Boletim Técnico da Escola Politécnica da USP
Departamento de Engenharia de Computação e
Sistemas Digitais

ISSN 1413-215X
BT/PCS/0124

Um Método de Teste de Software Baseado em Casos de Teste

Sérgio Ricardo Rotta
Kechi Hirama

São Paulo - 2001

O presente trabalho é parte da dissertação de mestrado apresentada por Sérgio Ricardo Rotta, sob a orientação do Prof. Dr. Kechi Hirama: "Um Método de Teste de Software Baseado em Casos de Teste", defendida em 20/06/01, na EPUSP.

A íntegra da dissertação encontra-se à disposição com o autor e na Biblioteca de Engenharia Elétrica da Escola Politécnica da USP.

FICHA CATALOGRÁFICA

Rotta, Sérgio Ricardo
Um método de teste de software baseado em casos de teste / S.R. Rotta, K. Hirama. – São Paulo : EPUSP, 2001.
p. – (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia de Computação e Sistemas Digitais, BT/PCS/0124)

1. Teste e avaliação de programas 2. Softwares – Qualidade I. Hirama, Kechi II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais III. Título IV. Série
ISSN 1413-215X CDD 005.14
005.1

UM MÉTODO DE TESTE DE SOFTWARE BASEADO EM CASOS DE TESTE

Kechi Hirama
Departamento de Engenharia de Computação e Sistema Digitais
Escola Politécnica da Universidade de São Paulo
CEP 05508-900 – São Paulo – SP
(11) 3818-5394 kechi.hirama@poli.usp.br

Sérgio Ricardo Rotta
Superintendência Técnica de Automação Banco Eletrônico
Banco Itaú S.A.
CEP 03105-000 – São Paulo – SP
(11) 3274-9747 sergio.rota@itau.com.br

RESUMO

Este documento propõe e detalha um método simples e prático para teste de software que seja capaz de aumentar a qualidade de um software e que não necessite de muito esforço para ser elaborado e colocado em execução.

ABSTRACT

This paper proposes and details a simple and practical software testing method that intends to be capable to magnify the software quality and that does not need too much effort to be elaborated and to be placed in execution

1 INTRODUÇÃO

Em relação ao teste de software, a realidade encontrada na maioria das empresas constituintes da indústria nacional de software caracteriza-se por atividades de teste limitadas a uma única fase, abreviadas nos últimos estágios de desenvolvimento e executadas sem a alocação de recursos suficientes para a realização das mesmas.

Esta situação tem causado, invariavelmente, graves conseqüências, desde atrasos na entrega do software até um estouro no orçamento devido às correções que devem ser executadas.

Dentro deste panorama cria-se um círculo vicioso no qual as atividades de teste não têm a sua importância reconhecida, no tocante à produção de softwares confiáveis a custos razoáveis, pois muitas empresas não têm consciência dos custos elevados envolvidos no processo de correção de erros após a implantação do software e vêem as atividades de teste como impraticáveis – excessivos recursos (tempo, custos e esforços) envolvidos para a aplicação das mesmas.

Apresentar a aplicabilidade de técnicas de teste de software na elaboração e execução de casos de teste dentro de um método que pode ser aplicado, capaz de controlar melhor o tempo e o orçamento necessários para o desenvolvimento de software e que seja capaz de conferir um certo grau de qualidade ao software desenvolvido – desmistificando o cenário descrito anteriormente – é a principal motivação para o presente trabalho.

Estatísticas revelam que aproximadamente 50% do tempo e mais de 50% do custo total da área de desenvolvimento de software são gastos no teste de programas ou sistemas em desenvolvimento [5] e [11].

O método aqui proposto tem como objetivo ser uma ferramenta:

- **Útil**: Capaz de contribuir para ter-se baixos índices de falha em campo, aumentando a qualidade do software desenvolvido, e capaz de aumentar a produtividade, uma vez que organiza a execução das atividades de teste;

- **Simples**: Não requer pessoal capacitado em técnicas de teste de software e apresenta fácil assimilação, uma vez que apenas formaliza e confere uma fundamentação teórica às atividades de teste executadas de maneira intuitiva por grande parte das equipes de desenvolvimento de software;

- **Prática**: Se bem projetados, os casos de teste podem ser executados com sucesso por qualquer membro da equipe de desenvolvimento – ou até por recursos terceiros – abreviando o tempo para conclusão do projeto;

- **Flexível**: Por basear-se no projeto e execução de casos de teste e posterior análise dos resultados obtidos, o método de teste pode ser aplicado tanto pela própria equipe de desenvolvimento quanto por um grupo independente de teste; e

- **Gerencial**: Através da análise dos resultados, pode-se determinar o perfil da equipe de desenvolvimento: quais são os erros mais comuns, quais módulos apresentam mais erros, etc.

2 MÉTODO DE TESTE DE SOFTWARE

Além de ser praticamente impossível testar todos os caminhos de execução e todas as condições lógicas de um programa [1], tal teste não representa uma garantia segura de detecção de todos os erros presentes em um software [6]. Desta forma, os casos de teste têm como meta garantir que as funcionalidades especificadas para o software desenvolvido foram corretamente implementadas [7], atenuando, em parte, a impossibilidade prática descrita anteriormente, pois se validando os pontos fundamentais dos algoritmos, das funções, das estruturas de dados e controle internas, etc de um programa, atenuam-se, pelo menos, a probabilidade dos caminhos básicos e das condições lógicas principais de um determinado programa apresentarem erros.

O objetivo é projetar casos de teste à luz das técnicas de teste de caixa branca – testes que se concentram na estrutura de controle do programa, derivando casos de teste para garantir que todas as instruções do programa sejam exercitadas pelo menos uma vez durante os testes e que todas as condições lógicas sejam exercitadas para valores verdadeiro e falso – e das técnicas de teste de caixa preta – testes projetados para validar os requisitos funcionais, sem considerar o funcionamento interno de um programa – capazes de testar o máximo possível o software desenvolvido para que, a partir de entradas consistentes – valores normais, pertencentes ao domínio de entrada para o qual o software foi projetado – o software produza as saídas desejadas e a partir de entradas inconsistentes – valores anormais, não pertencentes ao domínio de entrada para o qual o software foi projetado – este mesmo software

produza as saídas adequadas (assegurar que o software não degrade o ambiente sob o qual opera), aumentando a confiança de que o software funcione corretamente.

O objetivo descrito anteriormente, pretendido para o método de teste de software aqui proposto, apóia-se no fato de que casos de teste capazes de detectar erros simples também são capazes de detectar erros mais complexos [3].

Supondo que um determinado erro reportado por um dos usuários de um determinado software esteja realmente ocorrendo, a causa para este erro não ter sido detectado na fase de testes pode ser uma das seguintes [17]:

- **O usuário executou um trecho do código não testado**: Devido ao constante atraso experimentado pelos projetos de software, não é incomum que a equipe de desenvolvimento libere código não testado ou, pelo menos, não testado adequadamente; fazendo com que erros apareçam em campo;

- **A ordem na qual as operações foram executadas é diferente da ordem executada nos testes**: Uma determinada ordem de execução pode determinar se o software falha ou não;

- **O usuário aplicou uma combinação de valores de entrada não testados**: O número de combinações de entrada possíveis para um determinado software pode ser de tal magnitude que testar todas estas combinações é impraticável. Freqüentemente, a equipe de desenvolvimento, baseada em certas técnicas de teste, toma certas decisões sobre qual conjunto de testes deve ser utilizado. Nem sempre estas decisões são as mais corretas; e

- **O ambiente operacional do usuário nunca foi testado**: Freqüentemente não é possível, para a equipe de desenvolvimento, replicar o ambiente operacional do usuário, o qual pode ser constituído de vários equipamentos, periféricos e aplicativos. Além disto, dificilmente consegue-se estressar o software em laboratório da mesma maneira que o ambiente operacional do usuário o fará.

