"FUNDAÇÃO DE ENSINO EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES" DE MARÍLIA - UNIVEM

JOSÉ RICARDO SARMENTO

CRIAÇÃO DE UM *WORKFLOW* DE TESTE PARA O DESENVOLVIMENTO DE APLICAÇÃO ORIENTADO A OBJETOS.

MARÍLIA
2006

JOSÉ RICARDO SARMENTO

CRIAÇÃO DE UM *WORKFLOW* DE TESTE PARA O DESENVOLVIMENTO DE APLICAÇÃO ORIENTADO A OBJETOS.

Monografia de conclusão de curso, apresentada a Fundação de Ensino Superior Eurípides de Marília mantenedora do Centro Universitário Eurípides de Marília – UNIVEM – para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof.º Dr.º Edmundo Sergio Spoto.
Co-orientadora: Prof.ª Dr.ª Maria Istela Cagnin Machado.

Área de concentração

Engenharia de *Software*

UNIVEM – MARÍLIA

2006

SARMENTO, José Ricardo
Criação de um *Workflow* de Teste para o Desenvolvimento de Aplicação Orientado a Objetos/ José Ricardo Sarmento; Orientador: Edmundo Sergio Spoto. Marília, SP: [s.n.], 2006.
79 f.

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

1. *Workflow* 2. Teste 3. Orientado a Objetos.

CDD: 005.1

Dedico este trabalho
ao meu pai José Pereira Sarmento,
a minha mãe Rosana Alvarez Sarmento,
e a meu irmão Caio Rogério Sarmento
pela dedicação, incentivo, paciência e sacrifícios pela minha formação.

À minha namorada Susy de Paula Rodrigues,
pela convivência e compreensão por minha ausência,
amor, amizade e muito incentivo.

Pois sem a presença e o amor de todos nada teria conseguido.

AGRADECIMENTOS

A Deus, pela vida e por ter me proporcionado mais essa conquista.

À minha família, em especial meus pais, José Pereira e Rosana
pelo constante apoio e confiança que foram imprescindíveis
para a realização de meu sonho.

A Susy principalmente pela paciência a mim dedicada,
e pelo amor a todo momento por toda a jornada.

Ao meu Orientador Profº. Dr. Edmundo Sergio Spoto
e a minha Co-Orientadora Profª. Dr. Maria Istela Cagnin Machado
pelo incentivo, dedicação, competência e a amizade.

A todos os colegas de trabalho que vinham me acompanhando desde
o início das atividades deste projeto.

E a todos meus amigos Renato, Ronaldo e Jeferson
pelo apoio incentivo e compreensão.

"Se você conhece o inimigo e conhece a si

mesmo não precisa temer o resultado de cem batalhas.

Se você se conhece mas não conhece o inimigo,

para cada vitória ganha sofrerá também uma derrota.

Se você não conhece nem o inimigo nem a si mesmo,

perderá todas as batalhas."

SUN TZU

SARMENTO, José Ricardo. **Criação de um *Workflow* de Teste para o Desenvolvimento de Aplicação Orientado a Objetos**.. 2006. 79 f. Monografia (Graduação em Ciência da Computação) - "Fundação de Ensino Eurípides Soares da Rocha" - Centro Universitário "Eurípides" de Marília - UNIVEM, Marília, 2006.

RESUMO

Uma relação proposta para auxílio ao teste estrutural de programas de Aplicação Orientado a Objetos (OO). Uma aplicação é vista como um conjunto de programas cada programa como um conjunto de unidades. O presente trabalho visa contribuir na identificação e formulação de recursos de teste que possam ser utilizados no teste de *softwares* OO por meio de um *Workflow*, com ênfase nos critérios de teste baseados em fluxo de controle nos critérios todos-nos e todos-arcos, em fluxo de dados no critério todos-usos, intra-classe, com cobertura na fase de teste de unidade. Além disso, este projeto pode despertar o interesse no desenvolvimento de ferramentas automatizadas que permitam uma melhor análise para desenvolvedores de aplicações.Para apoiar o entendimento desses critérios são desenvolvidos exemplos para ilustrar as idéias.

Palavras-chave: Aplicação Orientado a Objetos, Teste, *Workflow, softwares.*

SARMENTO, José Ricardo. **Criação de Um *Workflow* de Teste para o Desenvolvimento de Aplicação Orientado a Objetos**.. 2006. 79 f. Monografia (Graduação em Ciência da Computação) - "Fundação de Ensino Eurípides Soares da Rocha" - Centro Universitário "Eurípides" de Marília - UNIVEM, Marília, 2006.

ABSTRACT

A relationship proposed for a help to the structural test of programs of Application to Object-Oriented (OO). An application is seen as a group of programs each program as a group of units. The present work has intending to contribute for identification and formulation of test resources that they can be used in the test of programs OO through a Workflow, with emphasis in the test criteria based on control flow in the whole criteria all-nodes and all-arches, in data flow on the all-uses criterion, intra-class, with covering in the phase of unit test. Besides, this project can wake up the interest in the development of automated tools that you/they allow a better analysis for applications developers. To support the understanding of those criteria, examples are developed to illustrate it idealizes.

Keywords: *Application Object-Oriented, Test, Workflow, Programs.*

LISTA DE FIGURAS E QUADROS

LISTA DE ABREVIATURAS E SÍMBOLOS

CCFG:	*Class Control Flow Graph*
CMMI:	*Capability Maturity Model® Integration*
DU:	Definição Uso
FADAT:	Ferramenta de Apoio a Documentação da Aplicação de teste de *Software*
IEEE:	*Institute of Electrical and Electronics Engineers*
JSP:	*Java Server Pages*
OO:	Orientado a Objetos
OOA:	Análise Orientada a Objetos
OOD:	Desenvolvimento Orientado a Objetos
RUP:	*Rational Unified Process*
SBBD:	Simpósio Brasileiro de Banco de Dados
SGBD:	Sistema Gerenciador de Banco de Dados
SQL:	*Structured Query Language*
WfMC:	*Workflow Management Coalition*
WfMS:	*Workflow Management System*

SUMÁRIO

INTRODUÇÃO

Atualmente com o aumento de informação constante as empresas de grande e até mesmo de pequeno porte estão se preocupando mais com a consistência da informação em aplicações OO, com isso dando uma maior importância ao teste de programas OO, visando explorar os conceitos e características de Programas OO. Esse tipo de atividade, o teste, é algo de grande custo e muito trabalhoso com isso podendo demorar certo tempo que certas empresas não o esperam, ficando assim com suas informações vulneráveis. Programas descritos com o paradigma OO tentam explorar tipos de dependências de dados e fluxo de informações entre classes distintas e com uso de herança em sua estrutura de reuso de atividades e funcionalidades, com isso o teste de programas OO visa explorar essas características mostrando como as informações são transitadas entre métodos e classes distintas.

O teste consiste basicamente em executar um programa fornecendo dados de entrada e comparar a saída alcançada com o resultado esperado, obtido na especificação do programa (MYERS, 1979).

A atividade de teste é dita como uma das atividades mais custosas do processo de desenvolvimento, podendo consumir 50% dos custos (HARROLD, 2000), a atividade de teste consiste em uma atividade dinâmica que visa a revelar defeitos. Além disso, Harrold destaca que o conjunto de informações obtido durante os testes é de fundamental importância para as atividades de depuração, manutenção e estimativa de confiabilidade (HARROLD, 2000).

Na tentativa de reduzir custos são propostas várias técnicas e vários critérios que auxiliam na condução e avaliação das atividades de teste. A diferença entre essas técnicas está basicamente na origem do dado que é utilizado para avaliar ou construir conjuntos de casos de

teste, sendo que cada técnica possui uma variedade de critérios para esse fim (VINCENZI, 1998).

A eficácia de um critério de teste está relacionada à habilidade do critério em levar o testador a selecionar dados que tenham uma boa chance de revelar os defeitos do programa (BATISTA, 2003).

O teste de produtos de *software* envolve quatro fases: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados dos testes (MYERS, 1979; BEIZER, 1990; MALDONADO, 1991; PRESSMAN, 2001). Tais etapas devem ser desenvolvidas no decorrer do próprio processo de desenvolvimento do *software* e finalizando-se com três etapas de teste: de unidade, de integração e de sistema.

O objetivo deste trabalho é estudar e analisar as etapas de testes de *Software* de Programas OO, visando apoiar a construção de um *workflow* de teste para o desenvolvimento de *software* e conduzir um roteiro das etapas de testes exigidas pela Norma IEEE std 829 de 1998.

O Teste de um *Software* é uma das formas de obtenção da qualidade de *software*, sendo assim este projeto visa auxiliar equipes de testes a melhorar e planejar as tarefas de testes que envolvem os níveis de Unidade, Integração e Sistema.

O *Workflow* de Teste, fruto dos resultados deste trabalho, servirá como uma ferramenta de orientação e apoio ao teste de *Software* em Sistemas OO.

Com pesquisas pude constatar a falta de subsídios para o teste na criação de sistemas, devido ao alto custo desta atividade.

Assim, a geração deste *workflow* para auxiliar nos testes do *software,* isto é, auxiliar na geração de um produto próximo de sua perfeição.

Este trabalho está dividido nos seguintes capítulos: No Capítulo 1 são apresentados os conceitos de teste de *software* e uma introdução sobre teste de *Software* OO. No Capítulo 2

são apresentados conceitos e terminologias sobre *Workflow* e *Workflow* de Teste com um exemplo de *workflow* de teste OO em uma ferramenta de documentação e apoio a teste. No Capítulo 3 é apresentada a Norma IEEE std 829 – 1998 que trata dos passos e técnicas aceitas a serem seguidos pelos testes de *software*.

No Capítulo 4 são apresentados os passos necessários e os procedimentos necessários em forma de *Workflow* de Teste como sendo o propósito deste projeto.

Em seguida as conclusões e trabalhos futuros, que por base deste projeto poderão ser conhecidas e desenvolvidas. E as Referências Bibliográficas, por fim.

1 – TESTE DE *SOFTWARE*

1.1 - Considerações Iniciais

Neste Capítulo são apresentados os conceitos de Teste de *Software*, bem como técnicas envolvidas no Teste.

1.1.2 - Conceitos de Teste de *Software*

Teste de *software* é a atividade responsável pelo sucesso na criação de um *software*, o produto desejado pelos clientes que os requisitam, esses os quais passam por fases durante o processo de desenvolvimento da aplicação.

O teste de *software* é uma das atividades de validação e verificação que tem por objetivo garantir que não haja defeitos na construção do *software*, com grandes verificações analíticas para dar uma garantia de que o *software* final será gerado com qualidade e segurança (MALDONADO, 2001).

Howden (1987), diz que no decorrer do desenvolvimento do *software* que todo e qualquer erro é humano e mesmo com todas as tecnologias, métodos e o pessoal mais capacitado para a execução dos testes pode acontecer de o *software* ficar com falhas, o que faz com que as atividades de revisão e testes tenham um papel fundamental no processo de desenvolvimento de *software*.

Para dar início e fundamentação a atividade de teste deve-se seguir um fluxo que envolve quatro etapas:

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 (MALDONADO, 2001).

Projeto de Casos de Teste: O projeto de testes para *software* e outros produtos que passam por engenharia pode ser tão desafiante quanto o projeto inicial do produto propriamente dito. No entanto, os engenheiros de *software* freqüentemente tratam o teste como algo que se lembraram depois, desenvolvendo casos de testes que podem "demonstrar ser bom", mas têm pouca garantia de ser completos.

Qualquer produto que passa por engenharia pode ser testado por uma das duas maneiras: 1) sabendo a finalidade para qual o produto foi projetado para realizar, podem ser realizados testes que demonstram que cada função esta plenamente operacional, enquanto ao mesmo tempo procuram erros em cada função; 2) conhecendo a operacionalidade interna do produto podemos realizar testes para garantir que seus mecanismos trabalhem conforme suas especificações. O primeiro ponto diz respeito à teste caixa-preta e o segundo, teste caixa-branca (PRESSMAN, 2005; SOMMERVILLE, 2003).

Teste Caixa-Branca: Testes caixa-branca são os testes estruturais também conhecidos como testes de 'caixa de vidro' ou testes 'caixa clara', para diferenciá-los dos testes caixa-preta. Estes testes consistem em uma abordagem de testes que são derivados do conhecimento da estrutura e da implementação do *software*.

Os testes estruturais são, em geral, aplicados a unidades de programa relativamente pequenas, como sub-rotinas, ou às operações associadas com um objeto. Como o nome sugere, o testador pode analisar o código e utilizar conhecimentos sobre a estrutura de um componente, a fim de derivar os dados para o teste. Partindo-se de uma análise do código

podendo descobrir quantos casos de teste são necessários para garantir que todas as declarações no programa ou componente sejam executadas pelo menos uma vez durante o processo de teste.

Teste Caixa-Preta: Testes caixa-preta também são conhecidos como Testes Funcionais que são realizados com base na especificação do programa ou dos componentes. O testador deverá ficar preocupado somente com a funcionalidade, e não com a implementação do *software*. Teste cujo comportamento somente pode ser determinado estudando-se as suas entradas e saídas relacionadas. O testador apresenta as entradas ao componente ou ao sistema e examina as saídas correspondentes. Se as saídas não são as esperadas então o teste detectou um problema no *software*, com sucesso.

Um dos grandes problemas que os responsáveis pelos testes encontram é a forma pelo qual devem escolher as entradas que tenham grande probabilidade de ser membros do conjunto.

Execução dos testes: A execução dos testes ou as fases de teste é dividida em fases, tanto na abordagem de desenvolvimento de *software* procedimental quanto na abordagem orientada a objeto e desenvolvida de forma incremental e complementar. Basicamente há três fases de execução dos testes: de unidade, de integração e de sistema (PRESSMAN, 2005; MALDONADO, 2001).

Teste de Unidade: Tem por objetivo explorar e minerar a menor unidade de projeto do *software* – o componente ou módulo de *software*, sempre procurando identificar erros de lógica e de implementação em cada módulo, separadamente. Unidade é caracterizada como sendo um ‘componente de *software*’ que não deve ser subdividido. Usando a descrição de projeto em nível de componente como guia, caminhos de controle importantes são testados para descobrir erros dentro dos limites do módulo. O teste de unidade é orientado para caixa-branca.

Teste de Integração: Testes de integração têm-se o objetivo de descobrir erros associados às interfaces entre os módulos quando esses são integrados para construir a estrutura do *software*, teste que é uma técnica criteriosa para a construção da estrutura do programa enquanto, ao mesmo tempo, conduz testes para descobrir erros associados às interfaces.

Teste de Sistema: Testes de sistema têm por objetivo identificar erros de funções e características de desempenho que não estejam de acordo com a especificação (PRESSMAN, 2000). Teste de sistema é na verdade uma série de diferentes testes cuja finalidade principal é exercitar por completo o sistema baseado em computador. No teste de sistema mesmo cada parcela de teste tem objetivos distintos, todos buscam a perfeita integração dos elementos do sistema e se estão respondendo de forma satisfatória suas atribuições.

Avaliação dos Resultados: A avaliação dos resultados é a fase na qual se faz validações e verificações dos passos decorridos nos teste, verificando se há a necessidade da execução de algum teste já efetuado ou a criação de um outro teste para fazer o fechamento do teste. Na avaliação dos resultados julga-se fase na qual aplica-se o conhecimento de todos os executores de testes juntamente com observações feitas pelos contratantes dos serviços de teste para que haja uma total qualidade no *software* assim desenvolvido.

1.2 - Teste Orientado a Objetos (OO)

Conforme Pressman (2005) o objetivo do teste é encontrar o maior número possível de erros, com mínimo de esforço aplicado, durante um intervalo de tempo realístico. Apesar de esse objetivo fundamental permanecer constante para o *Software* OO, a natureza dos programas OO muda tanto a estratégia quanto a tática de teste.

Pode-se argumentar que à medida que a Análise Orientada a Objetos (OOA) e o Desenvolvimento Orientado a Objetos (OOD) amadurecem, mais reuso de padrões de projeto vão atenuar a necessidade de testar completamente sistemas OO. A verdade é exatamente o contrário, Binder (1994) discute isso quando afirma:

Cada reuso é um novo contexto de utilização e o re-teste é prudente. Parece provável que mais, ao invés de menos, testes vão ser necessários para alcançar alta confiabilidade em sistemas orientados a objetos.

