UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

SISTEMA PARA GERENCIAMENTO DE TESTES FUNCIONAIS DE SOFTWARE

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

EVERTON LUIZ KOLM

BLUMENAU, NOVEMBRO/2001

2001/2-21

SISTEMA PARA GERENCIAMENTO DE TESTES FUNCIONAIS DE SOFTWARE

EVERTON LUIZ KOLM

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

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

Prof. Everaldo Artur Grahl — Supervisor na FURB

José Milton da Silva — Orientador na Empresa

Prof. José Roque Voltolini da Silva — Coordenador na FURB do Estágio Supervisionado

BANCA EXAMINADORA

Prof. Everaldo Artur Grahl

Prof. Marcel Hugo

Prof. Paulo César Rodacki Gomes

DEDICATÓRIA

Dedico este trabalho à minha família e à minha namorada, por compreenderem meus sonhos e me apoiarem nos momentos mais difíceis. Amo todos vocês.

AGRADECIMENTOS

A Deus por ter me guiado, me dando força e saúde para cursar esta faculdade.

Aos meus pais Siegfried e Marli Martins Kolm, pela paciência e compreensão que tiveram durante toda a minha vida acadêmica.

Ao meu irmão, Cleison José por ter aguardado a conclusão do meu curso de graduação antes de iniciar o seu.

A minha namorada Anacléia Fernanda Moretto, pela compreensão que teve durante todo meu curso de graduação.

Ao meu opa Albano Kolm, ao meu avô Alzidio José Martins e ao meu tio Fernando José Klock, que infelizmente não estão mais conosco, mas que com certeza estiveram olhando por mim.

Ao professor e orientador Everaldo Artur Grahl, pela atenção e auxílio dispensados na elaboração deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

DER	Diagrama Entidade Relacionamento
ERP	*Enterprise Resource Planning*
SMS	Solicitação de Manutenção de Sistema
VV&T	Verificação, validação e teste

RESUMO

Este trabalho teve como objetivo o desenvolvimento de um sistema para o gerenciamento de testes funcionais de software, com a utilização de *checklists* que facilita o processo de teste na empresa Benner Sistemas S/A. A utilização do sistema mostrou uma redução aproximadamente de cinqüenta por cento, baseado no número de erros encontrados pelos clientes nos sistemas.

ABSTRACT

This work had like object the development of system for the managemente of funcionals teste of software, with the utilization of checklists that facilitate the process of teste in the enterprise Benner Systems S/A. The utilization of the system showed a reduction approximately of fifteen percent, based in the number of errors found for the customers in the systems.

1 INTRODUÇÃO

Com a grande concorrência que há hoje na área de software, as grandes empresas de desenvolvimento estão precisando ter cada vez mais um diferencial em seu produto e um destes é o Teste de Software. A empresa Benner Sistemas S/A de Blumenau estava necessitando de um software para o gerenciamento do seu teste de software e um padrão para o seu processo de Teste de Software, pois não possuia nenhuma técnica e nenhum padrão para esses testes. Os testes hoje são realizados pelos próprios programadores e pelo setor de suporte, que verifica as funções básicas do sistema e rotinas novas ou alteradas durante uma versão. A empresa verificou que hoje 30% (trinta por cento) dos problemas identificados pelos clientes nos sistemas de Recursos Humanos e Corporativo, são problemas que não foram encontrados por falta de teste. A falta de teste está gerando uma grande quantidade de retrabalho, o que acaba ocasionando um custo maior e além de prejudicar a imagem da empresa frente ao cliente.

De acordo com Presmann (1995), as mudanças são inevitáveis quando um software de computador é construído. As mudanças aumentam o nível de confusão entre os engenheiros de software que estão trabalhando num projeto. A confusão surge quando as mudanças não são analisadas antes de serem feitas, registradas antes de serem implementadas, relatadas aos que precisam tomar conhecimento delas ou controladas de um modo que melhore a qualidade e reduza o erro.

Segundo Maldonado (1998), o teste funcional concentra-se nos requisitos funcionais do software. Através dele torna-se possível verificar as entradas e saídas de cada unidade. O teste funcional também é conhecido como teste caixa preta pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída. Na técnica de teste funcional são verificadas as funções do sistema sem se preocupar com os detalhes da implementação. O teste funcional envolve dois passos principais: identificar as funções que o software deve realizar e criar casos de teste capazes de checar se essas funções estão sendo realizadas pelo software. As funções que o software deve possuir são identificadas a partir de sua especificação. Assim, uma especificação bem elaborada e de acordo com os requisitos do usuário é essencial para esse tipo de teste.

Segundo Paula Filho (2001), método de Caixa Preta tem por objetivo determinar se os requisitos foram totais ou parcialmente satisfeitos pelo produto. Os testes de caixa preta não verificam como ocorre o processamento, mas apenas os resultados produzidos.

De acordo com Inthurn (2001), o teste funcional ou caixa preta procura descobrir basicamente:

a) funções incorretas ou ausentes;
b) erros de interface
c) erros nas estruturas de dados ou no acesso a bancos de dados externos;
d) erros de desempenho;
e) erros de inicialização e término.

1.1 EMPRESA

O estágio foi realizado na empresa Benner Sistemas S/A, cuja principal finalidade é produzir e desenvolver sistemas para aplicações nas áreas de Recursos Humanos e Corporativo, utilizando as ferramentas de Benner Runner, Benner Builder, Benner Integrator e Benner Report.

A empresa atualmente desenvolve sistemas para empresas de pequeno, médio e grande porte e possui um variado número de sistemas para controle de diversas áreas, tais como:

a) Sistema de Recursos Humanos;
b) Sistema Corporativo;
c) Sistema de Turismo e outros.

A Benner Sistemas possui dez analistas e trinta e cinto programadores que estão distribuídos nas áreas de desenvolvimento dos sistemas de recursos humanos, corporativo e turismo. A empresa não possui metodologia de desenvolvimento de sistemas, nem padrão de desenvolvimento. O analista define a proposta de solução, que seria a definição, em um editor de textos e utiliza a ferramenta Benner Builder para criar as tabelas e campos na base de dados ou passa para o programador fazer. Desta forma torna-se muito mais pratico a criação das tabelas e campos diretamente na base de dados, pois ele pode verificar mais rapidamente como ficará o *layout* dos campos na tela do sistema.

1.2 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho foi desenvolver um software para auxiliar o gerenciamento de testes funcionais de software na empresa Benner Sistemas S/A.

Os objetivos específicos do trabalho são:

a) criar um processo de testes e incluir *checklists* para fazer as verificações de erros;
b) avaliar o impacto da utilização de um processo formalizado na empresa.

1.3 ESTRUTURA DO TRABALHO

O primeiro capítulo apresenta uma introdução ao trabalho, iniciando alguns conceitos empregados em sua elaboração. São apresentados ainda, os objetivos, a empresa e a organização do texto.

O segundo capítulo compila a fundamentação teórica aplicada no desenvolvimento do trabalho. Apresenta informações sobre teste de software, tipos de teste e gerenciamento de testes. O capítulo reúne ainda os conceitos sobre qualidade de software.

O terceiro capítulo apresenta a especificação do protótipo, onde consta o diagrama de contexto, diagrama de casos de uso, diagrama de entidade e relacionamento (DER), dicionário de dados e o diagrama hierárquico funcional. Também no terceiro capítulo é apresentado o sistema de gerenciamento de testes funcionais de software.

No quarto capítulo são feitas as considerações finais sobre o trabalho, incluindo conclusões e sugestões.

2 TESTE DE SOFTWARE

Neste capítulo serão apresentadas considerações sobre teste de software, tipos de teste e gerenciamento de testes.

2.1 CONCEITOS INICIAIS

De acordo com Martin (1991), o teste em software nada mais é que a verificação dinâmica do software, ou seja, é o processo de executar e observar seu comportamento em relação aos requisitos acordados em contrato. É através do teste que se pode detectar a presença de erros em um programa, embora não se possa predizer a ausência deles. Isto equivale dizer que quando o processo de teste não detecta erro, a única afirmação possível é a de que o teste pode não ter sido suficientemente completo para revelar os erros existentes e jamais a de que o programa em questão está correto.

Segundo UNICRUZ (2001), Teste é uma área da Engenharia de Software na qual a distância entre a pesquisa de conhecimentos e a prática atual é muito grande. Uma das razões muitas vezes citadas é a quantidade de esforço extra necessário para o desenvolvimento da infra-estrutura necessária para um processo de teste compreensivo, tornando-o tedioso, pois requer a determinação dos dados para teste e dos resultados esperados, a execução do programa com os dados selecionados e a análise dos resultados computados.

De acordo com Inthurn (2001), freqüentemente gasta-se muito tempo e dinheiro em testes e na correção dos erros encontrados, devido especialmente ao fato de vários erros somente virem a ser detectados no final do processo de desenvolvimento. Além disso, sem uma infra-estrutura (facilidades automatizadas ou não) para a realização dos testes, torna-se, por vezes, impraticável a sua aplicação de forma adequada. Todavia, se a atividade de teste for executada como parte integrante do desenvolvimento de software, casos de teste podem ser criados nas diferentes fases do ciclo de vida para testar os produtos da própria fase e para serem usados na implementação do código.

A partir da especificação estruturada do sistema (gerada na fase de análise), deve-se começar com a atividade de geração de casos de teste de aceite. Após a codificação, cada módulo será testado individualmente, bem como sua integração com o sistema.

Primeiramente é gerado o plano de teste onde se estabelece a pessoa/grupo responsável por testar o sistema e se define procedimentos padrões, verificando-se os possíveis erros e comparando os resultados obtidos com os resultados esperados.

Nesta fase são efetuados também testes de desempenho, a fim de analisarem o tempo de resposta do sistema, testes de vias normais, que verificam se o sistema executa apropriadamente após uma entrada válida e finalmente os testes de vias de erro, onde as entradas fornecidas ao sistema são propositadamente valores não usuais ou errados. Ao final dos testes efetua-se um relatório com os resultados obtidos.

Um dos principais propósitos para a realização de testes pode ser considerado como sendo a busca por reduzir o risco envolvido na construção e no uso de software com erros. Apesar de não se conseguir eliminar todo o risco de desenvolver produtos com defeitos, torna-se muito importante aumentar o grau de confiança de que se está construindo um produto com o comportamento desejado. Teste é uma das atividades realizadas para garantir a qualidade do software, sendo de extrema relevância, por permitir a precaução em relação aos custos envolvidos com a ocorrência de falhas e, principalmente, por salvaguardar o seu funcionamento quando envolve riscos à vida humana ou grandes perdas financeiras.

Segundo Maldonado (1997), a atividade de teste deve ser considerada como uma atividade do desenvolvimento de software. Deve ser feito todo o planejamento, depois partir para o projeto dos casos de teste, depois executar os testes, colher os resultados e então confrontar os resultados obtidos com os esperados. O teste de software nada mais é que a verificação dinâmica do software, ou seja, é o processo de executar e observar seu comportamento em relação aos requisitos acordados em contrato. Através do teste que se pode detectar a presença de erros em um programa, embora não possamos predizer a ausência deles. Isto equivale a dizer que quando o processo de teste não detecta erro, a única afirmação possível é a de que o teste pode não ter sido suficientemente completo para revelar os erros existentes e jamais a de que o programa em questão está correto.

Conforme Martin (1991), um programa é exaustivamente testado se ele é executado com todos os possíveis conjuntos de dados de entrada. A aplicação de testes exaustivos pode garantir a validade de um programa, mas para a maioria dos programas isto não é pratico porque existe um número infinito ou incrivelmente grande de possíveis conjuntos de entrada

de dados. Em vez disso, a correção de um programa é usualmente demonstrada através do teste de uma pequena amostra de exemplos cuidadosamente escolhidos.

A tarefa do responsável pelo teste é eliminar condições e falhas de programa imprevistas e descobrir qualquer implementação das especificações dos requisitos incorreta ou incompleta, usando um conjunto razoável de exemplos de testes apropriados.

Embora tenha havido muita pesquisa para se desenvolver uma teoria de testes, são poucos os resultados animadores. Até agora, o teste ainda não tem uma base teórica sólida. Até mesmo em um ambiente estruturado, o teste é orientado apenas por regras heurísticas como as que se seguem:

a) testar todos os comandos do programa e todos os caminhos pelo menos uma vez;

b) testar minuciosamente as partes do programa mais importantes e mais intensamente usadas;

c) testar os módulos individualmente antes de serem combinados. Depois, testar as intersecções dos módulos;

d) organizar o teste de modo que ele progrida dos exemplos de teste mais simples aos mais complexos. Isto significa que os testes que envolvam lógica menos complexa devem ser executados primeiro. Significa também que o processamento normal com entradas válidas deve ser testado antes de o processamento excepcional ser checado;

e) calcular os resultados do teste antes de ele ser executado.

A realização de um teste é, em sua essência, um conceito simples. Ele consiste em selecionar o que dever ser medido (um atributo ou característica do software); criar entradas controladas ou situações de teste (casos) que ponham à prova ou revelem algo sobre o atributo ou característica que será medida; simular ou executar as situações de teste e analisar os resultados, comparando-os a um padrão ou comportamento esperado. Considera-se um teste “bem-sucedido” quando os resultados observados atendem as expectativas, e “mal-sucedido” quando isto não acontece, ou quando uma deficiência é detectada.