Na prática, muitas vezes depara-se com uma ou mais destas situações.

2.1 Idéia Central

Dentro do ciclo de vida de desenvolvimento de software a fase de teste constitui-se na última revisão dos requisitos, do projeto e da codificação; é nesta fase, principalmente, que erros – erros de requisitos (discrepância entre a especificação de requisitos e as funcionalidades necessárias), erros de projeto (discrepância entre a estruturação da solução e a especificação de requisitos) e erros de codificação (discrepância entre o código e o projeto) [4] – podem ser detectados e removidos.

A idéia central do método de teste de software proposto por este documento é testar software a partir da execução de casos de teste projetados à luz dos fundamentos da teoria de teste de software e formalmente documentados. Ressalta-se que este método de teste atende, em termos de estrutura, às recomendações do documento "IEEE Standard 829".

A atividade de documentação dos casos de teste a serem aplicados traz os seguintes benefícios ao processo de desenvolvimento de software:

- **Fornecer uma nova visão do sistema**: Projetar os casos de teste é uma grande oportunidade de se avaliar a especificação do software por um outro ângulo. O projeto dos casos de teste fornece uma outra relação de especificações funcionais. Durante esta atividade, várias condições ainda não consideradas podem surgir, servindo como estímulo para uma revisão das especificações. Erros revelados por esta atividade são difíceis de serem detectados e, provavelmente, iriam ser detectados apenas quando o software já estivesse em operação, consumindo uma grande quantidade de tempo e dinheiro para serem corrigidos. De maneira geral, 10% dos erros de um software podem ser detectados em decorrência do projeto de casos de teste [18];

- **Repetir cenários de teste**: Uma vez que os casos de teste estejam documentados, eles podem ser facilmente e precisamente repetidos. A repetição precisa dos casos de teste permite que os erros sejam reproduzidos, o que ajuda a assegurar que as eventuais correções tenham sido efetivamente implementadas [10];

- **Facilitar a execução dos testes de regressão**: Após terem sido testadas todas as funcionalidades, os testes podem ser integralmente repetidos para assegurar que uma determinada correção não tenha ocasionado um erro em uma outra funcionalidade ou introduzido um novo erro [13] e [17];

- **Reduzir o prazo de desenvolvimento**: Se os casos de teste especificam as condições ambientais para realização dos testes, o conjunto de dados de entrada, as ações a serem executadas e as saídas esperadas; uma outra pessoa que não esteja envolvida com o projeto pode perfeitamente executar os testes, o que pode ser uma grande contribuição no sentido de reduzir o prazo de desenvolvimento [10]. Assim sendo, se os casos de teste são corretamente documentados, o pessoal alocado para auxiliar no desenvolvimento do software proporciona uma contribuição eficaz, uma vez que tudo que eles têm a fazer é executar os casos de teste de acordo com as especificações destes;

- **Verificar a qualidade dos casos de teste**: Uma vez que os casos de teste estejam documentados, pode-se verificar se todas as funcionalidades do software foram contempladas por estes e se eles estão bem distribuídos entre testes de unidade, de integração e de validação, testes de resposta normal, testes de resposta anormal, testes de valores limite, testes de estresse, testes de desempenho, testes de recuperação e testes de segurança [14] e [18];

- **Permitir avaliar previamente a qualidade do software**: Se os casos de teste são corretamente projetados, pode-se ter indícios da qualidade do software sob teste durante a execução dos casos de teste. Se após a execução de 50 casos de teste, de um total de 200 casos de teste, 4 erros tiverem sido detectados, é de se esperar, utilizando-se uma estimativa grosseira, que o software ainda apresente 16 erros [18]; e

- **Auxiliar na validação das manutenções efetuadas**: Os casos de teste atuam como uma espécie de ligação entre as funcionalidades implementadas pelo software e

os trechos de código que efetivamente as implementam. Assim sendo, pode-se identificar os casos de teste que devem ser reaplicados a fim de se validar as manutenções efetuadas [16].

2.2 Fases e Atividades

Para chegar-se a elaboração dos casos de teste – para posteriormente executá-los – o método de teste compreende cinco fases: Planejamento, Projeto, Preparação da Infra-Estrutura, Execução e Análise.

A Figura 1 exibe as atividades que compõem cada uma das fases do método de teste.

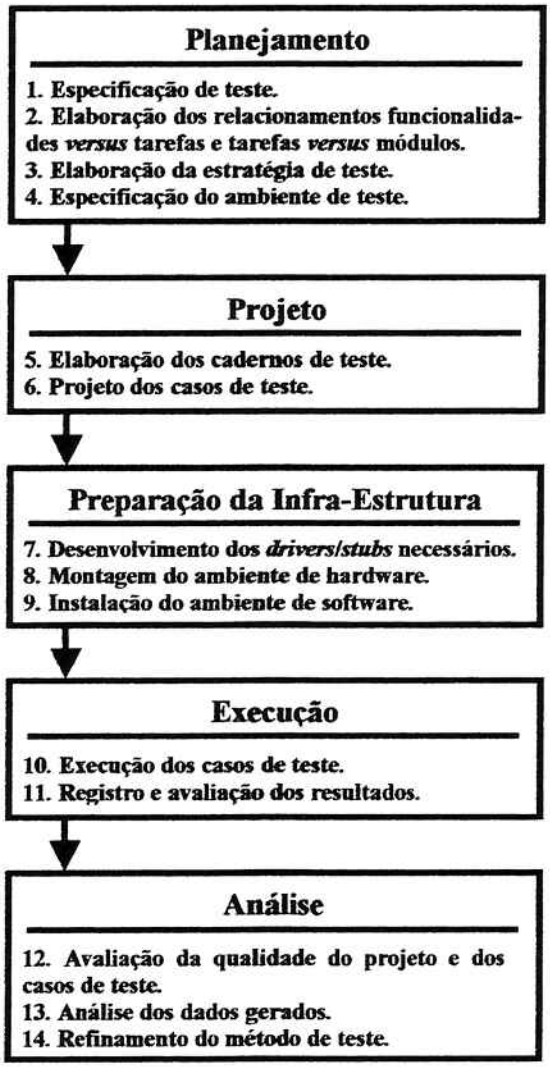

Figura 1 – Atividades das fases do método de teste.

2.2.1 Planejamento e Projeto

As atividades primordiais do método de teste são aquelas compreendidas pelas fases Planejamento – descrição dos testes a serem realizados e das funcionalidades a serem testadas, da abordagem utilizada, dos recursos necessários e dos critérios de aprovação a serem adotados – e Projeto – detalhamento dos testes a serem realizados e detalhamento da abordagem utilizada para a realização dos mesmos.