1.2.1 - Estratégias de teste OO

Iniciando-se com o teste em pequena escala e prosseguindo para o teste em larga escala, garantindo que todos os erros nos requisitos sejam descobertos. Como nos testes de *software* não orientados a objetos, os testes OO partem do teste de unidade, rumo ao teste de integração, validação e o de sistema.

Após uma seqüência de testes em unidades é gerada uma estrutura a partir da união das mesmas, enquanto que são realizados vários testes para verificação de falhas devido à interligação entre os módulos e a eventual adição de novas unidades. Finalmente o teste é feito em todo o sistema para assegurar que não há falhas (PRESSMAN, 2000).

1.2.2 - Teste de unidade no contexto OO

No contexto OO o teste de unidade muda um pouco em relação ao teste convencional de unidade, pois a menor unidade testável é o método ou objeto encapsulado. Acompanhando

essa mudança destacamos que uma classe pode ter varias operações e varias operações pode utilizar-se de uma classe em comum.

Pode-se descrever um exemplo, considere uma hierarquia de classes na qual uma operação X é definida para a superclasse e herdada por varias subclasses. Cada subclasse usa a operação X, mas ela é aplicada no conjunto dos atributos e operações privadas que foram definidas para a subclasse. Como o contexto no qual a operação X é usada varia de modo sutil, torna-se necessário testar a operação X no conjunto de cada uma das subclasses.

Isso significa que testar a operação X no vazio (a abordagem de teste de unidade tradicional) não é efetivo no contexto da orientação a objetos (PRESSMAN, 2000). O teste intra-método é equivalente ao teste de unidade para programas procedimentais (HARROLD e ROTHERMEL, 1994).

1.2.3 - Teste de integração no contexto OO

Na orientação a objetos métodos de mesma classe interagem entre si para desempenhar funções especificas, dando o sentido de integração entre os métodos a serem testados, tal seqüência denomina-se teste de integração ou inter-método conforme Harrold e Rothermel (1994).

A estratégia do teste de integração OO objetiva grupos de classes que colaboram ou se comunicam de algum modo.

Harrold e Rothermel (1994) definem ainda outros dois tipos de teste de integração para programas OO: teste intra-classe e teste inter-classe. Intra-classe testa-se as influências mútuas entre os métodos compartilhados fazendo invocações a esses métodos em seqüências diferentes. Procedimentos que efetuados com o intuito de encontrarem chamadas a métodos

que levem o objeto ao vazio. Inter-classe usa-se do mesmo tipo de invocações a métodos em seqüências diferentes, mas não necessariamente os métodos públicos devem estar na mesma classe. Contudo depois de efetuados procedimentos dos testes de unidade e integração pode-se iniciar o teste de sistema que não se diferencia sobre o teste procedimental.

1.2.4 - Teste de validação no contexto OO

A validação do *Software* OO focaliza as ações dos usuários – notadas e a saída do sistema usuário reconhecido.

Os testes convencionais de caixa-preta também podem ser usados como guias para a validação derivando de um modelo objeto – relacionamento e do diagrama de fluxos criados como parte de OOA.

1.2.5 - Projeto de casos de teste para *Software* OO

Os métodos de projeto de casos de teste para *Software* OO estão ainda em evolução, mas conforme Berard (1993) pode-se seguir uma abordagem geral para o projeto de casos de teste OO:

1) Identificar de modo único e associar explicitamente com a classe a ser testada com cada caso de teste;

2) Declarar as finalidades do teste;

3) Desenvolver uma lista de operações para cada teste, contendo:

3.1) Criação de uma listagem dos estados especificados para o objeto a ser testado;

3.2) Uma lista de mensagens e operações que serão exercitadas em conseqüência do teste.

3.3) Documentação das exceções que poderão vir a acontecer mediante testes ao objeto.

3.4) Lista das condições externas, isto é, variações do ambiente externo ao *software*, que devem existir.

3.5) Informações adicionais que irão ajudar no entendimento ou implementação do teste.

Diferente do projeto de casos de teste convencional, que é induzido por uma visão do *software* de entrada, processamento e saída ou pelos detalhes, seqüências de módulos individuais, o teste OO está direcionado para o projeto de seqüências apropriadas de operações, para exercitar os estados de uma classe.

1.3 - Técnicas de Teste Estrutural Baseada em Fluxo de Dados

Associa-se ao grafo de fluxo de controle informações sobre o fluxo de dados do programa, isto é, explora associações entre pontos do programa em que é atribuído um valor a uma variável e pontos em que esse valor é utilizado (MALDONADO, 1991).

Com a busca das informações sobre o fluxo de dados o teste visa identificar atribuições e utilizações das variáveis através do programa, gerando componentes elementares a serem exercitados pelo testador (SPOTO, 1995).

Método de teste que define os caminhos a serem percorridos em um programa a partir das localizações das definições e usos das variáveis no mesmo. Tendo identificado as definições e usos das variáveis, são estabelecidas associações definição-uso (ou associação DU), e a estratégia de teste de fluxo de dados exige que cada associação DU seja coberta pelo menos uma vez (PRESSMAN, 2005).

A notação utilizada para representar o Grafo de Fluxo de Dados é a mesma utilizada para representar Grafo de Fluxo de Controle mostrado na Figura 1.1.

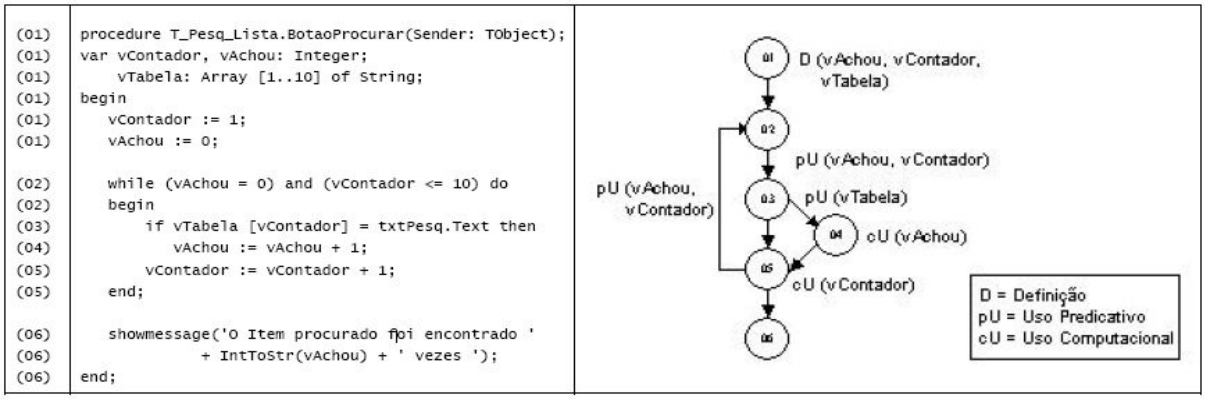

Figura 1.1 Grafo de Fluxo de Dados – Extraído de (GONÇALVES, 2003).

1.3.1 - Técnicas de Teste Estrutural Baseada em Fluxo de Dados – Programas Orientado a Objetos

A técnica estrutural de teste de programas demonstra uma série de limitações e desvantagens, tanto no processo de validação de *software* quanto na determinação de caminhos e associações não executáveis (HOWDEN, 1987; FRANKL, 1987; NTAFOS, 1988; RAPPS e WEYUKER, 1985, MALDONADO, 1991). Aspectos esses que geram sérias limitações na automatização do processo de validação de *software* (MALDONADO, 1991). Independente das desvantagens descritas, essa técnica é vista como complemento à técnica

funcional (PRESSMAN, 2001), as informações extraídas pela aplicação desses critérios têm sido consideradas acentuadas para as operações de manutenção, depuração e confiabilidade de *software* (OSTRAND e WEYUKER, 1988; HARTMANN e ROBSON, 1990; PRESSMAN, 2001; VEEVERS e MARSHALL, 1994; VARADAN, 1995; HARROLD, 2000).

Em meio dos critérios de fluxo de dados, destacam-se os critérios de Rapps e Weyuker (RAPPS e WEYUKER, 1982, 1985), inseridos nos anos 80. Para derivar os requisitos de teste exigidos por esses critérios é necessário adicionar ao grafo de programa informações sobre fluxo de dados, caracterizando o Grafo Def-Uso (*Def-Use Graph*) descrito por Rapps e Weyuker (RAPPS e WEYUKER, 1982, 1985). Em tal grafo são mineradas as associações entre o uso das variáveis determinadas, os caminhos a serem exercitados e a definição. Quando é usada em computação uma variável, diz-se que seu uso é computacional (c-uso); quando usada em condição, seu uso é predicativo (p-uso). Alguns critérios desta classe são: *todas-definições* (*todas-def*), *todos-usos*, *todos-du-caminhos* e *todos-p-usos*.

Entre o final da década de 80 e início dos anos 90, foi introduzida a família de critérios *Potenciais-Usos* e a correspondente família de critérios Executáveis por Maldonado, obtida pela eliminação dos caminhos e associações não executáveis (MALDONADO et al., 1988; MALDONADO, 1991). Esses são critérios de fluxo de dados e baseiam-se nas associações entre uma definição de uma variável e os seus possíveis subseqüentes usos para a derivação de casos de teste. De maneira parecida aos demais critérios de fluxo de dados, os Potenciais-Uso podem utilizar o Grafo Def-Uso como base para o estabelecimento dos requisitos de teste. Na verdade, basta estender o grafo de programa para que cada nó do grafo passe a conter somente informações a respeito das definições que ocorrem em cada um.

Na Figura 1.2 é mostrada a hierarquia formada pelos critérios de Rapps e Weyuker (1982, 1985) e Maldonado (1991). Nota-se que alguns obstáculos são impostos para que se

possa demonstrar a inclusão. Contudo, nem sempre esses obstáculos correspondem à realidade.

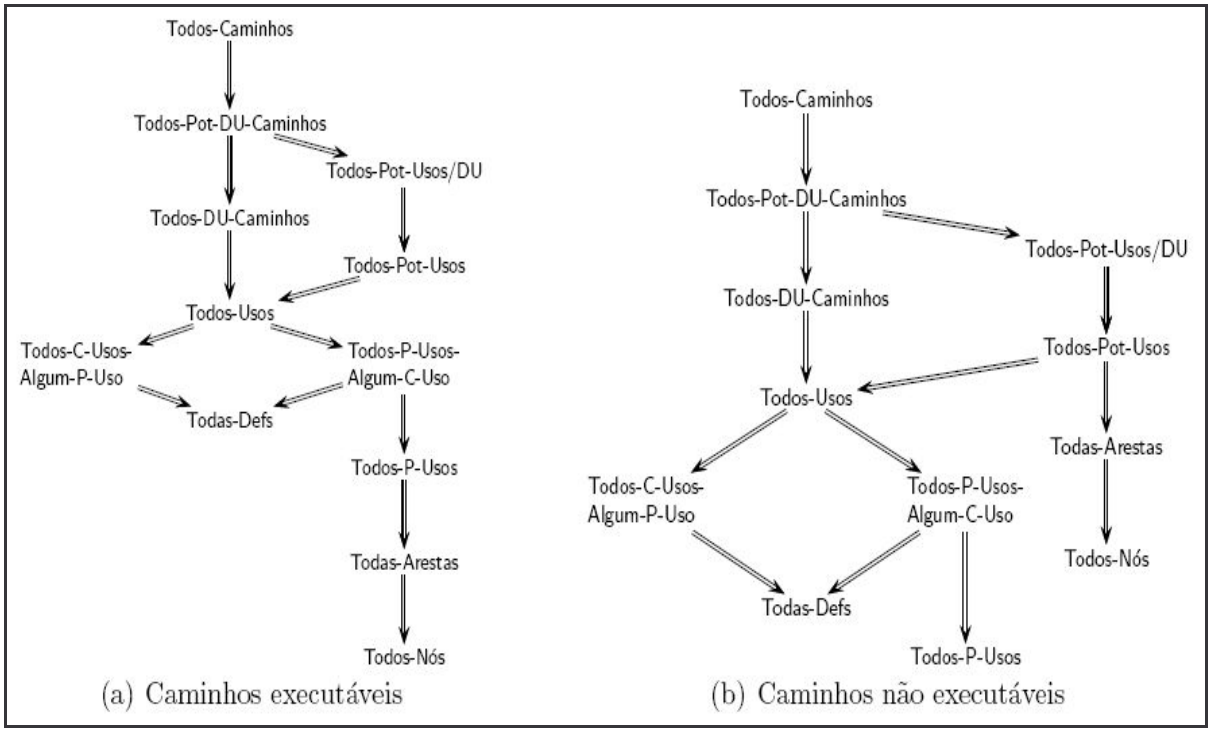

Figura 1.2 – Hierarquia dos critérios de Rapps e Weyuker (1982, 1985) e Potenciais Usos de Maldonado (1991).

Horgan e London (1990) mostram que se o programa em teste não atende à restrição de que cada comando de decisão faz referência a no mínimo uma variável – o que é indiferente, por exemplo, na Linguagem C – portanto a situação anteriormente mostrada na Figura 1.2(a) não necessariamente é verdadeira e o critério *todos-p-usos*, por exemplo, deixa de inserir o critério *todas-arestas*. Uma observação feita por Frankl e Weyuker (1988) que, na verdade, quando da presença de caminhos não executáveis, nenhum dos critérios de Rapps e Weyuker (1982, 1985) insere o critério *todas-arestas*, sendo então uma esperada propriedade de um bom critério de teste.

1.3.2 - Modelos de Fluxo de Dados

Aqui será descrito um dos modelos de fluxo de dados: Modelos às Linguagens OO.

1.3.3 - Modelos às Linguagens OO

Harrold e Rothermel (1994) dissolveram o teste de fluxo de dados para o teste de classes. Comenta que os critérios de fluxo de dados destinados ao teste de programas procedimentais podem ser utilizados tanto para o teste de métodos individuais quanto para o teste de métodos que interagem entre si dentro de uma mesma classe. No entanto, esses critérios não consideram interações de fluxo de dados quando os usuários de uma classe invocam seqüência de métodos em uma ordem arbitrária (RAPPS e WEYUKER, 1985; FRANKL e WEYUKER, 1988; HARROLD e SOFFA, 1989).

Para viabilizar o teste de fluxo de dados nos níveis intra-método, inter-método e intra-classe, foram propostas as seguintes representações de programa:

- grafo de chamadas de classe (*class call graph*);
- grafo de fluxo de controle de classe (CCFG – *class control flow graph*);
- e o *framed CCFG.*

Com base nessas representações, os três níveis de teste foram considerados (HARROLD e ROTHERMEL, 1994):

- Teste Intra-Método testa métodos individualmente, equivalente ao teste de unidade dos programas procedimentais;

- Teste Inter-Método testa métodos públicos em conjunto a outros métodos dentro da mesma classe, equivalente ao teste de integração dos programas procedimentais;
- Teste Intra-Classe testa a interação entre métodos públicos quando são chamados em seqüências diferentes. Usuários de uma classe podem invocar seqüências de métodos em ordem não definida, o teste intra-classe serve para aumentar a confiança que essas diferentes seqüências de invocação não colocam a classe em estado inconsistente. Somente um subconjunto dessas seqüências pode ser testado por ser infinita a seqüência de todas as possíveis invocações.

2 – *WORKFLOW*– Conceitos e Terminologias

Um *workflow* é definido como uma coleção de tarefas organizadas para realizar um processo, quase sempre de negócio. Essas tarefas podem ser executadas por um ou mais sistemas de computador, por um ou mais agentes humanos, ou então por uma combinação destes. A ordem de execução e as condições pela 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.

Outro conceito importante é o de Sistema de Gerenciamento de *Workflow* (WfMS - *Workflow Management System*). O WfMS supri a automação procedural de um processo de negócio gerenciando a seqüência de atividades de trabalho e chamando/invocando os recursos humanos e/ou eletrônicos apropriados associados com os vários passos das atividades.