Outra definição de teste dada por Hetzel (1987), é de que teste é processo de executar um programa ou sistema com a finalidade de encontrar erros. Esta conceituação faz de “encontrar erros” o objetivo principal, pois enfatiza que “se nossa finalidade for mostrar que um programa não tem defeitos, seremos inconscientemente levados a perseguir este objetivo; ou seja, tenderemos a escolher uma massa de dados que reduza a probabilidade de ocasionar erros. Por outro lado, se nosso objetivo for demonstrar que um programa tem defeitos, a massa de dados terá uma grande probabilidade de encontrar erros, e o sucesso do teste será maior”.

Segundo Fournier (1994), podem ser definidas duas categorias genéricas de defeitos: erros e ambigüidades. Erros são falhas no software de aplicação que farão o sistema falhar ou gerar resultados inválidos. Daí, é importante identificar o máximo possível de erros durante o ciclo de testes e eliminá-los antes de entregar o sistema a seus clientes. Para conseguir isso, os testes devem ser vistos como um processo destrutivo, que é executado com a intenção de encontrar erros.

2.2 REALIZAÇÃO DOS TESTES

Na maioria das empresas e para a maioria dos profissionais, a triste realidade é a de que a resposta para “o que testar” e “quando terminar” não resultam da aplicação de uma metodologia sistemática de teste. As abordagens aleatórias e individualizadas predominam. É comum encontrar dentro de uma mesma empresa uma gama enorme de técnicas e filosofias de teste – principalmente nas fases iniciais do processo. A experiência tem mostrado que políticas e padrões de teste de software definindo como o trabalho deve ser conduzido, o que deve ser testado, quem é responsável, e acompanhando a efetividade e o custo dos testes são coisas raras – e não existem em mais de dez por cento das empresas.

De acordo com Fournier (1994), a estratégia preliminar para testar um sistema é inicialmente desenvolvida durante a fase de análise detalhada. Os principais objetivos do processo global de teste são definidos com os usuários e devem cobrir os quatro níveis de teste - os ciclos de teste de unidade, integração de sistemas, aceitação pelo usuário e a aceitação pela produção, onde for aplicável. Outras questões relacionadas com os papéis e responsabilidade de cada grupo envolvido no processo de teste também discutidas. Por exemplo, durante essa fase, é tomada a decisão de permitir que a equipe de desenvolvimento

execute o processo de teste de integração de sistemas, em vez de atribuí-lo a uma equipe independente. Durante a fase de projeto, é desenvolvida a estratégia detalhada de teste com a criação de casos de teste efetivos. Finalmente, o ciclo formal de testes é oficialmente ativado durante a fase de implementação. Uma significativa vantagem dessa abordagem é o fato de que o processo de teste de software fica completamente integrado ao ciclo do desenvolvimento do sistema. Também o teste não é mais visto como uma atividade que ocorre somente após os programas terem sido codificados. Em vez disso, as atividades preparatórias de teste acontecem em paralelo com as atividades de análise detalhada (estratégia de teste de nível mais alto) e com as atividade de projeto (estratégia de teste detalhada e preparação dos casos de teste). Essa mudança de foco para os estágios iniciais do processo de desenvolvimento de software também permite que o gerente de desenvolvimento realce certas questões de teste muito importantes par a administração – tempo necessário para testar o sistema de forma apropriada, necessidade de usar ferramentas automatizadas de teste, custos globais dos testes do sistema e a disponibilidade de recursos de computação adequados para executar o processo de testes.

Segundo Paula Filho (2001), 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 sessenta a cem vezes maior que o custo para corrigi-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 teste sejam bem planejados e desenhados, para conseguir-se o melhor proveito possível dos recursos alocados para eles.

De acordo com Pressman (1995), a atividade de teste é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificação, projeto e codificação. A atividade de teste constitui uma anomalia interessante para o engenheiro de software. Durante as fases de definição e desenvolvimento anteriores, o engenheiro tenta construir o software, partindo de um conceito abstrato para uma implementação tangível. Agora, surge a fase de testes. O engenheiro cria uma série de casos de teste que tem a intenção de “demolir” o software que ele construiu. De fato, a atividade de teste é um passo do processo de

engenharia de software que poderia ser visto(psicologicamente, pelo menos) como destrutivo, em vez de construtivo. Os desenvolvedores de software são, por sua própria natureza, pessoas construtivas. A atividade de teste exige que o desenvolvedor descarte noções preconcebidas da “corretitude” do software que ele acabou de desenvolver e supere um conflito de interesses que ocorre quando erros são descobertos.

A maioria das empresas realiza pelo menos três tipos distintos de testes. Os programas são testados isoladamente e depois grupos de programas são testados num “teste de sistema”. Sistemas completos são, por fim, submetidos a um “teste de aceitação”. Pessoas e equipes diferentes podem encarregar-se de cada um desses níveis, sendo os testes de aceitação feitos, geralmente, pelo cliente ou usuário final. Em projetos maiores e mais complexos, outros níveis de teste podem ser utilizados.

Segundo Maldonado (1997), o processo de software possui três metas principais:

a) verificar que o software executa como especificado na documentação do projeto;
b) verificar que o software satisfaz todos os requisitos especificados na Especificação de Requisitos de Software; e
c) fornecer um status do progresso do projeto ao gerente do projeto.

Nesse estágio, várias estratégias, métodos e técnicas de teste podem ser utilizadas, nas diversas fases de teste: teste de unidade, teste de integração, teste de sistema e teste de aceitação.

Segundo Mueller (1998), teste é o processo de garantir que um programa se adapta aos requisitos da especificação e funciona em todos os casos em que se espera que funcione.

O teste consiste em:

a) selecionar um conjunto de dados de entrada com os quais se executará o programa;

b) determinar a saída que se espera ser reproduzida;

c) executar o programa;

d) analisar os resultados produzidos pela execução do programa.

Conforme Pressman (1995), a atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificação, projeto e codificação.

O objetivo é 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, de acordo com o objetivo estabelecido, ela descobrirá erros no software.

Muller (1998) estabelece uma série de regras que podem servir bem como objetivos de teste:

a) a atividade de teste é o processo de executar um programa com a intenção de descobrir erro;

b) um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;

c) um teste bem sucedido é aquele que revela um erro ainda não descoberto.

Sempre que vai se iniciar um teste, vem a pergunta onde testar? A resposta para esta pergunta depende da metodologia aplicada para a execução dos testes. Para os primeiros testes (teste de unidade e teste de integração), sugere-se que eles devem ser executados no ambiente de desenvolvimento (sendo o teste de sistema executado em ambiente alvo em condições reais de operação), no entanto nada impede que em contrato seja estabelecido que ele também seja executado no ambiente de desenvolvimento, mas sob supervisão do cliente.

Outra pergunta que sempre é feita quando se inicia um teste, é até quando testar? Corrigir todos os erros de um software é anti-produtivo, anti-realista, pois pode levar muito tempo para fazer o trabalho de correção, e o mesmo perder seu lugar no mercado. Para um produto possuir um nível de qualidade aceitável, o mesmo deve possuir taxas de erros menores que dez por cento. Pode-se utilizar para medir taxas de erros técnicas de estimativas de custos. Tem-se como exemplo a Análise por Pontos de Função (FPA), onde para um determinado número de pontos de funções estima-se um determinado número de erros,

buscando encontrar 90% dos erros estimados. Esta estimativa vai sendo aprimorada à medida que o histórico de desenvolvimento e testes de software da organização vai evoluindo.

2.3 TIPOS DE TESTE DE SOFTWARE

Um produto software pode ser testado utilizando-se algumas técnicas, sendo duas as mais usadas: quando se conhece as funções que foram especificadas para o software executar, o teste pode ser conduzido para demonstrar a operacionalidade das funções; e quando se conhece a estrutura interna do software, o teste pode ser conduzido para verificar se todos os componentes internos, quando exercitados, operam de maneira adequada. Essas duas técnicas são conhecidas, respectivamente, como Teste Caixa Preta ou Funcional e Teste Caixa Branca ou Estrutural.

Teste Caixa-Preta ou Funcional segundo Pressman (1995), os métodos deste tipo de teste concentram-se nos requisitos funcionais do software. Ou seja, esse teste possibilita que o engenheiro de software derive conjuntos de condições de entrada que exercitem completamente todos os requisitos funcionais para um programa. O teste de caixa preta não é uma alternativa para as técnicas de caixa branca. Ao contrário, trata-se de uma abordagem complementar que tem a probabilidade de descobrir uma classe de erros diferente daquela dos métodos de caixa branca. O teste de caixa preta procura descobrir erros nas seguintes categorias: (1) funções incorretas ou ausentes; (2) erros de interface; (3) erros nas estruturas de dados ou no acesso a bancos de dados externos; (4) erros de desempenho; e (5) erros de inicialização e término. Ao contrário do teste de caixa branca, que é executado cedo no processo de teste, o teste de caixa preta tende a ser aplicado durante as últimas etapas da atividade de teste, uma vez que o teste de caixa preta deliberadamente desconsidera a estrutura de controle, a atenção se concentra no domínio da informação.

Teste Caixa-Branca ou Estrutural segundo Pressman (1995), é um método de projeto de casos de teste que usa a estrutura de controle do projeto procedimental para derivar casos de teste.

Usando métodos de teste de caixa branca, o engenheiro de software pode derivar os casos de teste que (1) garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez; (2) exercitem todas as decisões lógicas para

valores falsos ou verdadeiros; (3) executem todos os laços em suas fronteiras e dentro de seus limites operacionais; e (4) exercitem as estruturas de dados internas para garantir a sua validade.

Segundo Rocha (2001), o teste de caixa branca estabelece os requisitos de teste com base em uma determinada implementação, verificando se atende aos detalhes do código e solicitando a execução de partes ou de componentes elementares do programa. Os critérios pertencentes a essa técnica são classificados com base no fluxo de controle, no fluxo de dados e na complexidade.

De acordo com Inthurn (2001), o teste estrutural tem como objetivo verificar se a estrutura interna da unidade está correta. Esta verificação é efetuada através de casos de teste que visam percorrer todos os caminhos internos possíveis da unidade e percorrer todos os caminhos.

2.4 TESTE FUNCIONAL

De acordo com Mueller (1998), o teste funcional também é conhecido como teste caixa preta pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída. Na técnica de teste funcional são verificadas as funções do sistema sem se preocupar com os detalhes de implementação.

Segundo Maldonado (1998), o teste funcional envolve dois passos principais: identificar as funções que o software deve realizar e criar casos de teste capazes de checar se essas funções estão sendo realizadas pelo software. As funções que o software deve possuir são identificadas a partir de sua especificação. Assim, uma especificação bem elaborada e de acordo com os requisitos do usuário é essencial para esse tipo de teste.

Segundo PUCRS, testes caixa-preta são aqueles que encaram o sistema a ser testado como uma função que mapeia um conjunto de valores de entrada em um conjunto de valores de saída sem se preocupar com a forma como esse mapeamento foi implementado. Testes caixa-preta baseiam-se exclusivamente na especificação de requisitos para determinar que tipo de saídas são esperadas para um determinado conjunto de entradas.

Inicialmente serão apresentadas as técnicas de teste caixa-preta tradicionais individualmente. Ao final deste capítulo será apresentado um modelo de plano de testes, bem como uma estratégia de aplicação das técnicas apresentadas.

2.4.1 TIPOS DE TESTE FUNCIONAL

Serão apresentados a seguir alguns tipos de teste funcional: teste de valor limite, generalizando a técnica de valor limite, limites da análise de valor limite, teste de robustez, teste de pior caso, testes a partir dos resultados

2.4.1.1 TESTE DE VALOR LIMITE

A técnica de teste de valor limite é a mais simples das técnicas tradicionais. Assim como as demais técnicas caixa-preta, explora a natureza funcional de um programa para identificar casos de teste.

Considere como exemplo um programa com 2 variáveis, *x1* e *x2*, com limites claramente determinados:

a <= x1 <= b
c <= x2 <= d

A técnica de análise do valor limite foca os limites dos espaços de entrada de maneira a determinar casos de teste. Este princípio parte da observação de que os erros costumam ocorrer próximos dos valores limites das variáveis de entrada.

A idéia básica da técnica é usar os valores de entrada no seu máximo, logo abaixo do máximo, um valor nominal, logo acima do mínimo e o valor mínimo para gerar casos de teste. Normalmente estes valores são nomeados: MIN, MIN+, NOM, MAX-, MAX.

Além disso, a técnica do valor limite baseia-se na idéia de que uma falha dificilmente tem origem em mais de um erro. Por esta razão os casos de teste são, em geral, obtidos fixando-se os valores de todas as variáveis de entrada menos uma, em seus valores nominais, deixando que a variável escolhida assuma seus valores extremos. Para o caso do exemplo, os casos de teste seriam: {<x1nom, x2min>, <x1nom, x2min+>, <x1nom, x2nom>, <x1nom, x2max->,x1nom,x2max+>, <x1min, x2nom>, <x1min+, x2nom>, <x1max-, x2nom>, <x2max, x2nom> }.

