UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

TESTES DE SOFTWARE A PARTIR DA FERRAMENTA *VISUAL TEST*

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

MARCIO TOMELIN

BLUMENAU, JUNHO/2001

2000/1-50

TESTES DE SOFTWARE A PARTIR DA FERRAMENTA *VISUAL TEST*

MARCIO TOMELIN

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____
Prof. Dr. Paulo César Rodacki Gomes - Orientador FURB

_____
Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____
Prof. Dr. Paulo César Rodacki Gomes

_____
Prof. Everaldo Artur Grahl

_____
Prof. Dr. Oscar Dalfovo

DEDICATÓRIA

Dedico este trabalho,

aos meus familiares e amigos

e principalmente a meus pais Ivo e Zulmira,

pelo apoio dado durante a elaboração deste trabalho.

AGRADECIMENTOS

Ao Professor Paulo César Rodacki Gomes, pela paciência e pelo interesse com o qual orientou este trabalho.

Ao Professor José Roque Voltolini da Silva, coordenador do Trabalho de Conclusão de Curso.

A todos os professores e funcionários do Departamento de Sistemas e Computação que auxiliaram para que este trabalho pudesse ser realizado.

À WK Sistemas, pelo apoio recebido e pela contribuição com o ensinamento da ferramenta Visual Test.

Aos colegas, tanto aqueles que ficaram no decorrer do curso, como aos que conseguiram junto comigo chegar ao fim de mais uma etapa de nossas vidas.

E a todos que de alguma forma contribuíram para a realização deste trabalho.

SUMÁRIO

LISTA DE QUADROS

LISTA DE FIGURAS

RESUMO

O teste de software é uma das fases do ciclo de vida de um software que mais contribuem para a garantia da qualidade do software. Várias ferramentas têm sido construídas com o objetivo de apoiar esta fase. Entre elas está o *Visual Test*. Esta ferramenta simula de forma automatizada a entrada de dados informadas pelo usuário final de forma a alimentar o sistema com cadastros, movimentações, relatórios e outros. Com o desenvolvimento deste trabalho pretendeu-se montar um projeto de software automatizado que agilize e dê qualidade a este tipo de atividade. Este projeto baseou-se em técnicas de testes sugeridas por normas de qualidade.

ABSTRACT

Software Testing represents an important process in a software's life-cycle regarding it's software quality. Several tools for helping such process have been designed. Rational's Visual Test is one of them. Such tool automates the simulation of final user's data input. The present work proposes an automated software project for Software Testing, providing better performance and quality to this kind of activity, which is currently left aside by most software houses. This project is based mainly on quality software testing standards.

1. INTRODUÇÃO

Os testes de software são vistos como uma tarefa contínua de avaliar e mensurar a qualidade do trabalho desenvolvido em cada etapa do processo de desenvolvimento de sistemas, desde a análise de requisitos até a fase de manutenção do software. Aproximadamente 50% do tempo e mais 50% do custo total de um sistema são gastos no teste de programas ou sistemas em desenvolvimento. Testar sistemas é uma atividade que a maioria das pessoas já faz por força ou por obrigação, é uma atividade na qual investe-se muito tempo e dinheiro. Ainda assim, não é definida com clareza. Na maioria das empresas, os testes são mal estruturados e feitos de maneira fortemente individualizada, quase aleatoriamente (Martimiano, 1995).

A noção “teste de programas” surgiu quase que simultaneamente ao aparecimento dos primeiros programas. Os programas executados nos primórdios da computação também tinham que ser testados. A realização de testes era uma atividade rotineira associada a processos de engenharia e produção industrial e a sua extensão ao desenvolvimento de software pode ser encarada como um desdobramento natural.

Segundo se pensava, os programas eram “escritos” e depois testados e depurados. Os testes eram considerados uma atividade secundária, englobando os esforços para a detecção de erros e também a sua correção e eliminação. Vários dos primeiros estudos publicados sobre teste de software, abordavam aa depuração. A dificuldade da correção e eliminação de erros foi vista durante muito tempo como um problema mais interessante. O tema vem crescendo em importância com a necessidade de todos os setores da informática no sentido de criar métodos práticos que assegurem a qualidade de seus produtos finais. No entanto o autor acredita que a maturidade ainda está longe de ser alcançada.

Os ambientes de engenharia de software orientados a processo visam apoiar as fases do processo de software, permitindo a definição de tarefas e a comunicação e o compartilhamento de dados entre as ferramentas que compõem o ambiente.

É fundamental em todos os ramos da engenharia de software garantir a produção de software de alta qualidade a fim de proporcionar aos usuários uma maior confiança e segurança na utilização do mesmo.

Com a necessidade de não apenas dar suporte aos objetos gerados durante o desenvolvimento de software, mas também de se definir e controlar o processo de desenvolvimento e manutenção de software, considerando dessa forma, o processo como parâmetro do ambiente (Gimenes, 1994) surgiram então, os ambientes de engenharia de software orientados a processos (PSEE). Os PSEEs caracterizam-se por prover suporte a descrição e execução de processos de modo a auxiliar e controlar todas as atividades do ciclo de vida de um software. Dentre estas atividades, tem-se a fase de teste de software, na qual se concentrará este texto.

Testes eficientes são essenciais para o controle de qualquer projeto de desenvolvimento de software. É uma forma de verificar se o sistema que está sendo desenvolvido está sendo feito de maneira correta e conforme os requisitos especificados pelo usuário. O teste de software envolve: planejamento de testes, projeto de casos de testes, execução e avaliação dos resultados obtidos. Segundo Hetzel (1987), teste é qualquer atividade que vise avaliar uma característica ou recurso de um programa ou sistema. Teste é uma forma de medir a qualidade do software. Um teste de software examina o comportamento do software através de sua execução em um conjunto de dados pertencentes a um domínio de teste definido pela técnica de teste a ser usada (Martimiano, 1995). O teste de software pode ser realizado durante todas as fases do processo de desenvolvimento do software, portanto trata-se de uma das atividades mais importantes no desenvolvimento de um software.

Em muitos casos os programas são testados isoladamente à medida que os módulos vão sendo concluídos, a fim de confirmar que o módulo foi codificado corretamente. Depois, grupos de programas são testados num "teste de sistema" onde é feito um teste de integração para testar as interfaces e assegurar que os módulos estão se comunicando da maneira esperada. Em seguida, o software é explorado como forma de detectar suas limitações e medir suas potencialidades. Em um terceiro nível, sistemas completos são, por fim, submetidos a um "teste de aceitação" para verificar a possibilidade de implantação e uso, geralmente feita pelo cliente ou usuário final.

Enfim, os testes de software podem ser baseados na norma brasileira NBR-12207 (ABNT, 1998) ou em diversas normas internacionais tais como, IEEE, CMM, ISO, etc. Os trabalhos de Ramirez (1999), Souza (2000) apresentam compilações da norma IEEE94.

Através do estudo destas normas e da ferramenta de automatização de testes *Visual Test*, foi implementada uma de teste automatizada. O software submetido ao teste foi o Radar Contábil desenvolvido pela *software house* WK WK Sistemas de Computação Ltda.

1.1 OBJETIVOS

O objetivo principal deste trabalho é implementar uma solução automatizada para os testes de software utilizando a ferramenta de testes *Visual Test*.

Os objetivos específicos são:

a) observar a atividade de teste na WK Sistemas, e a partir desta observação, identificar se a empresa está praticando as atividades de teste conforme as técnicas estudadas neste trabalho;
b) realizar um estudo sobre as recomendações para teste de software previstas em algumas normas de qualidade.

1.2 ESTRUTURA

O primeiro capítulo fornece uma introdução ao trabalho desenvolvido, demonstrando qual o objetivo do trabalho e apresentando os principais tópicos.

O segundo capítulo demonstra o que é Teste de Software suas metodologias, tipos, como e porque executá-los. Mostra também para que servem e como devem ser criados os casos de teste, oque algumas normas falam a respeito de teste de software e faz um breve comentário sobre uma ferramenta de automatização de testes, o *Visual Test*.

O terceiro trata da especificação do protótipo e demonstra sua implementação por meio de um diagrama de fluxo de dados. Fornece também outras informações sobre a implementação, tais como operacionalidade da mesma e, as técnicas e ferramentas utilizadas.

Por fim, o quarto capítulo faz uma análise conclusiva indicando as dificuldades encontradas e apresenta sugestões para futuras extensões do trabalho que poderão ser realizadas.

2. TESTE DE SOFTWARE

Este capítulo apresenta um conjunto de princípios, métodos e técnicas relativos à atividade de teste de software. Esta atividade por sua vez é parte integrante de um dos processos ciclo de vida de um software, conforme tratado pela engenharia de software.

A engenharia de software é uma disciplina tecnológica e gerencial preocupada com a produção sistemática de produtos de software, que são desenvolvidos e/ou modificados dentro do tempo e custo estimados, tendo como principal objetivo, a produção de software de alta qualidade e baixo custo (Souza, 2000).

Ainda seguindo a idéia do autor, defini-se teste de software como sendo um processo de executar um programa com o objetivo de revelar a presença de defeitos; ou, falhando nesse objetivo, aumentar a confiança sobre o programa.

2.1 ATIVIDADE DE TESTES

A atividade de testes é uma etapa crítica para o desenvolvimento de software. Freqüentemente, a atividade de testes insere tantos erros em um produto quanto a própria implementação. Por outro lado, o custo para correção de um erro na fase de manutenção é de 60 a 100 vezes maior que o custo para corrigí-lo durante o desenvolvimento.

Embora as revisões técnicas sejam mais eficazes na detecção de defeitos, os testes são importantes para complementar as revisões e aferir o nível de qualidade conseguido. A realização de testes é, quase sempre, limitada por restrições de cronograma e orçamento. É importante que os testes sejam bem planejados e desenhados, para se conseguir o melhor proveito possível dos recursos alocados para eles.

Um objetivo central de toda a metodologia dos testes é maximizar a cobertura destes. Deseja-se conseguir detectar a maior quantidade possível de defeitos que não foram apanhados pelas revisões, dentro de dados limites de custo e prazos. Os testes não provam que um programa está correto, somente contribuem para aumentar a confiança de que o software desempenha as funções especificadas” (Souza, 1996).

Existem duas grandes razões para se testar software, fazer um julgamento sobre qualidade e descobrir problemas. Na verdade, testa-se porque sabe-se que as atividades

humanas são passíveis de falha e, isso é especialmente verdade no domínio de desenvolvimento de software.

2.2 OBJETIVOS E LIMITES DO TESTE DE SOFTWARE

A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro, sendo assim, um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto. A atividade de teste não tem a capacidade de mostrar a ausência de *bugs*(falhas e/ou erros encontrados em um programa de computador). Ela só pode mostrar se defeitos de software estão presentes. Tem-se como principal objetivo, projetar testes que descubram sistematicamente diferentes classes de erros e façam-no com uma quantidade de tempo e esforço mínimos. Se a atividade de teste for conduzida com sucesso, ela descobrirá erros no software.

Um plano de testes realista prevê a solução de uns poucos casos de teste a partir de milhares de casos possíveis. Não importando quão minuciosos forem os testes, nunca encontra-se o último *bug* de um programa, ou se é encontrado, não se sabe que tal ocorreu.

Para Souza (2000), é impossível testar um programa completamente. Testar um programa completamente significa que não existe absolutamente mais nenhum bug a ser encontrado. Para garantir que um programa está completamente testado é necessário testar todas as combinações de entradas válidas (em um programa que soma dois dígitos são aproximadamente 39600 entradas válidas) , todas as combinações de entradas inválidas (testar todas as possíveis combinações do teclado que não sejam pares válidos de 1 ou 2 dígitos). Deve-se testar também todas as possibilidades de edição (se o programa permite editar os valores digitados, é preciso testar todos os recursos de edição e verificar se o resultado de uma edição ainda é uma entrada válida). Todas as possibilidades em relação ao tempo (por exemplo: se digitamos dois números rápido demais, intercalados por <*enter*> e o programa não consegue capturar um dos valores) (Oliveira, 1997).

Finalmente, para este autor, o mais complexo é testar todas as entradas possíveis em qualquer ponto que o programa permita entradas de dados, em todos os estados possíveis que ele pode atingir nesses pontos. Esta tarefa é praticamente impossível. Como exemplos de situações que normalmente não são testadas, pode-se citar:

a) um certo gerenciador de banco de dados falha quando certas tabelas tem exatamente 512 bytes de tamanho;
b) um certo editor de textos falha se o texto for muito longo (mais de 100.000 caracteres), desde que o texto estija muito fragmentado no disco (espalhado em setores não contíguos).

O objetivo de um testador de software, neste caso um testador profissional e não uma ferramenta de teste, não pode ser ele querer mostrar que o software está livre de *bugs*, porque ele jamais conseguirá provar isso. De alguma forma ou maneira ele provavelmente deixará escapar muitos *bugs* (Oliveira, 1997). O principal objetivo de um testador de software deve ser: encontrar problemas no sistema. E, quando encontrado, é encaminhá-lo para o conserto. Sendo assim: um teste falha quando não acha nenhum *bug*. Os *bugs* estão lá, cabe ao testador encontrá-los.

2.3 METODOLOGIAS DE TESTE DE SOFTWARE

Segundo o IEEE (1994), para serem eficazes, os testes devem ser cuidadosamente desenhados e planejados.

Os testes irreproduziveís e improvisados devem ser evitados. Durante e após a realização dos testes, os resultados de cada teste devem ser minuciosamente inspecionados, comparando-se resultados previstos e obtidos. Os desenvolvedores não são as pessoas mais adequadas para testar seus próprios produtos. Assim como nas revisões, os autores têm maior dificuldade em enxergar problemas, comparados com pessoas que não participaram do desenho e da implementação.

O trabalho de testadores independentes é particularmente importante no caso dos testes de aceitação. Possivelmente, os mesmos testadores independentes se encarregarão também dos testes de integração. Testes de aceitação e integração, serão vistos em seguida em uma maior profundidade.

O planejamento e o desenho dos testes devem ser feitos por pessoas experientes, que conheçam adequadamente a metodologia de testes usada. Por outro lado, a realização dos testes baseados em planos e especificações de testes bem definidos pode ser feita por pessoas

menos experientes, ou até parcialmente automatizada. Os testes são indicadores da qualidade do produto, mais do que meios de detecção e correção de erros.

Por fim, quanto maior o número de defeitos detectados em um software provavelmente maior também o número de defeitos não detectados. A ocorrência de um número anormal de defeitos em uma bateria de testes indica uma provável necessidade de redesenho dos itens testados.

Existem basicamente duas maneiras de se construírem testes (Poston, 1996) :

a) método da caixa branca, que tem por objetivo determinar defeitos na estrutura interna do produto, através do desenho de testes que exercitem suficientemente os possíveis caminhos de execução;
b) método da caixa preta, que tem por objetivo determinar se os requisitos foram total ou parcialmente satisfeitos pelo produto. Os testes de caixa preta não verificam como ocorre o processamento, mas apenas os resultados produzidos. Os testes de aceitação e de regressão normalmente são de caixa preta. Os testes de integração geralmente misturam testes de caixa preta e de caixa branca.