Por definição, podemos dizer que WfMS é um sistema que define, gerencia e executa completamente *WorkFlows* através da execução de *software* cuja ordem de atividades é dirigida por uma representação da lógica do *WorkFlow* no computador. As ferramentas de WfMS previnem as pessoas para não esquecerem coisas. Uma vez que um processo é definido, um WfMS certifica-se de que as atividades ocorram numa seqüência própria e que os usuários sejam informados para que possam executar suas tarefas. Por exemplo, em vez de confiar em Carlos para contar a Ana e Fernando que sua parte no trabalho está pronta e eles podem começar a executar as suas respectivas atividades, o WfMS faz uma notificação automática e imediatamente.

Também é possível estabelecer uma analogia bastante próxima entre a função de um WfMS e a de um sistema gerenciador de banco de dados (SGBD): enquanto um exerce controle sobre dados, o outro exerce controle sobre processos.

No nível mais alto, todos os sistemas WfMS podem ser caracterizados como suporte em três áreas funcionais:

1. Funções de tempo de construção (*build-time functions*): preocupam-se com a definição, e possível modelagem, do processo de *WorkFlow* e suas atividades constituintes;
2. Funções de controle em tempo de execução (*run-time control functions*): preocupam-se com gerenciamento de processos de *WorkFlow* em um ambiente operacional e seqüencial de várias atividades para serem manuseadas como parte de cada processo;
3. Interações em tempo de execução com usuários e ferramentas para processamento dos vários passos das atividades.

2.1 - Vantagens do uso de *WorkFlows*

Os benefícios de *WorkFlow* eletrônico são vários, entre eles (TURNER, 1998):

- Eliminação do incômodo e do lixo dos produtos de papel;
- Simplificação dos formulários previstos;
- Acesso remoto;
- Arquivamento e recuperação de informações simplificadas;
- Habilidade de rapidamente trilhar as informações submetidas;
- Possibilidade de saber os responsáveis de cada tarefa do processo;
- Aumento no tempo de linhas de informação.

2.2 - Funcionalidades mais comuns dos WfMS

Os sistemas de *WorkFlow* disponíveis no mercado possuem um conjunto relativamente comum de funcionalidades. As principais são (KHOSAFIAN, 1995; SILVER, 1995):

- Roteamento de trabalho - predefine a seqüência em que as atividades serão executadas, podendo ser baseado em respostas e em regras;
- Invocação automática de aplicativos - o aplicativo adequado para a realização da tarefa pode ser invocado automaticamente, através do WfMS;
- Distribuição dinâmica de trabalho - determinar qual participante irá executar a tarefa;
- Priorização de trabalho - a maioria dos sistemas de *WorkFlow* permite que a prioridade de uma instância seja alterada, em geral por um usuário 'administrador';
- Acompanhamento do trabalho - capacidade de acompanhar uma determinada instância de *WorkFlow* e descobrir imediatamente seu status atual de processamento, sob a responsabilidade de quem está no momento, e quanto tempo ela está esperando na atividade atual;
- Geração de dados estratégicos - através do armazenamento de certos atributos de cada instância de *WorkFlow* executada, pode-se criar uma base de dados que reflete a eficiência e a eficácia dos processos atualmente desempenhados pela organização.

2.3 - Distribuição e *WorkFlow*

A habilidade de distribuir tarefas e informação entre os participantes é uma característica maior distinguindo da infra-estrutura da execução do *WorkFlow*. A função de distribuição pode operar em uma variedade de níveis (grupo de trabalhos a interorganizações) dependendo do escopo dos *WorkFlows* podendo usar uma variedade de mecanismos de comunicações (e-mail, troca de mensagens, tecnologia de objetos distribuídos,...). Uma visão alternativa da arquitetura de *WorkFlow* enfatizada nesse aspecto de distribuição é mostrada na Figura 2.1.

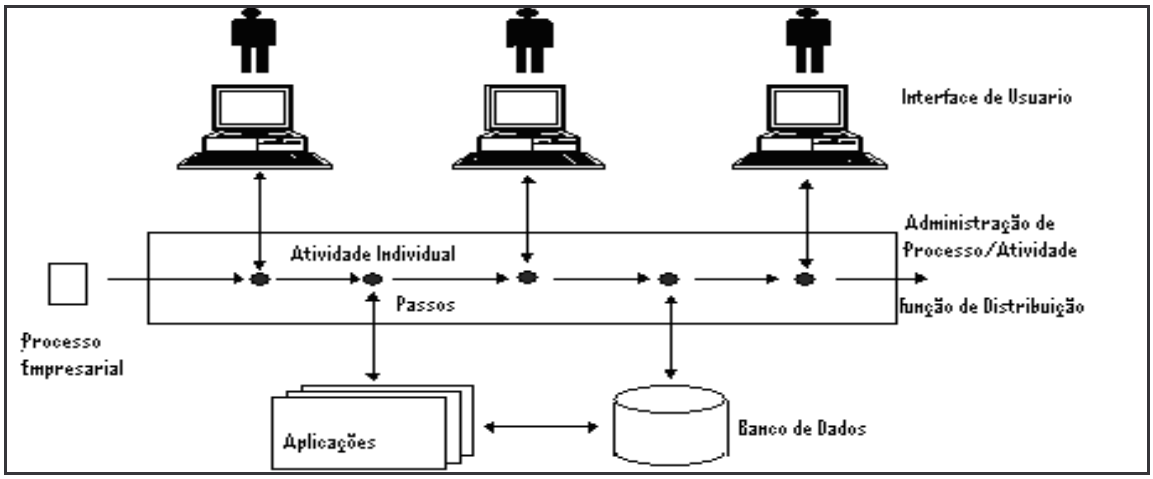

Figura 2.1 - Distribuição dentro do serviço de *WorkFlow*

O serviço de *WorkFlow* é mostrado como o coração da função de infra-estrutura com interfaces para usuários e aplicações distribuídas através do domínio do *WorkFlow*. Cada uma dessas interfaces é um ponto de integração entre o serviço de *WorkFlow* e outras infra-estruturas ou componentes de aplicações. O fluxo do trabalho pode envolver a transferência de tarefas entre diferentes vendedores de produtos de *WorkFlow* para habilitar partes diferentes do processo de negócio a ser desempenhado em plataformas diferentes ou sub-redes usando produtos particulares para aquele estágio do processo.

2.4 - Categorias de *WorkFlow*

Os produtos de *WorkFlow* podem ser divididos em três categorias gerais: (ATLEE, 1997).

I. *Document routing*: estabelece fluxo de informação e faz o roteamento dos mesmos. Também identificados como administrativos envolve processos repetitivos com regras de coordenação de tarefas simples, tal como roteamento de um relatório de despesa ou requisição de viagem através de um processo de autorização. A ordenação e coordenação de tarefas em *workflow* administrativo podem ser automatizadas (SBBD, 1999).

Na Figura 2.2 é demonstrado um modelo de *workflow* administrativo de revisão de artigos onde é possível destacar os processos de revisão tornando-se repetitivos, pois os revisores não tomam decisões em conjunto, cada revisor gera sua revisão e a expõe ficando por conta do editor tirar uma solução final.

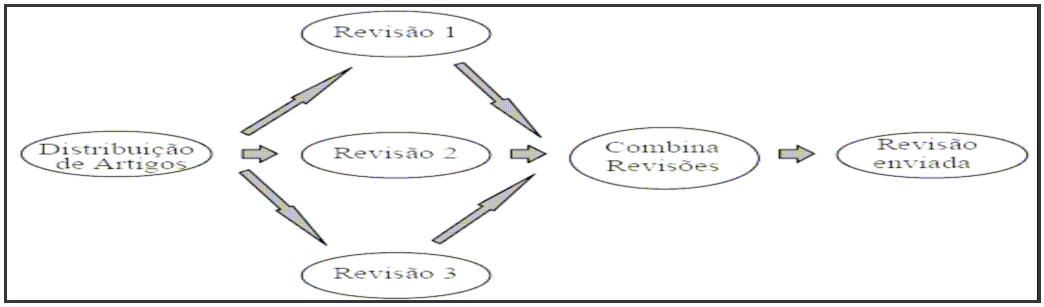

Figura 2.2 – *Workflow* administrativo de revisão de artigos. – Extraído de (SBBD,1999).

II. *Ad-hoc*: ferramentas de *groupware.* Não existe uma estrutura pré-definida para o processo, ou esta estrutura pode ser modificada em tempo de execução. Fornece o

gerenciamento de *WorkFlow* através de templates ou formulários baseados em mensagens. Envolvem coordenação manual, elaboração ou co-decisão. *Workflos ad-hoc* envolvem pequenos grupos de profissionais que têm a intenção de apoiar atividades que requerem uma solução rápida (SBBD, 1999).

Na Figura 2.3 esta sendo ilustrado um sistema de revisão de artigos usando o modelo de *workflow ad-hoc* demonstrando que há uma negociação para seleção de revisores caracterizando a colaboração entre os revisores para produção de uma revisão agrupada.

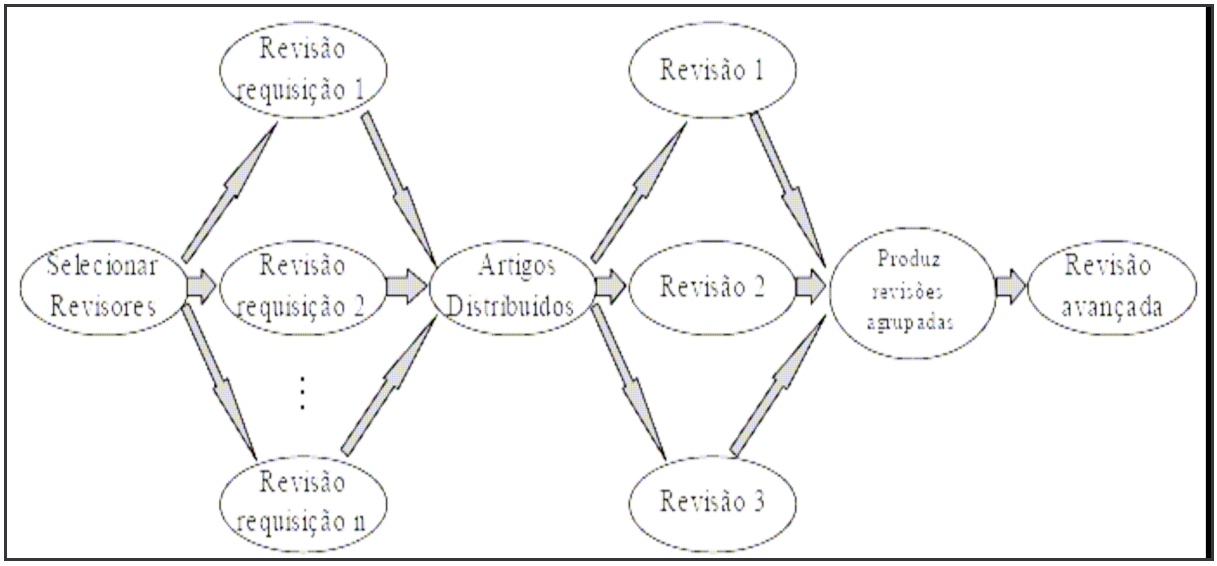

Figura 2.3 – *Workflow ad-hoc* de revisão de artigos. – Extraído de (SBBD, 1999).

III. Produção de processo de negócios: sistema para definir processos de negócios e implementação dos mesmos em *software*. Podem ser automatizados nas ordenações e coordenações de tarefas. Algumas diferenças em relação ao *Workflow ad-hoc* ou administrativo:

- Interação do sistema de informação como os processos de negócio;
- Uso de executores de tarefas automatizados, “não pessoas” (SBBD, 1999).

Um modelo ilustrando as características do *workflow* de produção é demonstrado na Figura 2.4.

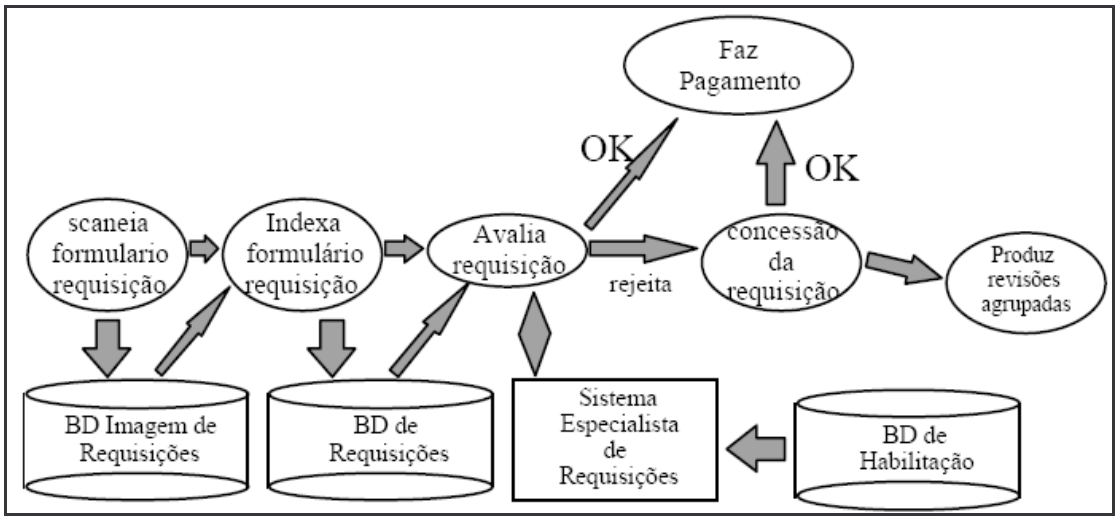

Figura 2.4 – *Workflow* de produção para o "Processo de requisição de seguro". – Extraído de (SBBD, 1999).

2.5 - Criando um *WorkFlow*

Um processo de *WorkFlow* pode ser criado como segue:

- Define-se uma atividade ou tarefa que um grupo de trabalho precisa realizar e as regras de serviço que gerenciarão a atividade;
- Divide-se a tarefa em sub-tarefas (passos). Cada passo representa uma lista bem definida de coisas que são realizadas por um indivíduo e que são feitas logicamente juntas. Uma tarefa pode ser quebrada em passos de maneiras diferentes. Nesse ponto, é exigido um julgamento do serviço para decidir onde dividir uma tarefa;

- Decide-se o conjunto de habilidades para realizar cada passo. Isso irá especificar as funções ou indivíduos de trabalho que podem ser chamados para realizar tal passo;
- Decide-se a seqüência em que cada passo deve ser realizado;
- Se algum dos passos é realizado em uma base condicional, identificam-se esses passos e definem-se as condições;
- Projeta-se um mapa do *WorkFlow* que identifica os passos e a seqüência, ou "fluxo" em que os passos devem ser realizados. Associam-se funções ou indivíduos de trabalho a cada passo;
- Criam-se os formulários, documentos e instruções que serão usados pelos indivíduos em cada passo para execução da sub-tarefa.

Como se pode perceber, *WorkFlow* envolve uma seqüência ou passos ou um processo. A tarefa "flui" de um passo para outro, sendo baseadas em regras e condições pré-definidas.

2.6 - Exemplo de *WorkFlow*

Na Figura 2.5 é mostrado um exemplo de *WorkFlow* que representa um sistema de suporte “on-line” para usuários. Este *WorkFlow*, como pode ser visto, executa acesso a bases de dados, armazenando os dados gerados em uma atividade para o uso nas atividades seguintes.