2.4.1.1.1 GENERALIZANDO A TÉCNICA DE VALOR LIMITE

A técnica básica de análise de valor limite pode ser generaliza de duas formas:

- pelo número de intervalos
- pelo tipo dos intervalos

Generalizar pelo número de intervalos é fácil. Para uma função de *n* variáveis teremos *4n+1* casos de teste. Generalizar pelo tipo depende da natureza das variáveis. Em alguns casos pode ser necessário usar enumerações (exemplo: meses do ano). Em outras situações as variáveis não tem limites discretos bem definidos (como no problema do triângulo). Neste caso limites artificiais tem de ser criados. No caso do problema do triângulo pode-se assumir, por exemplo, que o menor lado admitido é 1 e que o maior é MAXINT (ou o maior inteiro possível de ser representado). Análise de valor limite não faz sentido, por exemplo, em variáveis booleanas.

2.4.1.1.2 LIMITES DA ANÁLISE DE VALOR LIMITE

Análise de valor limite funciona bem quando o programa a ser testado é uma função de varias variáveis **independentes** que **representam conjuntos que tenham uma relação de ordem**, como por exemplo, quantidades físicas. Análise de valor limite preocupa-se apenas com os limites extremos das variáveis analisadas de forma independente, não levando em consideração a semântica das mesmas.

Por exemplo, em uma aplicação que lida com temperatura e pressão, é extremamente útil testar o que acontece com o programa de controle de uma caldeira quando a temperatura e pressão chegam próximas aos valores limites. Já em um programa que lida com números de telefone, qual seria a utilidade de testar os casos: { 0000, 0001, 9998, 9999}?

A análise de valor limite é a mais limitada e rudimentar das técnicas de geração de casos de teste e por esta razão os casos de teste gerados são igualmente limitados e rudimentares.

2.4.1.2 TESTE DE ROBUSTEZ

Teste de robustez é uma simples extensão do teste de valor limite que procura analisar o que ocorre quando os valores previstos para uma variável superam ligeiramente o limite superior (max+) ou ficam ligeiramente abaixo do limite inferior (min-). Neste caso termos 6n+1 casos de teste.

A principal diferença nesse caso diz respeito aos resultados esperados. Exemplo: o que acontece se o avião exceder o ângulo de aproximação? o que acontece se o elevador estiver com excesso de carga? o que acontece se o tamanho do arquivo for maior que o tamanho do disco?

2.4.1.3 TESTE DO PIOR CASO

A análise do valor limite verifica as variáveis isoladamente. O teste do pior caso esta interessado em verificar o que ocorre quando mais de uma variável pode assumir valores extremos simultaneamente. Para tanto consideramos os 5 valores assumidos para cada variável no teste de valor limite (min,min+,nom,max-,max) e montamos o produto cartesiano destes valores. Para *n* variáveis teremos, então, 5^n casos ao invés de *4n+1*.

Para as situações de extrema paranóia pode-se usar o teste de robustez para o pior caso. Nessa situação faz-se o produto cartesiano dos valores selecionados para o teste de robustez. Sendo assim, para *n* variáveis serão 7^n testes.

2.4.1.4 TESTANDO A PARTIR DOS RESULTADOS

Existem algumas situações onde o teste de valor limite (e suas variações) pode ser aplicado com melhores resultados se os intervalos de entrada forem definidos com base nas condições que determinam os valores de saída. Consideremos a seguinte situação: "... o cálculo do desconto por dependente é feito da seguinte forma: a entrada é a idade do dependente que deve estar restrita ao intervalo [0; 24]. Para dependentes até 12 anos (inclusive) o desconto é de 15%. Entre 12 e 18 (inclusive) o desconto é de 12%. Dos 18 aos 21 (inclusive) o desconto é de 5% e dos 21 aos 24 de 3%...".

Aplicando o teste de valor limite convencional serão obtidos casos de teste semelhantes a este: {0,1,12,23,24}. Mesmo usando o teste de robustez (por se tratar de apenas

uma variável de entrada o teste do pior caso não faz sentido) os valores de teste seriam: {-1,0,1,12,23,24,25}.

Note que com estes valores não é possível testar se o programa funciona corretamente. O primeiro e o último valores correspondem a valores fora de faixa. O segundo e o terceiro valor testam a primeira faixa de desconto. O quarto valor testa a segunda faixa de desconto e o quinto e o sexto valores testam a quarta faixa de desconto. Note que não foi gerado nenhum caso de teste para a terceira faixa de desconto, nem os limites destas faixas estão sendo testados.

Para resolver esse problema podemos subdividir o intervalo da variável de entrada de acordo com as saídas possíveis. Desta forma teríamos os seguintes intervalos para analisar: {[0;12], (12;18], (18;21], (21;24] }. Aplicando o teste convencional obteremos os seguintes casos de teste: {0,1,6,11,12,15,17,18,19,20,21,22,23,24 }. Observe que agora todos as faixas de desconto estão sendo testadas.

2.5 GERENCIAMENTO DOS TESTES

De acordo com Pressman (1995), em cada projeto de software, há um conflito de interesses inerente que acontece quando o teste se inicia. As pessoas que construíram o software agora são solicitadas a testar o software. Isso parece inofensivo em si mesmo, afinal de contas, quem conhece melhor o programa do que seus desenvolvedores? Infelizmente, esses mesmos desenvolvedores tem interesses subliminares de demonstrar que o programa é isento de erros, que ele funciona de acordo com as exigências do cliente e que ele será concluído no prazo e dentro do orçamento. Cada um desses interesses provoca um abrandamento no sentido de descobrir erros ao longo do processo de testes.

Freqüentemente, há uma série de interpretações errôneas que podem ser equivocadamente inferidas da discussão anterior:

a) que de modo algum o desenvolvedor de software deve testar;

b) que o software deve ser “jogado sobre o muro” a estranhos que o testarão impiedosamente;

c) que os responsáveis pelo teste se envolverão com o projeto somente quando os passos de teste estiverem prestes a se iniciar.

Cada uma das observações é incorreta.

A equipe de desenvolvimento do software é sempre responsável por testar as unidades individuais (módulos) do programa, para garantir que cada uma execute a função para a qual foi projetada. Em muitos casos, a equipe de desenvolvimento também realiza testes de integração – a etapa de teste que leva à construção (e teste) da estrutura de programa completa. Só depois que a arquitetura de software está completa é que um grupo de teste independente envolve-se.

Porém, a equipe de desenvolvimento do software não “joga” o programa para o grupo independente de teste (*Independent Test Group* – ITG) e afasta-se. A equipe de desenvolvimento e o ITG trabalham estreitamente unidos ao longo do projeto de software a fim de garantir que testes cuidadosos sejam levados a efeito. Enquanto a atividade de teste é realizada, a equipe de desenvolvimento deve estar à disposição para corrigir erros que sejam descobertos.

Segundo Hetzel (1987), a maioria dos conhecedores de técnicas gerenciais concordaria que são muitos os enfoques e estilos possíveis. O “melhor” enfoque é em grande parte fruto de preferências individuais, sendo a personalidade e o estilo de comportamento (como o indivíduo influencia as pessoas) os dois fatores críticos para o sucesso. Entretanto, além dos aspectos pessoais, há obrigações e responsabilidades que todo gerente deve assumir de modo a gerenciar os testes com eficiência.

Até aqui, tem-se considerado os testes sob um ponto de vista estritamente técnico. Quase não se dá atenção aos aspectos gerenciais. O que sabemos sobre o papel da gerência em qualquer empreendimento? Em termos simples, a missão da gerência é obter resultado através do trabalho dos outros. Muito já se escreveu sobre os motivos que levam alguns gerentes a conseguir tanto e outros tão pouco. As responsabilidades básicas são sempre as mesmas. Praticamente toda a atividade gerencial pode ser agrupada em uma dessas três esferas amplas de influência.

A esfera de liderança exige que se apontem os caminhos e se motivem os indivíduos em busca de metas comuns. Incluem-se aqui a definição de objetivos, expectativas e planos. A esfera de controle concentra-se nos procedimentos que fazem a empresa permanecer no caminho traçado. Ela envolve a monitoração, o *follow-up*, os relatórios, as avaliações e o redirecionamento. A esfera de suporte trata de instrumentos que facilitem o desempenho dos funcionários. Ela engloba o treinamento, os métodos de trabalho, as ferramentas disponíveis e assistência em geral.

Normalmente, será preciso empreender um programa de ação equilibrado, abrangendo as três esferas para que se obtenham progressos significativos ou duradouros na organização. Um exemplo simples: seria obviamente ineficaz tentar implantar uma nova técnica de teste (ou qualquer outra técnica), fornecendo apenas treinamento e suporte técnico, sem nenhum envolvimento pessoal, mantendo-se apenas uma esperança otimista de que a nova técnica tome de assalto, por milagre, todo o departamento de processamento de dados. Sem exceção, é fundamental a presença de uma liderança ativa e de controles efetivos paralelamente ao suporte/apoio para que as melhorias desejadas possam ser obtidas. O conceito de “programa de ação equilibrado” em todas as três esferas pressupõe que a gerência disponha de responsabilidades concretas de liderança e controle (além de incumbência restrita de contratar profissionais e assisti-los de todas as maneiras possíveis) que devam ser cumpridas e cobradas para que seja possível introduzir e consolidar métodos de teste eficazes.

Conforme Mueller (1998), as atividades de testes devem ser iniciadas no momento em que a equipe de desenvolvimento achar conveniente dentro de cada fase do ciclo de vida.

Durante o processo de testes uma série de pessoas participam de diversas atividades:

a) **Gerente de projeto**: É o agente responsável pelos assuntos administrativos relativos ao desenvolvimento do projeto.

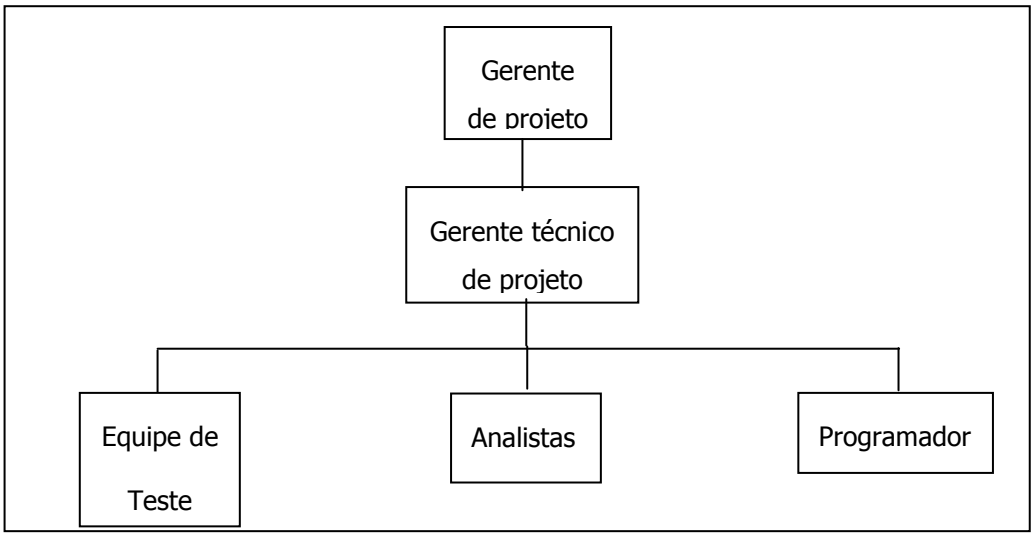
b) **Gerente Técnico de Projeto**: É o agente responsável pela condução dos assuntos técnicos relativos ao desenvolvimento do projeto.

c) **Equipe de teste**: É o grupo formado por elementos responsáveis pela condução dos testes e pela garantia de qualidade do projeto.

d) **Analista**: É o agente responsável pelo desenvolvimento ou manutenção dos módulos automatizados do sistema de informação.

e) **Programador**: É o agente responsável pela construção e teste das unidades e suas adequações ao projeto físico.

Figura 1 - Fluxograma do pessoal envolvido com a atividade de testes



2.6 CHECKLIST

Segundo Rocha (2001), dentre as atividade de garantia de qualidade de software estão as de VV&T (Verificação, Validação e Teste), com o objetivo de minimizar a ocorrência de erros e risco associados. O objetivo da verificação é assegurar que o software, ou uma determinada função do mesmo, esteja sendo implementado corretamente. Verifica-se, inclusive, se os métodos e processos de desenvolvimento forma adequadamente aplicados. As atividade de VV&T envolvem atividades de análise estática e de análise dinâmica do produto em desenvolvimento, que podem ser realizadas de maneira manual ou automatizada, dependendo da disponibilidade de ferramenta de apoio.

A análise estática não envolve a execução propriamente dita do produto e visa determinar propriedade do produto válidas para qualquer execução do produto final. Na realidade, a análise estática pode e deve ser aplicada em qualquer produto intermediário do processo de desenvolvimento. Por exemplo, em nível de especificação com base em *statecharts*. Utilizando-se o conceito de árvore de alcançabilidade, seria possível verificar se a especificação é consistente, se o produto não tem *deadlock* etc. revisões técnicas são o

exemplo mais clássico de análise estática. As revisões tem um processo subjacente e devem ser sistemáticas e planejadas. Em geral são conduzidas com base em uma *checklist* (lista de verificação ou conferência) adequada a cada fase ou produto.