2.4 BATERIAS DE TESTES

A seguir estão descritos brevemente os tipos de baterias de testes, ou conjuntos de testes de software de determinada categoria, conforme proposto pela IEEE segundo Poston (1996).

a) Testes de Unidade, que geralmente são de caixa branca, têm por objetivo verificar um elemento que possa ser localmente tratado como uma unidade de implementação; por exemplo, uma sub-rotina ou um módulo. Em produtos implementados com tecnologia orientada a objetos, uma unidade é tipicamente uma classe. Em alguns casos, pode ser conveniente tratar como unidade um grupo de classes correlatas;
b) Testes de Integração têm por objetivo verificar as interfaces entre as partes de uma arquitetura de produto. Esses testes têm por objetivo verificar se as unidades que compõem cada liberação funcionam corretamente, em conjunto com as unidades já integradas, implementando corretamente os casos de uso cobertos por essa liberação executável;

c) Testes de Aceitação têm por objetivo validar o produto, ou seja, verificar se este atende aos requisitos especificados. Eles são executados em ambiente o mais semelhante possível ao ambiente real de execução. Os testes de aceitação podem ser divididos em testes funcionais e não funcionais;

d) Testes de Regressão, que executam novamente um subconjunto de testes previamente executados. Seu objetivo é assegurar que alterações em partes do produto não afetem as partes já testadas. As alterações realizadas, especialmente durante a manutenção, podem introduzir erros nas partes previamente testadas. A maior utilidade dos testes de regressão aparece durante o processamento de solicitações de manutenção. Entretanto, testes de regressão podem ser executados em qualquer passo do desenvolvimento. Por exemplo, para cada liberação, deve ser feita uma regressão com os testes de integração das liberações anteriores.

2.5 ATIVIDADES DE TESTE DE SOFTWARE

O processo de testes tem dois grandes grupos de atividades para cada bateria de testes: preparação e realização dos testes. Durante a preparação, é elaborado o plano da bateria e são desenhadas as especificações de cada teste. Durante a realização, os testes são executados, os defeitos encontrados são, se possível, corrigidos, e os relatórios de teste são redigidos.

Normalmente, a preparação se inicia pelos testes de mais alto nível, ou seja, os testes de aceitação, baseados principalmente na Especificação de Requisitos do Software. Desses testes e do desenho das liberações, elaborado no fluxo de desenho e contido na descrição do desenho do software, são derivados os testes de integração. O desenho detalhado das unidades, realizado durante cada liberação, serve de base para o desenho dos testes de unidade (Ramirez, 1999). O Quadro 1 mostra as entradas necessárias para cada tipo de teste e seu respectivo desenho.

Quadro 1- ENTRADAS PARA O DESENHO DE TESTES

	ENTRADAS
Testes de aceitação	Especificação dos requisitos Plano de desenvolvimento Plano da qualidade
Testes de integração	Descrição do Projeto (projeto de alto nível) Descrição dos Testes (testes de aceitação)
Testes de unidade	Descrição do Projeto (projeto detalhado) Descrição dos Testes (testes de integração) Código Fonte da Unidade

Fonte: Ramirez (1990).

2.5.1 PLANEJAMENTO DOS TESTES

O planejamento dos testes de produtos não triviais é complexo, envolvendo muitos aspectos técnicos e gerenciais. Para facilitar a descrição, a atividade de planejamento foi dividida em três tarefas separadas, o plano de testes, o planejamento inicial e identificação dos itens a testar. A seguir temos uma descrição mais detalhada destas três tarefas.

2.5.1.1 PLANOS DE TESTES

Ele será normalmente preenchido durante a iteração de projeto inicial. Sua base é a especificação de requisitos do software; geralmente, cada caso de uso gera uma especificação de teste funcional.

Os planos de testes de integração são preenchidos no inicio de cada liberação. Os testes escolhidos para cada liberação partem normalmente de um subconjunto dos testes de aceitação, correspondente aos casos de uso implementados nessa liberação.

Testes adicionais de caixa branca são usados para verificar se as interfaces dos componentes implementados nessa liberação estão funcionando corretamente (entre si e com os componentes herdados das liberações anteriores). Estes testes adicionais podem ser derivados das realizações dos casos de uso de desenho, onde as interfaces são exercitadas através das colaborações entre componentes.

Cada plano de testes de unidade é preenchido durante a respectiva implementação. Os testes de unidade geralmente requerem a construção de componentes de teste. Normalmente, só são desenhados, registrados e guardados os testes das unidades críticas, para uso futuro em manutenção.

Tipicamente, uma organização começaria por planejar e desenhar os testes de aceitação. Na medida em que a cultura da organização absorvesse esses testes, passaria a planejar e a desenhar também os testes de integração, e finalmente os testes de unidades críticas.

O padrão adotado para planos de testes prevê a estrutura mostrado no Quadro 2.

Quadro 2 - ESTRUTURA DE PLANDO DE TESTES

Item	**Descrição**
Identificar o plano de testes	Identificador único para o plano
Introdução	Objetivos, histórico, escopo, referências a documentos
Itens a testar	Conjuntos dos itens cobertos pelo plano
Aspectos a testar	Conteúdos dos testes que serão feitos
Aspectos que não serão testados	Aspectos significativos que não serão testados
Abordagem	Opções metodológicas que são aplicáveis ao conjunto de testes
Critérios de completeza e sucesso	Condições que devem ser satisfeitas e estados que devem ser atingidos para que o conjunto de testes seja considerado bem sucedido
Critério de suspensão e retomada	Problemas graves que podem provocar interrupção dos testes
Resultado do teste	Artefatos que serão produzidos durante a realização da bateria de testes
Tarefas de teste	Lista detalhada das tarefas que serão cobertas por este plano
Ambiente	Hardware e software utilizados para o conjunto dos testes
Responsabilidades	Responsabilidades de cada um dos participantes da equipe de testes
Agenda	Data de inicio e de fim de cada tarefa do plano
Riscos e contingências	Riscos e contingências aplicáveis aos testes deste plano
Aprovação	Nome assinatura dos responsáveis pela criação do plano

Fonte: Ramirez (1990).

2.5.1.2 PLANEJAMENTO INICIAL

O Quadro 3 a seguir, mostra em detalhes o planejamento inicial dos testes. Os insumos são documentos de planejamento, como o plano de desenvolvimento de software, no caso de testes de aceitação, e a seção de plano das liberações executáveis da descrição do projeto do software, no caso de testes de integração.

Quadro 3 - PLANEJAMENTO INICIAL DOS TESTES

Descrição	**Planejamento inicial dos testes**
Insumos	Plano de desenvolvimento do software (testes de aceitação) Descrição do desenho do software – Plano de liberações (teste de integração)
Atividades	Escolher abordagens para os testes, identificando: Áreas de risco que devem ser testadas; Restrições existentes aos testes; Fontes existentes aos casos de testes; Métodos de validação dos casos de testes; Técnicas para registro, coleta, redução e validação das saída; Necessidades de estruturas provisórias. Especificar condições de completeza dos testes: Itens a serem testados; Grau de cobertura por itens. Especificar condições de término de testes: Condições para término normal; Condições de término anormal e procedimentos a adotar nestes casos. Determinar requisitos de recursos: Pessoas; Hardware; Software de sistema; Ferramentas de teste; Formulários; Suprimentos; Especificar agenda dos testes.
Resultados	Informação geral de planejamento dos testes. Requisições de recursos para testes.

Fonte: Ramirez (1990).

2.5.1.3 IDENTIFICAÇÃO DOS ITENS A TESTAR

A terceira tarefa do planejamento dos testes, compreende a identificação dos itens a testar. Os insumos são a especificação de requisitos do software, no caso dos testes de aceitação, e as partes apropriadas da descrição do desenho do software, no caso dos demais testes. São identificados os requisitos a testar, é determinado o *status* dos itens sob teste (por

exemplo, itens novos ou modificados) e são caracterizados os dados para os casos de teste. É produzida a lista dos itens e aspectos a testar.

2.5.2 DESENHO DE TESTES

Nesse atividade, conforme o Quadro 4, é feito o desenho da bateria de testes, estes desenhos vêm a ser a especificação completa desta atividade, ou seja, como e de que maneira cada parte integrante do software testado deve ser submetido ao teste. Esta especificação deve conter todas informações necessárias para as entradas suas respectivas saídas e a maneira com que o software deve se comportar ao ser submetido ao referido teste. São completadas as especificações dos testes da bateria, desenhando-se os procedimentos e casos de teste. Em seguida são selecionados para reutilização artefatos de teste de outros projetos que possam ser reaproveitados, enfim, é definida a ordem de execução dos casos de teste. (Ramirez, 1999)

Em certos casos, principalmente nos testes de unidade, pode ser necessário solicitar o redesenho de módulos, para melhorar a testabilidade destes. Ainda seguindo a idéia do autor, pode-se ver no Quadro 4 a seguir o que deve conter um desenho de bateria de testes:

Quadro 4 - DESENHO DA BATERIA DE TESTES

Descrição	Desenho da Bateria de testes
Insumos	Especificação dos requisitos do software (testes de aceitação). Descrição do desenho do software – desenho arquitetônico e plano das liberações dos executáveis(teste de integração). Descrição do desenho do software – desenho detalhado da unidade (testes de unidade). Plano de testes. Especificações de testes anteriores.
Atividades	Desenhar bateria de testes, estabelecendo: Objetivos dos testes; Reutilização de especificação de testes existentes; Ordenamento dos casos de testes. Especificar os procedimentos de testes. Especificar os casos de teste.
Resultados	Especificações dos testes. Solicitações de melhoria do desenho para testabilidade, quando necessário.

Fonte: Ramirez (1990).

2.5.2.1 ESPECIFICAÇÕES DOS TESTES

As especificações de testes contêm os detalhes dos testes a serem realizados. A separação entre planos e especificações permite o reaproveitamento das especificações em diversos planos.

Os procedimentos de teste devem conter uma seqüência de ações que devem ser executadas para realizar um grupo de testes semelhantes. Geralmente, cada procedimento de teste corresponde a um roteiro importante de um caso de uso. Um procedimento pode ser executado de forma manual, ou, automática. Neste último caso, deve ser codificado em linguagem de *script* da ferramenta de automação de testes.

Para os casos de teste é indispensável que se contenha valores de entradas e saídas esperados para cada instância de teste. Esses valores são escolhidos de acordo com critérios que maximizam a cobertura da especificarão de teste. Outro fator que não pode ser esquecido, é a especificação da ordem de execução, pois a execução correta de um caso pode depender de um estado de uma base de dados, que é produzido por um caso anterior. O padrão aqui adotado para especificações de testes prevê a estrutura mostrada no Quadro 5.

Quadro 5 - ESTRUTURA DE UMA ESPECIFICAÇÕA DE TESTES

Item		**Descrição**
Identificador da especificação de teste		Identificador único para teste.
Aspecto a serem testados		Aspectos a testar combinado neste teste.
Detalhes da abordagem		Detalhes da abordagem específicos deste.
Identificação dos testes	Procedimentos de teste	Procedimentos associados a este teste.
	Casos de teste	Casos de teste associados a este teste.
Critérios de completeza e sucesso		Critérios de completeza e sucesso específicos deste teste.
Procedimentos de teste		Descrição de cada um dos procedimentos de teste listados anteriormente.
Casos de teste		Descrição de cada um dos casos de teste listados anteriormente.

Fonte: Ramirez (1990).

2.5.2.2 DESENHO DE PROCEDIMENTOS DE TESTE

Os procedimentos de teste devem descrever a seqüência de passos necessária para executar uma variação de teste.

Nos testes funcionais, cada variação é tipicamente baseada em um roteiro do respectivo caso de uso. É conveniente prever procedimentos de teste para execução de seqüências erradas mas possíveis.

O fluxo do procedimento representa os passos que devem ser executados por um testador humano ou automatizado, em termos das telas, campos e comandos envolvidos. Os valores efetivos dos campos serão determinados por cada caso de teste.

2.5.2.3 DESENHO DE CASO DE TESTE

Cada teste consta da execução de um ou mais procedimentos de teste, que são aplicáveis a vários conjuntos de entradas, chamados de casos de teste. Cada caso de teste, por sua vez, tem um ou mais procedimentos de teste associados. O que é, e como identificar um caso de teste, será visto com maiores detalhes no capítulo 2.7 deste trabalho.

2.5.2.4 REVISÃO DO DESENHO DE TESTE

Um bom desenho de testes gera grande quantidade de informação. Para cada caso de uso, as respectivas especificações de testes geralmente ocupam várias páginas. Vê-se que, mesmo em sistemas relativamente simples, com poucas dezenas de casos de uso, a Descrição dos testes pode chegar a dezenas ou centenas de páginas.

Neste volume de informação, os desenhistas de testes facilmente introduzem defeitos. Por isso, é recomendável que os planos e especificações dos testes sejam submetidos a uma revisão técnica formal, pelo menos no caso dos testes de aceitação.

2.5.3 IMPLEMENTAÇÃO DOS TESTES

Nessa atividade, descrita no Quadro 6, é realizada a implementação dos testes. O ambiente de teste é completamente preparado, tornando-se disponíveis todos os recursos necessários.

Os itens a testar são instalados e configurados, conforme necessário, assim como as ferramentas e componentes de teste. Os componentes podem ser reaproveitados de testes anteriores, ou desenvolvidos especialmente para os testes em questão (Ramirez, 1999).

Quadro 6 - IMPLEMENTAÇÃO DOS TESTES

Descrição	**Implementação dos testes**
Insumos	Plano de testes. Especificações dos testes. Recursos para os testes. Itens a testar. Ferramentas de teste. Dados de atividades anteriores de teste, se houver. Estruturas provisórias de testes anteriores, se houver.
Atividades	Configurar ambientes de teste. Disponibilizar todos os recursos necessários. Instalar itens a testar, ferramentas e estruturas provisórias.
Resultados	Itens a testar, instalados e configurados. Ferramentas e estruturas provisórias instaladas e configuradas.

Fonte: Ramirez (1990).

2.5.4 EXECUÇÃO DOS TESTES

Essa atividade, executa os testes da bateria, produzindo os relatórios correspondentes. Podem resultar, dos problemas encontrados neste passo, solicitações de correção dos itens sob teste, assim como alterações nos planos e especificações dos próprios testes (Ramirez, 1999).

2.5.5 VERIFICAÇÃO DO TÉRMINO DOS TESTES

Segundo Ramirez (1999), essa atividade, determina se estão satisfeitas as condições para completeza e sucesso dos testes. Caso necessário para garantir a cobertura desejada, testes suplementares podem ser desenhados, retornando-se ao passo de execução.