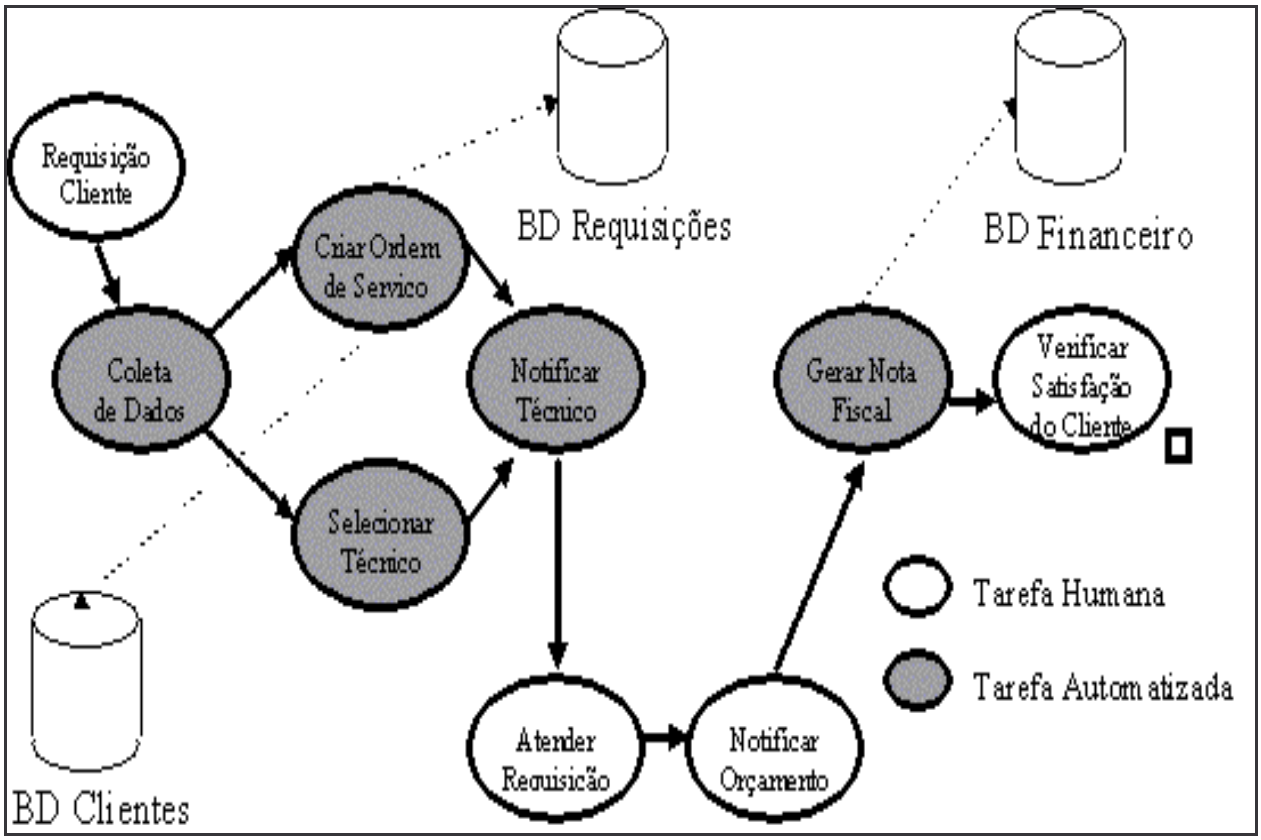

Figura 2.5 - Exemplo de *WorkFlow* representando um sistema de atendimento on-line.

Entre outros exemplos:

I. Sistema de recursos humanos que manuseiam empregados ou processos de transferência de trabalhos internos, coordenando as reclamações dos empregadores, gerenciando a partida de empregados e assim por diante;

II. Gerenciamento de ciclos de vendas e processamento de ordem on-line de compras;

III. Reenvio de informações que tenham sido passadas por fax, gravadas em fitas ou necessitam de intervenção manual para ligar dois formatos diferentes;

IV. Coordenação de agenciamento de viagens que incluem autorização gerencial.

2.7 - *WorkFlow* de Teste

Os *WorkFlows* de teste são usados para modelagem dos processos de testes de *software*. Com o uso dos *WorkFlows* de teste consegue-se atingir um nível alto na qualidade do produto final, pois testes tem tido grande repercussão no desenvolvimento de *software*s.

Deste modo é de conveniência dos desenvolvedores que sejam criadas técnicas para não somente automatizar o processo de teste, e sim melhorá-lo, de tal forma a chegar aos resultados esperados mais facilmente e com grande qualidade.

Nos tópicos seguintes serão relacionados dois modelos de *WorkFlow* de teste com suas principais características conforme análise comparativa baseada às práticas do CMMI (*Capability Maturity Model® Integration*).

2.7.1 - *WorkFlow* de Testes do RUP

RUP (*Rational Unified Process*) criado pela *Rational Software Corporation* é um método de projeto de *software*. No RUP são discriminados métodos para a construção do *software* usando métodos já testados por grupos desenvolvedores de *software*. RUP é um método de projeto indicado para grandes centros de desenvolvimento de *software* onde a necessidade e as dificuldades de gerenciamento são maiores, por essas características é tratado como um processo pesado.

O RUP é um processo baseado em ciclos, incremental, iterativo e analítico. Quando do termino de cada ciclo haverá um produto de *software*, sendo construído após passar por quatro fases: Iniciação, Elaboração, Construção e Transição conforme ilustrado na Figura 2.6.

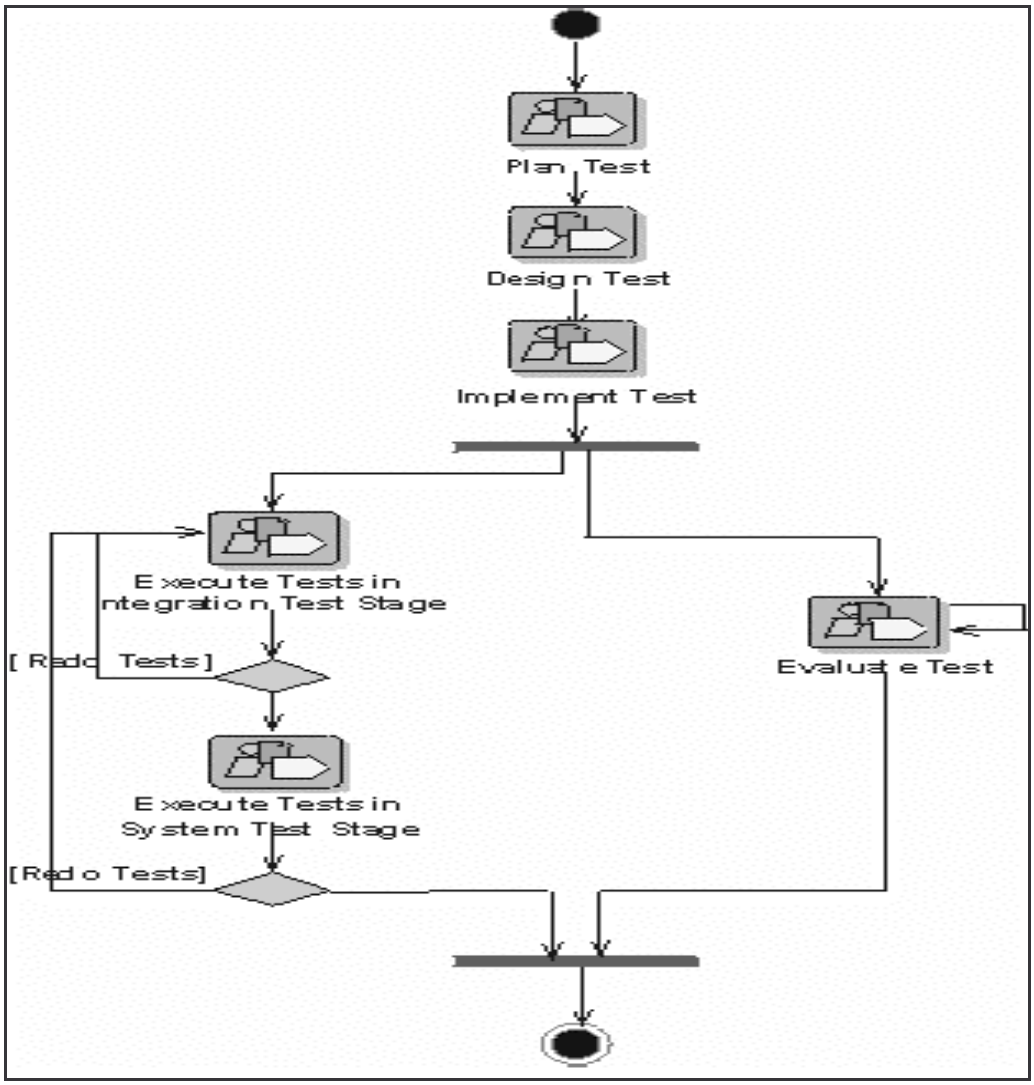

Figura 2.6 *WorkFlow* para testes de *Software* do RUP, extraído de (LEEDS, 2004; MATOS, 2004).

Em sua estrutura, o RUP possui um *WorkFlow* para cada etapa do ciclo de desenvolvimento:

- Modelagem do negócio;
- Elicitação e análise dos requisitos de *software;*
- Análise e projeto;
- Implementação;
- Teste;
- Implantação;
- Gerência de configuração e de mudanças;

- Gerência do projeto;
- Gerência do ambiente.

Aqui serão relatadas as características com foco no *WorkFlow* de teste do RUP, cujo ideal é promover o teste de *software* de forma controlada, ou seja, gerenciada. Para tal propósito deve-se seguir uma seqüência, verificar se os requisitos foram corretamente implementados, a interação entre os objetos implementados, a integração entre os componentes de *software*, e identificar defeitos antes da implantação do *software* (MASSONI, 1999).

O *WorkFlow* para testes do RUP utiliza-se de duas técnicas:

- Plano de testes: planejamento de todo o processo de testes, incluindo análise de riscos e definição de responsabilidades.
- Casos de teste: conjunto de dados desenvolvidos para um teste especifico contendo condições de execução e os resultados esperados (LEEDS, 2004; MATOS, 2004).

Por três diferentes dimensões pode ser visto a atividade de teste no RUP:

- Qualidade;
- Estágio de teste;
- Tipo de teste.

Com relação à qualidade, os aspectos de confiabilidade e performance precisam ser assegurados.

Estágios propõem uma criação de atividade de teste em fases distintas e progressivas. RUP cita quatro distintos estágios de teste:

- Teste de Unidade: teste em pequenas partes do *software*, separadamente.

- Teste de Integração: teste de subsistemas ou componentes individuais.
- Teste de Sistema: teste efetuado após a execução dos testes de unidade e integração, o teste é feito no sistema por completo.
- Teste de Aceitação: teste do *software* por usuários finais.

Existem diversos tipos de teste, descritos abaixo:

- Teste de *Benchmark*: compara os objetivos do teste com padrões conhecidos.
- Teste de Integridade: verifica confiabilidade, força e tolerância a falhas.
- Teste de Performance: testa a performance do teste em diferentes configurações.
- Teste de Stress: teste da performance em condições anormais ou extremas (LEEDS, 2004; MATOS, 2004).

2.7.2 - Atividades do *WorkFlow*

Na Figura 2.7 é mostrada uma representação em UML de um *WorkFlow* para teste de *Software* do RUP. O formato utilizado não é dado como uma boa forma de modelar um *WorkFlow*, se tomando como base o modelo de referência da WfMC, por não explicitar os atores, suas pré-condições e pós-condições, muito menos a forma de interação entre as atividades.

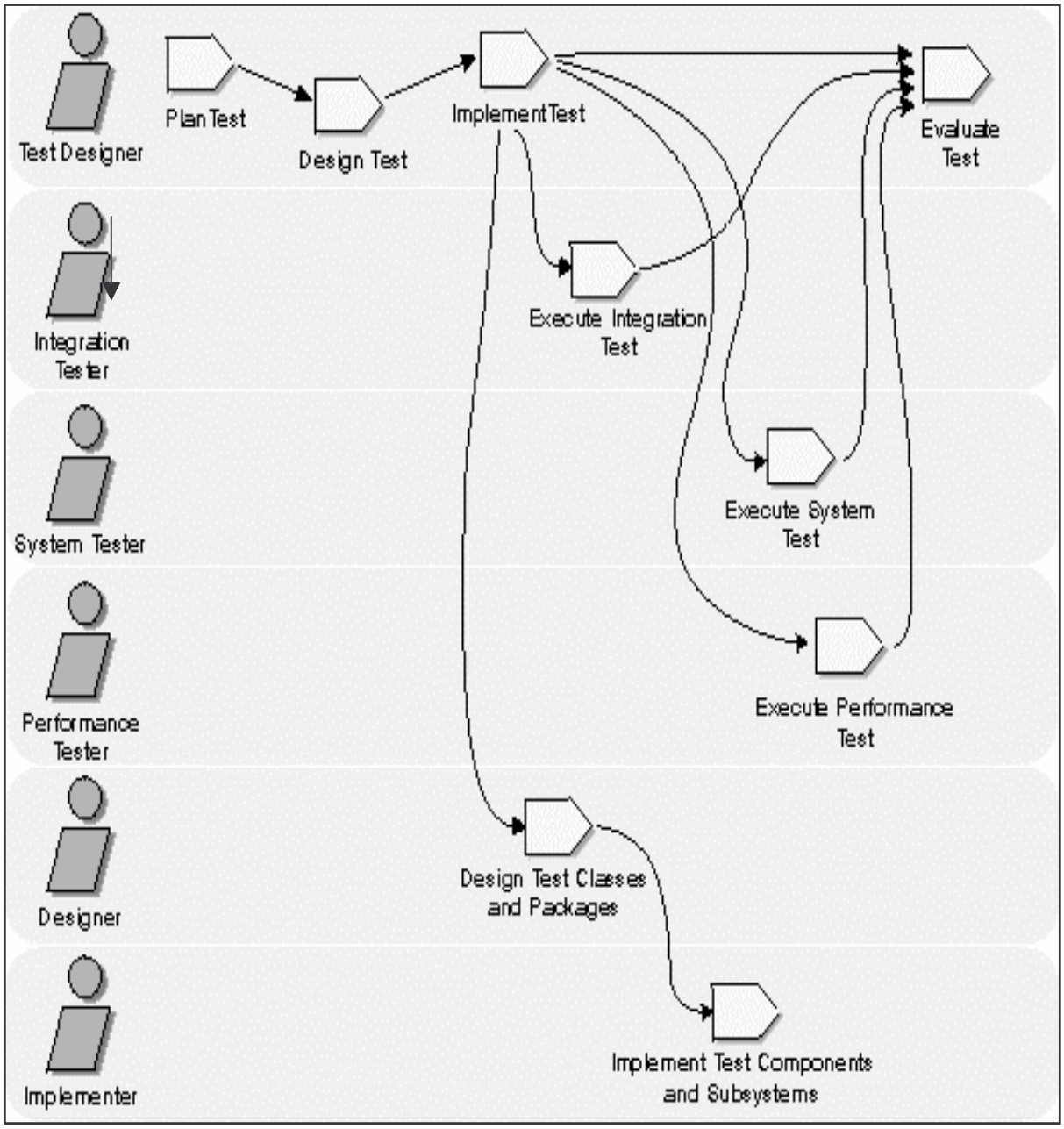

Figura 2.7 – UML - *WorkFlow* para Testes de *Software* do RUP - Extraído de (MASSONI, 1999).

No Quadro 2.1 é mostrado um resumo das atividades descritas no *workflow* para testes de *Software* do RUP, como demonstrado na Figura 2.7.

Quadro 2.1 – Atividades do *workflow* para testes de *Software* do RUP.

Planejamento – *plan test*	
Ator	Projetista de teste
Objetivo	Coletar e organizar informações sobre planejamento testes / Criar plano de testes.
Passos	▪ Identificar requisitos ▪ Avaliar riscos ▪ Desenvolver estratégia ▪ Identificar recursos ▪ Criar cronograma ▪ Preparar plano de testes
Saída	Plano de testes
Projeto – *design test*	
Ator	Projetista de teste
Objetivo	Identificar conjunto de casos de teste para cada *build* / Identificar procedimentos de teste que mostram como casos de teste são realizados
Passos	▪ Análise de carga (p/ testes de desempenho) ▪ Identificar e descrever casos de teste ▪ Identificar procedimentos de teste ▪ Revisar e avaliar cobertura
Saída	▪ Plano de testes ▪ Casos de teste ▪ Procedimentos de teste
Implementação – *Implement Test*	
Ator	Projetista de teste
Objetivo	Criar scripts de teste reutilizáveis
Passos	▪ Gravar ou programar scripts de teste ▪ Estabelecer conjuntos de dados externos
Saída	Scripts de Teste
Projeto de classes & pacotes de teste – Design Test Classes and Packages	
Ator	Projetista de teste
Objetivo	Projetar funcionalidade específica para testes
Passos	▪ Identificar pacotes e classes para testes ▪ Projetar interface para ferramentas de teste
Saída	▪ Classes projeto para teste ▪ Pacotes projeto para teste
Implementação de componentes & subsistemas de teste – Implement Test Components	
Ator	Implementador
Objetivo	Implementar funcionalidade específica para testes
Passos	▪ Implementar e testar *drivers* e *stubs* ▪ Implementar e testar interfaces para as ferramentas
Saída	▪ Componentes para teste ▪ Subsistemas para teste
Execução (Integração, Sistema, Desempenho) - Execute	
Ator	Executores de testes
Objetivo	Executar testes / Revisar resultados / Registrar defeitos
Passos	▪ Executar procedimentos de teste (ou scripts) ▪ Avaliar execução dos testes ▪ Recuperação de testes abortados ▪ Verificar resultado dos testes ▪ Investigar resultados inesperados ▪ Registrar Defeitos
Saída	Defeitos
Avaliação – *Evaluate Test*	
Ator	Projetista de testes
Objetivo	Métricas do progresso dos testes / Gerar relatório de avaliação
Passos	▪ Avaliar cobertura dos casos de teste ▪ Avaliar cobertura do código ▪ Analisar Defeitos ▪ Determinar completude dos testes e se critério de sucesso foi atingido
Saída	Relatório de avaliação dos testes

2.8 - *Workflow* de Teste de Aplicações OO na FADAT

FADAT – Ferramenta de Apoio a Documentação da Aplicação de Teste de *Software*, por Parckert (2006), com toda a dificuldade nas atividades de teste de *software*, mas com seu destaque relevante na importância para o desenvolvimento de qualquer *software*, Parckert em suas pesquisa descobriu a escassez de ferramentas de apoio e documentação na geração de *workflows* de teste de *software*, por essas particularidades surgiu a FADAT.

