da seguinte maneira:

- *Start* o Servidor Toncat;

- Digite o endereço *http://localhost:8888/jsp-examples/jsp2/el/index.html*;

Após estas ações a Ferramenta FADAT já poderá ser utilizada, deve se lembrar que a ferramenta ira funcionar se e somente se o código fonte for colocado dentro da pasta; em caso contrário todos os arquivos terão de ser alterados.

APÊNDICE B – Documento Gerado pela Ferramenta

RELATÓRIO DIÁRIO DE TESTE

EXECUTOR DA ATIVIDADE: JEFERSON DAVI PARCKERT
CARGO DO EXECUTOR: TESTADOR
DATA DE INICIO: 22/11/2006

OBJETIVO

Tem como principal objetivo apresentar os registros cronológicos dos detalhes relevantes relacionados com a execução dos testes.

IDENTIFICADOR DO DIÁRIO DE TESTE

DADOS NÃO INFORMADO PELO EXECUTOR

DESCRIÇÃO DO TESTE

Este diário registra a execução do procedimento de teste de conversão de dados. Os testes são submetidos por um analista de teste através do terminal do sistema.

REGISTRO DE ATIVIDADES E EVENTOS

Data 10 de setembro
Horário: 14:00 – José Coimbra iniciou os testes;
Horário: 14:45 – Inicio da geração da base de dados antiga;
Horário: 15:55 – Descoberto um possível erro no programa, onde o mesmo preencheu um relatório de incidente de teste ;
Horário: 16:20 – Completada a geração da base de dados antiga;
Horário: 17:00 – José Coimbra encerrou as atividades.