2.5.6 BALANÇO FINAL

Essa atividade, realiza o balanço final dos testes da bateria. São registradas as lições aprendidas, completando-se o relatório de resumo dos testes. Se possível, são identificados artefatos reutilizáveis em outros testes, inclusive procedimentos, casos e componentes de teste (Ramirez, 1999).

2.6 TÉCNICAS DE TESTE DE SOFTWARE

Nesta parte são descritas técnicas específicas de cada bateria de testes.

Os testes de integração são classificados de acordo com a abordagem escolhida para a integração. Já testes de aceitação são divididos em testes funcionais e não funcionais. E em seguida, são tratados os testes de regressão.

O tratamento dos testes de unidade é deixado para o fluxo de implementação. Esses testes geralmente são desenhados e realizados pelos próprios desenvolvedores, e não por uma equipe independente de testes, por este motivo não será visto neste texto.

2.6.1 TESTES DE INTEGRAÇÃO

O teste de integração sistematiza a atividade de integração dos módulos já submetidos ao teste de unidade, ao mesmo tempo em que as interfaces entre esses módulos são testadas. Estes testes diferem-se conforme a ordem em que são montadas as configurações de integração, ou construções, também conhecidas como *builds*. Em alguns casos, testes constantes da bateria de aceitação podem ser usados como testes de integração.

2.6.2 TESTES DE ACEITAÇÃO

Os testes de aceitação são testes de alto nível que devem ser realizados na fase inicial dos testes de software. Eles são divididos em dois itens principais, os testes funcionais e os testes não funcionais, estes dois itens serão vistos a seguir.

2.6.2.1 TESTES FUNCIONAIS

Para Ramirez (1999), os testes funcionais são desenhados para verificar a consistência entre o produto implementado e os respectivos requisitos funcionais. A completeza e a precisão da especificação dos requisitos do software são fundamentais para assegurar a qualidade desses testes. Os testes funcionais têm como suas principais características o análise do valor limite, testes de comparação e testes de tempo real.

A análise do valor limite é uma técnica para a seleção de casos de teste que exercitam os limites. O emprego dessa técnica deve ser complementar ao emprego da partição de equivalência. Assim, em vez de se selecionar um elemento aleatório de cada classe de

equivalência, selecionam-se os casos de teste nas extremidades de cada classe. Veja no Quadro 7 um exemplo de desenho de caso de teste de análise do valor limite.

Quadro 7 - DESENHO DE CASO DE TESTE DE ANÁLISE DO VALOR LIMITE

Entradas válidas	**Casos de teste**
Intervalo delimitado pelos valores *a* e *b*	Valores imediatamente abaixo de *a* e imediatamente acima de *b*
Série de valores	Valor imediatamente abaixo do mínimo, para o mínimo, para o máximo, imediatamente e acima do máximo.
Intervalo ou série de valores	Saídas máxima e mínimas definidas.
Estruturas de dados	Caso que exercite a estrutura em suas fronteiras.

Fonte: Ramirez (1990).

Existem situações em que é necessário comparar as saídas de diferentes versões de um sistema quando submetidas às mesmas entradas, estas situações são denominadas de testes de comparação. Esses testes se aplicam a situações como:

a) uso de sistemas redundantes para aplicações críticas como controle de aeronaves;
b) comparação de resultados de produtos em evolução.

Quando produtos são substituídos por versões mais novas que incluem mais funcionalidade, devem-se comparar os resultados das características que fazem parte de ambas as versões (não introduzem nova funcionalidade). Essas características já foram testadas pelos usuários com dados reais e tendem a estar mais estabilizadas. A comparação das saídas pode ser feita com o auxílio de uma ferramenta automatizada.

Nessa situação, casos de teste desenhados por meio de técnicas de caixa preta são usados para testar cada uma das versões existentes. Se as saídas forem consistentes, presume-se que todas as versões estejam corretas. Caso contrário, deve-se investigar em qual ou quais das versões se encontra o defeito.

O desenho de testes para sistemas de tempo real não pode considerar apenas casos de teste de caixa preta e branca, mas também a temporização dos dados e o paralelismo das tarefas que os manipulam. Os testes de tempo real podem ser executados através dos seguintes passos:

a) teste de tarefas isoladas, casos de testes de caixa preta e branca são desenhados para testar cada tarefa independentemente. Nessa etapa, não é possível detectar erros decorrentes de problemas de temporização. Apenas erros de lógica e funcionalidade são apontados;
b) teste comportamental, através de modelos executáveis por ferramentas de simulação é possível simular o comportamento de sistemas de tempo real e verificar como eles respondem a eventos externos. Cada um desses eventos identificados é testado individualmente, e o comportamento do sistema real é comparado ao comportamento apontado pelo modelo;
c) testes de interação entre tarefas, a fim de detectar erros de sincronização entre as tarefas, aquelas que se comunicam são testadas com diferentes taxas de dados e cargas de processamento. Devem ser desenhados casos de teste para avaliar situações de travamento, inanição e tamanho insuficiente de filas de mensagem.

2.6.2.2 TESTES NÃO FUNCIONAIS

Os testes não funcionais procuram detectar se o comportamento do software ou sistema está consistente com a respectiva especificação dos requisitos quanto aos aspectos não funcionais. Esses testes cobrem por exemplo, desempenho, dados persistentes e outros atributos, como pode ser visto no Quadro 8 com mais detalhes.

Quadro 8 - TIPOS DE TESTES DE SISTEMA

Tipos de requisitos	**Tipos de teste**
Desempenho	Número de terminais suportados Número de usuários simultâneos Volume de informação que deve ser tratado
Dados persistentes	Freqüência de uso Restrições de acesso Restrições de integridade Requisitos de guarda e retenção dados
Outros atributos de qualidade	Funcionalidade Confiabilidade Usabilidade Manutebilidade Portabilidade

Fonte: Ramirez (1990).

2.6.3 TESTES DE REGRESSÃO

As alterações feitas durante a correção dos erros detectados podem introduzir problemas em segmentos do código previamente testados. Os testes de regressão verificam novamente esses segmentos, para checar se eles continuam funcionando de maneira apropriada após a alteração de outras partes da aplicação.

Testes de regressão são particularmente importantes durante a manutenção, pois nesta fase é mais freqüente que as alterações realizadas afetem outras porções do código. São usados também durante a integração, para confirmar a estabilidade dos módulos já integrados anteriormente.

Essa técnica inclui a definição de uma bateria de testes de regressão, ou seja, um conjunto selecionado de testes de boa cobertura, que é executado periodicamente. Essa bateria geralmente é baseada nos testes de aceitação. Caso essa bateria seja muito grande, pode-se utilizar uma bateria menor com testes selecionados, para uso mais freqüente, testando-se a bateria completa em intervalos maiores. (Ramirez, 1999)

Os seguintes aspectos também devem ser verificados durante os testes de regressão:

a) se a documentação do sistema está consistente com o comportamento atual do sistema (por exemplo, se os documentos relevantes foram atualizados após as alterações realizadas);
b) se os dados e as condições de teste continuam válidos (por exemplo, mudanças em uma interface podem requerer alteração de procedimentos e casos de teste).

2.7 CASOS DE TESTE

Casos de teste são um conjunto específico de dados de teste juntamente com os resultados esperados seguindo um determinado objetivo, que pode ser a avaliação de um recurso de programa ou a verificação do atendimento de uma necessidade específica (Hetzel, 1987).

O mesmo ainda cita que, cada teste constante de uma bateria procura verificar uma característica dos itens sob teste. Tratando-se de uma bateria de testes de aceitação, cada teste geralmente verificará a implementação de um caso de uso do produto.

Um bom caso de teste tem por objetivo detectar defeitos ainda não descobertos, ao invés de apenas demonstrar que o programa funciona corretamente. Ele deve obrigatoriamente incluir uma descrição das saídas esperadas, que será usada para comparação com as saídas reais obtidas em cada execução do caso de teste. Devem existir casos de teste que cubram tanto entradas válidas quanto inválidas.

Os casos de teste devem cobrir as combinações de entrada relevantes para dar ao teste uma cobertura adequada. Além disso, é conveniente prever casos de teste para execução de procedimentos errados mas possíveis. Cada caso de teste deve corresponder a um conjunto de entradas e saídas esperadas. (Ramirez, 1999)

É conveniente classificar os casos de teste em categorias bem definidas. Uma classificação importante baseia-se na origem dos dados de teste (Hetzel, 1987). Os tipos principais, dentro dessa classificação incluem:

a) casos funcionais (derivados das especificações ou do entendimento do que o programa deve realizar);
b) casos estruturais (derivados da estrutura lógica da codificação e das instruções do programa);
c) casos de dados (derivados do entendimento e da definição dos elementos de dados e dos arquivos usados no programa);
d) casos aleatórios (derivados de uma técnica de aleatorização ou amostragem, como os produzidos por geradores paramétricos de casos de teste);
e) casos extraídos (aproveitados de outros sistemas ou arquivos de produção);
f) casos anormais ou extremos (selecionados numa tentativa deliberada de vencer o sistema, incluindo elementos como condições limítrofes e situações extremas).

2.7.1 IDENTIFICAÇÃO DE CASOS DE TESTE

Ainda seguindo a idéia do autor, as técnicas e metodologias de teste de software vistas até agora neste trabalho, objetivam a criação de casos de teste que nos permitem e identificar possíveis "bugs" dos sistemas testados. Neste contexto casos de teste são descritos por dois elementos, um conjunto de ações ou valores de entrada e um conjunto de ações ou resultados esperados.

Esse conjunto de informações, porém, não é suficiente para a aplicação do caso de teste. É preciso considerar que, a pessoa que determina o caso de teste não é, obrigatoriamente, a mesma que efetivamente irá aplicar os casos de teste na busca por "bugs" no sistema. Outro fator importante que deve ser considerado, é que pode se passar um intervalo de tempo razoável (dias, semanas, meses) entre o desenvolvimento dos casos de teste e sua efetiva aplicação.

Ë fundamental que um caso de teste seja repetível, ou seja, se pessoas distintas em momentos e lugares distintos aplicarem o mesmo caso de teste sobre a mesma versão de um produto, é imprescindível que os resultados encontrados sejam rigorosamente os mesmos. Isso é fundamental, principalmente, para que a equipe de desenvolvimento possa reproduzir os "*bugs*" relatados pela equipe de testes.

Face a isso, é razoável exigir uma documentação mais completa para cada caso de teste além da simples determinação de um conjunto de valores de entrada e de seus respectivos resultados esperados.

Normalmente as expressões "teste" e "caso de teste" são usadas indistintamente como sinônimos. Neste contexto iremos usá-las com significados distintos. Ao usar "casos de teste" estaremos nos referindo ao resultado das técnicas vistas até agora, ou seja composto por um conjunto de entradas e os respectivos resultados esperados. Ao usar a expressão teste, porém, estaremos nos referindo a descrição documentada de um teste, composta por todas as informações necessárias para que o mesmo possa ser executado e repetido sem problemas (note que por repetido entende-se chegar sempre ao mesmo resultado esperado).

2.7.2 MODELO PARA CRIAÇÃO DE UM CASO DE TESTE

A partir das características de um "teste" é possível definir diferentes modelos para sua descrição. Visando uniformizar o trabalho é proposto um modelo de referência a ser seguido em todos os casos de teste. Este modelo tem seus itens proposto por Oliveira (1997) e pode ser visto no Quadro 9 adaptado para o Radar como forma de exemplificação.

Quadro 9 – EXEMPLO DE UM MODELO DE CASO DE TESTE

Produto: Radar Contábil
Numero do teste: *01*
Dependências:
Objetivo: *verificar se o sistema grava corretamente o Plano de Contas Contábil.*
Ambiente:

Hardware: micro PC compatível com pelo menos 25Mb livres em HD
Software:

- *win95 ou superior*
- *Radar Contábil instalado no diretório c:\wkradar*

Passos:

1. *executar o Radar Contábil*
2. *Selecionar a Empresa "WKDemo"*
3. *observar o resultado esperado 2*
4. *selecionar o menu, Cadastro/Plano Empresarial/Contábil*
5. *observar o resultado esperado 4*
6. *selecionar o menu Editar/Incluir*
7. *Preencher o campo Classificação conforme a primeira linha do relatório de Plano de Contas a ser incluído.*
8. *Preencher o campo Descrição conforme a primeira linha do relatório de Plano de Contas a ser incluído.*
9. *Selecionar o(s) CheckBox referentes à característica da conta conforme a primeira linha do relatório de Plano de Contas a ser incluído.*
10. *Pressiona o Botão "OK"*
11. *Observar os resultados esperados 7, 8 e 9*

Resultados esperados:

1. O Plano de Contas deve ter sido gravado exatamente como foi digitado.

Fonte: Oliveira (1997).

2.8 NORMA BRASILEIRA ABNT NBR 12207

Conforme a ABNT (1998), software é uma parte fundamental da tecnologia de informação e de sistemas convencionais, tais como sistemas de transporte, militares, da área médica e financeiros. Tem havido uma proliferação de normas, procedimentos, métodos, ferramentas e ambientes de desenvolvimento e de gerência de software. Esta proliferação tem criado dificuldades na gerência e engenharia de software, principalmente na integração de produtos e serviços. A disciplina de software necessita migrar desta proliferação para uma estrutura comum que possa ser usada por profissionais de software para criar um padrão na criação e gerência de software. Esta Norma provê tal estrutura comum.

A estrutura cobre o ciclo de vida de software desde a concepção de idéias até a descontinuação do software, e consiste dos processos de aquisição e fornecimento de produtos e serviços de software. Adicionalmente, a estrutura provê o controle e a melhoria destes processos.

Os processos desta Norma formam um conjunto abrangente. Uma organização, dependendo de seu objetivo, pode selecionar um subconjunto apropriado para satisfazê-lo. Esta norma é, portanto, projetada para ser adaptada para uma organização, projeto ou aplicação específicos. Também é projetada para ser utilizada quando o software é uma entidade independente ou embutida ou integrada a um sistema.

Esta norma estabelece uma estrutura comum para os processos de ciclo de vida de software, com terminologia bem definida, que pode ser referenciada pela indústria de software. A estrutura contém processos, atividades e tarefas que servem para ser aplicadas durante a aquisição de um sistema que contém software, de um produto de software independente ou de um serviço de software, e durante o fornecimento, desenvolvimento, operação e manutenção de produtos de software.

Esta norma também provê um processo que pode ser utilizado para definir, controlar e melhorar os processos no ciclo de vida de software. Porém neste trabalho será dado enfoque apenas às partes que dizem respeito à teste de software, o processo de desenvolvimento, processo de operação e processo de apoio ao ciclo de vida.