As atividades das fases de Planejamento e Projeto do método de teste podem ser descritas da seguinte forma:

• **Especificação de teste**: Esta atividade compreende a reunião das informações contidas no documento gerado na fase de levantamento e validação dos requisitos a serem atendidos pelo sistema com as informações contidas nas especificações de projeto. O produto gerado por esta atividade é uma Especificação de Teste, documento que descreve as funcionalidades e as tarefas a serem testadas e contém o cronograma de teste a ser seguido;

• **Elaboração dos relacionamentos funcionalidades *versus* tarefas e tarefas *versus* módulos**: À medida que a arquitetura do sistema vai tomando forma, as funcionalidades e as tarefas a serem atendidas vão sendo distribuídas entre os vários módulos constituintes do sistema. Através do relacionamento entre funcionalidades *versus* tarefas e tarefas *versus* módulos pode-se estabelecer os critérios de aceitação do sistema, isto é, o comportamento esperado para cada um dos módulos. Além disto, os componentes de hardware e software necessários para suportar o sistema são definidos. Desta forma pode-se gerar um documento que se constitui em um Critério de Aceitação do Sistema;

• **Elaboração da estratégia de teste**: Uma vez que os módulos e suas respectivas interfaces são definidos, pode-se traçar a estratégia de teste a ser adotada, isto é, a ordem na qual as funcionalidades e as tarefas serão testadas e especificar os *drivers/stubs* necessários para a execução desta estratégia. Esta atividade gera como produtos a Estratégia de Teste e a Especificação de *Drivers/Stubs*;

• **Especificação do ambiente de teste**: Esta atividade está diretamente relacionada às duas atividades descritas anteriormente uma vez que o ambiente de teste constitui-se, no mínimo, na infra-estrutura de hardware e software necessária para suportar o sistema. A fim de maximizar a eficiência dos testes, o ambiente de teste deve se aproximar ao máximo possível do ambiente operacional no qual o sistema será utilizado [14] e [17]. Esta atividade gera como produto o documento Especificação do Ambiente de Teste;

• **Elaboração dos cadernos de teste**: Nesta atividade são gerados os Cadernos de Teste, documentos contendo informações como: módulo/tarefa/sistema a ser testado, ambiente de teste, plataforma de hardware e software para realização dos testes e funcionalidades a serem testadas; e

• **Projeto dos casos de teste**: Nesta atividade, os casos de teste são projetados, sendo que estes devem contemplar: testes de unidade, de integração e de sistema; testes de resposta normal (condições normais); testes de resposta anormal (condições anormais); testes de valores limite, testes de estresse, testes de desempenho, testes de recuperação e testes de segurança. Os documentos gerados por esta atividade são os próprios Casos de Teste.

2.2.2 Preparação da Infra-Estrutura

As atividades da fase de Preparação da Infra-Estrutura do método de teste estabelecem as plataformas

de hardware e software sob a qual os testes serão executados e podem ser descritas da seguinte forma:

• **Desenvolvimento dos *drivers/stubs* necessários**: Nesta atividade os *drivers/stubs* necessários para a implantação da estratégia de teste traçada são desenvolvidos;

• **Montagem do ambiente de hardware**: Esta atividade consiste em montar o ambiente de hardware – computadores, impressoras, rede TCP/IP, cabos de comunicação, etc – necessários para suportar o sistema a ser testado. Concomitantemente à montagem do ambiente de hardware, gera-se uma versão preliminar do Manual de Instalação do Sistema contendo informações como relação de equipamentos constituintes do sistema, configurações mínimas de equipamentos, topologia de rede, etc; e

• **Instalação do ambiente de software**: Esta atividade consiste em instalar o ambiente de software – sistema operacional, atualizações de software, servidores de banco de dados, protocolos de comunicação, softwares de interface, etc – necessário para suportar o sistema a ser testado. Nesta atividade, a versão preliminar do Manual de Instalação do Sistema é complementada com informações como: procedimentos para instalação de componentes de software, definição das tabelas do banco de dados, localização de arquivos, etc.

2.2.3 Execução

As atividades da fase de Execução do método de teste estão intimamente relacionadas entre si, sendo que não há uma delimitação claramente estabelecida entre elas. Estas atividades podem ser descritas da seguinte forma:

• **Execução dos casos de teste**: O âmago do método nesta fase consiste em executar os casos de teste. Além disto, para permitir que projetos futuros sejam planejados com base em estimativas mais precisas, deve-se registrar quanto tempo foi necessário para a execução de cada caso de teste;

• **Registro e avaliação dos resultados**: O resultado obtido – sucesso, erro ou observação – na execução de cada caso de teste deve ser registrado, sobretudo quando um caso de teste revela um erro. Para estes casos deve-se registrar em que condições este erro é revelado e qual o sintoma do erro. Sem estas informações pode-se repetir o teste e erroneamente acreditar que este erro já tenha sido removido até que ele reapareça. Além disto, muitas vezes, um erro pode ser provocado por um padrão de lógica errôneo que pode estar reproduzido em outras partes do código e, assim sendo, uma análise cuidadosa deste padrão de lógica pode resultar na descoberta de outros erros [13]; e

• **Coleta de dados durante a execução dos casos de teste**: Para maximizar a utilidade dos cadernos de teste deve-se manter um registro de todos os erros detectados, seus sintomas, quão críticos são e como eles foram corrigidos. Todas estas informações são úteis, principalmente as informações referentes à remoção de erros, pois freqüentemente um erro pode ser corrigido através de uma re-configuração do ambiente operacional, sendo que estas informações devem constar do documento de configuração do sistema.

2.2.4 Análise

As atividades da fase de Análise do método de teste podem ser vistas como uma ferramenta gerencial capaz de avaliar a eficiência do processo de desenvolvimento adotado e podem ser descritas da seguinte forma:

• **Avaliação da qualidade do projeto e dos casos de teste**: Avaliando-se a quantidade de erros detectados e a velocidade de execução dos casos de teste pode-se elaborar planos de ação caso sejam necessárias algumas correções de rumo. Além disto, esta avaliação mostra a produtividade da equipe de desenvolvimento na fase de teste;

• **Análise dos dados gerados**: A análise dos dados colhidos durante a execução dos casos de teste permite avaliar a eficiência do processo de desenvolvimento utilizado, a adequação dos casos de teste, a habilidade da equipe de desenvolvimento na execução do processo de teste do software, quais são os erros mais comuns cometidos por esta equipe de desenvolvimento e quais são as tarefas e os módulos do sistema mais propensos a erros; e

• **Refinamento do método de teste**: A partir da análise dos dados gerados, o método de teste de software pode ser melhorado. Casos de teste mais adequados podem ser projetados, isto é, casos de teste focados em verificar se os erros mais comuns cometidos pela equipe de desenvolvimento estão ou não presentes em um determinado sistema. Da mesma forma, o processo através do qual os casos de teste são projetados e executados pode ser otimizado para que a fase de teste possa ser executada de maneira mais eficiente, consumindo menos recursos (tempo, custos e esforços).

A análise dos dados registrados nos cadernos de teste pode, e deve, ser feita no dia a dia da fase de teste do software e após o projeto ter sido concluído.

Os objetivos da análise feita diariamente durante a fase de teste do software são determinar a produtividade da equipe de desenvolvimento durante esta fase, a adequação dos casos de teste projetados e determinar se há ou não necessidade de se suspender a execução dos testes e revisar imediatamente o projeto e a codificação do software sob teste. Desta forma, a análise diária dos dados registrados nos cadernos de teste auxilia a tomada de decisão por parte da gerência e da equipe de desenvolvimento sob a condução do projeto e para a elaboração de planos de ação, caso sejam necessárias algumas correções de rumo [1].

Uma vez que o projeto tenha sido concluído, a análise dos dados registrados nos cadernos de teste fornece informações como: quais são os erros mais comuns cometidos pela equipe de desenvolvimento (má utilização de memória, deixar arquivos abertos, etc), quais são as tarefas e os módulos do sistema mais propensos a erros e quão eficaz é o processo de desenvolvimento adotado.