A análise dinâmica também tem por objetivo detectar defeitos ou erros no software e envolve a execução do produto, seja o código propriamente deito ou mesmo uma especificação executável. As atividades de simulação e teste constituem uma análise dinâmica do produto. O teste é um elemento usado para fornecer evidências da confiabilidade do software em complemento a outras atividades, como o uso de revisões e de técnicas formais e rigorosas de especificação e de verificação, sendo relevante para a identificação e eliminação de erros que persistem no software.

Na essência, as técnicas de VV&T tem por objetivo identificar a presença de defeitos ou erros o mais cedo possível no processo de desenvolvimento e aprender com essa atividade e evoluir o processo de desenvolvimento, inclusive pela própria evolução das técnicas de VV&T utilizadas na empresa.

2.7 PROCESSO DE TESTE

O processo de teste antes da implantação deste sistema era feito da seguinte forma:

- As rotinas são desenvolvidas pelos programadores e repassadas junto com a SMS correspondente para o suporte técnico, para fazer os testes destas rotinas;
- O suporte técnico faz os testes e se ocorrem erros, é devolvida a SMS para o programador indicando o erro que aconteceu;
- Enquanto o suporte técnico não indicar que a rotina está correta, a SMS fica sendo passada para o programador e do programador para o suporte técnico.

O processo de teste depois da implantação deste sistema é feito da seguinte forma:

- As rotinas são desenvolvidas pelos programadores e repassadas junto com a SMS para o controle de qualidade, para fazer os testes destas rotinas;

- O controle de qualidade faz os testes e se ocorrem erros, é devolvida a SMS para o programador indicando o erro que aconteceu;

- Enquanto o controle de qualidade não indicar que a rotina está correta, a SMS fica sendo passada para o programador e do programador para o controle de qualidade.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo serão descritas as atividades desempenhadas em cada fase do modelo de desenvolvimento de software adotado para construção do trabalho. Também serão relacionados e discutidos os resultados obtidos a partir do mesmo.

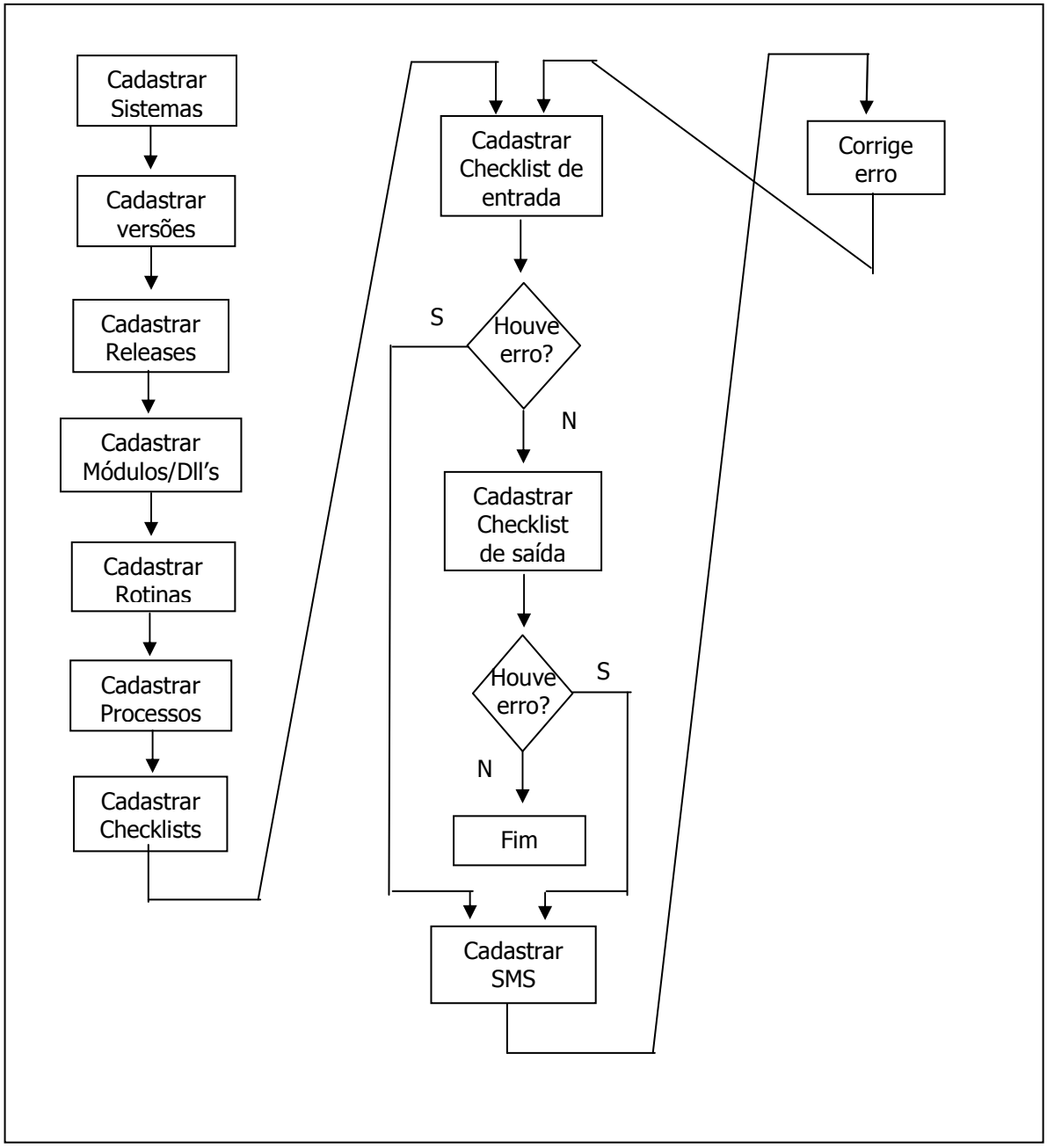
3.1 REQUISITOS DO PROBLEMA

Os testes na empresa Benner Sistemas S/A são feitos pelos programadores, que são verificações básicas do sistema, como valores aceitos por um determinado campo. Após o término do desenvolvimento é feita a liberação do programa para que o suporte técnico das áreas de recursos humanos e corporativo da empresa, que além de darem suporte a clientes Benner fazem os testes de processos do sistema, como integração entre os módulos, processos executados pelo sistema como folha de pagamento. Com o número crescente de clientes, o suporte estava ficando com pouco tempo para testar os sistemas, ocasionando muitas vezes a verificação de erros apenas quando o sistema já esta no cliente, causando um grande problema para a empresa, pois deveria liberar uma correção para uma versão quem não tinha nem sido começado a usar.

Com a verificação deste problema e o crescimento acelerado da Benner, verificou-se que deveria se ter uma área ou pessoa na empresa que se dedicasse integralmente ao teste dos sistemas Benner em cada área da empresa. A partir disto surgiu à idéia de realizar um estágio na área de teste de software.

Foram feitas reuniões semanais entre este autor, o Sr. Roger Wetzel, que é o responsável pelos treinamentos dos clientes para a utilização do sistema de Recursos Humanos e que irá utilizar o sistema de gerenciamento de testes neste setor e a Srta. Regiani Dalfovo, que é a responsável pelos treinamentos dos clientes para a utilização do sistema Corporativo e que irá utilizar o sistema de gerenciamento de testes neste setor. Nas primeiras reuniões foi definida a interface do programa, utilizando a ferramenta Benner Builder para a definição das tabelas do programa. Com o passar das reuniões o software foi tomando forma com a utilização da ferramenta Benner Runner para fazer a interface do software, como mostra o fluxograma da figura 2.

Figura 2 –Fluxograma



3.2 ANÁLISE DOS REQUISITOS

O fluxograma mostra que o sistema necessitará dos cadastros dos sistemas da Benner, das versões desses sistemas, das *releases* dessas versões, os módulos de cada sistema, as rotinas de cada módulo, os processos que são envolvidos por essas rotinas e por fim os *checklists* que possuem uma descrição básica do processo e o cadastro do *checklist* de entrada e de que contém os dados de entrada e de saída que serão utilizados para este teste. Se ocorrer

um erro, será cadastrada uma SMS, que indica ao analista ou programador responsável por aquele módulo ou rotina que ele tem um erro para corrigir e devolver a SMS ao testador que irá verificar se o erro foi corrigido. A SMS é o nome dado ao cadastro de um erro que aconteceu no sistema, no cadastro da SMS é indicado o programador, o sistema, o cliente, a versão do cliente e um resumo do erro que aconteceu.

O sistema de gerenciamento de teste funcional será um módulo do Sistema de Controle das atividades – SisCon – para ficar integrado com os sistemas utilizados por todos os diretores, programadores e clientes da Benner. O sistema que esta sendo desenvolvido será utilizado e visualizado apenas pelos diretores e pelos testadores.

O sistema de gerenciamento de teste funcional no primeiro momento, será utilizado para o gerenciamento apenas dos sistemas de Recursos Humanos e Corporativo, pois são os sistemas Benner mais utilizados, passando a ser utilizado em todos os sistemas de acordo com a necessidade da empresa.

O sistema de gerenciamento de teste funcional, é o primeiro passo para a implantação do setor de qualidade de sistemas na empresa, pois a partir deste deverão ser desenvolvidos mais sistemas que auxiliarão os funcionários que irão testar os sistemas.

3.3 ESPECIFICAÇÃO

Esta seção descreve os diagramas para a especificação do software, apresentando o diagrama de casos de uso, diagrama entidade relacionamento (DER) e o dicionário de dados. Utilizou-se para a especificação do protótipo a ferramenta CASE ErWin e o Rational Rose.

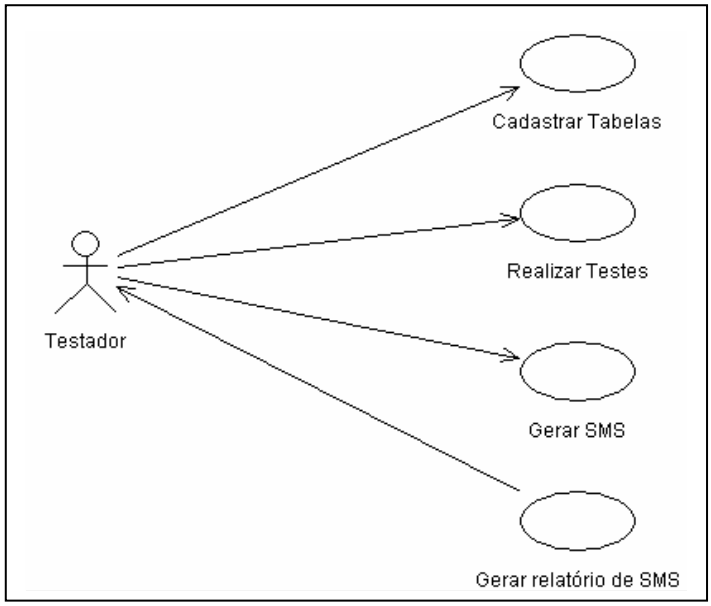
Para o desenvolvimento do sistema, utilizou-se o padrão de formulários, botões, cargas dos sistemas Benner. Em todas as tabelas dos Sistemas Benner possui um campo chamado handle, que é o campo de ligação entre as tabelas.

3.3.1 DIAGRAMA DE CASOS DE USO

O caso de uso identifica e esquematiza graficamente e descreve o comportamento do sistema. Estes diagramas apresentam uma visão geral nivelada de como o sistema é usado, na perspectiva de um estranho (ator). Este diagrama pode descrever alguns ou todos os casos de

uso de um sistema. Na Figura 3 estão representados os casos de uso implementados no protótipo do trabalho.

Figura 3 - Diagrama de casos de uso



A principal figura ou ator é referenciado como testador, ele é que fará entradas e receberá os resultados em forma de relatório. Para que seja possível atender todas as necessidades do testador, foram criados os seguintes casos de uso:

Cadastrar Tabelas: Este caso trata das informações cadastrais do aplicativo, através destes cadastros será possível partir para os próximos passos do aplicativo. As tabelas utilizadas para este caso são: Sistemas, Versões, *Releases*, Módulos, Rotinas, Atividades, RotinaProcessos, Checklists, ChecklistEntrada e ChecklistSaída.

Realizar Testes: Trata da realização dos testes, utilizando os checklists que foram cadastrados. O testador poderá realizar os testes baseados em documentos com *print-screens* das telas do processo que estará sendo testado.

Gerar SMS: Trata do cadastro de erros encontrados no teste. A SMS será encaminhada ao responsável do módulo que deverá corrigir o erro relatado e devolver a mesma para o testador, a fim deste verificar se o erro foi corrigido realmente.

Gerar Relatório de SMS’s: Utilizará as tabelas de movimento e cadastro, para gerar o comparativo gerencial de apontamento de resultados das métricas.