2.8.1 PROCESSO DE DESENVOLVIMENTO

O processo de desenvolvimento contém as atividades e tarefas do desenvolvedor. O processo contém as atividades para análise de requisitos, projeto, codificação, integração, testes, e instalação e aceitação relacionada aos produtos de software. Pode conter atividades relacionadas ao sistema, se estipulado no contrato. O desenvolvedor executa ou apoia as atividades neste processo, de acordo com o contrato.

Este processo consiste em diversas atividade, dentre as quais pode-se destacar três que são pertinentes à atividade de teste de software:

a) codificação e testes do software;
b) integração do software;
c) teste de qualificação do software.

A codificação e testes do software é uma atividade que deve ser realizada para cada item de software (ou item de configuração de software, se identificado) e consiste nas seguintes tarefas para o desenvolvedor de software:

a) deve desenvolver e documentar, cada unidade de software e base de dados e os procedimentos de teste e dados para testar cada uma dessas unidades e bases de dados;
b) deve testar cada unidade de software e base de dados, garantindo que sejam atendidos seus requisitos. Os resultados dos testes devem ser documentados;
c) deve atualizar a documentação do usuário, quando necessário;
d) deve atualizar os requisitos de teste e o cronograma, para integração do software;
e) deve avaliar o código do software e os resultados dos testes. Os resultados das avaliações devem ser documentados.

A atividade de integração do software, por sua vez deve ser realizada para cada item de software (ou item de configuração de software, se identificado) e consiste nas seguintes tarefas para o desenvolvedor de software:

a) deve desenvolver um plano de integração para integrar as unidades de software e componentes de software no item de software. O plano deve incluir requisitos de teste, procedimentos, dados, responsabilidades e cronograma e tudo deve ser documentado;
b) deve integrar as unidades e componentes de software e testar essas agregações à medida que forem sendo integradas, de acordo com o plano de integração. Deve ser garantido que cada agregação atenda os requisitos do item de software e que o item de software esteja integrado na conclusão da atividade de integração. Os resultados da integração e dos testes devem ser documentados;
c) deve atualizar a documentação do usuário, quando necessário;
d) deve desenvolver e documentar, para cada requisito de qualificação do item de software, um conjunto de testes, casos de teste (entradas, saídas e critérios de teste), e procedimentos de teste, para conduzir o teste de qualificação do software. O desenvolvedor deve garantir que o item de software integrado está pronto para o teste de qualificação do software;
e) deve avaliar o plano de integração, projeto, código, testes, resultados dos testes e a documentação do usuário. Os resultados das avaliações devem ser documentados.

A atividade de teste de qualificação do software deve ser realizada para cada item de software (ou item de configuração de software, se identificado) e consiste nas seguintes tarefas para o desenvolvedor de software:

a) deve conduzir o teste de qualificação de acordo com os requisitos de qualificação para o item de software. Deve ser garantido que a implementação de cada requisito do software seja testada para conformidade. Os resultados do teste de qualificação devem ser documentados;
b) deve atualizar a documentação do usuário, quando necessário;
c) deve avaliar o projeto, código, testes, resultados dos testes e a documentação do usuário. Os resultados das avaliações devem ser documentados;
d) deve apoiar auditorias. Os resultados das auditorias devem ser documentados. Se ambos, hardware e software, estão sendo desenvolvidos e integrados, as auditorias podem ser adiadas até o teste de qualificação do sistema.

2.8.1.1 PROCESSO DE OPERAÇÃO

O processo de operação contém as atividades e as tarefas do operador. O processo cobre a operação do produto de software e o suporte operacional aos usuários. Como a operação do produto de software está integrada à operação do sistema, as atividades e tarefas deste processo se referem ao sistema. Este processo tem como atividade pertinente ao de teste de software, o teste operacional.

Para cada liberação do produto de software, o operador deve executar o teste operacional e satisfazendo os critérios especificados, liberar o produto de software para uso operacional. O operador deve garantir que o código de software e as bases de dados sejam iniciados, executados e finalizados, como descrito no plano.

2.8.1.2 PROCESSOS DE APOIO DE CICLO DE VIDA

Este capítulo da norma, define os seguintes de apoio de ciclo de vida. As atividades e tarefas num processo de apoio são de responsabilidade da organização que o executa. Essa organização garante que o processo existe e é funcional.

O processo de validação é um processo para determinar se os requisitos e o produto final, sistema ou produto de software construído, atendem ao uso específico

pretendido. A validação pode ser conduzida nos estágios iniciais. Este processo pode ser conduzido como parte da atividade apoio à aceitação do software.

Este processo pode ser executado com variados graus de independência. O grau de independência pode variar da mesma pessoa ou outra pessoa da organização, para uma pessoa de outra organização. No caso em que o processo é executado por uma organização independente do fornecedor, desenvolvedor, operador ou mantenedor, é chamado de Processo de Validação Independente.

Este processo está dividido nas seguintes atividades, implementação do processo e a validação propriamente dita.

A implementação do processo consiste nas seguintes tarefas:

a) deve ser determinado se o projeto justifica um esforço de validação e o grau de independência organizacional. Os requisitos do projeto devem ser analisados em função dos fatores críticos. Estes fatores podem ser aferidos nos seguintes termos:
   - O potencial de que um erro não detectado em um requisito do sistema ou software possa causar morte ou dano pessoal, não alcance de objetivos, perda ou dano financeiro ou de equipamento;
   - A maturidade e riscos associados com a tecnologia de software a ser utilizada;
   - A disponibilidade financeira e de recursos.

b) Se o projeto justifica um esforço de validação, um processo de validação deve ser estabelecido para validar o sistema ou o produto de software. As tarefas de validação definidas a seguir, incluindo métodos, técnicas e ferramentas associados para executar as tarefas, devem ser selecionadas;

c) Se o projeto justifica um esforço de validação independente, deve ser selecionada uma organização qualificada responsável para conduzi-la. O condutor deve ter assegurada a independência e autoridade para executar as tarefas de validação;

d) Um plano de validação deve ser desenvolvido e documentado. O plano deve incluir, mas não estar limitado ao seguinte:

- itens sujeitos à validação;
- tarefas de validação a serem executadas;
- recursos, responsabilidades e cronograma para validação;
- procedimentos para encaminhar relatórios de validação ao adquirente e outras partes envolvidas.

O plano de validação deve ser implementado. Problemas e não-conformidades detectados pelo esforço de validação devem ser incluídos no processo de resolução de problemas. Todos os problemas e não-conformidades devem ser resolvidos. Os resultados das atividades de validação devem ser disponibilizados para o adquirente e outras organizações envolvidas. O processo de validação consiste nas seguintes tarefas:

b) preparar os requisitos de teste, casos de teste e especificações de teste selecionados para análise dos resultados dos testes;

c) assegurar que estes requisitos de teste, casos de teste e especificações de teste reflitam os requisitos particulares para o uso específico pretendido;

d) conduzir os testes nos dois itens anteriores, incluindo:
   - teste de estresse, limites e entradas específicas;
   - teste do produto de software para verificar sua habilidade em isolar e minimizar efeitos de erros; isto é, degradação suave em caso de falha, pedido de assistência do operador em caso de estresse, de exceder limites e de condições específicas;
   - teste para que usuários representativos possam executar, com sucesso, suas tarefas pretendidas usando o produto de software.

e) Validar um produto de software, significa garantir que ele satisfaça seu uso pretendido. Testar o produto de software, quando apropriado, nas áreas selecionadas do ambiente alvo.

2.9 A FERRAMENTA RATIONAL VISUAL TEST 6.5

O *Rational Visual Test,* é uma ferramenta de automatização de testes onde o programador de testes dispõe de diversos recursos ferramentas que o permitem criar os mais diversos tipos de caso de teste possível. Este capítulo mostrará quais são essas ferramentas, para que servem e os principais recursos do *Visual Test,* bem como considerações sobre automação dos testes de software e outras ilustrações rápidas sobre a ferramenta *Visual Test*.

2.9.1 AUTOMAÇÃO DO TESTE DE SOFTWARE

Durante os últimos anos, testar software se tornou uma tarefa complexa. A linha de sistemas operacionais *Windows* necessita de uma programação complexa, fazendo com que os sistemas se tornem mais vulneráveis a *Bugs*. Para que se possa encontrar rapidamente e com eficácia esses *Bugs* e para que se possa abordar os possíveis problemas de desempenho do sistema, é preciso, em um mercado de desenvolvimento de software competitivo como os de hoje, automatizar o processo de teste de software. Para isso, disponibiliza-se ferramentas adequadas à este tipo de atividade (Arnold, 1999).

Porém, nenhuma ferramenta de automatização do processo de teste de software é perfeita, automatização leva tempo, esforço e compromisso ali envolvido, inclusive uma compreensão sobre o que realmente uma automatização pode e não pode fazer.

Geralmente acredita-se que automatização de teste significa substituir o papel humano de teste. Testes automatizados são uma ajuda na execução dos testes somente, porém eles não substituem o testador humano, que tem que criá-los, e conferi-los para verificar se funcionam satisfatoriamente, e pensar em outros itens que precisam ser testados para assegurar a qualidade do software. A automatização é um ponto forte na re-execução com rapidez dos testes inúmeras vezes da mesma forma. Desta maneira, os *Bugs* são encontrados mais cedo permitindo que o software seja produzido com mais qualidade. Testar, pode levar 80% do tempo da produção do software.

Enfim, automatização de testes, emula ações de um usuário, digitação, entrada de dados, cliques em botões etc. Automatizar, gera velocidade e agilidade à essas tarefas tediosas e repetitivas e também assegura que essas tarefas seja repetidas inúmeras vezes na mesma ordem dando mais qualidade e consistência ao teste.

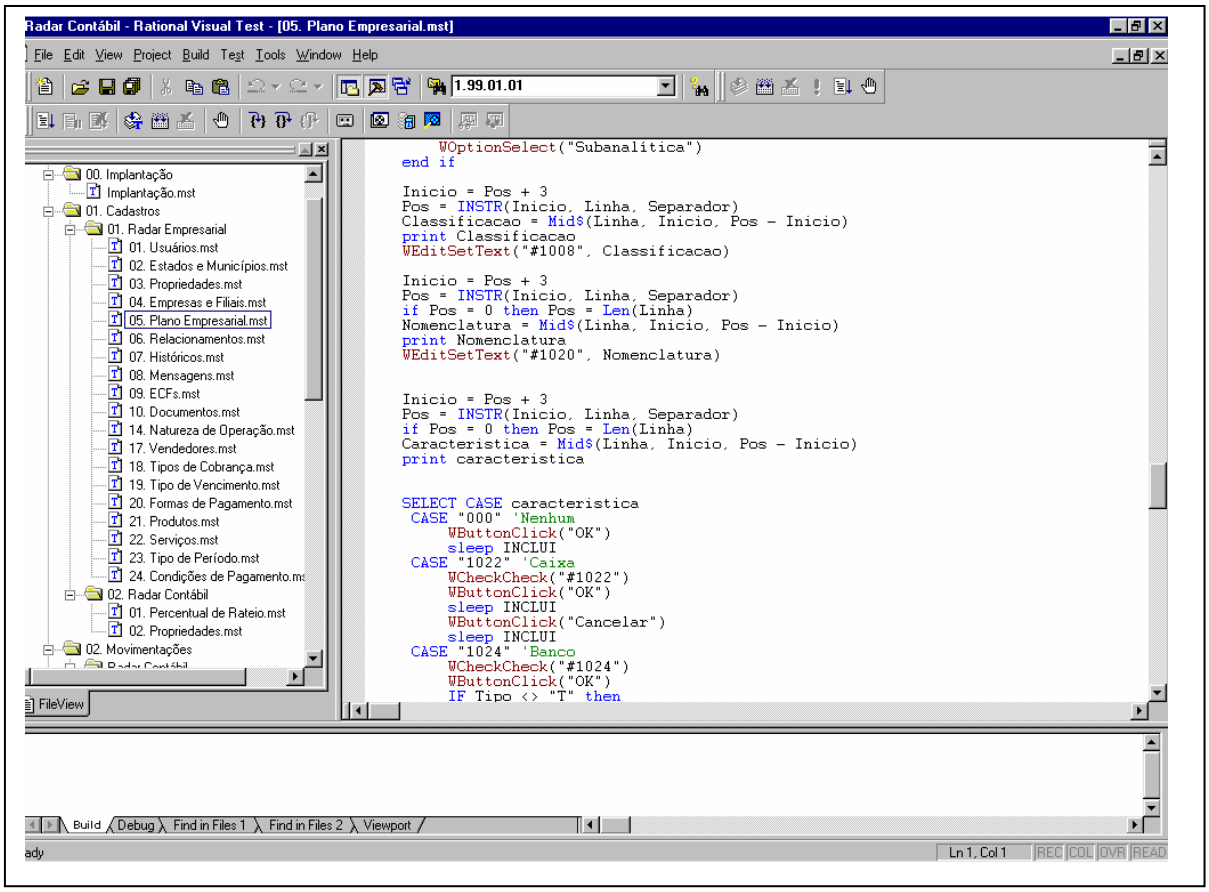
Destaca-se algumas razões chaves de porque deve-se automatizar os testes de software e utilizar uma boa ferramenta para esta atividade. Segundo Arnold (1999), a realização de testes de regressão; fácil repetição da reprodução dos passos de um teste; testes de compatibilidade e reuso de scripts em diversos teste, plataformas e casos diferentes.

2.9.2 INTERFACE DO *VISUAL TEST*

Especificamente o *Visual Test* da Rational, oferece benefícios consideráveis sobre esses assuntos mencionados anteriormente. O *Visual Test* está inserido em uma plataforma semelhante à do *Developer Studio Visual C++*, que não só é usado pelo *Visual Test* como também é o editor principal do *Microsoft Visual Studio*, um agrupamento das ferramentas de desenvolvimento da Microsoft. Devido a este fato, ele se adapta melhor a programas desenvolvido em *Visual C++*.

A interface do *Visual Test*, que pode ser vista na Figura 1 que está divida em três parte principais: *Workspace*, *Output*, *Source File*. A primeira, *Worspace* ou Área de Trabalho, mostra uma representação clara da hierarquia de arquivos do projeto. Isso faz com que seja mais fácil organizar os arquivos relacionados a um determinado projeto de teste. *Output* é a parte onde são mostradas as informações sobre erros encontrados durante a compilação e/ou execução de uma caso de teste. Por fim, a parte onde se trata do *Source File*, corresponde à janela utilizada para a digitação do código fonte propriamente dito, os *scripts*.