Uma Ferramenta de Apoio a Documentação de Teste de *Software*, ferramenta a qual foi desenvolvida na linguagem JSP disponibilizada aos usuários clientes por intermédio de *Browser* com sua estrutura de armazenamento de dados ligada ao SGBD FireBirdSQL .

Na Figura 2.8 é demonstrada a estrutura da FADAT cujas funcionalidades são descritas a seguir.

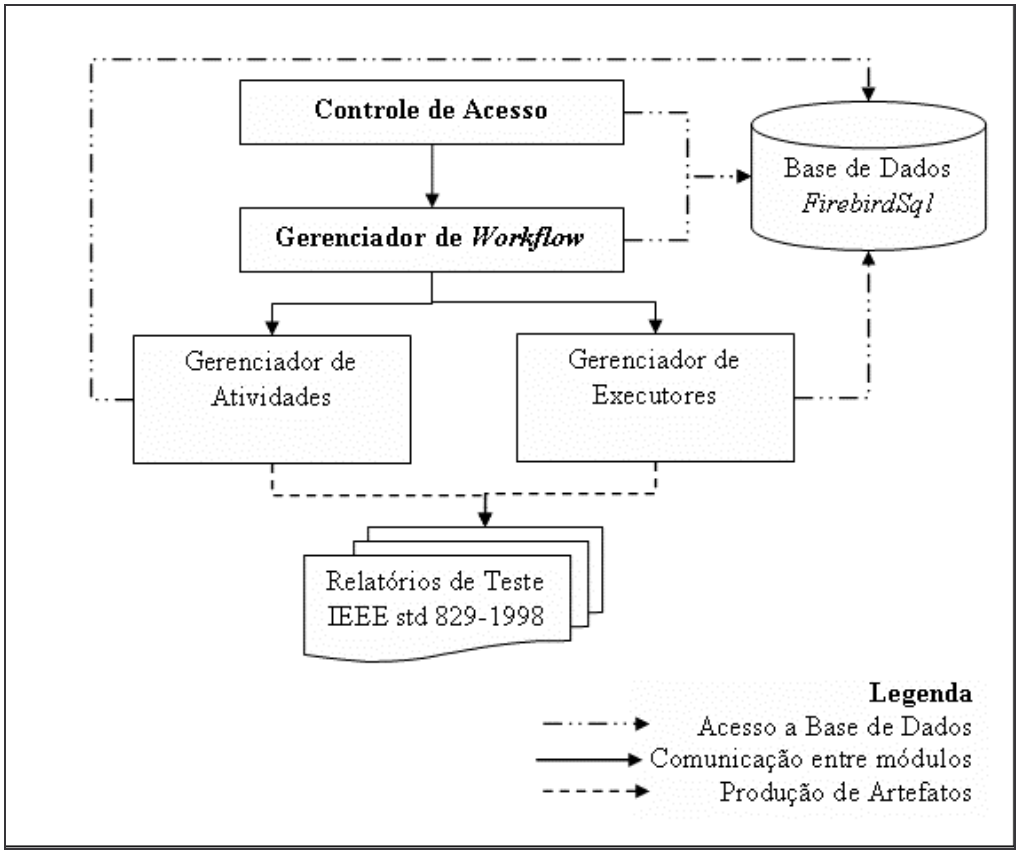

Figura 2.8 – Arquitetura da ferramenta FADAT, extraído de Parckert (2006).

Conforme visto na Figura 2.8 destacam-se quatro módulos da ferramenta os quais têm uma comunicação entre si e ao final desta comunicação o resultado final é a geração dos relatórios de teste segundo a Norma IEEE std 829 – 1998 conforme descrito no Capítulo 3. Os quatro módulos em destaques são: Módulo de Controle de Acesso, Módulo Gerenciador de *Workflow*, Módulo Gerenciador de Executores e o Módulo Gerenciador de Atividades (PARCKERT, 2006).

- **Controle de Acesso:** módulo responsável por fazer a verificação e liberação do acesso às funcionalidades da ferramenta ao usuário, após usuário informar nome de usuário e sua respectiva senha;
- **Gerenciador de *Workflow*:** responsável pelo gerenciamento dos *Workflows* cadastrados na ferramenta, suas funcionalidades são de consultar, inserir e remover os *Workflows* da ferramenta. *Workflows* só poderão ser excluídos caso não tenham nenhuma dependência;
- **Gerenciador de Executores:** módulo este que faz o gerenciamento de quais serão os executores responsáveis por cada atividade referente a um determinado *Workflow* atividade esta que só é permitida ao criador do *Workflow,* funcionalidades do executor é de inclusão, alteração e consulta de usuários por atividade;
- **Gerenciador de Atividades:** módulo responsável por gerenciar as atividades que deverão ser realizadas para a produção de um relatório de teste, tais atividades estão relacionadas à Figura 2.6.

3 - NORMAS DE TESTE

Neste Capítulo será discutido a importância e descrição dos aspectos da Norma IEEE std 829 – 1998.

3.1 - Visão geral Norma IEEE std 829 – 1998

Para um bom desenvolvimento de testes em algum *software* em especifico, por exemplo, para meios científicos, comerciais ou até mesmo militares devem ser seguidas algumas normas. A norma IEEE std 829 - 1998 é a mais usada pelos centros de normalização de *software*, ela descreve um conjunto de documentos para as atividades de teste de um produto de *software*. Ela também é definida de forma independente de técnicas, métodos, estratégias, recursos e ferramentas de teste (CRESPO *et al.*, 2004).

Alguns documentos são definidos pela norma IEEE std 829 - 1998 para garantir as tarefas de planejamento, especificação e relato de testes, 8 são eles e serão descritos abaixo:

1. **Plano de Teste** – Onde é feita a demonstração do planejamento do teste, cronograma de atividades de teste, recursos, abordagens e a caracterização das funcionalidades a serem testadas, tarefas de teste a serem realizadas e os riscos associados com os testes.

2. **Especificação de Projeto de Teste** – Aprimora o contexto do Plano de Teste e faz a identificação das características e funcionalidades que deverão ser testadas pelo projeto e por seus testes associados.

Na fatia de especificação de teste a norma determina 3 documentos:

3. **Especificação de Caso de Teste** – Faz a definição de um caso de teste, incluindo dados de entrada e resultados esperados e condições gerais para a execução do teste.

4. **Especificação de Procedimento de Teste –** Caracteriza os passos para executar um grupo de casos de teste.

Os relatórios de teste são tomados por 4 documentos:

5. **Diário de Teste** – Apresenta marcas com o passar do tempo dos detalhes relevantes relacionados com a execução dos testes.

6. **Relatório de Incidente de Teste** – Relaciona eventos que ocorra durante a atividade de teste e que necessitem de revisão posteriormente.

7. **Relatório - Resumo de Teste** – Demonstra de forma selecionada os resultados das atividades de teste associadas com uma ou mais características de projeto e fornece avaliações baseadas nesses resultados.

8. **Relatório de Encaminhamento de Item de Teste** – Seleciona os itens encaminhados para teste no caso de equipes distintas serem responsáveis pelas tarefas de desenvolvimento e de teste.

Na Figura 3.1 é ilustrada a organização dos documentos de teste de *software* criados pela Norma IEEE std 829 – 1998.

.

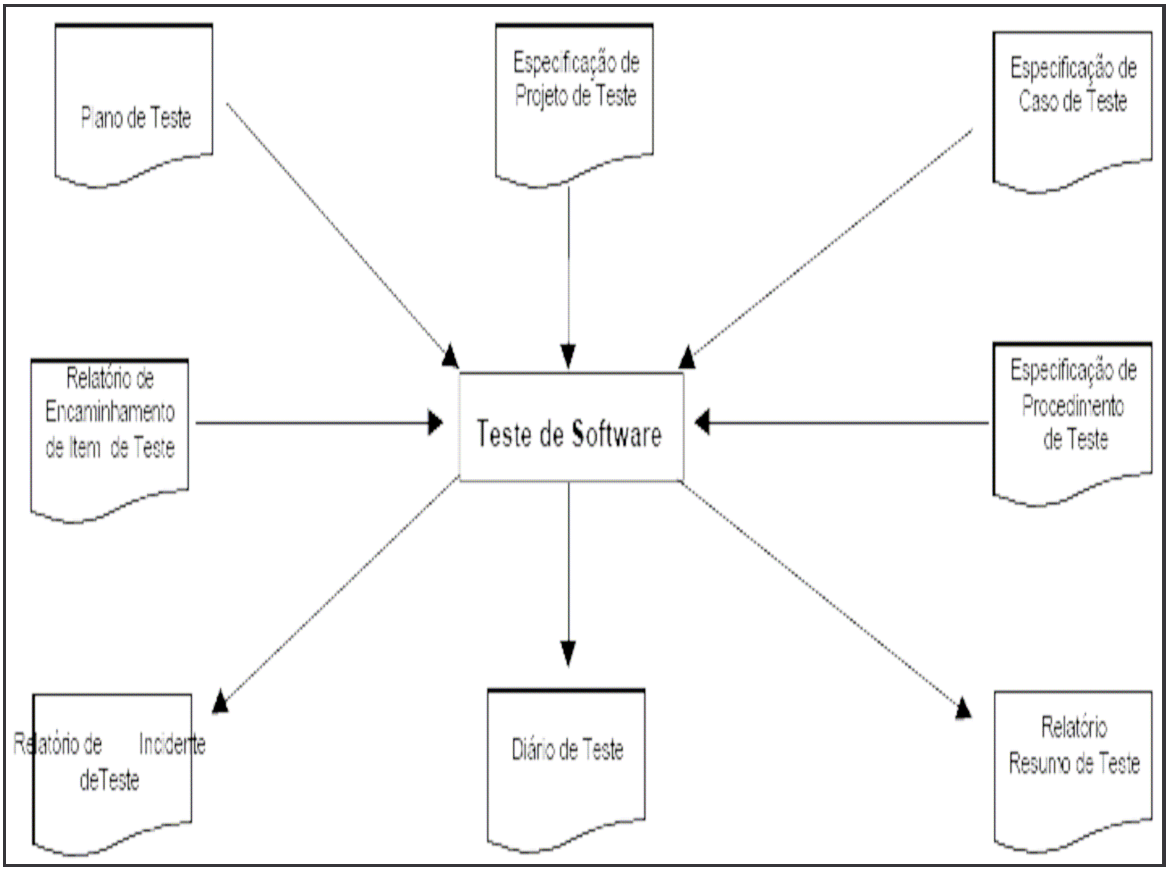

Figura 3.1 – Organização dos Documentos de Teste de *Software*, Extraído de (CRESPO *et al.*, 2004).

Em seqüência, na Figura 3.2, são demonstrados os relacionamentos entre os documentos de teste (itens com fundo cinza não foram definidos pela Norma IEEE std 829 – 1998).

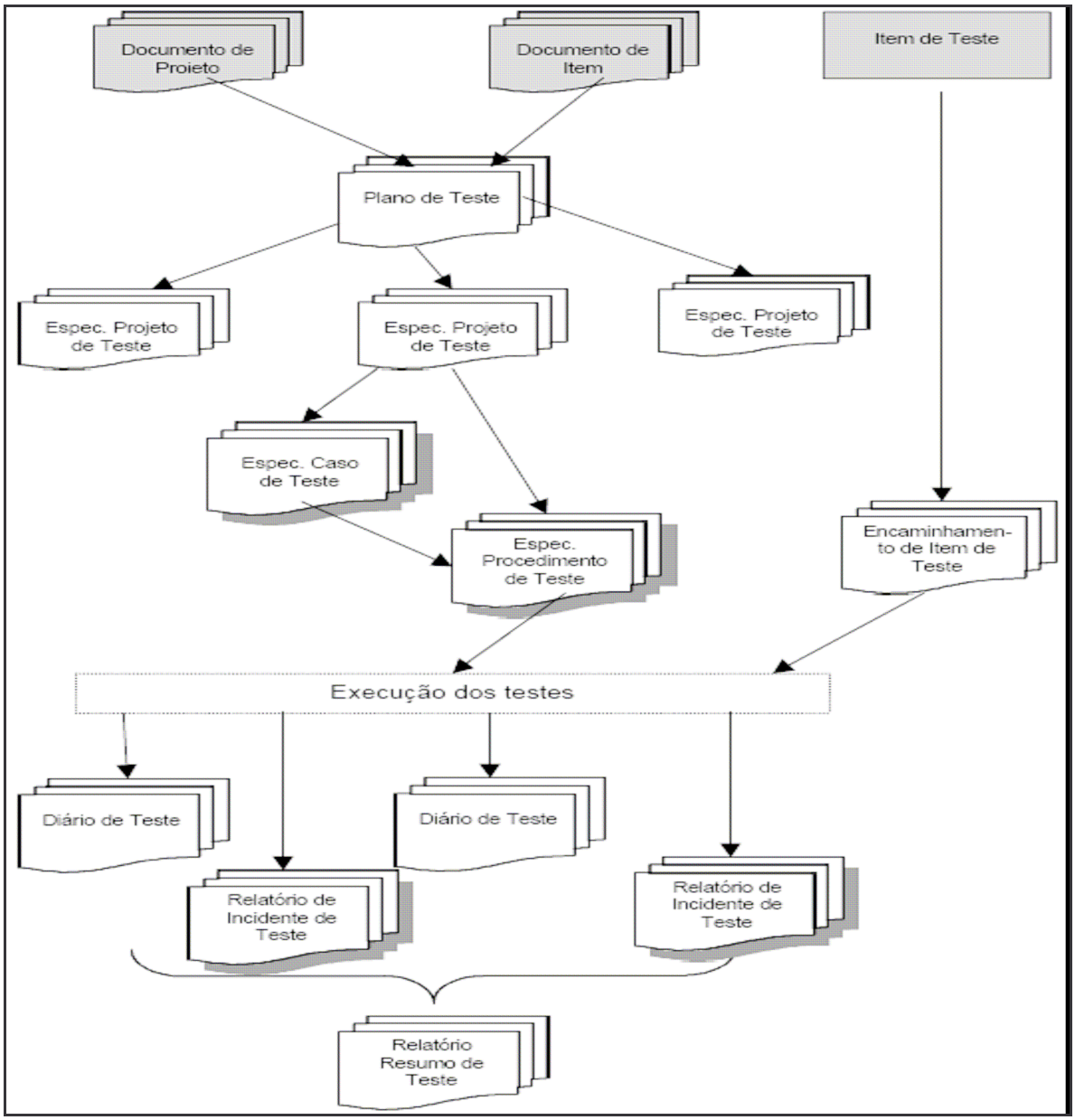

Figura 3.2 – Relacionamento entre os Documentos de Teste, Extraído de (CRESPO *et al.*, 2004).

Depois de relacionados os documentos de teste propostos pela Norma IEEE std 829 – 1998, iremos relacionar os processos necessários para a geração dos documentos de teste de *software* baseados na norma. Os processos abrangem a preparação, a execução e o registro dos resultados do teste e estão descritos segundo *Handbook for Process Management* (SNWSCSD, 1998).