A fim de se verificar a eficiência do método de teste adotado, do ponto de vista de detecção de erros, deve-se manter um registro dos erros encontrados em campo após a liberação do software. Tal eficiência pode ser obtida através da aplicação da métrica eficiência de remoção de defeitos que representa a porcentagem de

erros detectados pelas práticas de teste adotadas em relação à quantidade total de erros apresentados por um determinado software [8].

Por exemplo, se durante a fase de teste foram detectados 18 erros e após a liberação do software foram detectados 2 erros em campo, aplicando-se a métrica eficiência de remoção de defeitos tem-se:

Erros detectados na fase de teste:	18
Erros detectados em campo:	2
Total de erros detectados:	20
Eficiência de remoção de defeitos:	18/20 = 0,9 (90%)

Além disto, o registro dos erros detectados em campo contribui para o refinamento do método de teste, uma vez que a análise destes erros fornece subsídios para identificar porquê os mesmos não foram detectados pelas atividades de teste e implementar ao processo de teste os ajustes que se fizerem necessários para que a qualidade deste seja melhorada [1].

2.3 Casos de Teste

Os Casos de Teste descrevem todas as informações pertinentes a execução de um determinado teste e contém as seguintes informações:

- Módulo/Tarefa/Sistema alvo do caso de teste;
- Identificação do caso de teste;
- Tipo de teste: teste de unidade, de integração e de validação; teste de resposta normal (condições normais), teste de resposta anormal (condições anormais), teste de valores limite, teste de estresse, teste de desempenho, teste de recuperação e teste de segurança;
- Dependência entre casos de teste;
- Condições ambientais para realização dos testes;
- *Drivers* e *stubs* necessários para a realização dos testes;
- Conjunto de dados de entrada;
- Ações a serem executadas;
- Saídas esperadas; e
- Responsável(eis) pelo projeto do caso de teste.

CASO DE TESTE	
( ) Módulo: **( ) Tarefa:** **( ) Sistema:**	
Caso de Teste:	
( ) teste de unidade **( ) teste de integração** **( ) teste de validação**	**( ) teste de resposta normal** **( ) teste de resposta anormal** **( ) teste de valores limite** **( ) teste de estresse** **( ) teste de desempenho** **( ) teste de recuperação** **( ) teste de segurança**
Dependência:	
Condições Ambientais:	
***Drivers*:** ***Stubs*:**	
Conjunto de Dados de Entrada:	
Ações:	
Saídas Esperadas:	
Responsável(eis):	

Figura 2 – Exemplo de formato de Caso de Teste.

2.4 Regras para o Projeto de Casos de Teste

Uma das regras básicas para o projeto de casos de teste é a seguinte: projetar pelo menos um caso de teste para cada uma das funcionalidades que o software deva apresentar, sendo que este conjunto de funcionalidades compreende os requisitos especificados pelo cliente, as características implícitas que todo software profissional de qualidade deve apresentar e os aspectos de projeto do mesmo.

Uma abordagem prática é relacionar todas estas funcionalidades e considerar todas as condições em que uma determinada funcionalidade deve operar. Este processo acaba constituindo-se em uma revisão do projeto do software, uma vez que ele pode revelar condições não consideradas anteriormente e, portanto, não implementadas.

Tal abordagem acaba atuando como uma espécie de ligação entre uma determinada funcionalidade e o trecho de código que a implementou, simplificando a execução dos testes de regressão quando manutenções no software se fizerem necessárias [16].

A aplicação da técnica de teste particionamento de equivalência [9] é muito útil no projeto dos casos de teste, pois permite selecionar um determinado dado de entrada como representante de um domínio. Por exemplo, supondo-se que uma aplicação de Unidade de Resposta Automática (ARU, do inglês *Automatic Response Unit*) deva tocar uma mensagem de saudação diferente dependendo do horário, isto é: "Bom dia!" das 06:00h às 11:59h, "Boa tarde!" das 12:00h às 17:59h, "Boa noite!" das 18:00h às 23:59h e "Sistema inoperante!" das 00:00h às 05:59h. Para um sistema assim, pode-se ter os seguintes casos de teste: 08:00h para o período compreendido das 06:00h às 11:59h, 16:00h para o período compreendido das 12:00h às 17:59h, 21:00h para o período compreendido das 18:00 h às 23:59h e 05:00h para o período compreendido das 00:00h às 05:59h – note-se que o caso de teste 08:00h representa uma classe de equivalência válida para o período compreendido das 06:00h às 11:59h e os casos de teste 16:00h, 21:00h e 05:00h representam classes de equivalência inválidas para o mesmo período.

Outra técnica de teste muito útil no projeto dos casos de teste é a técnica de análise de valor limite [9], dada uma determinada condição de entrada que especifica um intervalo, casos de teste devem ser projetados com os valores mínimo e máximo e com valores imediatamente abaixo e imediatamente acima dos valores mínimo e máximo, respectivamente. Para o exemplo descrito anteriormente, para o período compreendido das 06:00h às 11:59h deve-se ter os seguintes valores de teste: 05:59h, 06:00h, 06:01h, 11:58h, 11:59h e 12:00h.

Os casos de teste a serem projetados também devem prever situações que fogem do controle do software sob teste, tais como: uma aplicação tentar abrir um arquivo que está sendo utilizado por outra aplicação, um dispositivo físico ser reiniciado durante a comunicação com o software, duas aplicações solicitarem simultaneamente um mesmo serviço para uma determinada API, etc.

2.5 Critérios para o Projeto de Casos de Teste

Pode-se aplicar os seguintes critérios para o projeto de casos de teste [18]:

- Um caso de teste para cada 10 LOC [18];
- Um caso de teste para cada condição lógica (*IF*) [18];
- Casos de teste com iteração 0, 1, 2, número máximo – 1, número máximo e número máximo + 1 para teste de laços [2];
- Casos de teste focados na validação do desempenho de execução do software: utilização dos recursos computacionais (capacidade de processamento, operações de acesso ao disco, consumo de espaço em disco, utilização de memória, etc), concorrência no acesso a bancos de dados, re-entrância em rotinas de tratamento de interrupção e de eventos, gerenciamento de filas de mensagens, tempos de resposta do sistema, capacidade de vazão do sistema (*throughput*), etc [15]; e
- Os casos de teste devem apresentar a seguinte distribuição percentual: 60% de casos de teste de resposta normal (incluindo-se a este percentual os casos de teste de segurança), 15% de casos de teste de resposta anormal (incluindo-se a este percentual os casos de teste de recuperação), 10% de casos de teste de valor limite e 15% de casos de teste de estresse (incluindo-se a este percentual os casos de teste de desempenho) [10].

2.6 Considerações sobre o Projeto de Casos de Teste

Orientar-se por projetos similares ao projeto em questão é uma boa prática, além de se poder obter casos de teste já prontos, eles podem fornecer algumas informações sobre o quê testar.

Observar que um determinado profissional tende a projetar muitos casos de teste para validar as funcionalidades que ele conhece plenamente e poucos casos de teste para funções que ele não conhece muito bem [12] e [18]. Deve-se comparar o número de casos de teste com o número de LOCs ou com o número de condições lógicas (*IFs*) de um software para verificar se o número de casos de teste projetados é coerente.

2.7 Verificação dos Casos de Teste

Após os casos de teste terem sido projetados, estes devem ser verificados quanto aos seguintes aspectos [14], [16], [17] e [18]:

- Os casos de teste cobrem todas as funcionalidades?
- Há uma boa distribuição entre casos de teste de resposta normal, resposta anormal, valores limite, estresse, desempenho, recuperação e segurança?
- Há uma boa distribuição entre casos de teste estruturais (testes de caixa branca) e casos de teste funcionais (testes de caixa preta)?
- Há uma boa distribuição entre casos de teste operacionais e casos de teste de performance?