Figura 1 – INTERFACE DO VISUAL TEST



2.9.3 LINGUAGEM DO *VISUAL TEST*

A linguagem utilizada pelo *Visual Test* é uma versão robusta do *Basic* e oferece bastante flexibilidade para escrever *scripts* complexos que satisfaçam as necessidades de teste. Porém, pode-se destacar alguns comandos essenciais e indispensáveis para a implementação de testes:

a) WEditSetText – comando utilizado para inserir um dado qualquer em um campo editável do tipo *Edit*. Tem como sua sintaxe, WeditSetText(#ID, “Dado”) onde o ID é o número de identificação do campo e o Dado, é o dado propriamente dito a ser inserido no campo;
b) WComboItemClk – comando utilizado para um item qualquer em um *combo box*. Tem como sua sintaxe, WComboItemClk(#ID, “Item”) onde o ID é o número de identificação do *combo box* e o Item, é o item propriamente dito a ser selecionado no *combo*;
c) WCheckCheck – comando utilizado para marcar um *check box*. Tem como sintaxe, WcheckCheck(#ID) onde o ID é o número de indentificação do *check box*. Temos também o WcheckUnCheck para desmarcar um *check box* utilizando a mesma sintaxe do anterior;
d) WoptionSelect – comando utilizado para selecionar uma opção única em um *radio button*. Tem como sintaxe WoptionSelect(#ID), onde o ID é o número de identificação do campo;
e) WbuttonClick – comando utilizado para clicar em um botão qualquer, tem como sintaxe, WbuttonClick(“nome_botão ou #ID”) onde o nome_botão é o nome do botão propriamente dito ou ainda pode-se utilizar o ID, que é o número de identificação do mesmo;
f) WMenuSelect - comando utilizado para acessar menus, tem como sintaxe, WmenuSelect (“caminho_do_menu”) onde o caminho_do_menu representa exatamente os nomes dos menus a serem acessados. Ex:. WMenuSelect (“&Arquivo/Sa&ir”), onde o “&” representa o atalho para o mesmo.
g) Sleep – comando utilizado para parar a execução do teste por um tempo determinado, tem-se a seguinte sintaxe, Sleep X, onde X representa em segundos o tempo em que a execução ficará parada, o parâmetro de tempo

pode ser utilizado em forma de constante ou então diretamente o número em segundos representado o tempo desejado.

2.9.4 UTILITÁRIOS DO *VISUAL TEST*

A ferramenta *Visual Test* conta com o apoio de alguns utilitários que facilitam a implementação dos *Scripts*. Os três principais utilitários são:

a) gravador de cenário(*Scenario Record*) quando ativado, este utilitário, trabalha como um gravador de todas as ações do usuário a partir daquele momento. Ele localiza todas as ações e as transforma em linguagem de teste para o próprio *Visual Test*. Uma vez terminada esta gravação, o script já está gerado e pode ser executado, e todas as ações executadas anteriormente pelo usuário e gravadas, serão repetidas pela execução do script no *Visual Test.*

b) utilitário de informações de janela (*Window Information Utility*). Este, por sua vez, traz informações detalhadas de uma determinada janela. Após sua execução, é aberta uma janela onde, através de um cursor, é possível identificar nomes de campos das janelas do sistema testado, bem como seus ID's (números de identificação gerados para cada campo do sistema), a que módulo pertence, entre outras informações.

c) *Suite Manager*, este funciona como um gerenciador de execução dos diversos casos de teste gerados em um projeto. Através de seu uso, é possível executar em uma seqüência qualquer desejada pelo programador de teste, os seus diversos arquivos de caso de teste, sem que os mesmos sejam executados um por um separadamente.

2.9.5 EXECUÇÃO DOS *SCRIPTS*

Como já foi visto anteriormente, os scripts de teste são executados no próprio *Visual Test*, mas isso não quer dizer que necessariamente precisa-se desta ferramenta para executar um determinado caso de teste.

O *Visual Test* conta com um recurso de criação de executáveis dos Scripts que podem ser executados independentemente de se ter ou não o *Visual Test* instalado na máquina. Isto faz com que os testes criados para um computador possam ser "carregados" para outras máquinas e executados nelas, aumentando assim a área de cobertura dos testes no

que diz respeito ao comportamento do software testado em diversas configurações de máquinas e sistemas operacionais possíveis.

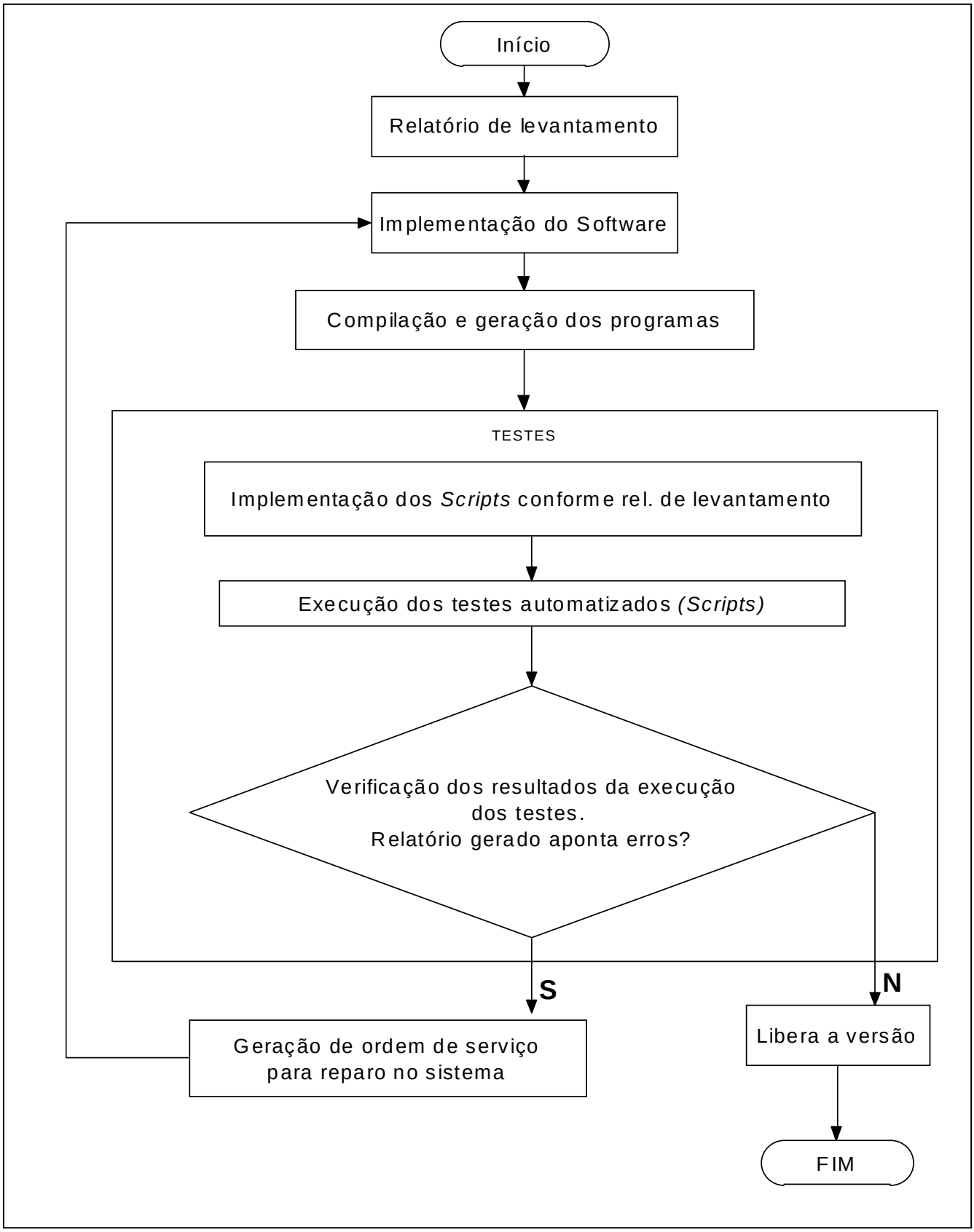
3. DESENVOLVIMENTO

Para o desenvolvimento deste projeto, tomou-se como ponto de partida os métodos utilizados pela WK WK Sistemas na atividade de teste de software. Permitindo com isto, verificar as deficiências em relação as normas e/ou técnicas estudadas neste trabalho de conclusão de curso, conforme apresentado no capítulo 2. A WK WK Sistemas é uma *software house* com sede em Blumenau e existe desde 1985. Projetou-se no mercado nacional ganhando prêmios com sistemas contábeis. Hoje, a empresa produz softwares de gestão empresarial que utilizam das últimas tecnologias disponíveis no mercado. A empresa conta com o trabalho de mais de 50 funcionários que contribuem na produção dos mais de 13 produtos de software desenvolvidos pela mesma.

A partir disso, iniciou-se então um estudo de normas e técnicas que possam uniformizar e melhorar esta atividade tão importante no ciclo de vida do software, que hoje não é vista como muito importante pelas *software houses*.

Na figura 2, é demonstrado o funcionamento da atividade de teste de software hoje empregado pela WK Sistemas. Este fluxograma demonstra simplificadamente as etapas de desenvolvimento do de software dando mais ênfase à atividade de teste.

Figura 2 – ETAPA DE DESENVOLVIMENTO DE SOFTWARE



O fluxograma mostrado pela Figura 2 pode ser interpretado mais facilmente através do texto a seguir:

Para iniciar um projeto de software é necessário cumprir-se várias etapas. A primeira delas é o levantamento das informações. Esta etapa consiste em especificar detalhadamente todos os requisitos básicos e necessidades do sistema, ou seja, funcionalidades, operações atendimento às exigências legislativas quando necessário, entradas e saídas de dados. A partir daí então, parte-se para a especificação e redação do relatório de levantamento. Nesta especificação, será descrito como se dará a criação dos arquivos de dados de entrada do sistema a ser testado bem como em que momento acontece a integração entre os módulos do sistema testado. O relatório de levantamento terá como conteúdo as informações obtidas na etapa de especificação e traz uma descrição formal da funcionalidade do sistema de forma que o programador possa implementá-lo de acordo com a especificação de cada etapa.

Com a especificação pronta, equipes de programadores iniciam a implementação de cada módulo do sistema. Assim que é liberada pelos programadores, a implementação é compilada e são gerados os programas executáveis, estes são encaminhados à equipe de teste juntamente com os relatórios de levantamento para as devidas conferências. É aí então que inicia-se uma das etapas mais importantes do desenvolvimento de sistemas, a etapa de testes.

A etapa de testes conta também com tecnologia de ponta. Uma ferramenta chamada *Visual Test* é utilizada para a implementação de *Scripts* que simulam a entrada de dados de um usuário qualquer (Arnold, 1999).

Por fim, implementa-se também rotinas que verificam valores calculados, dados gravados e toda e qualquer saída de resultados que o sistema em questão pode gerar. Para implementar este *Scripts*, deve-se fazer um estudo prévio do sistema através dos relatórios de levantamento para se poder saber como, onde, e de que maneira os dados digitados pelo usuário devem ser gravadas e ou calculadas. Utiliza-se também os casos de uso como fonte de informações para implementar um roteiro de testes. Os casos de uso são descritos pelos gerentes de produto e ou analistas e retratam a realidade do sistema de forma que contenha as informações que devem ser utilizadas para cada cadastro simulando a utilização do sistema por parte de um usuário final/cliente. É indispensável também que os casos de uso contenham

todas as informações sobre integrações, cálculos, tamanhos e tipos de cada campo de entrada e/ou armazenamento de dados, arquivos que são criados no decorrer dos cadastros e movimentações, resultados finais em relatórios e seqüência da implantação dos dados. Com todas essas informações bem detalhadas é possível implementar um roteiro de testes eficiente.

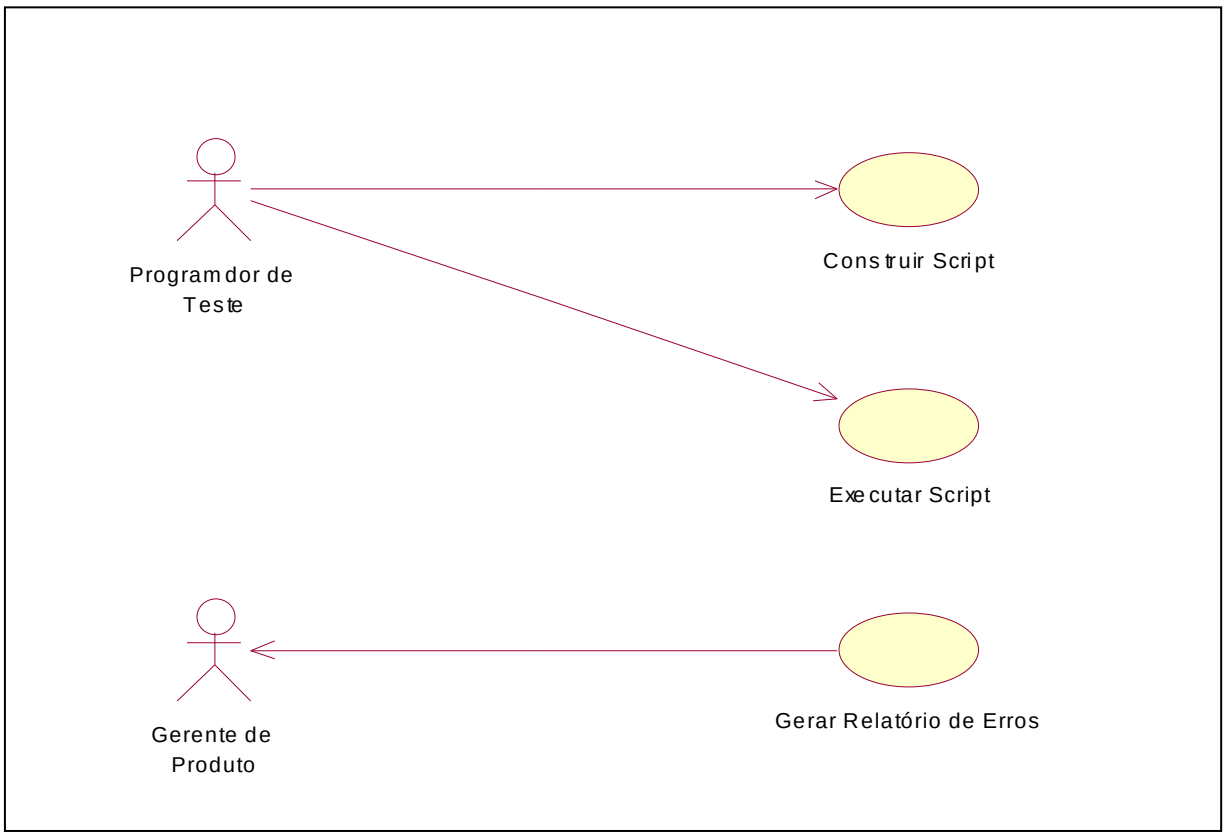
Uma vez implementados, os *scripts* de teste são executados. Eles geram um arquivo de relatório com os resultados da execução. Por exemplo, se algum procedimento especificado no teste não foi feito corretamente pelo software testado, este relatório de resultado apontará qual a falha, seja ela de gravação incorreta de algum dado ou de algum processo gerado incorretamente pelo sistema como cálculos e ou integrações. Estes resultados são analisados e, se nenhum erro for encontrado no sistema, este pode ser liberado para o mercado, caso contrário, o sistema volta para a programação com o resultado dos testes apontando os erros encontrados. Estes erros são corrigidos e o sistema retorna mais uma vez para os testes. Visto que os *Scripts* já estão prontos, basta executá-los novamente. É recomendável que os *scripts* sejam executados novamente do início ao fim e não apenas a partir da parte corrigida pelo programador, pois uma alteração no sistema, por menor que seja, pode afetar todo o programa bem como seu desempenho e funcionalidade. Eis aí uma imensa vantagem em se automatizar os testes de software. O teste de software geralmente é trabalhoso na sua implementação, entretanto quando pronto, pode ser repetido tantas vezes quantas forem necessárias sem qualquer trabalho adicional de implementação e com muita rapidez e agilidade.