3.3.2 DIAGRAMA ENTIDADE RELACIONAMENTO

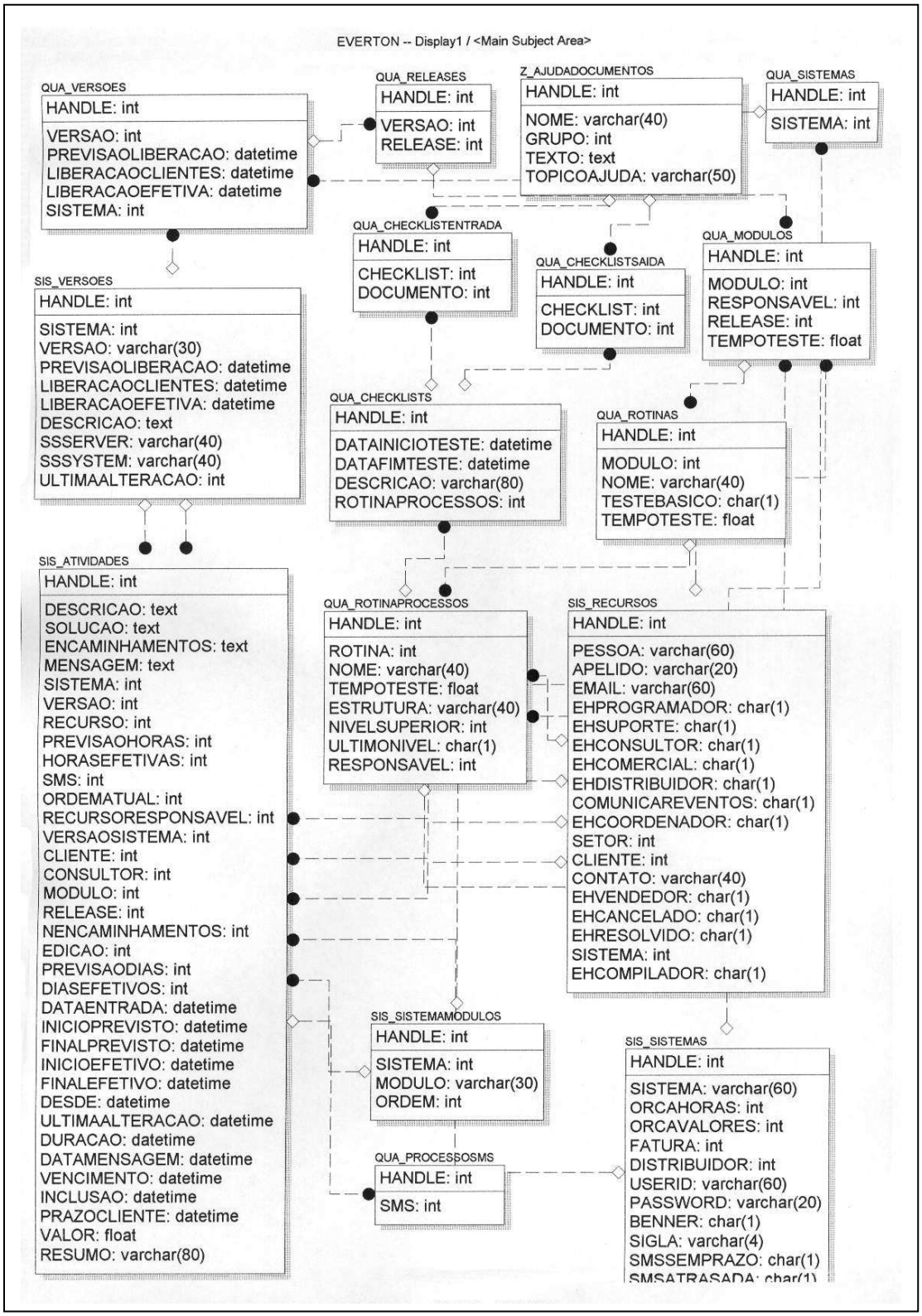
O diagrama entidade relacionamento (DER), enfatiza os principais objetivos ou entidades do sistema e seus relacionamentos.

Como se pode observar, todas as tabelas possuem um campo handle, que na Benner é o campo que indica o registro da tabela, que serve para fazer a ligação entre as tabelas, como por exemplo, de uma tabela de clientes fosse o código do cliente. Também pode-se observar que todas as entidades possuem obrigatoriedade com relação à entidade principal, que é a tabela de sistemas, permitindo um relacionamento de 1 para N ou 1 para 1, ou seja, torna-se obrigatório existir um registro pai para que se cadastre um ou mais registros na entidade filho, como é mostrado na figura 4.

3.3.3 DICIONÁRIO DE DADOS

O dicionário de dados contém a definição de todos os dados mencionados no DER, as entidades e seus atributos, incluindo detalhes do formato físico, como: tipo, tamanho, chave e descrição do atributo. Gerou-se um relatório na ferramenta CASE ErWin, utilizada para o desenvolvimento da modelagem, através dos Scripts. No anexo 1 estão descritos as tabelas de cada entidade com seus respectivos atributos, a primeira coluna mostra o nome das tabelas, a segunda coluna mostra as campos das tabelas, a terceira coluna mostra o tipo do campo e a quarta coluna indica se o campo é ou não *foreign-key*.

Figura 4 – Diagrama entidade relacionamento



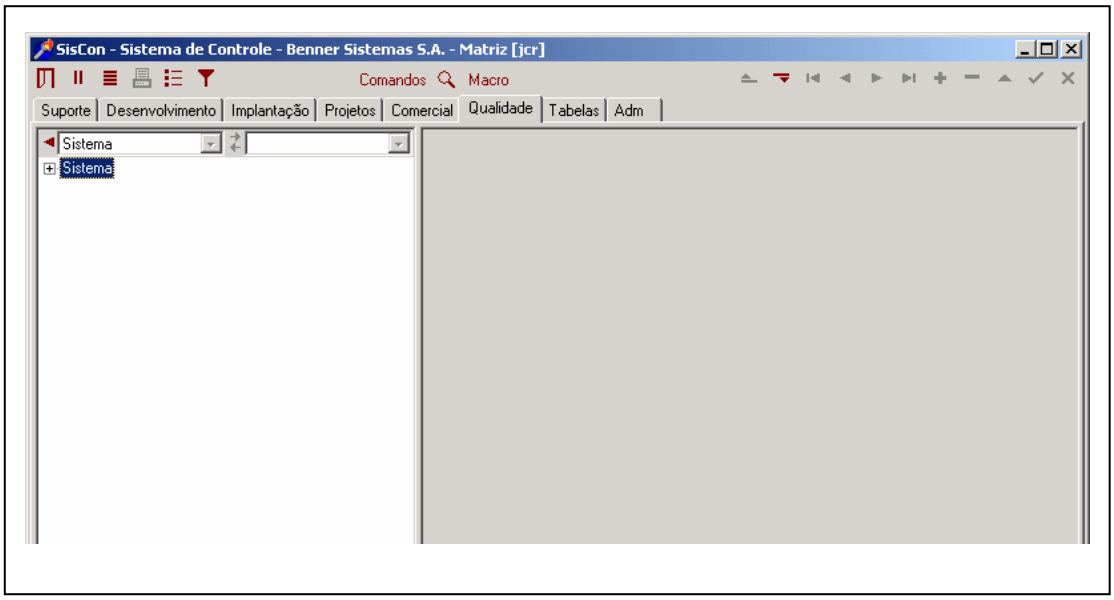
3.4 IMPLEMENTAÇÃO

A seguir são apresentadas as principais telas disponíveis para utilização passo a passo do protótipo. Com intuito de facilitar a demonstração e compreensão, será realizado um teste do sistema de recursos humanos da Benner Sistemas, que é o sistema utilizado para o gerenciamento dos dados cadastrais e da folha de pagamento dos funcionários.

3.4.1 MENU PRINCIPAL

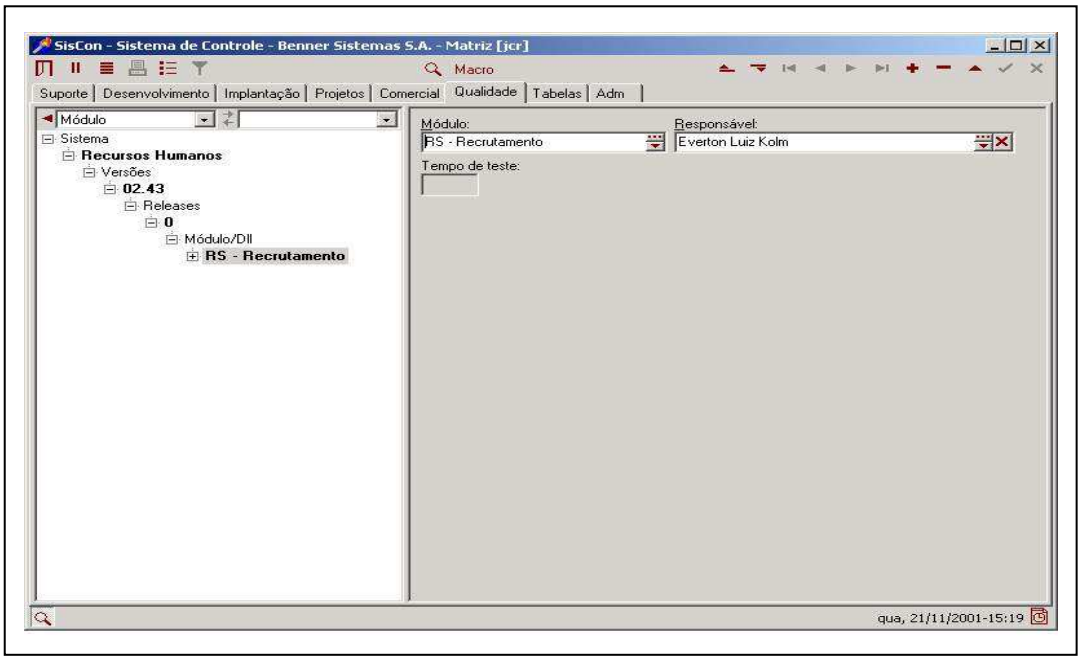
Ao executar o aplicativo, será apresentada a tela de abertura do sistema, disponibilizando acesso aos demais recursos do protótipo, conforme figura 5.

Figura 05 – Tela de abertura



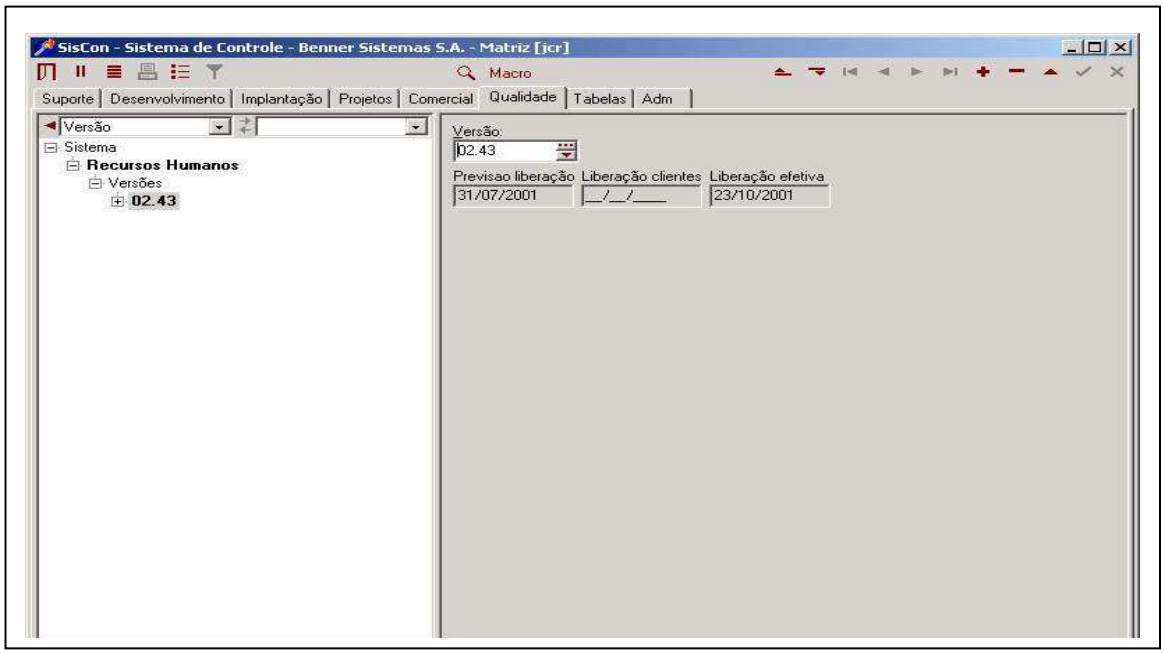
No formulário indicado na figura 6, serão cadastrados os sistemas Benner que serão testados utilizando o sistema de gerenciamento. Os sistemas estão cadastrados na tabela SIS_SISTEMAS, que é uma tabela já existente no Sistema de Controle da Benner. A tabela que conterá os dados referentes aos sistemas é a tabela QUA_SISTEMAS.