Tais processos estabelecem uma visão geral e, se necessário, podem ser modificados para adequar-se às situações particulares de organização envolvidas nas atividades de teste.

Um esquema é definido para cada documento da norma, conforme a estrutura a seguir:

1. Funções e responsabilidades no processo – participantes na execução das tarefas;
2. Critérios para o início do processo – elementos e/ou condições necessárias para iniciar a execução das tarefas;
3. Entradas do processo – dados, recursos ou ferramentas necessários para a execução das tarefas;
4. Tarefas do processo – ações necessárias para produzir as saídas do processo. Para cada tarefa são identificadas suas entradas, com indicação de possíveis fontes, e as saídas produzidas. A ordem de apresentação das tarefas mão reflete necessariamente a seqüência em que devem ser executadas.
5. Saídas do processo – dados ou produtos gerados pela execução das tarefas;
6. Critérios para término do processo – elementos e/ou condições necessários para encerrar a execução das tarefas;
7. Medições do processo – medidas a serem coletadas como parte da execução das tarefas.

Dependendo do domínio da aplicação, da estratégia ou da fase de teste, os processos podem ser adaptados de modo a produzir um conjunto maior ou menor de documento, no entanto os documentos de preparação para o teste devem incluir: planejamento de teste, projeto de teste, casos de teste e procedimentos de teste.

Quando necessário, os passos ou tarefas dos processos podem ser desdobrados, com objetivo de incluir ações adicionais, que possam eventualmente, resultar em novos documentos e/ou formulários (CRESPO *et al.*, 2004).

4 – *WORKFLOW* DE TESTE PARA APLICAÇÕES OO

Neste capítulo será demonstrado um *Workflow* de Teste para aplicações OO, com base no modelo do *Workflow* de Teste do RUP descrito na Seção 2.7.1 – *Workflow* de Teste do RUP, subseqüente os passos necessários para o desenvolvimento, cujo enfoque da construção do *Workflow* está na aplicação da Técnica de Teste Estrutural, observando o fluxo de controle e o fluxo de dados em OO, no nível de Teste de Unidade – normalizados pela Norma IEEE std 829 – 1998.

Em conjunto ao desenvolvimento do *Workflow,* serão apresentados os resultados obtidos com o exemplo gerado utilizando-se de um código construído para a geração do estudo de caso, para exemplificar os procedimentos utilizados no *Workflow,* por exemplo: criação de casos de testes baseados nos critérios *todos-nós, todos-arcos* referentes ao Fluxo de Controle e *todos-usos* referente ao Fluxo de dados, intra-classe, em aplicações OO objetivando-se o teste de unidade em teste estrutural.

A geração do *Workflow* de teste será com base nos quatro passos: Planejamento, Projeção e Implementação, Execução e Avaliação dos testes.

4.1 - Plano de Teste

É o planejamento do teste, representado por um cronograma de atividades de teste, com os recursos, abordagens e a caracterização das funcionalidades a serem testadas, tarefas de teste a serem realizadas e os riscos associados com os testes. No plano de teste são gerados os elementos de teste pelos critérios de teste estrutural intra-classe, os elementos de teste são

baseados nos critérios *todos-nós, todos-arcos* referentes ao Fluxo de Controle e *todos-usos* referente ao Fluxo de dados.

Massoni (1999) descreve que mesmo que os custos e a complexidade sejam elevados, o planejar de uma metodologia e o uso de ferramentas adequadas podem vir a aumentar a produtividade e efetividade dos testes. Com tal afirmação deduz que deve-se gerar Planos de Testes eficientes que demonstrem interação entre os objetos e componentes implementados, se os requisitos foram corretamente implementados e se existe erros antes da implantação do Sistema.

Com a união das informações como o cronograma de atividades de teste, recursos, planejamento do teste é gerado o Plano de Teste, por sua vez um relatório que é o próprio Plano de Teste (MASSONI, 1999).

4.1.1 - Relatório criado pelo Plano de Teste

O relatório descreve o planejamento de todas as atividades do teste, a extensão do teste, abordagem, recursos, cronograma e a definição do ambiente operacional.

De acordo com a complexidade do produto a ser testado decide-se se será manufaturado um único plano de teste ou um plano independente para as fases de teste de unidade, integração, sistema e regressão.

O Plano de Teste identifica os itens a serem testados, em que nível deve ser testado, uma abordagem especifica para cada um dos itens, tarefas destinadas a cada atividade de teste, os responsáveis por cada atividade e seus riscos relacionados ao plano de teste.

Este documento pode ser relacionado a um amplo projeto de teste ou a um dos níveis de teste: Plano de Teste de Unidade, Plano de Teste de Integração, e Plano de Teste de Sistema (IEEE std 829 - 1998).

4.1.2 - Estrutura do Relatório do Plano de Teste

De acordo com a norma IEEE std 829 - 1998, a estrutura para o Relatório do Plano de Teste deve seguir a seqüência abaixo.

a) **Identificador do Plano de Teste:** Especificar um identificador único para este Plano de Teste;

b) **Introdução:** Apresentar um sumário dos itens de *software* e das características de *software* que devem ser testados. A necessidade de cada item de *software* e seu respectivo histórico pode ser incluída.

c) **Itens de Teste:** Identificar os Itens de Teste constando seu identificador de versão/revisão.

d) **Funcionalidades e Características do *software* que devem ser testadas:** Identificar todas as funcionalidades e características simples e combinadas do *software* que devem ser testadas.

e) **Funcionalidades e Características do *software* que não devem ser testadas:** Identificar todas as funcionalidades e características simples e combinadas do *software* que não devem ser testadas, apresentando as razões.

f) **Abordagem:** Descrever a abordagem geral do teste. Para cada grupo de funcionalidades e características simples ou combinadas do *software* especificar a abordagem que garantirá o teste adequado desse grupo de

funcionalidades e características. Especificar as principais atividades, técnicas e ferramentas que são usadas para testar os grupos designados de funcionalidades e características.

g) **Critérios de Aprovação/Reprovação de Itens:** Especificar os critérios a serem utilizados para determinar se cada item de teste foi aprovado ou reprovado no teste.

h) **Critérios de Suspensão e Requisitos para a Retomada do Teste:** Especificar os critérios adotados para suspender ao todo ou em partes a atividades de teste dos itens de teste associados a este plano.

i) **Produtos do Teste:** Identificar todos os documentos gerados pelo teste e que podem ser encaminhados para a gerência. Os dados de entrada/saída do teste devem ser identificados como produtos. Ferramentas de teste , em geral controladores de módulos e "stubs", podem também ser inseridos como produtos.

j) **Tarefas de Teste:** Identificar o conjunto de tarefas necessárias à preparação e à execução do teste. Relacionar os pré-requisitos e as dependências entre as tarefas identificando as habilidades necessárias, o responsável, o esforço previsto e a data limite para a execução das tarefas.

k) **Requisitos de Ambiente:** Especificar tanto as propriedades necessárias como as propriedades desejadas do ambiente de teste, sendo que a mesma deve conter características físicas dos recursos incluindo o *hardware*, os meios de comunicação e o *software* de sistema, o modo de uso, e quaisquer outros *software*s e suprimentos necessários para a realização do teste. Especificar também o nível de segurança que deve ser providenciado para os recursos de

teste, *software* de sistema, e componentes proprietários tais como *software*, dados e *hardware.*

l) **Responsabilidades:** Identificar os grupos responsáveis pela gerência, projeto, preparação, execução, testemunho, verificação e aprovação. Identificar também os grupos responsáveis por providenciar os itens de teste identificados © e o ambiente de teste (k).

m) **Equipe e Treinamento Necessários:** Especificar a equipe necessária para o teste identificando os níveis requeridos de competência e habilidade.

n) **Cronograma:** Elaborar um cronograma de execução das tarefas identificadas no item (j).

o) **Riscos e Contingências:** Identificar as hipóteses e suposições (pressupostos) de alto risco do plano de teste.

p) **Apêndices:** Relacionar quaisquer outros documentos eventualmente necessários à execução do teste, tais como figuras, desenhos, gráficos, etc. que não se encaixem num dos itens anteriores do plano.

q) **Aprovações:** Especificar os nomes e os cargos das pessoas que devem aprovar este plano, deixando espaço para datas e assinaturas.

Caso seja necessário, as Seções adicionais podem ser incluídas no final do documento, antes da Seção referente às Aprovações. Se todo o conteúdo ou parte de uma seção estiver em outro documento, então, poderá ser listada uma referência a esse material no lugar do conteúdo correspondente e esse material referenciado, deverá ser anexado ao Plano de Teste ou colocado à disposição dos usuários do Plano de Teste.

4.2 – Projeção e Implementação do Teste

Nesta fase é demonstrado o processo do teste a começar pela Instrumentação, em seguida a Geração dos Elementos de Teste, a execução do teste propriamente dito e por fim a avaliação dos resultados dos testes.

4.2.1 - Instrumentação

A instrumentação do programa original é feita com objetivo de detalhar o código para facilitar na execução do teste, obter informações sobre a execução dos elementos de teste para analise dos mesmos pelos diversos critérios de teste. Conforme Spoto (2000) em programas de aplicação a instrumentação é dividida em duas fases: 1) ocorre a alteração do código original acrescentando comandos e informações, após as alterações esta nova versão é chamada de unidade instrumentada para o teste de unidade; 2) nesta fase visa colher dados referentes à identificação das unidades.

Parte fundamental da instrumentação é inserção de “pontas de prova”, são utilizadas para identificar e notificar o número de cada bloco de comando, assim possibilitando identificar o caminho percorrido na execução dos casos de teste.

A *ponta de prova* é um comando de escrita do número e do nó em arquivo nomeado *Pathint*, produzindo a seqüência de execução dos nós em cada caso de teste durante o teste.

Um número identifica o módulo de programa e a unidade de programa no teste de integração, essa informação é escrita no início do arquivo *Pathint.tes*, representada pelo número muuu (m é o número do módulo e uuu o número da unidade).

Na Figura 4.1 o código instrumentado utilizado no estudo de caso, e a partir do qual serão gerados os elementos de teste e casos de teste para satisfação dos critérios mencionados.

```
        import java.util.*;
        import java.io.*;
        import java.lang.*;
        import java.awt.*;

/*1'*/  class Soma
/*1'*/  {
/*1'*/    public static float media(String n[],String p[])
/*1'*/    {
/*1'*/      float soma=0;
/*1'*/                float peso=0;

/*2'*/                for(int i=0;i<n.length;i++)
                    {
/*3'*/                  Float saux = new Float(n[i]);
/*3'*/                  Float paux = new Float(p[i]);
/*3'*/                  soma = soma + ((saux.parseFloat(n[i]))*(paux.parseFloat(p[i])));
/*3'*/                  peso = peso + (paux.parseFloat(p[i]));
                     }
/*4'*/                 return soma/peso;
        }


/*1*/   public static void main(String arg[ ])throws Exception
/*1*/   {
/*1*/     float med=0;
/*1*/                String nota[] = new String[4];
/*1*/                String peso[] = new String[4];

/*1*/                BufferedReader key = new BufferedReader(new InputStreamReader(System.in));

/*1*/                System.out.println();
/*1*/                System.out.println("     CALCULO DE MEDIA (PONDERADA) - 4 NOTAS   ");
/*1*/                System.out.println();
/*1*/                System.out.println();

/*1*/                System.out.print("DIGITE NOTA 1 : ");
/*1*/                nota[0] = key.readLine();
/*1*/                System.out.print("DIGITE PESO 1 : ");
/*1*/                peso[0] = key.readLine();

/*1*/     System.out.println();

/*1*/                System.out.print("DIGITE NOTA 2 : ");
/*1*/                nota[1] = key.readLine();
/*1*/                System.out.print("DIGITE PESO 2 : ");
/*1*/                peso[1] = key.readLine();

/*1*/     System.out.println();

/*1*/                System.out.print("DIGITE NOTA 3 : ");
/*1*/                nota[2] = key.readLine();
/*1*/                System.out.print("DIGITE PESO 3 : ");
```

```
/*1*/                peso[2] = key.readLine();

/*1*/          System.out.println();

/*1*/                System.out.print("DIGITE NOTA 4 : ");
/*1*/                nota[3] = key.readLine();
/*1*/                System.out.print("DIGITE PESO 4 : ");
/*1*/                peso[3] = key.readLine();
       ponta_de_prova(1);
/*2*/                med = media(nota,peso);

/*2*/                System.out.println();
       ponta_de_prova(2);
/*3*/                System.out.println("A MEDIA PONDERADA = "+med);
/*3*/          System.out.println();

/*3*/                if (med >= 7)
       ponta_de_prova(3);
/*4*/                  System.out.println("ALUNO APROVADO!!!");
       ponta_de_prova(4);
/*5*/                else {
/*5*/                  if (med < 5)
       ponta_de_prova(5);
/*6*/                    System.out.println("ALUNO REPROVADO!?!?");
       ponta_de_prova(6);
/*7*/             else
/*7*/                    System.out.println("ALUNO DE EXAME!?!?!");
       ponta_de_prova(7);
/*8*/           }
/*8*/
/*8*/         }
/*8*/       }
```

Figura 4.1 – Código Instrumentado utilizado no caso de teste.

Partindo-se do código instrumentado é possível criar os grafos de fluxo de controle e de dados, para facilitar o teste de unidade, abaixo o grafo de fluxo de controle na Figura 4.2 e em seguida o grafo de fluxo de dados na Figura 4.3, gerados sem o auxilio de ferramentas.

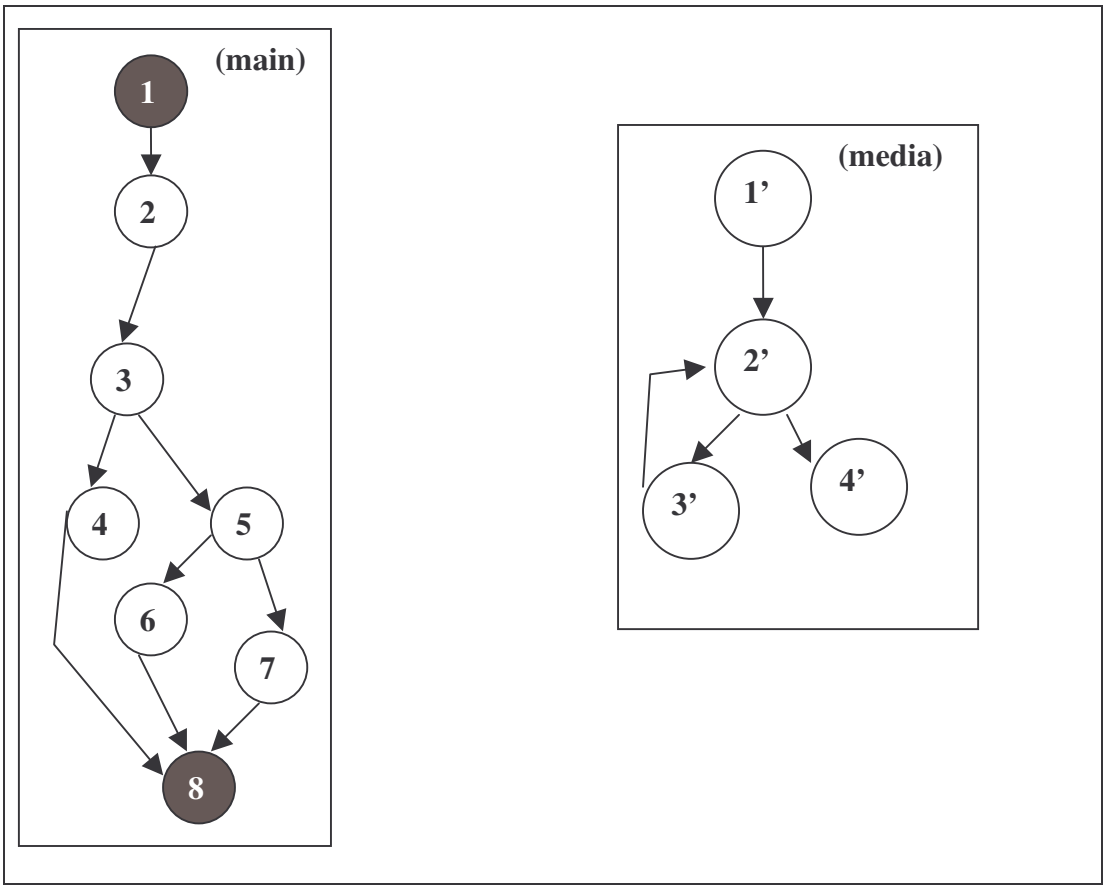

Figura 4.2 – Grafo de Fluxo de Controle à (Esq.)Função main a (Dir.) Função media.