Nos próximos capítulos, serão apresentados de que maneira chegou-se a uma solução automatizada de teste de software baseando-se em normas e técnicas específicas à este tipo de atividade. Estas normas e técnicas foram estudadas ao longo do desenvolvimento deste trabalho e tomadas como referência para a implementação do protótipo.

3.1 ESPECIFICAÇÃO

Apresenta-se na especificação, um diagrama que esquematiza a realização dos testes de software de uma forma automatizada. Este diagrama, representado pela Figura 3, demonstra a funcionalidade do protótipo.

Figura 3 – DIAGRAMA USE-CASE DO PROTÓTIPO



Programador de teste: responsável pelo desenvolvimento dos *scripts* de teste, bem como sua execução.

Construir *Script*: esta ação resume-se pela codificação dos scripts de teste. Esta codificação é feita pelo programador de teste utilizando-se a ferramenta *Visual Test.*

Executar *Script*: esta ação representa a execução dos scripts de teste codificados na etapa anterior sobre o software submetido ao teste. Neste caso, o Radar Contábil.

Gerar Relatório de Erros: esta ação é responsável pela geração de um relatório em formato de arquivo texto, com os resultados obtidos no teste. Este relatório contém todos os erros encontrados durante a execução do teste no referido software testado.

Gerente de Produto: Ator responsável pela análise e avaliação dos resultados obtidos com a execução do teste.

Partindo da funcionalidade deste diagrama, deve-se implementar uma solução automatizada de teste de software seguindo a especificação a seguir.

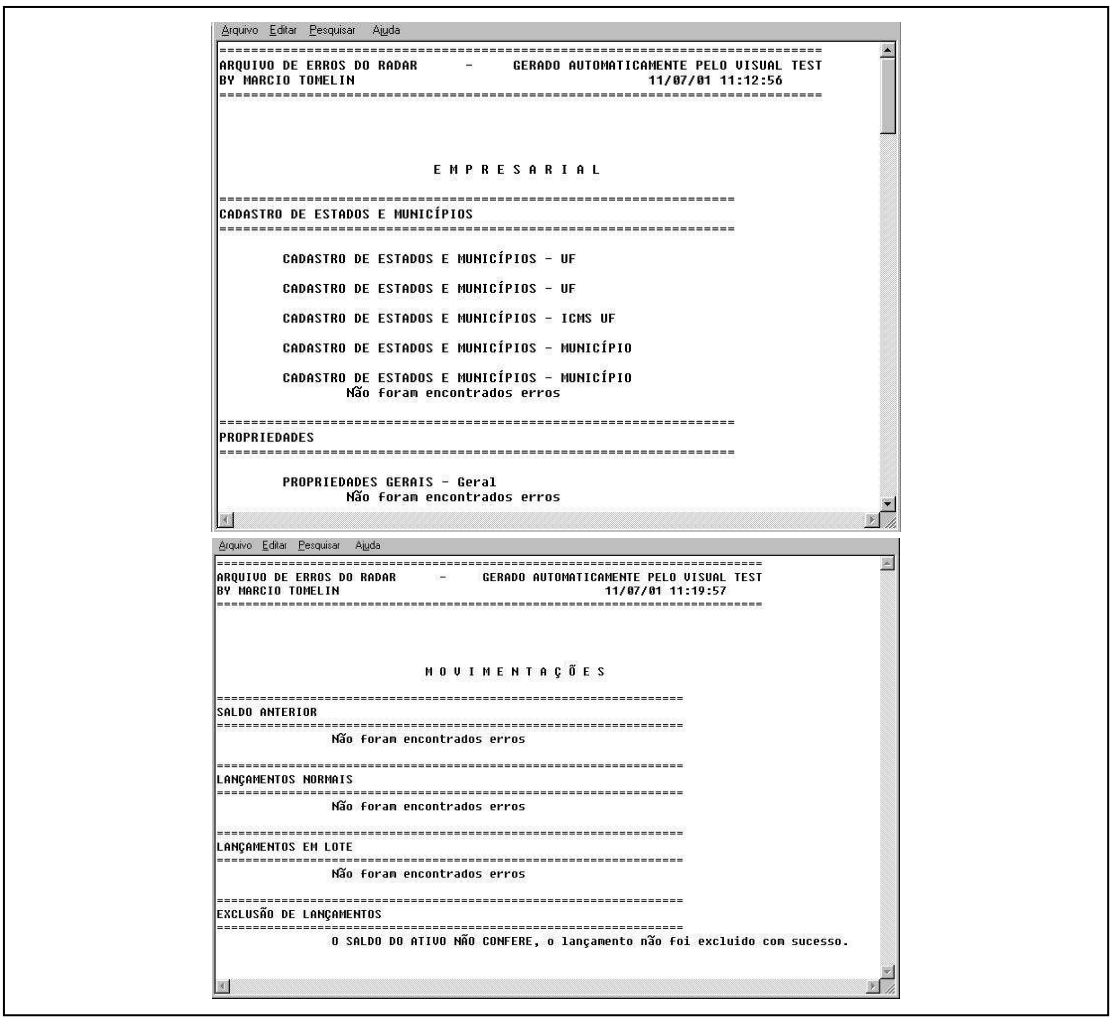
O software a ser submetido ao teste é o Radar Contábil da empresa WK Sistemas. Deve-se implementar um conjunto de scripts de teste que o alimentem a partir de uma base vazia com informações necessárias a fazer movimentações contábeis. Em cada caso de teste, cada um correspondendo a um *Script* implementado, deve ser incluída a restauração da base de dados gerada pelo caso de teste anterior, no início de sua execução. Ao seu final, deve-se efetuar o backup do estado atual da base de dados para que possa ser utilizado no caso de teste seguinte.

Depois de todos os cadastros terem sido implementados, deve-se partir para a movimentação contábil propriamente dita, ou seja, lançamentos contábeis de recebimentos, pagamentos, transferência entre outros. Como última etapa, tem-se as verificações. Esta é a etapa mais importante, nela deverão ser analisados todos os campo de cada cadastro. Se algum dado tiver sido gravado incorretamente, o sistema automatizado de teste deve informar em um relatório de erros qual o campo e de que cadastro não houve sucesso na gravação

Para as movimentações, o sistema de automatização de teste deve fazer as movimentações propriamente ditas, verificar as gravações dos registros das mesmas através dos saldos contábeis, executar a exclusão de um registro verificar novamente o saldo. Se encontrado algum erro durante em alguma destas operações, o sistema de testes deve registrar a descrição do mesmo no relatório de erros.

Este relatório de erros, é um arquivo de formato texto, seu modelo pode ser visto através da Figura 4.

Figura 4 – MODELO DE RELATÓRIO DE RESULTADOS DOS TESTES



O protótipo ainda deve conter uma solução automatizada para os testes de aceitação, regressão e integração conforme especificado pela norma estudada neste trabalho. Além disso os testes de análise do valor limite, testes de desempenho e restrições de integridade, todos estudados neste trabalho, devem ser implementado no protótipo.

3.2 IMPLEMENTAÇÃO

Conforme o proposto pelo trabalho, a implementação do protótipo baseou-se nas normas e técnicas de teste de software estudadas neste trabalho de conclusão de curso. Seguindo esta idéia, este capítulo apresentará de que maneira o software implementado para automatização de teste atende às normas e técnicas estudadas.

Conforme apresentado no capítulo 2.2, o objetivo central de um teste de software é descobrir erros, e partindo deste princípio que esta implementação foi iniciada. Os testes

foram implementados para encontrar erros no sistema, porém o sistema submetido ao teste, o Radar Contábil, não apresentou defeitos. Mas, como Souza (2000) afirma, é impossível testar um software completamente, portanto a não descoberta de defeitos no Radar Contábil, não significa que este esteja livre de *bugs*.

Segundo as metodologias de teste de software propostas pela IEEE (1994), deve-se fazer um comparativo dos resultados previstos com os obtidos na execução do teste. Neste projeto o comparativo foi feito na própria implementação do teste automatizado, onde o valor esperado para o saldo de uma determinada conta contábil, é comparado com o obtido na execução do teste. Se este valor não for idêntico ao esperado, um *log* sobre o erro é gravado no arquivo texto que contém o resultado completo do teste. Ainda seguindo as metodologias de teste de software, tem-se basicamente duas maneiras de se construir testes, os testes de caixa branca e os de caixa preta. Os testes de caixa branca, que agregam os testes de unidade, como já foi visto anteriormente têm por objetivo testar a estrutura interna do sistema, ou seja, o código fonte, por este motivo não foi implementado, pois a ferramenta de automatização de testes utilizada, o *Visual Test*, não permite tal implementação. Este teste deve ser feito pelo programador, assim como os testes de unidade que são parte integrante da metodologia de teste de caixa branca. Já os teste de caixa preta, agregam os testes de aceitação e regressão cuja forma de implementação no protótipo será vista adiante.

As bateias de testes estudadas são os testes de unidade, testes de integração, testes de aceitação e regressão. Os testes de unidade, como vimos anteriormente não foram executados, pelo motivo de ser teste que exige a disponibilidade do código fonte do produto, na qual não teve-se acesso parta este projeto, e pelo fato de a ferramenta de automatização de testes utilizada neste projeto, não permitir este tipo de teste. As demais baterias de teste podem ser vista a seguir.

O teste de regressão é o mais indicado para a ser automatizado, pois seu objetivo é testar o sistema por completo partido de uma base de dados vazia. Sendo assim, o protótipo implementado atende perfeitamente a este teste. Por ser um teste completo do sistema, não tem-se uma parte específica do protótipo que implementa este teste, a execução completa do projeto de teste, ou seja, todos os *scripts*, implementam a solução para este teste.

Para o teste de aceitação pode-se apontar uma parte do protótipo como sendo fundamental para este teste. Como o sistema submetido ao teste é um sistema contábil, para implementar o teste de aceitação, tomou-se como base os saldos contábeis, estes indicam se a finalidade do sistema que é de contabilizar valores, está de acordo com o que é previsto pela legislação. A codificação referida a este teste pode ser vista a seguir no Quadro 10.

Os testes de integração estão implicitamente inseridos ao longo de todo o protótipo. Como já visto anteriormente, eles têm como objetivo verificar se as unidades que compõem cada liberação funcionam corretamente em conjunto com as unidades já liberadas. Contudo, este teste é executado implicitamente ao se executar um teste completo do sistema.

Quadro 10 – CODIFICAÇÃO PARA O TESTE DE ACEITAÇÃO

```
Scenario "Verifica Saldo"

    'VERIFICA SALDO DO ATIVO
    WEditSetText("#1211","17") 'Conta
    sleep 1
    Play "{TAB}"
    WEditSetText("#1212","01/01/99") 'Data Inicial
    sleep 1
    Play "{TAB}"
    WEditSetText("#1213","31/12/01") 'Data Final
    Sleep 1
    Play "{TAB}"
    WButtonClick("OK")
    Sleep 1
    IF ((StaticText("#1060") = "95.028,00") and (StaticText("#1061") = "107.724,00")) and (StaticText("#1062") = "237.304,00") then
            certo1 = 1
    else
        LogErros(Cabecalho,"O SALDO DO ATIVO NÃO CONFERE, algum lançamento não foi efetuado com  sucesso.", Cabecalho)
    end if
    'VERIFICA SALDO DO PASSIVO
    WEditSetText("#1211","939") 'Conta
    sleep 1
    Play "{TAB}"
    WEditSetText("#1212","01/01/99") 'Data Inicial
    sleep 1
    Play "{TAB}"
    WEditSetText("#1213","31/12/01") 'Data Final
    Sleep 1
    Play "{TAB}"
    WButtonClick("OK")
    Sleep 1
    IF ((StaticText("#1060") = "8.400,00") and (StaticText("#1061") = "340,00")) and (StaticText("#1062") = "241.940,00") then
            certo2 = 1
    else
        LogErros(Cabecalho,"O SALDO DO PASSIVO NÃO CONFERE, algum lançamento não foi efetuado com sucesso.",Cabecalho)
    end if
    Sleep 1
    tudocerto = (certo1 + certo2)
    IF tudocerto = 2 then
       LogErros(cabecalho,"Não foram encontrados erros",Cabecalho)
       MSGBOX "TUDO CERTO, os LANÇAMENTOS foram efetivados com Sucesso!!!",MB_ICONINFORMATION,"Teste de Lçtos
Normais"
    else
      VisualizaArquivo()
    End IF
    WButtonClick("Cancelar")
    WMenuSelect("&Arquivo\&Sair")
    Maximiza()
End Scenario
```

O capítulo 2.5 mostrou que as atividades de teste de software estão divididas em dois grandes grupos: preparação e realização dos testes. Para etapa de preparação dos testes para este trabalho de conclusão de curso, foi utilizado o material desenvolvido pela WK WK Sistemas na qual não pode ser retirado da empresa. Por este motivo não está exposto neste trabalho.

No capítulo 2.5.1 foi mostrado como devem ser tratados o planejamento dos testes. Este planejamento está dividido em 3 tarefas separadas: planos de testes, planejamento inicial e identificação dos itens a testar. Para este trabalho de conclusão de curso foi utilizado o seguinte plano de teste demonstrado no Quadro 11.