Figura 06 – Tela de cadastro de sistemas



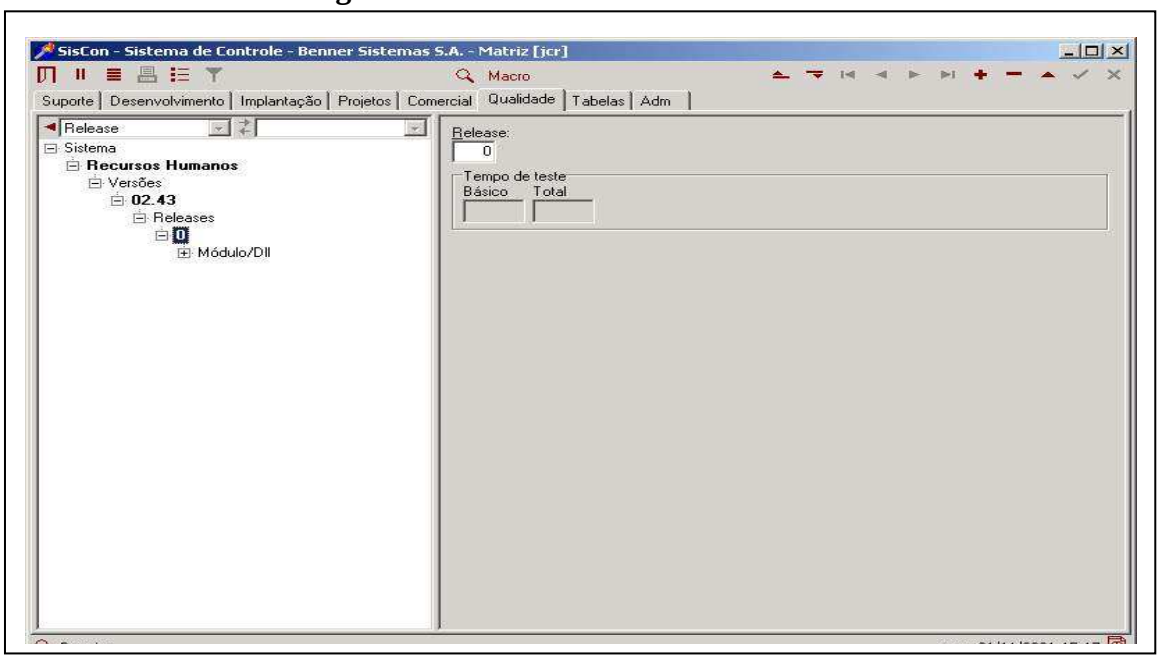
No formulário indicado na figura 7, serão cadastrados as versões dos sistemas Benner que serão testados. As versões estão cadastradas na tabela SIS_VERSOES, que é uma tabela já existente no Sistema de Controle da Benner. A tabela que conterá os dados referentes as versões é a QUA_VERSOES.

Figura 07 – Tela de cadastro de versões



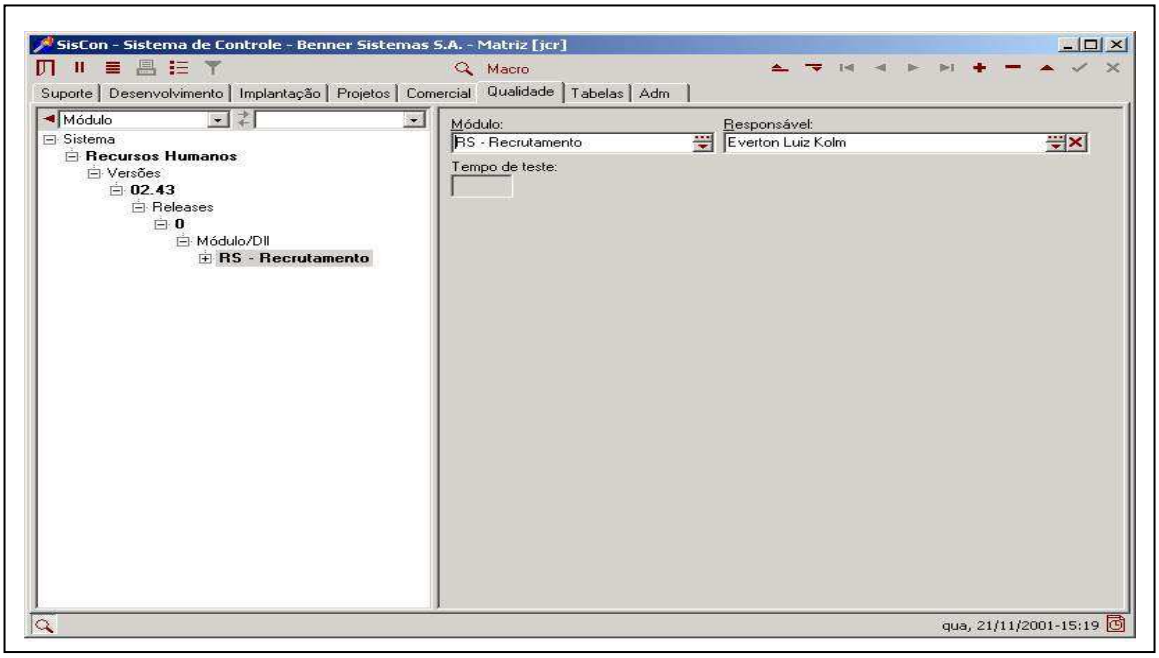
No formulário indicado na figura 8, serão cadastradas as *releases* de versões dos Sistemas Benner que serão testados. A tabela que conterá os dados referentes as *releases* do sistema é a QUA_RELEASES.

Figura 08 – Tela de cadastro de *releases*



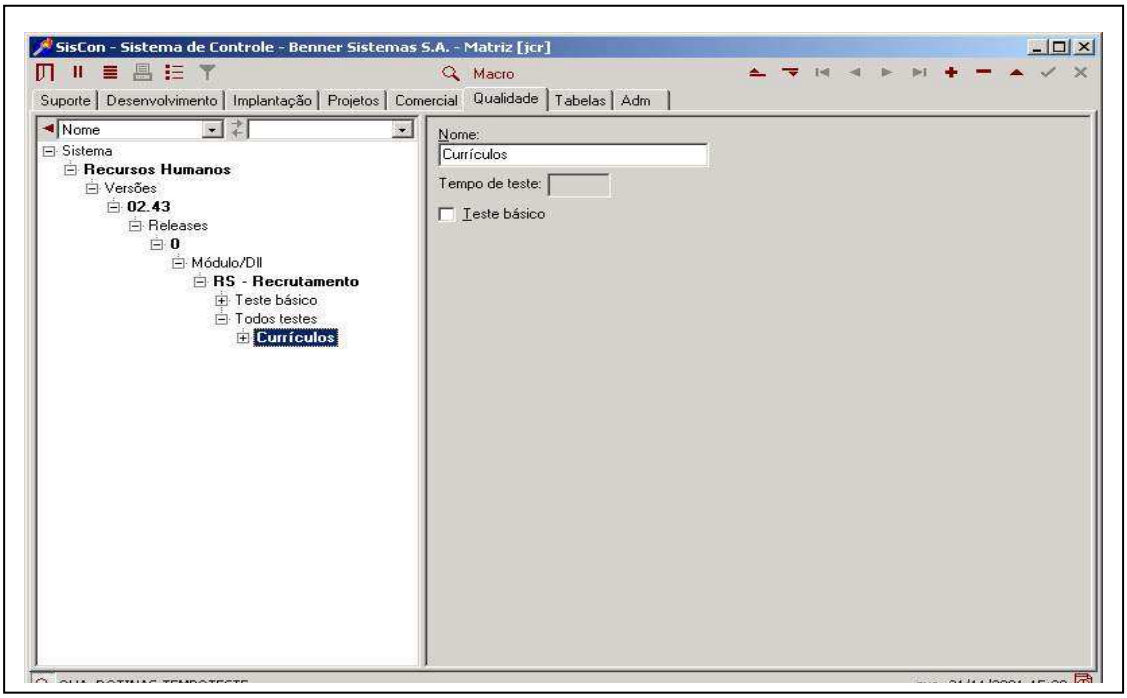
No formulário indicado na figura 9, serão cadastrados os módulos dos sistemas Benner que serão testados. Os módulos dos sistemas estão cadastrados na tabela SIS_SISTEMAMODULOS, que é uma tabela já existente no Sistema de Controle da Benner. A tabela do sistema que conterá os dados referentes ao módulos é a QUA_MODULOS.

Figura 09 – Tela de cadastro de módulos



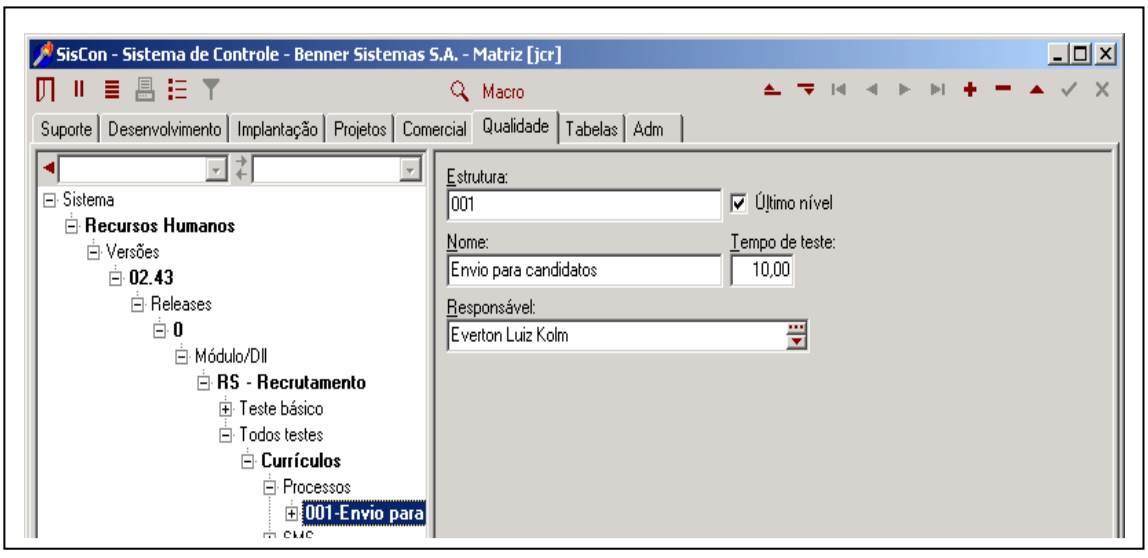
No formulário indicado na figura 10, serão cadastradas as rotinas dos sistemas Benner que serão testadas. A tabela do sistema que conterá os dados referentes as rotinas é a QUA_ROTINAS.

Figura 10 – Tela de cadastro de rotinas



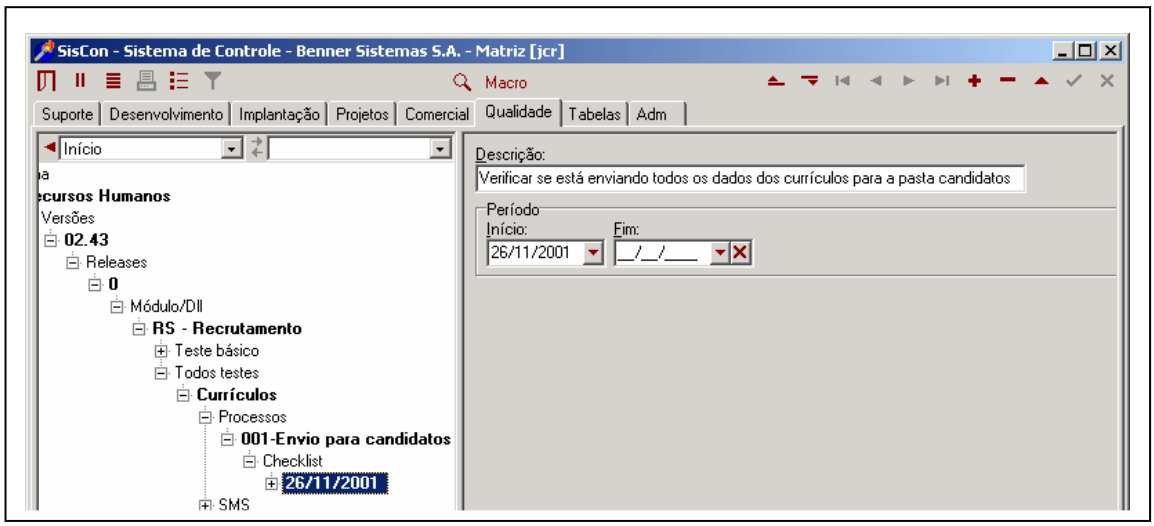
No formulário indicado na figura 11, serão cadastrados os processos das rotinas dos sistemas Benner que serão testadas. A tabela do sistema que conterá os dados referentes aos processos é a QUA_ROTINAPROCESSOS.