2.8 Relação de Produtos Gerados

A Tabela 1 sintetiza os produtos gerados pelas atividades do método de teste.

Tabela 1 – Produtos gerados pelas fases do método de teste.

Fase	Produto Gerado
Planejamento	• Especificação de Testes; • Critério de Aceitação do Sistema; • Estratégia de Teste; • Especificações de *Drivers/Stubs*; e • Especificação do Ambiente de Teste.
Projeto	• Cadernos de Teste; e • Casos de Teste.
Preparação da Infra-Estrutura	• *Drivers/Stubs* necessários desenvolvidos; • Ambiente de Hardware montado; e • Ambiente de Software instalado.
Execução	• Teste do Sistema; • Preenchimento dos Cadernos de Teste; e • Módulo/Tarefa liberado para produção.
Análise	• Adequação dos Casos de Teste; • Levantamento do perfil da equipe de desenvolvimento; • Identificação dos erros mais comuns que são cometidos; • Identificação dos módulos mais propensos a erros; e • Otimização do Processo de Teste.

3 ESTUDOS DE CASO

Como aplicação do método de teste são analisados dois estudos de casos. O primeiro estudo de caso analisa os dados referentes ao teste de um software de interface para o sistema operacional Microsoft Windows (Windows 3.1 e Windows 95/98) que se caracteriza por operar de maneira isolada, interagindo apenas com o sistema operacional, cujo objetivo é detectar teclas ou combinações de teclas – denominadas teclas quentes – em aplicações Windows e em janelas DOS. Já o segundo estudo de caso analisa os dados referentes ao teste de um sistema de atendimento a clientes constituído por quatro tarefas com alto grau de interação entre si.

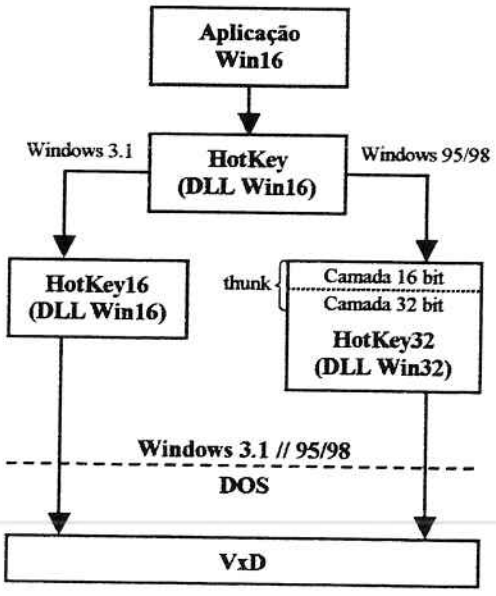

Figura 3 – Diagrama do Software de Interface.

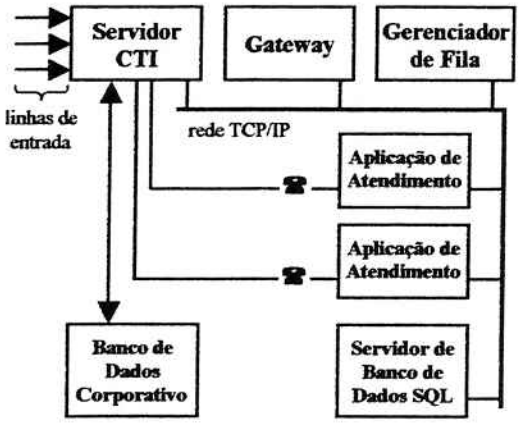

Figura 4 – Diagrama do Sistema de Atendimento a Clientes.

A Tabela 2 apresenta uma comparação entre os dados referentes aos dois estudos de caso apresentados e os dados previstos pelos critérios adotados pelo método de teste para projeto de casos de teste e uma comparação entre o tempo gasto em atividades de desenvolvimento – levantamento e validação dos requisitos, projeto e codificação – e o tempo gasto em atividades de teste para os dois estudos de caso.

Tabela 2 – Quantidade e distribuição dos casos de teste aplicados.

Item	Previsto	Estudo de Caso I	Estudo de Caso II
Linhas de Código (LOC)	—	1614	2597
Condições Lógicas (*IFs*)	—	153	251
Casos de Teste Aplicados	—	136	225
Casos de Teste / LOC	10	11,9	11,5
Casos de Teste / *Ifs*	1	1,1	1,1
Teste de Resposta Normal + Teste de Segurança	60%	59,6%	58,2%
Teste de Resposta Anormal + Teste de Recuperação	15%	27,9%	23,6%
Teste de Valores Limite	10%	5,9%	12,4%
Teste de Estresse + Teste de Desempenho	15%	6,6%	5,8%
Velocidade de Execução	—	4 casos/h	3 casos/h
Atividades de Desenvolvimento	—	88 h	176 h
Atividades de Teste	—	118 h (34,1%)	204 h (15,9%)
Planejamento	—	18 h	24 h
Projeto	—	24 h	36 h
Infra-Estrutura	—	44 h	68 h
Execução	—	32 h	76 h

De acordo com os dados apresentados na Tabela 2, 1 caso de teste para cada 10 LOCs ou 1 caso de teste para cada condição lógica do programa (*IF*) parecem ser critérios adequados para a previsão da quantidade de casos de teste a ser projetada, uma vez que a quantidade de casos de teste projetada para os dois estudos de caso apresentados se aproxima muito dos valores obtidos mediante a aplicação destes critérios; sendo que o número de condições lógicas de um programa (*IFs*) parece ser um critério mais preciso, uma vez que está diretamente relacionado à quantidade de caminhos executáveis de um programa.

Em relação à distribuição dos casos de teste projetados, como os próprios dados apresentados na Tabela 2 mostram e devido à variabilidade de características apresentada por sistemas de software distintos, uma previsão baseada em uma faixa percentual de casos de teste parece ser a mais adequada.

Ainda de acordo com os dados apresentados na Tabela 2, verifica-se que para o estudo de caso software de interface o tempo gasto em atividades de teste é 34,1% superior ao tempo gasto em atividades de desenvolvimento e para o estudo de caso sistema de atendimento a clientes o tempo gasto em atividades de teste é 15,9% superior ao tempo gasto em atividades de desenvolvimento. Tais dados são significativamente superiores aos dados esperados, uma vez que estatísticas revelam que aproximadamente 50% do tempo total da área de desenvolvimento de software são gastos no teste de programas ou sistemas em desenvolvimento [5] e [11].

4 CONCLUSÕES

4.1 Resultados Importantes

Em relação à aplicação do método de teste, destacam-se os seguintes resultados: benefícios advindos da documentação dos casos de teste, eficiência e aplicabilidade do método de teste e capacidade do método de teste em identificar quais são os erros mais comuns cometidos pela equipe de desenvolvimento.

A documentação dos casos de teste a serem aplicados traz os seguintes benefícios ao processo de desenvolvimento de software:

- **Auxiliar nos testes do software em campo**: Quando da instalação do software em campo, os casos de teste fornecem uma maneira sistemática de se verificar se as funcionalidades especificadas foram implementadas corretamente;

- **Reutilizar em projetos similares**: Uma boa parte dos casos de teste podem ser integralmente reutilizados em projetos similares e outros tantos podem ser facilmente adaptados; e

- **Auxiliar no refinamento do processo de teste**: Uma vez que os casos de teste estejam documentados, pode-se identificar porquê um determinado erro conseguiu passar desapercebido pela atividade de teste e implementar ao processo de projeto de casos de teste os ajustes que se fizerem necessários para que a qualidade destes seja melhorada.