Como pode ser observado o grafo de fluxo de controle é um grafo orientado, com um único nó de entrada (nó: 1) e um único nó de saída (nó: 8). Observando-se o código instrumentado anteriormente, à esquerda das instruções do programa estão representados os números dos nós do grafo, por exemplo, tem-se que o nó 1 representa o bloco 1, que é composto pelos comandos correspondendo ao agrupamento que foi feito para identificar cada nó, representado pelos símbolos /*01*/. Cada arco representa uma transferência de controle entre esses blocos. Com base no grafo de Fluxo de Controle são identificados os componentes (nós, arcos ou caminhos) que devem ser executados para satisfazer determinado critério, caracterizando assim o Teste Estrutural.

Os critérios de Teste Estrutural baseiam-se em diferentes tipos de componentes para determinar quais partes do programa devem ser executadas. Eles estão classificados em

critérios baseados em fluxo de controle e critérios baseados em fluxo de dados (MYERS, 1979). Dentre eles, comentam-se a seguir alguns critérios referentes às duas classes.

Critérios Baseados em Fluxo de Controle: Esses critérios usam informações do controle da execução do programa para derivar os requisitos de teste. Dessa classe, são exemplos os critérios, os quais serão dados maiores ênfases: Todos-Nós e Todos-Arcos.

O critério Todos-Nós exige que cada nó do grafo seja exercitado ao menos uma vez, o que corresponde a fazer com que cada comando do programa seja executado.

Já o critério Todos-Arcos requer que cada aresta do grafo, ou seja, cada desvio do programa seja exercitado pelo menos uma vez.

Critérios Baseados em Fluxo de Dados: Esses critérios usam informações do fluxo de dados do programa para derivar os requisitos de teste. Por isso é necessário adicionar ao grafo do programa informações sobre o fluxo de dados, gerando o chamado grafo Def-Uso, definido por Rapps & Weyuker (RAPPS & WEYUKER, 1982-1985). O grafo mostrado na figura abaixo apresenta os pontos que exibem definição e uso de variáveis. Com base nesse grafo são determinados quais caminhos devem ou não ser exercitados para atender os critérios baseados em fluxo de dados. Dessa classe, são exemplos os critérios: Todos-Usos (RAPPS & WEYUKER, 1982-1985) e Todos-Potenciais-Usos (MALDONADO, 1991).

O critério a ser adotado para o Critério Baseado em Fluxo de Controle será o Todos-Usos o qual requer que todas as associações entre uma definição de variável e seus subseqüentes usos sejam exercitadas pelos casos de teste, sendo que uma associação é estabelecida entre uma atribuição de valor a uma variável (sua definição) e um subseqüente uso dessa variável através de um caminho livre de definição, ou seja, um caminho em que a variável não seja redefinida. Seja um grafo de fluxo de controle **G** = (**N**,**E**, **s**) onde **N** representa o conjunto de nós, **E** o conjunto de arcos, e **s** o nó de entrada. Um caminho pode ser definido informalmente como sendo uma seqüência finita de nós (n1,n2,..,nk), k 2, tal que

exista um arco de ni para ni +1 para i = 1,2,.. k 1 (BARBOSA et al, 2000). Um caminho pode ser chamado de não executável quando não existe um dado de entrada que leve à sua execução.

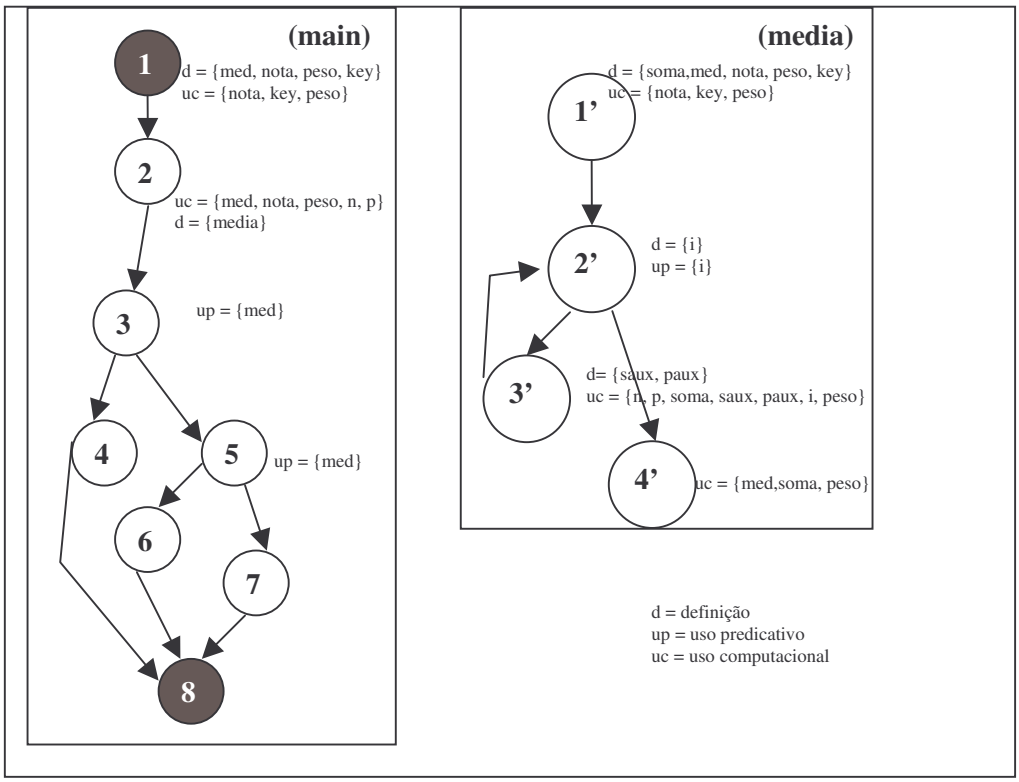

Figura 4.3 – Grafo de Fluxo de Dados com apontamentos de uso e definição.

4.2.2 – Geração dos Elementos de Teste

Abaixo segue os elementos de teste, também conhecidos como elementos requeridos, gerados com base no código instrumentado anteriormente, no modelo de associação: **Número_da_Associação** – {**No_definição_da_variavel**, (**No ou Arco_do_uso_da_variavel**), {**Nome_das_variáveis**}}.

Função (main):
Nó (1)
1) {1,(2),{med,nota,peso}}

Nó (2)
2) {2,(3,4),{med}}
3) {2,(3,5),{med}}
4) {2,(5,6),{med}}
5) {2,(5,7),{med}}

Função (media):
Nó (1)
6) {1,(1),{nota,key}}
7) {1,(1,4),{med}}
8) {1,(3,4),{soma,peso}}

Nó (2)
9) {2,(2,3),{i}}

Nó (3)
10) {3,(3),{saux,paux}}

4.2.3 – Geração dos Casos de Testes

A geração dos casos de teste de ser feita de maneira minuciosa com intuito de com o mínimo de esforço e o menor número de caso de teste conseguir a cobertura de um ou vários elementos de teste.

Com base nos elementos de teste gerados anteriormente, segue alguns casos de testes que foram criados para exercitá-los.

1 - Aluno aprovado.

Dados de teste: 5.0,2.0;8.0,4.0;7.5,3.09.5,3.0;

Caso de teste: #1;

Caminho percorrido: 1 2 3 4 8;

Elemento de teste coberto: 1 e 2.

2 – Aluno de exame.

Dados de teste: 3.0,2.0;4.0,4.0;6.0,3.0;8.0,3.0;

Caso de teste: #2;

Caminho percorrido: 1 2 3 5 7 8;

Elemento de teste coberto: 1, 3 e 5.

3 – Aluno reprovado.
Dados de teste: 3.0,2.0;4.0,4.0;2.5,3.0;4.0,3.0

Caso de teste: #3;

Caminho percorrido: 1 2 3 5 6 8;

Elemento de teste coberto: 1, 3 e 4.

Verificando os casos de testes e seus respectivos elementos de teste cobertos podemos verificar que temos 100% de cobertura. Com os três casos de teste temos a verificação dos critérios *todos-nos*, *todos-arcos* referente ao Fluxo de Controle e *todos-usos* referente ao critério de Fluxo de Dados.

Conforme Spoto (2000), a partir do exercício dos casos de teste tem-se uma análise da satisfação de um critério em teste estrutural, contudo esta análise é feita pela instrumentação de cada unidade do programa em teste, pelo acompanhamento dos caminhos percorridos e executados pelos casos de teste e pela determinação dos elementos de teste executados e não executados.

4.2.4 – Relatório gerado na Projeção e Implementação do Teste

O relatório gerado na projeção do teste tem como propósito especificar refinamentos da abordagem de teste e identificar as funcionalidades e características a serem testadas por este projeto e seus testes associados. A estrutura a seguir no relatório deve ter a seguinte

seqüência: Identificador da Especificação de Projeto de Teste; Funcionalidades e Características Tratadas; Refinamentos da Abordagem de teste; Identificação dos Casos de Teste e Procedimentos de Teste Associados e Critérios de Aprovação/Reprovação. De acordo com a Norma IEEE std 829-1998, como no Plano de Teste, as seções adicionais poderão ser incluídas no final. Caso todo conteúdo ou parte de uma seção estiver em outro documento, devera ser feita a devida referência a este documento, e este material referenciado deverá estar anexado ao respectivo relatório gerado, ou ser colocado à disposição dos usuários do relatório em questão.

4.2.5 – Especificação de Caso de Teste

Este documento tem como propósito definir um Caso de Teste identificado por uma Especificação de Projeto de Teste, ou seja, é um Planejamento de Teste mais criterioso. Visto que um Caso de Teste pode ser referenciado por várias Especificações de Projeto de Teste, e utilizados por diferentes grupos durante um longo período de tempo, as informações específicas devem estar incluídas na Especificação de Caso de Teste para permitir o reuso.

A estrutura do documento deve seguir a seguinte seqüência de acordo com a Norma IEEE std 829-1998: Identificador da Especificação de Caso de Teste, Itens de Testes, Especificações de Entrada, Especificações de Saída, Requisitos de Ambiente de Teste, Requisitos de Procedimentos Especiais, Dependências entre Casos de Teste.

4.2.6 – Especificação de Procedimento de Teste

O objetivo é de especificar os passos para executar um conjunto de Casos de Testes ou, de uma maneira geral, os passos usados para analisar um item de *software* com o objetivo de avaliar um conjunto de funcionalidades e características.

A Norma não faz especificação de como agrupar os casos de teste, tal agrupamento deve levar em consideração casos de teste que sejam relacionados ou dependentes, ou que possuam os mesmos requisitos de ambiente.

Conforme a Norma IEEE std 829 -1998 uma Especificação de Procedimento de Teste deve ter a seguinte estrutura: Identificação da Especificação de Procedimento de Teste, Propósito do Procedimento de Teste, Requisitos Especiais, Passos do Procedimento de Teste.

4.3 - Execução de Teste

Nesta fase os executores de testes: unidade, integração e sistema, têm como objetivo, executar os testes, revisar os resultados e registrar os defeitos.

No processo de execução dos casos de testes é gerado um arquivo: *Pathint.tes*, cujo conteúdo é armazenar o número do caso de teste, o número da unidade exercitada e o caminho feito para a execução do caso de teste (SPOTO, 2000).

4.3.1 – Relatório Gerado na Execução de Teste

O Relatório de Execução de Teste ou Diário de Teste também como é conhecido, apresenta registros cronológicos dos detalhes relevantes relacionados com a execução dos testes.

O relatório deverá seguir a seguinte ordem e seqüência: Identificador do Diário de Teste, Descrição do Teste, Registros de Atividades e Eventos.

De acordo com a Norma, quando da necessidade de adicionar seções deverão ser adicionadas no final, se parcial ou completamente o conteúdo de uma seção estiver em outro documento então poderá ser listada uma referência a este material no lugar do conteúdo correspondente. O material referenciado deverá estar anexado no Diário de Teste ou ser colocado à disposição dos usuários deste diário.

4.4 - Avaliação dos Testes

A fase de avaliação dos resultados é onde analisa a cobertura dos casos de testes quanto aos critérios de teste, avaliando cada caso e quais unidades foram cobertas para em seguida avaliar as associações atendidas.

A avaliação dos resultados é a fase na qual se faz validações e verificações dos passos decorridos nos teste, verificando se há a necessidade da execução de algum teste já efetuado ou a criação de um outro teste para dar conclusão ao teste. Na avaliação dos testes julga-se fase na qual se aplica o conhecimento de todos os executores de testes juntamente com observações feitas pelos contratantes dos serviços de teste para que haja uma total qualidade no *software* assim desenvolvido.

O resultado obtido nesta fase é o relatório de avaliação dos testes.

4.4.1 – Relatório da Avaliação dos Testes

Em seqüência a execução dos testes e avaliação dos testes pode-se gerar um relatório para apresentar um resumo dos resultados das atividades de teste e prover avaliações baseadas nesses resultados, esse relatório é denominado pela Norma IEEE std 829 – 1998 como Relatório-Resumo de Teste.

Proposto pela Norma um Relatório-Resumo de Teste deverá ter a seguinte estrutura: Identificador do Relatório-Resumo de Teste, Resumo dos Itens de Teste, Desvios das especificações, Avaliação de Abrangência do Teste, Resumo de Resultados, Avaliação dos Itens de Teste, Resumo das Atividades e Aprovações.

As seções deverão ser ordenadas na seqüência especificada, e como nos relatórios anteriores apresentados, seções adicionais poderão ser incluídas no final. Se parcial ou todo conteúdo de uma seção estiver em outro documento, deverá constar referência ao documento, e este material referenciado deverá estar anexado ao respectivo relatório, ou ser colocado à disposição dos usuários do relatório em questão.

4.5 – Execução do *Workflow* na FADAT

Com base na descrição e na arquitetura da FADAT (PARCKERT, 2006) descrita na Seção 2.8, será detalhada nesta seção a execução do *workflow* de teste em aplicações OO na FADAT. De acordo com o que foi proposto no decorrer deste projeto a geração de um *workflow* de teste de *software,* para o teste estrutural, nos critérios todos-nos e todos-arcos do fluxo de controle e todos-usos do fluxo de dados, intra-classe, foram seguidos todos os passos discriminados no Capítulo 4 e implementados na Ferramenta FADAT.

Como podemos ver na Figura 4.4, a tela principal da ferramenta, neste ponto devemos cadastrar um usuário para a utilização e gerenciamento do *workflow* que será gerado.

Figura 4.4 – Tela inicial com o menu principal da FADAT.

Figura 4.5 – Tela de cadastro de usuário da FADAT.

O usuário cadastrado conforme a Figura 4.5 será o responsável pelo cadastro do *workflow* conforme a Figura 4.6.

Figura 4.6 – Tela de cadastro de um *workflow* da FADAT.

A seguir na Figura 4.7 a tela onde é feita a delegação do executor a manipular o relatório escolhido, como demonstrado na figura o escolhido foi o Plano de Teste por ser o passo inicial conforme descrito no início deste capítulo. Observando na parte superior da figura podemos ver todos os outros relatórios propostos pela Norma IEEE std 829-1998, os quais podem ser gerados normalmente desde que o criador do *workflow* delegue a atividade de manusear o relatório a um executor, conforme está sendo ilustrado na Figura 4.8.