Quadro 11 – PLANO DE TESTES

Item	**Descrição**
Identificar o plano de testes	Teste do Radar Contábil
Introdução	Implementação de testes automatizado afim de encontrar erros ainda não descobertos.
Itens a testar	Implantação da Base de dados. Cadastros de: Usuários, Estado e Municípios, Propriedades, Empresas/Filiais, Plano Empresarial, Relacionamentos, Históricos, Mensagens, ECFs, Documentos, Naturezas de Operação, Vendedores, Tipos de Cobrança, Tipos de Vencimento, Forma de Pagamento, Produtos, Serviços, Tipos de Período, Condições de Pagamento e Percentual de Rateio. Movimentações: Saldo Anterior, Lançamentos Normais e Lançamentos em Lote.
Aspectos a testar	Gravação e conferência dos registros.
Aspectos que não serão testados	Utilização do sistema em modo multi-usuário.
Abordagem	Não definido.
Critérios de completeza e sucesso	Verificar se o dado inserido na gravação, é realmente o que foi gravado. Verificar valores contábeis conforme especificação da WK WK Sistemas. (Esta especificação não é liberada pela empresa)
Critério de suspensão e retomada	Se a base de dados for danificada em qualquer ponto do teste, o mesmo deve ser reiniciado com uma base vazia.
Resultado do teste	Os resultados do teste devem ser registrados em um arquivo texto pela própria automatização do teste.
Tarefas de teste	Não definido.
Ambiente	Windows 98.
Responsabilidades	Não definido, pois para este trabalho não utilizou-se uma equipe de teste.
Agenda	Não definido por se tratar de um trabalho de conclusão de curso.
Riscos e contingências	Não definido.
Aprovação	Nome assinatura dos responsáveis pela criação do plano não é possível por se tratar de um trabalho de conclusão de curso.

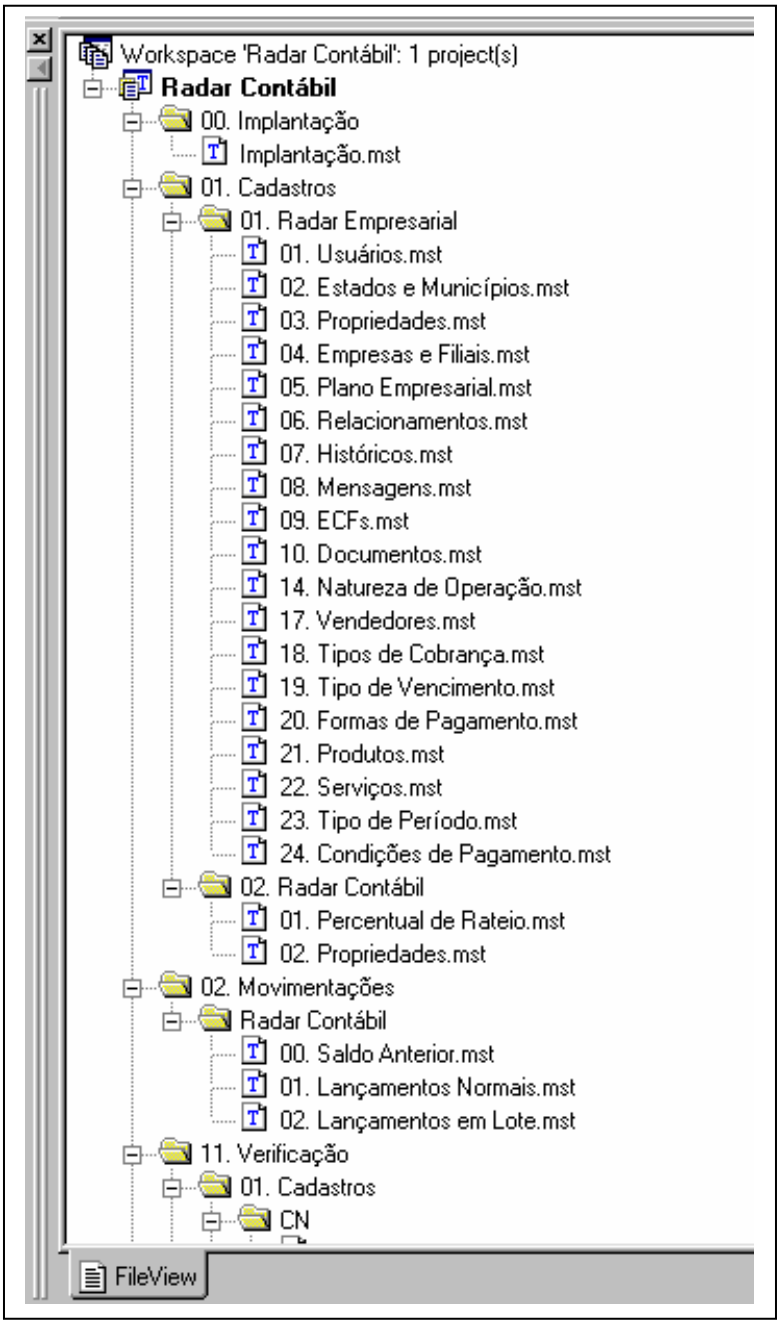
O planejamento inicial dos testes não foi executado pois seria necessário ter acesso aos documentos sobre a especificação do software na qual o acesso é restrito ao uso interno da empresa WK WK Sistemas. A identificação do itens a testar não foi especificada pois foi utilizado o item “itens a testar” do Quadro 11 visto anteriormente.

O capítulo 2.5.2 apresentou como deve ser construído um desenho de caso de teste. Conforme exposto neste mesmo capítulo, seriam necessários documentos como a especificação dos requisitos do software submetido ao teste entre outros insumos para a execução deste item. Por se tratar de um trabalho de conclusão de curso, o acesso a esses documentos não foi possível, eles são de uso restrito da empresa e por este motivo este item não foi executado neste projeto.

Conforme visto no item 2.5.2.1, que trata sobre especificações de teste, os testes de software devem seguir uma ordem lógica de execução porque geralmente cada procedimento de teste corresponde a um resultado de teste anterior, o que gera uma dependência com a execução anterior e sua respectiva base de dados gerada. Esta dependência e ordem lógica de execução pode ser percebida na implementação pela disposição dos arquivos de caso de teste que são criados prevendo-se esta dependência como pode ser visto na Figura 5.

O capítulo 2.5.2.2 traz uma denotação sobre desenho de procedimento de teste, onde seu principal insumo é descrever a seqüência de passos necessária para executar uma variação de teste. Neste projeto esta seqüência está aplicada na própria estrutura hierárquica dos arquivo de *scripts* de teste como pode ser visto na figura 5 a seguir.

Figura 5 – ESTRUTURAÇÃO LÓGICA DOS SCRIPTS DE TESTE



A dependência da base de dados gerada por uma caso de teste anterior também é implementada no protótipo e seu código fonte pode ser visto no Quadro 12. Este quadro apresenta a codificação de restauração de dados de um caso de teste anterior e o seu respectivo backup para o caso de teste seguinte.

Quadro 12 – RESTAURAÇÃO E BACKUP DA BASE DE DADOS

```
Scenario "Restaura Dados"
        if carinha = 1 or carinha = 3 then
                run "deltree /y c:\wkradar\dados\TESTE"
                run "xcopy d:\TCC\BKP\MENSG\*.* c:\wkradar\dados\TESTE /s /i"
        end if
End Scenario



Scenario "BKP dos Dados"
        if carinha = 2 or carinha = 3 then
                run "deltree /y d:\TCC\BKP\ECFS"
                run "xcopy c:\wkradar\dados\TESTE\*.* d:\TCC\BKP\ECFS /s /i"
        end if
End Scenario
```

Na seqüência, o capítulo 2.5.2.3 tratou sobre desenho de caso de teste e que pôde ser visto também com mais detalhes no capítulo 2.7. Estes desenhos são um conjunto específico de dados de teste juntamente com os resultados esperados. Estas informações devem ser descritas pelo gerente do produto submetido ao teste. Para este trabalho de conclusão de curso o desenho de caso de teste não foi realizado pela falta de informações e documentos necessários para a completeza deste item, pois os materiais descritos pelo gerente de produto do software submetido ao teste, pertencem a empresa WK WK Sistemas não podem ser retirados da mesma, por conseqüência disto, o capítulo 2.5.2.4 que trata sobre a revisão do desenho de teste não pôde ser realizada.

O próximo capítulo estudado foi o 2.5.3, implementação dos testes, que é a etapa onde o ambiente de teste é preparado. Para aplicação deste item neste trabalho, a preparação do ambiente foi realizada instalando-se o software submetido ao teste, o Radar Contábil, e a ferramenta de automatização teste, o *Visual Test*. Dando seqüência a esta etapa, tem-se a execução dos testes, vista no item 2.5.4. Esta etapa é cumprida quando a execução propriamente dita dos scripts é realizada e os relatórios referentes aos resultados dos testes são gerados.

A verificação do término dos testes descrita no capítulo 2.5.5 é realizada através de observação direta da execução dos testes automatizados verificando-se, se o seu término efetivou-se de uma maneira normal.

O balanço final exposto pelo capítulo 2.5.6, não foi realizado. Pois, a realização deste item está voltada para testes que possam ter continuidade em outros softwares. Este trabalho não se enquadra a este item pelo fato de submeter um único sistema aos testes.

O item 2.6.2.1 fala sobre testes funcionais. Este tipo de teste está subdividido em análise do valor limite, testes de comparação e testes de tempo real. O protótipo de automatização de teste implementado neste trabalho, atende em parte o teste funcional. A análise de valor limite está implementada no protótipo e pode ser vista através do Quadro 12. O teste de comparação proposto, não foi implementado explicitamente mas se tomarmos duas versões do Radar Contábil, software submetido ao teste, e submeter estas duas versões à execução dos testes implementados no protótipo deste trabalho comparando os resultados das duas versões, teremos o teste de comparação efetuado. Já a automatização dos testes de tempo real não é viável por serem testes minuciosos e variados. Além disso, o software em questão submetido ao teste, não é um tipo de sistema que possa ser submetido a este tipo de teste.

Quadro 13 – ANÁLISE DO VALOR LIMITE

```
Scenario "Cadastro Portadores"
        sleep .5
        WmenuSelect("&Cadastros\Port&adores...")
        sleep .5
        WEditSetText("#1001","005")
        WEditSetFocus("#1003")
        QueSetSpeed(100)
        Play "Banco Internacionalimmm" ' 20 caracteres
        WEditSetFocus("#1004")
        Play "Banco Internacional S/A Pan-Americano do Desenvolvimento" ' 50 caracteres
        WButtonClick("OK")
        Sleep 2
        LogErros("","================================================================================","3")
        LogErros("","================================================================================","3")
        Cabecalho = ("                    T E S T E S   F U N C I O N A I S")
        LogErros(Cabecalho,"","1")

        Cabecalho = ("ANÁLISE DO VALOR LIMITE")
        LogErros(Cabecalho,"","2")


        WMenuSelect ("&Editar\&Incluir")
        WEditSetText("#1001","005")
        play "{TAB}"
        if EditText("#1003") = "Banco Internacionali" then
                resultado% = 1
        else
                LogErros(Cabecalho,"O Campo FANTASIA não foi gravado corretamente",Cabecalho)
        end if

        if EditText("#1004") = "Banco Internacional S/A Pan-Americano do Desenvolv" then
                resultado% = resultado% + 1
        else
                LogErros(Cabecalho,"O Campo DENOMINAÇÃO SOCIAL não foi gravado corretamente",Cabecalho)
        end if

        if resultado% = 2 then
                LogErros(cabecalho,"Não foram encontrados erros",Cabecalho)
                MsgBox "Todos os dados foram gravados corretamente!",MB_ICONINFORMATION,"Verificação"
        Else
                VisualizaArquivo()
        End If
        SLEEP 1
        WButtonClick("Cancelar")
        WMenuSelect("&Arquivo\&Sair")
End Scenario
```

O item 2.6.2.2 trata sobre testes não funcionais, esses testes cobrem por exemplo, desempenho, dados persistentes e outros atributos de qualidade. Dois desses itens, desempenho e dados persistentes, estão implementados no protótipo. Quanto ao desempenho, foi implementado um teste na qual é feito um *looping* definido pelo programador de teste, com os lançamentos contábeis. Parte de um exemplo de *script* para esse tipo de teste pode ser visto no Quadro 13.

Quadro 14 – TESTE DE DESEMPENHO

```
ano$="01"
y = 3
m = 3
TestaX = 0
TestaI = 0

for x = 1 to y
        for i=1 to m

                If i=1 then mes$="01"
                If i=2 then mes$="02"
                If i=3 then mes$="03"
                If i=4 then mes$="04"
                If i=5 then mes$="05"
                If i=6 then mes$="06"
                If i=7 then mes$="07"
                If i=8 then mes$="08"
                If i=9 then mes$="09"
                If i=10 then mes$="10"
                If i=11 then mes$="11"
                If i=12 then mes$="12"


                dias$ = GeraDia()
                WEditSetText("#1002","1") ' FILIAL
                Sleep ENTRE
                Play"{TAB}"
                WEditSetFocus("#1003") 'DATA
                Play (dias$) 'DIA
                Play (mes$) 'MES
                Play (ano$) 'ANO
                Sleep ENTRE
                Play"{TAB}"
                WEditSetText("#1018","46")      'CTA DÉBITO
                Sleep ENTRE
                Play"{TAB}"
                WEditSetText("#1019","81") 'CTA CRÉDITO
                Sleep ENTRE
                Play"{TAB}"
                WEditSetText("#1080","1000")  'VALOR
                Sleep ENTRE
                Play"{TAB}"
                WEditSetText("#1033","07.04") 'HISTÓRICO
                Play "{TAB}"
                Sleep ENTRE
                WButtonClick("OK")
                Sleep INCLUI
                TelaGerencial() 'VERIFICA SE TEM GERENCIAL
```

Ainda continuando com os testes não funcionais tem-se o teste de restrições de integridade através de dados persistentes. Este teste também é implementado pelo protótipo e pode ser visto no Quadro 14. Esta implementação dá-se através da tentativa de incluir um registro em um determinado cadastro com sua chave duplicada, o resultado é registrado no relatório de resultados do teste.

Quadro 15 – RESTRIÇÕES DE INTEGRIDADE

```
Scenario "Cadastro de Tipos de Cobrança"

	WmenuSelect("&Cadastros\Tipos de Co&brança...")
	sleep .5

	'RESTRIÇÕES DE INTEGRIDADE
	'CADASTRO DE TIPOS DE COBRANÇA
	WEditSetText("#1003", "Teste Teste Teste") 'Descrição
	WEditSetText("#1001", "1") 'Código
	sleep 1
	WButtonClick("OK")
	Sleep 2


	Cabecalho = ("RESTRIÇÕES DE INTEGRIDADE")
	LogErros(Cabecalho,"","2")

	WmenuSelect("&Editar\&Incluir")
	WEditSetText("#1001","1") 'Código
	play "{TAB}"
	Cabecalho = ("CADASTRO DE TIPOS DE COBRANÇA")
	LogErros(Cabecalho,"","2")

	if EditText("#1003") = "Cobrança Simples" then 'Descrição
		LogErros(cabecalho,"Não foram encontrados erros",Cabecalho)
		MsgBox "Todos os dados foram gravados corretamente!",MB_ICONINFORMATION,"Verificação"
	else
		LogErros(Cabecalho,"O Campo DESCRIÇÃO não foi gravado corretamente",Cabecalho)
		VisualizaArquivo()
	end if

	Sleep 1
	WButtonClick("Cancelar")
	Sleep .5
	WMenuSelect("&Arquivo\&Sair")
End Scenario
```

No item 2.5.4, execução dos testes, é sugerido por Ramirez (2001) que os testes devem resultar em um relatório que contenha os problemas encontrados. O protótipo de automatização de teste implementado neste trabalho, possui uma função que pode ser vista no Quadro 13, que implementa a criação de um relatório com o resultado completo dos testes conforme proposto pela IEEE segundo Ramirez (2001). Este relatório é gerado automaticamente pelo protótipo com a descrição de cada erro encontrado em um arquivo texto gerado em um diretório especificado previamente pelo programador de teste.