Figura 11 – Tela de cadastro de processos



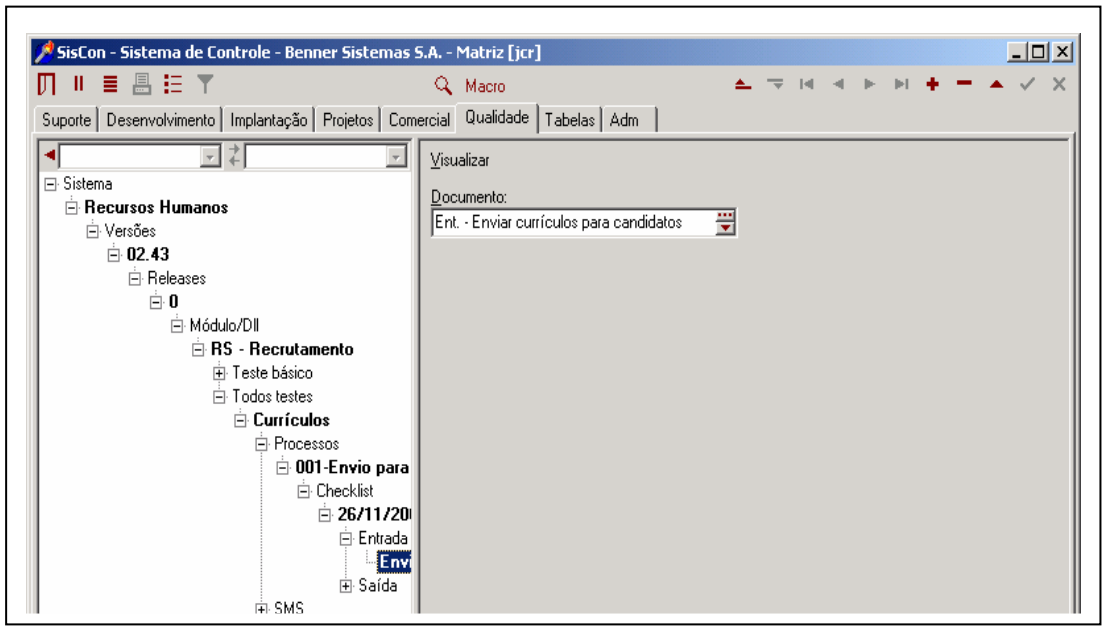
No formulário indicado na figura 12, será cadastrado o *checklist* com uma descrição básica do que será testado nos sistemas Benner. A tabela do sistema que conterá os dados referentes aos *checklists* é a QUA_CHECKLISTS.

Figura 12 – Tela de cadastro de checklists



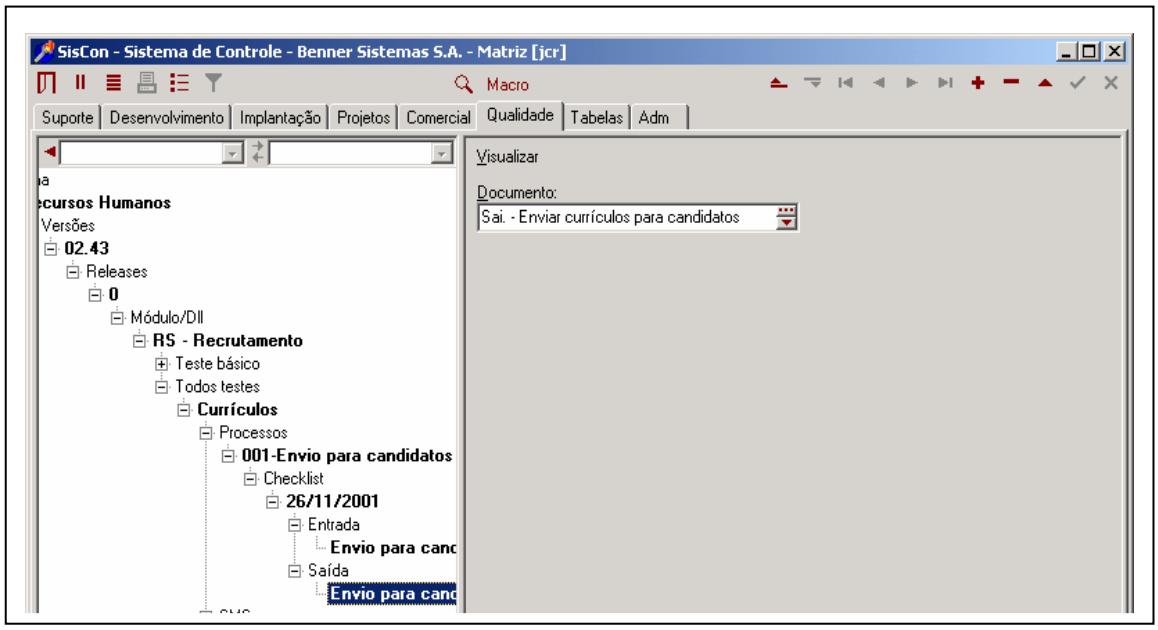
No formulário indicado na figura 13, será cadastrado o documento que conterá a forma como deverá ser a entrada de dados para a execução do teste. A tabela do sistema que conterá os dados referentes aos *checklists* de entrada é a QUA_CHECKLISTENTRADA.

Figura 13 – Tela de cadastro de checklists de entrada



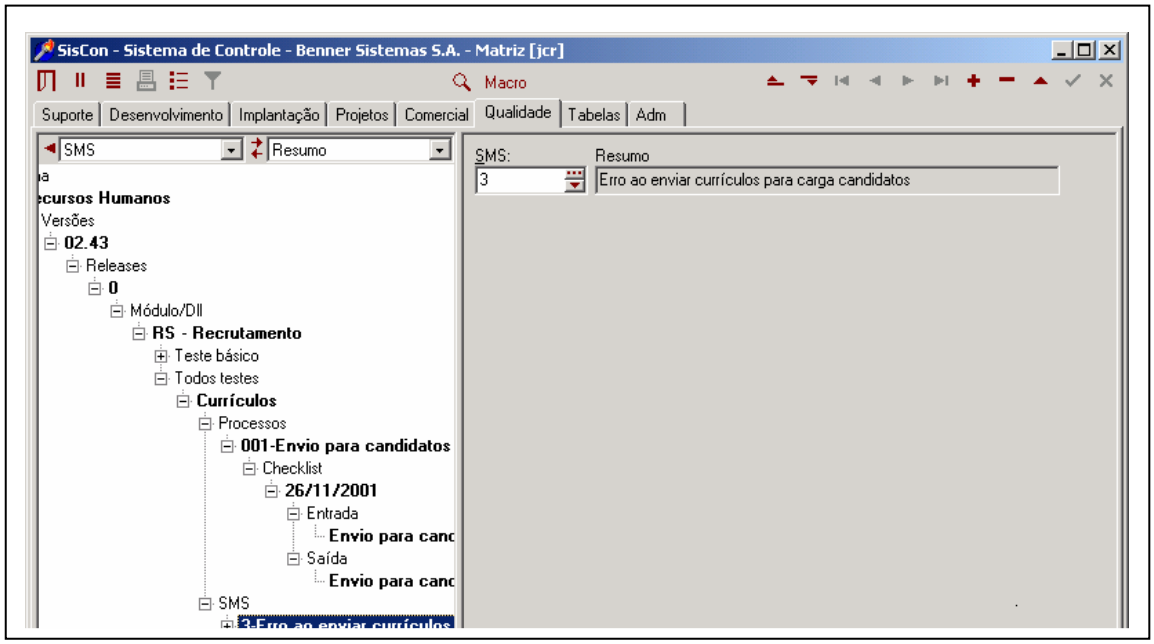
No formulário indicado na figura 14, será cadastrado o documento que conterá a forma como deverá ser a saída dos dados do teste. A tabela do sistema que conterá os dados referentes aos *checklists* de saída é a QUA_CHECKLISTSAIDA.

Figura 14 – Tela de cadastro de checklists de saída



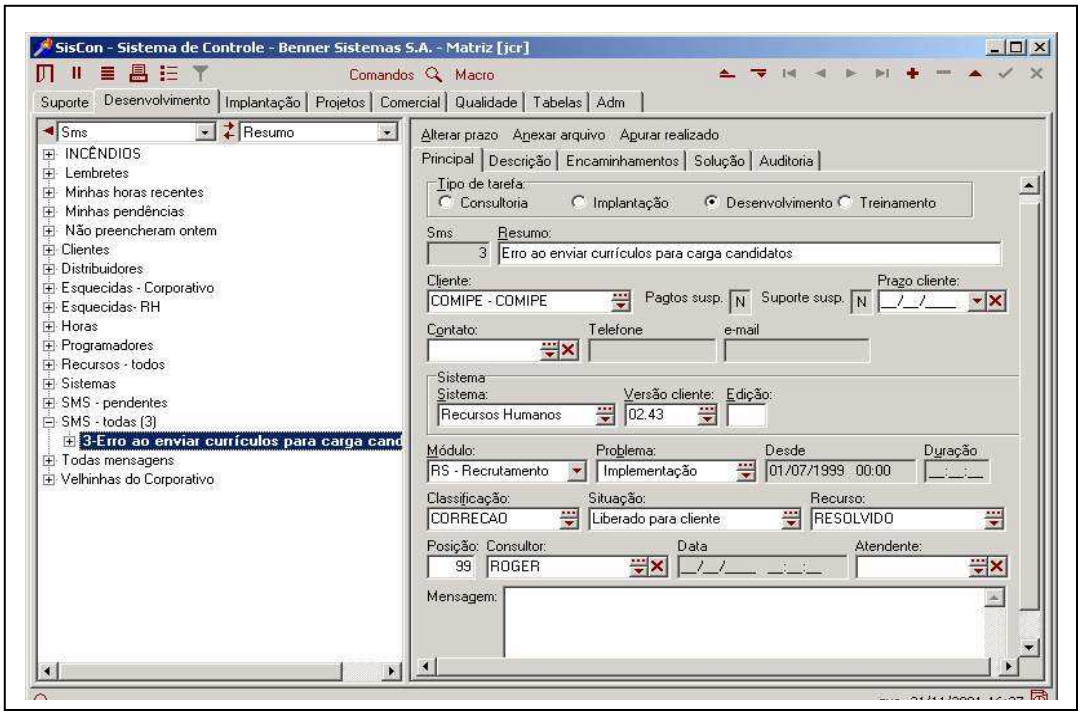
No formulário indicado na figura 15, será cadastrado o documento que conterá a forma como deverá ser a saída dos dados do teste. A tabela do sistema que conterá os dados referentes aos *checklists* de saída é a QUA_PROCESSOSMS.

Figura 15 – Tela de cadastro de SMS's do processo



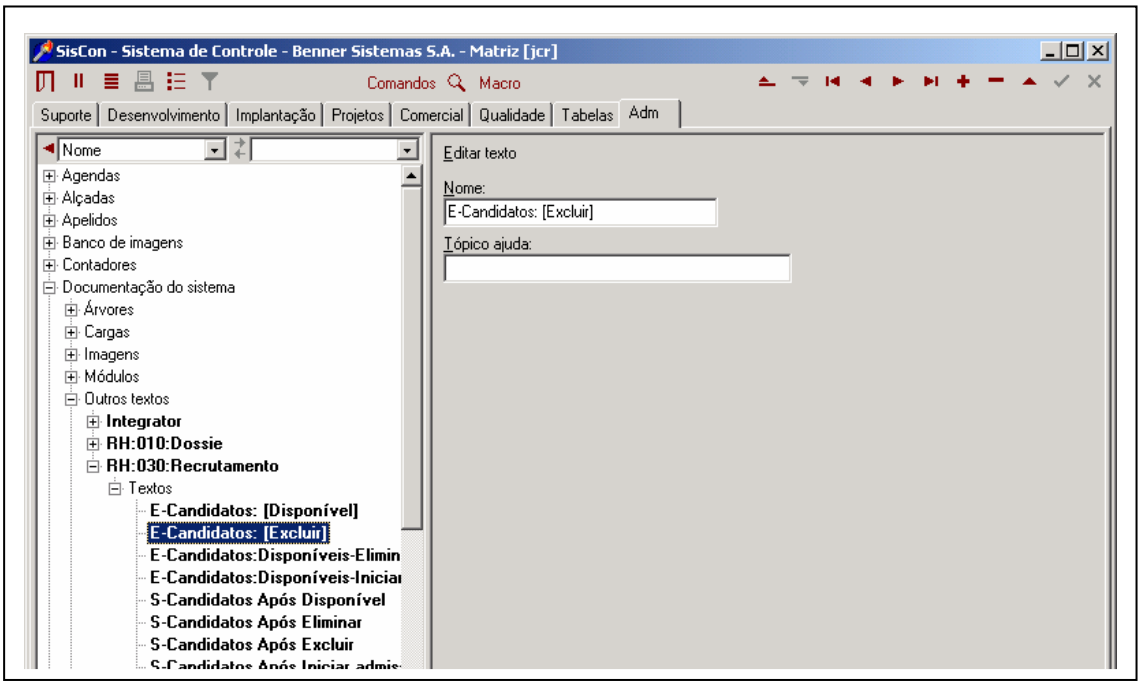
No formulário indicado na figura 16, será cadastrado o documento que conterá a forma como deverá ser a saída dos dados do teste. A tabela do sistema que conterá os dados referentes aos checklists de saída é a SIS_ATIVIDADES.