Aplicando-se a métrica de eficiência de remoção de defeitos [8] aos dados apresentados pelos dois estudos de caso, tem-se uma eficiência média de 90% (observando-se que esta eficiência pode estar mascarada devido ao fato de terem sido analisados apenas dois estudos de caso) e, portanto, um índice de falhas em campo de 10%, sendo que este percentual de falhas em campo deve-se a uma deficiência claramente identificada no método de teste, testes de desempenho e testes de estresse inadequados, ou seja, há uma boa probabilidade de redução deste percentual caso elimine-se esta deficiência.

Quanto à aplicação do método de teste, destacam-se as seguintes características:

• **Simplicidade**: Não há necessidade de ter-se pessoal capacitado em técnicas de teste de software e nem de treinamento intensivo, uma vez que o método de teste apresenta fácil assimilação;

• **Praticidade**: Quando bem projetados, os casos de teste podem ser executados com sucesso por qualquer membro da equipe de desenvolvimento – ou até por recursos terceiros – abreviando o tempo para conclusão do projeto; e

• **Adequação**: Freqüentemente a mesma equipe que desenvolve o software é a equipe que vai testá-lo. Assim sendo, muitas vezes os testes são executados apenas com o objetivo de mostrar que o software funciona corretamente e não com o objetivo de detectar erros de maneira rigorosa e sistemática. O método de teste procura quebrar este paradigma mediante a sistemática proposta para o projeto e execução dos casos de teste.

4.2 Dificuldades Encontradas na Aplicação do Método de Teste

Entre as dificuldades encontradas na aplicação do método de teste, destacam-se:

• (1) Conforme previsto, realmente observa-se que um determinado profissional tende a projetar muitos casos de teste para validar as funcionalidades que ele conhece plenamente e poucos casos de teste para funções que ele não conhece muito bem;
• (2) Ausência de critérios para a execução dos testes de regressão; e
• (3) Dificuldade de se realizar testes de desempenho e testes de estresse adequados.

Quanto à dificuldade (1), deve-se procurar distribuir o mais homogeneamente possível o projeto de casos de teste entre os diversos membros da equipe de desenvolvimento, uma vez que é natural que um profissional conheça de maneira mais profunda as funcionalidades cuja implementação estão sob sua responsabilidade.

Quanto à dificuldade (2), esta é uma questão complexa. O processo de correção de um determinado erro detectado pelos testes pode conduzir a quatro situações distintas: o erro detectado é corrigido; o erro detectado não é corrigido; o erro detectado é corrigido, mas introduz-se um novo erro; e o erro detectado não é corrigido e introduz-se um novo erro.

Dadas estas possibilidades, alguns autores defendem a tese de se executar novamente todos os testes aplicados [17], solução esta que apresenta um custo elevado.

Uma medida prática que pode ser adotada é relacionar todas as funcionalidades que podem ser afetadas pela correção de um determinado erro, identificar os casos de teste projetados para validar estas funcionalidades e executá-los novamente (esta foi a solução adotada nos dois estudos de caso apresentados).

Quanto à questão sobre os testes de desempenho e os testes de estresse (3), é imperiosa a adoção de ferramentas especiais para a realização destes testes – adquiridas no mercado ou desenvolvidas internamente. Os testes de desempenho combinados aos testes de estresse são realizados para confrontar o software com situações nas quais ele seja muito exigido e com situações anormais. Para tanto, muitas vezes é necessária a utilização de ferramentas de hardware e/ou de software para criar tais situações.

5 BIBLIOGRAFIA

[1] Bach, J. A Framework for Good Enough Testing. **Computer,** v.31, n.10, p.124–126, out. 1998.
[2] Beizer, B. **Software Testing Techniques.** New York, Van Nostrand Reinhold, 1983.
[3] Demillo, R. A.; Lipton, R. J.; Sayward, F. G. Hints on Test Data Selection: Help for the Practicing Programmer. **IEEE Computer,** v.11, n.4, p.34–41, abr. 1978.
[4] Goodenough, J. B.; Gerhart, S. L. Toward a Theory of Test Data Selection. **IEEE Transactions on Software Engineering,** v.1, n.2, p.156–173, jun. 1975.
[5] Hetzel, W. **The Complete Guide to Software Testing.** Wellesley, Massachusetts, QED Information Sciences, 1984.
[6] Howden, W. E. Reliability of the Path Analysis Testing Strategy. **IEEE Transactions on Software Engineering,** v.2, n.3, p.208–215, set. 1976.
[7] Howden, W. E. Effectiveness of Software Validation Methods. In: WESTLEY, A. E. **State of the Art Report, Software Testing,** Berkshire, Infotech International, v.2, p.129–146, 1979.
[8] Jones, C. Software Defect-Removal Efficiency. **IEEE Computer,** v.29, n.2, p.94–95, abr. 1996.
[9] Myers, G. J. **The Art of Software Testing.** New York, John Wiley & Sons, 1979.
[10] Onoma, A. K.; Yamamura, T. Practical Steps Toward Quality Development. **IEEE Software,** v.12, n.5, p.68–77, set. 1995.
[11] Rettig, M. Testing made Palatable. **Communications of the ACM,** v.34, n.5, p.25–29, maio 1991.
[12] Thomas, D. Program Testing – Helping Programmers to Help Themselves. In: MILLER, E.; HOWDEN, W. E. **Tutorial: Software Testing & Validation Techniques,** 2.ed New York, IEEE Computer Society Press, catálogo n.EHO 180-0, p.271–281, 1981.
[13] Van Vleck, T. Three Questions About Each Bug You Find. **ACM Software Engineering Notes,** v.14, n.5, p.62–63, jul. 1989.
[14] Voas, J. Certifying Software for High-Assurance Environments. **IEEE Software,** v.16, n.4, p.48–54, jul./ago. 1999.
[15] Vokolos, F. I.; Weyuker, E. J. Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study. **IEEE Transactions on Software Engineering,** v.26, n.12, p.1147–1156, dez. 2000.
[16] Weyuker, E. J. Testing Component – Based Software: A Cautionary Tale. **IEEE Software,** v.15, n.5, p.54–59, set./out. 1998.
[17] Whittaker, J. A. What Is Software Testing? And Why Is It So Hard? **IEEE Software,** v.17, n.1, p.70–77, jan./fev. 2000.
[18] Yamaura, T. How To Design Practical Test Cases. **IEEE Software,** v.15, n.6, p.30–36, nov./dez. 1998.

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PCS/9301 - Interligação de Processadores através de Chaves Ômicron - GERALDO LINO DE CAMPOS, DEMI GETSCHKO

BT/PCS/9302 - Implementação de Transparência em Sistema Distribuído - LUÍSA YUMIKO AKAO, JOÃO JOSÉ NETO

BT/PCS/9303 - Desenvolvimento de Sistemas Especificados em SDL - SIDNEI H. TANO, SELMA S. S. MELNIKOFF

BT/PCS/9304 - Um Modelo Formal para Sistemas Digitais à Nível de Transferência de Registradores - JOSÉ EDUARDO MOREIRA, WILSON VICENTE RUGGIERO

BT/PCS/9305 - Uma Ferramenta para o Desenvolvimento de Protótipos de Programas Concorrentes - JORGE KINOSHITA, JOÃO JOSÉ NETO

BT/PCS/9306 - Uma Ferramenta de Monitoração para um Núcleo de Resolução Distribuída de Problemas Orientado a Objetos - JAIME SIMÃO SICHMAN, ELERI CARDOSO

BT/PCS/9307 - Uma Análise das Técnicas Reversíveis de Compressão de Dados - MÁRIO CESAR GOMES SEGURA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9308 - Proposta de Rede Digital de Sistemas Integrados para Navio - CESAR DE ALVARENGA JACOBY, MOACYR MARTUCCI JR.