Quadro 16 – **FUNÇÃO DE GERAÇÃO DO RELATÓRIO**

```
Function LogErros(Msg1$,Msg2$,Verifica$) as String
DIM ARQUIVO%
DIM Nome, Cabecalho as String
DIM Erro as String
ARQUIVO% = FREEFILE
Cabecalho = Msg1
Erro = Msg2

if EXISTS ("D:\TCC\Resultado.TXT") then
        OPEN "D:\TCC\Resultado.TXT" FOR APPEND AS #ARQUIVO

        if verifica = "1" then 'quando tem que escrever o M O D U L O
                PRINT #ARQUIVO, Cabecalho
        end if

        if verifica = "2" then 'quando tem que escrever um título
                PRINT #ARQUIVO , ""
                PRINT #ARQUIVO , "================================================================="
                PRINT #ARQUIVO, Cabecalho
                PRINT #ARQUIVO , "================================================================="
        end if

        if verifica = "3" then 'quando tem que escrever um título ou sub-título
                PRINT #ARQUIVO , ""
                PRINT #ARQUIVO, "    ",Cabecalho
        end if

        if Cabecalho = Verifica then 'quando tem que escrever somente o erro
                PRINT #ARQUIVO, "          ",erro
        end if

else
        OPEN "D:\TCC\Resultado.TXT" FOR OUTPUT AS #ARQUIVO
        print "nao existe arquivo"
        PRINT #ARQUIVO , "================================================================="
        PRINT #ARQUIVO , "ARQUIVO DE ERROS DO RADAR   -   GERADO AUTOMATICAMENTE PELO VISUAL TEST"
        PRINT #ARQUIVO , "BY MARCIO TOMELIN                    " + DATE$ + " " + TIME$
        PRINT #ARQUIVO , "================================================================="
        PRINT #ARQUIVO, Cabecalho
        PRINT #ARQUIVO, ""
        Verifica = Cabecalho
end if
Close ARQUIVO
End Function
```

O item 2.8 trata sobre a norma brasileira ABNT NBR 12207 e suas sugestões e recomendações quanto aos testes de software. Dos três capítulos abordados pela norma, processo de desenvolvimento, processo de operação e processo de apoio ao ciclo de vida, apenas os dois últimos estão implementados no protótipo. O processo de desenvolvimento,

trata de uma forma teórica como deve ser estruturado o processo de desenvolvimento de software sendo utilizada apenas como uma fase introdutória às demais. Por este motivo , não é viável e nem possível a sua implementação. Quanto ao processo de operação, como já foi visto anteriormente este é efetuado pelo operador e é a execução dos testes propriamente ditos, ou seja, executando todos os *scripts* de atende-se a este item da norma ABNT NBR 12207.

O processo de apoio de ciclo de vida tem como enfoque ao teste de software o teste de validação, este serve para determinar se os requisitos do sistema submetido ao teste atendem à sua especificação. Este teste também está implicitamente implementado no protótipo de automatização de teste, pois se, após executado por completo, não forem encontrados erros, significa que o sistema atende aos seus requisitos especificados. Deve-se levar em consideração que o teste automatizado deve estar implementado corretamente de forma com que os valores e dados utilizados no teste possa validar corretamente o produto.

Enfim, todas as técnicas e normas estudadas neste trabalho que foram possíveis de ser implementadas, estão inseridas no protótipo, algumas de forma implícita e outras explicitamente conforme proposto pelas normas avaliadas no presente trabalho.

3.2.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para a referida implementação da automatização dos testes, foi utilizada uma ferramenta específica para este tipo de atividade, que é o *Rational Visual Test 6.5*. Esta ferramenta que já foi vista no capítulo 2.10, funciona basicamente como uma máquina de execução de *Scripts* de casos de teste que são escritos e ou gerados nela própria.

3.2.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

O processo de automatização de teste foi implementado para ser aplicado ao sistema de contabilidade Radar Contábil da WK Sistemas. Este tem como objetivo detectar erros no referido software.

Para executar este processo automatizado de teste, basta abrir o projeto na ferramenta de teste *Visual Test*, selecionar o caso de teste desejado para a execução, certificar-se de que o sistema em questão a ser testado esteja instalado na máquina e executá-lo,

pressionado a tecla F5 ou o ícone próprio para esta tarefa. Ou então, tem-se ainda a opção de executar o programa de teste gerado em formato executável correspondente ao referido caso de teste escolhido.

Uma vez executado o teste, aguarda-se o fim do mesmo e verifica-se o arquivo de resultados gerado pelo mesmo, este conterá a descrição dos erros se eles existirem.

3.3 RESULTADOS E DISCUSSÃO

Com o desenvolvimento deste trabalho, pode-se chegar ao seu objetivo principal, que era o de elaborar uma solução automatizada para testes de software de acordo com o que é proposto pelas normas e técnicas especificadas neste trabalho. Além disso, através de uma observação direta foi analisada a atividade de teste de software empregada pela WK Sistemas identificado quais dos itens estudados nas propostas de testes sugeridas pelas normas e técnicas especificadas neste projeto, se enquadram a atividade de teste de software da WK Sistemas, mostrando de que maneira a empresa está se enquadrando às normas e o que deve fazer para melhorar ainda mais a sua atividade de teste, e pode ser visto a seguir.

Ao longo de sua existência, a WK Sistemas vem realizando a atividade de teste de software de acordo com sua própria experiência adquirida no seu dia-a-dia de prática de desenvolvimento. Nos últimos anos, esta atividade tem ganhado mais força com a implantação de automatização dos testes, porém, até o presente momento não havia sido utilizada nenhuma norma como modelo para esta implantação.

Com o estudo e análise das recomendações propostas nas normas , verificou-se que, mesmo sem seguir algum modelo, a WK Sistemas executa suas atividades de teste de uma maneira que se encaixa em parte em alguns dos itens das normas citadas neste trabalho. Estes itens que hoje estão sendo utilizados pela WK Sistemas podem ser vistos a seguir.

Os testes de regressão são executados constantemente pela equipe de testes da WK Sistemas e de uma forma automatizada, conforme sugerido pela norma.

Tem-se também os teste de aceitação que estão divididos em funcionais e não funcionais e que são executados regularmente de forma manual e também automatizada. O mesmo acontece com os testes de integração são feitos de uma maneira na qual se enquadram

no que é sugerido pela norma. Porém os testes de unidade, que conforme a norma sugere, deve ser realizado na fase de desenvolvimento pelo próprio programador, testando as funções, classes, procedimentos internos do sistema, ou seja, o código fonte, não é praticado pela empresa conforme a norma sugere. É importante salientar que a empresa possui uma equipe de testes capacitada para esta atividade. Esta equipe faz uso de ferramentas que auxiliam na produção dos testes de uma forma eficiente e rápida. Os resultados dos testes são analisados e encaminhados à equipe de desenvolvimento quando necessário.

A utilização de testes automatizados pela empresa, mostra no Quadro 16, que a velocidade dos testes manuais em relação à automatização de um mesmo item de teste, aumenta espantosamente. Estes itens de testes estão relacionados ao teste do Radar Contábil, software submetido ao teste utilizado ao longo deste projeto.

Quadro 17 – Velocidade de execução de Testes Manuais x Testes Automatizados

Item submetido ao teste	Tempo em minutos do teste manual	Tempo em minutos do teste automatizado
Cadastro de um plano de contas completo	480	3
Cadastro de empresas filiais	90	0.7
Movimentações contábeis referentes a um ano de exercício.	260	2

Enfim, a empresa, mesmo sem ter tomado como referência as propostas destas normas, está praticando o sugerido pelas mesmas, mas com uma certa limitação. Esta limitação caracteriza-se pelo fato de que apenas alguns tipos e técnicas de testes de software sugeridos pelas normas são utilizados pela WK, porém outros que não são utilizados mas considerados importantes para o autor, não estão sendo empregados. Como é o caso dos desenhos de casos de teste e testes de caixa preta na qual introduzem os testes de unidade, que são os testes na estrutura internas dos sistemas. Se este teste fosse adotados, certos erros básicos de codificação (por exemplo, erros de análise de valor limite em relação ao tamanho e/ou tipo de dados de entrada) poderiam ser detectados já na fase de desenvolvimento pelos próprios programadores do sistema.

Para aprimorar ainda mais sua atividade de teste, o autor sugere que a WK Sistemas comece a criar seus casos de teste para cada um de seus sistemas, bem como executar os teste de caixa preta. O autor acredita que, com a implementação destes dois itens,

pode-se dizer que a WK Sistemas torna-se uma empresa que possui um processo de testes de seus softwares que atende às sugestões das normas, estudadas neste trabalho.

Estas recomendações estendem-se a todas as empresas de desenvolvimento de software, no sentido de melhorarem a qualidade de seu processo de desenvolvimento, e, conseqüentemente, de seus produtos.

4. CONCLUSÕES

O presente trabalho apresenta os resultados da elaboração de um estudo de algumas normas de qualidade e as atividades de teste de software empregadas pela WK Sistemas, empresa usada como objeto de estudo deste trabalho. Além disso, este trabalho sugere melhorias e aperfeiçoamentos no processo de teste de software nesta empresa, que podem ser estendidos a outras empresas com atividade de desenvolvimento de software. O trabalho também implementa um protótipo para a automatização dos testes que contém diversos tipos de testes sugeridos pelas normas estudadas. Os tipos de testes implementados pelo protótipo são: testes de aceitação incluindo os funcionais e não funcionais, testes de regressão, aceitação e integração, além de a própria estruturação dos arquivos de teste(*Scripts*) estarem dispostos de forma organizada seguindo a ordem lógica de execução conforme a norma recomenda. Contudo, o protótipo permite dar qualidade, rapidez e eficiência aos testes de software aumentando cerca de 140 vezes a velocidade de execução de um teste manual para um automatizado. Este número diz respeito ao teste automatizado criado para este trabalho de conclusão de curso destinado ao Radar Contábil que foi o software submetido ao teste.

O protótipo implementado no decorrer deste trabalho de pesquisa realiza de maneira automatizada diversos tipos de teste, tais como testes de regressão, testes funcionais e testes não funcionais. Além de também gerar relatórios com a descrição de erros encontrados no sistema testado. Tais relatórios são de extrema utilidade para a posterior avaliação dos problemas existentes no produto testado. Portanto, conclui-se que a criação do protótipo foi bem sucedida.

Além disso, o teste de software criado para o Radar Contábil neste trabalho, baseou-se nas normas de qualidade estudadas no decorrer capítulo 2, e pode ser visto pelo capítulo 3.2 de que maneira estas técnicas e normas de teste de software foram aplicadas. Contudo, concluiu-se que, nem todas as metodologias, técnicas e/ou normas de qualidade aqui estudadas, puderam ser aplicadas neste trabalho. Pois algumas delas dependem de documentos entre outros insumos que não foram disponibilizados.

Outra observação conclusiva refere-se ao fato de que a análise das normas estudadas indica que mesmo sendo impossível testar um sistema por inteiro, a qualidade final

dos softwares testados tende a melhorar significativamente se as suas sugestões forem seguidas.

Um dos grandes motivos que levaram ao desenvolvimento desta pesquisa foi a percepção por parte do autor em relação à grande deficiência no processo de desenvolvimento de software hoje, evidenciada pela falta de uma estratégia bem definida para a realização de testes, devido principalmente ao desconhecimento das normas e técnicas utilizadas para este tipo de atividade.

4.1 LIMITAÇÕES

O protótipo atende apenas ao teste do sistema para qual ele foi implementado, se for desejado testar algum outro sistema, deve-se implementar novos scripts de teste.

A ferramenta utilizada para a geração dos *Scripts*, é voltada para sistemas desenvolvidos em *Visual C++*, mas isso não significa que seja limitado somente a esta linguagem, porém funciona com mais eficiência nela.

4.2 EXTENSÕES

Como extensões deste trabalho, sugere-se, a realização de avaliações de outras ferramentas para automatização de testes de software, pois é possível que determinadas ferramentas sejam mais adequadas a determinados tipos de testes ou determinados tipos de sistemas, e isto foge ao escopo do presente trabalho. Como caso típico, pode-se imaginar ferramentas que sejam mais adequadas ao teste de softwares não convencionais tais como sistemas que realizem processamento vetorial ou softwares que implementem algoritmos paralelos.

Além disso, uma possível extensão seria a aplicação dos testes propostos neste trabalho em empresas que não desenvolvem atividades de teste de uma forma organizada e metódica. Assim, é possível fazer um levantamento do grau de melhoria de qualidade que os testes de software, nos padrões das recomendações da ABNT, podem trazer ao processo de desenvolvimento de software dessas empresas e, principalmente, a qualidade de seus produtos finais.

REFERÊNCIAS BIBLIOGRÁFICAS

ARNOLD , Thomas R. **Visual test 6 bible.** Chicago: IDG Books World Wide, 1999.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBRISO/IEC12207:** tecnologia de informação – processos de ciclo de vida de software. Rio de Janeiro, 1998.

GIMENES, I. M. S. **Ferramentas CASE.** Maringá: Makrow, 1994.

HETZEL, W., **Guia completo ao teste de software.** Rio de Janeiro: Campus, 1987.

IEEE. IEEE Standard Collection. Software Engineering. IEEE, New York, 1994.

MARTIMIANO, L. A. F., **Integração da ferramenta de teste de software Poke-Tool em PCTE**. 1995. Trabalho de Graduação, DIN/UEM, Maringá.

OLIVEIRA, Flávio Moreira de. **Teste de software**, Porto Alegre, dez. [1997]. Disponível em: <http://www.inf.pucrs.br/~flavio/teste/>. Acesso em: 27 abr. 2001.

POSTON, R.M. **Automating specification-based software testing. IEEE computer society.** Chicago: Books, 1996.

RAMIREZ, Jaime Arturo. **Teste de software,** Belo Horizonte, dez. [1990]. Disponível em: <http://ead1.eee.ufmg.br/~renato/engsoft/Teste_Soft.pdf>. Acesso em: 13 fev. 2001.

ROBERT, MaryAnn; MARYANSKI, Fred J. **Automated test plan generator for database application systems.** California: Transctions of ACM, 1991.

SOUZA, Simoni do Rocio Senger. **Introdução ao teste de software**, Paraná, out. 2000. Disponível em: <www.pbnet.com.br/openline/cefet/horario_sbes_englihs.htm>. Acesso em: 18 abr. 2001.