Figura 16 – Tela de cadastro de SMS's



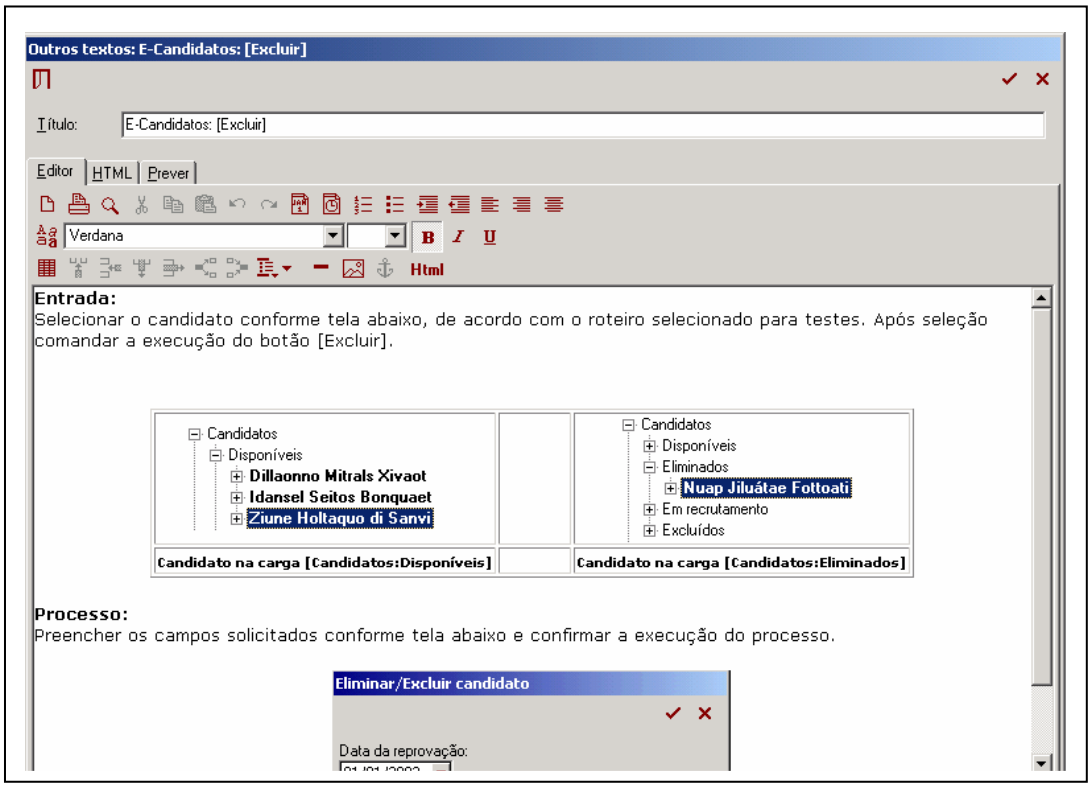
No módulo ADM serão cadastradas as figuras onde estarão os dados de entrada e de saída do teste, como mostra a figura 17. Foi definido o seguinte padrão para o nome das figuras: que as figuras que iniciam com a letra "E" as figuras que contem a entrada de dados para o teste e que as figuras que iniciam com a letra "S" as figuras que contem a saída dos dados do teste. A tabela do sistema que conterá os dados referentes aos documentos de saída é a SIS_AJUDADOCUMENTOS.

Figura 17 – Tela de cadastro de documentos



Selecionando um documento, o usuário irá definir o *checklist* com figuras, pois desta forma pode-se utilizar o teste também como documentação do sistema. Foi definido desta forma, pois desta forma as informações estariam gravadas junto com a base de dados do sistema e poderiam ser utilizadas como *help* do sistema. O padrão foi definido pelo usuário do sistema de recursos humanos, pois este usuário é o mesmo que faz a documentação e treinamento deste sistema. O exemplo segue na figura 18.

Figura 18 – Tela de exemplo de figura de teste cadastrada no sistema



3.4.2 CONSIDERAÇÕES SOBRE A IMPLEMENTAÇÃO

O banco de dados utilizado foi o MSSql versão 7.0, para a criação das tabelas do sistema foi utilizada a ferramenta Benner Builder e para a geração da interface foi utilizada a ferramenta Benner Runner, que são ferramentas proprietárias da Benner Sistemas. Na figura 19 mostra-se a ferramenta Benner Builder.

Segundo Benner (2001), a ferramenta Benner Runner é a ferramenta que apresenta a interface dos sistemas Benner aos usuários, de acordo com as especificações elaboradas no Benner Builder.

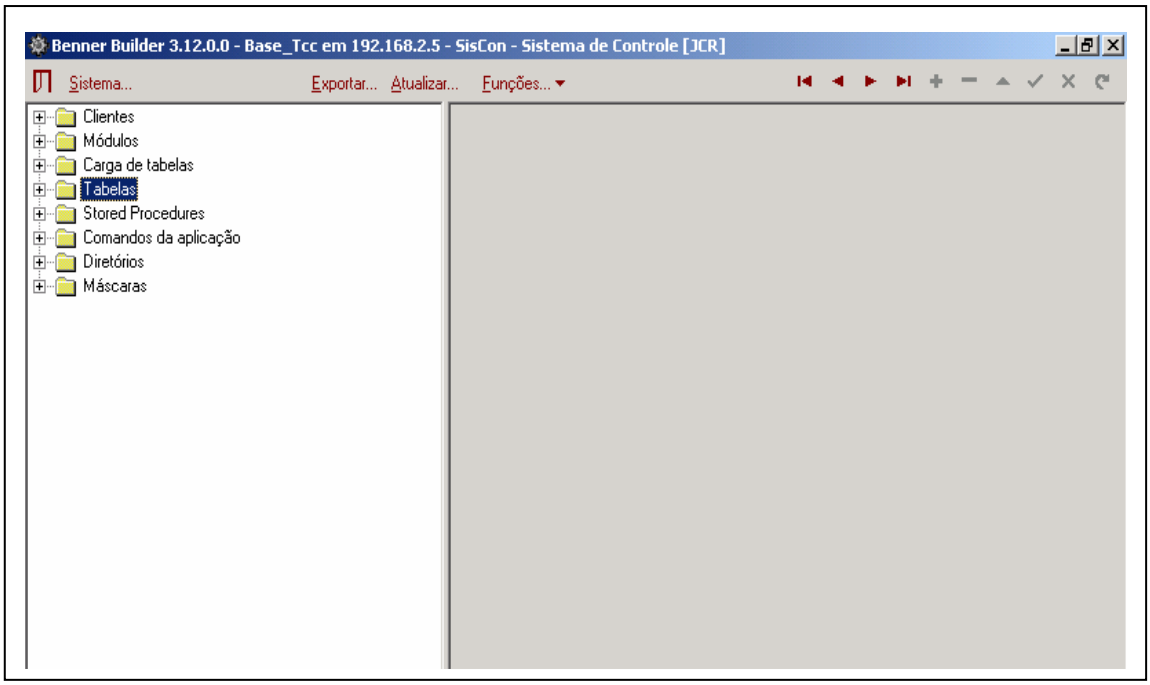
Segundo Benner (2001), a ferramenta Benner Builder é uma ferramenta de criação e manutenção das estruturas de dados e interface com o usuário, que pode ser utilizada com os principais bancos de dados do mercado. É ágil, fácil e permite atualizações automáticas de bases de dados de clientes que tiveram seus sistemas modificados para atendimento de características e peculiaridades.

No anexo 2 contém o script gerado pela ferramenta Benner Builder mostrando como são gerados os dados da ferramenta para serem utilizados em outras bases de sistemas Benner.

Para a implementação da rotina que faz a cópia de testes que devem ser realizados em toda versão de sistema que for liberada, foram utilizados alguns componentes padrões da linguagem Delphi 5.0, como os componentes Combo-box e label, e outros da Benner Sistemas, como os botões para seguirem o padrão Benner.

Para desenvolvimento do relatório utilizou-se a ferramenta Benner Report de geração de relatórios proprietária da Benner Sistemas.

Figura 19 – Tela da ferramenta Benner Builder



Segue abaixo uma tabela para fins de comparativo de *SMS's* geradas como correção antes (até 10/2001) e após (11/2001) a utilização do sistema de gerenciamento de testes de software:

Entrada	Todas	Correções	Melhoria	Verificações
06/2001	83	44	14	25
07/2001	103	64	14	25
08/2001	123	64	27	32
09/2001	177	88	30	59
10/2001	253	107	80	66
11/2001	214	53	71	90

Fonte: Benner Sistemas S/A.

4 CONCLUSÕES

O objetivo do estágio foi alcançado visto que o protótipo atende as atividades de teste funcional e estrutural pretendidas pela empresa, a partir de relatórios que a diretoria tem acesso que apontaram uma redução de cinqüenta por cento através de um desses relatórios mostra os erros encontrados pelos clientes nos sistemas. A verificação pode ser feita através do relatório semanal que a direção da empresa recebe com os erros encontrados pelos usuários de todos os sistemas Benner, através da comparação do relatório emitido anterior e após o uso do sistema de gerenciamento de teste de software funcional, que a direção percebeu a redução.

Como a empresa não possuía nenhum tipo de formalidade nos testes dos sistemas, este sistema serviu para mostrar como é importante fazer uma boa carga de testes, pois o custo para a correção de erros que foram encontrados em rotinas antes da liberação para clientes, é bem menor, além de não gerar problemas relacionados a funções básicas do sistema.

Com a utilização de *checklists*, ficou bem mais fácil a parte de testes, pois pode ser feita uma breve descrição do que deve ser testado, ficando mais fácil para os testadores saberem o que estão testando e o os resultados esperados daquele teste, pois neste sistema foi definido que as telas com os dados de entrada e saída seriam incluídas no *checklist*.

Este sistema esta utilizando a técnica de teste funcional: testar a partir de resultados. A utilização desta técnica mostrou-se eficiente para a situação desta empresa, pois ela não possuía nenhuma técnica de teste implantada.

A parte de gerenciamento que antes era feita sem saber se uma rotina foi testada, agora através do sistema você sabe quem testou, quando testou e se ocorreram erros. Assim poderá ser verificado se algum programador está fazendo muita correção de rotinas antigas e novas, e procurar saber o porque de estar acontecendo isto, se foi por falha na definição da rotina ou no levantamento dos dados, podendo assim ser cobrado o responsável real pelo erro.

4.1 SUGESTÕES

Para trabalhos futuros sugere-se o desenvolvimento de um sistema para o teste estrutural, que possibilite a verificação da execução do programa, pois desta forma os programadores poderiam verificar mais rapidamente os erros encontrados no teste das rotinas.

Também é sugerível a utilização de outras técnicas de teste funcional que não seja a de testar a partir de resultados, para o desenvolvimento de sistemas para o gerenciamento de teste.

Também é sugerível a aquisição de outra ferramenta para testes já existente no mercado.

REFERÊNCIAS BIBLIOGRÁFICAS

BENNER. **Tecnologia**, Blumenau. Disponível em: <http://www.benner.com.br>. Acesso em: 22 ago. 2001.

CANTÚ, Marco. **Dominando o Delphi 5 –** a bíblia. Tradução João E. N. Tortello. São Paulo: Makron Books, 2000.

DEMARCO, Tom. **Análise estruturada e especificação de sistemas**. Tradução de Maria Beatriz G. S. Veiga de Carvalho. Rio de Janeiro, 1989.

FOURNIER, Roger. **Guia prático para o desenvolvimento e manutenção de sistemas estruturados**. São Paulo: Makron Books, 1994.

HETZEL, Willian. **Guia completo ao teste software**. Rio de Janeiro: Campos, 1987.

INTHURN, Cândida. **Qualidade e teste de software.** Florianópolis: Visual Books, 2001.

MALDONADO, José Carlos et al. Gestão de configuração na atividade de teste. In: Workshop Qualidade de Software, 1., 1997, Fortaleza. **Anais do Workshop Qualidade de Software**: SBC, 1997. p. 107-116.

MALDONADO, José Carlos. Aspectos Teóricos e Empíricos de Teste de Cobertura de Software. In: VI Escola Regional de Informática, 6., 1998, Curitiba. **Anais da VI Escola Regional de Informática:** FURB, PUC-PR, UCP, 1998. p. 53-86.

MARTIN, James. McClure, Carma. **Técnicas estruturadas e CASE**. São Paulo: Makron Books, 1995.

MULLER, James. McClure, Carma. **Técnicas estruturadas e CASE**. São Paulo: Makron Books, 1995.

PAULA FILHO, Wilson de Pádua. **Engenharia de software:** fundamentos, métodos e padrões. Rio de Janeiro: LTC, 2001.

PRESSMAN, Roger. **Engenharia de software.** São Paulo: Makron Books, 1995.

PUCRS. **Professor Dr. Flávio Moreira de Oliveira**, Porto Alegre. Disponível em: <http:// http://www.inf.pucrs.br/~flavio/>. Acesso em: 30 nov. 2001.

ROCHA, Ana Regina C. da; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de software –** Teoria e prática. São Paulo: Prentice Hall, 2001.

UNICRUZ. **Universidade de Cruz Alta**, Cruz Alta. Disponível em: <http://dinf.unicruz.edu.br/~facco/projeto.htm>. Acesso em: 30 nov. 2001.

YOURDON, Edward. **Análise estruturada moderna**. Tradução Dalton Conde de Alencar. Rio de Janeiro: Campus, 1990.

Anexo I

Anexo II