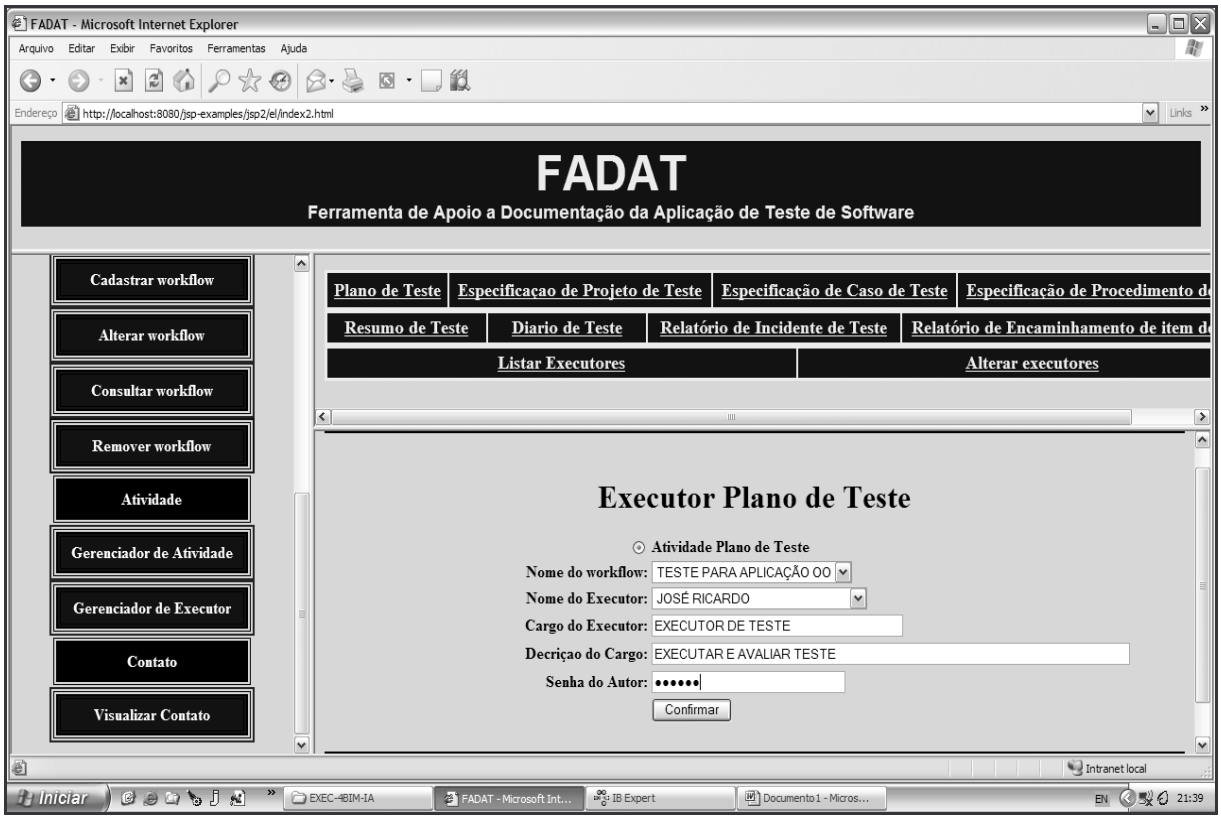

Figura 4.7 – Tela de delegação das tarefas a um executor para um *Workflow*.

Figura 4.8 – Tela de atribuição de uma atividade a um executor.

Na Figura 4.9 a tela de inserção dos dados no Plano de Teste, nesta fase é onde escolhe-se uma ferramenta de apoio se necessário, o escopo do teste, sua técnica e critério e todos os outros seqüentes requisitos propostos pela Norma IEEE std 829-1998. Processo este que deve ser feito para todos os relatórios que irão apoiar na geração final do *workflow*.

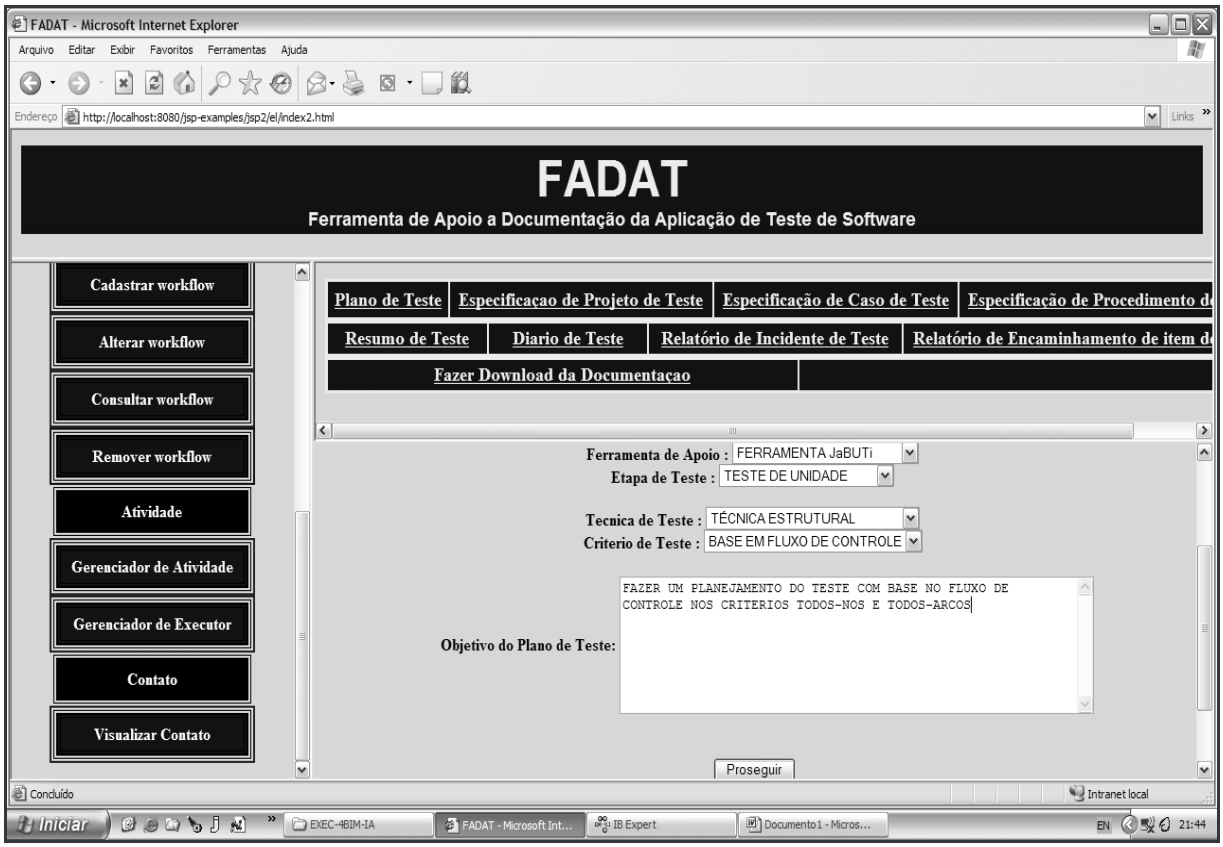

Figura 4.9 – Tela de inserção de dados a um determinado Plano de Teste.

E por fim na Figura 4.10 a tela de *download* dos relatórios gerados no *workflow* os quais serão a base para o bom entendimento do teste.

Figura 4.10 – Tela para *download* dos relatórios gerados no *Workflow*.

CONCLUSÃO

Neste trabalho foram descritos os processos para um desenvolvimento de atividades de teste deixando assim uma visão de que atualmente empresas de grande e pequeno porte necessitam ainda mais do trabalho de equipes de desenvolvedores de *software*.

Contudo todas as técnicas aqui citadas são de extrema importância no desenvolvimento de testes de um sistema. O ato de ignorar as técnicas de testes de *software* pode trazer inúmeros problemas principalmente para a empresa que esta contratando os serviços da equipe de testes quanto para a empresa que esta prestando os serviços que posteriormente serão encarregadas de arcarem com os custos desperdiçados.

Este trabalho faz uma total descrição das principais técnicas de testes de *software* e *software* OO (escopo principal deste projeto), visando assim abranger todas as técnicas existentes que possam contribuir com a construção de um *Workflow* de teste para sistemas Orientado a Objetos.

O uso de *Workflow* no processo de desenvolvimento de *software* é um mecanismo que auxilia cada vez mais o gerenciamento das atividades do processo de desenvolvimento e com isso garante mais eficiência na realização das tarefas e modificações com mais controle. Sendo assim, o *Workflow* de teste vem auxiliar a etapa de teste e geração da documentação monitorada de maneira que o testador pode ser assistido com suas atividades de teste e seguir um padrão na obtenção de resultados em suas avaliações. Este trabalho mostrou uma técnica utilizada no teste de Sistemas OO, utilizando o padrão apresentado em CRESPO *et al.*, 2004.

Trabalhos Futuros

Tendo como base este projeto é proposta como alterações futuras a expansão dos critérios para OO, deixando assim um ramo pouco mais abrangente e alimentando este *Workflow* para uma maior eficiência no desenvolvimento de novos *Softwares*. Um dos critérios que pode ser inserido é o todos-caminhos a nível de unidade, e ainda a inserção dos testes de integração, a atividade de polimorfismo e herança.

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.

BEIZER, B. *Software testing techniques*. 2nd *ed. New York: Van Nostrand Reinhold Company*, 1990.

BERARD. E. V., *Essays on Object-Oriented Software Engineering*, vol. 1, Addison-Wesley, 1993.

BINDER, R. V. *Object-Oriented Systems, CACM*, vol.37, no.9, 1994.

CHAIM, M. L. Poke-tool – uma ferramenta para suporte ao teste estrutural de programas baseado em análise de fluxo de dados. Dissertação de Mestrado, DCA/FEEC/UNICAMP, Campinas, SP, 1991.

CHAYS, David; VOKOLOS, Filippos I.; WEYUKER, Elaine J. *A Framework for Testing Database Applications, ISSTA, 00, ACM, Oregon, PO*, pp: 147-156, 2000.

CRESPO A. N., Silva O. J., Borges C. A., Salviano C. F., Jino M., Uma Metodologia para Teste de *Software* no Contexto da Melhoria de Processo, Campinas, São Paulo, 2004.

DUNDAN, I.; ROBSON, D. *Na exploratory study of common coding faults in C programs. Software Maintenance: Research and Practice*, 1996.

FRANKL, P. G.; WEISS, S. N.; WEYUKER, E. J. SSET: *A system to select and evaluate tests. In: IEEE Conference on Software Tools, New York, NY,* 1985.

FRANKL, F. G. *The use of data flow information for the selection and evaluation of Software test data.* Tese de Doutoramento, Universidade de *New York, New York, NY*, 1987.

FRANKL, P. G.; WEYUKER, E. J. *An applicable family of data flow testing criteria. IEEE Transactions on Software Engineering,* 1988.

GONÇALVES, K. V., Teste de *Software* em Aplicações de Banco de Dados Relacional, 2003.

HARROLD, M. J.; SOFFA, M. L. *Interprocedural data flow testing. In: 3th Testing, Analysis, and Verification Symposium, Key West, Florida: ACM Press*, 1989.

HARROLD, M. J.; ROTHERMEL, G. *Performing data flow testing on classes. In: Second ACM SGSOFT Symposium on Foundations of Software Engineering, New York: ACM Press,* 1994.

HARROLD, M. J. *Testing A roadmap. In: 22th International Conference on Software Engineering – Future of SE Track*, 2000.

HARTMANN, J.; ROBSON, D. J. *Techniques for selective revalidation. IEEE Software*, 1990.

HORGAN, J. R.; LONDON, S. A. *Atac – automatic test coveragr analysis for c programs, draft*, 1990.

HORGAN, J. R.; LONDON, S. A. *Data flow coverage and the C language. In: Symposium Software Testing, Analysis, and Verification, Victoria, British Columbia, Canada: ACM Press*, 1991.

HOWDEN, W. E. *Software engineering and technology: Functional program testing and analysis. New York:* McGrall-Hill, 1987.

IEEE-Std-829, IEEE: *Standard for Software Test Documentation, Software Engineering Technical Committee of the IEEE Computer Society, September*, 1998.

ITEA*, Information Technology for European Advancement – The DESS Methodology – Software Development Process for Real-Time Embedded Software Systems (DESS). Deliverable* D.1. *Version* 01, 2001.

KHOSAFIAN, S.; BUCKIEWICZ, M. *Workflow: Computer-Supported Collaborative Work-Processing. In: Introduction to Groupware, Workflow and Workgroup Computing. New York: John Wiley & Sons,* 1995.

LEEDS*, UNIVERSITY OF. IN22 Lecture 16 RUP workflows*. Slides. Disponível em:<http://www.scs.leeds.ac.uk/in22/in22_lec16.ppt> Acesso em: 1998.

MALDONADO, J. C.; CHAIM, M. L.; JINO, M. Seleção de casos de testes baseada na análise de fluxo de dados através dos critérios potenciais usos. In: II SBES – Simpósio Brasileiro de Engenharia de *Software*, Canela, RS, 1988.

MALDONADO, J. C. Critérios potenciais usos: Uma contribuição ao teste estrutural de *Software*. Tese de Doutoramento, DCA/FEE/UNICAMP, Campinas, SP, 1991.

MALDONADO, J.; Rocha, A. R.; Weber, K. Qualidade de *Software*: Teoria e prática. Primeira edição. Prentice Hall, 2001.

MASSONI, Tiago Lima. Testes – Estudo do RUP. Slides. Universidade Federal de Pernambuco. Recife, Brasil. 1999. Disponível em: <http://www.cin.ufpe.br/~phmb/RUP/MaterialDeEnsino/Testes.ppt> Acesso em:15/10/2004.

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

MIRELLA. M. M., *Workflow*, Instituto de Informática UFRGS, RS, 1998.

MYERS, G. *The Art of Software Testing. Wiley, New York*, 1979.

NTFATOS, S. C. *A comparison of some structural testing strategies. IEEE Transactions on Software Engineering*, 1988.

OSTRAND, T. J.; WEYUKER, E. J. *Data flow analysis for regression testing. In: Sixth Annual Pacific Northwest Software Quality Conference, Portland – Oregon*, 1988.

OSTRAND, T. J.; WEYUKER, E. J. *Data flow based test adequacy for languages with pointers. In: Symposium on Software Testing, Analysis and Verification* – TAV4, Victoria, *British Columbia, Canada*: *ACM Press*, 1991.

PARCKERT, Jeferson D., "Aspectos de Automatização da Documentação da Aplicação de *Workflow* de Teste .", Trabalho de Conclusão de Curso de Graduação, UNIVEM, Marília, SP, Novembro, 2006.

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

PRESSMAN, R. S. *Software engineering – a practitioner's approach* 5 *ed*. McGraw-Hill, 2001.

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

RAPPS, S,; WEYUKER, E. J. *Data flow analysis techniques for program test data selection. In: 6th International Conference on Software Engineering, Tokio, Japan*, 1982.

RAPPS, S,; WEYUKER, E. J. *Selecting Software test data using data flow information. IEEE Transactions on Software Engineering*, 1985.

RUIZ, D. D., Introdução a *Workflow,* 1999.

SBBD, XIV Simpósio Brasileiro de Banco de Dados - Florianópolis, 1999.

SILVER, Bruce. *Automating the Business Environment. In: New Tools for New Times: The Workflow Paradigm. Lighthouse Point: Future Strategies*, 1995.

SOMMERVILLE, I. Engenharia de *Software*. Sexta edição. Pearson Addison Wesley, 2003.

SNWSCSD - *Space and Naval Warfare Systems Center San Diego, Software Engineering Process Office, D12; Handbook for Process Management, Version* 1.0*, San Diego. September*, 1998.

SPOTO, Edmundo S., Um Estudo de Critérios de Teste de *Software* Baseados em Fluxo de Dados. Campinas: UNICAMP-FEE-DCA, Tese de Graduação, 1995.

TURNER, Terence J.; HALL, Rex; et al. *Eletronic Workflow Using the World Wide Web*. Disponível em: <http://www.admin.ufl.edu/division/oa/pc_stuff/cumrec96t2-2.htm> Acesso em: 1998.

VARADAN, G. S. *Trends in reliability and test strategies. IEEE Software*, 1995.

VEEVERS, A.; MARSHALL, A. *A relationship between Software coverage metrics and reliability. Software Testing, Verification and Reliability*, 1994.

VINCENZI, Auri M.R., Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação, Dissertação de Mestrado. ICMC/USP. São Carlos/SP, 1998.

Workflow Management Coalition. The Workflow Reference Model. Hampshire, UK. 1995.

Workflow Management Coalition. The Workflow Reference Model. Disponível em: http://www.wfmc.org Acesso em: 06/1998.