BT/PCS/9309 - Sistemas UNIX para Tempo Real - PAULO CESAR CORIGLIANO, JOÃO JOSÉ NETO

BT/PCS/9310 - Projeto de uma Unidade de Matching Store baseada em Memória Paginada para uma Máquina Fluxo de Dados Distribuído - EDUARDO MARQUES, CLAUDIO KIRNER

BT/PCS/9401 - Implementação de Arquiteturas Abertas: Uma Aplicação na Automação da Manufatura - JORGE LUIS RISCO BECERRA, MOACYR MARTUCCI JR.

BT/PCS/9402 - Modelamento Geométrico usando do Operadores Topológicos de Euler - GERALDO MACIEL DA FONSECA, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9403 - Segmentação de Imagens aplicada a Reconhecimento Automático de Alvos - LEONCIO CLARO DE BARROS NETO, ANTONIO MARCOS DE AGUIRRA MASSOLA

BT/PCS/9404 - Metodologia e Ambiente para Reutilização de Software Baseado em Composição - LEONARDO PUJATTI, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9405 - Desenvolvimento de uma Solução para a Supervisão e Integração de Células de Manufatura Discreta - JOSÉ BENEDITO DE ALMEIDA, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9406 - Método de Teste de Sincronização para Programas em ADA - EDUARDO T. MATSUDA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9407 - Um Compilador Paralelizante com Detecção de Paralelismo na Linguagem Intermediária - HSUEH TSUNG HSIANG, LÍRIA MATSUMOTO SAITO

BT/PCS/9408 - Modelamento de Sistemas com Redes de Petri Interpretadas - CARLOS ALBERTO SANGIORGIO, WILSON V. RUGGIERO

BT/PCS/9501 - Sintese de Voz com Qualidade - EVANDRO BACCI GOUVÊA, GERALDO LINO DE CAMPOS

BT/PCS/9502 - Um Simulador de Arquiteturas de Computadores "A Computer Architecture Simulator" - CLAUDIO A. PRADO, WILSON V. RUGGIERO

BT/PCS/9503 - Simulador para Avaliação da Confiabilidade de Sistemas Redundantes com Reparo - ANDRÉA LUCIA BRAGA, FRANCISCO JOSÉ DE OLIVEIRA DIAS

BT/PCS/9504 - Projeto Conceitual e Projeto Básico do Nível de Coordenação de um Sistema Aberto de Automação, Utilizando Conceitos de Orientação a Objetos - NELSON TANOMARU, MOACYR MARTUCCI JUNIOR

BT/PCS/9505 - Uma Experiência no Gerenciamento da Produção de Software - RICARDO LUIS DE AZEVEDO DA ROCHA, JOÃO JOSÉ NETO

BT/PCS/9506 - MétodOO - Método de Desenvolvimento de Sistemas Orientado a Objetos: Uma Abordagem Integrada à Análise Estruturada e Redes de Petri - KECHI HIRAMA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9601 - MOOPP: Uma Metodologia Orientada a Objetos para Desenvolvimento de Software para Processamento Paralelo - ELISA HATSUE MORIYA HUZITA, LÍRIA MATSUMOTO SATO

BT/PCS/9602 - Estudo do Espalhamento Brillouin Estimulado em Fibras Ópticas Monomodo - LUIS MEREGE SANCHES, CHARLES ARTUR SANTOS DE OLIVEIRA

BT/PCS/9603 - Programação Paralela com Variáveis Compartilhadas para Sistemas Distribuídos - LUCIANA BEZERRA ARANTES, LIRIA MATSUMOTO SATO

BT/PCS/9604 - Uma Metodologia de Projeto de Redes Locais - TEREZA CRISTINA MELO DE BRITO CARVALHO, WILSON VICENTE RUGGIERO

BT/PCS/9605 - Desenvolvimento de Sistema para Conversão de Textos em Fonemas no Idioma Português - DIMAS TREVIZAN CHBANE, GERALDO LINO DE CAMPOS

BT/PCS/9606 - Sincronização de Fluxos Multimídia em um Sistema de Videoconferência - EDUARDO S. C. TAKAHASHI, STEFANIA STIUBIENER

BT/PCS/9607 - A importância da Completeza na Especificação de Sistemas de Segurança - JOÃO BATISTA CAMARGO JÚNIOR, BENÍCIO JOSÉ DE SOUZA

BT/PCS/9608 - Uma Abordagem Paraconsistente Baseada em Lógica Evidencial para Tratar Exceções em Sistemas de Frames com Múltipla Herança - BRÁULIO COELHO ÁVILA, MÁRCIO RILLO

BT/PCS/9609 - Implementação de Engenharia Simultânea - MARCIO MOREIRA DA SILVA, MOACYR MARTUCCI JÚNIOR

BT/PCS/9610 - Statecharts Adaptativos - Um Exemplo de Aplicação do STAD - JORGE RADY DE ALMEIDA JUNIOR, JOÃO JOSÉ NETO

BT/PCS/9611 - Um Meta-Editor Dirigido por Sintaxe - MARGARETE KEIKO IWAI, JOÃO JOSÉ NETO

BT/PCS/9612 - Reutilização em Software Orientado a Objetos: Um Estudo Empírico para Analisar a Dificuldade de Localização e Entendimento de Classes - SELMA SHIN SHIMIZU MELNIKOFF, PEDRO ALEXANDRE DE OLIVEIRA GIOVANI

BT/PCS/9613 - Representação de Estruturas de Conhecimento em Sistemas de Banco de Dados - JUDITH PAVÓN MENDONZA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9701 - Uma Experiência na Construção de um Tradutor Inglês - Português - JORGE KINOSHITA, JOÃO JOSÉ NETO

BT/PCS/9702 - Combinando Análise de "Wavelet" e Análise Entrópica para Avaliar os Fenômenos de Difusão e Correlação - RUI CHUO HUEI CHIOU, MARIA ALICE G. V. FERREIRA

BT/PCS/9703 - Um Método para Desenvolvimento de Sistemas de Computacionais de Apoio a Projetos de Engenharia - JOSÉ EDUARDO ZINDEL DEBONI, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9704 - O Sistema de Posicionamento Global (GPS) e suas Aplicações - SÉRGIO MIRANDA PAZ, CARLOS EDUARDO CUGNASCA

BT/PCS/9705 - METAMBI-OO - Um Ambiente de Apoio ao Aprendizado da Técnica Orientada a Objetos - JOÃO UMBERTO FURQUIM DE SOUZA, SELMA S. S. MELNIKOFF

BT/PCS/9706 - Um Ambiente Interativo para Visualização do Comportamento Dinâmico de Algoritmos - IZAURA CRISTINA ARAÚJO, JOÃO JOSÉ NETO

BT/PCS/9707 - Metodologia Orientada a Objetos e sua Aplicação em Sistemas de CAD Baseado em "Features" - CARLOS CÉSAR TANAKA, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9708 - Um Tutor Inteligente para Análise Orientada a Objetos - MARIA EMÍLIA GOMES SOBRAL, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9709 - Metodologia para Seleção de Solução de Sistema de Aquisição de Dados para Aplicações de Pequeno Porte - MARCELO FINGUERMAN, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9801 - Conexões Virtuais em Redes ATM e Escalabilidade de Sistemas de Transmissão de Dados sem Conexão - WAGNER LUIZ ZUCCHI, WILSON VICENTE RUGGIERO

BT/PCS/9802 - Estudo Comparativo dos Sistemas da Qualidade - EDISON SPINA, MOACYR MARTUCCI JR.

BT/PCS/9803 – The VIBRA Multi-Agent Architecture: Integrating Purposive Vision With Deliberative and Reactive Planning – REINALDO A. C. BIANCHI , ANNA H. REALI C. RILLO, LELIANE N. BARROS

BT/PCS/9901 – Metodologia ODP para o Desenvolvimento de Sistemas Abertos de Automação – JORGE LUIS RISCO BECCERRA, MOACYR MARTUCCI JUNIOR

BT/PCS/9902 – Especificação de Um Modelo de Dados Bitemporal Orientado a Objetos – SOLANGE NICE ALVES DE SOUZA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9903 – Implementação Paralela Distribuída da Dissecação Cartesiana Aninhada – HILTON GARCIA FERNANDES, LIRIA MATSUMOTO SATO

BT/PCS/9904 – Metodologia para Especificação e Implementação de Solução de Gerenciamento – SERGIO CLEMENTE, TEREZA CRISTINA MELO DE BRITO CARVALHO

BT/PCS/9905 – Modelagem de Ferramenta Hipermídia Aberta para a Produção de Tutoriais Interativos – LEILA HYODO, ROMERO TORI

BT/PCS/9906 – Métodos de Aplicações da Lógica Paraconsistente Anotada de Anotação com Dois Valores-LPA2v com Construção de Algoritmo e Implementação de Circuitos Eletrônicos – JOÃO I. DA SILVA FILHO, JAIR MINORO ABE

BT/PCS/9907 – Modelo Nebuloso de Confiabilidade Baseado no Modelo de Markov – PAULO SÉRGIO CUGNASCA, MARCO TÚLIO CARVALHO DE ANDRADE

BT/PCS/9908 – Uma Análise Comparativa do Fluxo de Mensagens entre os Modelos da Rede Contractual (RC) e Colisões Baseada em Dependências (CBD) – MÁRCIA ITO, JAIME SIMÃO SICHMAN

BT/PCS/9909 – Otimização de Processo de Inserção Automática de Componentes Eletrônicos Empregando a Técnica de Times Assíncronos – CESAR SCARPINI RABAK, JAIME SIMÃO SICHMAN

BT/PCS/9910 – MIISA – Uma Metodologia para Integração da Informação em Sistemas Abertos – HILDA CARVALHO DE OLIVEIRA, SELMA S. S. MELNICOFF

BT/PCS/9911 – Metodologia para Utilização de Componentes de Software: um estudo de Caso – KAZUTOSI TAKATA, SELMA S. S. MELNIKOFF

BT/PCS/0001 – Método para Engenharia de Requisitos Norteado por Necessidades de Informação – ARISTIDES NOVELLI FILHO, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/0002 – Um Método de Escolha Automática de Soluções Usando Tecnologia Adaptativa – RICARDO LUIS DE AZEVEDO DA ROCHA, JOÃO JOSÉ NETO

BT/PCS/0101 – Gerenciamento Hierárquico de Falhas – JAMIL KALIL NAUFAL JR., JOÃO BATISTA CAMARGO JR.

BT/PCS/0102 – Um Método para a Construção de Analisadores Morfológicos, Aplicado à Língua Portuguesa, Baseado em Autômatos Adaptativos – CARLOS EDUARDO DANTAS DE MENEZES, JOÃO JOSÉ NETO

BT/PCS/0103 – Educação pela Web: Metodologia e Ferramenta de Elaboração de Cursos com Navegação Dinâmica – LUISA ALEYDA GARCIA GONZÁLEZ, WILSON VICENTE RUGGIERO

BT/PCS/0104 – O Desenvolvimento de Sistemas Baseados em Componentes a Partir da Visão de Objetos – RENATA EVANGELISTA ROMARIZ RECCO, JOÃO BATISTA CAMARGO JÚNIOR

BT/PCS/0105 – Introdução às Gramáticas Adaptativas – MARGARETE KEIKO IWAI, JOÃO JOSÉ NETO

BT/PCS/0106 – Automação dos Processos de Controle de Qualidade da Água e Esgoto em Laboratório de Controle Sanitário – JOSÉ BENEDITO DE ALMEIDA, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/01/07 – Um Mecanismo para Distribuição Segura de Vídeo MPEG – CÍNTIA BORGES MARGI, GRAÇA BESSAN, WILSON VICENTE RUGGIERO

BT/PCS/0108 – A Dependence-Based Model for Social Reasoning in Multi-Agent Systems – JAIME SIMÃO SICHMAN

BT/PCS/0109 – Ambiente Multilinguagem de Programação – Aspectos do Projeto e Implementação – APARECIDO VALDEMIR DE FREITAS, JOÃO JOSÉ NETO

BT/PCS/0110 – LETAC: Técnica para Análise de Tarefas e Especificação de Fluxo de Trabalho Cooperativo – MARCOS ROBERTO GREINER, LUCIA VILELA LEITE FILGUEIRAS

BT/PCS/0111 – Modelagem ODP para o Planejamento de Sistemas de Potência – ANIRIO SALLES FILHO, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/0112 – Técnica para Ajuste dos Coeficientes de Quantização do Padrão MPEG em Tempo Real – REGINA M. SILVEIRA, WILSON V. RUGGIERO

BT/PCS/0113 – Segmentação de Imagens por Classificação de Cores: Uma Abordagem Neural – ALEXANDRE S. SIMÕES, ANNA REALI COSTA

BT/PCS/0114 - Uma Avaliação do Sistema DSM Nautilus –MARIO DONATO MARINO, GERALDO LINO DE CAMPOS

BT/PCS/0115 – Utilização de Redes Neurais Artificiais para Construção de Imagem em Câmara de Cintilação – LUIZ SÉRGIO DE SOUZA, EDITH RANZINI

BT/PCS/0116 – Simulação de Redes ATM – HSU CHIH WANG CHANG, WILSON VICENTE RUGGIERO

BT/PCS/0117 – Application of Monoprocessed Architecture for Safety Critical Control Systems – JOSÉ ANTONIO FONSECA, JORGE RADY DE ALMEIDA JR.

BT/PCS/0118 – WebBee – Um Sistema de Informação via WEB para Pesquisa de Abelhas sem Ferrão – RENATO SOUSA DA CUNHA, ANTONIO MOURA SARAIVA

BT/PCS/0119 – Parallel Processing Applied to Robot Manipulator Trajectory Planning – DENIS HAMILTON NOMIYAMA, LÍRIA MATSUMOTO SATO, ANDRÉ RIYUITI HIRAKAWA

BT/PCS/0120 – Utilização de Padrão de Arquitetura de Software para a Fase de Projeto Orientado a Objetos – CRISITINA MARIA FERREIRA DA SILVA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/0121 – Agilizando Aprendizagem por Reforço Através do uso de Conhecimento sobre o Domínio – RENÊ PEGORARO, ANNA H. REALI COSTA

BT/PCS/0122 – Modelo de Segurança da Linguagem Java Problemas e Soluções – CLAUDIO MASSANORI MATAYOSHI, WILSON VICENTE RUGGIERO

BT/PCS/0123 – Proposta de um Agente CNM para o Gerenciamento Web de um Backbone ATM – FERNANDO FROTA REDÍGOLO, TEREZA CRISTINA MELO DE BRITO CARVALHO